Integrative analysis of ChIP-chip datasets in *Saccharomyces cerevisiae*

Mark Bennett
Cancer and Genetics Institute
Cardiff University School of Medicine
Cardiff University

A thesis submitted to Cardiff University
for the degree of
Doctor of Philosophy

2012

This work was funded by
a University of Wales
Lord Merthyr Research Scholarship

Acknowledgments

The work contained in the following pages would not have been possible without a large number of people, to all of whom I owe a debt of gratitude. Firstly I must thank my principle supervisor, Dr. Simon Reed, both for giving me the opportunity to undertake this PhD and for providing support and guidance along the way. Your passion, knowledge and seemingly limitless capacity to recall every detail of papers published decades ago is inspirational. I would also like to thank my second supervisor, Dr. Peter Giles, for all of your help and for doing your best to answer my obscure, technical queries, and Prof. Ray Waters, for providing invaluable feedback in lab meetings and elsewhere.

Dr. Shirong Yu, Dr. Yumin Teng, Dr. Matthew Leadbitter, Dr. James Powell, Dr. Richard Webster, Yenbo Deng, Rachel Pearson, Mark Robinson and, especially, Dr. Katie Evans, who started her PhD journey at the same time as me and has made it so much more enjoyable and productive: I thank you all for the hours you spent in the lab to produce the data that I have analysed here, especially so for those assays carried out at my request, without which I would not have been able to present this work. I also thank you for all the feedback, suggestions and error finding that has lead to the production of the software presented here. In addition, I would like to thank everyone else in the lab, both past and present, that have helped make things go by that little bit more smoothly, including Trish, Amy, Craig, Patrick, Neil, Becky, Yach, Huayan and Zheng. It has been great to work and play with such a fantastic team of people.

I must also thank my family, especially my Mum and Dad, for their support over the last 27-and-a-bit years. I hope the next 300-odd pages will clarify for you what it is that I have been doing for the most recent chunk of that time. Finally I must thank Maria, who over the course of this PhD has gone from my girlfriend to fiancée to wife, for your daily support and encouragement, for your tolerance during the times I spent glued to a computer barely able to acknowledge your presence, and for the time you spent proof reading my final draft.

Summary

ChIP-chip is a technology originally developed to determine the binding sites of proteins in chromatin on a genome wide scale. Its uses have since been expanded to analyse other genome features, such as epigenetic modifications and, in our laboratory, DNA damage. Datasets comprise many thousands of data points and therefore require bioinformatic tools for their analysis. Currently available tools are limited in their applications and lack the ability to normalise data so as to allow relative comparisons between different datasets. This has limited the analyses of multiple ChIP-chip datasets from different experimental conditions.

The first part of the study presented here is bioinformatic, presenting a selection of tools written in `R` for ChIP-chip data analysis, including a novel normalisation procedure which allows datasets from different conditions to be analysed together, permitting comparisons of values between different experiments and opening up a new dimension of analysis of these datasets. A novel enrichment detection procedure is presented, suited to many formats of data, including protein binding (which forms peaks) and epigenetic modifications (which can form extended regions of enrichment). Graphical tools are also presented, to facilitate the analysis of these large datasets. A method of predicting the output of a ChIP-chip dataset is presented, which has been used to show that ChIP-chip is capable of detecting sequence dependent damage events. All functions work together, using a common data format, and are efficient and easy to use.

The second part of this study applies these bioinformatic tools in a biological context. An analysis of Abf1 protein binding datasets has been undertaken, revealing many more binding sites than had previously been identified. Analysis of the sequences at these binding sites identified the previously determined consensus binding motif in only a subset, with no novel motif identifiable in the remainder, suggesting binding may be influenced by factors other than sequence.

Contents

List of Figures

List of Tables

List of R Scripts

List of Abbreviations

A	Adenine
Abf1	ARS binding factor 1
ARS	Autonomously replicating sequence
BER	Base excision repair
bp	Base pair(s)
C	Cytosine
cDNA	Complementary DNA
ChIP	Chromatin immunoprecipitation
ChIP-chip	ChIP on a microarray chip
CNV	Copy number variation
CPD	Cyclobutane pyrimidine dimer
DE	Differentially expressed
DNA	Deoxyribonucleic acid
DSB	Double strand break
dsDNA	Double stranded DNA
FDRE	False discovery rate equivalent
FN	False negative
FP	False positive
G	Guanine
GG-NER	Global genome nucleotide excision repair
GRF	General regulatory factor
H3Ac	Histone H3 acetylation
HAT	Histone acetyl transferase
HEJ	Homologous end joining
HMM	Hidden Markov model
IP	Immunoprecipitated
LOH	Loss of heterozygosity
Mb	Megabase(s)
min	Minute(s)
miRNA	MicroRNA
mRNA	Messenger RNA

MS	Mass spectrometry
NDE	Non-differentially expressed
NER	Nucleotide excision repair
NHEJ	Non-homologous end joining
nm	Nanometre(s)
ORF	Open reading frame
PARP	Poly(ADP-ribose) polymerase
PBR	Potential binding region
PCR	Polymerase chain reaction
PIC	Pre-incision complex
Q-PCR	Quantitative polymerase chain reaction
Q-Q	Quantile-quantile
RNA	Ribonucleic acid
RNAPII	RNA polymerase II
RPA	Replication protein A
sec	Second(s)
SNP	Single nucleotide polymorphism
SSB	Single strand break
ssDNA	Single stranded DNA
SWI/SNF	Switch/ sucrose non-fermentable
T	Thymine
TC-NER	Transcription coupled nucleotide excision repair
TLS	Translesion synthesis
TN	True negative
TP	True positive
U	Uracil
UV	Ultra violet

Chapter 1

Introduction

Investigations into the induction of, cellular responses to, and the repair of DNA damage have been crucial to understanding the mechanisms that have developed within cells to deal with this damage and the implications this can have on many diseases. Multiple assays have been developed to analyse different aspects of these in the laboratory. Here, DNA microarrays have been used to measure some of these end points. Ultra violet (UV) light has been used to induce DNA damage in cells, which is itself measured with a novel microarray technology, along with the measurements of various cellular responses. This study focuses on bioinformatic analyses of data produced from these investigations and the development of new bioinformatic tools to facilitate these analyses. It is therefore split into two broad sections: microarray technology and its associated bioinformatic processing, and the (nucleotide excision) repair of damaged DNA.

1.1 Microarrays

A microarray, in the molecular biological context, is an array of spots each containing a sample of DNA for use in genetic testing (OED, 2011). Simply put, it is a small, rectangular, solid surface containing a regular pattern of spots of multiple copies of single stranded DNA molecules bound to the surface at one end. A pool of DNA applied to this surface and allowed to hybridise to any complementary probes can be detected by means of the

attachment of fluorescent molecules to the pooled DNA. The sequences of DNA probes at the features on the microarray are known, and so the presence of fluorescence at any given location can be associated with that sequence. The microarrays available today contain tens of thousands to millions of features each, which can result in the production of vast quantities of data.

There is some variation in terminology in the microarray field. For purposes of clarity, the following definitions will be used throughout this thesis:

Probe An individual strand of DNA of known sequence, bound to the surface of a microarray.

Feature A collection of a number of probes, all of the same sequence, in an area of defined size and shape and at a known location on the microarray.

Microarray A collection of a quantity of different features arranged in a regular pattern of known, defined properties.

Slide The physical object upon which one or more microarrays reside.

This section outlines the different microarray applications currently available and the *in silico* procedures used to transform the signals these produce into biologically meaningful results.

1.1.1 Hypothesis generation

The basic principle of a scientific experiment is to measure a factor of interest under different conditions so as to prove or disprove a hypothesis relating to the response of the factor to the different conditions. For example, one may, based on prior knowledge and observations, formulate the hypothesis that a protein binds to DNA following a particular treatment. To test this one could measure the amount of the protein bound to DNA before and after the treatment. The hypothesis would be proved if the protein was seen bound to the DNA only after the treatment. This result would then become prior knowledge and used to formulate and test further hypotheses. This 'classical' approach to science, using the accumulation of knowledge to generate and

test new ideas, has been used for centuries and continues to be a valuable tool in scientific progression. A famous example is Edward Jenner's testing of the hypothesis that cowpox infection conferred smallpox immunity, by inoculating James Phipps with cowpox and subsequently attempting to infect him with smallpox, which failed. In addition, many scientific discoveries have occurred by serendipity, that is, a result was not obtained via the testing of a hypothesis but by the product of coincidental events. A famous example is the discovery of penicillin by Alexander Flemming: he did not set out to test whether or not penicillin was an antibiotic, rather he found the product of a mould growing on some discarded culture plates of *Staphylococci* caused the surrounding cells to undergo lysis.

The advent of the 'genomics era' has allowed the development of technologies that allow new results to be found 'by chance', rather than by conducting specific hypothesis testing investigations. These experiments are termed 'hypothesis generating' and facilitate the discovery of results that may not otherwise have been specifically looked for. This is achieved by the testing of a large number of individual biological conditions at once, such as the expression levels of thousands of genes, or the properties of thousands of genomic regions with respect to some condition, such as protein binding. This is equivalent to performing thousands of individual assays by conventional methods, which is not practical, which is why only those designed to test specific hypotheses are ever carried out. Removing this limitation means that data can be generated that otherwise would not be and so new and unexpected phenomena may be found, leading to the generation of new hypotheses. Microarray investigations are often carried out with the intention of being hypothesis generating. The results of an investigation or investigations can be used to construct a hypothesis that may or may not have otherwise been conceived. That is to say, new or unusual phenomena may be seen that would never have been specifically looked for and so would remain unknown unless found by chance.

Genome wide technologies, such as microarrays, generally have too low a signal-to-noise ratio to prove a hypothesis outright, meaning results need to be corroborated by alternative, more sensitive technologies. The microarray

results may lead to the generation of a hypothesis relating to a small number of factors, all of which can be investigated with more sensitive tests, such as differential expression in a small number of genes. This can completely prove or disprove the hypothesis. It may also produce a hypothesis relating to a large number of factors, such as a protein binding at multiple sites throughout a genome. In this situation more sensitive tests can be carried out with a subset of representative data. If these microarray results are shown to be correct it is reasonable to assume that the data are consistent across the rest of the microarray. The hypothesis therefore remains valid, but is not completely proven. A set of results, once generated, has the potential to be used to generate many more hypotheses, possibly far removed from the reason the original assay was carried out. This can greatly accelerate the rate of scientific discovery.

Much of the bioinformatic work presented in this thesis aims to facilitate the discovery of new results and the generation of new hypotheses. The graphical display tools (Chapter 3) allow potential patterns in the data to be discerned; the normalisation procedure (Chapter 4) allows datasets to be compared, revealing potential novel responses to changes in conditions; and the enrichment detection procedure (Chapter 5) allows potential novel binding regions to be identified.

1.1.2 Types of microarray

The first microarray paper was published in 1995 (Schena et al., 1995), where cellular DNA amounts were measured to gain information on gene expression levels in the plant *Arabidopsis thaliana*. This microarray contained only a fraction of the number of features available on today's arrays, at 48. The advances in microarray technology over the 17 years since this publication have meant microarray use has expanded into several applications. Hoheisel (2006) provides a summary of these which, although outdated in some cases, provides a good indication of the range of potential applications. The main applications available today are summarised in the following sections, along with their non-microarray alternatives.

1.1.2.1 RNA detection

Measurement of messenger RNA (mRNA) levels allows the gathering of information relating to gene expression and the regulation thereof. RNA is purified from the cell and amplified as complementary DNA (cDNA) which is applied to the microarray.

Gene expression profiling

Gene expression profiling is the measurement of the level of expression of genes, indicative of the level of protein production. This can provide important information about how, when and where genes are expressed or repressed and how this changes in response to particular conditions or between different cell types, genetic mutants, individuals or disease states. Several methods have been developed to achieve this, such as creating reporter gene constructs, where a level of fluorescence or enzymatic activity represents the level of gene expression, or Northern blots, where the mRNA amount produced from a gene is visualised by autoradiography using electrophoretic gels and nitrocellulose paper (Alberts et al., 2002). These technologies are time consuming and allow the analysis of only small numbers of genes at a time. Microarrays can overcome this limitation by allowing measurements of the expression levels of thousands of genes in a single assay.

Profiling of 45 *A. thaliana* cDNAs was the use of the first microarray (Schena et al., 1995). This remains the most popular microarray use today (NIH Entrez search results; data not shown), with microarrays available containing gene sequences from many organisms. Expression profiling allows the expression levels of genes to be determined by measuring the amounts of mRNA produced from gene expression. These microarrays contain probes corresponding to mRNA sequences, or parts thereof. There are two sources of these sequences, the most common of which is cDNA molecules (Schena, 2003), originally used by Schena et al. (1995). These cDNAs are produced from mRNAs by the reverse transcriptase enzyme, which uses RNA templates to produce DNA sequences, and applied to the slide surface. As the sequences are produced directly from the mRNAs they are intended to mea-

sure they can be long in length (typically 500-2500 bases) and produce good hybridisation signals (Schena, 2003). The alternative method is to produce oligonucleotides by chemical synthesis, first used on microarrays by Lockhart et al. (1996). These may be synthesised directly on the slide surface, base-by-base, or elsewhere and applied as a single strand in the same way as a cDNA (Schulze and Downward, 2001). These technologies have been reviewed and compared extensively over the course of their development (for example, Schulze and Downward, 2001; Li et al., 2002; Tan et al., 2003).

Alternative splicing analysis

Alternative splicing is another method by which protein production can be regulated by the (eukaryotic) cell (Alberts et al., 2002). Genes are divided into expressed sequences (exons) and intervening sequences (introns). Initially all exon and intron sequences are transcribed into a pre-RNA, and then the intron sequences are removed by RNA splicing on the way to producing the final mRNA. Alternative splicing is a process by which different exons are incorporated into the final mRNA, which go on to produce different proteins, meaning single genes can produce multiple products. Gene transcription microarrays can be extended to examine alternative splicing by incorporating probes representing all exons (for example, Johnson et al., 2003). This allows not only the expression of genes to be measured, but the identification of which exons are present in the final mRNAs.

MicroRNAs

MicroRNAs (miRNAs) are short (typically 21–22 nt) RNA molecules derived from non-protein coding genes (Watson et al., 2003). These molecules play important regulatory roles in animals and plants by targeting mRNAs with sequence homology for cleavage or translational repression, thereby regulating protein production levels (reviewed in Bartel, 2004). A microarray containing probes corresponding to known miRNA sequences allows the detection and identification of those present in a given sample (Liu et al., 2004).

1.1.2.2 DNA detection

Measurement of DNA levels allows the gathering of information relating to events which occur directly on or in DNA. DNA is purified from the cell, amplified and applied to the microarray.

Comparative genome hybridisations

Copy number variation (CNV) is a change in the numbers of copies of regions of DNA arising from deletions, insertions and duplications of DNA segments. These can range in size from kilobases to megabases and the frequencies of these variations themselves vary between populations (Redon et al., 2006). CNVs can alter gene expression and phenotypic variation, cause diseases and confer risk to complex disease traits. They are common in cancers and their investigation can provide markers of prediction of disease outcome, treatment responses and identification of genes to target for therapy (reviewed in Albertson et al., 2003).

Comparative genome hybridisation (CGH) is a method of analysing whole genomes for CNVs. The original technology used metaphase chromosomes as a representation of a genome, to which differentially labelled total genomic test and reference DNA was allowed to hybridise. Visualisation of the chromosome allowed the physical locations of CNVs to be mapped (Pinkel and Albertson, 2005). This method limits the detection of events involving regions of less than 20 Mb, a problem which can be overcome by using microarrays in place of the reference metaphase chromosome (Solinas-Toldo et al., 1997; Pinkel et al., 1998), termed array-CGH.

Array-CGH microarrays contain probes corresponding to regions of a genome. Fragmented, labelled genomic DNA is allowed to hybridise to these probes, providing an indication of the amount present at each location. Variations in copy number show up as increased or decreased signal intensities. Depending on the probe density and length of the CNV region, this may occur at a single probe or over multiple, consecutive probes. This type of microarray therefore allows regions of CNV between samples to be found at a resolution theoretically limited only by the probes used.

Chromatin immunoprecipitation on microarray chips

DNA-protein interactions play important roles in genome regulation. Transcription factors, for example, regulate the expression of genes, nucleosomes play structural roles and repair enzymes process sites of DNA damage. Determining where proteins bind in a genome is therefore crucial to investigating both their function(s) and their site(s) of action. Methods to achieve this at particular sites include DNAse footprinting (Galas and Schmitz, 1978), which uses radioactive end-labelling of a DNA fragment of interest and the protective effect of the protein against enzymatic degradation of the DNA to identify binding sites as 'footprints' with gel electrophoresis, and the electrophoretic mobility shift assay (EMSA) (Garner and Revzin, 1981), which uses gel electrophoresis to identify protein-bound DNA by the associated shift in band position on the gel, which can be combined with an antibody against the protein to create a 'super-shift'.

Chromatin immunoprecipitation (ChIP) is another method that can be used to identify protein binding sites. It uses an antibody to the protein of interest to extract, or immunoprecipitate, it and any DNA it is bound to, from the pool of total genomic DNA. PCR primers designed at a region of interest are then used to determine if that region is present in the immunoprecipitated pool. All of these assays allow the identification of protein binding at individual, pre-determined sites of interest only.

ChIP combined with microarray chips (ChIP-chip) is a technology developed to determine the genome wide binding sites of proteins (Ren et al., 2000), hence its alternative name of genome wide location analysis. It works on a similar principle to array-CGH, in that a whole genome (or a genome section of interest) is represented by the features on a microarray. ChIP is carried out, using an antibody to the protein of interest to separate chromatin fragments bound to the protein from the rest of the genome (history and protocol information available in Carey et al., 2009). These separated fragments are then purified, amplified, fluorescently labelled and applied to the microarray. The amount of bound material at any given location allows an estimate of the binding site of the protein to be made, at a resolution

limited by the probes and DNA fragment lengths applied.

Since its first use the technology has been developed to allow the analysis of other features of chromatin, such as histone modifications (Kurdistani et al. (2004), for example), DNA methylation (Weber et al. (2005), for example) and DNA damage (Teng et al., 2011). The technology has also advanced to allow binding site locations to be determined at high resolution, to within several base pairs (Lee et al. (2007), for example).

DNA immunoprecipitation on microarray chips

The DNA sequences at which proteins bind can be used to determine binding sequence specificity and predict sites in the genome at which proteins may bind. These can be derived from *in vivo* protein-DNA interaction assays, such as those described in the previous section, or by specific *in vitro* techniques. For example, SELEX (Systematic Evolution of Ligands by Exponential Enrichment; Tuerk and Gold, 1990) uses randomly generated DNA sequences to iteratively determine protein binding sites. Such *in vitro* techniques are free of the cooperative and/or competitive effects of other proteins in the cell, meaning the protein can bind to any and all potential sites.

DNA immunoprecipitation (DIP) on microarray chips (DIP-chip) is an alternative *in vitro* method to ChIP-chip to determine the genome wide identification of protein binding sites (Liu et al., 2005; Gossett and Lieb, 2010). A sample of purified protein of interest is incubated with naked, sheared genomic DNA and allowed to hybridise. These bound fragments are separated by immunoprecipitation or affinity purification and purified, amplified, fluorescently labelled and applied to a genome wide microarray as used in ChIP-chip. The advantage of this method is any amount of protein can be added to the mix, so it can be performed with proteins whose cellular expression levels may be too low for effective ChIP-chip. The disadvantage is that, as it is an *in vitro* technique, the result is not necessarily representative of what happens in a cell. There may be potential binding sites of a protein that are never actually bound by the protein *in vivo*, and so are not biologically relevant, but this technique will still show them. Similarly, there

may be sites that are only bound *in vivo* under certain conditions, but this technique will not be able to distinguish between them.

Genotyping

Single nucleotide polymorphisms (SNPs) are single base pair differences detected in the genomes of individuals at a frequency more than 1% in a population. These represent approximately 90% of the genetic variation between humans (reviewed by Brookes, 1999). They have important implications in many diseases, including cancers, as certain SNPs are associated with a greater risk of developing the disease. They can be detected with a variety of methods, such as PCR, where primers are designed over the SNP site such that a fragment will only be amplified with one version of the polymorphism, or the use of restriction enzymes, which are able to cut at one version of the polymorphism and not the other. Gel assays are then used to resolve both of these. Microarrays can be used to determine the presence of known SNPs by using probes covering the regions of sequence variation (Gunderson et al., 1998). Presence of DNA on a feature containing probes representing a SNP shows that that SNP is present in the genome being examined.

Mutations introducing alterations in the form of SNPs can lead to loss of heterozygosity (LOH). This is the loss of two functional copies of a gene, which with respect to cancer usually refers to a tumour suppressor gene or similar. A mutation in one copy of the gene on one chromosome leaves a second functional copy on the second chromosome and so often no phenotypic effect is seen, that is, the individual remains healthy. LOH follows a mutation in the remaining functional copy of the gene leading to a loss of functionality. Microarrays can be used to specifically analyse these LOH-associated SNPs.

DNA resequencing is the sequencing of known DNA regions to detect unknown mutations (Sram et al., 2008). This can be achieved using microarrays with probes containing sequences that vary from the known sequence being investigated, by incorporating a single base change in the sequence. Presence of DNA bound to such a fragment, over the unmodified sequence, indicates the presence of the mutation. This was first applied to the human

mitochondrial genome (Chee et al., 1996).

1.1.2.3 Protein detection

Microarrays have also been used to investigate protein-DNA binding *in vitro* by applying epitope-tagged proteins directly to a microarray containing double stranded DNA probes (Mukherjee et al., 2004). The probes these proteins bind to can be detected with a labelled antibody against the epitope tag. Binding sequence motifs can then be determined based on all of the bound probes.

1.1.2.4 Other applications

As well as these general, whole genome approaches, microarrays can be used for more specific, targeted investigations. For example, in the field of genotoxicity testing both gene expression and SNP arrays can be used to investigate the effects of compounds on cells. Gene expression arrays can be used to screen for changes in expression levels linked to particular genotoxic responses and may enable the distinction of different classes of genotoxic compounds that operate via different modes of action (Newton et al. (2004), for example).

1.1.3 Normalisation

Microarrays can produce large amounts of data: one or two intensity values per feature (from one or two colour format microarrays respectively) mean each microarray can produce upwards of two million separate values. These data are produced by a scanner, which records the intensity of light emitted from each feature on the slide. The more labelled DNA that has hybridised to a feature, the brighter the fluorescence. The dyes used to label the DNA have specific excitation and emission frequencies. A laser of the correct frequency is used to excite the dye and the resulting fluorescence at the excitation frequency is recorded. The features on two colour microarrays are excited and recorded at the two relevant frequencies. An image of the microarray is created from all the excitation values, recording the regions surrounding

features as well as the features themselves. Software is then used to convert the features in the image into numerical values, which may also take account of the fluorescence in the background regions. These values are stored along with probe specific information. Full details of this procedure are given in Chapter 2.

Microarray investigations are not generally carried out individually but, as with many other experimental procedures, are performed in replicate. Certain types of investigation, such as gene expression profiling, examine multiple conditions, with replicate assays carried out for each condition. The intention with these investigation is to compare the results relative to each different experimental condition, to determine changes between them. This differs to other uses, such as genotyping, which produce binary results, where the presence or absence of DNA on each feature is the final result, showing, for example, the presence or absence of a given SNP. ChIP-chip is generally currently used as in the latter example, but the data have the potential to generate results as in the former, as demonstrated later in this thesis, greatly expanding the use of the technology. Microarray analysis is reliant on a robust normalisation procedure, as outlined in the following section. Multiple assays are performed because of random 'noise' in datasets, caused by inherent variations in the microarray technology. Readings representing genuine biological phenomena, or 'signal', can only be reliably differentiated from this noise by their consistency across different datasets, whereas anomalous, high, chance readings caused by the noise occur randomly across the datasets. Therefore the more variation between datasets, the more difficult it is to distinguish biologically relevant signals from this background noise. Some of the noise can be minimised by good working practices when preparing the microarrays, but much of it is always present due to factors beyond the control of the operator (Johnson et al., 2008). This needs to be removed by the application of a normalisation procedure to the data. Gentleman (2005) summarises this problem:

> "In an ideal experiment, no normalisation would be necessary, as the technical variations would have been avoided in the first place. In a real experiment, a certain amount of technical varia-

> tion cannot be avoided, and can be accounted for and corrected in the subsequent analysis through a normalisation procedure. However, if the technical variations become too large, they turn into a quality problem."

Normalisation in this context refers to the processing of multiple datasets so as to make them comparable with each other by removing any variation not caused by the biological factor under investigation. It is therefore essential and as such has been extensively discussed and different methods compared, initially with reference to gene expression microarray investigations (see Butte, 2002; Quackenbush et al., 2002; Bolstad et al., 2003; Shedden et al., 2005; Do et al., 2006; Fujita et al., 2006, for examples) and more recently ChIP-chip investigations (see Buck and Lieb, 2004; Peng et al., 2007; Johnson et al., 2008; Adriaens et al., 2012, for examples). Gentleman (2005) summarises the concept:

> "The purpose of normalisation is to identify and remove systematic technical variation while retaining the biological signal. Among the sources of technical variation are different labelling efficiencies and scanning properties of the Cy3 and Cy5 dyes [and] different scanning parameters... Normalisation procedures aim to ensure that the observed differences in intensity indeed reflect the differential gene expression and not artificial biases due to such technical factors."

Although written with reference to gene expression microarrays, the principle of this quote is equally applicable to other types of microarray experiments and the words "gene expression" can be substituted for "DNA amounts" to reflect the same problem faced with ChIP-chip data.

Normalisation can be broadly divided into two categories, intra- and inter-dataset, which, depending on the method of normalisation chosen, are carried out separately or as one process. For replicate (biological or technical) assays measuring a single condition, such as biological repeats of gene expression assays to determine mRNA levels, this means removing any variations arising in the carrying out of separate assays, termed intra-dataset normalisation.

Theoretically these datasets should be exactly the same, as the same variable is being measured. This is not actually the case, because of the noise in the data. Intra-dataset normalisation aims to remove as much of this variation as possible. For experiments with two or more conditions, such as those performed over a period of time or subject to different treatments, in addition to performing intra-dataset normalisation on the replicate assays, inter-dataset normalisation also needs to be performed. This aims to remove the same variation as intra-dataset normalisation, but leave any of the biologically relevant variation caused by the deliberate changing of experimental conditions, thus allowing a comparison of the effect of the experiment to be made. Some normalisation methods combine both of these principles.

1.1.3.1 Gene expression

Several normalisation methods have been developed for gene expression microarray data, which aim to allow comparisons of gene expression levels between different conditions. These methods work on the assumption that the majority of gene expression levels do not change under any given change in conditions, and of those that do change there will be approximately equal numbers of up- and down-regulated genes, resulting in an approximately symmetrical distribution of probe values (Figure 1.1). These unchanging 'background' levels act as internal controls against which the changed levels can be measured. The unchanged values can be easily identified as they represent the bulk of the data. The normalisation methods aim to make these background probes consistent across all datasets, altering the changed probes as they do so, with the result that the changed probes can be compared across all datasets relative to the background values. In gene expression investigations these comparisons show increases or decreases in gene expression levels in response to the change in experimental conditions.

A simple normalisation was applied to the first microarray datasets (Schena et al., 1995) to correct for differences in the two fluorescent dyes being used. A DNA sample of known concentration (1:1,000 dilution) was added to both pools of DNA and labelled with the respective dye. The values from the two

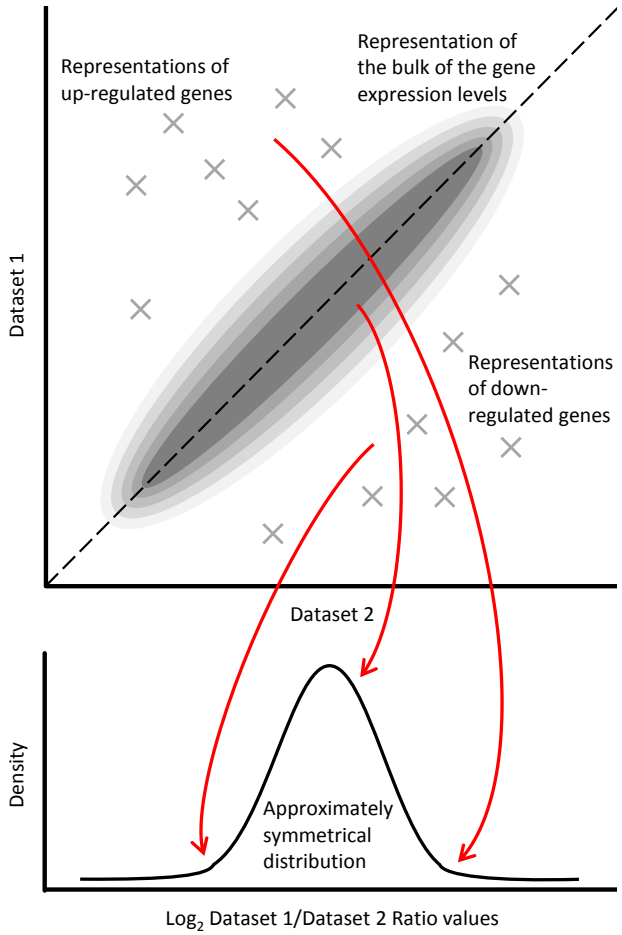


Figure 1.1: Gene expression microarray data representation: One set of gene expression microarray values are represented plotted against another (top panel). The oval represents the region containing the majority of the data, with darker greys representing more data, where the values are the same in both datasets. A small number of genes are up- or down-regulated in one dataset relative to the other, indicated with grey crosses. The ratios of the two datasets creates an approximately symmetrical distribution (bottom panel). Red arrows show the regions of data in the top panel which form the parts of the distribution in the bottom panel.

pools were adjusted by 'matching' the signals from the added DNA. As the sensitivity and numbers of features on gene expression microarrays has increased, so normalisation methods have been adapted to provide robust and reliable results. More recently, normalisation methods have been adapted and developed for use with other types of microarray, such as ChIP-chip and array-CGH (reviewed by Lai et al. (2005)).

The following sections give brief descriptions of some of the more commonly applied gene expression microarray normalisation methods.

Total intensity normalisation

Total intensity normalisation normalises the signals from two channels based on their total intensities (summarised from Quackenbush et al. (2002)). It relies on several assumptions about the assay, the first of which is that equal amounts of material for the two samples are applied to the microarray. As there are millions of individual molecules in each sample, it follows that the average mass of each molecule is the same and consequently the same number of molecules are present in each sample. The probes on the microarray are assumed to randomly interrogate the two samples, meaning the probes must represent a random sample of genes. Approximately the same numbers of molecules from each sample should hybridise to each microarray, because it is assumed that equal numbers of genes will be up- and down-regulated in the two samples. Therefore it is assumed that the total signal intensities of the two channels should be the same. A normalisation factor is calculated based on the actual total signal intensities, which is used to calculate the normalised intensities. There are several ways in which this can be applied, such as adjusting both channels to a defined mean or median ratio. These methods are generally considered to be too simplistic as they do not take into account systematic biases which may occur in the data. In theory, plotting one set of signal intensities against the other should show data centred around a straight line. This is based on the same assumptions as above, that a random sample of genes is represented with equal numbers of up- and down-regulated genes (implicit in this is the assumption that most genes are not

disregulated between the two samples). However, intensity dependent effects may mean that values are higher or lower than expected at different points across the spread of data, creating a non-linear relationship. If not corrected for this may produce spurious results in the analysis as the higher (lower) values may be incorrectly interpreted as up-regulated (down-regulated) genes. Figure 1.2 represents this procedure with two datasets that have a non-linear relationship, showing the adjustment cannot take account of this. The following methods attempt to take account of non-linear relationships between data.

Lowess

Locally weighted linear regression (lowess) (Cleveland, 1979) is a method of smoothing a scatter plot via a polynomial fit to the data, using weighted least squares. This allows a non-linear fit to be calculated between the two sets of signal intensities. Lowess applies a locally weighted linear regression through the data, which emphasises the contributions of data close to other points, thus minimising the effects of disregulated genes. Each data point is adjusted based on the fit, creating data centred around an approximately straight line. Figure 1.3 represents this procedure with two datasets that have a non-linear relationship, showing the adjustment takes account of this to bring the whole bulk of the data to lie on the line $y = x$. This may be applied globally, to all the data at once, or locally, to subsets at a time.

In the event that there is a linear intensity dependent relationship between datasets, linear regression can be applied to perform a correction.

VSN

A variance stabilisation normalisation (VSN) was created to take account of the fact that the variance of intensity values can increase with their means (Huber et al., 2002). This means values that differ by the same factor can produce varying significance levels depending on their intensity values, which has important implications in gene expression monitoring. A lowly expressed gene whose transcription level doubles is as biologically significant as a highly

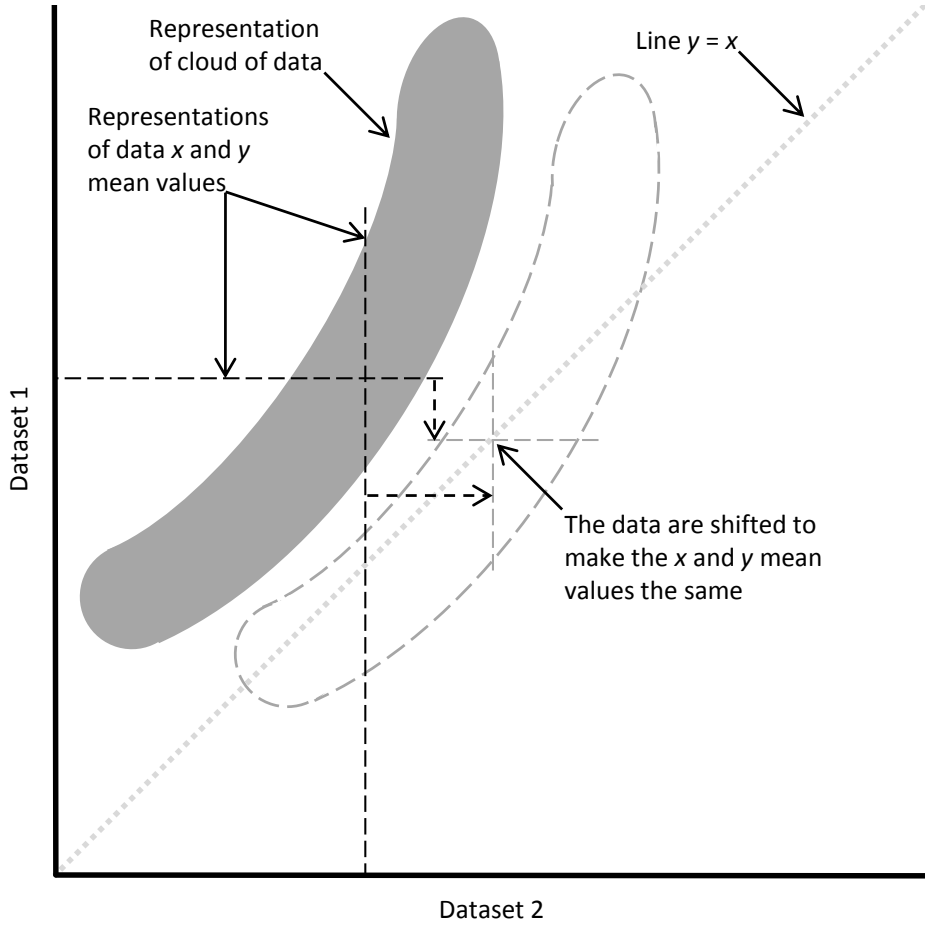


Figure 1.2: Total intensity normalisation representation: The grey shape represents the bulk of the data when two datasets are plotted together, showing a non-linear relationship. The mean for each dataset is shown with a dashed black line. The procedure adjusts the data to make these means lie on the same value, bringing them to lie on the line $y = x$, represented with dashed grey lines. This shows that the procedure cannot bring all of the data to lie on this line.

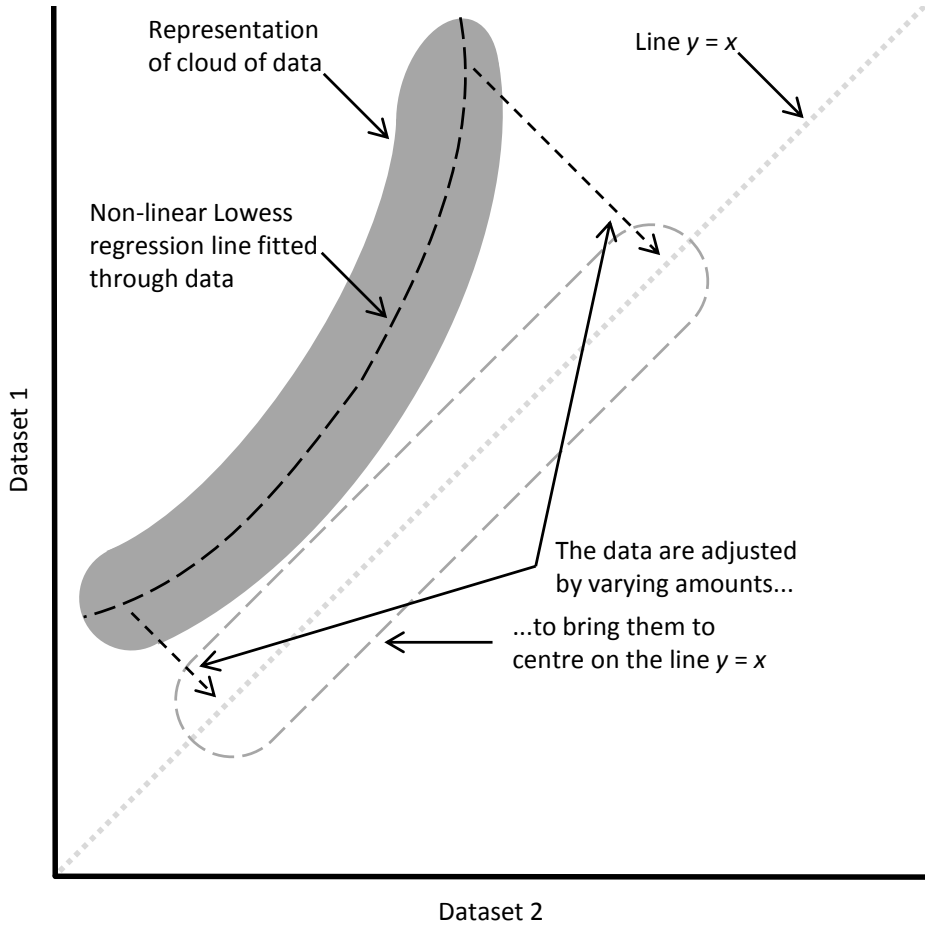


Figure 1.3: Lowess normalisation representation: The grey shape represents the bulk of the data when two datasets are plotted together, showing a non-linear relationship. The Lowess polynomial fit through these data is represented with a dashed black line. The procedure adjusts the data to make this line lie on the line $y = x$, represented with dashed grey lines. This shows that the procedure brings all of the data to centre on this line.

expressed gene with the same increase, but intensity-dependent variance variations may mean this is not borne out statistically. This is corrected for by applying a variance stabilising transformation to the data. This replaces the log-ratios with a difference statistic which displays approximately constant variance independent of the spot intensity (Huber et al., 2002).

Quantile

Quantile normalisation aims to make the distributions of probe intensities across a set of microarrays the same (Bolstad et al., 2003). This is achieved by sorting the values in each set of data, replacing all the values of each row with the mean of the sorted data in the row, and rearranging the values back to their original order. This comes from the fact that quantile-quantile (Q-Q) plots display identical distributions as data points aligned along the line $y = x$, which also applies in n-dimensional space with n datasets. The quantile normalisation procedure adjusts the data such that all datasets lie on this line in n dimensional space, or any two lie on the line $y = x$ in a standard Q-Q plot. This has the effect of adjusting all distributions to the same 'average' distribution. The procedure is described in more detail in Section 4.2.4.1.

1.1.3.2 ChIP-chip

ChIP-chip was originally developed to determine the *in vivo* binding positions of chromatin associated proteins (Ren et al., 2000). Here 6,361 DNA fragments, each representing one intergenic region of the yeast genome, on a microarray were used to analyse the binding sites of the Gal4 and Ste12 transcription factors. Since this first use the technology has been developed to allow the analysis of other features of chromatin, such as histone modifications (Kurdistani et al., 2004, for example), and DNA damage (Teng et al., 2011). The technology has also advanced to allow locations to be determined to within several base pairs (Lee et al. (2007), for example). All of these methods required normalisation of some sort to remove the variations inherent in the technologies. Because the technology has generally been

used to determine the binding positions of factors of interest, normalisation has been focussed on removing variation between replicate datasets so as to enhance downstream analysis. This allows locations to be assigned binary values representing whether or not they are deemed binding sites. Within these replicate datasets the values can be analysed to determine relative differences in their levels at the different sites identified. These indicate relative differences in the levels of occupancy of the factor under investigation. This is akin to the analyses that are performed with array-CGH data, which identify changes in binding levels with the same datasets to find regions of genome duplications. This has been used to show, for example, relative differences in the levels of histone occupancy across the yeast genome (Lee et al., 2007). The current limitation with ChIP-chip normalisation lies in the lack of a normalisation method that allows comparisons of this type to be made between different datasets from different experimental conditions. This is exemplified by analyses such as those carried out by Schlecht et al. (2008), where three datasets of Abf1 binding were created from three different experimental conditions, but comparisons between them were limited to identifying sites of shared or different occupancy.

The format of data generated by ChIP-chip microarrays is different to that described for gene expression microarrays in the previous section. All of these methods rely on the assumptions that the bulk of the data points do not vary between experiments and that the data form roughly symmetrical distributions. The next stages of processing go on to identify the relatively small number of values that change significantly between datasets, representing differentially expressed genes. These same assumptions do not hold for ChIP-chip data. The distribution of ChIP-chip data is asymmetrical because as immunoprecipitated material binds to the probes of the microarray the resulting intensity values can only be larger than the background, never smaller, which gives the distribution a positive skew (this has been pointed out several times, for example, by Buck and Lieb, 2004, . Figure 1.4 shows a representation of this type of data) Additionally, a large proportion of probe values may change between different conditions because a large proportion of the genome may be immunoprecipitated. These features of the data violate

the assumptions of the normalisation procedures. Therefore their application to ChIP-chip data can have the effect of removing biological variations from datasets of different experimental conditions. For example, where large sections of a genome show an increase in the factor of interest between two experimental conditions, the distributions of the two datasets will be markedly different. Specifically, the modal points of the distributions will be different, from the low end in the first dataset (where the bulk of the data represent unenriched regions of the genome) to the high end in the second dataset (where the bulk of the data now represent enriched regions of the genome), which is an important biological variation which should be maintained between the two datasets. However, normalisation methods for gene expression data will seek to remove this difference, under the assumption that the bulk of the data should not change between conditions. Quantile normalisation, for example, would take the two different data distributions and produce a third, 'averaged' distribution, thereby removing all biologically relevant information in the two datasets and producing a third, unrepresentative dataset.

These differences in the properties of the data, meaning that methods developed for gene expression data are not necessarily suitable for application to ChIP-chip data, is an important fact which has been pointed out previously by Ponzielli et al. (2008):

> "Experimental design parameters for mRNA expression arrays have been extensively evaluated by a number of groups over the last decade. As a result, the key factors are well understood and the assays have been optimised. It is possible, for example, to estimate the number of biological repeats required to sufficiently power a specific hypothesis-testing question. Despite this clear evidence that parameter optimisation can greatly improve the quantity and quality of information retrieved from an array analysis, ChIP-chip design parameters have not yet been thoroughly and systematically investigated, and it cannot be assumed that parameters and processes would be the same for both mRNA and ChIP-chip arrays".

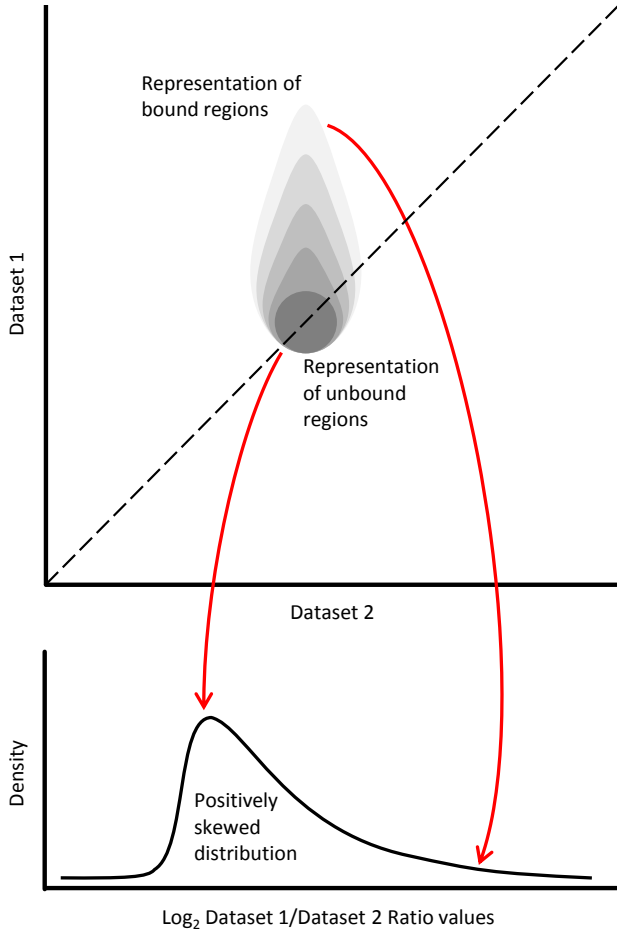


Figure 1.4: ChIP-chip microarray data representation: One set of ChIP-chip microarray values are represented plotted against another (top panel). The grey shape represents the region containing the majority of the data, where a large proportion of values are higher in one dataset. The darker greys represent more data. The ratios of the two datasets creates a positively skewed distribution (bottom panel). Red arrows show the regions of data in the top panel which form the parts of the distribution in the bottom panel.

ChIP-chip data normalisation has generally focused on the task of peak detection, that is, on methods of data manipulation that aid the finding of sites of enrichment across multiple datasets in order to identify protein binding sites. Buck and Lieb (2004) discuss the following two methods.

Median percentile ranks

Median percentile ranks converts all data to ranks, thus removing any biases in the data and removing the need for further processing. This is therefore not strictly a normalisation procedure, but in allowing multiple datasets to be analysed together it performs a similar function and therefore warrants discussion. The ranks are used to identify regions of enrichment across multiple datasets. The median percentile rank for data point x is calculated as the proportion of the data less than x. In this way the ranks run from 0 to 1. If several replicate datasets consist only of random noise, the rankings of the datasets will be random. Therefore the medians of the ranks of each data point will form a normal distribution. If a subset of data points are enriched they will lie at the upper end of their respective rankings and will therefore produce high median values. With enough replicates and high enough levels of enrichment these high median values can be distinguished from the rest of data as a second distribution, alongside the normal distribution of background values. A cutoff value can be determined and used to identify enriched regions. Proteins with small amounts of enrichment may not be detectable via this method. In converting to ranks all binding data is lost which may limit further analyses.

Single array error model

The single array error model was used in early microarray investigations (Ren et al., 2000; Roberts et al., 2000) which tended to contain few probes, with single probes representing sites of enrichment. It allows replicate experiments to be averaged with suitable weightings to take account of differences in variance between different datasets (Buck and Lieb, 2004), thus improving the ability to detect sites of genuine enrichment.

Peng's method

Peng et al. (2007) present a novel normalisation method specifically designed for ChIP-chip data, without the potential problems associated with gene transcription microarray normalisation methods. This is focused on removing the need to carry out mock controls in experiments. These are assays carried out with no antibody, or an antibody against a target not present in the sample, with the aim of gauging any biases which may occur. This data can then be subtracted from the data created with the antibody, to leave only enrichment resulting from the antibody. It uses the first order differences of probe values to create a symmetrical distribution of values. A straight line fitted through this data provides a means of rotating it to lie on the line $y = 0$. The same rotation factor can be applied to the original data to cause the background portion to lie on the line $y = 0$. Lowess normalisation can then be used to correct for any non-linear artifacts in this data. A horizontal line can then be used to represent the cutoff between background and enriched data points. This method doesn't allow for multiple datasets to be normalised together.

A recent evaluation by Adriaens et al. (2012) compared six of the above mentioned normalisation methods commonly used with ChIP-chip and DNA methylation investigations: VSN; lowess; quantile; T-quantile; Tukey's biweight scaling; and the method of Peng et al. (2007). They determined T-quantile normalisation to be the best, which is quantile normalisation applied separately to different datasets in batches, as it conserved enriched and un-enriched signals, allowed identification of regions known to be enriched and improved comparability between datasets.

1.1.4 ChIP-chip data processing

Johnson et al. (2008) carried out an investigation to evaluate variability in ChIP-chip experiments using predefined DNA targets. These targets consisted of 100 samples of cloned genomic DNA sequences added at levels ranging from 1.25- to 196-fold above the background level (a commercial human genomic DNA preparation). Known quantities of DNA sequences

such as these are termed 'spike ins', or simply 'spikes', and adding them in this manner is termed 'spiking in'. The overall aim of this investigation was to test the performance of the various microarray platforms and analysis methods. They found that the microarray platform used was not the primary determinant of overall performance and attribute the variations in results to variations in performance between labs, protocols and algorithms.

Ponzielli et al. (2008) carried out an assessment of a range of ChIP-chip experimental design parameters including antibody selection, array batches, dye effects, scanner calibrations, environmental conditions, handling, amplification methods and hybridisation controls. They found that all the aspects tested had an influence on the final results, which is not unexpected given that each of the aspects is known to have some inherent variation of its own. One surprising result from the study was that the day on which hybridisation was performed had a greater influence than the array batch used or dye swaps. There is no explanation for this and it goes to demonstrate that, even if it were possible to eliminate all variations of known origin, there would still be some variation between experiments caused by as yet unidentified factors. It is this range of influences on the final results that necessitates the need for robust normalisation methods.

1.1.4.1 Peak detection

ChIP-chip analyses often require the application of an algorithm to detect regions of enrichment or peaks. These algorithms attempt to separate sections of genuine enrichment from background noise or non-specific binding. Several algorithms have been developed for this purpose, which are discussed in Chapter 5. They allow the large amounts of data generated by ChIP-chip experiments to be reduced to a list of sites of enrichment. These sites can be used in further investigations, such as confirmation of the presence of the factor of interest by other methods, and sequence analysis, to identify common sequences in the binding regions.

1.1.4.2 Making comparisons between datasets

Conspicuous by its near absence in all of the ChIP-chip data normalisation literature is discussion of making robust comparisons *between* datasets generated under different biological conditions. This is standard practice with other types of microarray investigations. Gene expression microarrays, for example, are used to determine which genes are disregulated by certain experimental conditions. Array-CGH, as stated in the name, compares the hybridisations of different genomes. For the most part ChIP-chip has been used to investigate only one factor of interest at a time, such as determining where in the genome a protein binds. Little work has been done to normalise datasets from different conditions to allow comparisons of binding levels to be made between them, which is one aspect of the work presented in this thesis.

Some early ChIP-chip investigations do make limited comparisons between different datasets, such as Pokholok et al. (2005), who compare histone H3 and H4 profiles over an average gene. This is done graphically, and no mention is made of any normalisation carried out to specifically allow the comparison of data between different datasets. No conclusions are drawn from this comparison. Penterman et al. (2007) compare ChIP-chip DNA methylation profiles by a simple subtraction of mutant from wild type datasets. This allowed regions of different methylation levels to be determined between the two. These were analysed statistically to identify difference values greater than those expected by chance. Farthing et al. (2008) compare DNA methylation profiles in mouse promoters with, "algorithms... developed to assess differences in promoter methylation between the different cell types." This uses "a 500 bp sliding window to identify areas where the methylation state consistently and significantly changed between the two tissues being compared." No mention of a normalisation procedure prior to the application of these procedures is made in either investigation. With datasets such as these, where the factor being investigated occurs over large stretches of the genome, it is relatively easy to determine whether or not those regions correspond in different datasets. It is not possible, without

proper normalisation, to infer any numerical information from the comparisons, which these investigations do not attempt.

A package for the analysis of ChIP-chip data, CARPET (Cesaroni et al., 2008), includes a method to compare different datasets. It is stated that "... cross-comparison of [ChIP-chip] experiments is a significant problem that needs to be addressed." However, the method presented provides only a limited solution to this problem, by extracting common or unique regions from two datasets analysed with their peak detection algorithm. No normalisation method is provided to ensure that the results of peak detection on the two datasets are comparable. As mentioned above, if the peak detection method is robust, simply defining regions sharing enrichment is valid, provided no numerical information is inferred, that is, no comparisons of peak heights are made.

Several methods of analysing datasets of the same type but created by different methods or on different platforms have emerged. These allow more than one dataset to be combined to enhance the detection of the factor of interest. Choi et al. (2009) propose a hierarchical hidden Markov model (HMM) to allow the joint analysis of ChIP-chip and ChIP-seq data of the same condition to enhance the detection of binding sites. It is limited to one ChIP-chip and one ChIP-seq dataset only. Another package for the analysis of ChIP-chip data, JAMIE (joint analysis of multiple ChIP-chip experiments; Wu and Ji, 2010) allows multiple experiments to be analysed together. This is achieved through a hierarchical mixture model which analyses multiple datasets from different experiments and platforms to "capture correlations" between them. This allows improved peak detection over analysing each of the datasets individually, which may allow more potential binding sites to be identified and therefore increases the amount of potential information that can be extracted from datasets. Chen et al. (2011) present MM-ChIP, a model-based meta-analysis software to integrate results from different investigations to improve the sensitivity and specificity of detection of enriched regions. This uses a two-step process which first fits the data to a platform-specific model to increase the signal-to-noise ratio and calculates a score to quantify signal enrichment via a sliding window, then calculates Z-scores for

each region, which are combined for all datasets. The statistical properties of these scores are used to determine enriched regions across all datasets. None of these methods contain any normalisation procedures and so comparisons cannot be made between the binding levels in different datasets.

Johannes et al. (2010) acknowledge the problem with respect to the comparison of chromatin profiles with ChIP-chip and ChIP-seq data. They say, "with ChIP samples collected from different tissue types and/or individuals, we can now begin to characterize stochastic or systematic changes in epigenetic patterns... this requires statistical methods that permit a simultaneous comparison of multiple ChIP samples on a global as well as locus-specific scale." However, it is computationally limited to a small number of datasets and is recommended as a "first-pass algorithm" to identify candidate regions, which should be followed with another, unspecified, algorithm to fine tune the results. As it is intended for use with ChIP-chip and ChIP-seq data together it will not necessarily work with ChIP-chip alone.

To date, the only publication which addresses the problem with a specific normalisation method is by Landfors et al. (2011), which is aimed at normalising any skewed data sets, not specifically ChIP-chip data. Here data is first normalised by a "standard normalisation technique of the user's choosing," then a method of determining whether or not the distribution of the dataset is skewed (detection of skewed experiments, DSE) is combined with a hidden Markov model-assisted normalisation method. Non-skewed data is deemed to not require further normalisation. The HMM-assisted normalisation procedure aims to identify regions that are unaltered between the different datasets. These regions are used to estimate a normalisation function which is used to normalise the whole dataset. The HMM algorithm identifies the unaltered regions on which to perform the next normalisation procedure by analysing the ratios of the averaged two datasets. Those closest to zero are classified as unaltered.

None of these methods specifically address the problem of the lack of a robust normalisation procedure to allow comparisons to be made between ChIP-chip datasets produced from more than one different biological conditions. A novel normalisation method which aims to achieve this is presented

in Chapter 4 and applied to real ChIP-chip data in Chapters 5 and 7.

1.2 DNA damage

The work presented in this thesis is all linked to DNA damage, either directly, in the case of its measurement, or indirectly, in the case of measuring other cellular responses to damage induction. All of the datasets used in this thesis were produced for analyses related to the repair of DNA damage. This section gives an overview of the DNA molecule, the damage it can acquire and the processes that exist to repair it.

1.2.1 DNA

Deoxyribonucleic acid — DNA — is the fundamental building block of life. It encodes everything crucial for the correct functioning of cells and is the molecule of genetic inheritance. Its stability is therefore critical to it being able to correctly perform its functions. It has posed life on Earth a problem that this molecule is not chemically inert, but faces constant challenges to its stability. Water, for example, can hydrolyse bonds in DNA, free radicals can oxidise bonds, ultra violet (UV) radiation can fuse adjacent bases together and ionising radiation can break the double stranded backbone. Organisms of all known types have developed a range of mechanisms to deal with these various genetic insults. Incorrect repair of DNA damage can result in various outcomes, from no phenotypic change if, for example, a silent mutation is created, to cell death if, for example, the damage is so great the cell cannot deal with it or functional mutations are created in critical regions of the genome. Somewhere between these two extremes lies a region where cells survive the damage event but do so in a way that they become rogue; beyond the normal cellular controls in such a way that they will eventually, without intervention, lead to the death of the organism, as is the case in cancer.

1.2.1.1 Structure

The structure of DNA was famously elucidated in 1953 by Watson and Crick, using x-ray crystallography images generated by Rosalind Franklin (Crick and Watson, 1953). It was immediately obvious to these researchers that the structure of DNA lends itself to copying, stating, "if the sequence of bases on one chain is given, then the sequence on the other chain is automatically determined." This has indeed shown itself to be true, and it is the duplication of this sequence that is required for the continued proliferation of life.

DNA comprises two polynucleotide chains twisted around each other to form a double helix. Nucleotides, the building blocks of DNA, consist of a phosphate molecule joined to a sugar molecule joined to a base. The orientation of a nucleotide strand is described by the free carbon molecules of the sugar at either end of the chain: $3'$ and $5'$. This polarity enables the differentiation of the two strand ends.

The 'backbone' of DNA is formed by repeating sugar–phosphate groups, and so these are constant in all nucleotides. The variation comes about by different base structures. There are two classes of base: purines and pyrimidines. The purines are named adenine (A) and guanine (G) which are derived from double ring structures. The pyrimidines are named thymine (T) and cytosine (C) which are derived from single ring structures. The two strands of the double helix are held together by hydrogen bonds between these nucleotides and the variations in the four structures are such that pairings can only form between specific pairs: A always pairs with T and G always pairs with C. Two hydrogen bonds form between A and T, while three form between G and C, making this second pairing slightly stronger. This bonding means that the nucleotides are positioned inside the helix, with the sugar-phosphate backbone on the outside. The two strands run antiparallel to each other, that is, one runs $5'$ to $3'$ and its complement runs $3'$ to $5'$.

There are multiple forms DNA can take: in cells it is usually a right-handed helix in the 'B' form. This has each base pair (bp) displaced by approximately 36°, resulting in there being approximately 10 bp per complete revolution of the helix. DNA is a reasonably flexible molecule and even allows

bases to 'flip out' or protrude from the helix. This flexibility has important implications in DNA repair.

The order of DNA's nucleotides encodes its information, in a way that can be thought of as a four-letter alphabet. Genes, or open reading frames (ORFs), encode proteins. These sequences are transcribed to mRNA strands which are then processed by ribosomes to produce proteins. Sequences of triplicate nucleotides, named codons, represent amino acids. The sequence is read by the ribosome which adds the correct amino acid to an extending polypeptide chain. Much information is also contained in the sequences surrounding ORFs, which do not encode proteins but carry important regulatory information to control how cellular processes act upon the DNA.

1.2.1.2 Chromatin

In eukaryotic cells DNA does not exist in the naked state of the double helix described above. Instead it is packaged into a supercoiled, complex, highly condensed DNA-protein structure named chromatin. This is a hierarchical structure, summarised as follows (adapted from Watson et al., 2003). The first level is the nucleosome, which consists of a 160–165 bp segment of DNA wrapped around a core of 8 histone proteins, comprising 2 copies each of histones H2A, H2B, H3 and H4. The N-termini of these proteins protrude from the nucleosome. A segment of 13–18 bp of linker DNA between each nucleosome gives the structure a 'beads-on-a-string' appearance under the electron microscope. Histone H1 binds this linker region, stabilising the structure, creating a more defined angle between the nucleosomes. The next level of structure is the 30 nm fibre: an assembly of nucleosomes 30 nm in diameter. This is further condensed by looping the fibre around a protein core. This creates a structure around 10,000 times more condensed than naked DNA.

1.2.1.3 Replication

The two complementary strands of DNA make it possible to copy information into new cells, enabling the inheritance of the genetic code. When a cell

reproduces, each of the strands serves as a template for the creation of a new, complementary, strand. This results in two identical copies of the original molecule which are distributed to the two daughter cells.

DNA replication requires a DNA polymerase: an enzyme which reads the existing DNA sequence and inserts the appropriate complementary base into the synthesising sequence. There are various types of DNA polymerase, each with different functions and characteristics, but all perform this same basic task. The enzymes have a single active site, which catalyses the addition of any of the four bases, arranged as a pocket around the DNA molecule. When the correct complementary base enters this pocket the molecular arrangement is such that the reaction to join it to the new DNA molecule is catalysed. DNA polymerases are capable of adding up to 1,000 nucleotides a second to a synthesising DNA strand as they move along the template strand (Watson et al., 2003).

Specific exonucleases proof read newly synthesised DNA to correct erroneously added nucleotides. Approximately 1 in 10^5 nucleotides are incorrectly incorporated by the DNA polymerase during replication. Exonucleases degrade DNA from its ends and so the proof reading exonuclease can remove an incorrect nucleotide at the end of the newly synthesised DNA strand, allowing the polymerase another opportunity to add the correct nucleotide. This increases the accuracy of DNA replication to approximately 1 incorrect insertion per 10^7 nucleotides.

1.2.2 DNA damage and repair

The DNA molecule is susceptible to many types of damaging agents. These damages can occur spontaneously, as a result of normal cellular conditions, or from extra-cellular sources. Endogenous DNA damage events arise as a result of reactions with substances present in cells for normal functionality. These spontaneous lesions can occur at high rates because of the constant contact with the damaging chemicals. Exogenous DNA damages arise outside of the cell and as such generally occur at lower frequencies than endogenous damages. They may be caused by chemical or physical insults. Several repair

pathways have evolved in cells to process the various types of damage that they are constantly subject to. This section briefly outlines the main types of damage and the repair pathways that deal with them, split into three sections: events which modify single bases, events which alter the DNA structure and events which cause the breakage of DNA strands. Information is taken from Alberts et al. (2002), Watson et al. (2003) and Friedberg et al. (2006).

1.2.2.1 Base modifications

Base modification events occur at single bases, causing deviation from the intended DNA sequence. They can cause bases to become unrecognisable by the replication and transcription machinery, inhibiting or disrupting these processes, or substitute incorrect bases into the sequence, causing the potential proliferation of mutations. Transitions are interchanges of the same class of base, that is, a purine for a purine or a pyrimidine for a pyrimidine. Transversions are interchanges of different classes of base, that is, a purine for a pyrimidine or *vice versa*.

Deamination

Deamination is the removal of an amine group from a nucleotide, which occurs via a hydrolysis reaction. Cytosine is the most frequently deaminated base, at a rate of ~100–500 per diploid human cell per day. This produces uracil (U), a base not naturally present in DNA. Uracil is a pyrimidine which pairs with adenine, so leads to the incorporation of an adenine, rather than guanine, into the opposite strand during replication, inducing a C:G to T:A mutation.

Adenine and guanine can also be deaminated to hypoxanthine and xanthine respectively, both of which pair with cytosine, but at a much lower rate than cytosine. Adenine deamination therefore induces an A:T to G:C mutation, while xanthine forms only two of the usual three hydrogen bonds with cytosine and so may result in the arrest of DNA synthesis. The abnormal bases created by these deamination reactions can be recognised by the cell and repaired. However, 5-methyl cytosine (a cytosine modified by a methyl

transferase enzyme for regulatory purposes) can be deaminated to thymine, inducing a C:G to T:A mutation. As this will not be recognised by the cell as an abnormal base the mutation can become fixed, and methylated cytosines are known mutation hot spots (Denissenko et al., 1997, for example).

Depurination

Depurination is the removal of a purine base from the deoxyribose sugar. Like deamination, it is a spontaneous hydrolysis reaction. Depurination of adenine and guanine can produce an abasic site, that is, a deoxyribose lacking a base. This occurs at a rate of ~18,000 per diploid human cell per day. The genetic information can be recovered from the complementary strand if this occurs in double stranded DNA, but may be mutagenic in single stranded DNA.

Depyrimidination

Depyrimidination occurs by the same mechanism as depurination. Pyrimidine nucleosides are more stable than purine nucleosides, and so the rate of depyrimidination is less, at ~600 per diploid human cell per day.

Oxidation

Oxidation can occur at all four DNA bases, but is most common at guanine due to it having the highest oxidation potential. Reactive oxygen species are generated as by-products of oxygen metabolism and as such oxidative stress is an unavoidable consequence of life in an oxygen-rich atmosphere. It can produce over 80 types of base damage. 7,8-dihydro-8-oxoguanine (commonly shortened to 8-oxoG) is among the most common, occurring at a rate of ~1,000-2,000 per diploid human cell per day. This can pair with adenine as well as cytosine and thereby induce G:C to T:A mutations. Ring saturated pyrimidines, such as thymine glycol, and lipid peroxidation products arise at a rate of ~2,000 and ~1,000 per diploid human cell per day respectively.

Alkylation

Alkylation is the transfer of an alkyl group to bases or the phosphates of the DNA backbone. One of the most vulnerable sites is the oxygen of carbon atom 6 of guanine. This produces O^6-methylguanine which often mispairs with thymine, inducing a G:C to A:T mutation. There is a variety of a endogenous and exogenous alkylating agents.

All of the above types of damage are repaired via the base excision repair (BER) pathway. DNA glycosylase enzymes recognise and remove specific damage types, leaving an apurinic or apyrimidinic (AP) site. In the case of spontaneous depurination and depyrimidination this glycosylase step is not required. The sites are cleaved by an AP endonuclease enzyme, of which there are four classes. All cleave at the phosphate groups $3'$ and $5'$ to the baseless site, but leave either $3'$-OH and $5'$-phosphate or $3'$-phosphate and $5'$-OH termini. Some glycosylases are bi-functional, that is, they also perform the cleavage step and therefore do not require the action of an AP endonuclease. The resulting single strand break is repaired by either short- or long-patch BER. Short-patch repair replaces the single removed nucleotide while long-patch repair replaces a stretch of 2-10 nucleotides.

Incorrect base incorporation

Incorrect bases can be incorporated through the simple insertion of the wrong base by a DNA polymerase during DNA synthesis, resulting in mismatches. Additionally, free bases may be damaged before being incorporated into DNA.

1.2.2.2 Structural alterations

Structural alterations may occur at single or multiple bases, or within the backbone of the double helix. They may arise from reactions within the DNA structure or with outside molecules. They alter the structure of the helix such that it becomes distorted, that is, it deviates from the normal

structure previously discussed. These distortions can block replication and transcription.

Base substitution

Molecules of sufficiently similar structure — base analogues — can be incorporated into DNA in the place of true bases. Although similar enough to be processed by the cell as a real base, they do not behave as such once incorporated into the DNA and so can lead to mistakes during replication. One of the most mutagenic base analogues is 5-bromouracil. This is an analogue of thymine but can mispair with guanine in double stranded DNA. Tautomers of the bases also exist, where the protons occupy different positions in the molecule. These can theoretically cause mispairings between bases, but no evidence has been found that they do so in cells (Pray, 2008).

Intercalation

Flat, polycyclic ring-containing molecules can slip between bases within the DNA double helix. This can cause the deletion or addition of bases during replication, creating frame shift mutations. This can have severe effects on proteins if arising within coding genes.

Inter- and intra-strand cross links

Chemical bonds can be created between nucleotides on the two strands of the double helix (inter), or between nucleotides on the same strand of the helix (intra). These can block the DNA polymerase, thereby causing cell death.

Bulky adducts

Large, bulky molecules can be chemically bonded to DNA, distorting its shape. An example that can cause multiple types of damage is the platinating class of chemotherapeutic drugs (cisplatin, oxaliplatin and carboplatin). These can create crosslinks between guanine bases on the same strand, adjacent to each other or separated by one base, and on opposite strands,

separated by one base. They can also form adducts on individual guanine bases. The absorption of UV radiation can cause the covalent bonding of adjacent pyrimidine bases (see Section 1.4.1.1), creating a DNA adduct without it interacting with another chemical.

These types of damage are repaired in part via the nucleotide excision repair (NER) pathway. The first stage of this pathway, the recognition of a lesion, is split into two sub-pathways. Lesions arising in genes that are being actively transcribed cause the arrest of the RNA polymerase, recruiting the rest of the NER machinery to the site. This is transcription coupled NER (TC-NER). All lesions, including those in genes being actively transcribed, but primarily those in non-coding regions, genes not being transcribed and the non-coding strand of genes being transcribed, can be recognised by specific proteins, including the human DDB and XPC-Rad23B complexes, which recruit the NER machinery to the site. This is global genome NER (GG-NER). Thus there is faster repair of adducts in genes being actively transcribed than the rest of the genome because they are able to be detected by both sub-pathways. After the initial recognition of damage sites the pathways converge. The DNA is unwound at the site of damage by two helicase enzymes, XPB and XPD, and two incisions are made in the strand containing the damage, one either side, by two endonuclease enzymes, XPG and XPF. This creates a strand 25-30 nucleotides long which is removed from the helix. A DNA polymerase then fills this gap, reading from the remaining strand, and DNA ligase seals the nicks. GG-NER is discussed in more detail in Section 1.4.3.

Some types of DNA damage can be directly reversed by enzymes. One such example is photoreactivation, where the energy from light is harnessed by the photolyase enzyme to break apart the covalent bonds between two bases in a cyclobutane pyrimidine dimer (CPD), restoring the original configuration. This enzyme is not present in humans. This type of reaction acts directly on the damaged bases, converting them back to their original forms, and as such does not require a DNA template or the incorporation of new nucleotides.

In some cases DNA synthesis will continue across a damage site, which may be preferable to the cell rather than aborting the synthesis to repair the lesion. This is achieved via the translesion synthesis (TLS) pathway. This requires the use of specialised polymerases to synthesise across the damage. In general, these polymerases have low fidelity, that is, they often incorporate the wrong base with an undamaged DNA template. However, they are much more efficient at incorporating the correct base at damage sites than normal polymerases. They often have larger active sites to allow the adduct to fit. At sites of damage the PCNA protein is ubiquitinated, which signals the polymerase switching.

1.2.2.3 Strand breakages

Strand breakages are the introduction of nicks into the phosphodiester backbone of DNA. They may arise in one or both strands.

Single strand breaks

Single strand breaks (SSBs) are the breakage of the backbone of one strand of the double helix which can be induced by certain wavelengths of radiation and as a result of other cellular repair processes. They leave the DNA molecule intact by virtue of the remaining second strand. Two single strand breaks close together on opposite strands however can lead to a more severe double strand break (DSB), as can an approaching replication fork. Cells have therefore developed efficient systems to detect and repair these breaks to prevent them becoming more severe. Poly(ADP-ribose) polymerase (PARP) is one such sensor, which binds to SSBs with high affinity and acts as a signal for repair. One important protein in the repair process, which interacts with PARP, is XRCC1. This has multiple roles in SSB repair and is thought to act as an important scaffold protein in interacting with several other proteins with roles in the repair pathway, including end processing, gap filling and ligation.

Double strand breaks

DSBs form when the backbones of both strands of the double helix are severed in close proximity, creating two separate DNA molecules, which can lead to chromosomal rearrangements. There are two main mechanisms for the repair of these breaks: directly rejoining the free ends of the two molecules (non-homologous end joining; NHEJ), which does not necessarily join the correct two free ends, or utilising regions of similar or identical sequence to the free ends elsewhere in the genome to aid the joining of the free ends to what is likely to be the correct sequence (homologous end joining; HEJ).

HEJ uses a short section of a sister chromosome with sequence homology to the free DNA ends to rejoin the ends together without loss of information. The MRN-complex recognises DSBs and locates to their sites, and mediates the processes required for HEJ to take place. The $5'$ ends of the DNA strands are resected, leaving $3'$ overhangs. An overhang can then invade the region of the sister chromosome with sequence homology, base pairing with the homologous sequence and displacing the complementary strand. A DNA polymerase can then extend the $3'$ ends based on the complementary sequences of the invaded chromosome. This forms two Holliday junctions, the resolution of which produces either cross-over or non-crossover products. This utilisation of identical sequences means that repair by this method is error free.

NHEJ occurs when a sister chromosome is not present, that is, before DNA replication has taken place and therefore represents the major DSB repair pathway in humans. Two free ends are simply ligated together, mediated by Ku proteins. Small regions of homology present on single stranded DNA (ssDNA) at the ends of the strands, which can be as little as one nucleotide in length (serendipitous microhomologies) are utilised for this process. This is therefore an error prone repair process but as so little of the human genome is coding it seems that this small loss of information is generally tolerated. However, there is no way of knowing that the two free ends being joined by this process are correct, that is, were originally joined together before the break, so this type of repair can induce chromosomal rearrangements, a

hallmark of cancer.

1.2.3 Consequences of defective DNA repair

Repair of DNA damage is vital for cellular proliferation and organism survival. Organisms with serious DNA repair defects often do not survive beyond the very early stages of life. There are many congenital diseases of repair pathways in humans which confer increased risk of cancers or/and reduced lifespans. Additionally, acquired diseases can arise following the incorrect repair of a damage event in an otherwise healthy individual.

1.2.3.1 Congenital diseases

Over 20 diseases are associated with mutations in around 50 genes involved in DNA repair (reviewed by Lehmann and O'Driscoll, 2010). The following information is taken from this review.

Three main diseases are associated with defects in the NER pathway: xeroderma pigmentosum (XP), Cockayne syndrome (CS) and trichothiodystrophy (TTD). XP arises from mutations in any one of the seven XP genes, designated A to G (see Section 1.4.3), and is characterised by sensitivity to sunlight, skin pigmentation changes and multiple skin cancers on areas exposed to sunlight. It is also associated with neurological abnormalities in some cases. CS is associated with mutations in the CSA and CSB genes. The proteins encoded by these genes are required to displace the stalled polymerase at damage sites in genes being actively transcribed and as such this disease is specifically related to TC-NER. The symptoms include dwarfism, severe physical and mental retardation, neurological and retinal degeneration, ataxic gait, deafness and sun sensitivity. Notably, there is no skin pigmentation or increased risk of skin cancer. It is thought the symptoms of the disease are related to the accumulation of oxidative damage, the removal of which CSA and CSB are also involved with. TTD arises from mutations in the XPB and XPD genes and is characterised by sulphur-deficient, brittle hair, ichthyotic skin, beta thalassaemia trait, physical and mental retardation and sun sensitivity, but again no skin pigmentation or increased skin cancer

risk. It is thought these symptoms are related to defects in transcription, in which XPB and XPD also play a role in initiation, whereas in XP they are related to defects in repair.

Defects in the TLS pathway can also cause an XP variant, XP-V, with 20% of patients having normal NER but mutations in polymerase η. This polymerase is able to carry out TLS past UV-induced and other DNA adducts. Without its activity cells cannot effectively repicate across damage when necessary.

Patients with clinical symptoms related to defects in HEJ and NHEJ are rare. The BRCA1 and BRCA2 genes, mutations in which are linked to familial breast cancer, play roles in HEJ. Several diseases are linked to defects in the signal transduction cascade activated by DNA breaks. The ATM and ATR kinases coordinate these downstream events, activated by DSBs and ssDNA respectively. Ataxia-telangiectasia, ataxia telangiectasia-like disorder and Nijmegen breakage syndrome are linked to defects in ATM, with symptoms including increased cancer predisposition. Seckel syndrome is associated with ATR mutations.

Mutations in three of the five RECQ genes, involved in homologous recombination and suppression of illegitimate recombination, result in several disorders. Bloom syndrome (BLM) is characterised by a reduced stature, reduced fertility, chromosome abnormalities and high cancer incidence. These cells have a very high incidence of sister chromosome exchanges. Werner syndrome (WRN) is characterised by features of premature aging and an increased incidence of soft tissue carcinomas. These cells show a high frequency of illegitimate recombination. Rothmund-Thompson syndrome is linked with abnormalities of the skin and skeleton and an increase in cancer incidence.

Fanconi anaemia patients suffer from defects in interstrand cross link repair. It results from mutations in any one of 13 FANC genes. Cells are hypersensitive to cross linking agents and slightly sensitive to ionising radiation. Clinical symptoms are progressive aplastic anaemia, skeletal abnormalities and lymphoid malignancy.

Defects in mismatch repair cause the autosomal dominant condition hereditary nonpolyposis colon carcinoma (HNPCC). The majority of cases arise

from mutations in the genes MSH2 — involved in mismatch recognition — and MLH1 — recruited to mismatches once recognised. The reason for tumours arising only in the colon when mismatch repair is vital in all cells is currently unclear, but is thought to be related to the high replicative turnover in the gut layer.

Various diseases are associated with defects in the various parts of BER. Spinocerebellar ataxia with axonal neuropathy (SCAN1), a disease associated with neurodegeneration, results from mutations in the TDP1 gene. This is involved in removing topoisomerase I enzymes from DNA, the inhibition of which results in a reduced ability to repair single strand breaks. Ataxia-oculomotor apraxia type 1 (AOA1), a disease associated with cerebellar atrophy and sensorimotor neuropathy, results from mutations in the APTX gene. This is involved in the removal of intermediates in the repair of DNA breaks, inhibition of which it is thought causes the build up of breaks in cerebellar DNA.

Other diseases include cerebro-oculo-facio-skeletal (COFS) syndrome, resulting from mutations in the *CSB* gene and PIBIDS — named after the features of Photosensitivity, Ichthyosis, Brittle hair, Intellectual impairment, Decreased fertility and Short stature — which is considered to by synonymous with the photosensitive form of TTD (Friedberg et al., 2006). Because these diseases are so rare it can be difficult to categorise and compartmentalise them, and there is likely a spectrum of diseases with overlapping clinical features (Friedberg et al., 2006).

1.2.3.2 Acquired diseases

Cancers can arise in any person. Certain DNA repair defects can cause increased incidence or risk of cancers, as outlined in the previous section, but DNA repair errors in otherwise healthy individuals can also lead to cancer formation. There are a number of unusual properties a cell must gain in order to make the transition from normal to cancerous. These changes take the cells beyond the normal controls and allow them to proliferate in the uncontrollable manner associated with cancers. Multiple functions exists in

cells to prevent them acquiring these properties, but occasionally these can fail. These properties have been reviewed in two papers by Hanahan and Weinberg (2000, 2011) and are outlined below.

Self sufficiency in growth signals

Normal cells require extracellular growth signals to make them actively proliferate. Without these signals the cells remain quiescent. This homeostatic mechanism ensures the proper behaviour of the various cell types within a tissue. Various mutations have been identified in cancerous cells which enables them to evade this control. These include the cell abnormally synthesising growth factors to which it itself responds or altering the cellular circuitry such that the response to growth signals is activated even without the presence of the growth signals. This allows the cells to be in a continuous proliferative state.

Insensitivity to antigrowth signals

Anti-growth factors also exist, with the reverse function of growth factors, that is, they signal cell quiescence. Different types of anti-growth factors can induce temporary or permanent quiescent states. Cells avoid these signals by downregulating or displaying dysfunctional cell surface receptors or altering the cellular circuitry such that no response to anti-growth signals is made. This allows cells to ignore growth-inhibiting signals from surrounding cells.

Apoptosis evasion

Apoptosis is programmed cell death, an essential process by which cells 'dismantle' themselves in precisely choreographed steps and their components are taken up by surrounding cells. Various events can signal apoptosis including DNA damage and hypoxia, as well as controls to maintain tissues in the correct configuration. The p53 tumour suppressor is heavily involved in the apoptotic pathway and its inactivation is seen in over half of human cancers (Harris, 1996). This evasion allows the cells to persist even in situations

where they would normally be signalled to destroy themselves (reviewed by Fulda, 2010).

Limitless replicative potential

Even with the three capabilities listed above cells cannot replicate forever. Cells have a finite replicative potential, after which they enter senescence. Circumventing this allows cells to replicate for a further period until they enter crisis, which is characterised by cell death. This limit is imposed by telomeres, short sections of which are lost with every cell cycle (reviewed by Stewart and Weinberg, 2002). This progressive erosion means that the whole telomere will eventually be lost, causing end-to-end chromosomal fusions and the death of the cell. Unlimited replicative potential can be acquired by modifying telomeres in a way that means they never become lost. Nearly all cancer cells show such telomere maintenance (Shay and Bacchetti, 1997).

Sustained angiogenesis

Angiogenesis is the production of new blood vessels. In order for a tumour to grow it must have the ability to create new blood vessels to carry oxygen and nutrients to all cells within it. Initially the cells do not have this ability. Various mechanisms are acquired in order to stimulate angiogenesis, including the production of angiogenesis-initiating signals (reviewed by Kerbel, 2000).

Tissue invasion and metastasis

Once cancerous cells become free of the normal growth constraints and become immortal, they may become limited in the space available for them to physically grow into. Overcoming this involves producing cells that can move away from the original tumour and form new tumours elsewhere (metastasis) and developing the ability to grow into and through surrounding tissues (invasion), traits not associated with normal cells. The majority — as much as 90% — of human cancer deaths are due to metastases (reviewed by Chaffer and Weinberg, 2011). These traits are acquired by modifying proteins involved in tethering cells to their surroundings, which have various regulatory

functions, and upregulating extracellular proteases, which can degrade and disrupt surrounding cells.

Reprogramming energy metabolism

In order for cancer cells to proliferate at the faster rate allowed by the above modifications, they must also change the way they produce energy. Most cancer cells limit their metabolism to glycolysis, even in the presence of oxygen, creating a state referred to as "aerobic glycolysis" (reviewed by Hsu and Sabatini, 2008). This process produces less energy than conventional metabolism and so is commonly accompanied by upregulation of glucose transporters. The reason for this is unclear but may allow glycolysis intermediates to be diverted into other biosynthetic pathways, including nucleosides and amino acids, aiding the production of new cells.

Evading immune detection

The immune system is highly effective at removing malignant cells. It is thought that the majority of cancerous cells are detected and eliminated by the immune system and therefore that cancerous cells that survive this process do so by somehow evading the immune system (recent evidence discussed by Bindea et al., 2010). There is some evidence to show that the immune system plays an important role in cancer removal in mice.

1.3 Measuring DNA damage

In order to investigate DNA damage and its subsequent repair, methods of measuring the amount of DNA damage present in a cell or cells are required. Repair is estimated by measuring damage at time points after exposure to a particular damaging agent. Any decrease in the level of damage over time can be indicative of repair having taken place. Repair assays can be divided into two broad categories: high and low resolution. This refers to the ability to resolve the sites at which damage occurs. Low resolution techniques provide a very limited, if any, way of determining where in the genome damage

occurs. Many of the assays allow total damage to be measured, that is, they show damage throughout entire genomes, but the sites at which the damage is located within the genome cannot be determined. Conversely, high resolution techniques allow the locations of damage to be determined to within a number of base pairs. Typically, these techniques allow only a small section of a genome to be investigated at a time, and so do not enable genome wide damage to be investigated. The following sections outline the techniques that have been used to measure DNA damage, with specific reference to their use in measuring CPD incidence where appropriate.

1.3.1 Low resolution techniques

Early investigations of DNA damage were low resolution and whole genome. The first used radioactive labels to indirectly measure the presence of damage. For example, an alkaline sucrose gradient method developed by McGrath and Williams (1966), used to separate long strands of (radioactively labelled) genomic DNA by length, thus producing a distribution of fragments by molecular weight, has been used to measure the repair of such things as *Escherichia coli* DNA exposed to x-ray radiation (McGrath and Williams, 1966) and lymphoma cells treated with alkylating agents (Peterson et al., 1973). Here the presence of damage is indicated by the amounts of differing length DNA fragments. Radioactive labelling can also be used to investigate the removal of CPDs in *E. coli* (Setlow and Carrier, 1964), for example, by damaging cells containing labelled thymine and monitoring its replacement, as it is repaired, with unlabelled thymine.

The first assay to demonstrate NER in mammalian cells in culture used a radiolabelled DNA precursor (Rasmussen and Painter, 1964). Only during S phase would this normally be expected to be incorporated into the genome. Dipping slides containing cells into a photographic emulsion for autoradiography highlights areas of radiolabelling by the accumulation of silver grains in the nucleus. Cells in S phase show this labelling, while those in other phases do not. Following UV irradiation all cells show labelling, indicative of DNA synthesis as a result of NER.

Pulsed field gel electrophoresis (PFGE) can be used to monitor double strand breaks (first used by Contopoulou et al., 1987). This technique allows individual chromosomes to be resolved as individual bands on the gel, allowing their repair to be monitored separately. This technique has an increased resolution from whole genome to whole chromosome, but is still considered low resolution as the sites of damage cannot be resolved. The FAR (fraction of activity released) assay (Rydberg et al., 1994) provides another measure of double strand breaks using PFGE, by measuring the amount of DNA that enters the gel from a plug containing a cell sample.

A very simple gel based assay allows for double stranded breaks to be visualised by separation by electrophoresis. The more breaks present the more short fragments present, which travel further on the gel creating a 'smear.' With no or few breaks only long fragments are present which do not travel very far on the gel and so no, or only a small, smear is created.

The alkaline unwinding assay can be used to monitor single strand breaks in DNA (first used by Elkind and Chang-Liu, 1972). Here chemical conditions are made such that the DNA helix can unwind from sites of single strand breakages, causing the two strands to separate. The amount of single stranded DNA present is then indicative of the number of breaks present. This technique can also be used to monitor NER (Erixon and Ahnström, 1979), by measuring the short fragments of excised DNA. During the NER process short fragments of DNA containing damage are excised, following the introduction of nicks either side of the damage. Monitoring these short, single stranded fragments shows the amount of NER activity. An alternative method, alkaline filter elution (developed by Kohn and Grimek-Ewig, 1973), is based on the same principle, but without the centrifugation required by the previous two assays. Instead filters are used to separate the DNA fragments in a more reproducible manner. This technique has also been used to monitor NER (Fornace et al., 1976).

Cadet et al. (1983) discuss some other methods of analysing NER (available at the time) which either indirectly measure repair, such as a method which separates DNA containing radioactive nucleotides removed through repair in isopycnic gradients (Pettijohn and Hanawalt (1964) cited in Cadet

et al., 1983), or directly analysing sites of damage with endonucleases that cut at damage sites, such as with alkaline elution (Fornace Jr (1982) cited in Cadet et al., 1983). They address the problem that these techniques do not allow identification of the types of lesions present by applying HPLC (high performance liquid chromatography) to the detection of damage which is able to detect the different dimers produced at the four dipyrimidine sites (TT, TC, CT and CC) following UV radiation.

The comet assay, first described in 1990 (Olive et al., 1990), is a popular method still used today and has been reviewed extensively (see, for example, Collins, 2004). It is a single cell assay which uses an electrophoretic gel to separate damaged (short) DNA fragments from undamaged (long) genomic DNA. As well as detecting single and double strand breaks, other damages such as cross links and base damage can be detected (Speit and Hartmann, 2006). The shape of the resulting DNA resembles a comet, with the undamaged DNA in the body and the damaged DNA forming the tail, the length and shape of which can be used to describe the damage. Software is available to automate this process (discussed, along with the rest of the protocol, in Olive and Banáth, 2006). More recent adaptations of the technology use microwell arrays, where a single comet can be produced in each of 24 or 96 wells (Wood et al., 2010). This is amenable to high throughput screening, enhancing the generation of results. The comet assay can be used to monitor NER (Myllyperkiö et al., 2000, for example) by measuring the single stranded excised DNA fragments.

An alternative to the comet assay for some types of DNA damage is the halo assay (Sestili et al., 2006). Whereas the comet assay uses an electric field to 'pull' DNA out of cells, the halo assay relies on the diffusion of fragments out of the cell. Like the comet assay, it is named after the shape produced as the damaged DNA migrates out of the cell, which surrounds each cell in a ring as it is not attracted in any one direction. This assay can also be adapted to high throughput screening (Qiao et al., 2011).

The TUNEL assay (terminal deoxynucleotidyl transferase-mediated dUTP-biotin nick end labelling, developed by Gavrieli et al., 1992) can be used to visualise DNA breaks with a microscope by the attachment of a fluorescent

molecule to the free ends.

Various formats of mass spectrometry (MS), such as gas chromatography (GC, first used by Dizdaroglu, 1984, to investigate radiation induced damage) and HPLC-electrospray (first used by Wolf and Vouros, 1994, to investigate deoxynucleoside adducts) have been used to detect particular types of DNA damage. It has been used to detect CPDs (for example, by Douki et al., 2000, to analyse the formation of different thymine dimers). Although very sensitive and specific, MS assays can be limited in the amount of material that can be tested and the damages that can be detected. Cost and availability can be prohibitive in widespread use.

Electrochemical methods, which use biosensors, have been used to detect radiation- (Wang et al., 1997) and chemical-induced (Zhou and Rusling, 2001) DNA damage. These methods are specific, and therefore limited, to the damage they are developed to detect. A different sensor would be required to analyse each different type of damage, which limits their widespread application.

Immunological assays use antibodies against antigens of interest, in this case DNA damages, to examine the presence of the damage. Immuno-slot blots are a common sensitive and specific assay. They are a simple alternative to the western, northern and Southern blots, omitting the separation stage of these techniques. An antibody against the damage of interest is used to identify its presence in a sample, which is simply transferred or 'dotted' onto a membrane. The lack of a separation stage means products of different sizes cannot be distinguished, which does not represent a problem if the aim of the assay is to determine the total amount of damage present. The technique has been applied to investigating UV damage (for example, Perdiz et al., 2000, analysed the formation of the different types of UV-induced lesions).

Other types of immunological assay, including the RIA (radio immunoassay) and ELISA (enzyme linked immunosorbent assay) can also be used quantify total damage amounts. The multi-well format of ELISAs makes them useful for high throughput analyses. A RIA, for example, was used to show that 6-4 photoproducts are removed from DNA faster than CPDS (Mitchell et al., 1985).

1.3.2 High resolution techniques

More recent DNA damage measurement techniques have allowed analyses focussed on particular genome regions to be carried out. The PCR reaction can be used to quantify damage amounts, by taking advantage of the fact that the polymerase cannot replicate across sites of damage. This was first demonstrated by Moore and Strauss (1979) and first used as a quantitative PCR assay to detect CPDs by Govan III et al. (1990). Using primers to a particular region of interest allows the presence of damage in the region to be determined. The reduction in reaction yield compared to undamaged DNA is indicative of the amount of damage present in the region. Although not able to identify the locations of damage events, this type of assay allows damage in specific regions to be investigated and is therefore higher resolution than the previously discussed assays.

Another assay uses enzymes that cut at sites of damage, with true high resolution results. With respect to CPDs, a CPD specific nuclease can be used to cut at CPD sites. A method developed by Teng et al. (1997) uses phage T4 endonuclease to cut at damage sites and probes specific to two ends of a region of interest to separate the two DNA strands of the region from the rest of the genomic DNA. The lengths of the separated fragments thus vary in length, depending on whether and where they are cut by the endonuclease. Undamaged fragments will be full length, whereas the lengths of shorter fragments represent the position of the closest cut damage site to the probe. These different lengths of fragment can be radiolabelled and resolved on a gel. A sequence ladder run on the same gel allows precise identification of the sequence at which damage has been detected. Running several samples from different time points allows repair to be visualised, as the number of short fragments decreases. The separation of the two strands allows repair rates of these to be analysed separately, which allows, for example, the different repair rates of TC-NER and GG-NER to be visualised, with the strand being analysed containing a gene (or section thereof) being actively transcribed, and its complement strand, which is not transcribed.

A novel use of microarray technology developed in our laboratory (Teng

et al., 2011) allows high resolution, genome wide measurements of DNA damage. This overcomes the problem of the above techniques, which are limited to short (several hundreds of base pairs) regions of a genome due to the limiting size of the gels. In the same way that ChIP-chip can be used to identify the binding locations of proteins with an antibody raised against them, an antibody against damage, CPDs in this case, is used to immunoprecipitate DNA containing damage. Applying this material to an appropriate microarray covering a whole genome, or section thereof, allows the relative amount of damage in a given region to be determined. This is discussed in more detail in Chapter 6.

1.4 CPDs and NER: a paradigm

UV radiation is used in our laboratory as a way of generating DNA damage, namely CPDs, which is measured over time to investigate its repair by NER. This section outlines this paradigm.

1.4.1 Ultra violet radiation

UV radiation is a well studied, relevant DNA damaging agent, as many organisms, including humans, are exposed to it on a regular basis in solar radiation, and have been for millennia. The UV spectrum is divided into three segments based on wavelength: UV-A (320–400 nm); UV-B (295–320 nm); and UV-C (100–295 nm). The ozone layer absorbs UV at wavelengths up to around 300 nm, meaning little UV-C radiation reaches the Earth's surface. The UV absorption peak of DNA is 260 nm, so the most DNA-damaging wavelengths are filtered out by the atmosphere.

In the laboratory, UV lamps provide a convenient and readily available method of inducing DNA damage in cells in order to investigate repair. This is often done with wavelengths in the UV-C region of the spectrum which, although not strictly environmentally relevant, induces the same damages as UV-A and UV-B, but at a higher efficiency. All UV-induced damage referred to in this thesis was produced with 254 nm radiation at a rate of 100 J/m^2.

This produces, on average, 1 damage event per 1,000 bp (Courcelle et al., 2006).

1.4.1.1 Cyclobutane pyrimidine dimers

The absorption of UV radiation by DNA causes alterations to its structure, called photoproducts. These include pyrimidine-pyrimidone (6-4) adducts ((6,4)PPs), spore photoproducts, purine-containing lesions, pyrimidine hydrates, thymine glycol and CPDs. These are the most frequently formed photoproducts and are used in our laboratory to investigate damage and repair.

CPDs are formed by covalent linkages between adjacent pyrimidines, forming a 4-membered ring structure resulting from saturation of the pyrimidine 5,6 double bonds (Figure 1.5). The two pyrimidines are unable to form correct pairing with their complementary bases. This lesion causes a distortion in the shape of the DNA helix of approximately 30° (Park et al., 2002, Figure 1.6). This distortion inhibits DNA replication and transcription and thus is lethal to cells if left unrepaired. It is this distortion, specifically the resulting reduced base stacking, that is thought to be recognised by repair enzymes (Yang, 2007).

CPDs form between two adjacent pyrimidines on the same DNA strand, but the frequency of formation between different dipyrimidine combinations varies. The ratio of formation of dimers at TT:TC:CT:CC sites in plasmid DNA with UV-C radiation is 68:16:13:3 (Mitchell et al., 1992).

1.4.2 *Saccharomyces cerevisiae* as a model organism

A model organism is an organism used in place of humans to investigate and gain insights into human diseases. Several such organisms are used in biomedical research. The budding yeast *Saccharomyces cerevisiae* has been used as a model organism for many years due to it being easy to grow and manipulate in the laboratory, particularly with respect to genetic analyses. The yeast genome is approximately 13 Mbp, 270 times smaller than the human genome, and codes for around 6,000 genes. With respect to NER,

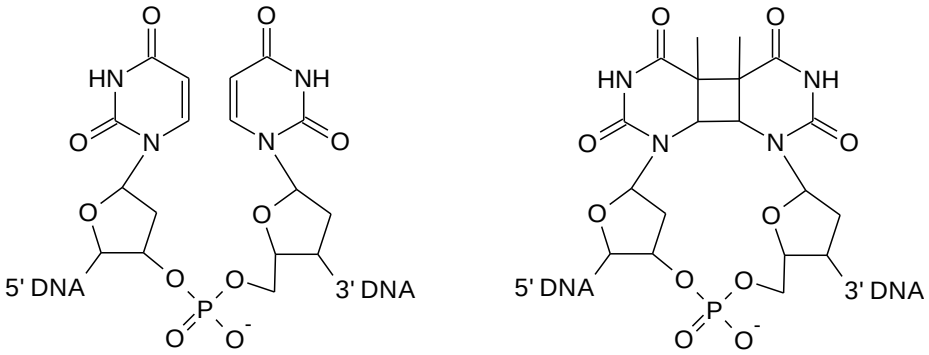


Figure 1.5: The cyclobutane pyrimidine dimer: Molecular structure of two adjacent thymine nucleotides (left) and a cyclobutane pyrimidine dimer (right) formed from saturation of the pyrimidine 5,6 double bonds.

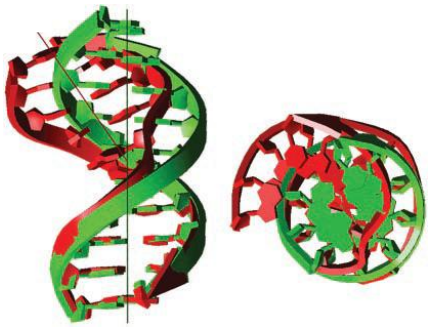


Figure 1.6: Kinked DNA molecule: Side (left) and top (right) view of the kink induced in the DNA molecule by the presence of a CPD (Park et al., 2002). Damaged DNA (red) bends approximately 30° relative to undamaged DNA (green).

the great majority of the proteins identified in yeast cells have homologous partners in human cells.

1.4.3 Nucleotide excision repair

NER is the pathway by which CPDs and a variety of other DNA lesions are removed. The speed of removal varies with different types of lesion. (6-4)PPs, for example, which distort the DNA to a greater extent than CPDS, are repaired 5–10 times faster than CPDs (Friedberg et al., 2006). The position of incisions relative to the lesion and the length of the excised fragment also varies slightly depending on the type of lesion being repaired (Friedberg et al., 2006). GG-NER occurs throughout a genome, with no specificity for any particular region(s). It is complemented by TC-NER which occurs only within genes that are being actively transcribed, when the polymerase stalls at the site of a lesion. The repair process in the two pathways is identical; they differ only in this initial recognition step.

The whole NER process has been elucidated for prokaryotic cells (reviewed by Truglio et al., 2006), which requires only three proteins: UvrA, UvraB and UvrC. UvrA and UvrB are involved in damage detection and verification. UvrB and UvrC perform the damage removal. A trimer of two UvrA and a UvrB molecules detects DNA damage via the UvrA dimer. The trimer then binds the damage via UvrB, whereupon the UvrA dimer leaves the complex. A UvrC molecule then forms a dimer with the UvrB molecule bound to the damage and performs the incisions at both the $5'$ and $3'$ sides. UvrD (DNA helicase II) then removes the incised fragment, allowing DNA polymerase I to remove the bound UvrB molecule and fill the gap. Finally DNA ligase seals the newly synthesised DNA ends. NER in eukaryotic cells follows the same sequence of events but requires upwards of 30 different proteins, reflecting the more complex environment of the eukaryotic cell in which they have to function.

The actual repair of lesions by eukaryotic NER on naked DNA is also reasonably well understood, and an *in vitro* system of purified yeast proteins able to fully repair damage on naked DNA has been available for a number

of years (Prakash and Prakash, 2000). The repair process itself, which occurs once the GG-NER and TC-NER pathways converge, is briefly described in Section 1.4.3.2. It is how the lesion is recognised and accessed in the complex context of chromatin in the GG-NER pathway that is the current focus of research in this area, which is outlined in the following section.

1.4.3.1 Lesion recognition

In TC-NER, RNA polymerase II (RNAPII) stalls at sites of lesions on the strand of DNA being transcribed (Donahue et al., 1994). Genes being transcribed by RNA polymerase I and III are not subject to TC-NER (Tornaletti and Hanawalt, 1999, and references therein). Several other obstacles can cause the temporary stalling of RNAPII, such as secondary DNA structures, which is overcome by transcription elongation factor SII (TFIIS), but this factor still does not allow transcription across a CPD (Friedberg et al., 2006, and references therein). It is this permanent stalling that directs the rest of the NER machinery to the site, initiating TC-NER, although the exact mechanism of this remains unclear in both yeast and humans (Reed, 2011). In human cells, three other proteins are required for TC-NER: CSA, CSB, and XAB2 (Nakatsu et al., 2000). In yeast, Rad26, the homologue of human CSB, and Rpb9, a subunit of RNAPII, are required for TC-NER (van Gool et al., 1994; Li and Smerdon, 2002). Rad28, the closest homologue of human CSA, is not required for strand specific repair (Bhatia et al., 1996).

Lesions in all regions of the genome, including those on strands being actively transcribed, are repaired via the GG-NER pathway. In yeast this requires the Rad16, Rad7 and Abf1 proteins (Verhage et al., 1994; Reed et al., 1999) although the mechanism of recognition has yet to be elucidated. These proteins form a complex and are required to generate superhelical torsion in DNA, which is required for NER (Yu et al., 2004). This requires Abf1's ability to bind to specific binding sites throughout the genome (Yu et al., 2009). A Rad7/Rad16 complex specifically binds UV damaged DNA in an ATP dependent manner (Guzder et al., 1997). Rad7 and Rad16 mutants show reduced, but not eliminated, repair following UV irradiation (Miller

et al., 1982; Prakash, 1977), because repair by TC-NER can still take place in their absence.

Rad16 is a member of the SWI/SNF (switch/ sucrose non-fermentable) superfamily of proteins (Bang et al., 1992). This family of proteins have the ability to remodel chromatin through their DNA translocase activity (see, for example, Wilson and Roberts, 2011). Rad16's DNA translocase activity allows the complex to create superhelicity in DNA, which is required for the excision of the damage-containing fragment (Yu et al., 2004), but it does not have the ability to slide nucleosomes (Yu et al., 2009), which is a property of some other SWI/SNF proteins. This is suggested not to permit unregulated gene transcription upon repair of repressed regions of the genome (Yu et al., 2009).

UV irradiation leads to histone H3 hyperacetylation (H3Ac) and chromatin remodelling in yeast (Yu et al., 2005). (Histone acetylation has long been known to be required for efficient repair, first shown in human cells, for example, Smerdon et al. (1982); Ramanathan and Smerdon (1989)). This hyperacetylation requires Rad16 and Rad7 (Teng et al., 2007; Yu et al., 2011). Regions of hyperacetylation induced in the absence of Rad16 and Rad7 exhibit Rad7- and Rad16-independent repair (Teng et al., 2007), showing that histone hyperacetylation is an important first step in the repair process.

Rad7 and Rad16 control the H3Ac level by controlling the accessibility to chromatin of the histone acetyl transferase (HAT) Gcn5 (Yu et al., 2011). Gcn5 has the ability to acetylate N-terminal lysines on histones H2B and H3 (Grant et al., 1997) and is involved in transcriptional regulation (reviewed in Lee and Young, 2000). The ATPase and RING domains of Rad16 play an important role in this (Yu et al., 2011). Mutations introduced into each domain individually result in intermediate UV survival, while a double mutant is as sensitive as a *Rad16* deletion. The two single mutants do not display very different phenotypes to the wild type with respect to UV-induced H3Ac change and Gcn5 occupancy. Conversely, the H3Ac and Gcn5 occupancy increases seen in the wild type are abolished in the double mutant. In keeping with these results, repair of CPDs in the two single mutants is of an intermediate level, while in the double mutant it is reduced to the level seen in

the *Rad16* delete strain.

Investigation of the Abf1 member of this protein complex is the main biological avenue of investigation in this thesis. It is hypothesised to function to sequester the GG-NER complex at points throughout the genome, ready for repair to take place where necessary (Reed et al., 1999; Yu et al., 2009). Abf1 is described in more detail in Section 1.4.4.

No human homologues of Rad7, Rad16 or Abf1 have been identified, but the human proteins DDB1 and DDB2 share several functional similarities to yeast's Rad7 and Rad16 (Reed, 2011).

1.4.3.2 Lesion repair

The repair of lesions by NER can be broadly split into five steps: damage recognition, incision either side of the damage site, excision of the resulting damage containing fragment, the filling of the resulting gap and ligation of the newly synthesised strand. The first step requires the assembly of a number of proteins at the damage site, termed the pre-incision complex (PIC). An essential component of this is Rad14, which binds specifically to UV-damaged DNA (Guzder et al., 1993). The human homologue, XPA, also preferentially binds UV-damaged DNA (Jones and Wood, 1993). Although not required for the initial damage recognition, this protein is essential for repair, with *Rad14* delete yeast and human XP-A patients having no functional NER (Guzder et al., 1993; Tanaka et al., 1990).

A second essential component of the PIC is Replication protein A (RPA), a 3-subunit complex able to bind ssDNA, composed of RFA1, RFA2 and RFA3. Human RPA contains the proteins RPA1, RPA2 and RPA3. This complex is involved in processes which have ssDNA intermediates, such as replication and transcription, as well as playing an essential role in NER (Huang et al., 1998; Coverley et al., 1991), in which it has a role in positioning the nucleases which perform the incisions (De Laat et al., 1998). It has a preference for binding DNA damaged by a variety of agents (Friedberg et al., 2006, and references therein).

TFIIH is the final essential component of the PIC. It is a complex made

up of nine proteins in two units and has dual roles in transcription initiation and DNA repair (Takagi et al., 2003). Several of the proteins are essential for efficient NER but only as part of the complex (Feaver et al., 2000).

Rad4 and Rad23 (human XPC and HR23B) form a complex which binds damaged DNA with a much higher preference than undamaged DNA *in vitro* (Guzder et al., 1998; Masutani et al., 1994). Rad23 contains a ubiquitin-like domain at its amino terminus (Watkins et al., 1993) which allows it to interact with the 26S proteasome (Schauber et al., 1998). Mutations in the proteasome can reduce cell survival following UV irradiation but blocking its protein degradation activity does not affect NER *in vitro* or *in vivo*, showing independent functions of the proteasome in the two processes (Russell et al., 1999).

Following the assembly of the PIC, the DNA section containing the damage fragment can be cut. The two incisions are made by two different proteins or protein complexes, one each for the 5′ and 3′ sides of the lesion. The Rad1-Rad10 (human XPF-ERCC1) complex cuts at the 5′ side (Bardwell et al., 1994; Park et al., 1995) and Rad2 (human XPG) at the 3′ side (Harrington and Lieber, 1994; O'Donovan et al., 1994).

Rad1 and Rad10 are DNA binding proteins with a preference for ssDNA but not damaged DNA (Sung et al., 1993, 1992). The complex binds to Rad14 suggesting that this is the means by which it is directed to sites of damage (Guzder et al., 1996).

The Rad2 protein forms a complex with TFIIH (Habraken et al., 1996). It has ssDNA endonuclease activity but no specificity for damaged DNA (Habraken et al., 1993). TFIIH is required to recruit XPG to the PIC (Zotter et al., 2006).

1.4.4 The Abf1 protein

The autonomously replicating sequence (ARS) binding factor 1 (Abf1) protein is an abundant, multifunctional *S. cerevisiae* protein which binds at many locations throughout the genome. It was first characterised in the late 1980s as a protein with a regulatory role of the silencing of the HML and

HMR mating type loci, by binding to specific sequences at the E and I elements (Shore et al., 1987; Buchman et al., 1988). It has since been shown to have a number of additional roles and as such has been classed as a general regulatory factor (GRF) (Chasman et al., 1990). GRFs are characterised as being abundant, essential and multifunctional, acting as obligate synergisers with many binding sites throughout the genome (Fourel et al., 2002).

Abf1 is essential for the viability of yeast cells (Rhode et al., 1989) and is often associated with the binding of the Rap1 protein, another GRF (see Yarragudi et al., 2007, for example). It has roles in DNA replication, positive and negative regulation of transcription, chromatin silencing and remodelling, NER, genome partitioning and telomere maintenance (Buchman et al., 1988; Fourel et al., 2002; Lascaris et al., 2000; Loo et al., 1995; Miyake et al., 2002; Reed et al., 1999; Rhode et al., 1992).

Abf1's role in replication is context dependent. Early investigations showed that plasmids with Abf1 binding sites at their replication origins showed reduced stability when these binding sites were mutated, showing Abf1 stimulates the efficiency of replication but is not essential for it (Walker et al., 1989, 1990). Cells expressing the temperature sensitive DNA binding mutant *abf1-1* also showed reduced stability of such plasmids at the restrictive temperature (Rhode et al., 1992). The Abf1 binding site of ARS1 can be functionally replaced by the binding sites of the GRFs Rap1 and Gal4 (Marahrens and Stillman, 1992). Furthermore, adding an Abf1 binding site to certain ARSs without one can reduce replication efficiency (Kohzaki et al., 1999). The precise nature of the role played by Abf1 in these different contexts has not yet been elucidated.

Abf1 alone is a weak transcription factor, and strong transcriptional activation is only achieved in conjunction with other transcription factors (Buchman and Kornberg, 1990). It also plays a negative role in transcription in some cases, with the removal of an Abf1 binding site increasing the level of transcription of some genes (Einerhand et al., 1995, for example).

Abf1 has been shown to be important for maintaining chromatin structure at many locations in the genome, where mutation of a binding site causes a loss in the positioning of nucleosomes (Lascaris et al., 2000). It is also

associated with nucleosome depleted regions (Badis et al., 2008), which may function to allow other transcription factors to bind to their intended sites. It also has been shown to be important in maintaining barriers between different chromatin states (Fourel et al., 2002).

A multitude of Abf1 binding sites have been identified since its first characterisation and it is now recognised as having a function at hundreds of promoters and other sites throughout the yeast genome (Ganapathi et al., 2011). It has been shown to bind at the well characterised consensus DNA binding motif 5′-RTCRYNNNNNNACG-3′, which is present at numerous sites throughout the yeast genome including promoter elements, mating-type silencers and ARSs (Rhode et al., 1992). Initially this consensus sequence came from investigations of selected binding sites of interest (Buchman et al., 1988; Rhode et al., 1992; Della Seta et al., 1990). Following this, many more individual sites of interest were identified as Abf1 binding sites with techniques such as electrophoretic mobility shift assays (EMSAs). Latterly, global techniques such as transcription microarrays (Yarragudi et al., 2007) and ChIP-chip (Lee et al., 2002; Harbison et al., 2004; Schlecht et al., 2008) have been used to identify Abf1 binding sites throughout the yeast genome in vivo. Lee et al. (2002) and Harbison et al. (2004) analysed the binding sites of 106 and 203 DNA binding proteins respectively, by the use of Myc epitope tagging. These identified 462 and 458 Abf1 binding sites respectively, but the analyses were limited to a subset of intergenic regions only. Schlecht et al. (2008) analysed binding during fermentation, sporulation and respiration, using an antibody for Abf1, identifying 1,689 potential binding sites, of which 1,169 occurred in all three conditions. Additionally, a SELEX approach (Beinoravičiūtė-Kellner et al., 2005) and protein binding microarrays (PBMs) (Mukherjee et al., 2004) have been used to identify Abf1 binding sites in vitro.

These genome wide investigations consistently identified the consensus sequence at Abf1 binding sites, but there were many instances where Abf1 binding was observed without this sequence, and no other consensus sequence could be identified. Suggestions for why this may be include Abf1 having a non-specific DNA binding affinity (Schlecht et al., 2008) and overlapping

binding sequences precluding the identification of a single motif (Ganapathi et al., 2011). The positive element distal (PED) region in the promoter region of the Spt15 gene binds strongly to Abf1 but does not contain the consensus motif (Schroeder and Weil, 1998). Instead it contains the sequence RTARYNNNNNACG, with an adenine replacing the cytosine at the third position. This suggests that Abf1 has the ability to bind to sequences similar to, but not exactly matching, the consensus motif.

Structurally, Abf1 comprises two main components: a C-terminal activation domain and an N-terminal DNA binding domain. The C-terminal domain is not required for DNA binding alone, but its loss confers loss of full functionality (Rhode et al., 1992; Cho et al., 1995; Li et al., 1998). It consists of two regions: C-terminal sequence 1 (CS1) is required for proper nuclear localisation of Abf1 and may be involved in negative transcriptional regulation; CS2 is required for activating DNA replication, chromatin remodelling and transcriptional activation (Miyake et al., 2002; Loch et al., 2004). The N-terminal domain contains a bipartite DNA binding domain consisting of a zinc finger motif (Rhode et al., 1989) and a novel DNA binding domain (Cho et al., 1995). Mutations in both DNA binding domains result in reduced or eliminated DNA binding (Cho et al., 1995). A temperature sensitive binding mutant with a point mutation in the zinc finger binding domain, *abf1-1*, exhibits normal functionality at the permissive temperature but looses its binding ability at the semi-permissive temperature (Rhode et al., 1992). This mutant has been used to investigate various functions of Abf1 including transcriptional regulation (Miyake et al., 2004) and NER (Reed et al., 1999).

1.4.4.1 Role in NER

Abf1 was initially implicated in NER when it was observed to copurify with the Rad7 and Rad16 proteins (Reed et al., 1999). Reducing the cellular level of the Abf1 protein, via the addition of a temperature-dependent degradation signal, resulted in a severe NER deficiency at the restrictive temperature compared to the permissive temperature. Additionally, the temperature sen-

sitive DNA binding mutant *abf1-1* was shown to be deficient at the removal of photoproducts from damaged DNA at the restrictive but not the semipermissive temperature.

This Rad/Rad16/Abf1 complex was subsequently shown to generate superhelical torsion in DNA, via the Rad16 protein, which is required for NER (Yu et al., 2004). This superhelical torsion is required for the excision of the damage-containing fragment of DNA, following the introduction of the two incisions either side of the damage. The binding of Abf1 to DNA is required for efficient NER in a region adjacent to the binding site (Yu et al., 2009). The mutation of an Abf1 binding site in the HMLα promoter caused a region of reduced repair efficiency over a region of ~400 bp in one direction from the Abf1 binding site. Reversing the direction of the non-mutated Abf1 binding site creates the same domain of reduced repair, showing the orientation of the binding of Abf1 to the site significantly affects its function in NER. It has been hypothesised that Abf1 functions to position the Rad7/Rad16 complex to chromatin in the absence of damage, which can then facilitate histone acetylation by allowing access to the HAT (Yu et al., 2011, shown in Figure 1.7).

The Rad7 and Rad16 proteins are required for the UV-induced histone H3 acetylation required for efficient repair, by controlling the occupancy of the histone acetyl transferase Gcn5 on chromatin (Yu et al., 2011). It may be that Abf1 provides a means of positioning the Rad7 and Rad16 proteins throughout the chromatin, which enables the promotion of super helical torsion, histone acetylation and subsequent repair upon the detection of damage. This may be achieved locally to the Abf1 binding site, or by the Rad7/Rad16 complex translocating a longer distance along the DNA. The previously identified directionality of the complex may determine the direction in which the superhelical torsion or acetylation is initiated.

An investigation by Dr. Matthew Leadbitter showed that Abf1 preferentially binds at promoters, with the majority of inter-genic regions showing some level of binding (Leadbitter, 2011). The binding of the Rad16 protein was also investigated by ChIP-chip and this was shown to colocalise with Abf1 at many sites, which associated with UV-induced H3Ac at the sites.

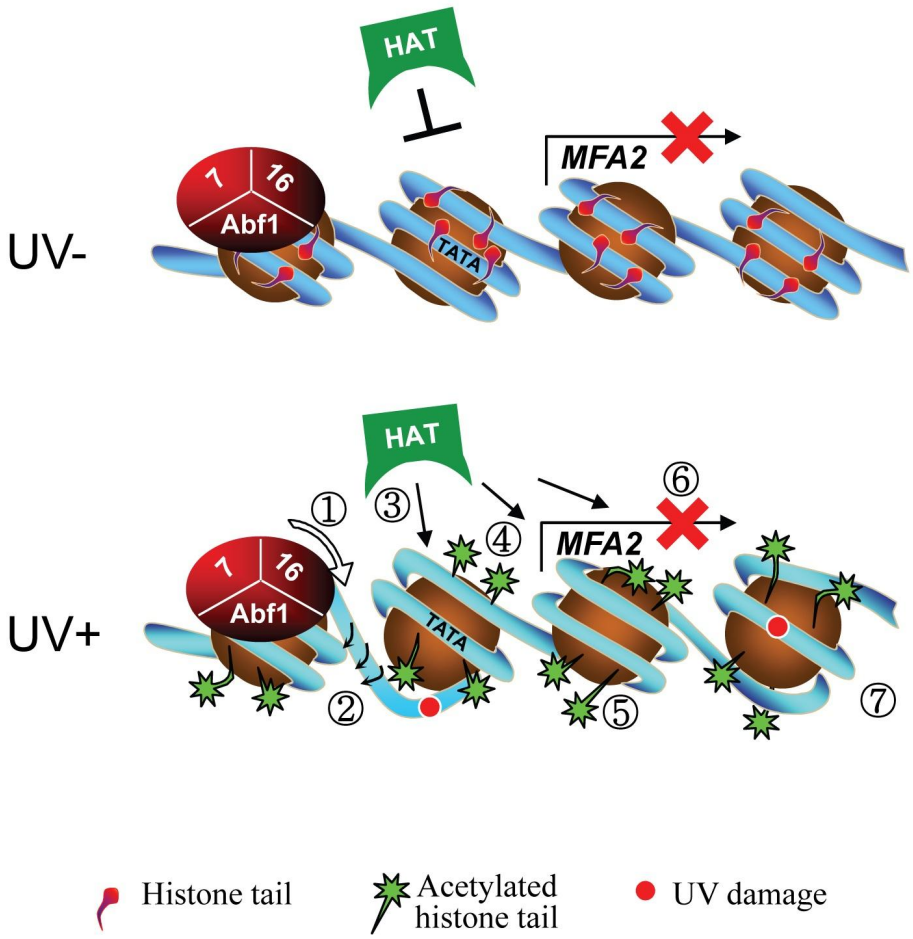


Figure 1.7: Abf1 and GG-NER: Hypothesised model for UV-induced chromatin remodelling, taken from Yu et al. (2011). The Abf1/Rad7/Rad16 complex is present on the DNA in the absence of damage (top panel) but access by the HAT is inhibited. Following UV irradiation (bottom panel) the DNA translocase (1) and E3 ligase (2) activities of Rad16 promote accessibility by the HAT (3) and acetylation (4), leading to a more open chromatin structure (5). Transcription remains inhibited during the process (6) while allowing repair of damage (7).

Several Abf1 binding sites showed reduced levels of binding following UV irradiation and preliminary data from this study suggest that there may be UV induced changes in the DNA binding kinetics of Abf1.

1.5 This study

This study can be divided into two overlapping parts. The first is mainly bioinformatic, in which ChIP-chip data from various investigations have been used to develop bioinformatic tools which can be applied to analyse other ChIP-chip datasets. This includes normalisation and peak detection algorithms. The second is mainly biological, in which Abf1 binding ChIP-chip datasets have been analysed to increase understanding of the protein's binding behaviour. The two overlap in the development of a method to predict damage profiles throughout genomes and the development of bioinformatic tools to display and interogate ChIP-chip data, which have been applied to a selection of real ChIP-chip datasets to produce biologically significant results.

The first three results chapters present a collection of R scripts of tools for the analyses of ChIP-chip data using novel normalisation and enrichment detection algorithms. The aim when developing these tools was to allow the processing of ChIP-chip data from its raw state, through quality assessment, to its loading into R, where normalisation and enrichment detection can be applied and graphical outputs showing features of interest produced. The objective of the novel normalisation procedure is to process data from multiple ChIP-chip datasets so as to allow relative comparisons between them. The objective of the novel enrichment detection procedure is to identify either regions of enrichment or peaks indicative of binding sites across datasets in a way that does not require the application of a multiple testing correction, by dynamically adjusting the threshold of detection based on the probe density of the region being analysed. The tools were developed with the objective that the data should be easily accessible and well described such that separate, possibly more advanced, analyses can be carried out by users if required. Thus all code is shown, annotated and described and a separate condensed, instruction document is provided for users.

Chapter 6 presents a method of predicting the output of a DIP-chip assay measuring UV induced CPDs in the yeast genome, based on the genome sequence and known frequencies of occurrence of the different possible dipyrimidine combinations. The objective of this work was to determine the capability of the DIP-chip assay to detect these CPDs, which was shown to be the case. A comparison of the two datasets was undertaken with the aim of determining whether or not there were any genome regions with higher or lower damage levels than those predicted based on the sequence alone. Here the null hypothesis tested was that there would be no significant differences between the two, which is shown to be the case.

Chapter 7 presents an application of the bioinformatic tools, using them to analyse Abf1 protein binding. The objective of this work was both to show the practical applications of the bioinformatic tools with real ChIP-chip datasets, analyse the binding sites in their own right and compare the binding sites identified with those of previously published genome wide investigations. Sequences at the binding sites were analysed which showed that many Abf1 binding sites do not contain the previously identified consensus binding motif and no further consensus sequences could be identified. A much larger number of binding sites were identified in this investigation than had been previously, the locations of which were analysed under the null hypothesis of no significant overlaps with previously identified genome wide binding locations, which is shown not to be the case.

Chapter 2

Technical Overview

ChIP-chip, and modified versions thereof, is the main technology used to produce the results analysed in this thesis. This chapter describes the 'wet' laboratory processes leading up to the generation of the data, the 'dry' laboratory processing of which is described in later chapters. Only the dry laboratory analyses of the data have been carried out in this thesis. The following sections describe the techniques, employed by colleagues in the laboratory, used to generate the data for these analyses. Much of this work has been carried out by Evans (2011) and Leadbitter (2011). Some of the information presented in this chapter is adapted from, and further information can be found in, these works.

2.1 Microarrays

The microarray is the component of the ChIP-chip technology that allows the generation of genome-wide data (described in Section 1.1). All microarray data produced in our laboratory analysed in this investigation are from microarrays manufactured by Agilent Technologies Inc. These microarrays contain probes of average length 60 nt, 'printed' onto the slide surface, one base at a time, to build up each sequence, (Agilent Technologies Inc., 2003). The yeast slides used (product number G4493A) contain 4 microarrays each (Figure 2.1), each of which contains 45,219 features. Of these, 41,775 contain probes against the yeast genome. The remainder are various technical

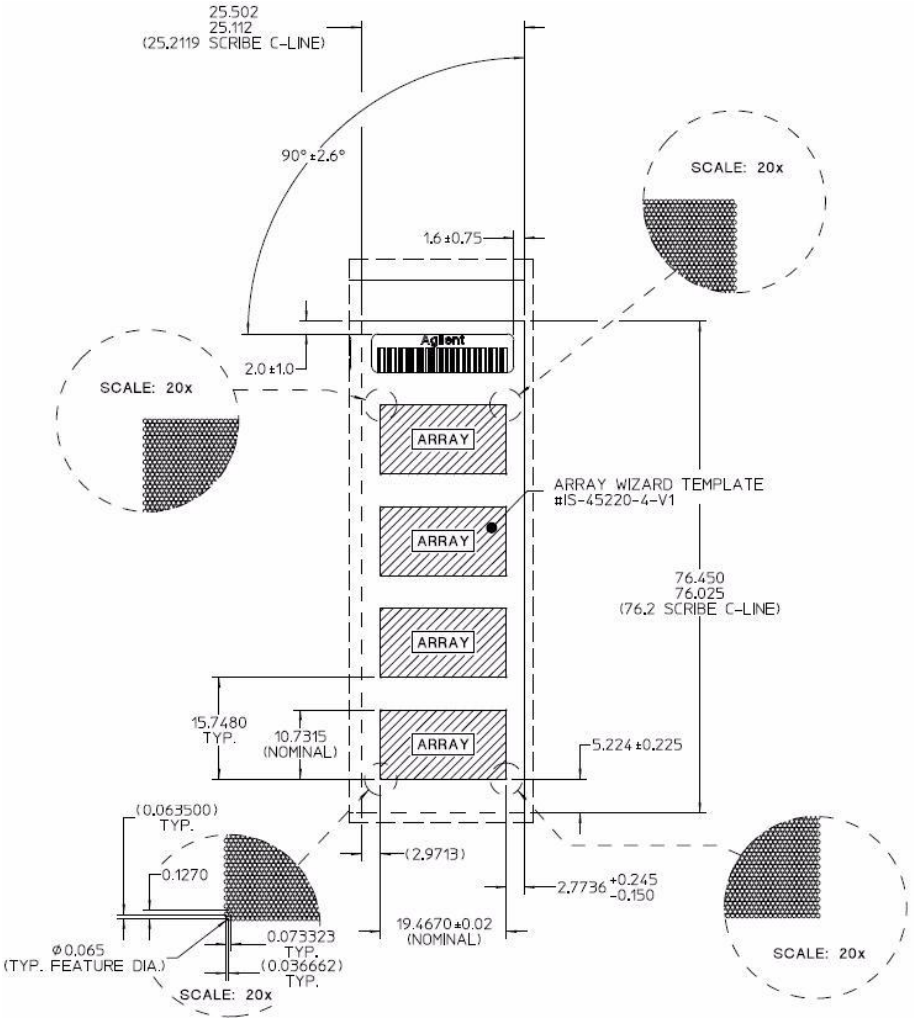


Figure 2.1: Agilent 4 x 44k microarray format: Layout and dimensions of Agilent 4 x 44k microarrays (Agilent Technologies Inc., 2007), of which the G4493A is used here.

probes, including positive and negative controls and special patterns to allow correct alignment to the grid by the Feature Extraction software, described in Section 2.6. This means four sets of results are generated from each slide of this type.

2.2 Chromatin immunoprecipitation

The first stage of the procedure, ChIP, is a useful tool in its own right for identifying sites in chromatin, genome wide, to which proteins of interest are bound. The following description is adapted from that in Alberts et al. (2002) with specific details taken from Evans (2011). The standard DNA purification procedures include a protein digestion stage and so are not suitable for immunoprecipitation of proteins in chromatin, as this would mean no proteins would be available for antibodies to bind. Therefore a chromatin purification procedure is employed. Proteins are covalently cross-linked to DNA in living cells by treatment with formaldehyde. Following this, glycine is added to stop the crosslinking. Cells are then collected and washed over three centrifugation steps before being lysed using glass beads and vortexing. The cell lysate is separated from the glass beads and any non-crosslinked soluable proteins with a further three centrifugation/wash steps. The remaining pellet of chromatin is resuspended and subject to sonication with a Bioruptor for 6 cycles of 20 sec on/40 sec off. This produces high frequency sound waves which shear the chromatin into fragments of average length 600 nt. After two further centrifugation/wash steps the fragmented chromatin supernatant is snap frozen with liquid nitrogen and stored at -80°C.

Antibodies raised against a protein of interest can be used to separate fragments of DNA bound to that protein from the purified chromatin. Alternatively, proteins may be separated by means of an exogenous epitope tag, to which antibodies are available, fused to the protein. These antibodies bind to the epitope on the protein, facilitating the immunoprecipitation process. This also captures the DNA fragment to which the protein is covalently bound. An antibody titration experiment is first carried out to determine the optimal amount of antibody to use. The antibodies are incubated with,

and allowed to bind to, magnetic Dynabeads. These are then collected with a magnet, washed, resuspended and added to the sonicated chromatin. The bead-DNA-protein mix is then separated from the rest of the genomic DNA by the use of a magnet, and washed several times. Collected DNA is eluted off of the beads.

An input sample is also prepared, which does not undergo this immunoprecipitation procedure and therefore contains all fragmented genomic DNA. Two DNA samples are applied to the Agilent microarrays, comprising the immunoprecipitated material, henceforth referred to as the IP sample, and purified total genomic DNA taken before the immunoprecipitation process, referred to as the input sample. All samples are incubated with RNase and purified with a Quiagen PCR purification kit.

Used as a stand-alone technique, at this point DNA primers of a region of interest can be used to determine whether or not the protein of interest is bound at that region, by PCR amplification. If the region is bound to the protein, the immunoprecipitated material will contain DNA fragments to which the primers will anneal and it will therefore be amplified by PCR, the product of which can be detected by a variety of methods. If the protein is not bound, no amplification will take place. Quantitative-PCR (Q-PCR) can be employed to determine quantitative protein binding levels at the selected genomic region.

2.3 Amplification

Combining ChIP with microarrays in ChIP-chip removes the need to carry out individual assays for each region of interest, as multiple locations are represented on the microarray. These locations may cover a whole genome or represent specific genomic regions of interest. The detection of DNA is achieved through the measurement of fluorescence from fluorescently labelled hybridised DNA. The amount of DNA generated by ChIP is not sufficient to apply directly to a microarray and so it is amplified by ligation mediated (LM) PCR. T4 DNA polymerase (an endonuclease) is used to blunt end the DNA fragments which then undergo a phenol/chloroform extraction, cold

ethanol precipitation, centrifugation and resuspension in purified water. This is then incubated with a ligation mixture, which ligates a section of linker DNA to the blunt-ended fragments. These linkers are used to amplify the DNA by PCR, using a single set of primers to the linker sequences. Two sets of PCR reactions are carried out to ensure a sufficient quantity of DNA.

2.4 Fluorescent labelling and hybridisation

The Agilent microarrays used to generate the data in this investigation are two colour. This means two DNA samples are represented on each microarray, here the IP and input samples, each distinguished by labelling with a different coloured fluorescent dye. This is in contrast to other platforms, such as Affymetrix, which are one colour systems where each microarray represents a single sample. The input sample acts as an internal control on the microarray, removing the effects of differential hybridisation, due to factors such as sequence variations, at different points in the genome. This is achieved by calculating the $\log_2$ IP:input sample signal ratio as the final data. This enables genuine regions of binding to be identified (where the IP sample is present at a larger amount than the input sample) over regions where other factors cause high levels of binding to the microarray (where the IP and input samples are present at the same, albeit large, amounts). This is demonstrated in real ChIP-chip data with Abf1 binding IP and input signals over a section of chromosome 1 in Figure 2.2.

The two DNA samples are differentially labelled with the dyes Cy3 and Cy5. Cy3 produces a green fluorescence (555/565 nm excitation/emission respectively) and Cy5 a red fluorescence (650/670 nm excitation/emission respectively). Dye bias arises when one fluor produces a higher signal intensity than the other. Several microarrays produced in our lab have been analysed for dye bias, including the Abf1 binding data presented in Chapter 7 (Leadbitter, 2011), where it was shown that this did not have an effect on the data, and so there was no need to apply any correction for this effect on the datasets analysed here. The labelling reaction is achieved via an amplification step which incorporates nucleotides labelled with the dye, along

with unlabelled nucleotides, into a newly synthesised DNA fragment. Two pools of labelled DNA are thus created, which are applied to the microarray together and allowed to hybridise. A cover slip containing rubber gaskets prevents the samples leaking and keeps those applied to each microarray separate. This process takes place overnight in a rotating oven at 65°C, which allows the pool of labelled DNA fragments to cover the whole microarray and bind to any corresponding probes. The slides are then washed to remove any unbound material. The concentration of DNA applied to the microarrays is the same level for all experiments, regardless of the initial IP and input sample concentrations, so as to ensure optimal performance of the microarray in the following stages.

2.5 Microarray processing

Following hybridisation, the microarrays are scanned using an Agilent microarray scanner (model G2505B) to produce a TIFF image, from which the signal intensities can be extracted. The scanner uses two lasers at 532 and 633 nm. These scan across the slide surface, exciting the fluors of the labelled DNA as they do so. Fluorescence is detected and an image produced at a resolution of 5 microns. The brighter the fluorescence of a feature the more DNA is bound to it. There is not however a linear relationship between fluorescence and DNA amount (Schena, 2003) so when analysing results one can only infer relative, rather than absolute, DNA amounts.

2.6 Feature extraction

The TIFF image produced by the microarray scanner is loaded into Agilent's Feature Extraction software (Agilent Technologies Inc., 2010b, current version 10.10.1.1). This aligns a grid to the image, determining the positions of all features, and analyses the red and green colouration of each. This converts the fluorescence intensities into numerical values. The software contains information about the genomic region each feature represents which is linked

together with the intensity values. This is written to a tab delimited text file containing over 40 columns of data, including information such as scanner settings and a range of diagnostics. The layout of this file is represented in Figure 2.3.

All analyses presented in this thesis use background subtracted data. This is data from which the intensity values surrounding the features are subtracted from the intensity values of the features themselves. This ensures that the intensity values analysed are due to the specific hybridisation of labelled DNA to the probes and not the general fluorescence of the slide.

2.7 Data analysis

All data were loaded into `R` (current version 2.14.2) for processing and analysis. A PC with a 3.20 GHz Intel i7 processor and 24 GB of RAM, running 64 bit Microsoft Windows 7, was used for the extraction and analysis of data and the creation and testing of all `R` scripts. These `R` scripts were then used in our laboratory on several different computers running various versions of Microsoft Windows and Mac OS X.

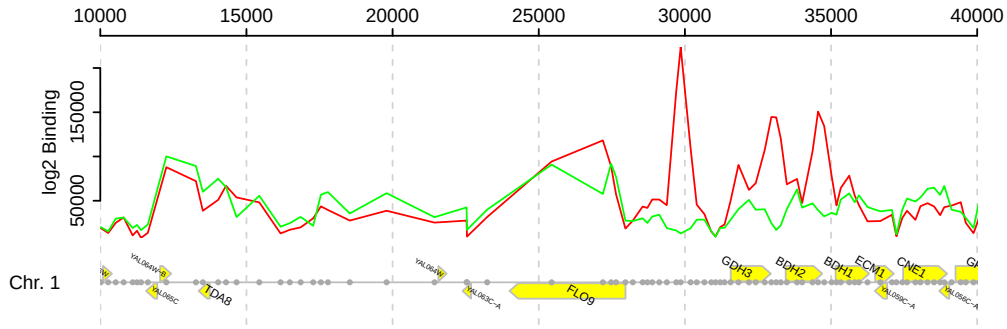


Figure 2.2: The importance of IP and input samples: A short section of chromosome 1 showing Abf1 binding IP (red) and input (green) sample data. There are peaks in the IP data at ~12,500 and ~25,000, which are matched by the input data, in contrast to the peaks at ~30,000, ~33,000, and ~35,000. This shows that only the last three are likely to be representative of binding. Without the input sample the first two may also be incorrectly considered to be binding sites.

Line	Contents	
1	Data type	Parameters → 152 columns
2	Data name	
3	Data values	
4	[Blank line]	
5	Data type	Statistics → 147 columns
6	Data name	
7	Data values	
8	[Blank line]	
9	Data type	Features → 112 columns ↓ *p* rows
10	Data name	
11	Data Values	

Figure 2.3: Feature extraction file format: The contents and layout of the tab delimited text file produced by Agilent's Feature Extraction software are represented. Parameters and statistics run for three lines each. Features run for p rows, where p is the number of probes on the microarray. For each column the data type, its name and a value are provided.

Chapter 3

Creation of a collection of R scripts to process and interrogate ChIP-chip data

3.1 Introduction

The primary objective of the work presented in this thesis was to analyse ChIP-chip datasets produced in our laboratory. To achieve this, the major issues in current ChIP-chip data analysis methodologies have been investigated and new tools developed to fill some of the existing gaps. Two of the main aspects of this work, the development of novel normalisation and peak detection methods, are presented in the following two chapters. This chapter details how these and several other aspects have been brought together into a collection of R scripts for use by researchers wishing to analyse their own data, both in our laboratory and beyond.

To date the only R package dedicated to the analysis of ChIP-chip data is RINGO (Toedling et al., 2007). This package contains several facilities for analysing ChIP-chip data, including data import, quality assessment, normalisation, visualisation and enrichment detection, developed for the analysis of Nimblegen data. It uses aspects of other Bioconductor packages for some of these functions, such as loading data in the limma package's RGList format (Smyth, 2005). The normalisation consists of applying a choice of existing normalisation procedures to the data, from the packages VSN (Huber

et al., 2002) and `limma` and the Tukey-biweight scaling. The package therefore is only suitable for analysing technical replicates of a single experimental condition, for the reasons outlined in Section 1.1.3.2 and Chapter 4.

ChIPMonk, a Java application (Andrews, 2007), contains a similar set of tools to RINGO, and at the time of publication was limited to only being able to analyse Nimblegen data. The methods are basic and as with RINGO do not facilitate the simultaneous analysis of different types of data.

Tilescope (Zhang et al., 2007) also provides similar tools, such as normalisation and peak detection, as web server, with data uploaded as tab-delimited text files. The publication does not list Agilent data as a format that can be uploaded and at the time of writing the website was unavailable for testing.

A short report by Toedling and Huber (2008) outlines more methods for the analysis of ChIP-chip data in `R`, none of which allow the comparative analysis of datasets from different experimental conditions. Here several different methods and packages are used to load data, assess its quality, load genome annotations, process, plot and analyse the data, showing that no single package is capable of performing all of these tasks. Although this may not pose a problem for someone adept at data analysis and familiar with the `R` interface, the aim of the functions created here was to enable researchers with little or no bioinformatic experience to quickly use the tools described in the coming chapters to extract meaningful results from their data, while maintaining scope for more detailed analyses should these be required. The objective of the work presented in this chapter was therefore to produce a collection of annotated, documented, integrated tools, using the `R` programming language, able to process ChIP-chip data from its raw state through to the generation of meaningful biological results, in such a way that basic analyses can be conducted easily and efficiently, while the scope for more advanced analyses is maintained.

3.2 The scripts

All of the analyses presented in this thesis have been carried out in `R` (R Development Core Team, 2011) using Bioconductor (Gentleman et al., 2004).

A collection of functions and classes was created to perform these analyses. This section describes these different functions and shows the corresponding scripts. Scripts are described with reference to specific line numbers denoted by an 'L' in brackets. All scripts are also available in the "R scripts" folder of the electronic appendix (see Page 367). A condensed version of this chapter, in the form of a user guide, has been produced ("instructions.pdf" file in the electronic appendix; see Page 367). This is available alongside the full set of scripts and is intended to provide users with instructions for running the functions and interpreting the outputs, along with overviews of the more complex functions.

Figure 3.1 shows a diagram of all of the different components created here and how they relate to each other. All objects are shown, indicating the functions used to create them and the functions which act upon them. All functions are shown, indicating which objects they act upon or in which other functions they are used. External Bioconductor packages are also shown, indicating the function in which they are used.

Functions and objects, and their associated descriptions, have been split into the following eight broad categories, which form the basis of this and subsequent chapters.

- How data is loaded into and organised within `R` (Section 3.2.1).
- Data quality assessment (Section 3.2.2).
- Accessing data-containing objects (Section 3.2.3).
- Manipulated data-containing objects (Section 3.2.4).
- Textual display of data and results (Section 3.2.5).
- Graphical display of data and results (Section 3.2.6).
- Data normalisation (Chapter 4).
- Enrichment detection (Chapter 5).

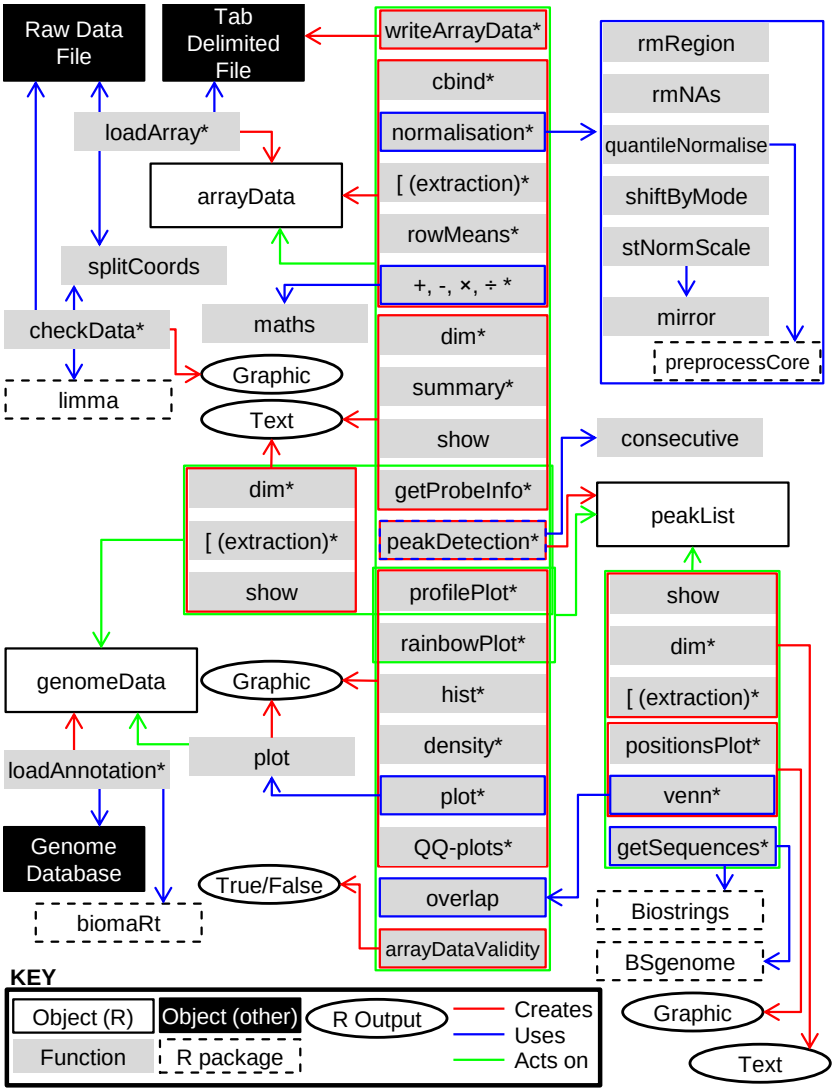


Figure 3.1: Overview of the functions presented: Diagram showing the various functions and objects and how they relate to each other. R objects and functions are those newly written or created for this work. Other objects are files or databases outside of R. R packages are those already available on Bioconductor. Coloured arrows indicate functions that create or act on objects and use other functions. For clarity coloured boxes highlight some functions an arrow relates to. The direction of the arrows shows the relationship between two items. For example, the loadArray function "uses" splitCoords, "acts on" raw data and tab delimited files and "creates" arrayData objects. Asterisks denote functions intended to be called by users while the rest are called within other functions.

3.2.1 Loading data

The production and format of raw microarray data are described in Chapter 2. This section details the methods used to load this data into `R` and the format this takes. It also describes the loading of other other data relevant to the analyses.

3.2.1.1 Utilising limma

The `limma` package was developed to analyse transcription array datasets (Smyth, 2005). This has a function, `read.maimages`, to read microarray data files into `R` in the form of an `RGList` object. These contain the following components.

`R` Red signal intensity values.

`G` Green signal intensity values.

`Row` Probe microarray row number.

`Col` Probe microarray column number.

`ProbeUID` Unique probe ID.

`ProbeName` Probe name.

`GeneName` Corresponding gene name.

`SystematicName` Genome position details.

`Description` Probe position relative to gene.

Originally, the `limma` package was used here to load data into `R` as `RGLists`, and the functions were written to act on the data in this format. Because `limma` is intended for transcription microarray data analysis its `RGList` does not include readily accessible data on probes' genomic positions. This data therefore had to be extracted from the "Systematic Name" column of

the Agilent Feature Extraction file, split into its component parts (chromosome number and start and end coordinates) and added to the `RGList` as new columns.

These modified `RGList`s were used as the basis of early analyses and all new functions were written to extract relevant data from them. This initially proved satisfactory but became limiting when more advanced scripts were being written. The structure of the `RGList`s — dataframes contained within a list — meant that some analyses, especially those requiring repeated looping, were slow. It is generally the case with the analysis of transcription microarrays that the data is processed relatively quickly, in a small number of steps, to produce a final set of results consisting of information on the genes determined to be differentially expressed. There is then generally no further use for the original microarray data and so it can be disregarded. Depending on the type of data generated and the analyses being undertaken, this is not necessarily the case with ChIP-chip data, as all features can be equally informative in a number of different applications. It is therefore useful to have all data in a format that can be quickly and easily processed in multiple ways, which the `RGList` format is not best suited to. To overcome this a new data format was created, containing similar information to the `RGList` but in a format better suited to ChIP-chip data and the analyses to be performed on it. This new class was named `arrayData` and replaced the `RGList` for ChIP-chip data storage in `R`. All existing functions were adapted, and all new functions written, to work with this new format. The following sections detail this `arrayData` format and its creation upon loading data into `R`.

3.2.1.2 The `arrayData` class

The minimum information required to perform analyses on ChIP-chip data is each probe's fluorescence intensity values and the genomic position it represents. This information allows the fluorescence values to be linked to the relevant sections of the genome. A unique reference is also an important component, which allows each probe to be identified individually. The `arrayData` class was written to contain all of this data, along with any other data speci-

fied by the user and information relevant to the processing of the data. This section describes the format of an `arrayData` object; how data is loaded into this object is detailed in the next section. Each `arrayData` object takes the format of a list containing the following five components.

`coordinates` A three-column matrix containing the chromosome number, start position and end position of each probe as a separate row.

`annotations` A matrix consisting of at least one column, but can have any number containing user specified information. The first column must contain a unique identifier for each probe.

`ratios` A matrix containing one column of $\log_2$ red:green ratios for each dataset. Column names show the file name from which the data were taken and are edited by some functions to show processes that they have undergone.

`grid_name` A character vector containing only the name of the microarray grid the data have come from. This is used to ensure data of different types are not accidentally mixed.

`status` A list containing information on the normalisation procedures carried out on each dataset. This starts out as "raw" when data is initially loaded and is updated by the various normalisation functions to show the processing that has taken place. It is checked by other functions to ensure that datasets have been processed in the ways that they require.

3.2.1.3 Creating new `arrayData` objects

The `loadArray` function (Script 3.1) is the most crucial, as it is the only one able to take data from an external source, load it into `R` and convert it to the required format for an `arrayData` object. It would be possible to do this manually, but via a number of stages and so would take longer and require more work. The function can load data from two file types: the text file created by the Agilent Feature Extraction software, which is of a specific format, or a tab-delimited file, which can be created containing data from

any source. The feature extraction file is of a fixed format (Section 2.6 and Figure 2.3) and the majority of the data are not needed for analyses. Therefore, in order to allow efficient processing of the data, the function scans the file for the required regions and only these are loaded into `R`. This data is represented in four 'Features' columns of the file: one contains probe chromosome numbers, start coordinates and end coordinates as a single string, one each contain the red and green intensity values and one contains unique IDs in the form of probe names.

Tab-delimited files must contain all the same essential data in a certain format for the function to create a correct `arrayData` object from it. The layout is outlined in Figure 3.2. The first line ("Data types") details the type of data in the column and may be one of "coords", "anno" or "ratios". The second line ("Data names") contains the column names. Coordinates must contain "probeChr", 'probeStart" and "probeEnd". Annotations must contain "probeID" and any other optional columns. Ratios must contain data names. All values follow for p rows, where p is the number of probes. A grid name and statuses may also be provided. In this case the first line ("Extra data names") contains "grid_name" and the data names. The second ("Extra data values") contains the grid name and each status. The rest of the file then starts on line three. The function looks for the text "grid_name" in the first position; if this is not present it assumes the probe data starts on the first line. The order of the status file names are compared to the order of the data file names and the correct status assigned to each set of data. The function has the following arguments:

fileName Names of .txt or .tab files to load (no default).

essentialColumns A list of the names of the columns in the Feature Extraction text file containing essential information. Red and green intensity values are taken from the "rBGSubSignal" and "gBGSubSignal" columns respectively, coordinates from "SystematicName", probe names from "ProbeName" and control types from "ControlType" by default and do not need to be modified under normal circumstances.

otherColumns A list of other columns specified by the user (no default).

Line	Contents
1	Extra data names
2	Extra data values
3 (1)	Data types
4 (2)	Data names
5 (3)	Data values ↓ *p* rows

Figure 3.2: Tab-delimited file format: Representation of the contents of the tab-delimited text file for loading into `R` by the `loadArray` function. The first two lines (grey) are optional and specify the grid name and normalisation statuses. Lines 3–5 (1–3 if the optional data is not included) show data types (coordinates/annotations/ratios), data names (chrNum/chrStart/...etc) and the corresponding data values. These run for a minimum of five columns but can contain multiple annotation and ratio columns. Values run for p rows, where p is the number of probes.

processCoords The name of the function required to split the coordinates into their component parts (default "splitCoords"; Script 3.2).

spikes Specifies whether or not spike in probes are present on the microarray, the processing of which is different to genomic probes (default FALSE).

Script 3.1: loadArray: loads data from Agilent Feature Extraction files or tab delimited files with correct column headings into a new arrayData object

```
1  ## loadArray function ##
2  ## arguments: fileName (file(s) to load), essentialColumns (list of
       essential columns to load from FE file), otherColumns (list of other
       columns to load from FE file), processCoords (function to process
       systematicName data), spikes (whether the dataset contains spikes)
3  loadArray<-function(fileName, essentialColumns=list(red="rBGSubSignal",
       green="gBGSubSignal", coords="SystematicName", probe="ProbeName"),
       otherColumns=list(), processCoords=splitCoords,spikes=FALSE) { #define
       function
4    if (length(essentialColumns$red) == 0 | length(essentialColumns$green) ==
         0 | length(essentialColumns$coords) == 0 | length(essentialColumns$
         probe) == 0) stop("Essential columns missing", call.=F) #check all
         essential columns (red, green, coords, probe) are defined
5    if (missing(fileName)) { #if file names are not provided
6      fileName.txt<-list.files(pattern=".txt$") #get all .txt files in working
           directory
7      fileName.tab<-list.files(pattern=".tab$") #get all .tab files in working
           directory
8    }else{ #file names are provided
9      ext<-character(length = length(fileName)) #initialise vector to store
           file types
10     fileName.txt<-fileName.tab<-character() #initialise vectors to store
           file names
11     for (n in 1:length(fileName)) { #loop through file names
12       if (substring(fileName[n],nchar(fileName[n])-3,nchar(fileName[n])) ==
             ".txt")  ext[n]<-"A" #search for .txt files
13       if (substring(fileName[n],nchar(fileName[n])-3,nchar(fileName[n])) ==
             ".tab")  ext[n]<-"D" #search for .tab files
14     } #exit loop
15     fileName.txt<-fileName[ext == "A"] #get all .txt file names provided
16     fileName.tab<-fileName[ext == "D"] #get all .tab file names provided
17   }
18   if (length(fileName.txt) > 0 & length(fileName.tab) > 0) stop("Mixtures of
         file types cannot be loaded together", call.=F) #stop with message if
         files of both types are present
19   if (length(fileName.txt)== 0 & length(fileName.tab) == 0) stop("No files
       to load", call.=F) #stop with message if no files are present
20   if (length(fileName.txt) > 0) { #if .txt files are provided/present
21     fileName<-fileName.txt #store them
22     txt<-TRUE #set txt to TRUE
23   }else{ #.tab files are provided/present
24     fileName<-fileName.tab #store them
25     txt<-FALSE #set txt to FALSE ie tab is true
26   }
27   for (n in 1:length(fileName)) { #loop through files
28     if (txt) { #a .txt file is to be loaded
29       message (paste("Loading:",fileName[n])) #print the name of the file
             being loaded
```

```
30        columnNames<-scan(fileName[n],skip=1,nlines=1,what="",quiet=T) #get
              column names from row 2 of the file
31        totalColumns<-length(columnNames) #get total numer of columns of file
32        columnRead<-rep("NULL",totalColumns) #set all columns to NULL (so they
               are not read)
33        columnRead[columnNames == "Grid_Name"]<-NA #set grid name column to NA
               (in order to be read)
34        grid_name<-as.character(as.matrix(read.table(fileName[n],colClasses=
              columnRead,skip=2,fill=T,sep="\t",nrows=1))) #load the grid name
35        grid_name<-strsplit(grid_name,"_")[[1]][1] #split the grid name and
              store the first part
36        if (length(grid_name) == 0) warning("No grid name found", call.=F) #
              warn if no grid name is found
37        columnNames<-scan(fileName[n],skip=9,nlines=1,what="",quiet=T) #get
              column names from row 10 of the file
38        if (length(grep(essentialColumns$red,columnNames)) != 1 | length(grep(
              essentialColumns$green,columnNames)) != 1 | length(grep(
              essentialColumns$coords,columnNames)) != 1 | length(grep(
              essentialColumns$probe,columnNames)) != 1) stop("Essential columns
               not present", call.=F) #stop with message if not all essential
              columns present in file
39        totalColumns<-length(columnNames) #get total numer of columns of data
40        columnRead<-rep("NULL",totalColumns) #set all columns to NULL (so they
               are not read)
41        columnsF<-which(columnNames %in% c(essentialColumns,otherColumns)) #
              get columns specified
42        columnRead[columnsF]<-NA #set listed columns with NA (to be read)
43        arrayFile<-as.matrix(read.table(fileName[n],colClasses=columnRead,skip
              =9,header=T,fill=T,sep="\t",quote="")) #read data into R
44        if (spikes) {
45          spikeData<-arrayFile[grep(">",arrayFile[,which(colnames(arrayFile)
                == "SystematicName")]),]
46        }
47        arrayFile<-arrayFile[grep("chr",arrayFile[,which(colnames(arrayFile)
              == "SystematicName")]),] #get data representing non-control probes
48        coords<-processCoords(arrayFile[,which(colnames(arrayFile) ==
              essentialColumns$coords)]) #split coordinates with defined
              function
49        chromosomes<-sort(unique(coords[,1])) #get ordered, unique chromosome
              numbers
50        startWarn<-as.numeric(options("warn")); on.exit(options(warn=startWarn
              )) #get current warning state and maintain on exit
51        options(warn=-1);chromosomes.numeric<-as.numeric(chromosomes);options(
              warn=startWarn) #make chromosomes numeric (without warnings for
              non-numeric values)
52        chromosomesNonNumeric<-which(is.na(chromosomes.numeric)) #identify non
              -numeric chromosomes
53        if (length(chromosomesNonNumeric) > 0) { #if there are non-numeric
              chromosomes
54          for (cnn in 1:length(chromosomesNonNumeric)) { #loop through non-
                numeric chromosomes
55            coords[coords[,1] == chromosomes[chromosomesNonNumeric[cnn]],1]<-(
                  max(chromosomes.numeric,na.rm=T)+cnn) #assign non-numeric
                  entries with the next available numeric value
56          }
57        }
58        coords<-matrix(as.numeric(coords),ncol=3) #convert coordinates to
              matrix
59        options(warn=-1);ratios<-matrix(log2(as.numeric(arrayFile[,colnames(
              arrayFile) == essentialColumns$red])/as.numeric(arrayFile[,
              colnames(arrayFile) == essentialColumns$green])),ncol=1);options(
              warn=startWarn) #calculate log2 ratios from red and green values (
```

```
        without warnings for negative values)
60      annotations<-matrix(arrayFile[,colnames(arrayFile) == essentialColumns
            $probe]) #get essential annotations
61      otherAnnotations<-matrix(arrayFile[,colnames(arrayFile) ==
            otherColumns]) #get other (user define) annotations
62      if (length(otherAnnotations) > 0) annotations<-cbind(annotations,
            otherAnnotations) #join annotations if more than one set are
            loaded
63      annoNames<-c("probeID",otherColumns) #set column names for annotation
            matrix
64      status<-as.list("raw") #set status of data to "raw"
65      ratioNames<-fileName[n] #set column names for ratio matrix
66    }else{  #tab delimited file is to be loaded
67      message (paste("Loading:",fileName[n])) #print the name of the file
            being loaded
68      columnNames<-scan(fileName[n],skip=0,nlines=1,what="",quiet=T) #get
            column names from row 1
69      if (columnNames[1] == "grid_name") { #extra data is provided
70        values<-scan(fileName[n],skip=1,nlines=1,what="",quiet=T) #get
              values from row 2
71        grid_name<-values[1] #get grid name (first position)
72        statusFileNames<-as.matrix(columnNames[2:length(columnNames)][!is.na
              (columnNames[2:length(columnNames)])]) #get file names for
              statuses
73        statuses<-as.matrix(values[2:length(values)][!is.na(values[2:length(
              values)])]) #get statuses
74        columnTypes<-scan(fileName[n],skip=2,nlines=1,what="",quiet=T) #get
              column types from row 3
75        columnNames<-scan(fileName[n],skip=3,nlines=1,what="",quiet=T) #get
              column names from row 4
76        arrayFile<-as.matrix(read.table(fileName[n],header=F,sep="\t",quote=
              "",skip=4)) #load remainder of tab delimited file
77        dataFileNames<-as.matrix(columnNames[columnTypes=="ratios"]) #get
              data file names
78        status<-as.list(statuses[apply(dataFileNames,1,function(x){which(
              statusFileNames == x)})]) #reorder statuses
79      }else{ #no extra data is provided
80        columnTypes<-scan(fileName[n],skip=0,nlines=1,what="",quiet=T) #get
              all row 1 column names
81        columnNames<-scan(fileName[n],skip=1,nlines=1,what="",quiet=T) #get
              all row 1 column names
82        arrayFile<-as.matrix(read.table(fileName,header=F,sep="\t",quote="",
              skip=2)) #load tab delimited file
83        grid_name<-"unspecified" #set grid name to "unspecified"
84        status<-as.list(rep("raw",length(which(columnTypes=="ratios")))) #
              set status to "raw"
85      }
86      coordTypes<-which(columnTypes=="coords") #get columns containing
            coordinates
87      coordColumns<-c(coordTypes[which(columnNames[coordTypes] == "probeChr"
            )],coordTypes[which(columnNames[coordTypes] == "probeStart")],
            coordTypes[which(columnNames[coordTypes] == "probeEnd")]) #get
            coordinate columns with specified column names
88      if (length(coordColumns) != 3) stop("Incorrect column names present",
            call.=F) #stop with message if the wrong number of coordinate
            columns are provided
89      annoTypes<-which(columnTypes=="anno") #get columns containing
            annotations
90      annoColumns<-c(annoTypes[which(columnNames[annoTypes] == "probeID")],
            annoTypes[!columnNames[annoTypes] == "probeID"]) #get annotation
            columns with specified and unspecified (user defined) names
91      if (length(annoColumns) < 1) stop("Incorrect column names present",
```

```
          call.=F) #stop with message if no annotation columns are provided
92        ratioColumns<-which(columnTypes=="ratios") #get columns containing
              ratios
93        coords<-matrix(as.numeric(arrayFile[,coordColumns]),ncol=3)#get
              coordinates as numeric
94        ratios<-matrix(as.numeric(arrayFile[,ratioColumns]),ncol=length(
              ratioColumns)) #get ratios as numeric
95        annotations<-matrix(nrow=nrow(coords),arrayFile[,annoColumns]) #get
              annotations
96        annoNames<-c("probeID",columnNames[annoTypes][!columnNames[annoTypes]
              == "probeID"]) #get column names for annotation matrix
97        ratioNames<-columnNames[columnTypes=="ratios"] #get ratio names for
              ratio matrix
98      }
99      if (length(which(duplicated(annotations[,1]))) > 0) stop("Non-unique IDs
            present", call.=F) #check all IDs are unique; stop message if not
100     arrayFile<-new("arrayData",list(coordinates=coords,annotations=
            annotations,ratios=ratios,grid_name=grid_name,status=status)) #
            create new arrayData object with loaded data
101     arrayFile<-arrayFile[order(arrayFile$coordinates[,1],arrayFile$
            coordinates[,2]),] #order data by coordinates
102     colnames(arrayFile$coordinates)<-c("probeChr","probeStart","probeEnd") #
            set coordinates matrix column names
103     colnames(arrayFile$annotations)<-annoNames #set annotations matrix
            column names
104     colnames(arrayFile$ratios)<-ratioNames #set ratios matrix column names
105     if (n > 1) { #if loading beyond the first file
106       allArrayFiles<-cbind(allArrayFiles,arrayFile) #cbind arrayData objects
107     }else{ #loading the first file
108       allArrayFiles<-arrayFile #store arrayData object
109     }
110   }
111   return(allArrayFiles) #return data
112 }
```

The function first checks that a column name is provided for each of the items in the "essentialColumns" list and is stopped with the warning message "Essential columns missing" if not (L4).

The file names to load are determined from the "fileName" argument or `R` working directory (L5–26). If file names are not provided the working directory is scanned for all files with '.tab' and '.txt' extensions (L6–7). All file names, either user specified or from the working directory, are examined for .txt and .tab extensions (L9–16). If both are present the function stops with the error message "Mixtures of file types cannot be loaded together" (L18). This prevents files of different formats being loaded together. If neither are present the function stops with the error message "No files to load" (L19). If only .txt files are provided the "txt" object is set to TRUE, or if only .tab files are provided it is set to FALSE (L20–26). This is used to determine the processing of the data as it is loaded.

A loop is initiated to load each file individually (L27). For '.txt' files the processing is based on the Agilent Feature Extraction file format (L28–67). The name of each file is printed as it is loaded (L29). The second line of the file, containing the parameter names, is scanned and the column containing the text "Grid_Name" specified to load (L30–33). The corresponding position in the third line, containing the grid name, is loaded (L34). This contains extra text separated by a '_', which is removed (L35). If no grid name is found the warning message "No grid name found" is displayed (L36). The tenth line of the file containing the feature names is red (L37). If the column names from the "essentialColumns" argument are not all present the function stops with the warning "Essential columns not present" (L38), otherwise the corresponding features values are set to load from the eleventh line (L39–43). Spike probe data are separated (L44–46) and control probe values removed (L47). The function specified in the "processCoords" argument is used to split the coordinates into their component parts (L48). Non-numeric chromosome numbers are converted to the next available number and the coordinates stored in a new matrix (L48–58). Ratio values are calculated as the $\log_2$ ratio of red:green values and stored in a new matrix (L59). Warning messages are disabled while there is the potential for non-numeric values to be treated as numeric, during the processing of chromosome numbers and ratios. Annotations — essential and user specified — are stored in a new matrix and the column names saved (L60–63). The status is set to "raw" (L64) and the ratios column names saved (L65).

For .tab files (L67–98) the processing is based on the file format shown in Figure 3.2. The name of each file is printed as it is loaded (L67). The first line is read (L68) and if the text of the first position is "grid_name" (L69) the first two lines are known to contain extra data. These values are loaded (L70) and the grid name, statuses and status file names saved (L71–73). The data types, names and values are read from lines 3, 4 and 5 onwards (L74–76). Statuses are reordered according to the order of the data file names (L77–78). If no extra data is provided the values are read from the first line (L80–82). The grid name is stored as 'unspecified" (L83) and the statuses as "raw" (L84). Coordinate columns are sought (L86–87) and the function

stopped with an error message if these are not all present (L88). The same is done for annotation data (L89–91). Coordinates, annotations and ratios are stored as new matrices and column names saved (L92–97).

Following the loading and processing of data, annotations are checked for unique entries (L99). A new `arrayFile` is created with the data (L100), ordered by coordinate values (L101), and column names are set (L102–104). If the `arrayData` object is not the first to be loaded it is joined with the others with the `cbind` method, otherwise it is left on its own (L105–109). The final `arrayData` object is returned when all files have been loaded (L111).

The `splitCoords` function

The `splitCoords` function (Script 3.2) runs within the `loadArray` function to process the probe coordinate data in the Agilent Feature Extraction file. This is in one column of the file, named "SystematicName", and contains genomic probe data in the format **"chr"** [**chromosome number**] [**colon**] [**start coordinate**] [**hyphen**] [**end coordinate**], for example, `chr1:100-160` for a probe on chromosome 1 that runs from position 100 to 160. Probe names are listed for non-genomic probes, such as controls. The `loadArray` function passes all SystematicNames that begin with "chr" to this function, which splits them into their component parts, that is, a chromosome number and start and end coordinate. These are put into a three-column matrix and returned to the `loadArray` function. This processing allows each probe to be associated with a genomic location in the `arrayData` object. The function has one argument:

x A character vector containing the coordinates to be split (no default).

Script 3.2: splitCoords: script to split the "systematicName" column of the Agilent Feature Extraction file into its component parts, namely the chromosome number and probe start and end coordinates.

```
1  ## splitCoords function ##
2  ## arguments: x (SystematicName column data to split)
3  splitCoords<-function(x) { #define function
4    split1A<-strsplit(x,"-") #split by "-"
5    doubles<-as.numeric(summary(split1A)[,1]) == 2 #identify doubles ie those
         split into two
```

```
6   split1<-matrix(unlist(split1A[doubles]),ncol=2,byrow=T) #get doubles
7   split2<-matrix(unlist(strsplit(split1[,1],":")),ncol=2,byrow=T) #split
        first half by ":"
8   split3<-matrix(unlist(strsplit(split2[,1],"chr")),ncol=2,byrow=T)[,2] #
        remove "chr" text from first half
9   coords<-matrix(ncol=3,nrow=length(x)) #initialise matrix to store
        coordinates
10  coords[doubles]<-matrix(cbind(split3,split2[,2],split1[,2]),ncol=3,byrow=F
        ) #recombine coordinates
11  return(coords) #return data
12 }
```

The function first breaks strings at the "-" location (L4) and the positions of those split into two are stored (L5). The first component of the strings split in two is split at the ":" location (L6–7) and the "chr" text removed from the beginning (L8). All coordinates are recombined in a new matrix and returned to the `loadArray` function (L9–11).

Validating `arrayData`

The `arrayDataValidity` function (Script 3.3) performs a series of checks on `arrayData` objects to ensure that they are in the correct format, as described in Section 3.2.1.2 and below, and is specified as the validity function when setting the `arrayData` class. Validity functions are run automatically in `R` every time a new object is created, to ensure that the object being specified matches the expected format. Functions in this package that modify `arrayData` objects return new objects, and so this checking is performed every time they are run. Its main purpose therefore is to check that the modification of the object has been carried out correctly. Users do not have any need to directly modify the contents of an `arrayData` object, but if they do so in such a way that they change the format of the object, this will create an error when another function attempts to modify it. The `arrayDataValidity` function can be called directly by a user if required, but there is no need for this to be done as a matter of routine. Other functions that use, but do not modify, `arrayData` objects do not perform this validity checking. The following details are checked:

- The object contains five slots (for coordinates, annotations, ratios, grid name and status).

- The first three slots contain matrices.
- The last slot contains a list.
- The names of the objects in the slots are correct ("coordinates", "annotations", "ratios", "grid_name" and "status" respectively).
- The number of columns of the matrices are correct (3, 1+ and 1+).
- The number of statuses equals the number of datasets.
- The coordinates and ratios are numeric.
- The first column of annotations contains only unique values.
- The three matrices have the same number of rows.
- A single grid name is present.

The function has the following argument:

object The `arrayData` object to be validated (no default).

Script 3.3: arrayDataValidity: script to check presented arrayData objects are in the correct format. Called by functions about to use an arrayData object. Returns TRUE if the format is correct, in which case the function carries on using the object, or an error message if not.

```
1  ## arrayDataValidity function ##
2  ## arguments: object (an arrayData object)
3  arrayDataValidity<-function(object) { #define function
4    if (length(object) == 5) { #length of arrayData is equal to 5
5      if (is.matrix(object[[1]]) & is.matrix(object[[2]]) & is.matrix(object
           [[3]]) & is.character(object[[4]]) & is.list(object[[5]])) { #first
           three slots contain matrices, fourth contains a chacacter vector and
            fifth contains a list
6        if (names(object)[1] == "coordinates" & names(object)[2] == "
             annotations" & names(object)[3] == "ratios" & names(object)[4] ==
             "grid_name" & names(object)[5] == "status") { #slots contain items
              of the correct names
7          if (ncol(object[[1]]) == 3 & ncol(object[[2]]) >= 1 & ncol(object
               [[3]]) >= 1 & length(object[[4]]) == 1 & length(object[[5]]) ==
               ncol(object[[3]])) { #slots contain items of the correct
               dimensions
8            if (nrow(object[[1]]) > 0) if (is.numeric(object[[1]][1,1]) & is.
                 numeric(object[[3]][1,1])) { #coordinates and ratios are
                 numeric
```

```
9             if(nrow(object[[1]]) > 0) if (length(which(duplicated(object
                  [[2]][,1]))) == 0) { #no probe names are duplicated
10              if (nrow(object[[1]]) == nrow(object[[2]]) & nrow(object[[1]])
                      == nrow(object[[3]])) { #all matrices are of the same
                    number of rows
11                if (nchar(object[[4]]) >= 1) { #a grid name is present
12                  TRUE #all criteria have been met; return TRUE
13                } else print("Grid name is absent") #a grid name is not
                      present; print message
14              } else print("Unequal matrix lengths") #matrices are of
                    differing numbers of rows; print message
15            } else print("Non-unique IDs present in annotations") #some
                  probe names are duplicated; print message
16          } else paste("Non-numeric entries found which should be numeric")
                #non-numeric coordinates/ratios found; print message
17        } else print("Items of incorrect length") #slots contain items of
              incorrect dimensions; print message
18      } else print("Slot names incorrect") #slots contain items of incorrect
            names; print message
19    } else print("Slots contain incorrect objects") #slots do not contain
        the expected objects; print message
20  }else print("Incorrect number of slots") #length of arrayData is not equal
      to 5; print message
21 }
22
23 ## Define arrayData class ##
24 setClass("arrayData",representation("list"),validity=arrayDataValidity) #set
     class with validity
```

The function checks the components in a specific order, each check relying on the last to be correct. That the length of the object is 5 is checked first (L4) allowing the formats (L5), names (L6) and dimensions (L7) of these 5 components to be checked. Coordinates and ratios are checked to be numeric (L8) and probe IDs are checked to be unique (L9). Components are checked to contain the same number of probes (L10). The grid name is checked to contain one entry (L11). If all of these checks are passed the function returns TRUE (L12), otherwise an error message corresponding to the failed check is returned (L13–20). This function is set as the validity check of the `arrayData` class (L24).

3.2.1.4 Writing `arrayData` to external files

There may be occasions where a user wishes to use data from an `arrayData` object in a different program, or for some other reason save their data outside of `R`. The function `writeArrayData` allows this by writing the data to a tab-delimited text file. The format of the text file is that shown in Figure 3.2, and as such the files written with this function can be loaded back into R

as an `arrayData` object with the `loadArray` function. The function has the following arguments:

object The `arrayData` object to be written to the file (no default).

fileName The name of the file to create (no default). The '.tab' extension is added if not present so the file will be correctly recognised by the `loadArray` function.

Script 3.4: writeArrayData: script to write an arrayData object to a tab-delimited text file. All information in the object is written in a format that can be read back in as an arrayData object by the loadArray function

```
1  ## writeArrayData function ##
2  ## arguments: object (an arrayData object), fileName (the name of the file
       to be created)
3  writeArrayData<-function(object,fileName) { #define function
4    validObject(object,test=T) #check arrayData object is correct
5    data<-matrix(ncol=3+ncol(object$annotations)+ncol(object),nrow=nrow(object
         )) #initialise matrix to store data
6    data[,1:3]<-matrix(object$coordinates,ncol=3) #put coordinates in first 3
         columns
7    data[,4:(3+ncol(object$annotations))]<-matrix(object$annotations,ncol=ncol
         (object$annotations)) #put annotations in next columns
8    data[,(4+ncol(object$annotations)):(3+ncol(object$annotations)+ncol(object
         ))]<-matrix(object$ratios,ncol=ncol(object)) #put ratios in last
         columns
9    top<-matrix(ncol=ncol(data),nrow=2) #initialise matrix to store extra info
10   top[,1]<-c("grid_name",object$grid_name) #store grid name
11   for (n in 1:ncol(object)) top[,(n+1)]<-c(colnames(object$ratios)[n],paste(
         object$status[[n]],collapse=",")) #store each data name
12   data<-rbind(top,c(rep("coords",ncol(object$coordinates)),rep("anno",ncol(
         object$annotations)),rep("ratios",ncol(object$ratios))),c(colnames(
         object$coordinates),colnames(object$annotations),colnames(object$
         ratios)),data) #combine all together into final format
13   if(missing(fileName)) fileName<-deparse(substitute(object))
14   if(substr(fileName,nchar(fileName)-3,nchar(fileName)) != ".tab") substr(
         fileName,nchar(fileName)-3,nchar(fileName))<-".tab" #change file
         extension to ".tab" if not already
15   write.table(data,fileName,quote=F,sep="\t",row.names=F,col.names=F) #write
          to tab delimited file
16   message(paste("arrayData object \"",deparse(substitute(object)),"\"
         written to ",fileName,sep="")) #print message
17 }
```

The function first confirms the validity of the `arrayData` object (L4), to ensure that all the data is in the required format to write to the file. A new matrix is created and the coordinate, annotation and ratio data added to it (L5–8). Another matrix is created for the extra data and the grid name, file names and statuses added to it (L9–11). Both matrices are combined (L12).

If no file name is provided it is set to be the object name (L13) and a '.tab' extension added to the end if required (L14). The object is then written (L15) and a confirmatory message displayed (L16).

3.2.1.5 The `genomeAnnotation` class

In addition to loading microarray data, the `biomaRt` (Durinck et al., 2005) package has been used to load complementary genome annotation data. This is used to determine the positions of probes relative to ORFs, which is used in several plotting functions and to assign location-specific information to probes. This information is loaded into a `genomeAnnotation` object which is of a similar format to the `arrayData` object, in that it is a list structure containing several matrices, in the following format.

`coordinates` A three column matrix containing the chromosome number, start position and end position of each ORF.

`annotations` A three column matrix containing the name, chromosome name and strand of each ORF. The chromosome number is always numeric, and is based on the chromosome name in the database. If this is numeric both will be the same, otherwise the next available number is assigned. These values are stored in the "coordinates" matrix while the original names are stored in the "annotations" matrix.

`dataset` The name of the `biomaRt` dataset used to load the data.

The `biomaRt` package links to databases of genome annotations for several organisms, including *S. serevisiae* and humans. The `loadAnnotation` function (Script 3.6) was written to extract information from these. The function contains the following arguments:

mart A character vector specifying the database to use (default "ensembl", which should not need to be changed.)

dataset A character vector specifying the name of the dataset to be accessed (default "scerevisiae_gene_ensembl").

attributes A vector containing the names of the attributes in the database to be loaded (defaults "external_gene_id" (gene name), "chromosome_name"

(chromosome name or number), "start_position" (ORF start coordinate), "end_position" (ORF end coordinate), and "strand" (ORF strand), which should be consistent between different datasets and therefore should not need to be changed).

chromosomes A vector containing the names of the chromosomes to be loaded (default Roman numerals from 1 to 16 for *S. serevisiae*; for humans they should be listed as the numbers 1 to 22 plus "X" and "Y").

Script 3.5: loadAnnotation: script to load genome annotation data using the biomaRt package. Gene names, coordinates and strands along with the dataset name are stored as the newly defined `genomeAnnotation` class.

```
1  ## loadAnnotation function ##
2  ## arguments: mart (biomaRt mart to use), dataset (biomaRt dataset to use),
       attributes (data to load from database), chromosomes (the organisms
       chromosoms names as they appear in the database)
3  loadAnnotation<-function(mart="ensembl",dataset="scerevisiae_gene_ensembl",
       attributes=c("external_gene_id","chromosome_name","start_position","end_
       position","strand"),chromosomes=as.roman(1:16)) { #define function
4    require(biomaRt) #load biomaRt package if not already done so
5    annotation<-as.matrix(getBM(mart=useMart(mart, dataset = dataset),
         attributes=attributes)) #download data (gene name, chromosome name,
         gene start, gene end, strand) from mart/dataset
6    chromosomes<-as.character(chromosomes) #set chromosomes as characters
7    annotation<-annotation[which(as.character(annotation[,2]) %in% chromosomes
         ),] #get data matching defined chromosomes
8    annotation<-cbind(annotation,matrix(ncol=1,nrow=nrow(annotation))) #add
         new column to data
9    for (n in 1:length(chromosomes)) { #loop through chromosomes
10     annotation[which(annotation[,2] == chromosomes[n]),6]<-n #set numerical
           value for each chromosome
11   }
12   coords<-matrix(as.numeric(annotation[,c(6,3,4)]),ncol=3) #get coordinates
13   annotations<-matrix(annotation[,c(1,2,5)],ncol=3) #get annotations
14   anno<-new("genomeAnnotation",list(coordinates=coords,annotations=
         annotations,dataset=dataset)) #put data in new genomeAnnotation object
15   anno<-anno[order(anno$coordinates[,1],anno$coordinates[,2])] #order
         genomeAnnotation object
16   return(anno) #return genomeAnnotation object
17 }
```

The function first ensures the `biomaRt` package is loaded (L4) and uses it to load the specified annotations (L5). Provided chromosomes are converted to characters (L6) and used to get the required data from the loaded annotations (L7). A new column is added to this data (L8) and used to store numeric chromosome values based on the order they are provided (L9–11). Data

is split into coordinates (L12) and annotations (L13) and added to a new `genomeAnnotation` object (L14). This is ordered by coordinate values (L15) and returned to the user (L16). Messages are generated by `biomaRt` if errors occur.

Manipulation of `genomeAnnotation` objects

The `genomeAnnotation` objects are not intended to be called directly by users or modified by functions other than `loadArray` and so has few methods for its manipulation (Script 3.6). The "dim" method has been defined to show the number of ORFs present, and square brackets to extract details for particular positions. To prevent the object being displayed in full and filling up the `R` console the `show` method has been defined to give a message showing the number of ORFs in the object and the dataset from which they came.

Script 3.6: genomeAnnotation: scripts to process `genomeAnnotation` objects. The class is first defined. The show method displays a single line of information, the dim method returns the dimensions of the coordinates matrix and the extraction method allows particular genes to be extracted.

```
1  ## define genomeAnnotation class ##
2  setClass("genomeAnnotation",representation("list"))
3
4  ## genomeAnnotation show function ##
5  ## arguments: object (a genomeAnnotation object)
6  setMethod("show", "genomeAnnotation", function(object) { #define function
7    message(paste("genomeAnnotation object of length",nrow(object),"from",
         object$dataset)) #print message
8  }
9  )
10 ## genomeAnnotation extract function ##
11 ## arguments: object (a genomeAnnotation object), i (rows)
12 setMethod("[", "genomeAnnotation", function(x,i,...) { #define function
13   if (nargs() != 2) stop("One subscript required", call. = FALSE) #check
         only one subscript (for rows)
14   return(new("genomeAnnotation",list(coordinates=matrix(x[[1]][i,],ncol=ncol
         (x[[1]])),annotations=matrix(x[[2]][i,],ncol=ncol(x[[2]])),dataset=x$
         dataset))) #return new genomeAnnotation object
15 }
16 )
17 ## genomeAnnotation dim function
18 ## arguments: x (a enomeAnnotation object)
19 setMethod("dim", "genomeAnnotation", function(x) { #define function
20   return(dim(x[[1]]))
21 }
22 )
```

The "show" method (L6) prints the length of the object (the number of ORFs it contains) and the name of the dataset it was downloaded from (the organism name) (L7). The extraction method (L12) accepts one argument (L13) and returns all information relating to the specified positions (L14). The "dim" method (L19) prints the number of genes in the object.

3.2.2 Quality assessment

It is prudent to assess the quality of data produced by a microarray before any analyses are undertaken, as there are several stages at which problems can be introduced into the ChIP-chip procedure which can render the final data unreliable. The `checkData` function (Script 3.7) loads Agilent Feature Extraction files and produces a set of graphics which can aid in this assessment. This helps any faults or irregularities in the data or on the microarray to be identified and a decision made as to whether or not to go on to load the data for analysis. The `limma imageplot` function is used to create pseudo-images of the arrays. This is achieved by loading the coordinates of the physical locations of the probes on the array and representing the corresponding probe intensities at those positions in the image. One image is created for each of the two channels. These allow artifacts on the microarray surface, such as scratches, or regions of poor hybridisation to be visualised. At this point a user may decide to reject a dataset from further analyses if there is an obvious defect on the microarray which will likely make some, if not all, of the data unreliable.

The intensity values of the two channels are shown as box plots, which allows a visual estimation of the range of values, which can be compared between different datasets to identify any with unusual features. Additional box plots can be displayed for microarrays containing custom spike in probes (see Section 4.4.1), allowing the signal intensities of the spike probes to be compared to the rest of the probes. This can be useful to see whether or not the spike in values are in the correct range, which would be expected to be similar to the range of the genomic probe values if spike ins are included, or lower than the genomic values if they are not. Density plots of the genomic

probes are also created, which can show unusual patterns in the distributions, such as skewness or unevenness, which may suggest that the data are unreliable. A density plot of the red:green $\log_2$ ratios is also created, showing the pattern of the final results.

A scatter plot of the red and green values allows the relationship between the two channels to be visualised. The function `smoothScatter` is used to create this plot, which displays more dense areas of points with darker colours. This allows the data to be better visualised than a standard scatter plot. The function has the following arguments:

fileName A character vector specifying the names of the files to load (no default).

essentialColumns A list of the names of the columns in the Feature Extraction text file containing essential information. Red and green intensity values are taken from the "rBGSubSignal" and "gBGSubSignal" columns respectively, probe row and column positions from "Row" and "Col" respectively and coordinates from "SystematicName" by default and do not need to be modified under normal circumstances.

spikes A logical value specifying whether or not spike probes are included on the microarrays (default FALSE).

spikeStart A character vector specifying text used to identify spike probes (default ">").

Script 3.7: checkData: script to load Agilent Feature Extraction text files and create a series of graphics in order to assess the quality of the microarray and the data it has produced.

```
1 ## checkData function ##
2 ## arguments: fileName (name of Agilent FE file), essentialColumns (required
      columns from the feature extraction file), spikes (whether or not the
     array contains spikes, spikeStart (character defining spike probes)
3 checkData<-function(fileName, essentialColumns=list(red="rBGSubSignal",
     green="gBGSubSignal", row="Row", col="Col", coords="SystematicName"),
     spikes=F,spikeStart=">") { #define the function
4   if (length(essentialColumns$red) == 0 | length(essentialColumns$green) ==
       0 | length(essentialColumns$row) == 0 | length(essentialColumns$col)
       == 0 | length(essentialColumns$coords) == 0) stop("Essential columns
       missing", call.=F) #check all essential columns are defined
5   require(limma) #ensure limma package is loaded
6   if (missing(fileName)) fileName<-list.files(pattern=".txt$") #search for
       all .txt file if no filename is provided
```

```
7   for (n in 1:length(fileName)) { #loop through files
8     message(paste("Loading:",fileName[n])) #print the name of the file being
          loaded
9     columnNames<-scan(fileName[n],skip=9,nlines=1,what="",quiet=T) #get
          column names from row 10 of the file
10    if (length(grep(essentialColumns$red,columnNames)) != 1 | length(grep(
          essentialColumns$green,columnNames)) != 1 | length(grep(
          essentialColumns$row,columnNames)) != 1 | length(grep(
          essentialColumns$col,columnNames)) != 1 | length(grep(
          essentialColumns$coords,columnNames)) != 1) stop("Essential columns
          not present", call.=F)  #stop with message if not all essential
          columns present in file
11    totalColumns<-length(columnNames) #get total numer of columns of data
12    columnRead<-rep("NULL",totalColumns) #set all columns to NULL (so they
          are not read)
13    columnRead[which(columnNames %in% essentialColumns)]<-NA #set listed
          columns with NA (in order to be read)
14    arrayFile<-as.matrix(read.table(fileName[n],colClasses=columnRead,skip
          =9,header=T,fill=T,sep="\t",quote="")) #read data into R
15    red<-as.numeric(arrayFile[,which(colnames(arrayFile) == essentialColumns
          $red)]) #get red channel values
16    green<-as.numeric(arrayFile[,which(colnames(arrayFile) ==
          essentialColumns$green)]) #get green channel values
17    rows<-as.numeric(arrayFile[,which(colnames(arrayFile) ==
          essentialColumns$row)]) #get row coordinates
18    cols<-as.numeric(arrayFile[,which(colnames(arrayFile) ==
          essentialColumns$col)]) #get column coordinates
19    maxR<-max(rows) #get maximum row value
20    maxC<-max(cols) #get maximum column value
21    full<-paste(sort(rep(1:maxR,maxC)),rep(1:maxC,maxR),sep="-") #get all (
          full) potential positions
22    actual<-paste(rows,cols,sep="-") #get actual array positions
23    same<-full %in% actual #get full positions actually present
24    redNew<-greenNew<-numeric() #initialise vectors
25    redNew[same]<-red #get red values in position
26    greenNew[same]<-green #get green values in position
27    red<-redNew #redefine red
28    green<-greenNew #redefine green
29    par(bty="n") #don't plot boxes around plots
30    layout(matrix(c(1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 2, 2, 2,
           2, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8),
          byrow=T,ncol=8),height=c(2,5,5,10,10)) #define layout
31    if(spikes) layout(matrix(c(1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3,
          3, 2, 2, 2, 2, 4, 4, 4, 4, 5, 5, 5, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9,
           9, 9),byrow=T,ncol=8),height=c(2,5,5,10,10)) #define layout with
          spikes
32    par(mar=c(0,0,3,0)) #set margins
33    plot(0,1,type="n",bty="n",xaxt="n",yaxt="n",main=fileName[n],xlab="",
          ylab="") #print file name
34    plot(0,1,type="n",bty="n",xaxt="n",yaxt="n",main="",xlab="",ylab="",xlim
          =c(0,1),ylim=c(0,6)) #initialise plot to print text:
35    text(0,1,"Cy3 (Green):",pos=4)
36    text(0,2,"Cy5 (Red):",pos=4)
37    text(0,3,"Organism & Strain:",pos=4)
38    text(0,4,"Date:",pos=4)
39    text(0,5,"Name:",pos=4)
40    text(0,6,"Description:",pos=4)
41    imageplot(c(red,rep(0,((max(cols)*max(rows))-length(red)))),list(ngrid.r
          =1,ngrid.c=1,nspot.r=max(rows),nspot.c=max(cols)),low = "white",
          high = "red",  zlim = c(0, mean(red,na.rm=T) + 3*mad(red,na.rm=T)),
          legend=F,mar=c(0.1,3,0.1,3),xlab="",ylab="") #create red pseudoimage
           using limma's imageplot
```

```
42      imageplot(c(green,rep(0,((max(cols)*max(rows))-length(red)))),list(ngrid
            .r=1,ngrid.c=1,nspot.r=max(rows),nspot.c=max(cols)),low = "white",
            high = "green",  zlim = c(0, mean(green,na.rm=T) + 3*mad(green,na.rm
            =T)),legend=F,mar=c(0.1,3,0.1,3),xlab="",ylab="") #create green
            pseudoimage using limma's imageplot
43      if(spikes) { #if spikes are present
44        spikeData<-arrayFile[grep(spikeStart,arrayFile[,which(colnames(
              arrayFile) == essentialColumns$coords)]),] #get spike data
45      }
46      arrayFile<-arrayFile[grep("chr",arrayFile[,which(colnames(arrayFile) ==
            essentialColumns$coords)]),] #get only probes with chromosomal
            coordinates
47      red<-as.numeric(arrayFile[,which(colnames(arrayFile) == essentialColumns
            $red)]) #get red channel values
48      green<-as.numeric(arrayFile[,which(colnames(arrayFile) ==
            essentialColumns$green)]) #get green channel values
49      startWarn<-as.numeric(options("warn")); on.exit(options(warn=startWarn))
             #maintain warning state on exit
50      options(warn=-1); ratios<-log2(red/green) #calculate log2 ratios without
             warnings
51      red<-log2(red) #log2 red values
52      green<-log2(green) #log2 green values
53      rS<-gS<-NA #set red and green spike values to NA
54      if(spikes) { #if spikes are present
55        rS<-log2(as.numeric(spikeData[,which(colnames(spikeData) == "
              rBGSubSignal")])) #get red spike values
56        gS<-log2(as.numeric(spikeData[,which(colnames(spikeData) == "
              rBGSubSignal")]) ) #get green spike values
57      }
58      options(warn=startWarn) #reset warning state
59      par(mar=c(5, 4, 1.5, 1)) #set margins
60      boxplot(list(red,green),col=c(2,3),names=c("Red","Green"),main="Signal
            Intensities",ylab="Log2 Signal",ylim=c(range(c(red,green),na.rm=T)))
             #plot signal intensities as boxplot
61      if(spikes) boxplot(list(rS,gS),col=c(2,3),names=c("Red","Green"),main="
            Spikes",ylab="Log2 Signal",ylim=c(range(c(red,green),na.rm=T))) #
            plot signal intensities as boxplot
62      par(xaxt="s") #set x-axis type
63      smoothScatter(green,red,main="Red v Green",xlab="Log2 Green Signal",ylab
            ="Log2 Red Signal",pch=20) #red v green plot
64      abline(0,1,col="blue") #add line y=x
65      r<-density(red,na.rm=T) #calcualte red density
66      g<-density(green,na.rm=T) #calculate green density
67      plot(r,col="red",lwd=2,xlab="Log2 Signal",main="Signal Intensities",xlim
            =range(c(r$x,g$x)),ylim=range(c(r$y,g$y))) #plot red density
68      lines(g,col="green",lwd=2) #add green density
69      plot(density(ratios,na.rm=T),lwd=2,main="Red/Green Ratios",xlab="Log2
            Ratio") #plot log2 ratio densities
70    }
71  }
```

The function first checks that the names of the required columns have been specified (L4) and loads the `limma` package if required (L5). File names with the '.txt' extension are taken from the working directory if not provided in the "fileName" argument (L6). A loop is initiated to load each file individually (L7). The name of each file is is printed as it is loaded (L8). The tenth

line of the file is scanned for the required column names (L9) and an error message displayed if they are not all present (L10). The data values from the specified columns are then read from the eleventh line of the file onwards (L11–14).

Red, green, row and column values are extracted (L15–18). Row and column numbers are used to determine the probe positions within the grid (L19–23) and these positions used to reassign the red and green values to correspond to their correct positions (L24–28). The plotting parameters and layouts are defined (L29–32) and a series of text labels printed in the first plot region (L33–40). Pseudo-images of the red and green values are created (L41–42). Spike data are extracted if required (L43–45) and genomic probes stored (L46). Red and green values are again extracted (L47–48) and used to calculated $\log_2$ red and green values as well as their ratios, with warning messages disabled (L49–58). Parameters are adjusted (L59) for box plot plotting (L60–61) and again (L62) for scatter plotting (L63–64). Red and green intensity value densities are plotted (L65–67) followed by their $\log_2$ ratios (L69).

An example of the output of this function is shown in Figure 3.3, created from an Abf1 binding dataset (see Chapter 7). The pseudo-images do not show any obvious abnormalities on the microarray surface. The box plots show the bulk of the intensity values are in the expected range and the density curves show smooth distributions, indicating that both the red and green channels have produced good quality results. The scatter plot shows a good relationship between the two channels, with some probes higher in the red (IP sample) channel than the green (input sample), as expected. This is also shown in the density plot, with the tail on the right hand side as a result of the enriched regions.

The application of a statistical test could also be used to determine the similarity or otherwise of replicate datasets. A test such as Spearman's rank correlation coefficient may be used to test the relationship between two datasets in a non-parametric manner, allowing comparisons to be made between all replicate datasets. This would allow the objective identification of any datasets that do not follow the same properties of their replicates at

a defined P-value cutoff (under the hypothesis that the rho value is equal to zero; after the application of a multiple testing correction where appropriate). This can then be used in conjunction with the quality control graphics to determine whether or not to use datasets for further analyses.

3.2.3 Accessing data

Extracting specific probe data from an `arrayData` object is important both for users and functions working with them. In `R` this is achieved with a square bracket notation to define the required part(s) of an object. This notation is specific to the object type being used, and each has a method associated with it which defines how data is extracted from it. A method has been defined for the `arrayData` object to allow probe data to be extracted from it (Script 3.8). With regard to 2-dimensional objects, row and column numbers are specified. For example, `matrixName[2,3]` returns data from the third column of the second row of the named matrix. This same format is used to access probe data from an `arrayData` object, with row numbers relating to probes and column numbers relating to datasets. In addition, probe names may be specified as the rows argument, in which case the relevant rows are determined from the unique probe names in the `annotations` matrix. A new `arrayData` object is created with ratio data from the specified probe(s) and dataset(s), coordinate and annotation data for the probe(s), status data for the dataset(s) and the grid name for the object. This method can also be used to split `arrayData` objects into multiple different objects. This may be required if, for example, multiple datasets that represent different conditions have been loaded at the same time. These may be subject to different downstream processing and so it is useful to store them, and therefore process them, separately. The dollar ($) notation can be used to directly access any of the components of an `arrayData` object, for which no specific methods need be defined.

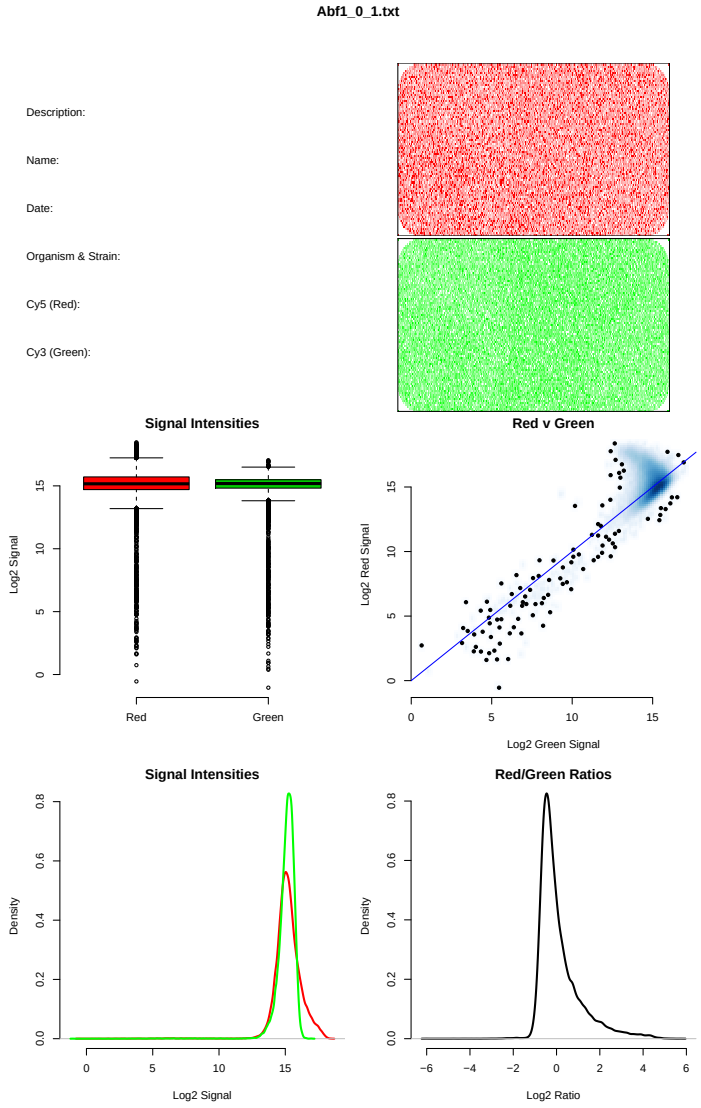


Figure 3.3: Output of the `checkData` function: The name of the file is printed at the top of the page with space for details of the microarray to be written. Pseudo images (top right) can show artifacts on the slide surface. Box plots and density curves of red and green signal intensities show the distributions of the values. A scatter plot shows the relationship between the two channels and a density curve shows the distribution of the $\log_2$ ratio values.

Script 3.8: arrayData extraction: extracts data from the given positions into a new arrayData object.

```
1  ## [ (extract) function ##
2  ## arguments: x (an arrayData object), i (required row number(s)), j (
       required column number(s))
3  setMethod("[", "arrayData", function(x,i,j) { #define function
4    if (nargs() != 3) stop("Two values required", call. = FALSE) #ensure
         correct dimensions are specified
5    if (missing(i)) i<-1:nrow(x$ratios) #if no colums specified get all
6    if (missing(j)) j<-1:ncol(x$ratios) #if no rows specified get all
7    if (is.character(i)) { #if IDs are provided
8      for (n in 1:length(i)) { #loop through IDs
9        probe<-which(x$annotations[,1] == i[n]) #search for ID
10       if (length(probe) > 0) { #if ID is found
11         i[n]<-probe #get row number
12       }else{ #ID is not found
13         i[n]<-0 #don't get a row
14       }
15     }
16     i<-as.numeric(i) #ensure numeric
17   }
18   newArrayData<-new("arrayData",list(coordinates=matrix(x$coordinates[i,],
         ncol=ncol(x$coordinates),dimnames=list(NULL,colnames(x$coordinates))),
         annotations=matrix(x$annotations[i,],ncol=ncol(x$annotations),dimnames
         =list(NULL,colnames(x$annotations))),ratios=matrix(x$ratios[i,j],ncol=
         ifelse(j[1]>0,length(j),(ncol(x)-length(j))),dimnames=list(NULL,
         colnames(x$ratios)[j])),grid_name=x$grid_name,status=x$status[j])) #
         create new arrayData object from extrated data
19   return(newArrayData) #return desired data
20 }
21 )
```

The function checks the number of arguments is correct (L4) and assigns full row and column ranges if they are not provided (L5–6). If characters are specified for rows the first "annotations" column is used to identify the corresponding row number (L7–17). A new (arrayData) object is created containing the specified probes and datasets (L18) and returned (L19).

3.2.4 Manipulation of `arrayData` objects

Mathematical manipulation of the ratio values of `arrayData` objects forms a crucial part of their processing, most importantly here as part of the normalisation procedure. Users may also need to perform these operations directly, either to create further data to analyse, such as calculating the differences between two datasets, or to temporarily adjust data, such as to display different datasets on a similar scale on the same graph. The standard mathematical operator methods (+, -, * and /) have been defined to apply the operations to the ratio values of `arrayData` objects (Script 3.9). Processing common to

all procedures is carried out by the `maths` function. Two arguments must be specified, the first of which must be an `arrayData` object. The second can be another `arrayData` object, or a single number. A single number is used to create a new `arrayData` object, the ratio values of which are filled with that number. Probes common to both are maintained and any others removed, with a warning message. If different numbers of datasets are provided the extras are removed with a warning message. These processed objects are returned to the individual functions, which perform the mathematical operations on the ratio values of these objects. The results of the calculations are returned as the ratios of a new `arrayData` object.

Script 3.9: Mathematical operators: Scripts to manipulate arrayData objects by the standard mathematical operators +, -, * and /. The maths function performs processing common to all operations. The individual methods apply the operator and return a new arrayData object.

```
## mathematical operator functions
## arguments: e1 (first arrayData object), e2 (second arrayData object or single number)
## processing common to all operations:
maths<-function(e1,e2) { #define function
  if(length(e2) == 1 & is.numeric(e2)) { #if a single number is provided in e2
    number<-e2 #store the number
    e2<-e1 #set e2 as arrayData object from e1
    e2$ratios<-matrix(ncol=ncol(e2),nrow=nrow(e2),number) #fill ratios with number
    colnames(e2$ratios)<-rep(number,ncol(e2)) #set column names
  }
  if (ncol(e1) != ncol(e2)) {
    if (ncol(e1) > ncol(e2) ) e1<-e1[,1:ncol(e2)] else e2<-e2[,1:ncol(e1)]
    warning("Differing numbers of datasets: extras removed", call.=F)
  }
  matchingProbes1<-which(e1$annotations[,1] %in% e2$annotations[,1]) #find matching probes in e1
  matchingProbes2<-which(e2$annotations[,1] %in% e1$annotations[matchingProbes1,1]) #find matching probes in e2
  object<-e1[matchingProbes1,] #get arrayData containing matching probes
  if(length(matchingProbes1) != nrow(e1) | length(matchingProbes2) != nrow(e2)) warning("Differing arrayData lengths: non-matching probes removed", call. = FALSE) #warn if not all probes match
  e2<-e2[matchingProbes2,]
  object$status<-as.list(rep("Processed",ncol(object))) #set status of datasets to "processed"
  return(list(e2=e2,object=object)) #return all arrayData objects
}
##Add datasets
setMethod ("+","arrayData", function(e1,e2) { #define function
  processed<-maths(e1,e2) #get processed arrayData objects
  e1<-processed$object #get new object
  e2<-processed$e2 #get new e2
```

```
28    e1$ratios<-e1$ratios+e2$ratios #add ratios
29    colnames(e1$ratios)<-paste(colnames(e1$ratios),"+",colnames(e2$ratios),sep
          ="") #adjust column names
30    return(new("arrayData",e1)) #return arrayData object
31  }
32  )
33  ##Subtract datasets
34  setMethod ("-","arrayData", function(e1,e2) { #define function
35    processed<-maths(e1,e2) #get processed arrayData objects
36    e1<-processed$object #get new object
37    e2<-processed$e2 #get new e2
38    e1$ratios<-e1$ratios-e2$ratios #subtract ratios
39    colnames(e1$ratios)<-paste(colnames(e1$ratios),"-",colnames(e2$ratios),sep
          ="") #adjust column names
40    return(new("arrayData",e1)) #return arrayData object
41  }
42  )
43  ##Multiply datasets
44  setMethod ("*","arrayData", function(e1,e2) { #define function
45    processed<-maths(e1,e2) #get processed arrayData objects
46    e1<-processed$object #get new object
47    e2<-processed$e2 #get new e2
48    e1$ratios<-e1$ratios*e2$ratios #multiply ratios
49    colnames(e1$ratios)<-paste(colnames(e1$ratios),"*",colnames(e2$ratios),sep
          ="") #adjust column names
50    return(new("arrayData",e1)) #return arrayData object
51  }
52  )
53  ##Divide datasets
54  setMethod ("/","arrayData", function(e1,e2) { #define function
55    processed<-maths(e1,e2) #get processed arrayData objects
56    e1<-processed$object #get new object
57    e2<-processed$e2 #get new e2
58    e1$ratios<-e1$ratios/e2$ratios #divide ratios
59    colnames(e1$ratios)<-paste(colnames(e1$ratios),"/",colnames(e2$ratios),sep
          ="") #adjust column names
60    return(new("arrayData",e1)) #return arrayData object
61  }
62  )
```

The maths function examines the second argument to see if it is a number (L5). In this case a new `arrayData` object is created and a single set of ratio values filled with this number (L6–10). If two `arrayData` objects containing different numbers of datasets are provided the longer is reduced to the length of the shorter and a warning message displayed (L11–14). If two `arrayData` objects containing different numbers of probes are provided, those common to both are maintained and a warning message displayed if required (L15–19). The status of the first object is changed (L20) and both adjusted objects returned to the original function (L21). These adjusted datasets are processed, the dataset names updated, and the results returned as a new `arrayData` object for the addition (L24–32), subtraction (L34–42),

multiplication (L44–52) and division (L54–62) functions.

rowMeans

The `rowMeans` method has been defined to calculate average ratio values from multiple datasets. This can be used to reduce replicate datasets to a single averaged dataset.

Script 3.10: arrayData rowMeans: Script to calculate ratio row means using the rowMeans method. A new arrayData object is returned containing a single set of averaged ratios.

```
1  ## rowMeans function ##
2  ## arguments: x (an arrayData object)
3  setMethod("rowMeans", "arrayData", function(x) { #define the function
4    averaged<-x[,1] #initialise new object to store means
5    averaged$ratios<-as.matrix(ncol=1,rowMeans(x$ratios)) #calculate ratio row
         means
6    colnames(averaged$ratios)<-paste("rowMeans of: ",paste(colnames(x$ratios),
       collapse=","),sep="") #set new data name
7    averaged$status<-x$status[1] #set status of first object
8    return(new("arrayData",averaged)) #return mean data
9  }
10 )
```

A new `arrayData` object is created with a single dataset to store the averaged values (L4). The `rowMeans` function is applied to the "ratios" matrix and the results stored (L5). The dataset name is set to show all its component datasets (L6), the status modified (L7) and the resulting object returned (L8).

cbind

Many of the functions written here accept only one `arrayData` object for processing. There may be instances where multiple `arrayData` objects are present, either because they have been loaded separately or processed to create separate results, which a user requires to pass together to another function. The `cbind` method allows multiple columns of data to be combined together. The method has been defined for `arrayData` objects (Script 3.11), which joins multiple columns of ratio values together. This allows multiple sets of ratios to be joined and associated with a single set of coordinate and

annotation data. The function will only join objects with the same grid name, preventing different data formats coming together in the same object. If differing numbers of probes are provided for the same grid name, only those appearing in all datasets are maintained and a warning message is displayed. A new `arrayData` object containing the combined ratios is returned.

Script 3.11: arrayData cbind: Script to combine data from multiple arrayData objects using the cbind method. The ratio values are combined for probes common to all datasets and a new arrayData object returned.

```
1  ## cbind function ##
2  ## arguments: ... (any number of arrayData objects)
3    cbind.arrayData<-function(...,deparse.level) {
4    objects<-list(...) #get arrayData objects
5    gridName<-objects[[1]]$grid_name #get first grid name
6    cbinded<-objects[[1]] #initialise new arrayData object to store results
7    warn<-FALSE #set warn to FALSE
8    if (length(objects) > 1) { #if there are more than 1 arrayData objects
9      for (n in 2:length(objects)) { #loop through arrayData objects
10       if(cbinded$grid_name != objects[[n]]$grid_name) warning("Differing
             grid names", call.=F) #stop if grid names are different
11       matchingProbes1<-which(cbinded$annotations[,1] %in% objects[[n]]$
             annotations[,1]) #find matching probes
12       matchingProbes2<-which(objects[[n]]$annotations[,1] %in% cbinded$
             annotations[matchingProbes1,1]) #find matching probes
13       if(length(matchingProbes1) != nrow(cbinded) | length(matchingProbes2)
             != nrow(objects[[n]])) warn<-TRUE #set warn to TRUE if differing
             numbers of probes
14       cbinded<-cbinded[matchingProbes1,] #get matching probes
15       cbinded$ratios<-as.matrix(cbind(cbinded$ratios,objects[[n]][
             matchingProbes2,]$ratios)) #join ratios with identical IDs
16       cbinded$status<-c(cbinded$status,objects[[n]]$status) #join statuses
17     }
18   }
19   if(warn) warning("Differing arrayData lengths: non-matching probes removed
         ", call. = FALSE) #warning message if probes have been removed
20   return(new("arrayData",cbinded)) #return data
21 }
```

The function first stores all separate `arrayData` objects in a list (L4), gets the grid name from the first object (L5) and initialises a new object to store the results (L6). The warning state is set to FALSE (L7). A loop is initiated starting at the second object if more than one is provided (L8–9). Grid names are compared to the first object and the function stopped with a message if they do not match (L10). Probes common to both objects are maintained and the warning state set to TRUE if any are removed (L11–14). The ratio values and statuses are then combined (L15–16). A warning is printed if the warning state is TRUE (L19) and the processed object returned (L20).

3.2.5 Displaying data

The `arrayData` object format provides a method of storing ChIP-chip data, but is not well suited to displaying this in a meaningful way. Therefore the `dim`, `show` and `summary` methods have been defined for this object, to enable useful information to be quickly accessed without any processing required by the user (Script 3.12). The `dim` method returns dimensions of an object, that is, numbers of rows and columns. Dimensions of an `arrayData` object are taken from the `ratios` matrix. Therefore an object containing 5 datasets of 1,000 probes will be deemed to have 1,000 rows and 5 columns.

Script 3.12: Displaying `arrayData objects`: 1. Script to calculate the dimensions of an arrayData object, taken from the ratios slot. Therefore the number of rows is the number of probes and the number of columns is the number of datasets. 2. Script to print details of an arrayData object to the R console using the show method. Displays data names, normalisation statuses and the number of probes. 3. Script to print a summary of an arrayData object to the R console using the summary method. Displays grid name, probe and dataset counts, annotation names, file names and probe statistics.

```
## arrayData dim function ##
## arguments: x (an arrayData object)
setMethod("dim", "arrayData", function(x) { #define function
  return(dim(x$ratios)) #dimensions relate to ratios
}
)
## arrayData show function ##
## arguments: object (an arrayData object)
setMethod("show", "arrayData", function(object) { #define function
  message("An arrayData object containing:") #print message
  for (n in 1:length(colnames(object$ratios))) { #loop through datasets
    message(paste("\t",colnames(object$ratios)[n])) #print data name
    message(c("\t\tNormalisation procedures: ",paste(object$status[[n]],
        collapse="->"))) #print data status
  }
  message("Number of probes:") #print message
  message(paste("\t",nrow(object))) #print the number of probes
}
)
## arrayData summary function ##
## arguments: object (an arrayData object)
setMethod("summary", "arrayData", function(object) { #define the method
  message("Summary of arrayData object") #print message
  arraySummary<-data.frame(1) #initialise data frame
  arraySummary[1,1]<-object[[4]] #get grid name
  arraySummary[2,1]<-nrow(object) #get number of probes
  arraySummary[3,1]<-ncol(object) #get number of datasets
  rownames(arraySummary)<-c("Grid","Probes","Datasets") #name rows
  colnames(arraySummary)<-"" #blank column names
  annos<-matrix(colnames(object$annotations),nrow=1) #get annotations
```

```
30    rownames(annos)<-"" #blank row names
31    colnames(annos)<-1:length(colnames(object$annotations)) #name columns
32    chrSummary<-matrix(ncol=7,nrow=length(unique(object[[1]][,1]))) #
          initialise matrix
33    colnames(chrSummary)<-c("Chromosome","Number of probes","Lowest coordinate
          ","Highest coordinate","Lowest ratio","Highest ratio","NA values") #
          name columns
34    chrSummary[,1]<-sort((unique(object[[1]][,1]))) #get chromosomes
35    rownames(chrSummary)<-chrSummary[,1] #name rows
36    files<-matrix(colnames(object$ratios),nrow=1) #get data names
37    rownames(files)<-"" #blank row names
38    colnames(files)<-1:length(colnames(object$ratios)) #name columns
39    for (n in 1:length(chrSummary[,1])) { #loop through chromosomes
40      data<-which(object[[1]][,1] == chrSummary[n,1]) #get chromosome data
41      chrSummary[n,2]<-length(data) #number of probes
42      chrSummary[n,3]<-min(object[[1]][data,2]) #min coordinate
43      chrSummary[n,4]<-max(object[[1]][data,3]) #max coordinate
44      chrSummary[n,5]<-min(object[[3]][data,],na.rm=T) #min ratio
45      chrSummary[n,6]<-max(object[[3]][data,],na.rm=T) #max ratio
46      chrSummary[n,7]<-length(which(is.na(object[[3]][data,]))) #NA count
47    }
48    summaryList<-list(Summary=arraySummary,Annotations=annos,Files=files,
          Chromosomes=chrSummary[,-1]) #create list
49    return(summaryList) #return summary data
50  }
51  )
```

The `dim` method returns the dimensions of the ratio component of an `arrayData` object (L4). The `show` method (L9) prints a series of messages showing the names of datasets (L12), the normaliation procedures applied (L13) and the number of probes (L16). The `summary` method (L21) compiles a series of information. A data frame is created (L23) containing the grid name (L24), number of probes (L25) and number of datasets (L26). A matrix is created (L29) to store the names of the annotation columns (L31). A second matrix is created (L32–38) to store chromosomes. Statistics for each chromosome are calculated (L39-47), all information combined in a list (L48) and returned to the user (L49).

The `show` method is used to print an object to the `R` console (an example is shown in Figure 3.4). Unspecified, the whole object would be printed, which is not useful for the large `arrayData` objects. Therefore the `show` method was adapted to print a series of useful information about the object. For each dataset the name and normalisation status is shown along with the total number of probes in the dataset. All of the data within the object can still be accessed by a user if required, using the dollar ($) notation.

```
> abf1
An "arrayData" object containing:
         Abf1_0m1.txt
                Normalisation procedures: raw
         Abf1_0m2.txt
                Normalisation procedures: raw
         Abf1_0m3.txt
                Normalisation procedures: raw
         Abf1_30m1.txt
                Normalisation procedures: raw
         Abf1_30m2.txt
                Normalisation procedures: raw
         Abf1_30m3.txt
                Normalisation procedures: raw
         Abf1_Un1.txt
                Normalisation procedures: raw
         Abf1_Un2.txt
                Normalisation procedures: raw
         Abf1_Un3.txt
                Normalisation procedures: raw
Number of probes:
         41775
```

Figure 3.4: Output of `arrayData` `show` method: Exemplified with Abf1 binding datasets (see Chapter 7). The names of the datafiles, their normalisation states and the total number of probes in the dataset are shown.

The `summary` method provides a summary of the data in an object (an example is shown in Figure 3.5). This returns more information than the `show` command. Four sets of information are gathered into a list. By default this is printed to the `R` console, but can be manipulated to access the data within it. The first slot contains a summary of the object, showing the grid name and the number of probes and datasets. The second shows the names of the annotations, which will show if any additional columns are present. The third shows the names of all of the data files in the object. The last shows statistics on the probe coordinates and ratios, showing, for each chromosome, a count of the number of probes, the coordinate range, the ratio range and the number of NA values present.

3.2.6 Plotting data

As well as displaying data on the `R` console, several functions have been written to plot `arrayData` object ratios in various ways. This allows patterns or other aspects of data to be visualised and identified, which may not be possible from viewing the data alone.

3.2.6.1 Genome plots

Genome plots allow `arrayData` object ratios to be plotted against their respective genomic positions (Script 3.13). This creates a graph with genome position on the x axis and ratio values on the y axis. This plots either data over a specified range or the whole dataset, which automatically creates a new PDF file to store the plots.

Plots are produced as several sub-plots across rows on a page. The number of rows per page and the length of the region in each of these can be set by the user. The default shows the first chromosome over four rows. Multiple sets of `arrayData` ratios can be plotted on the same page as individual sub-plots, allowing data from different regions or different datasets to be plotted together. The function has the following arguments:

object An `arrayData` object to be plotted (no default). Each dataset is plotted as a separate line.

```
> summary(abf1)
Summary of arrayData object
$Summary

Grid     014810
Probes    41775
Datasets      9

$Annotations
 1
 "probeName"

$Files
 1              2              3              4              5              6
 "Abf1_0m1.txt" "Abf1_0m2.txt" "Abf1_0m3.txt" "Abf1_30m1.txt" "Abf1_30m2.txt" "Abf1_30m3.txt"
 7              8              9
 "Abf1_Un1.txt" "Abf1_Un2.txt" "Abf1_Un3.txt"

$Chromosomes
   Number of probes Lowest coordinate Highest coordinate Lowest ratio Highest ratio NA values
1               791                48              230115    -1.488125      5.283004         0
2              2794              5832              809934    -4.714263      5.978113         6
3              1088              1147              315561    -3.739050      5.578150        12
4              5265              1700             1525352    -1.528943      6.141873         0
5              1982              4062              570332    -1.743521      5.510419         0
6               928              1415              269826    -1.594912      6.519207         0
7              3749                41             1084229    -1.573475      5.919817         1
8              1934              2263              556838    -1.514610      5.768460         0
9              1512             19615              438728    -1.483459      5.813902         0
10             2562             19561              728073    -1.553156      6.128066         0
11             2290                69              666141    -1.517389      6.186058         0
12             3705              5818             1071675    -1.601188      6.024438         0
13             3177              8447              924291    -1.557526      6.040292         0
14             2695              6866              781549    -1.713444      6.193896         0
15             3750               195             1081635    -1.604709      5.794888         4
16             3258             13168              942842    -1.663635      5.905157         0
17              295                49               85778   -11.967250      5.834379       200
```

Figure 3.5: Output of `arrayData summary` method: Exemplified with Abf1 binding datasets (see Chapter 7). The first part (Summary) show the grid name, number of probes and number of datasets. The second part (Annotations) show the names of annotation data. The third part (Files) shows the names of the datasets. The fourth part (Chromosomes) shows probe and $\log_2$ ratio information for each chromosome.

annotationData A `genomeAnnotation` object used to plot annotations along the genome (no default).

chr A numeric vector specifying the chromosome number of data to plot when the whole genome is not being plotted (default 1).

from A numeric vector specifying the position to start plotting when the whole genome is not being plotted (defaults to the minimum value of the chromosome, rounded down to the nearest thousand).

to A numeric vector specifying the position to stop plotting when the whole genome is not being plotted (defaults to the maximum value of the specified chromosome, rounded upwards to the nearest thousand).

rows A numeric vector specifying the number of sub-plots to create on each page (default 4).

size A numeric vector specifying the length of the region to plot in each sub-plot (default 100,000; changes to the total plot region ("to" coordinate minus "from" coordinate) divided by the number of rows to plot when the whole genome is not being plotted).

ylab A character vector specifying the y-axis label (default "log2 binding").

wholeGenome A logical value indicating whether or not to plot the whole genome (default FALSE).

fileName Character vector specifying the name of the PDF file to create when "wholeGenome" is set to TRUE (default "plot.pdf").

paper Character vector specifying the page size of the PDF file to create when "wholeGenome" is set to TRUE (default "a4").

width Numeric vector specifying the plot width of the PDF file to create when "wholeGenome" is set to TRUE (default 7).

height Numeric vector specifying the plot height of the PDF file to create when "wholeGenome" is set to TRUE (default 15).

constantMinMax A logical value indicating whether or not to maintain the same y-axis limits on all subplots (default TRUE).

ylim A numeric vector specifying the limits of the y-axis (defaults to the overall range of the data if "constantMinMax" is TRUE, otherwise is calculated from the values of each subplot).

geneColour A character or numeric vector specifying the colour to fill boxes

representing ORFs, passed to the `genomeAnnotation` plot function (default "yellow").

geneBorder A character or numeric vector specifying the colour of the border of boxes representing ORFs, passed to the `genomeAnnotation` plot function (default "orange").

cols Character or numeric vector specifying the colours to plot each line for each dataset (defaults to standard `R` colours).

alpha A numeric vector specifying the alpha (transparency) value for each line (no default).

highlightProbes A matrix containing the names of probes to highlight, passed to the `genomeAnnotation` plot function (no default).

highlightRegions A matrix containing coordinates of regions to highlight, passed to the `genomeAnnotation` plot function (no default).

geneNames A logical value indicating whether to print gene names on the plots, passed to the `genomeAnnotation` plot function (default TRUE).

muti A 5 column matrix specifying the details for plotting multiple datasets or regions on different rows of the same graph (no default). Contains chromosome numbers, start and end coordinates and `arrayData` dataset number ranges. All datasets to be plotted are specified as the "object" argument and the matrix specifies the data for each plot.

geneNameCutoff A numeric vector specifying the length of a gene name over which the name is printed in a smaller font size (default 1).

lab.adjust A numeric vector to adjust the position of plot labels when using "multi" (no default).

type A character vector specifying the plot type to create (default "l").

Script 3.13: plot arrayData: script to plot arrayData object ratios values against their coordinates. Each set of ratios is displayed as a separate line.

```
1 ## plot (arrayData) function ##
2 ## arguments: object (an arrayData object), annotationData (a genomeData
      object), chr (chromosome to plot), from (coordinate to plot from), to (
      coordinate to plot to), size (length of each plot region), rows (number
      of rows per page), ylab (y acis label), wholeGenome (TRUE/FALSE),
      constantMinMax (TRUE/FALSE), ylim (y axis limits), geneColour (colour of
       gene boxes), geneBorder (colour of gene box borders), cols (colours of
      lines), geneValues (values to plot), fileName (name of PDF file), paper
      (PDF paper size), width (PDF plot width), height (PDF plot height),
```

```
    alpha (colour transparency), highlightProbes (probes to highlight),
    highlightRegions (coordinates of regions to highlight), geneNames (names
     of genes to highlight), multi (matrix specifying multiple data to plot)
    , geneNameCutoff (small font limit), lab.adjust, type (type of plot)
plot.arrayData<-function(object,annotationData,chr=1,from=0,to=240000,size
    =60000,rows=4,ylab="log2 Binding",wholeGenome=F,constantMinMax=T,ylim,
    geneColour="yellow",geneBorder="grey",cols,geneValues,filename="plot.pdf
    ",paper="a4r",width=15,height=7,alpha,highlightProbes,highlightRegions,
    geneNames=TRUE,multi,geneNameCutoff=1,lab.adjust=10,type="l",...) { #
    define function
  on.exit(layout(1)) #reset plot layout on exit
  if(!missing(multi)) { #multi is provided
    if(!is.matrix(multi)) stop("multi must be a matrix",call.=F) #check
        format
    rows<-nrow(multi) #set row value as number in multi
    wholeGenome<-FALSE #set not to plot whole genome
  }else{
    if(!missing(from) & !missing(to)) if (from > to) stop("\'from\' must be
        greater than \'to\'") #check provided coordinates
  }
  maxYValue<-minYValue<-F #set FALSE max and min Y values
  if(missing(alpha)) alpha<-255 #define alpha value
  if (missing(cols)) { #no colours provided
    cols<-1:nrow(object) #define colours
  }
  if (!missing(geneValues)) { #gene values provided
    if (constantMinMax) { #constant min/max values required
      geneValues<-rbind(c(NA,max(as.numeric(geneValues[,2]),na.rm=T)),
          geneValues) #find maximum gene value
      geneValues[2:nrow(geneValues),2]<-(as.numeric(geneValues[2:nrow(
          geneValues),2])/max(as.numeric(geneValues[2:nrow(geneValues),2]),
          na.rm=T))*geneValues[1,2] #scale gene values to log2 ratios
    }
  }
  if(!missing(highlightRegions)) highlightRegions<-matrix(ncol=3,as.numeric(
      highlightRegions)) #format highlightRegions
  if(wholeGenome) { #the whole genome is to be plotted
    plot.all<-object #get all data
    on.exit(dev.off()) #shut down graphics when function exits
    pdf(filename,paper=paper,width=width,height=height) #create PDF to store
        whole geneome plot
  }else{ #the whole genome is not to be plotted
    op<-par(no.readonly = TRUE) #get current par
    on.exit(par(op)) #reset par on exit
    if(!missing(multi)) { #multi is provided
      plot.all<-object #get all data
    }else{ #multi is not provided
      if(missing(from)) {
        from<-floor(object$coordinates[object$coordinates[,1] == chr,2][1]/
            1000)*1000 #calculate lower plot boundary
        to<-ceiling(object$coordinates[object$coordinates[,1] == chr,3][
            length(which(object$coordinates[,1] == chr))]/1000)*1000 #
            calculate upper plot boundary
      }
      if(missing(size)) size<-ceiling(((to-from)/rows)/1000)*1000 #calculate
          size
      region<-which(object$coordinates[,1] == chr & object$coordinates[,2]
          >= from & object$coordinates[,3] <= to) #find data to plot
      if(length(region) == 0) stop("No data to plot", call.=F) #stop if no
          data in range
      r1<-min(region) #get lowest data point
      r2<-max(region) #get highest data point
```

```
43      if(r1 > 1) if(object$coordinates[r1,1] == object$coordinates[(r1-1)
            ,1]) r1<-(r1-1) #reduce lowest data point if same chromosome
44      if(r2 < nrow(object)) if(object$coordinates[r2,1] == object$
            coordinates[(r2+1),1]) r2<-(r2+1) #increase highest data point if
            same chromosome
45      plot.all<-object[r1:r2,] #get data to plot
46    }
47    if(nrow(plot.all) == 0) stop("No data to plot", call.=F) #stop if no
        data in range
48  }
49  plan<-matrix(ncol=3,nrow=0) #create matrix to store plot details
50  allChrs<-unique(plot.all$coordinates[,1]) #get unique chromosomes
51  if (length(allChrs) == 0) allChrs<-chr #get chromosome number
52  for (currentChr in allChrs) { #loop through chomosomes
53    min.value<-max.value<-1
54    if (wholeGenome) { #the whole genome is to be plotted
55      from<-0 #set start point
56      to<-max(plot.all$coordinates[which(plot.all$coordinates[,1] ==
            currentChr),3]) #set end point
57    }
58    if(!missing(multi)) { #multi provided
59      plan<-multi #get plan from multi
60    }else{ #multi not provided
61      plan.froms<-seq(from,to,size) #get all from points for plot
62      plan.froms<-plan.froms[!plan.froms==to] #remove from = to point
63      plan.tos<-plan.froms+size #get all to points for plot
64      plan.chrs<-rep(currentChr,length(plan.froms)) #get chromosome numbers
65      plan.chr<-cbind(plan.chrs,plan.froms,plan.tos) #join all together
66      plan<-rbind(plan,plan.chr) #add to plan
67    }
68  }
69  if (constantMinMax) { #constant min/max required
70    minYValue<-min(plot.all$ratios,na.rm=T) #get min ratio
71    maxYValue<-max(plot.all$ratios,na.rm=T) #get max ratio
72  }
73  if (!missing(ylim)) { #ylim is defined
74    minYValue<-ylim[1] #set min value
75    maxYValue<-ylim[2] #set max value
76  }
77  mat<-matrix(ncol=3,nrow=rows) #initialise matrix for layout
78  column<-1 #set column to 1
79  for (m in c(1,2,1)) { #loop to create layout
80    mat[,column]<-seq(m,by=2,length.out=rows) #get layout values
81    column<-column+1 #increase column by 1
82  }
83  mat<-matrix(t(mat),ncol=1,byrow=F) #transpose mat
84  layout(mat,height=c(rep(c(0,0.75,0.25),rows))) #define layout
85  for (p in 1:nrow(plan)) { #loop through rows of plan
86    if(!missing(multi)) { #multi provided
87      size<-multi[p,3]-multi[p,2] #calculate size based on multi
88    }
89    whichData<-which(plot.all$coordinates[,1] == plan[p,1] & plot.all$
          coordinates[,2] >= plan[p,2] & plot.all$coordinates[,3] <= plan[p
          ,3]) #get data
90    if (!missing(annotationData)) currentAnnotationData<-annotationData[
          which(annotationData$coordinates[,1] == plan[p,1] & annotationData$
          coordinates[,3] >= plan[p,2] & annotationData$coordinates[,2] <=
          plan[p,3])] else currentAnnotationData<-new("genomeAnnotation",list(
          coordinates=matrix(nrow=0,ncol=1),annotations="",dataset="")) #get
          annotation data for region or set as empty
91    if(length(whichData) <= 1) { #no/one probes in region
92      less<-plot.all[plot.all$coordinates[,1] == plan[p,1] & plot.all$
```

```
          coordinates[,3] <= plan[p,3],] #find probes below plot region
93      greater<-plot.all[plot.all$coordinates[,1] == plan[p,1] & plot.all$
          coordinates[,2] >= plan[p,2],] #find probes above plot region
94      if (nrow(less) == 0) less<-plot.all[1,] #set lowest point if not found
95      if (nrow(greater) == 0) greater<-plot.all[nrow(plot.all),] #set
          highest point if not found
96      whichData<-c(which(plot.all$annotations[,1] == less$annotations[which.
          min(plan[p,2] - less$coordinates[,3]),1]),which(plot.all$
          annotations[,1] == greater$annotations[which.min(greater$
          coordinates[,2] - plan[p,3]),1])) #define which data
97    }
98      min.value<-min(whichData) #get min probe
99      max.value<-max(whichData) #get max probe
100     if (min.value > 1) { #first probe is above 1
101       if (plot.all$coordinates[min.value-1,1] == plot.all$coordinates[min.
            value,1]) { #previous probe on same chromosome
102         min.value<-min.value-1 #include previous value
103       }
104     }
105     if (max.value < nrow(plot.all)) { #last probe is before the end probe
106       if (plot.all$coordinates[max.value+1,1] == plot.all$coordinates[max.
            value,1]) { #next probe on same chromosome
107         max.value<-max.value+1 #include next probe
108       }
109     }
110     if(!missing(multi)) { #multi is provided
111       plot.current<-plot.all[plot.all$coordinates[,1] == multi[p,1] & plot
            .all$coordinates[,2] >= multi[p,2] & plot.all$coordinates[,3] <=
             multi[p,3],multi[p,4]:multi[p,5]] #get data to plot based on
            multi
112     }else{ #multi is not provided
113       plot.current<-plot.all[min.value:max.value,] #arrayData taken from
            plan
114     }
115     if(length(whichData) > 0) probePositions<-rowMeans(matrix(ncol=2,plot.
          current$coordinates[,2:3])) #get probe middles
116     if (!missing(highlightProbes)) { #probes are to be highlighted
117       highlightProbes.current<-which(highlightProbes %in% plot.current$
            annotations[,1]) #find probes to be highlighted
118       if(length(highlightProbes.current) > 0 ) { #probes are to be
            highlighted in the current plot
119         highlightProbes.current<-rowMeans(matrix(plot.current[
              highlightProbes[highlightProbes.current],]$coordinates[,2:3],
              ncol=2)) #get positions of probes to highlight
120       }else{ #probes are not to be highlighted in the current plot
121         highlightProbes.current<-NULL #set as NULL
122       }
123     }else{ #no probes to be highlighted
124       highlightProbes.current<-NULL #set as NULL
125     }
126     if (!missing(highlightRegions)) { #ranges to be highlighted
127       highlightRegions.current<-matrix(highlightRegions[which(
            highlightRegions[,1] == plan[p,1] & highlightRegions[,2] >= plan
            [p,2] & highlightRegions[,3] <= plan[p,3]),],ncol=3) #get ranges
             to be highlighted
128     }else{ #no ranges to be highlighted
129       highlightRegions.current<-NULL #set as NULL
130     }
131     if (!constantMinMax) { #not constant min/max values
132       minYValue<-min(plot.current$ratios,na.rm=T) #get min ratio value
133       maxYValue<-max(plot.current$ratios,na.rm=T) #get max ratio value
134     }else{ #constant min/max values
```

```
135         minYValue<-min(plot.all$ratios,na.rm=T) #get min ratio value
136         maxYValue<-max(plot.all$ratios,na.rm=T) #get max ratio value
137       }
138       if (!missing(ylim)) { #ylim is defined
139         minYValue<-ylim[1] #set min value
140         maxYValue<-ylim[2] #set max value
141       }
142     par(mar=c(1,5,1,2),bty="n") #set plot for annotation
143     plot(currentAnnotationData,plan[p,1],plan[p,2],plan[p,3],probePositions,
            geneColour,geneBorder,constantMinMax,geneValues,highlightProbes.
            current,highlightRegions.current,geneNames,geneNameCutoff=
            geneNameCutoff) #plot annotation data
144     par(mar=c(1,5,2,2),bty="n",mgp=c(1.75,1,0)) #set plot for ratios
145     plot(1,1,type="n",xlim=c(plan[p,2],plan[p,3]),ylim=c(minYValue,maxYValue
            ),xaxs="i",yaxs="i",xlab="",ylab=ylab,xaxt="n",yaxt="n") #initialise
             plot
146     abline(h=0,lty=2,col="lightgrey") #add zero line
147     for (a in 1:ncol(plot.current)) { #loop through datasets
148       startWarn<-as.numeric(options("warn")); on.exit(options(warn=startWarn
              )) #get current warning state and maintain on exit
149       options(warn=-1) #don't warn about NA values
150       if (!is.na(cols[a]) & !is.null(plot.current)) { #data is to be plotted
151         if (length(whichData) > 1) { #there is data to plot in the range
152           points(approx(rowMeans(matrix(plot.current$coordinates[,2:3],ncol
                =2)),plot.current$ratios[,a],xout=rowMeans(matrix(plot.current
                $coordinates[,2:3],ncol=2))),col=rgb(matrix(ncol=3,col2rgb(
                cols[a])),max=255,alpha=alpha),type=type,...) #plot data
153         }else{ #there is no data to plot in the range
154           points(sum(matrix(plot.current$coordinates[,2:3],ncol=2))/2,plot.
                current$ratios[,a],col=rgb(matrix(ncol=3,col2rgb(cols[a])),max
                =255,alpha=alpha),pch=19,cex=0.5) #plot extended probes
155         }
156       }
157       options(warn=0) #reset warnings
158     }
159     if(!missing(multi)) { #multi is provided
160       mtext(LETTERS[p],side=3,at=plan[p,2]-(size/lab.adjust),las=1.75,cex
              =1.5) #print letters if multiple plots
161     }
162     axis(2,tcl=-0.3,padj=0.8) #format axis
163     axis(3,tcl=-0.3,padj=1,line=0.5) #format axis
164   }
165 }
```

The function first ensures the plot is reset when it exits (L4). Then checks are performed and a series of parameters set and configured, based on the arguments provided, in order to correctly plot the data (L5–23). If the whole genome is not to be plotted the specified subset is extracted, extending the data by one upwards and downwards if on the same chromosome (L43–44), otherwise all data is kept and a PDF initiated (L24–48). A "plan" is created, detailing the data to plot on each row of the display (L49–68). y-axis limits are set (L69–76) and the plot layout defined (L77–84). A loop for each row of the plan is initiated (L85). The plot size is created dynamically with

"multi" (L86–88). Data for the current region is identified (L89) along with the annotation data, if required (L90). If no or one probes are present in the range to be plotted (L91), this is redefined to include the previous and next probes (L92–97).

The data range is then found (L98–99). Data extending out from the plot region are specified (L100–109) which allows the plots to extend to the edges of their regions. The data to be plotted is extracted (L110–114) and probe positions calculated (L115). Probes (L116–125) and ranges to be highlighted (L126–130) are identified. Plot y-axis limits are defined (L131–141). Plot margins are defined (L142) and the `genomeAnnotation` data plot is created (L143), showing ORF information where provided. Plot margins are defined (L144) and the `arrayData` plot is initialised (L145–146). Data for each dataset is plotted in a loop (L147–158) with warnings for NA values disabled. Letters are printed at the edge of "multi" plots (L159–161). Finally, axes are added to the plots (L162–163).

A `genomeAnnotation` object can also be specified, which adds extra information to the plots. This is carried out by the genomeAnnotation plot method (Script 3.14). This plots a line representing the genome, showing all ORFs over the given region as boxes indicating the direction of transcription. Gene names, taken from the gene name column of the `genomeAnnotation` annotations matrix, can optionally be displayed over each ORF. Additionally, this function indicates the position of each probe in the genome with a grey dot. Particular probes can be highlighted, with the addition of a red cross. Regions can also be highlighted, with the addition of a coloured box covering the region of the genome. The function has the following arguments, all of which are specified by the `arrayData` plot function and so none have defaults.

object The `genomeAnnotation` object to be plotted.
chr A numeric vector specifying the current chromosome number.
from A numeric vector specifying the current start coordinate.
to A numeric vector specifying the current end coordinate.
arrayProbes A numeric vector specifying the positions of probes to plot.

geneColour A character or numeric vector specifying the colour to fill boxes representing ORFs.

geneBorder A character or numeric vector specifying the colour of the border of boxes representing ORFs.

constantMinMax A logical value indicating whether or not to maintain the same y-axis limits on all subplots.

highlightProbes A matrix containing the names of probes to highlight on the plots.

highlightRanges A matrix containing coordinates of regions to highlight on the plots.

geneNames A logical value indicating whether or not to print gene names on the plots.

geneNameCutoff A numeric vector specifying the length of a gene name over which the name not printed.

Script 3.14: plot genomeAnnotation: script to plot genomeAnnotation object annotations within the arrayData plot function. ORFs and probe positions are plotted.

```
1  ## plot (genomeAnnotation) function ##
2  ## arguments: object (a genomeAnnotation object), chr (chromosome number),
       from (from coordinate), to (to coordinate), arrayProbes (probe positions
       ), geneColour (colour of gene boxes), geneBorder (colour of gene box
       borders), constantMinMax (whether or not constantMinMax), geneValues (
       values to plot gene bars), highlightProbes (probes to highlight),
       highlightRanges (ranges to highlight), geneNames (names of genes),
       geneNameCutoff (gene name length cutoff)
3  plot.genomeAnnotation<-function(object,chr,from,to,arrayProbes,geneColour="
       yellow",geneBorder="orange",constantMinMax,geneValues,highlightProbes,
       highlightRanges,geneNames=TRUE,geneNameCutoff=0) { #define function
4    plot(1,1,type="n",xlim=c(from,to),ylim=c(-1,8),xlab="",ylab="",xaxt="n",
         yaxt="n",xaxs="i") #initialise plot
5    grid(nx=NULL,ny=0,lty=2) #add grid
6    if(nrow(object) > 0) { #if genomeAnnotation is provided
7      square<-((to-from)/100) #calculate size of arrow section
8      y<-c(0.53,0.28,0.03,0.03,0.53) #set y values for polygons
9      textCentre<-0.2 #set the centre point for gene names
10     genes.U.labels<-genes.L.labels<-matrix(ncol=4,nrow=0) #initialise
           matrices to store labels for upper and lower strands
11     genes.U<-object[which(object[[2]][,3] == " 1")] #get upper strand genes
           in range
12     genes.L<-object[which(object[[2]][,3] == "-1")] #get lower strand genes
           in range
13     if (nrow(genes.U) > 0) { #if genes are present on the upper strand
14     genes.U.labels<-matrix(ncol=3,nrow=nrow(genes.U)) #initialise matrix to
           store labels
15       for (n in 1:nrow(genes.U)) { #loop through genes
```

```
16          left<-genes.U[[1]][n,2] #get left hand (start) value
17          right<-genes.U[[1]][n,3]-square #get box end value
18          point<-genes.U[[1]][n,3] #get right hand (end) value
19          if (right<left) {right<-left} #set box end as start if it goes
                beyond the start
20          x<-c(right,point,right,left,left) #set x values for polygon
21          polygon(x,y,col=geneColour,bor=geneBorder) #plot the current gene
22          genes.U.labels[n,1]<-(left+point)/2 #get the middle of the gene
23          genes.U.labels[n,2]<-0.6 #set the font size
24          if (nchar(genes.U[[2]][n,1]) > 6) genes.U.labels[n,2]<-0.4 #reduce
                the font size for long gene names
25          if(point - left > geneNameCutoff) genes.U.labels[n,3]<-genes.U[[2]][
                n,1] #set the gene name
26        }
27      }
28      if (nrow(genes.L) > 0) { #if genes are present on the lower strand
29        genes.L.labels<-matrix(ncol=3,nrow=nrow(genes.L)) #initialise matrix
              to store labels
30        for (n in 1:nrow(genes.L)) { #loop through genes
31          right<-genes.L[[1]][n,3] #get right hand (start) value
32          left<-genes.L[[1]][n,2]+square #get box end value
33          point<-genes.L[[1]][n,2] #get left hand (end) value
34          if (right<left) {left<-right} #set box end as start if it goes
                beyond the start
35          x<-c(left,point,left,right,right) #set x values for polygon
36          polygon(x,-y,col=geneColour,bor=geneBorder) #plot the current gene
37          genes.L.labels[n,1]<-(right+point)/2 #get the middle of the gene
38          genes.L.labels[n,2]<-0.6 #set the font size
39          if (nchar(genes.L[[2]][n,1]) > 6) genes.L.labels[n,2]<-0.4 #reduce
                the font size for long gene names
40          if(right - point > geneNameCutoff) genes.L.labels[n,3]<-genes.L
                [[2]][n,1] #set the gene name
41        }
42      }
43      if(geneNames) { #show gene names
44        if (nrow(genes.L.labels) > 0) { #if there are genes on the lower
              strand to be labelled
45          text(as.numeric(genes.L.labels[,1]),-0.2,labels=genes.L.labels[,3],
                cex=as.numeric(genes.L.labels[,2]),srt=-30,pos=4,offset=0) #add
                the gene name text
46        }
47        if (nrow(genes.U.labels) > 0) { #if there are genes in the upper
              strand to be labelled
48          text(as.numeric(genes.U.labels[,1]),0.2,labels=genes.U.labels[,3],
                cex=as.numeric(genes.U.labels[,2]),srt=-30,pos=2,offset=0) #add
                the gene name text
49        }
50      }
51      if (!missing(geneValues)) { #geneValues are provided
52        labels<-at<-pretty(0:as.numeric(geneValues[1,2])) #define labels
53        axis(4,tcl=-0.3,padj=-0.8,labels=labels,at=labels*(8/as.numeric(
              geneValues[1,2]))) #add axis
54        mtext("Gene Value", side=4, line=2,cex=0.7) #label axis
55        currentGeneValues<-matrix(geneValues[which(geneValues[,1] %in% object
              [[2]][,1]),],ncol=2) #get gene values
56        if (!constantMinMax) {   #not constant min max
57          currentGeneValues<-rbind(c(NA,max(as.numeric(currentGeneValues[,2]),
                na.rm=T)),currentGeneValues) #get values
58          currentGeneValues[2:nrow(currentGeneValues),2]<-(as.numeric(
                currentGeneValues[2:nrow(currentGeneValues),2])/max(as.numeric(
                currentGeneValues[2:nrow(currentGeneValues),2]),na.rm=T))*8
59          currentGeneValues<-currentGeneValues[2:nrow(currentGeneValues),]
```

```
60        }
61        if(length(currentGeneValues != 0)) { #some values are present
62          for (g in 1:nrow(currentGeneValues)) { #loop through values
63            rect(object[[1]][which(object[[2]][,1] == currentGeneValues[g,1])
                  ,2],0,object[[1]][which(object[[2]][,1] == currentGeneValues[g
                  ,1]),3],as.numeric(currentGeneValues[g,2]),col=rgb
                  (0.3,0.3,0.3,alpha=0.5),bor=NA) #add rectangles
64          }
65        }
66      }
67    }
68    if (!is.null(highlightRanges)) rect(highlightRanges[,2],rep(-0.25,nrow(
          highlightRanges)),highlightRanges[,3],rep(0.25,nrow(highlightRanges)),
          bor=0,col=rgb(255,192,203,max=255,alpha=200)) #highlight ranges
69    abline(h=0,col="grey") #add a horizintal grey line at h=0
70    if (!missing(arrayProbes)) points(arrayProbes,rep(0,length(arrayProbes)),
          pch=19,col="darkgrey",cex=0.5) #add a dot at each probe position
71    if (!is.null(highlightProbes)) points(highlightProbes,rep(0,length(
          highlightProbes)),pch=4,col="red") #add a dot at each highlighted
          probe position
72    mtext(paste("Chr.",chr),side=2,las=1,line=1.75,at=0,cex=0.7) #label the
          chromosome number
73  }
```

The `arrayData` plot function determines the `genomeAnnotation` data for the region being plotted and passes it to this function. A plot of the correct size is first initialised (L4) and a grid added (L5). If genome annotation data is provided (L6) Parameters for the ORF polygons are determined (L7–9) and matrices to store their labels created (L10). ORFs on the upper and lower strands are separated (L11–12). For ORFs on the upper strand (L13) the labels are extracted (L14). A loop is initialised for each (L15) where the polygon parameters are determined (L16–20), the polygon plotted (L21), and the gene name text parameters determined (L22–25). The same procedure is repeated for ORFs on the lower strand (L28–42). Gene names are added if required (L43–50). Gene values are used to add bar plots if required (L51–66). Ranges to be highlighted are drawn (L68), a central line added (L49) and probes drawn (L70) and highlighted (L71) if required. Finally the chromosome name is printed (L72).

Figure 3.6 shows an example of the output of the two plot functions, showing data plotted along a short section of chromosome 1 with examples of the additional information that can be plotted. The `arrayData plot` function creates the plot of the the data specified in the `arrayData` object along with the two sets of axises. The `genomeAnnotation plot` creates the representations of ORF and probe positions, highlights specified probes and

regions

The functions are used to generate more plots shown in the following chapters.

3.2.6.2 Histograms, density and Q-Q plots

Histograms are graphical representations of the distribution of continuous data. They consist of a series of bars, the areas of which represent the frequencies of data points falling into a set of discrete bins. They are useful for visualising the distribution of a set of data and can be used to estimate an underlying probability density function. The existing `R` histogram method was adapted to produce histograms from `arrayData` ratios (Script 3.15), using the following argument:

`x` An `arrayData` object to create the histogram from (no default). Multiple histograms are not easy to distinguish on the same graph and so only values from the first dataset are used.

Script 3.15: arrayData histogram: script to plot a histogram of ratios from an `arrayData` object.

```
1 ## hist (arrayDat) function ##
2 ## arguments: x (an arrayData object)
3 setMethod("hist", "arrayData", function(x,...) { #define function
4   hist(x$ratios[,1],...)
5 }
6 )
```

The function plots a standard histogram from the first column of the `arrayData` ratios (L4).

An example histogram is shown in Figure 3.7, created from a normalised Abf1 binding dataset, showing the skew of the data as a result of the protein binding.

An alternative to the histogram is the kernel density plot. This uses kernel density estimation to estimate the probability density function of a set of data. Various kernels can be used to create this estimation. Rather than binning data and displaying bars, as in a histogram, kernel density

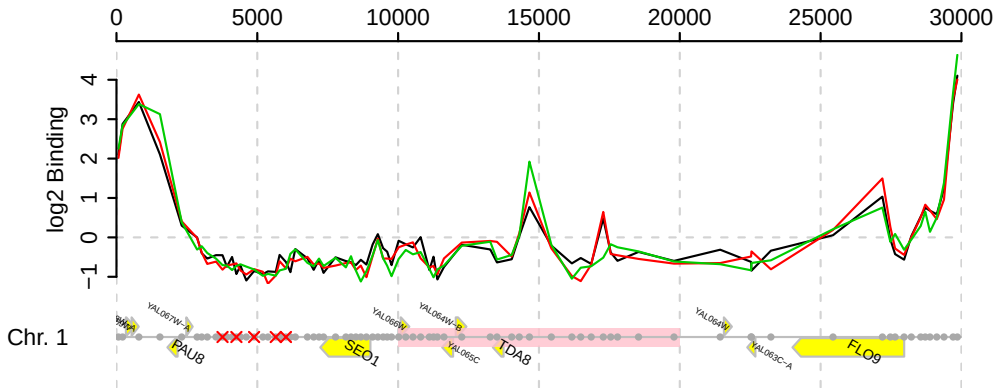


Figure 3.6: Output of `arrayData` `plot` method: Exemplified with three Abf1 binding datasets (see Chapter 7) plotted over a short section of chromosome 1. The genome coordinates are plotted along the top of the graph and the dataset units on the left. Genome annotations are shown at the bottom with the chromosome number printed to the left hand side. Each of the three datasets is shown as a separate line, in this case using the default different colours. Genes are represented with yellow shapes, with the arrow representing the direction of transcription. Gene names are printed, those with longer names in a smaller font size. Probe positions are represented with grey circles and some have been highlighted, shown with red crosses. A region has also been highlighted, shown with a pink box.

estimation produces a curve representing the distribution of the data. This is described further in Section 4.2.4.2.

The existing `R` density and plot methods were adapted to produce density plots from `arrayData` ratios (Script 3.16). The density function produces the kernel density estimate as a density class, and the plot method for this class produces a graph showing the curve. The function plots each dataset in the provided `arrayData` object as a separate line on the same graph. All kernel density estimates are first calculated, in order to determine the x and y-axis ranges for all data. This plot is then initialised, and each line plotted. By default each line is given a different colour, which can be manually overridden.

The function also has the option to mirror values about zero to display a representation of the estimated background region (see Section 4.2.4.2). The density of all negative values, along with the positive equivalents, is added to the plot as a red dashed line. This shows a mirror image of the section of the density plot below zero. To this is added a standard normal curve, with a mean of zero and standard deviation of one, as a blue dotted line. This provides a method of visually comparing the background estimation to the normal distribution. In this case only the first `arrayData` object dataset is plotted. The function has the following arguments:

x An `arrayData` object to create the density plots from (no default).

cols Character or numeric vector specifying the colours to plot each line for each dataset (defaults to standard `R` colours).

mirror Logical vector indicating whether or not to display mirrored estimated background values and the standard normal distribution for the first dataset (default FALSE).

Script 3.16: arrayData density plot: script to plot a density plot of ratios from an `arrayData` object. Each set of ratios is plotted as a separate line and the axes are scaled accordingly. The densities of negative data mirrored about zero and the standard normal distribution can also be plotted to visualise the estimated background population.

```
1  ## density function ##
2  ## arguments: x (an arrayData object), cols (colours to plot lines), mirror
       (whether or not to show mirror information)
3  setMethod("density", "arrayData", function(x,cols,mirror=F,xlim,ylim,...) {
       #define function
4    if (missing(cols)) cols<-1:ncol(x) #define colours if missing
5    if (mirror) cols<-rep(1,ncol(x)) #redefine colours if mirroring
6    xs<-ys<-list() #initialise lists to store values
7    for (n in 1:ncol(x)) { #loop through datasets
8      d<-density(x$ratios[,n],na.rm=T) #get kernel densities
9      xs<-c(xs,list(d$x)) #get x values
10     ys<-c(ys,list(d$y)) #get y values
11   }
12   if(missing(xlim)) xlim<-range(xs)
13   if(missing(ylim)) ylim<-range(ys)
14   plot(1,1,type="n",xlim=xlim,ylim=ylim,...) #initialise plot
15   for (n in 1:ncol(x)) { #loop through datasets
16     points(xs[[n]],ys[[n]],type="l",col=cols[n],...) #add points
17     if (mirror) { #data is to be mirrored
18       points(c(xs[[n]][xs[[n]]<=0],abs(xs[[n]][xs[[n]]<=0][length(which(xs[[
             n]]<=0)):1])),c(ys[[n]][xs[[n]]<=0],ys[[n]][xs[[n]]<=0][length(
             which(xs[[n]]<=0)):1]),type="l",lty=2,col="red",...) #add mirrored
              points
19       norm<-dnorm(c(xs[[n]][xs[[n]]<=0],abs(xs[[n]][xs[[n]]<=0][length(which
             (xs[[n]]<=0)):1]))) #get normal distribution
20       norm<-norm * max(ys[[n]][xs[[n]]<=0])/max(norm) #scale norm
21       points(c(xs[[n]][xs[[n]]<=0],abs(xs[[n]][xs[[n]]<=0][length(which(xs[[
             n]]<=0)):1])),norm,col="blue",type="l",lty=3) #plot normal curve
22     }
23   }
24 }
25 )
```

The function first defines the plot colours if not specified (L4) or the data is to be mirrored (L5). Lists are created to store the x and y density values (L6) and a loop through datasets initialised (L7) where the kernel densities are calculated (L8–10). Axis limits are set if required (L12–13). A plot is initialised with limits based on the density values (L14). A second loop of datasets is initialised (L15) and the density lines plotted (L16). If data are to be mirrored the mirror and standard normal representation line are added (L17–22).

An example density curve is shown in Figure 3.7, created from a normalised Abf1 binding dataset, showing the skew of the data as a result of the protein binding.

Quantile-quantile (Q-Q) plots are a means of graphically comparing two distributions. Quantiles of the two distributions are calculated and displayed as points, each representing one quantile from the first distribution (on the y axis) and the same quantile in the second distribution (on the x axis). A QQ-line can be added to the plot to represent equal distributions. If the points fall on this line the two distributions are equal. The `R` `qqnorm` and `qqplot` methods were adapted to display `arrayData` object ratio quantiles against normal distribution quantiles or another set of `arrayData` ratios respectively (Script 3.17). The `qqline` method was adapted to show the line representing identical distributions. This plots only the first dataset in the provided `arrayData` object(s).

The same mirror option as the density function is also included in the `qqnorm` and `qqline` functions, providing another method of comparing the estimated background distribution to the normal distribution.

The functions have the following arguments:

y An `arrayData` object specifying the first set of values to create the Q-Q norm, Q-Q plot or Q-Q line from (no default).

mirror Logical vector indicating whether or not to display mirrored estimated background value Q-Q norm or Q-Q line (default FALSE).

x An `arrayData` object specifying the second set of values to create the Q-Q plot or Q-Q line from (no default; optional for Q-Q line, if left blank uses standard normal values).

Script 3.17: arrayData Q-Q plot: scripts to create Q-Q plots with `arrayData` objects. Q-Q norm plots ratios against a normal distribution, Q-Q plot plots two sets of ratios against each other and Q-Q line adds a Q-Q line to the plot. The quantiles of negative data mirrored about zero can also be plotted with Q-Q norm and Q-Q line to visualise the estimated background population.

```
1  ## qqnorm function ##
2  ## arguments: y (an arrayData object), mirror (whether or not to show mirror
       information)
3  setMethod("qqnorm", "arrayData", function(y,mirror=F,...) { #define function
4    if (!mirror) { #plot all data
5      qqnorm(y$ratios[,1],...) #create QQ plot
6    }else{ #plot mirrored data
7      qqnorm(c(y$ratios[y$ratios[,1]<0,1],abs(y$ratios[y$ratios[,1]<0,1]))
           ,...) #create mirrored data QQ plot
8    }
9  }
10 )
11 ## qqplot function ##
12 ## Arguments: x, y (both arrayData objects)
13 setMethod("qqplot", "arrayData", function(x,y,...) { #define function
14     qqplot(x$ratios[,1],y$ratios[,1],...) #create QQ plot
15 }
16 )
17 ## qqline function ##
18 ## Arguments: y (an arrayData object), mirror (whether or not to show mirror
       information)
19 setMethod("qqline", "arrayData", function(y,x,mirror=F,...) { #define
       function
20   if (missing(x)) { #single dataset
21     if (!mirror) { #plot all data
22       qqline(y$ratios[,1],...) #add QQ line
23     }else{ #plot mirrored data
24       qqline(c(y$ratios[y$ratios[,1]<0,1],abs(y$ratios[y$ratios[,1]<0,1]))
           ,...) #add mirrored data QQ line
25     }
26   }else{ #two datasets
27     qqline(x$ratios[,1],y$ratios[,1],...) #add QQ line
28   }
29 }
30 )
```

The `qqnorm` method (L3) plots either a Q-Q plot of the whole first column of `arrayData` ratios (L5) or only those values less than zero (L7), if "mirror" is TRUE, against normal quantiles. The `qqplot` method (L13) plots a Q-Q plot of the first columns of two `arrayData` objects (L14). The `qqline` method (L19) adds a Q-Q line with a single `arrayData` object (L20) against normal quantiles using the first set of ratios (L22) or only those less than zero (L24) if "mirror" is TRUE, or using two `arrayData` objects (L27).

An example normal Q-Q plot is shown in Figure 3.7, created from a normalised Abf1 binding dataset, showing the data do not follow a normal distribution. A mirrored Q-Q plot is also shown, with a mirrored Q-Q line

in red, showing these data do approximate a normal distribution.

3.2.6.3 Profile plots

"Profile plot" is the name given to a type of plot which overlays several sections of ChIP-chip data on the same graph, which gives an overview of a particular feature. The format is similar to that of the `arrayData` genome plots, except that rather than continuing the plotted data as a single line over a long distance, the data are split into defined sections and multiple lines representing these are 'piled up' on top of each other over a short distance. A trend line can be added to these to show any overall pattern in the data. The `profilePlot` function (Script 3.18) allows three types of plot to be created, plotting data over genic, intergenic and peak regions. The genic plot shows ORFs with sections of their upstream (promoter) and downstream regions, aligned so all run left-to-right across the plot. The intergenic plot shows the inverse of the genic, that is, whole intergenic regions with sections of the flanking ORFs at either side. The peak plots are centred on probes found by the `peakDetection` function (Chapter 5) and show the regions flanking these. Gene data is taken from a `genomeData` object and peak data from a `peakList` object, created by the `peakDetection` function.

Plotting a single dataset has the option to plot all individual lines of data and/or the trend line. Only trend lines are plotted when more than one dataset is provided. Trend lines are calculated from a number of averaged points, the number of which can be specified. The ends of trend lines can sometimes become distorted if there are few data points from which to calculate the average. These ends can be clipped, by specifying a cutoff based on the fraction of data present at each point to be averaged. Standard errors of the trend lines can be shown as coloured shapes. The standard error increases with standard deviation and as fewer points are used to calculate it, and so thicker shapes may indicate less reliable regions.

The data to be plotted can be specified in various ways, such as providing names or size ranges of genes to plot. ORF start/end boundaries are always aligned at the same points in genic and intergenic plots, as indicated by the

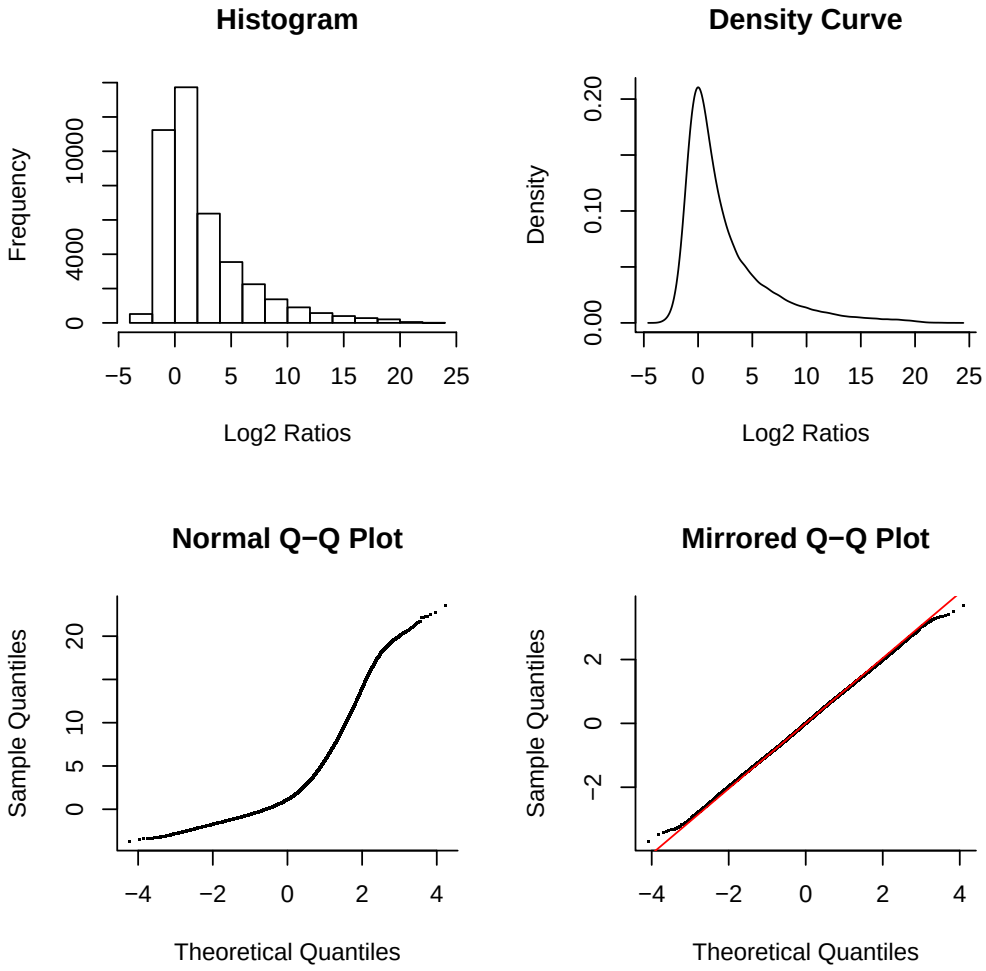


Figure 3.7: Output of `arrayData` statistical graphics: Examples of a histogram, density curve, normal Q-Q plot and a mirrored Q-Q plot showing a mirrored Q-Q line using one set of normalised Abf1 binding data.

label on the y-axis. This can be achieved by either scaling or splitting the data. Scaling can be done relative to the largest or smallest region being plotted, maintaining the shortest line and scaling all longer lines down to fit, or maintaining the longest line and scaling all shorter lines up to fit. Scaling all lines down to the shortest ensures that no regions go beyond the boundaries of the plot region. However, it can be useful to plot data in this way in certain circumstances. Splitting allows data to be split at the centres of ORFs. Each of the two resulting lines is aligned at the respective boundary. In this way all lines are shown at the same scale. The function has the following arguments:

object An `arrayData` object to be plotted (no default). Each dataset can be plotted as a separate trend line or the first dataset is used to plot all individual lines.

plotType A character vector specifying the type of plot to create; either "genic", "intergenic" or "peaks" (default "genic").

annotation A `genomeAnnotation` object, used to specify ORF positions when creating genic or intergenic plots (no default).

peakList A `peakList` object, used to specify peak positions when creating peak plots.

showAllLines A logical vector indicating whether or not to plot all individual lines from a single dataset (default TRUE; when more than one dataset is provided is set to FALSE).

allLines.col A character vector specifying the colour of the (non-trend line) lines (default `rgb(0,0,0,alpha=0.5)`).

showTrendLine A logical vector indicating whether or not to plot trend lines from all provided datasets (default TRUE; when more than one dataset is provided is reset to TRUE).

col A character or numeric vector specifying the trend line colour (defaults to standard `R` colours).

lty A character or numeric vector specifying the trend line type (default 1 (solid line)).

lwd A numeric vector specifying the trend line width (default 1).

averagePoints A numeric vector specifying the number of points at which

to calculate the trend line averages (default 50).

tidy A numeric vector specifying the minimum fraction of data points to be present to plot a trend line point (default 0).

showSEs A logical vector indicating whether or not to show standard errors as coloured shapes around trend lines (default FALSE).

range A numeric vector specifying the size range of genes or intergenic regions to include in plots (defaults 1000 and "Inf").

geneList A character vector specifying genes (from the `genomeAnnotation` object) to include in genic and intergenic plots (no default). Overrides the "range" argument.

extend A numeric vector specifying the size of the regions to plot extending from the gene/intergene ends or the probes at the centres of peaks (default 1000).

keepSmallest A logical vector specifying whether to scale data relative to the smallest or largest gene being plotted (default TRUE).

split A logical vector specifying whether to split or scale data (default FALSE).

ylim A numeric vector specifying the y-axis limits (defaults to the data range).

ylab A character vector specifying the y-axis label (default "Binding Value").

main A character vector specifying the plot title (defaults to the graph type created and the number of lines plotted or used to calculate the trend line).

labels A character vector specifying the labels for the x-axis (defaults vary depending on the plot type).

add A logical vector specifying whether to add the plotted lines to an existing plot or create a new plot (default FALSE).

Script 3.18: profilePlot: script to display profiles of arrayData object data showing genic or intergenic regions, defined by an annotationData object properties or list of genes, or peak regions, defined with a list of probes.

```
## profilePlot function ##
## arguments: object (an arrayData object), plotType (type of plot to create), annotation(a genomeAnnotation object), peakList (a list of probes), geneList (a list of genes), range (range of gene sizes to plot), extend (value to extend), showAllLines (whether to show all lines), showTrendLine (whether to show trend lines), showSEs (whether to show standard errors), averagePoints (number of trend line points), tidy (tidy value), ylim (y axis limits), ylab (y axis label), main (title), keepSmallest (whether to scale to the smallest gene), alLines.col (colour of all lines), col (colour of trend line), lty (trend line type), lwd (trend line width), add (whether to add to an existing plot), labels (x axis labels), split (whether to split the data)
profilePlot<-function(object, plotType="genic", annotation, peakList, geneList, range=c(1000,Inf), extend=1000, showAllLines=TRUE, showTrendLine=TRUE, showSEs=FALSE, averagePoints=50, tidy=0, ylim, ylab, main ,keepSmallest=TRUE,allLines.col=rgb(0,0,0,alpha=0.5),col=2,lty,lwd,add=FALSE,labels,split=F) { #define function
  if(missing(plotType)) plotType<-"genic" #plot genes if region not specified
  if (ncol(object) > 1) { #more than one dataset to plot
    showAllLines<-FALSE #do not show all lines
    showTrendLine<-TRUE #do show trend line
  }
  if(missing(ylab)) ylab<-"Height" #specify ylab if missing
  if(missing(col)) col<-2:(1+ncol(object)) #specify colours if missing
  if(missing(lty)) lty<-rep(1,ncol(object)) #specify trend line types if missing
  if(missing(lwd)) lwd<-rep(2,ncol(object)) #specify trend line widths if missing
  if(length(col) != ncol(object)) col<-2:(1+ncol(object)) #specify colours if wrong number
  if(length(lty) != ncol(object)) lty<-rep(1,ncol(object)) #specify trend line types if wrong number
  if(length(lwd) != ncol(object)) lwd<-rep(2,ncol(object)) #specify trend line widths if wrong number
  if (showSEs) lty<-lty<-rep(2,ncol(object)) #set line type if showing SEs
  if (showSEs) tidy<-0 #set tidy value if showing SEs
  p<-0 #set probe count to zero
  objectChr<-object[0,] #get empty arrayData object
  probesList<-list() #initialse list to store probes
  allRegions<-matrix(ncol=4,nrow=0) #initialise matrix to store coordinates
  orientations<-regionSizes<-numeric() #initialse vectors to store orientations and gene sizes
  for (chr in unique(annotation$coordinates[,1])) { #loop through chromosomes
    p<-p+nrow(objectChr) #increase probe count
    annotationChr<-annotation[annotation$coordinates[,1] == chr] #get current annotation
    objectChr<-object[object$coordinates[,1] == chr,] #get current arrayData
    if (nrow(objectChr) == 0) next() #skip chr if no data
    coordinates<-objectChr$coordinates #get coordinates
    chrMax<-max(coordinates[nrow(objectChr),3],annotationChr$coordinates[nrow(annotationChr),3]) #get max chr value
    if (plotType == "genic") { #genic plot
      if(missing(labels)) labels<-c(paste("-",extend,sep=""),"Promoter","ORF start","Inside","ORF end","Downstream",paste("+",extend,sep="")) #set labels if missing
```

```
32      regions<-cbind(c(1,annotationChr$coordinates[1:(nrow(annotationChr)-1)
            ,3]),annotationChr$coordinates[,2:3],c(annotationChr$coordinates
            [2:nrow(annotationChr),2],chrMax)) #get regions
33    }else if (plotType == "intergenic") { #intergenic plot
34      if(missing(labels)) labels<-c(paste("-",extend,sep=""),"ORF","ORF
            boundary","Intergenic","ORF boundary","ORF",paste("+",extend,sep="
            ")) #set labels if missing
35      regions<-cbind(c(1,annotationChr$coordinates[,2]),c(1,annotationChr$
            coordinates[,3]),c(annotationChr$coordinates[,2],chrMax),c(
            annotationChr$coordinates[,3],chrMax)) #get regions
36    }else if (plotType == "peak") { #peaks plot
37      if(missing(labels)) labels<-c(paste("-",extend,sep=""),"Peak Centres",
            paste("+",extend,sep="")) #set labels if missing
38      get<-which(objectChr$annotations[,1] %in% peakList) #peak probes
39      centres<-rowMeans(matrix(ncol=2,coordinates[get,2:3])) #peak centres
40      regions<-cbind(c(1,centres[1:(length(centres)-1)]),centres,centres,c(
            centres[2:length(centres)],chrMax)) #get regions
41    }else{ #another plotType provided
42      stop("Incorrect plotType", call.=F) #stop with error message
43    }
44    regions[,1]<-ifelse((regions[,2]-regions[,1]) > (2*extend),regions[,2]-
          extend,(regions[,1]+regions[,2])/2) #modify regions downwards
45    regions[,4]<-ifelse((regions[,4]-regions[,3]) > (2*extend),regions[,3]+
          extend,(regions[,3]+regions[,4])/2) #modify regions upwards
46    if (plotType != "peak") { #genic/intergenic plot
47      regionSize<-apply(regions[,2:3],1,diff,na.rm=T) #calculate region
            sizes
48      if(missing(geneList)) keep<-(regionSize >= range[1] & regionSize <=
            range[2]) else keep<-which(annotationChr$annotations[,1] %in%
            geneList) #find required regions
49      if(length(keep) == 0) next() #move onto next chr if no probes
            extracted
50      regions<-matrix(ncol=4,regions[keep,]) #get regions to keep
51      regionSize<-regionSize[keep] #get region sizes to keep
52    }
53    if (plotType=="genic") orientation<-as.numeric(annotationChr$annotations
          [keep,3]) else orientation<-rep(1,nrow(regions)) #get/set
          orientation values
54    probesListChr<-vector("list", nrow(regions)) #initialise list to store
          probes
55    if (nrow(regions) > 0) { #regions are found
56      for (n in 1:nrow(regions)) { #loop through regions
57        probesListChr[[n]]<-which(rowMeans(matrix(ncol=2,coordinates[,2:3]))
              >= regions[n,1] & rowMeans(matrix(ncol=2,coordinates[,2:3])) <=
              regions[n,4])+p #get probe numbers
58      }
59    }
60    probesList<-c(probesList,probesListChr) #save probes together
61    allRegions<-rbind(allRegions,regions) #save regions together
62    orientations<-c(orientations,orientation) #save orientations together
63    if (plotType != "peak") regionSizes<-c(regionSizes,regionSize) else
          regionSizes<-c(0,1)#save region sizes together
64  }
65  if(!missing(geneList)) range<-range(regionSizes) #redefne range if a
        genelist is provided
66  if(split) keepSmallest<-F #if spitting data set keepsmallest to TRUE
67  if(range[2] == Inf) range[2]<-max(regionSizes) #set max range value
68  if(range[1] == -Inf) range[1]<-min(regionSizes) #set min range value
69  if(missing(main)) main<-paste("Profile plot of ",plotType," regions\n(n =
        ",length(probesList),")",sep="") #set main if missing
70  In<-ifelse(keepSmallest,range[1],range[2]) #set inside length
71  if (!add) { #create new plot
```

```
72      if(showAllLines) { #showing all lines
73        if (plotType != "peak") { #genic/intergenic plot
74          if(missing(ylim)) ylim<-range(object[unique(unlist(probesList)),]$
                ratios,na.rm=T) #define ylim if missing
75          plot(1,1,type="n",bty="n",xaxt="n",xlab="",ylab=ylab,xlim=c(-extend,
                In+extend),ylim=ylim,main=main) #initialise plot
76          axis(1,c(-extend,In+extend),c("",""),lwd.ticks=0,line=0) #add axis
77          axis(1,c(-extend,0,In,In+extend),c(labels[1],labels[3],labels[5],
                labels[7]),line=0,tck=0.02) #add axis
78          axis(1,c(-(extend/2),(In/2),In+(extend/2)),c(labels[2],labels[4],
                labels[6]),tick=F,line=-1) #add axis
79        }else{ #peaks plot
80          if(missing(ylim)) ylim<-range(object[unique(unlist(probesList)),]$
                ratios,na.rm=T) #define ylim if missing
81          plot(1,1,type="n",bty="n",xaxt="n",xlab="",ylab=ylab,xlim=c(-extend,
                extend),ylim=ylim,main=main) #initialise plot
82          axis(1,c(-extend,extend),c("",""),lwd.ticks=0,line=0) #add axis
83          axis(1,c(-extend,0,extend),labels,line=0) #add axis
84        }
85      }
86    }
87    if (keepSmallest) scale<-range[1]/regionSizes else scale<-range[2]/
          regionSizes #calculate scales
88    if (plotType=="peak") scale<-rep(1,length(probesList)) #remove scales if
          peaks
89    if (plotType != "peak") trendLinePoints<-seq(-extend,In+extend,length.out=
          averagePoints) else trendLinePoints<-seq(-extend,extend,length.out=
          averagePoints)#define points at which to calculate trend line
90    coordinates<-object$coordinates #get coordinates
91    approxLines<-approxLinesSEs<-matrix(ncol=averagePoints,nrow=ncol(object))
          #initialse matrix to store approx values
92    for (r in 1:ncol(object)) { #loop through datasets
93      ratios<-object$ratios[,r] #get ratios
94      approxLine<-matrix(ncol=averagePoints,nrow=length(probesList)) #
            initialse matrix to store approx values
95      for (n in 1:length(probesList)) { #loop through probes list
96        if (length(probesList[[n]]) > 0) { #if probes are present
97          y<-ratios[probesList[[n]]] #get y-axis values
98          x<-rowMeans(matrix(ncol=2,coordinates[probesList[[n]],2:3])) #get x-
                axis values
99        if (!split) { #data not to be split
100         if (orientations[n] == 1) { #watson strand
101           x<-(x-allRegions[n,2])*scale[n] #shift and scale x-axis values
102         }else{ #crick strand
103           x<-(((max(x)-x)+min(x))-(max(x)-allRegions[n,3]+min(x)))*scale[n]
                  #flip, shift and scale x-axis values
104         }
105       }else{ #data to be split
106         halfway<-(allRegions[n,2]+allRegions[n,3])/2 #get gene halfway
                values
107         if (orientations[n] == 1) { #watson strand
108           y<-c(y[x <= halfway],NA,y[x > halfway]) #add NA to middle of y
                values
109           x<-c(x[x <= halfway]-allRegions[n,2],NA,(x[x > halfway]-allRegions
                [n,3])+In) #split x values
110         }else{ #crick strand
111           y<-c(y[x >= halfway],NA,y[x < halfway]) #add NA to middle of y
                values
112           x<-c(((max(x)-x[x >= halfway])+min(x))-((max(x)-allRegions[n,3])+
                min(x)),NA,((max(x)-x[x < halfway])+min(x))-((max(x)-
                allRegions[n,2])+min(x))+In) #split x values
113         }
```

```
114       }
115       if (showAllLines) points(x,y,type="l",col=allLines.col) #plot all
              lines
116       if (showTrendLine) if (length(which(!is.na(y))) > 1) approxLine[n,]<-
              approx(x,y,xout=trendLinePoints)$y #calculate approx line
117       }
118     }
119     counts<-100*apply(approxLine,2,function(x) {return(length(which(!is.na(x
            ))))})/nrow(approxLine) #get counts
120     approxLines[r,]<-colMeans(approxLine,na.rm=T) #calculate trend line
121     approxLinesSEs[r,]<-apply(approxLine,2,sd,na.rm=T)/sqrt(apply(approxLine
            ,2,function(x) {return(length(which(!is.na(x))))})) #calculate
            standard error
122     approxLines[r,counts < tidy]<-NA #'tidy' data
123   }
124   if (!add) { #create a new plot
125     if (!showAllLines) { #not showing all lines
126       if (plotType != "peak") { #genic/intergenic plot
127         if(missing(ylim)) { #no y limits provided
128           if (showSEs) ylim<-range(c(approxLines+approxLinesSEs,approxLines-
                  approxLinesSEs),na.rm=T) else ylim<-range(approxLines,na.rm=T)
                   #get ylim
129         }
130         plot(1,1,type="n",bty="n",xaxt="n",xlab="",ylab=ylab,xlim=c(-extend,
                In+extend),ylim=ylim,main=main) #initialise plot
131         axis(1,c(-extend,In+extend),c("",""),lwd.ticks=0,line=0) #add axis
132         axis(1,c(-extend,0,In,In+extend),c(labels[1],labels[3],labels[5],
                labels[7]),line=0,tck=0.02) #add axis
133         axis(1,c(-(extend/2),(In/2),In+(extend/2)),c(labels[2],labels[4],
                labels[6]),tick=F,line=-1) #add axis
134       }else{ #peaks plot
135         if(missing(ylim)) { #y limits not provided
136           if (showSEs) ylim<-range(c(approxLines+approxLinesSEs,approxLines-
                  approxLinesSEs),na.rm=T) else ylim<-range(approxLines,na.rm=T)
                   #get ylim
137         }
138         plot(1,1,type="n",bty="n",xaxt="n",xlab="",ylab=ylab,xlim=c(-extend,
                extend),ylim=ylim,main=main) #initialise plot
139         axis(1,c(-extend,extend),c("",""),lwd.ticks=0,line=0) #add axis
140         axis(1,c(-extend,0,extend),labels,line=0) #add axis
141       }
142     }
143   }
144   if (showTrendLine) { #trend line to be plotted
145     for (r in 1:ncol(object)) { #loop through datasets
146       points(trendLinePoints,approxLines[r,],col=col[r],lty=lty[r],lwd=lwd[r
              ],type="l") #add trendline
147       if (showSEs) { #standard errors to be shown
148         SDinclude<-!is.na(approxLines[r,]) & !is.na(approxLinesSEs[r,]) #
                remove NA values
149         polygon(c(trendLinePoints[SDinclude],trendLinePoints[SDinclude][
                length(trendLinePoints[SDinclude]):1]),c((approxLines[r,
                SDinclude]+approxLinesSEs[r,SDinclude]),(approxLines[r,SDinclude
                ][length(trendLinePoints[SDinclude]):1]-approxLinesSEs[r,
                SDinclude][length(trendLinePoints[SDinclude]):1])),col=rgb(
                matrix(ncol=3,col2rgb(col[r])),max=255,alpha=100),lty=0) #add SE
                 polygon
150       }
151     }
152   }
153 }
```

The function sets the plot type to "genic" if not provided (L4) and adjusts the lines to plot depending on the number of datasets provided (L5–8). The y axis label is set if not provided (L9). Colours and line types are set if those provided are not suitable (L10–15). Adjustments are made if standard errors are to be shown (L16–17). A count, `arrayData` object, list, matrix and vector are initialised to store data (L18–22) and a loop through chromosomes initiated (L23). The count is increased by the number of probes on the previous chromosome (L24) and `arrayData` and `annotationData` from the current chromosome extracted (L25-26). The loop moves to the next chromosome if no data is found for the current chromosome (L27). Coordinate data are extracted (L28) and the maximum chromosome coordinate determined (L29). Labels and regions to plot are determined for genic, intergenic or peak plots (L30–41). The function stops with a message if one of these types is not specified (L42). Plot regions are extended in both directions based on the "extend" value and adjacent plot region positions (L44-45). For genic and intergenic plots, regions are extracted based on region sizes or the "geneList" (L47–51). ORF orientations are stored (L53). Probes for each region are stored in a list (L54–59). Probes, regions, orientations and sizes for all chromosomes are stored together (L60–63). Parameters for plots are defined based on the identified probes (L65–70). If a new plot is required, and all lines are to be shown, this is created based on the extracted probe values and required plot type (L71–86). A scale factor is determined (L87-88) and the points at which to define the trend line defined (L89). Coordinates are extracted (L90) and matrices initiated to store trend line and SE values for each dataset (L91). A loop through datasets is initiated (L92). Ratio values are extracted (L93) and a matrix initiated to store the values from which to calculate the trend line and SEs (L94). A loop in initiated through each region with probes to be plotted (L95–96) and the line's x and y values determined (L97–98). These are shifted, scaled, flipped and/or split as determined by the region's scale factor, orientation and the plot type (L99–114). This line is plotted if required (L115) and the trend line points calculated (L116). Counts are calculated for the "tidy" argument (L119), the trend line and SEs calculated from all regions (L120–121) and the "tidy" argument applied

(L122). A new plot is created if all lines are not being shown, based on the trend line and SE values and plot type, and the trend lines/SEs calculated (L124–143). The trend lines/SEs are then added to the current plot, which either contain all other lines or just trend lines (L144–152). The process is largely vectorised for efficiency.

Examples of the types of plots this function can create are shown in Figure 3.8. The genic plot (top panel) is demonstrated with normalised Abf1 binding data over ORFs of defined size with regions of defined size upstream and downstream of the ORFs. All data lines are shown in grey, taken from each of the ORFs and overlayed and scaled to align the start and end positions together. The trend line, shown in red, is calculated from these data. The intergenic plot (middle panel) is demonstrated with a single trend line showing standard errors as a coloured shape around the line. The peak plot is demonstrated with peaks from the `peakDetection` function (Chapter 5), showing three trend lines from three replicate datasets.

3.2.6.4 Rainbow plots

The normalisation procedure presented here (Chapter 4) allows for comparisons to be made between datasets. The "rainbowPlot" function, so named because of the multi-coloured lines it produces, allows these changed to be visualised at a subset of probes. The values for each probe across the datasets are plotted as a single line, which shows the variations between datasets. Data are ordered by the first set of ratios and colours defined by the `rainbow` function. This produces the relevant number of different colours. The order of plotting is calculated from the total differences in probes values between datasets. In this way the largest differences are plotted last, making them more visible on the plot. The clarity of the plot can be further increased by plotting only data where the total difference between values is over a defined amount. The function has the following arguments:

object An `arrayData` object containing more than one dataset to be plotted (no default).

probes A character vector specifying the probes to display data for (no

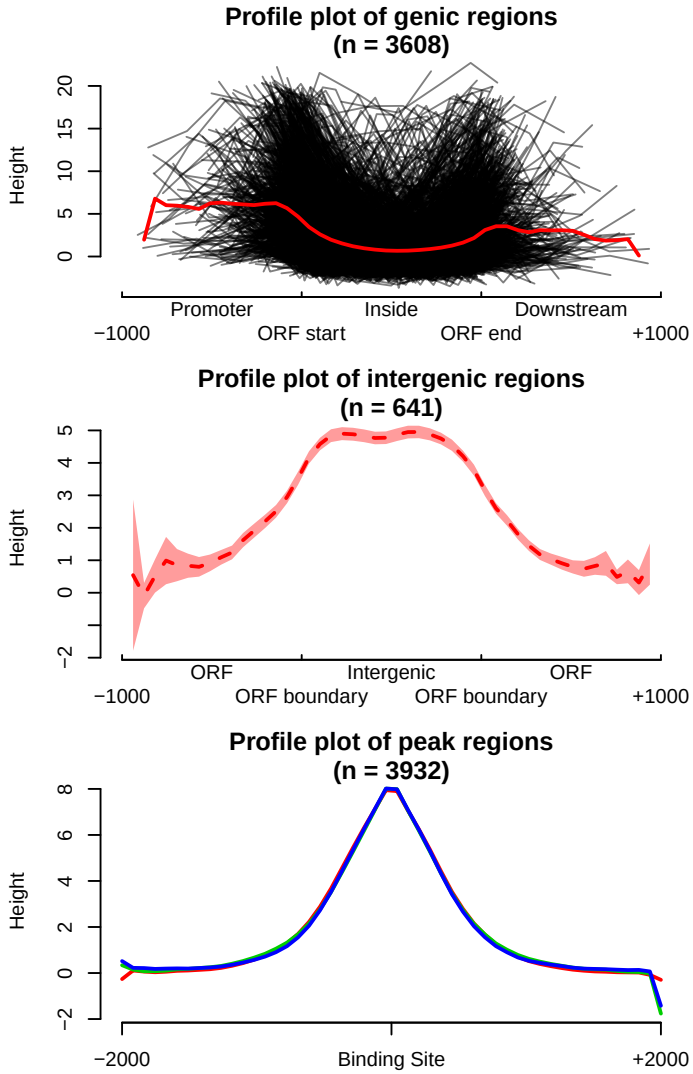


Figure 3.8: Output of `profilePlot` function: Examples of the three types of plot that can be created by the `profilePlot` function using a normalised Abf1 binding dataset. The genic plot (top panel) shows lines for the data across 3,608 ORFs meeting the default criteria (grey), along with a trend line (red). The intergenic plot (middle panel) shows the trend line, with standard errors shown, created from 641 intergenic regions meeting the default criteria. The peak plot (bottom panel) shows the data with three trend lines over 3,932 detected peaks, extending the plot region to ±2000 bp.

default).

dataNames A character vector specifying the names of data to show on the graph (default numeric).

cutoff a numeric vector specifying the difference cutoff (default 0).

ylab A character vector specifying the y-axis label (default "Binding Value (log2)").

Script 3.19: rainbowPlot: script to display the relationship between the same probes of different arrayData objects. Coloured lines show changes in values between datasets.

```
1  ## rainbowPlot function ##
2  ## arguments: object (an arrayData object), probes(a list of probes),
       datanames (for data labels), cutoff (differences value cutoff), ylab (y
       axis label)
3  rainbowPlot<-function(object,probes,dataNames,cutoff=0,ylab="Binding Value (
       log2)",...) { #define function
4    if(ncol(object) < 2) stop("Too few datasets to plot",call.=F) #stop with
         error if not enough data
5    data<-object[probes,]$ratios #get data for relevant peaks
6    data<-data[order(data[,1]),] #sort data by values of first column
7    plot(1,1,type="n",xlim=c(0.8,ncol(data)+0.2),ylim=range(data),bty="n",xaxt
         ="n",xlab="",ylab=ylab,...) #initlaise plot
8    if(missing(dataNames)) dataNames<-1:ncol(object) #set names if missing
9    axis(1,1:ncol(data),dataNames,tick=F,line=-1) #draw axis
10   diffs<-apply(abs(diff(data)),1,sum) #calculate total differences
11   data<-data[diffs > cutoff,] #get data above defined differences cutoff
12   plotOrder<-order(diffs[diffs > cutoff]) #set plot order by differences
13   col<-rainbow(nrow(data),start=0,end=0.9) #set colours
14   for(n in plotOrder) { #loop through data
15     points(1:ncol(data),data[n,],type="l",col=col[n]) #add data line
16   }
17   grid(nx=0,ny=NULL) #add grid
18 }
```

The function first checks enough datasets are provided and stops with a message if not (L4). The ratios to be plotted are then extracted (L5) and ordered (L6). A new plot is created based on the ratio values and number of datasets (L7). Data names are specified if required (L8) and the x-axis drawn (L9). Total differences across all datasets for each probe are calculated (L10) and those above the "cutoff" extracted (L11). The plot order is determined by the difference values, from smallest to largest (L12). Rainbow colours are calculated (L13) and each set of lines added in the correct order with the relevant colour (L14–16). Finally horizontal lines of a grid are added (L17). A rainbow plot has been used to analyse Abf1 binding data in Figure 7.10.

3.2.7 Annotating data

Each probe in an `arrayData` object is associated with its genomic location, but this does not give any context to that location, specifically how it relates to ORFs. It is useful to be able to determine this information as it adds another layer of depth to analyses that can be performed. The function `getProbeInfo` (Script 3.20) was written to perform this task. For each probe, the nearest ORF is found, based on the data of a `genomeAnnotation` object, and its name and the probe's distance to its start coordinate stored. Whether or not the probe is in that ORF is determined, and a labelling process initiated accordingly. Probes in ORFs are labelled as "Inside". Intergenic probes are labelled depending on their distance from their nearest ORF, the orientation of that ORF and the "distance" value. This specifies a cutoff value used when annotating probes in intergenic regions, with distances over this cutoff being labelled "Intergenic", rather than "Promoter" or "Downstream". The adjacent ORF is also examined and a "Divergent" label applied if the ORF is in the opposite orientation to the closest ORF. This may result in a divergent promoter region, where two ORFs point away from each other, or a divergent downstream region, where two ORFs point towards each other (Figure 3.9). The process is largely vectorised for efficiency. The function has the following arguments:

object An `arrayData` object specifying the probe(s) to be processed (no default).

annotation A `genomeAnnotation` object from which to take annotation information (no default).

distance A numerical vector specifying the distance cutoff (default `Inf`).

append A logical vector specifying whether or not to append the resulting data to the `arrayData` object as extra annotation columns, or return it as a new matrix (default FALSE).

Script 3.20: getProbeInfo: script to associate probes with their nearest ORFs and provide information on their positions relative to these.

```
## getProbeInfo function ##
## arguments: object (an arrayData object), annotation (a genomeAnnotation object), distance (distance to consider promoter/downstream), append (whether to add the results to the arrayData object)
getProbeInfo<-function(object,annotation,distance=Inf,append=FALSE) { #define function
  results<-matrix(ncol=4,nrow=nrow(object)) #initialise matrix to store results
  colnames(results)<-c("ClosestGene","Distance","Association","Divergent") #set result column names
  x<-1 #start count at 1
  for (chr in unique(object$coordinates[,1])) { #loop through chromosomes
    annoChr<-annotation[annotation$coordinates[,1] == chr] #get annotation for current chromosome
    objectChr<-object[object$coordinates[,1] == chr,] #get arrayData for current chromosome
    if (nrow(annoChr) == 0) { #if no annotation for current chromosome
      results[x:(x+nrow(objectChr)-1),]<-NA #store NA values
      x<-x+nrow(objectChr) #increase count
    }else{ #annotation present for current chromosome
      result<-matrix(ncol=4,nrow=nrow(objectChr))
      afterGene<-inGene<-matrix(ncol=1,nrow=nrow(objectChr))
      mids<-rowMeans(matrix(ncol=2,objectChr$coordinates[,2:3])) #get mid points
      for (n in 1:nrow(objectChr)) { #loop through probes
        A<-annoChr$coordinates[,3] >= mids[n] #find genes greater than mid
        B<-annoChr$coordinates[,2] <= mids[n] #find genes less than mid
        ingene<-which(A+B > 1) #find if probe is in a gene
        if (length(ingene) > 0) { #probe is in a gene
          if (length(ingene) > 1) ingene<-ingene[which.max(apply(annoChr$coordinates[ingene,2:3],1,diff))] #get longest gene if more than one
          inGene[n]<-ingene #set ORFs probes are in
        }else{ #probe is not in a gene
          afterGene[n]<-which(c(A,1)+c(1,B) > 1)-1 #find ORFs probes are after
        }
      }
      present<-which(!is.na(inGene)) #probes in a gene
      result[present,]<-cbind(annoChr$annotations[inGene[present],1],ifelse(as.numeric(annoChr$annotations[inGene[present],3]) == 1,mids[present]-annoChr$coordinates[inGene[present],2],annoChr$coordinates[inGene[present],3]-mids[present]),rep("Inside",length(present)),rep(FALSE,length(present))) #get info for probes in genes
      mids<-mids[!is.na(afterGene)] #get mids for intergenic probes
      afterGene<-afterGene[!is.na(afterGene)] #get non-na aftergenes
      intergenic<-matrix(ncol=7,nrow=length(afterGene)) #create matrix to store data
      intergenic[,4]<-mids #insert mids
      intergenic[,1]<-afterGene #insert LHS positions
      intergenic[(intergenic[,1] == 0),1]<-NA #remove 0 values
      intergenic[,5]<-afterGene+1 #insert RHS positions
      intergenic[(intergenic[,5] > nrow(annoChr)),5]<-NA #remove values too large
      intergenic[,2]<-ifelse(as.numeric(annoChr$annotations[intergenic[,1],3]) == 1,1,0) #define if lhs gene is watson
      intergenic[,3]<-mids-annoChr$coordinates[intergenic[,1],3] #calculate distances
```

```
40        intergenic[,6]<-ifelse(as.numeric(annoChr$annotations[intergenic
              [,5],3]) == 1,1,0) #define if rhs gene is watson
41        intergenic[,7]<-annoChr$coordinates[intergenic[,5],2]-mids #calculate
              distances
42        present<-which(is.na(inGene)) #intergenic positions
43        closer<-intergenic[,3] < intergenic[,7] #LHS closer than RHS
44        result[present,1]<-ifelse(closer,annoChr$annotations[intergenic
              [,1],1],annoChr$annotations[intergenic[,5],1]) #get closest gene
              names
45        result[present,2]<-ifelse(closer,intergenic[,3],intergenic[,7]) #get
              closest gene ditances
46        result[present,3]<-ifelse(closer,ifelse(intergenic[,2] == 1,"
              Downstream","Promoter"),ifelse(intergenic[,6] == 1,"Promoter","
              Downstream")) #define promoter/downstream
47        result[present,4]<-intergenic[,2] != intergenic[,6] #define divergent
48        present<-which(is.na(inGene))[is.na(intergenic[,1])] #probes before
              first gene
49        result[present,]<-cbind(annoChr$annotations[intergenic[is.na(
              intergenic[,1]),5],1],intergenic[is.na(intergenic[,1]),7],ifelse(
              intergenic[is.na(intergenic[,1]),6] == 1,"Promoter","Downstream"),
              rep(FALSE,length(present))) #info for probes before first gene
50        present<-which(is.na(inGene))[is.na(intergenic[,7])]     #probes after
               last gene
51        result[present,]<-cbind(annoChr$annotations[intergenic[is.na(
              intergenic[,7]),1],1],intergenic[is.na(intergenic[,7]),3],ifelse(
              intergenic[is.na(intergenic[,7]),2] == 1,"Downstream","Inside"),
              rep(FALSE,length(present))) #info for probes after last gene
52      results[x:(x+nrow(objectChr)-1),]<-result #store all results
53      x<-x+nrow(objectChr) #increase count
54      }
55    }
56    results[as.numeric(results[,2]) > distance & results[,3] != "Inside",3]<-"
          Intergenic" #set intergenic by distance value
57    if(append) { #data is to be appended to arrayData
58      object$annotations<-cbind(object$annotations,results) #add results to
            annotations
59      return(new("arrayData",object)) #return arrayData
60    }else{ #data is not to be appended
61      return(results) #return matrix
62    }
63  }
```

The function creates a matrix to store the results (L4–5), creates a count (L6) and initiates a loop through chromosome numbers (L7). The `arrayData` and `genomeAnnotation` for each chromosome are extracted (L8–9). If no annotation is available for the current chromosome, NA values are set for all probes on that chromosome and the count incremented (L10–12). Otherwise three matrices are created (L14–15), the probe mid-points calculated (L16) and a loop through probes initialised (L17). ORF end points above, and start points below, the mid-points are identified (L18–19). These are used to determine whether the probe is in an ORF (L20). If it is (L21), the longest gene is taken in the case of overlapping genes (L22) and the values stored

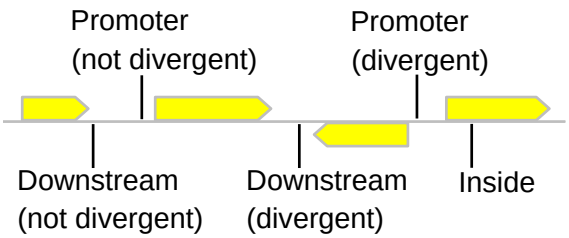


Figure 3.9: ORF position examples: the labels that may be assigned to probes in different positions relative to their nearest ORF. Yellow boxes represent ORFs with the arrow indicating the direction of transcription.

(L23). Otherwise the preceding gene is identified (L25). For probes in ORFs (L28) the results are compiled (L29). For intergenic probes the mid-points (L30) and numbers are extracted (L31). Details of ORFs to the left- and right-hand-sides are gathered (L32–41). These are used to determine the association details for internal (L42–47) and external ORFs (L48–51). The results are stored (L52) and the count increased (L53). Intergenic probes are identified, based on the "distance" value (L56). If the results are to be appended to the `arrayData` object this is carried out and the new object returned (L57–59), otherwise the matrix of results is returned (L61).

3.2.7.1 Positions plot

The positions of peak positions relative to ORFs can be important to know, especially for proteins that bind throughout a genome. The `getProbeInfo` function can be used to find this information for each probe at the top of a potential binding region (PBR). The `positionsPlot` function (Script 3.21) uses this information to create three graphics. The function has the following arguments:

peakList The `peakList` object to be plotted (no default).

probeInfo The matrix of results of the `getProbeInfo` function for the `peakList` being plotted (no default).

Script 3.21: positionsPlot: script to display positional information from a `peakList` object.

```
1  ## positionsPlot function ##
2  ## arguments: probeInfo (result of getProbeInfo), peakList (a peakList
       object)
3  positionsPlot<-function(probeInfo,peakList) {
4    op<-par(no.readonly = TRUE); on.exit(par(op)) #reset current par on exit
5    Pr<-In<-Do<-0 #initialise vectors to store max values
6    if(length(which(probeInfo[,3] == "Promoter")) > 0) Pr<-max(as.numeric(
         probeInfo[probeInfo[,3] == "Promoter",2]),na.rm=T) #max promoter value
7    if(length(which(probeInfo[,3] == "Inside")) > 0) In<-max(as.numeric(
         probeInfo[probeInfo[,3] == "Inside",2]),na.rm=T)  #max inside value
8    if(length(which(probeInfo[,3] == "Downstream")) > 0) Do<-max(as.numeric(
         probeInfo[probeInfo[,3] == "Downstream",2]),na.rm=T)  #max downstream
         value
9    if(Pr < In/2) Pr<-In/2 #scale promoter if required
10   if(Do < In/2) Do<-In/2 #scale downstream if required
11   layout(matrix(c(1,2,3),3,1),heights=c(0.3,0.7,0.3)) #set layout
```

```
12    par(mar=c(0,4,2,2)) #set par
13    plot(density(c(Pr-as.numeric(probeInfo[probeInfo[,3]=="Promoter",2]),Pr+as
          .numeric(probeInfo[probeInfo[,3]=="Inside",2]),Pr+In+as.numeric(
          probeInfo[probeInfo[,3]=="Downstream",2])),na.rm=T),main="",bty="n",
          xaxt="n",yaxt="n",xlab="",ylab="Density",mar=c(1,2,2,2),xlim=c(0,Pr+In
          +Do),lwd=2) #plot density
14    abline(v=c(Pr,Pr+In),col = "lightgray", lty = "dotted") #add vertical
          lines
15    axis(2,labels=F,lwd.tick=0) #add axis
16    par(mar=c(2,4,0,2)) #set par
17    plot(1,1,type="n",bty="n",xaxt="n",xlab="Genome Position",ylab="Peak
          Height",xlim=c(0,Pr+In+Do),ylim=range(c(0,peakList$stats[,2]))) #
          initialise plot
18    axis(1,c(0,Pr+In+Do),c("",""),lwd.ticks=0,line=0) #add axis
19    axis(1,c(0,Pr,Pr+In,Pr+In+Do),c(-Pr,0,In,paste("+",Do,sep="")),line=0) #
          add axis
20    axis(1,c(Pr/2,Pr+(In/2),Pr+In+(Do/2)),c("Promoter","Inside","Downstream"),
          tick=F,line=-1) #add axis
21    par(pch=16,cex=0.6,col=rgb(0,0,0,alpha=0.5)) #set par
22    points(Pr-as.numeric(probeInfo[probeInfo[,3]=="Promoter",2]),as.numeric(
          peakList$stats[probeInfo[,3]=="Promoter",2])) #plot promoter points
23    points(Pr+as.numeric(probeInfo[probeInfo[,3]=="Inside",2]),as.numeric(
          peakList$stats[probeInfo[,3]=="Inside",2])) #plot inside points
24    points(Pr+In+as.numeric(probeInfo[probeInfo[,3]=="Downstream",2]),as.
          numeric(peakList$stats[probeInfo[,3]=="Downstream",2])) #plot
          downstream points
25    abline(v=c(Pr,Pr+In),col = "lightgray", lty = "dotted") #add verital lines
26    grid(nx=0,ny=NULL)    #add grid
27    par(mar=c(4,8,2,8),las=1) #set par
28    barplot(matrix(c(100*(length(which(probeInfo[,3]=="Downstream" & probeInfo
          [,4]=="FALSE"))/nrow(peakList)),100*(length(which(probeInfo[,3]=="
          Downstream" & probeInfo[,4]=="TRUE"))/nrow(peakList)),100*(length(
          which(probeInfo[,3]=="Inside"))/nrow(peakList)),0,100*(length(which(
          probeInfo[,3]=="Promoter" & probeInfo[,4]=="FALSE"))/nrow(peakList))
          ,100*(length(which(probeInfo[,3]=="Promoter" & probeInfo[,4]=="TRUE"))
          /nrow(peakList))),ncol=3,byrow=F),horiz=T,beside=FALSE,xlab="
          Percentage",names.arg=c("Downstream","Inside","Promoter")) #plot
          barplot
29    }
```

The function gets the current `par` settings to be reset when exiting (L4) and creates vectors for the storage of x-axis values (L5). The maximum promoter, inside and downstream values are extracted (L6–8) and scaled if required (L9–10). These define the values along the x-axis and the positions of the promoter/inside and inside/downstream boundaries. The plot layout and parameters are set (L11–12) and the density plot created (L13). Lines indicating the boundaries (L14) and a y-axis (L15) are added. The plot margins are adjusted (L16) and a plot initialised for the probes (L17). The x-axis showing the ORF positions is added (L18–20) and the point parameters set (L21). Points are plotted for the promoter (L22), inside (L23) and downstream (L24) regions. Lines indicating the boundaries (L25) and $\log_2$

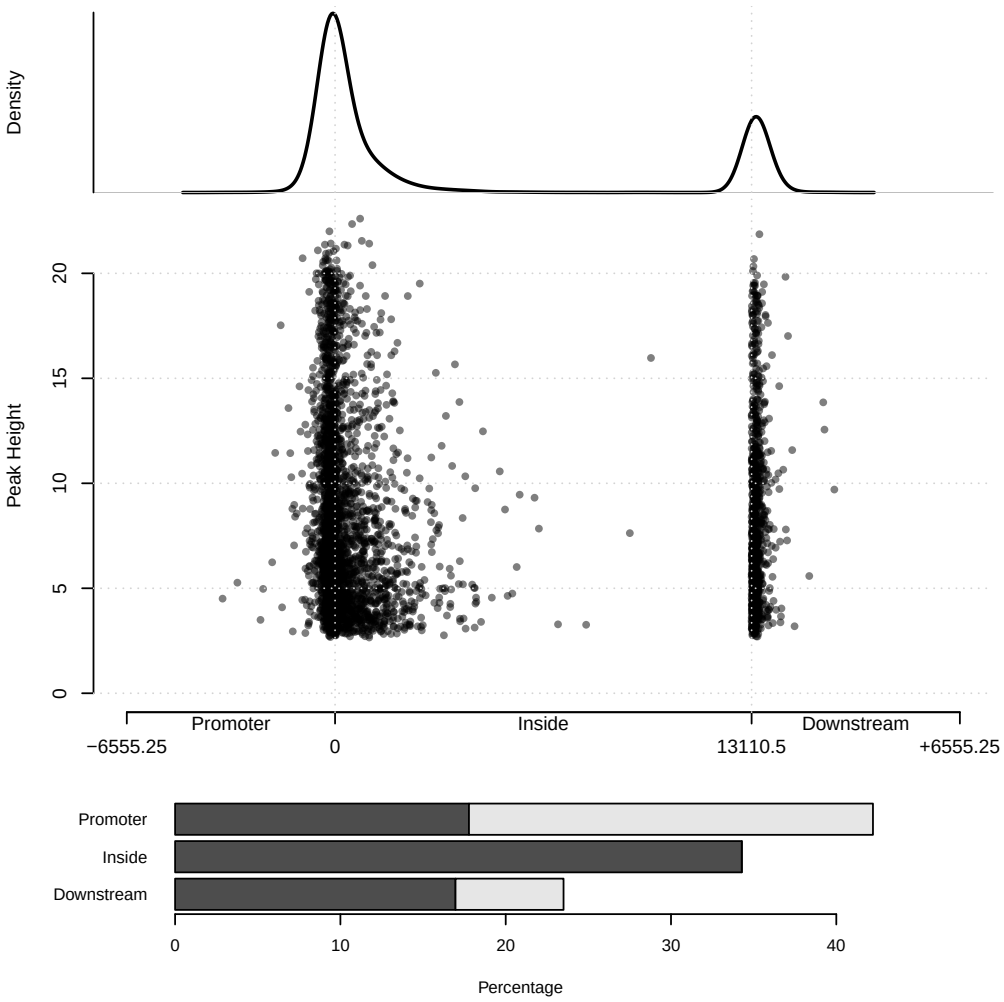


Figure 3.10: Output of `positionsPlot` function: Example of the graphics created, using results from peak detection performed on Abf1 binding datasets. The middle panel shows probe positions plotted against their heights with the top panel showing a density plot of this data. The bottom panel shows the percentage of probes in different categories, with intergenic regions shown in lighter grey.

ratios (L26) are added. The margins for the final plot are set (L27) and the bar plot created (L28).

An example of the `positionsPlot` output is shown in Figure 3.10. The centre of the graphic shows a plot of peak heights relative to their nearest ORF. A promoter, inside and downstream region are displayed along the x-axis, with $\log_2$ ratios on the y-axis. The range of the x-axis is determined by the largest value in the probe information. Each probe is shown with a dot, relating its genome position to its height. Above this is a density plot, representing the numbers of probes along the promoter/ORF/downstream region. Overlapping dots, in regions with lots of probes, obscure each other and therefore prevent an accurate estimation of the relative numbers of probes. The density plot overcomes this by indicating the numbers of probes at a given location, allowing the region(s) with the most probes to be identified. The bottom of the graphic shows the percentage of probes in promoter, inside and downstream regions as a bar graph, indicating those from divergent regions with lighter shades of grey.

3.2.7.2 Venn diagrams

Venn diagrams are a method of representing similarity between different groups of items, with item counts displayed in shapes created by overlapping circles. The function `venn` (Script 3.22) has been written to create Venn diagrams to show similarity between two or three `peakList` objects. The function provides two methods for displaying this data. The first creates a standard Venn diagram, where the numbers represent counts of identical probes between different lists. The second employs the `overlap` function (Script 3.23), which identifies PBRs from different `peakList` objects which overlap, potentially representing the same binding site but which may have different probes at their peaks and therefore would not be identified by the standard Venn diagram format. The function has the following arguments:

peakList1 The first `peakList` object (no default).
peakList2 The second `peakList` object (no default).
peakList3 The third `peakList` object (optional; no default).

overlap A logical vector specifying whether or not to employ the overlap function (default FALSE).

arrayData An `arrayData` object containing probe coordinates (required by the `overlap` function; no default).

windowSize A numeric vector specifying the window size to use when calculating overlaps (no default).

cex A numeric vector defining the size of the plotted circles (default 40).

Script 3.22: venn: script to create a Venn diagram showing the relationship between the contents of two or three `peakList` objects.

```
## venn function ##
## arguments: peakList1 (first peakList), peakList2 (second peakList), peakList3 (third optional peakList), overlap (whether to use overlap), arrayData (an arrayData object for overlap), windowSize (window size value for overlap), cex (cirle size adjustment).
venn<-function(peakList1,peakList2,peakList3,overlap=F,arrayData,windowSize,cex=40) { #define function
  if (peakList1$grid_name != peakList2$grid_name) stop("peakList grid names do not match",call.=F)
  if(!overlap) { #no overlap required
    a<-matrix(peakList1$IDs[,1]) #get first probe IDs
    b<-matrix(peakList2$IDs[,1]) #get second probe IDs
    ab<-matrix(intersect(a,b)) #a b intersect
    if(!missing(peakList3)) { #a third list is povided
      if (peakList1$grid_name != peakList3$grid_name) stop("peakList grid names do not match",call.=F) #stop if grid names don't match
      c<-matrix(peakList3$IDs[,1]) #get third probe IDs
      bc<-matrix(intersect(b,c)) #b c intersect
      ac<-matrix(intersect(a,c)) #a c intersect
      abbc<-matrix(intersect(ab,bc)) #ab bc intersect
      abac<-matrix(intersect(ab,ac)) #ab ac intersect
      bcac<-matrix(intersect(bc,ac)) #bc ac intersect
    }
  }else{ #overlap required
    if (arrayData$grid_name != peakList1$grid_name) stop("arrayData and peakList grid names do not match",call.=F) #grid names don't match
    a<-peakList1 #get first peak list
    b<-peakList2 #get second peak list
    ab<-overlap(a,b,arrayData,windowSize) #a b overlap
    if(!missing(peakList3)) { #a third list is provided
      if (peakList1$grid_name != peakList3$grid_name) stop("peakList grid names do not match",call.=F)  #stop if grid names don't match
      c<-peakList3 #get thrid peak list
      bc<-overlap(b,c,arrayData,windowSize) #b c overlap
      ac<-overlap(a,c,arrayData,windowSize) #a c overlap
      abbc<-overlap(ab,bc,arrayData,windowSize) #ab bc overlap
      abac<-overlap(ab,ac,arrayData,windowSize) #ab ac overlap
      bcac<-overlap(bc,ac,arrayData,windowSize) #bc ac overlap
    }
  }
  par(bty="n",xaxt="n",yaxt="n") #set par
  plot(c(1,1.835),c(1,1),cex=cex,xlim=c(0,3),ylim=c(-1,2),xlab="",ylab="",pch=16,col=c(rgb(1,0,0,0.3),rgb(0,1,0,0.3))) #initialise plot
```

```
35    if(!missing(peakList3)) { #third list is present
36      points(1.42,0.19,cex=cex,pch=16,col=rgb(0,0,1,0.3)) #third circle
37      text(0.62,1.22,nrow(a)-nrow(ab)-nrow(ac)+nrow(abac)) #show numbers
38      text(2.2,1.22,nrow(b)-nrow(ab)-nrow(bc)+nrow(abac)) #show numbers
39      text(1.42,-0.26,nrow(c)-nrow(ac)-nrow(bc)+nrow(abac)) #show numbers
40      text(1.42,1.45,nrow(ab)-nrow(abac)) #show numbers
41      text(2,0.35,nrow(bc)-nrow(abac)) #show numbers
42      text(0.84,0.35,nrow(ac)-nrow(abac)) #show numbers
43      text(1.42,0.72,nrow(abac)) #show numbers
44    }else{
45      text(0.62,1,nrow(a)-nrow(ab)) #show numbers
46      text(2.2,1,nrow(b)-nrow(ab)) #show numbers
47      text(1.42,1,nrow(ab)) #show numbers
48    }
49  }
```

The function first checks that the grid names of the first and second `peakList` objects match (L4). If the `overlap` function is not to be used (L5) the `peakList` IDs are extracted (L6-7) and those identical identified (L8). If a third `peakList` is provided (L9) its grid name is checked (L10) and its IDs extracted (L11). Identical IDs are identified between all possible combinations (L12–16). Where the overlap function is to be used (L18) the grid name of the `arrayData` object is checked against that of the first `peakList` (L19). The `overlap` function is applied to the first two `peakList`s (L20-22). If a third `peakList` is provided (L23) its grid name is checked (L24) and `overlap` applied to all possible combinations (L25–30). The parameters of the plot are set (L33) and a plot created containing the first two circles (L34). If a third `peakList` is provided (L35) a third circle is drawn (L36) and all calculated counts printed at the relevant positions (L37–43). Otherwise calculated counts for the first two `peakList`s are printed (L44–47) at the relevant positions.

The `overlap` function determines overlapping PBRs using coordinates from a provided `arrayData` object and the defined window size value. Probes in one dataset which occur within the window size distance away from adjacent probes in the second dataset are deemed to be overlapping. In this way PBRs containing at their peaks adjacent probes within the window size distance of each other, called as peaks in different `peakList`s, will be deemed to be overlapping and therefore created by the same binding site, whereas those separated by one or more probes are considered to be created by separate binding sites. The function is vectorised making it efficient at comparing

between lists of hundreds or thousands of PBRs. It is called by the `venn` function, providing the two `peakList` objects to be analysed ("peakList1" and ""peakList2""), the `arrayData` object ("arrayData") and the window size ("windowSize").

Script 3.23: overlap: script to determine overlapping PBRs from `peakList` objects.

```
1  ## overlap function ##
2  ## arguments: peakList1, peakList2, arrayData, windowSize
3  overlap<-function(peakList1,peakList2,arrayData,windowSize) {
4    probes<-arrayData$annotations[,1] #get probe names
5    positions<-rowMeans(arrayData$coordinates[,2:3]) #get probe mid points
6    chrs<-arrayData$coordinates[,1] #get coordinates
7    match1<-which(peakList1$IDs[,1] %in% peakList2$IDs[,1]) #get matching
         positions
8    arrayDataPositions<-which(probes %in% peakList2$IDs[,1]) #probes in
         peakList2
9    arrayDataPositions.d<-arrayDataPositions[arrayDataPositions > 1] #probes
         to look down
10   arrayDataPositions.u<-arrayDataPositions[arrayDataPositions < nrow(
         arrayData)]   #probes to look up
11   match2<-which(peakList1$IDs[,1] %in% probes[ifelse(chrs[arrayDataPositions
         .d] == chrs[arrayDataPositions.d-1] & (positions[arrayDataPositions.d]
          - positions[arrayDataPositions.d-1]) < windowSize,arrayDataPositions.
         d-1,arrayDataPositions.d)]) #overlap down
12   match3<-which(peakList1$IDs[,1] %in% probes[ifelse(chrs[arrayDataPositions
         .u] == chrs[arrayDataPositions.u+1] & (positions[arrayDataPositions.u]
          - positions[arrayDataPositions.u+1]) < windowSize,arrayDataPositions.
         u+1,arrayDataPositions.u)]) #overlap up
13   return(peakList1[unique(c(match1,match2,match3))]) #return overlapping
         probes
14 }
```

The function extracts probe names (L4), mid-points (L5) and chromosome numbers (L6) from the `arrayData` object. Matching probes from both of the `peakList` objects are identified (L7). `arrayData` positions of probes in the second `peakList` are identified (L8). Those suitable for looking downwards and upwards from, that is, those not at the start or end of the genome, are identified (L9–10). Probe names of "peakList1" are compared to probe names downstream of each "peakList2" probe name, within the window size and on the same chromosome (L11). This is repeated for upstream probes (L12). A `peakList` containing these identified PBRs is returned (L13). The coordinate component of this `peakList` will vary depending on the order the two `peakLists` are provided to the function, and so these are not useful for downstream analyses, but the numbers of peaks returned and the IDs

component will always be the same, making it suitable for use by the `venn` function and use in further `overlap` analyses.

3.2.7.3 Extracting sequence information

In addition to displaying information about detected peaks, analysis of the DNA sequences of PBRs can reveal important information about binding sites. Identification of a binding motif is one potential use of these sequences. The function `getSequences` was written to extract these sequences from `peakList` PBRs using the `BSgenome` package (Pages, 2012b). This package downloads entire genome sequences and, among its features, allows portions of these to be extracted. The `getSequences` function uses this feature to extract sequences based on the coordinates of the PBRs stored in the `peakList`. The resulting sequences can be analysed in `R`, with packages such as `Biostrings` (H et al.), or written as FASTA files to be loaded into and processed in other programs. The function has the following arguments:

peakList The `peakList` object to get sequences for (no default).

genome A (previously downloaded) `BSgenome` genome from which to get sequences (no default).

unmask A logical vector specifying whether to apply the `unmasked` function to masked sequences (default FALSE).

Script 3.24: getSequences: script to get sequence information from PBRs of a `peakList` using the `BSgenome` package.

```
1  ## getSequences function ##
2  ## arguments: peakList (a peakListObject), genome (a previously downloaded
       genome), unmask (whether to unmask the genome)
3  getSequences<-function(peakList,genome,unmask=FALSE) { #define function
4    require(BSgenome) #load package
5    results<-matrix(ncol=1,nrow=nrow(peakList)) #initialse matrix to store
         results
6    x<-1 #set x = 1
7    for (chr in unique(peakList$coordinates[,1])) { #loop through chromosomes
8      if(unmask) genomeChr<-unmasked(genome[[chr]]) else genomeChr<-genome[[
           chr]] #get chromosome sequence
9      peakListChr<-peakList[peakList$coordinates[,1] == chr] #get chromosome
           peak list
10     coordinates<-round(peakListChr$coordinates[,2:3],0)
11     coordinates[coordinates < 1]<-1 #correct coordinates less than 1
12     coordinates[coordinates > length(genomeChr)]<-length(genomeChr) #correct
            coordinates greater than the chromosome length
```

```
13      if (nrow(peakListChr) > 0) { #if a peak is on the chromosome
14        for (n in 1:nrow(peakListChr)) { #loop through peaks
15          results[x]<-as.character(genomeChr[coordinates[n,1]:coordinates[n
                ,2]]) #get sequence in PBR
16          x<-x+1 #increment x
17        }
18      }
19    }
20    rownames(results)<-peakList$IDs[,1]
21    return(results) #return sequences
22  }
23
24  #source("http://bioconductor.org/biocLite.R")
25  #biocLite("BSgenome.Scerevisiae.UCSC.sacCer3")
26  #require(BSgenome.Scerevisiae.UCSC.sacCer3)
27  #yeastGenome<-get("Scerevisiae","package:BSgenome.Scerevisiae.UCSC.sacCer3")
```

The function first loads the `BSgenome` package if required (L4), creates a matrix to store the extracted sequences (L5) and starts a count (L6). A loop through chromosomes is initiated (L7) and the chromosome sequence set (L8). PBR coordinates for the current chromosome are extracted (L9–10) and adjusted if they go beyond the chromosome sequence (L11–12). Where peaks are present on the chromosome (L13) a loop for peaks is initiated (L14), the sequence over each PBR stored (L15) and the count incremented (L16). Probes IDs from the `peakList` are added as row names (L20) and the sequences returned (L21). An example of loading the yeast genome sequence for use in the function is shown (L24–27).

3.3 Discussion

These tools provide methods for easily processing ChIP-chip data of any format in `R`. A new format for the storage of this data is created in the form of the `arrayData` object, which all other functions presented use. Importantly, the novel normalisation and enrichment detection procedures, presented in the following two chapters, also work with data in this format. All functions presented are therefore compatible with each other, with the results of one able to be used by another.

Previously, the analysis of ChIP-chip data was generally limited to simply determining the locations of the presence of the factor under investigation, such as protein binding sites or the locations of epigenetic modifications (such

as the investigation of Abf1 binding under different conditions by Schlecht et al., 2008). Such analyses require a method of determining sites of enrichment (discussed in Chapter 5), the results of which reduce the data to a list of locations. The format of the complete data is not therefore very important, because it would only be used in this single process. Available tools for additional analyses are limited and disparate, meaning a simple workflow from raw data to the type of results generated here did not exist (software packages include those developed by Toedling et al., 2007; Andrews, 2007; Zhang et al., 2007; Toedling and Huber, 2008). This was a particular problem for biological researchers with limited or no bioinformatic experience, for whom analysing ChIP-chip data using these separate tools could prove difficult. The tools presented here have been created with these people in mind, so as to provide fast and simple ways of generating results from raw data and plotting these graphically where relevant. The instruction document ("instructions.pdf" in the electronic appendix; see Page 367) shows these processes. This does not however limit the tools, as the power of the `R` software allows people with the ability to use it to more advanced levels to perform their own, customised analyses on the data in addition to those presented here. The data can also be written to a text file during any stage of its processing, which can be used in other programs if required. This data may then be read back into `R`, if required, for further analysis or to plot the results of this external processing.

The work here has opened up a new level of analysis of ChIP-chip data, outlined further in the following chapters, expanding the potential of the technology and along with it the results that can be gleaned from a dataset. The tools presented here allows this to be achieved through the new `arrayData` format, which has been created to contain all of the relevant data in an easily accessible format. Previously, most microarray data were loaded into `R` in a format designed for gene expression analysis, such as the `RGList` object from `limma` (for example, Toedling et al., 2007). While these may have been adequate for basic ChIP-chip analyses, they do not have the flexibility required to manipulate ChIP-chip data in the ways described here. The `arrayData` format is simple and fully described, allowing users to access specific data

where required, which may be used by other `R` functions or to create graphics for which specific functions have not been written.

Not all of the tools presented in this chapter are completely unique, nor are they intended to replace their existing counterparts. The plotting of data along the genome, for example, can be achieved through a number of other programs, such as the UCSC genome browser (Kent et al., 2002). The other graphical functions are relatively simple and similar plots have been created previously, although the programs used to do this have not been published (see, for example, Pokholok et al., 2005). The functions presented here are instead intended to bring all of the tools relevant to the ChIP-chip analysis and processing together, within a single program, using a common data format. This simplifies the analysis process, with data being able to be loaded in its raw state, normalised, enrichment detection performed and a series of graphics created in a matter of minutes. More complex analyses, such as within the context of genome data in the `genomeAnnotation` object or sequence data extracted from a downloaded genome, can then be performed repeatedly to ask varying questions of the data. Other Bioconductor packages can also be used to analyse the results, depending on the investigation.

A number of the functions presented in this chapter produce graphical outputs of some sort. While these can be used as a way of demonstrating a final result, they are primarily intended to facilitate the discovery of results, extending the hypothesis generating status of ChIP-chip by indicating patterns or features in data that may warrant further investigation. These patterns may not be identifiable in the data as a whole, but become apparent through these methods of plotting. Mathematical processing of the data, such as calculating the differences between the values of one dataset and another, can also be plotted using these functions, expanding the potential for the generation of new hypotheses. These can then be confirmed and demonstrated by other methods, such as the use of other technologies or applying statistical tests to subsets of the data.

As previously stated, it can be extremely difficult, if not impossible, to corroborate the entire results of a ChIP-chip dataset. Confirming results for a representative subset of data with a different technology is as close as it

may be possible to get. The tools presented here can facilitate this in two ways. Firstly, they can be used to determine appropriate points in the data to test with other technologies. For example, in the enrichment detection chapter (Chapter 5) a selection of data points are validated with Q-PCR. To choose these points, the `arrayData plot` function was used to plot all of the datasets. From this a selection of probes representing different aspects of the data, such as different binding values, were chosen. Primers designed around these probe locations were then used to perform Q-PCRs to validate the results. Secondly, they can be used to display likely genuine results on the basis of this subset of tested results. For example, if a subset of detected peaks from a dataset are shown by another technology to be genuine binding sites, it is reasonable to assume that all peaks represent genuine binding sites. These peaks can then be analysed or plotted together, in various ways, to expand the confirmed results to results that span the whole genome.

Chapter 4

Development of a novel normalisation method

4.1 Introduction

Normalisation, in the context of microarray data, is the processing of different datasets so as to reduce technical variations between them. Normalisation of replicate datasets of the same condition (intra-dataset normalisation) aims to reduce technical variations within these while maintaining the underlying biological signal. Normalisation of datasets of different conditions (inter-dataset normalisation) aims to perform the same function as intra-dataset normalisation whilst maintaining any biologically important differences between the biological signals from the different conditions. Several normalisation methods have been developed to achieve these two objectives for gene expression microarrays, which allows comparisons to be made between mRNA levels from different experimental conditions. No such procedure exists for the inter-dataset normalisation of ChIP-chip data, meaning the levels from from different experimental conditions cannot be reliably compared with each other. The importance of normalisation to enable comparisons is summarised by Gentleman (2005):

> "Observed intensities need to be adjusted to give accurate measurements of specific hybridisation. Without proper normalisation, it is impossible to compare measurements from different array hybridisations due to many obscuring sources of variation."

Most ChIP-chip investigations to date have been into a single condition, or comparisons between different conditions have been very limited. In these investigations datasets are treated in isolation and the properties of the factor under investigation are determined independently of any other factor. Therefore the results of the investigations are generally lists of chromosomal locations at which the factor under investigation is present, as determined by a suitable enrichment detection method. This restricts the analysis of the results, as each different experimental condition can only be compared in a binary fashion, such as the Abf1 binding analysis carried out by Schlecht et al. (2008). That is to say, only the presence or absence of the factor under investigation at the same site in the datasets can be compared, not the relative binding amounts between the datasets. This means that when comparing a given location between different experimental conditions the gain or loss of the factor at the site can be identified, if it is detected in one dataset and not the other, but a change in the level of binding between the two, if present in both, cannot be determined, because there is no suitable method of normalising them to allow these comparisons to be made. Comparisons of binding levels can be made between different locations within the same dataset, but these cannot be reliably compared to other datasets.

This inability to reliably compare relative binding levels between different ChIP-chip datasets severely limits the potential of the technology and has been highlighted previously (Cesaroni et al., 2008, for example). Much more information than the simple presence or absence of a binding site is present in datasets, but this information cannot be utilised without a suitable normalisation procedure. To address this problem, a novel normalisation method has been developed here which allows any number of ChIP-chip datasets from the same experiment to be normalised together to allow comparisons to be made between them, in a manner similar to existing approaches for transcription array datasets. This chapter describes this procedure, which is based on the quantile normalisation procedure (Bolstad et al., 2003). This chapter is written with specific reference to the Agilent yeast G4493A microarray, but is equally applicable to data from any ChIP-chip microarray, and has also been used with data from other platforms.

4.2 Algorithm

The normalisation process has been written as several different functions, described in this section, which work with certain assumptions of the data being analysed.

4.2.1 Expectations of the data

This normalisation process, and the peak detection process described in the next chapter, is based on certain assumptions about the data being analysed, in order to determine which probes have values representing enrichment and which have values representing background noise. These assumptions are that the data are split into two subpopulations and that the estimated background region of the data forms a normal distribution distinct from the distribution of the enriched region. This is the 'perfect' scenario, but in many cases data may not exactly conform to these assumptions. In these situations the normalisation and peak detection may not perform well, or may not be able to be applied at all. Figure 4.1 shows representations of some different possible ChIP-chip data distributions. The skewed distribution (A) is the most commonly described ChIP-chip distribution, which has the majority of the data points in the background sub-population, with a small proportion of enriched values which create a tail on the right hand side of the distribution of the background population. This is most often seen with protein binding datasets, where a relatively small proportion of probes is enriched. The Abf1 binding datasets presented in Chapter 7 provide examples of these types of distributions (Figure 7.2). Bimodal distributions may be created when the enriched sub-population is larger, where this enriched sub-population may have a lower (B) or higher density (C) than the background sub-population. This is often seen with datasets measuring epigenetic modifications that occur throughout a genome. The histone acetylation datasets presented later in this chapter provide examples of this type of distribution (Figure 4.8). The background sub-population cannot be distinguished from the enriched sub-population if all, or the vast majority, of the data points represent the enriched sub-population (D). The CPD datasets presented in Chapter 6 provide

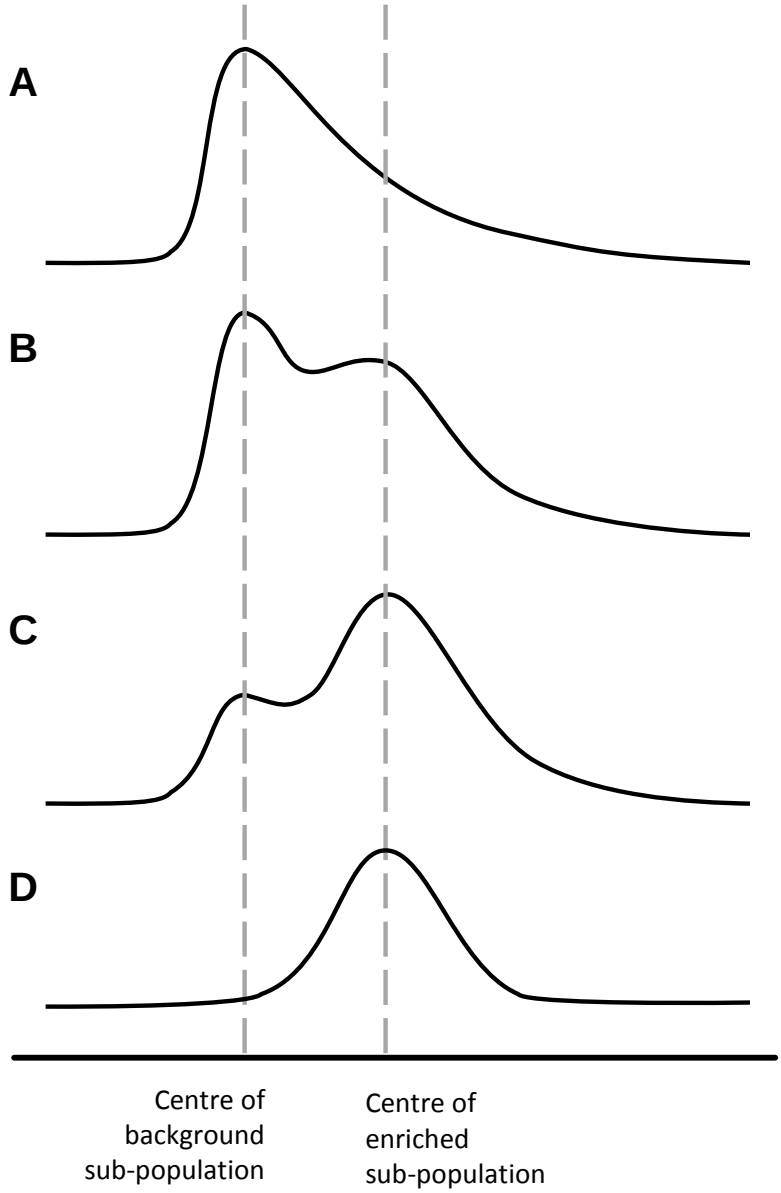


Figure 4.1: Examples of data distributions: ChIP-chip data may form skewed (A), bimodal (B and C) and symmetrical (D) distributions as a result of varying proportions of probes falling into the background and enriched sub-populations.

examples of this type of distribution (Figure 6.1). In this case the normalisation procedure described in this chapter cannot be applied. A potential alternative is outlined in Section 4.4.

4.2.2 Overview

The normalisation procedure is performed on data in an `arrayData` object in the following stages:

1. Remove irrelevant probe values.
2. Remove absent probe values.
3. Quantile normalise replicate datasets.
4. Shift all datasets' pseudo-modal points to zero (the pseudo-modal shift).
5. Scale all datasets' estimated background regions to the standard normal distribution (the background scaling).

Each is written in a separate function, meaning procedures are always carried out in the correct order and can be omitted if required. All are called by the `normalise` function (Script 4.1), which has the following arguments:

object The `arrayData` object to be normalised (no default).

batches A list specifying replicate datasets to be normalised together (no default). All datasets are treated as replicates if not specified.

reorder A logical vector specifying whether or not to reorder data according to the order in the "batches" argument, which ensures replicate datasets are together (default `TRUE`).

rmRegions A logical vector specifying whether or not to call the function to remove irrelevant probe values (default `TRUE`).

regions A three-column matrix specifying regions from which to remove probe values (no default).

rmNAValues A logical vector specifying whether or not to call the function to remove absent probe values (default `TRUE`).

quantile A logical vector specifying whether or not to call the function to perform the quantile normalisation procedure (default `TRUE`).

shift A logical vector specifying whether or not to call the function to perform the pseudo-modal shift (default `TRUE`).

customShift A numerical vector specifying the centres of the estimated background regions if these are not the pseudo-modes (no default).

scale A logical vector specifying whether or not to call the function to perform the background scaling (default `TRUE`).

rowMeans A logical vector specifying whether or not to perform normalisation on averaged datasets (default FALSE).

Script 4.1: normalise: script containing the normalisation functions.

```
## normalise function ##
## arguments: object (an arrayData object), batches (list of groups of datasets to process together), regions (regions to be deleted: passed to rmRegions function), customShift (custom shift values: passed to shiftByMode function), rmRegions (apply the rmRegions function), rmNAValues (apply the rmNAValues function), quantile (apply the quantileNormalise function), shift (apply the shiftByMode function), scale (apply the stNormScale function), reorder (reorder resulting datasets to match the order of batches)
normalise<-function(object,batches,regions,customShift,rmRegions=T, rmNAValues=T,quantile=T,rowMeans=F,shift=T,scale=T,reorder=T) { #define the function
  if (missing(batches)) batches<-list(1:ncol(object)) #if no batches are provided define 1 batch
  if (missing(customShift)) customShift<-rep(NA,length(batches)) #if no custom shifts provided define no custom shifts
  if (!is.list(batches)) stop("Batches must be a list", call.=F) #check batches is a list, stop with message if not
  if(rmRegions & missing(regions)) warning("\"regions\" missing: unable to rmRegions", call.=F) #warning message if no regions are provided for the rmRegions function
  if (rmRegions & !missing(regions)) object<-rmRegions(object,regions) #if regions are to be removed run the rmRegions function
  if (rmNAValues) object<-rmNAs(object) #if NA values are to be removed run the rmNAs function
  count<-0 #set count to 1
  object.n<-object #create arrayData object to store normalised data
  for (o in 1:length(batches)) { #loop through batches
    temp<-object[,batches[[o]]] #get data in current batch
    if(quantile) temp<-quantileNormalise(temp) #if quantile normalisation is to be performed run the quantileNormalise function
    if(rowMeans) temp<-rowMean(temp)
    if (shift) temp<-shiftByMode(temp,custom=customShift[[o]]) #if a pseudo-modal shift is to be performed run the shiftByMode function
    if (scale) temp<-stNormScale(temp) #if a background scale is to be performed run the stNormScale function
    object.n$status[batches[[o]]]<-temp$status #get updated statuses
    object.n$ratios[,batches[[o]]]<-temp$ratios #store normalised data
  }
  if(reorder) object.n<-object.n[,unlist(batches)] #reorder datasets
  return(new("arrayData",object.n)) #return normalised data
}
```

The function adds all datasets to the same batch (L4) and sets all custom shift values to NA (L5) if they are not user defined. If "batches" is not a list the function stops with a message (L6). If `rmRegions` is set to run but no "regions" argument is provided, `rmRegions` is set not to run and a warning message printed (L7). If set to run, `rmRegions` (L8) and `rmNAValues` (L9) are performed on all datasets. A count is defined (L10) and a new `arrayData` object created (L11) before a loop through the batches is initiated (L12). The data for the current batch is extracted (L13) and provided to `quantileNormalise` (L14). Data are averaged at this point if required (L15). `shiftByMode` (L16), with the corresponding "customShift" values) and `stNormScale` (L17) are run when required. Data are reordered if required (L21) and finally returned (L22).

4.2.3 Preprocessing

Before the data can be normalised it must be prepared for the process. Preprocessing is a generic term given to the small changes made to datasets to prepare them for the main processing to take place. The phrase 'garbage in, garbage out' is commonly used in computing as a way of highlighting the fact that a computer will only return correct results if correct data is first input. This initial pre-processing step is required to ensure that only the correct data is passed on to the next processing stage and therefore that the results produced in the subsequent analyses are correct. There are three stages to the preprocessing of the G4493A microarray datasets, all of which remove irrelevant data points so as to 'tidy' the datasets.

4.2.3.1 Removing irrelevant probe values

The first stage is to remove all probes corresponding to irrelevant data. On the G4493A microarray these are probes for the mitochondrial genome and a selection of genes deleted in the yeast strain BY4742. This is performed by the `rmRegions` function (Script 4.2) which has the following arguments:

object The `arrayData` object to be processed (no default).

deleteRegions The three-column matrix supplied by the "regions" argument of the `normalise` function, containing the chromosome number and coordinates of the region(s) to be removed.

Script 4.2: rmRegions: script to remove specified regions from an `arrayData` object.

```
1  ## rmRegions function ##
2  ##arguments: object (an arrayData object), deleteRegions (a matrix of
       coordinates of regions to remove)
3  rmRegions<-function(object,deleteRegions) { #define function
4    remove<-numeric() #initialse vector to store probes to remove
5    for (n in 1:nrow(deleteRegions)) { #loop through coordinates of regions to
         be removed
6      remove<-c(remove,which(object$coordinates[,1] == deleteRegions[n,1] &
           object$coordinates[,2] <= max(deleteRegions[n,2:3]) & object$
           coordinates[,3] >= min(deleteRegions[n,2:3]))) #get positions of
           probes to be deleted
7    }
8    for (n in 1:ncol(object)) { #loop through datasets
9      object$status[[n]]<-c(object$status[[n]],"rmRegions") #add 'rmRegions'
           to arrayData status
10     object$status[[n]]<-object$status[[n]][!object$status[[n]] == "raw"] #
           remove 'raw' from status
11   }
12   if (length(remove) > 0) { #if some probes are to be deleted
13     return(new("arrayData",object[-remove,])) #remove the probes
14   }else{ #no probes are to be deleted
15     return(new("arrayData",object)) #return the original object
16   }
17 }
18
19 regions<-matrix(ncol=3,nrow=5) #regions to delete on the yeast 4x44k array:
20 regions[1,]<-c(2,473920,469742)
21 regions[2,]<-c(14,721947,722609)
22 regions[3,]<-c(3,91324,92418)
23 regions[4,]<-c(5,116167,116970)
24 regions[5,]<-c(17,0,Inf)
```

The function creates a vector to store the probe numbers to be removed (L4) and initialises a loop for each defined region to be deleted (L5), where all probes falling in the region are identified (L6). The status is updated for each dataset (L8–11) and, where probes are identified to be removed, they are taken from the returned dataset (L12–16). The matrix used to remove regions from the yeast microarray is shown (L19–24).

Removing probe values for the mitochondrial genome

The G4493A microarray contains 295 probes for the yeast mitochondrial genome. These probes do not hybridise to mitochondrial DNA as well as the genomic probes do to genomic DNA, shown with box plots of the IP sample (red) and input sample (green) genomic and mitochondrial probes from microarrays measuring CPDs (Figure 4.2, see Chapter 6 for further information on the data). The average binding levels of the mitochondrial probes are several orders of magnitude less than the genomic probes. There is also a greater range of data values: the average standard deviation values of the 0 hr genomic, 0 hr mitochondrial, 2 hr genomic and 2 hr mitochondrial $\log_2$ ratios are 0.2, 1.5, 0.1 and 3.4 respectively. There is also more variation in the range of standard deviation values for the mitochondrial probes than the genomic. This suggests greater variability in the binding of mitochondrial DNA to their probes than genomic DNA, making the values unreliable.

The reason for this inconsistency is not known, but is most likely due to the GC content of the mitochondrial genome which is, at an average of 17.2%, much lower than the genomic DNA, at an average of 37.9% (Figure 4.3). This low GC content has two important implications in the context of the microarray assays. Firstly, it can affect the two PCR amplification steps carried out during the DNA preparation stage, as well as the labelling reaction, which is eqivalent to a single PCR cycle. It is known that PCR efficiency is affected by GC content, due to this causing variations in the melting temperature of the dsDNA (Marmur and Doty, 1962). It is possible to modify the PCR reaction to better suit this low GC content DNA (Kramer and Coen, 2001, for example), but at the expense of the genomic DNA and so full amplification of all DNA together is not possible. Secondly, the low GC content of the probes and complementary DNA sequences means that they will not hybridise as efficiently to the microarrays. The bonding between G and C nucleotides is via three hydrogen bonds whereas there are two between A and T nucleotides. The bond between A and T is therefore weaker than that between G and C. The abundance of A and T nucleotides means that the binding of mitochondrial DNA to the corresponding probes

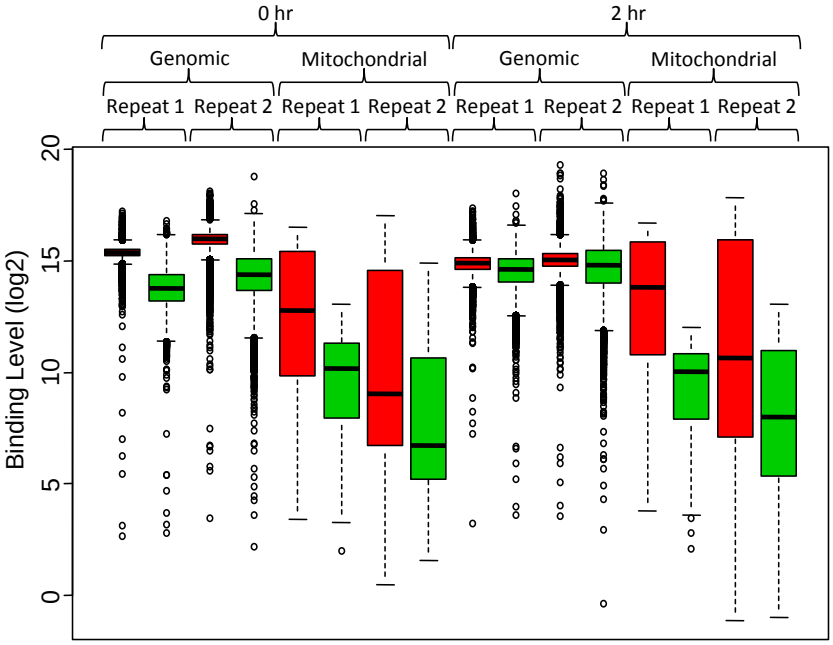


Figure 4.2: Mitochondrial probe binding values: Box plots showing the range of fluorescence values of genomic and mitochondrial probes from 4 microarrays measuring CPD incidence. Red shows IP sample values, green input sample. The mitochondrial probes show lower, more varied binding values than the genomic probes.

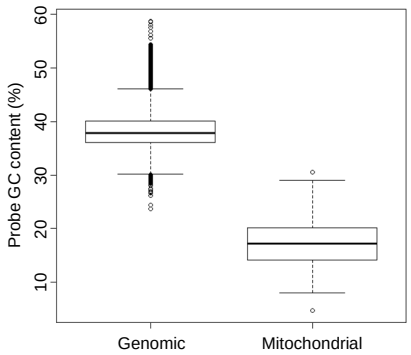


Figure 4.3: Probe GC contents: Box plots showing the range of GC content (%) of yeast genomic and mitochondrial probes. The mitochondrial probes show lower values than the genomic probes, reflecting the overall content.

will not be as strong as the genomic DNA, potentially resulting in a smaller amount of bound DNA, which will give a lower fluorescence signal, which is known to be the case with oligonucleotide micorarrays (Heller, 2002).

The high AT content — as a result of the low GC content — means there exists a greater potential for the formation of CPDs, as there will be a high number of TT dinucleotide sequences. This in turn should mean that the immunoprecipitated binding values from the mitochondria should be higher than the rest of the genome. In addition, there are multiple mitochondria per cell, meaning there are many more copies of mitochondrial than genomic DNA. This increased copy number should give higher signals to the mitochondrial probes than the rest of the genome. This is not the case, shown in Figure 4.2, which further suggests the mitochondrial probes do not provide an accurate representation of UV damage in these datasets.

This idea was confirmed by predicting the binding levels from the genome sequence, as has been done for genomic DNA (Chapter 6). This showed a good correlation between predicted and actual values with a Spearman's rank correlation coefficient of 0.77, indicating the microarray results are a genuine representation of what is occurring in the cell. The same procedure applied to the mitochondrial genome sequence shows no correlation between the predicted and actual values with a Spearman's rank correlation coefficient of -0.01. Taking only the coding regions of the genome, which have a higher GC content than the non-coding regions, does not improve the correlation, with a Spearman's rank correlation coefficient of 0.03.

Taken together these results suggest that the probes for the mitochondrial genome do not provide reliable data for the datasets produced here and therefore these probe values have been removed from all datasets analysed.

Removing probe values for deleted genes

The G4493A microarrays contain probes covering the whole yeast genome. However, many laboratory strains have some genes deleted, the corresponding probes for which need to be removed from analyses. The BY4742 yeast strain most commonly used in our laboratory has four deleted genes: Lys2;

His3; Leu2; and Ura3, together covering 6,741 bp. There are 26 probes on the G4493A microarray complementary to sites within these regions.

4.2.3.2 Removing absent values

The data analysed from the microarrays is calculated as the logarithm (base 2) of the red:green signal intensity ratios. This conversion to logarithmic space can create absent values if either the red or green signal is negative, as the logarithm of a negative number is not possible. This can arise after the background subtraction has taken place, if the background intensity is greater than the probe intensity. All data for probes across a set of datasets where at least one has such a missing value are removed, so that all probes have a full complement of replicates. Any statistics applied to the arrays later on can therefore be interpreted at the same level. This should not remove more than a small number of probes from the datasets and so will not adversely affect the remaining analyses. The removal of many probes may suggest a problem with one or more datasets which would require further investigation before proceeding with the rest of the normalisation procedure. The `rmNAs` script (Script 4.3) performs this function, which has the following argument:

object The `arrayData` object to be processed (no default).

Script 4.3: rmNAs: script to remove absent values across `arrayData` datasets.

```
1  ## rmNAs function ##
2  ## arguments: object (an arrayData object)
3  rmNAs<-function(object) {#define function
4    NAs<-numeric() #initialise vector to store probes
5    for (n in 1:ncol(object)) { #loop through datasets
6      NAs<-c(NAs,which(is.na(object$ratios[,n]))) #get probes with NA values
7    }
8    NAs<-unique(NAs) #get unique probes
9    if(length(NAs) > 0) object$ratios[NAs,]<-NA #replace all NA probe values
         with NAs
10   message(paste(length(NAs),"NA probes")) #print the number of probes
         removed
11     for (n in 1:ncol(object)) { #loop through datasets
12       object$status[[n]]<-c(object$status[[n]],"rmNAs") #add 'rmNAs' to
             arrayData status
13       object$status[[n]]<-object$status[[n]][!object$status[[n]] == "raw"] #
             remove raw from status
14     }
15   return(new("arrayData",object)) #return data
16 }
```

The function creates a vector to store probes identified as containing NA values (L4) and loops through each dataset to find these probes (L5–7). Identified probes have all ratio values set as NA (L9) and a message is printed showing the number of these probes found (L10). The status of all datasets is updated (L11–14) and the new object returned (L15).

4.2.4 Full processing

The full normalisation procedure begins once the pre-processing has been completed. Pre-processing is applied to all datasets irrespective of their nature (inter-dataset normalisation). For the reasons discussed previously the full normalisation procedure cannot be applied in this way and so the following functions are applied to groups of replicate datasets separately, as defined by the "batches" argument of the `normalise` function (intra-dataset normalisation).

4.2.4.1 Quantile normalisation

Quantile normalisation is performed by the `quantileNormalise` function (Script 4.4), using the `preprocessCore` package which implements the method described by Bolstad et al. (2003). Briefly, the quantile normalisation procedure aims "to make the distribution of probe intensities for each array in a set of arrays the same" (Bolstad et al., 2003). This means that all normalised datasets are transformed to follow the same distribution, which can be thought of as the average distribution of all the datasets. The procedure is demonstrated with some randomly generated example data consisting of 3 replicates and 10 points (Table 4.1, plotted in Figure 4.4.).

Each column of the raw data is sorted from largest to smallest values and the mean of each of these new rows is calculated (Table 4.2).

The values of each position in the row are replaced with the new mean value and each column is reordered to its original order in the raw data (Table 4.3).

This procedure has the effect of bringing the replicate values closer together, thus reducing the variation between datasets (Figure 4.4A, compare

Position	Replicate 1	Replicate 2	Replicate 3
1	1.66	1.88	3.36
2	1.44	1.01	4.56
3	2.80	2.22	4.25
4	2.21	2.26	5.95
5	3.14	3.83	5.24
6	3.70	3.37	5.97
7	3.12	3.41	4.43
8	2.54	2.82	4.38
9	1.73	2.54	3.45
10	1.92	1.64	3.12

Table 4.1: Quantile normalisation example data: Three replicate sets of ten randomly generated values, representing values from a microarray containing ten probes. Plotted in Figure 4.4A and B.

	Replicate 1		Replicate 2		Replicate 3	**Mean**
2	1.44	2	1.01	10	3.12	**1.86**
1	1.66	10	1.64	1	3.36	**2.22**
9	1.73	1	1.88	9	3.45	**2.35**
10	1.92	3	2.22	3	4.25	**2.80**
4	2.21	4	2.26	8	4.38	**2.95**
8	2.54	9	2.54	7	4.43	**3.17**
3	2.80	8	2.82	2	4.56	**3.39**
7	3.12	6	3.37	5	5.24	**3.91**
5	3.14	7	3.41	4	5.95	**4.17**
6	3.70	5	3.83	6	5.97	**4.50**

Table 4.2: Quantile normalisation example processing: Each replicate sorted from smallest to largest, with its original position from Table 4.1 shown in red and the mean for each row shown in bold.

black and red lines). One of the example datasets was deliberately given larger values to demonstrate this point. It follows that this procedure also reduces the standard deviation of replicates that are not initially similar (Figure 4.4B, compare black and red lines). It also gives each dataset the same statistical properties, including the same distribution. This has the effect of giving the replicate datasets the same density profile (Figure 4.4C, compare black and red lines).

In performing these numerical manipulations the quantile normalisation procedure removes a large proportion of any variations between replicate datasets. This is demonstrated by the reduction in the size of the error bars in Figure 4.4B. The example data generated for this demonstration consists of 3 replicates, one of which is deliberately different from the others (Table 4.1 and Figure 4.4A) to demonstrate the ability of the normalisation procedure to remove this difference. This is representative of real ChIP-chip data, which are rarely similar before normalisation, hence the need to apply a normalisation procedure. Figure 4.4B shows that the means of the datasets are not greatly changed by the procedure.

Figure 4.5 shows Q-Q plots of the raw and quantile normalised data. As expected, all normalised data follow the same distribution, evidenced by all their points lying on the line $y = x$.

Quantile normalisation is applied to each set of replicate datasets (intra-dataset normalisation) to remove variations between theoretically identical datasets. Figure 4.4C demonstrates why this procedure cannot be applied to datasets from different biological conditions. If, for example, the third example dataset (which is deliberately shown with larger values than the other two datasets to demonstrate the normalisation procedure) was from a different experimental condition and as a result had higher values due to a change in biology, a normalisation method should not seek to remove this, but maintain these differences between the two. Quantile normalisation does remove these differences, by creating a new distribution somewhere between the initial distributions. This not only reduces the larger values but increases the smaller values, which is detrimental to both sets of results and removes the biological relevance. This is why groups of replicate datasets are provided

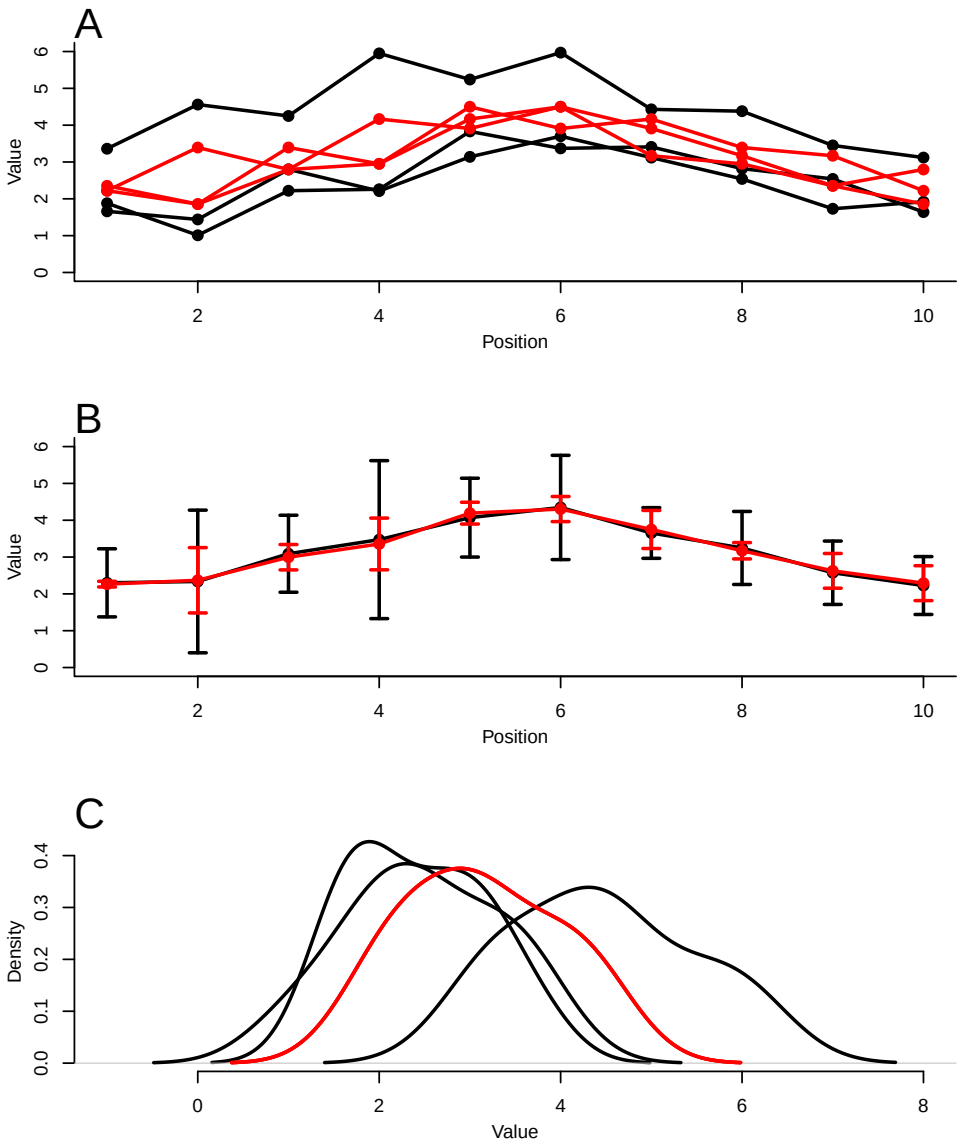


Figure 4.4: The effect of quantile normalisation on data: A: The example data from Table 4.1 (black) and the quantile normalised data from Table 4.3 (red). B: Means of the example data from Table 4.1 (black) and the quantile normalised data from Table 4.3 (red). Error bars represent ±1 standard deviation. C: Density profiles of the example data from Table 4.1 (black) and the quantile normalised data from Table 4.3 (red). The quantile normalisation procedure makes all datasets follow the same distribution, hence the appearance of a single red line.

Position	Replicate 1	Replicate 2	Replicate 3
1	2.22	2.35	2.22
2	1.86	1.86	3.39
3	3.39	2.80	2.80
4	2.95	2.95	4.17
5	4.17	4.50	3.91
6	4.50	3.91	4.50
7	3.91	4.17	3.17
8	3.17	3.39	2.95
9	2.35	3.17	2.35
10	2.80	2.22	1.86

Table 4.3: Quantile normalisation example result: The original data replaced with the mean values from Table 4.2 and reordered to the original order in Table 4.1 to create quantile normalised data. Plotted in Figure 4.4A and B.

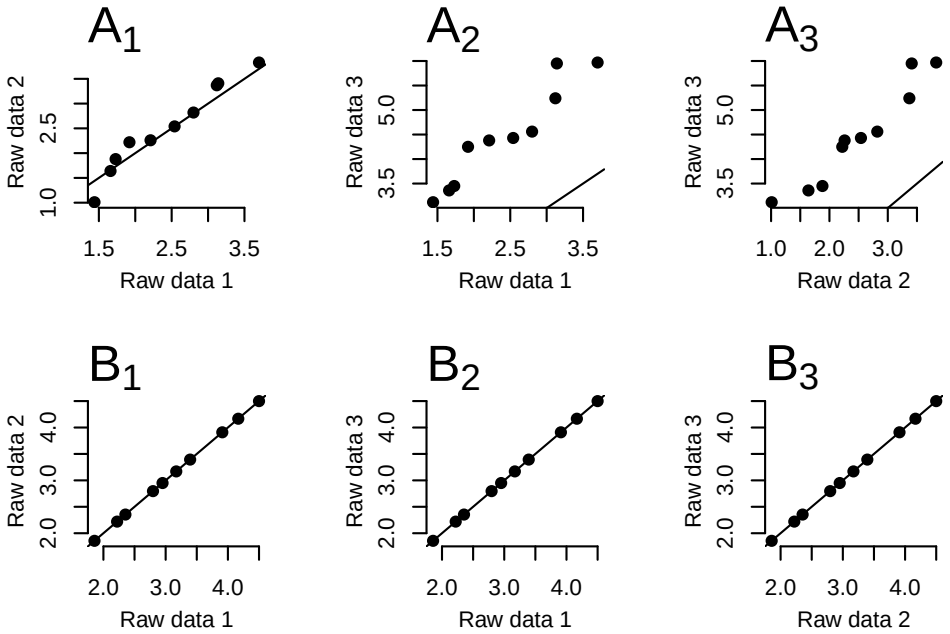


Figure 4.5: Example data quantile-quantile plots: A: Raw data do not follow the same quantiles. B: quantile normalised data follow the same quantiles. Lines show $y = x$.

separately. The function has the following arguments:

object The `arrayData` object to be processed (no default).

Script 4.4: quantileNormalise: script to apply the `preprocessCore` `normalise.quantiles` function to an `arrayData` object.

```
1  ## quantileNormalise function ##
2  ## arguments: object (an arrayData object)
3  quantileNormalise<-function(object) { #define function
4    require(preprocessCore) #load package containing qualtile normalisation
         function
5    object$ratios<-normalize.quantiles(object$ratios) #perform quantile
         normalisation
6      for (n in 1:ncol(object)) { #loop through datasets
7        object$status[[n]]<-c(object$status[[n]],"quantileNormalise") #add '
             quantileNormalise' to status
8        object$status[[n]]<-object$status[[n]][!object$status[[n]] == "raw"] #
             remove 'raw' from status
9      }
10   return(new("arrayData",object)) #return quantile normalised data
11 }
```

The function loads the `preprocessCore` package if not already present (L4) and performs the quantile normalisation on the `arrayData` ratio values (L5). The status of all datasets is updated (L6–9) and the new object returned (L10).

The intra-replicate quantile normalisation process is followed by a novel inter-replicate normalisation procedure which normalises them together, whilst seeking to maintain any biologically relevant differences, allowing relative comparisons to be made between them.

4.2.4.2 Pseudo-modal shift and background scaling

The extended normalisation procedure aims to mimic the quantile normalisation procedure on only a subset of data. The subset used is an estimate of the background population of probes, that is, probes that do not represent enriched sections of the genome. This background population should therefore be a list of zero values, but the inherent variation of noise in the assay means it approximates a normal distribution. This distribution should be centred around zero, but variations in the assay mean it may not be. In an assay performed where no material is enriched, all of the probes would

fall within this approximately normal background distribution. If two such assays were performed and were to be compared, they could be quantile normalised together to form a third normal distribution, because *a priori* knowledge states that they are theoretically identical and therefore quantile normalisation is applicable. An equivalent procedure would be to shift and scale one or both distributions such that they follow the same distribution, that is, have the same mean and standard deviation. If both were exactly normally distributed they would both exactly follow the new distribution, as achieved with quantile normalisation. If they both approximate normal distributions, as they would in real data, they would approximately follow the same distribution following the shift and scale. Therefore taking two sets of background values and shifting them to centre on the same point and scaling them to have the same standard deviation makes them comparable, in the same way they would be if they were quantile normalised. This is the basis of the novel normalisation procedure presented here, where the overall distribution of values is used to estimate the background values, which are shifted to centre on 0 and scaled to a standard deviation of 1 (creating the standard normal distribution).

Real ChIP-chip data consist of two sub-populations: enriched and non-enriched probes. These two sub-populations overlap to create the previously described skewed distribution, the skew as a result of the enriched sub-population consisting of larger values than the background. The fact that most ChIP-chip datasets have this background subpopulation as well as the enriched subpopulation means that it can be used as a constant between different datasets. Shifting and scaling each background population to follow the same distribution makes them comparable, as described above. Shifting and scaling the enriched population along with them means that these also become comparable between the different datasets.

As the two subpopulations of data are not distinct, an estimate of the background subpopulation is made by calculating kernel density estimates of the data via the `density` command in R and shifting the data to centre the highest point on zero (Figure 4.6). This is equivalent to aligning the data by their modal points, but the nature of the data is such that each value is

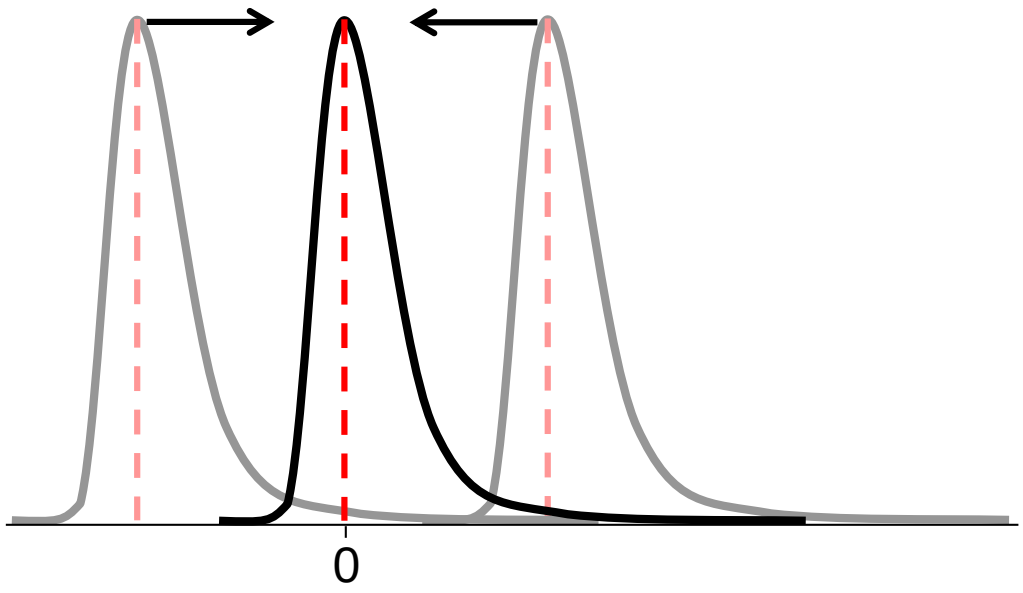


Figure 4.6: Representation of the pseudo-modal shift: Grey lines represent datasets, with their pseudo-modal points indicated with a dashed red line. The datasets are shifted, upwards or downwards, so that this point lies on zero. The black line represents a dataset following the pseudo-modal shift.

rarely replicated and so this pseudo-mode has to be calculated. This is based on the assumption that this point represents the centre of the background subpopulation. This is true if the background subpopulation consists of more than half of the probes. If more than half are enriched the highest point of the distribution represents the enriched subpopulation, but the peak of the background subpopulation can still be estimated if the distribution is bimodal. If a large proportion of probes are enriched and the peak of the background subpopulation cannot be estimated then this method cannot be applied.

A Gaussian smoothing kernel is used to create the density plot from which the pseudo-mode is predicted. Kernel density estimation is non-parametric, that is, it does not assume any underlying distribution in the data. This is important as there can be a lot of variation in the distributions of different datasets and the distribution of any given dataset is not known. A histogram is a simple form of non-parametric density estimation, which uses counts of data in a number of bins of set width throughout the dataset. The smoothing estimate uses a mathematical formula to create the density, which is composed of a number of kernels, one at each data point. In the case of Gaussian smoothing, each follows a Gaussian distribution. The overall density is the total of these kernels. R provides 6 other smoothing methods: epanechnikov; rectangular; triangular; biweight; cosine; and optcosine, which use varying distributions as the kernels. All of these methods provide similar or identical pseudo-modes to the Gaussian method for the H3Ac and Gcn5p binding datasets ("Kernel density estimates.pdf" file in electronic appendix; see Page 367). Where there are variations between the different pseudo-modes, the Gaussian method always produces the most common result for these datasets, showing it is the most appropriate method to use. However, the variations between the different methods are so small that the impact on the final result would be negligible alongside the other sources of variation.

The shifting of the data is achieved via the `shiftByMode` function (Script 4.5), which has the following arguments:

object The `arrayData` object to be processed.
custom A value from the distribution to be centred over 0, where the

pseudo-mode is not the centre of the estimated background population.

Script 4.5: shiftByMode: script to shift an `arrayData` object to centre its pseudo-modal value on zero.

```
1  ## shiftByMode function ##
2  ## arguments: object (an arrayData object), custom (a value to shift to zero
       if not the psedo-mode)
3  shiftByMode<-function(object,custom=NA) { #define function
4    if (missing(custom)) custom<-NA #check for custom shift
5    for (m in 1:ncol(object)) { #loop through datasets
6      if (is.na(custom)) { #if no custom value specified
7        shift<-density(object$ratios[,m],na.rm=T)$x[which.max(density(object$
             ratios[,m],na.rm=T)$y)] #find pseudo-mode
8      }else{ #custom value specified
9        shift<-custom #get custom shift value
10     }
11     object$ratios[,m]<-object$ratios[,m]-shift #shift dataset by appropriate
           amount
12   }
13     for (n in 1:ncol(object)) { #loop through datasets
14       object$status[[n]]<-c(object$status[[n]],"shiftByMode") #add '
             shiftByMode' to status
15       object$status[[n]]<-object$status[[n]][!object$status[[n]] == "raw"] #
             remove 'raw' from status
16     }
17   return(new("arrayData",object)) #return pseudo-modal shifted data
18 }
```

The function sets the custom shift value to NA if none is provided (L4) and initiates a loop through the datasets (L5). Where no custom shift is defined the `density` function is used to identify the pseudo-mode of the dataset which defines the value to shift the data by, otherwise the custom value is used (L6–10). The dataset is then shifted by this amount (L11). The status of all datasets is updated (L13–16) and the new object returned (L17).

The pseudo-modal shift takes account of additive differences between different datasets. For example, if one dataset has all intensity values a number of units greater than those of another dataset, the shift would correct this by bringing both datasets to the same background level such that the differences in intensity values are removed. It cannot however remove multiplicative (scale) differences which occur at increasing or decreasing levels throughout a dataset. Smyth and Speed (2003) point out the need to take account of these potential scale differences,

"Sometimes there are substantial scale differences between mi-

> croarrays, because of changes in the photomultiplier tube settings of the scanner or for other reasons. In these circumstances it is useful to scale-normalize between arrays."

This is the function of the background scaling part of the normalisation process, performed by the `stNormScale` function (Script 4.6). Mirroring the left had side of the background population (all of the negative values following the shift to zero) onto the right hand side (Figure 4.7 inset) allows an estimate of the standard deviation to be calculated. A scale factor is then calculated as $1 \div$ the standard deviation of this mirrored data. The whole dataset is multiplied by this factor to scale it to the required distribution (Figure 4.7). The background subpopulation then approximates the same (standard normal) distribution for all datasets and comparisons of enriched values can be made relative to this constant. The function has the following argument:

object The `arrayData` object to be processed.

Script 4.6: stNormScale: script to scale the mirrored background distribution to approximate the standard normal distribution.

```
## stNormScale function ##
## arguments: object (an arrayData object)
stNormScale<-function(object) { #define function
  for (n in 1:ncol(object)) { #loop through datasets
    if ("shiftByMode" %in% object$status[[n]]) { #if dataset has been
        shifted by its mode
      ratios<-object$ratios[,n] #get ratios
      ratios.low<-ratios[which(ratios < 0)] #get ratio values below mode
      SD<-sd(c(ratios.low,abs(ratios.low))) #calculate standard deviation of
          mirrored data
     object$ratios[,n]<-object$ratios[,n]*(1/SD) #multiply dataset by factor
         to make background standard deviation 1
     object$status[[n]]<-c(object$status[[n]],"stNormScale") #add '
         stNormScale' to status
     object$status[[n]]<-object$status[[n]][!object$status[[n]] == "raw"] #
         remove 'raw' from status
    }else{
      warning(paste("shiftByMode has not been applied to dataset ",colnames(
          object$ratios)[n],": stNormScale not applied"), call.=F) #warning
          message if dataset has not been shifted by its mode
    }
  }
  return(new("arrayData",object)) #return background scaled data
}
```

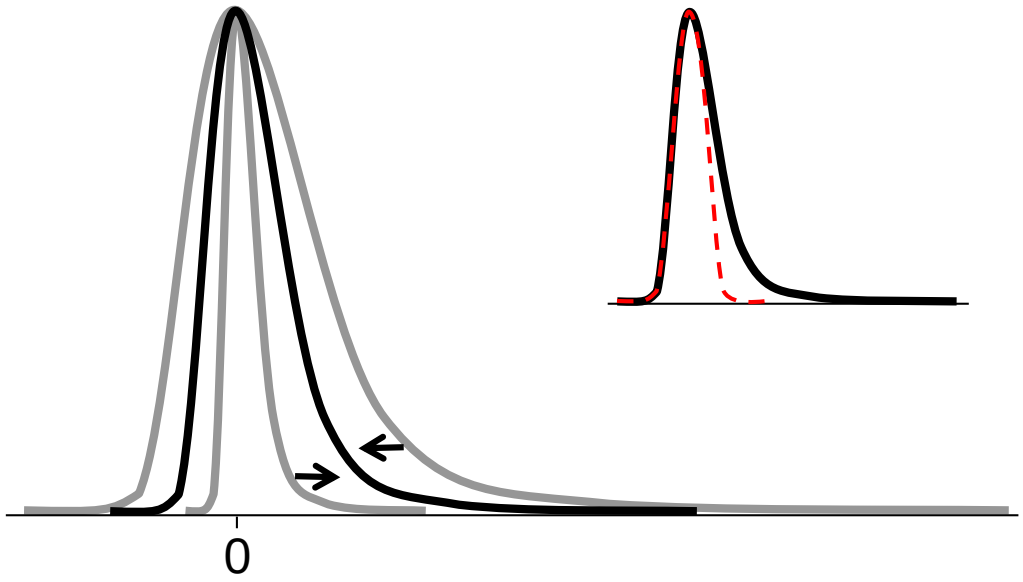


Figure 4.7: Representation of the background scaling: Grey lines represent datasets to be scaled, with arrows indicating whether they should be made larger or smaller. The black line represents the datasets following scaling, where the estimated background values follow the standard normal distribution. Inset: the standard normal distribution (red dashed line) approximates the background values of the shifted and scaled data (black line).

The function initiates a loop through the datasets (L4) and checks that each one has had the `shiftByMode` function applied (L5). If so, the ratio values are extracted (L6) and those below zero taken (L7) and mirrored to find the standard deviation (L8). The whole dataset is then multiplied by the reciprocal of this value (L9). The status of the dataset is updated (L10–11). If the dataset has not had the `shiftByMode` function applied a warning message is printed and the scale is not applied (L12–13). The new object is returned (L16).

4.3 Application

The full normalisation procedure was applied to two sets of ChIP-chip data: 10 histone H3 acetylation (H3Ac) datasets under two conditions: no UV treatment (0, 5 replicates) and 60 minutes after UV treatment (60, 5 replicates); and 11 datasets of Gcn5p binding (Gcn5p) under four conditions: no UV treatment (U, 3 replicates), immediately after UV treatment (0, 3 replicates), 15 minutes after UV treatment (15, 2 replicates) and 60 minutes after UV treatment (60, 3 replicates). These datasets were generated by Dr. Katie Evans and Dr. Richard Webster and their full analysis is presented in Evans (2011). All data files are available in the electronic appendix (see Page 367).

The effect of the normalisation procedure from one stage to the next for this data is shown as density curves for all datasets (Figures 4.8 and 4.9) and genome plots with the sets of untreated replicates (Figures 4.10 and 4.12) and averaged datasets from all different conditions (Figures 4.11 and 4.13). The density plots show data from each normalisation stage (dashed red lines) with the data from the previous stage (solid black lines). The genome plots show profiles of the data at each normalisation stage across a section of chromosome 5.

The preprocessing of the data has little effect on the overall shape of the distributions of the data: the red lines, showing the preprocessed data densities, follow the same pattern as the black lines, showing raw data densities (parts A of Figures 4.8 and 4.9). This is because the majority of the data remain unaltered by the process, with only a relatively small number of probes

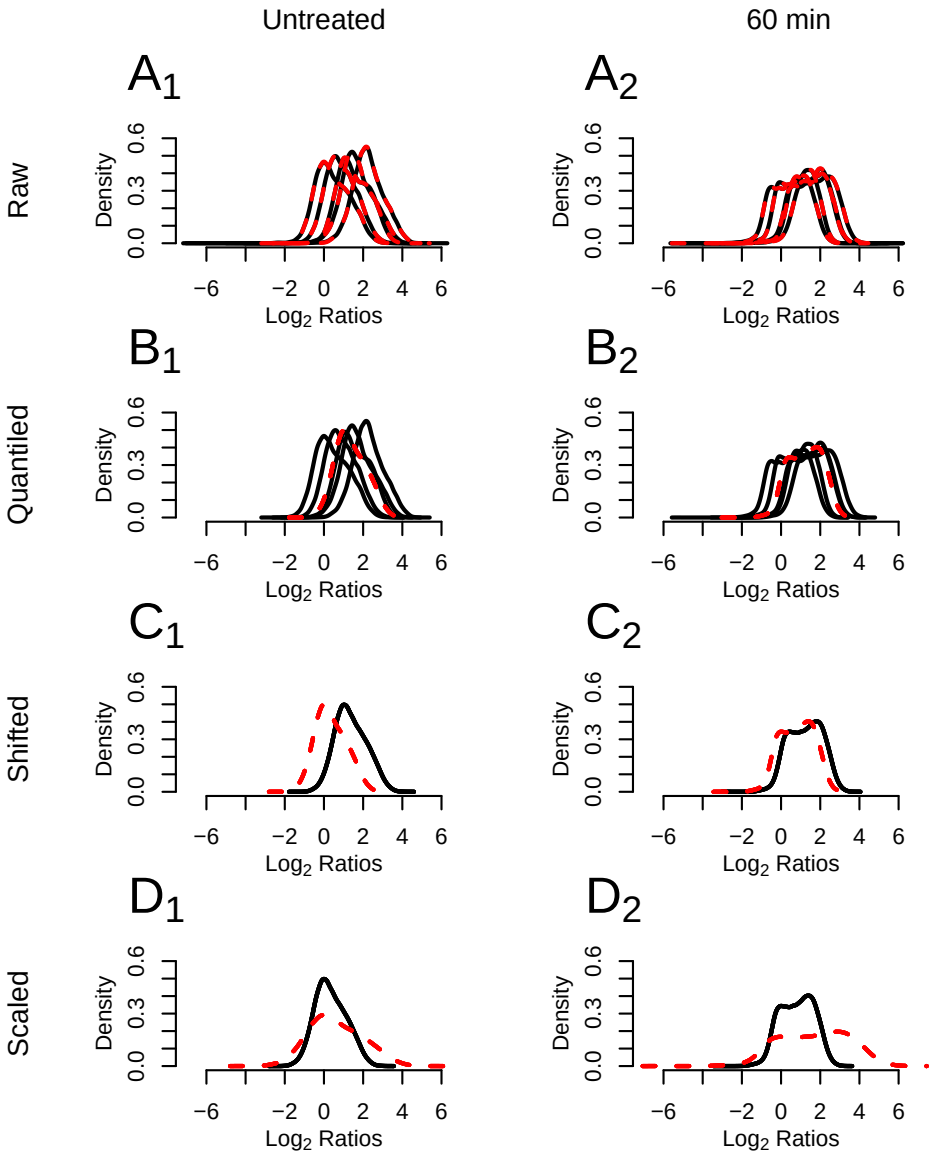


Figure 4.8: Density plots of H3Ac data undergoing normalisation: The application of preprocessing (A), quantile normalisation (B), pseudo-modal shifting (C) and background scaling (D) of no UV treatment (1) and 60 min after UV treatment (2) datasets. Black lines show data pre- and dashed red lines post-application of each step.

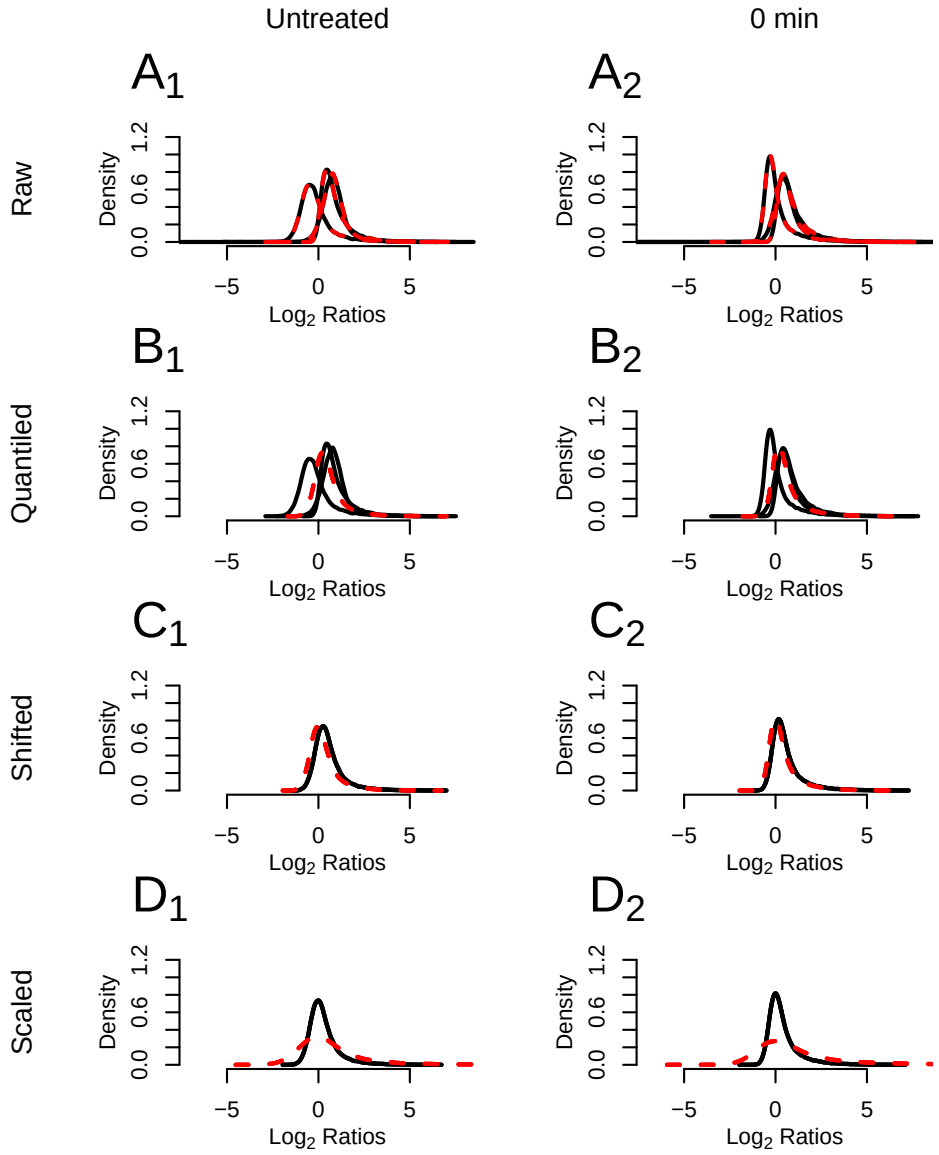


Figure 4.9: Density plots of Gcn5p data undergoing normalisation: The application of preprocessing (A), quantile normalisation (B), pseudo-modal shifting (C) and background scaling (D) of no UV treatment (1), 0 min after UV treatment (2), 15 min after UV treatment (3) and 60 min after UV treatment (4) datasets. Black lines show data pre- and dashed red lines post-application of each step. Continued on next page...

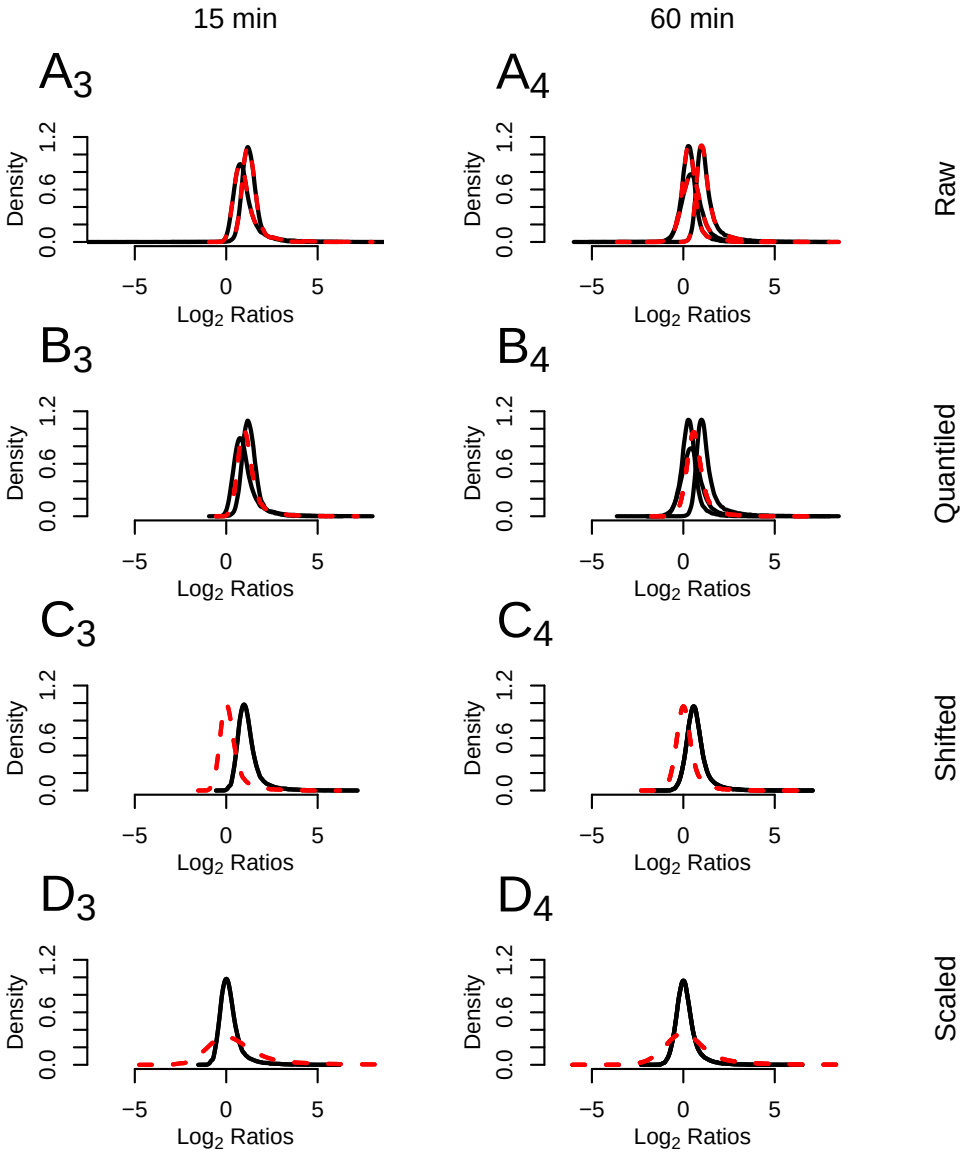


Figure 4.9: Continued from previous page.

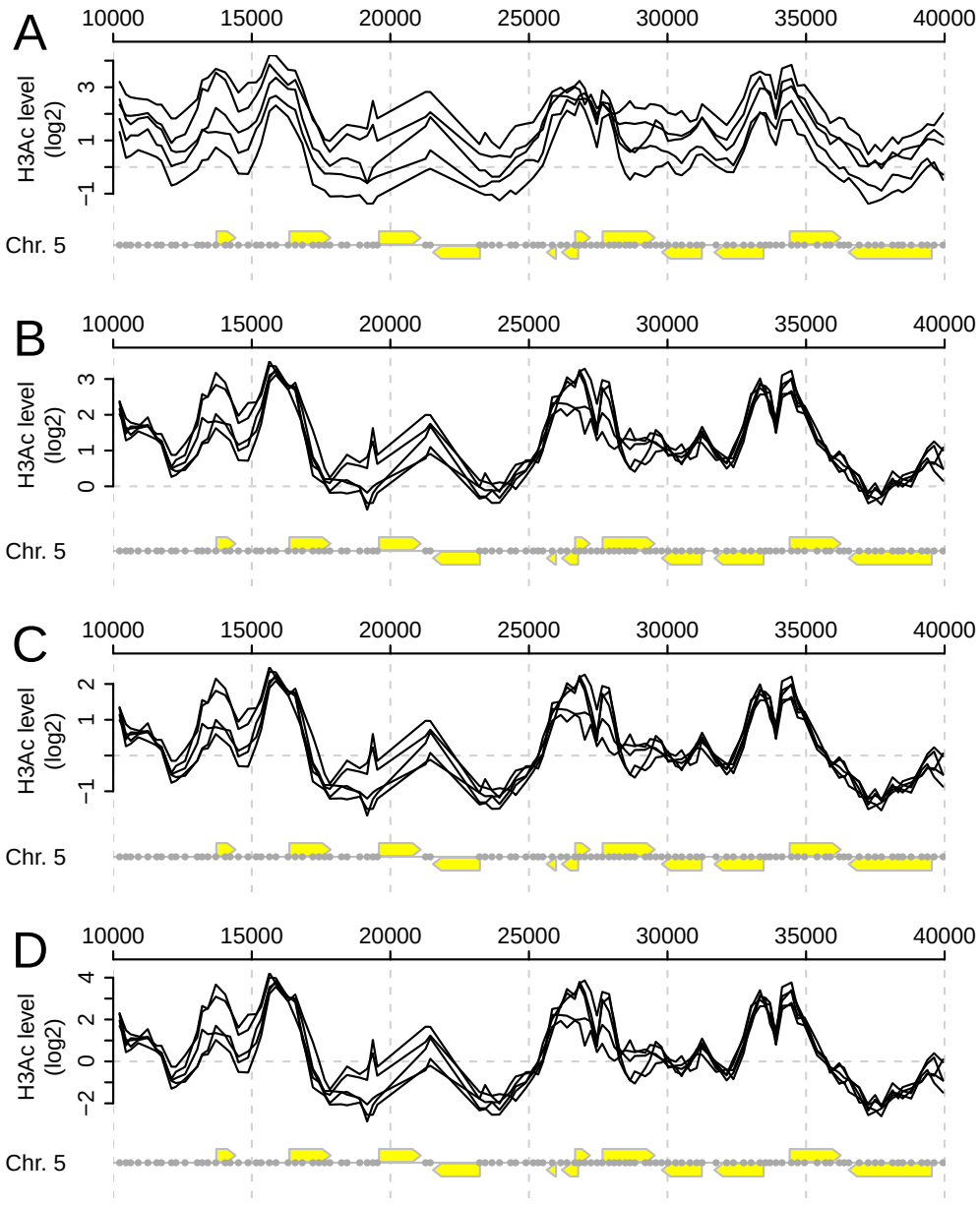


Figure 4.10: Profiles of replicate H3Ac data undergoing normalisation: Pre-processed (A), quantile normalised (B), pseudo-modal shifted (C) and background scaled (D) replicate no UV treatment data over a section of chromosome 5. Note the changes of y-axis scales between the different plots.

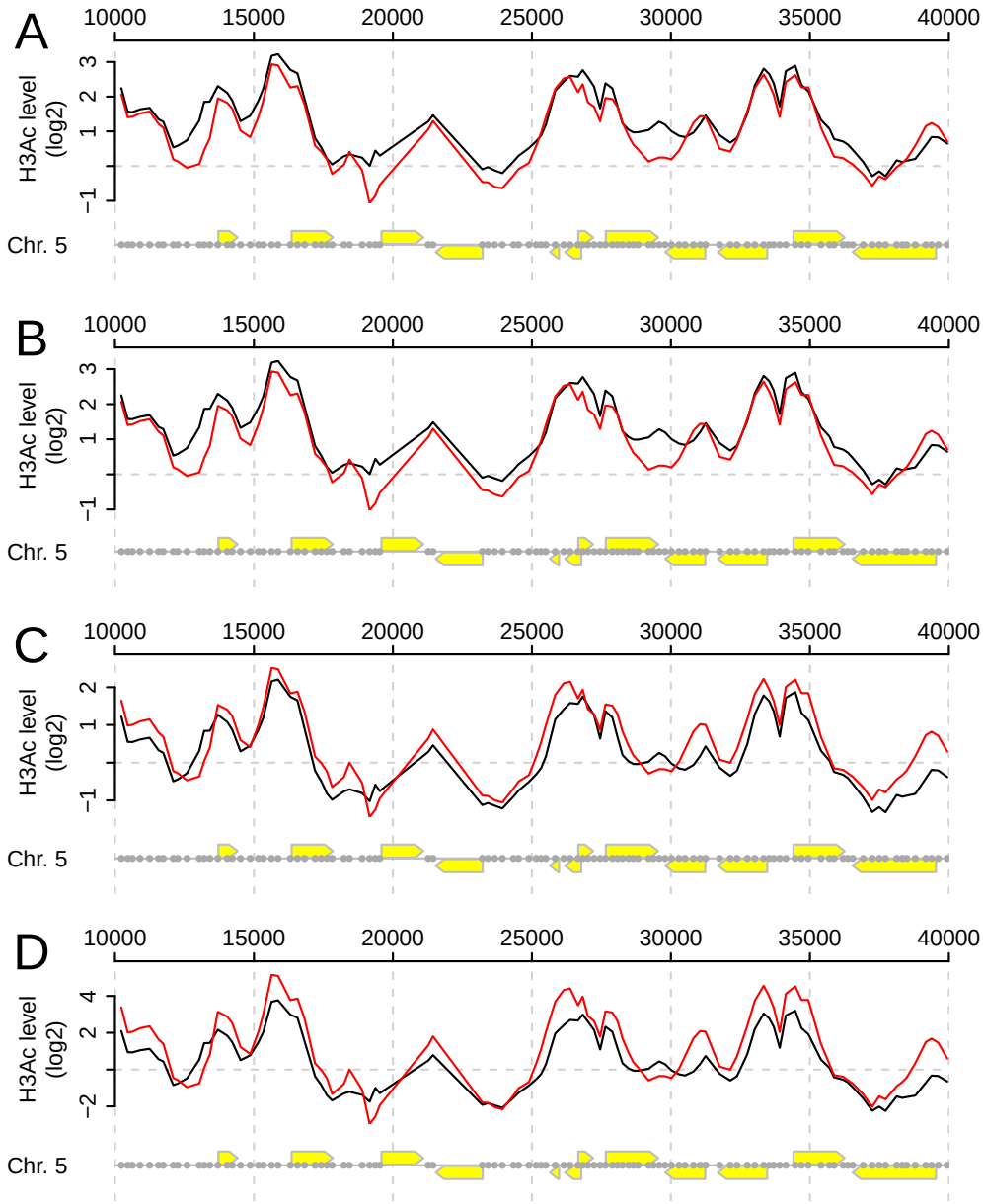


Figure 4.11: Profiles of averaged H3Ac data undergoing normalisation: Pre-processed (A), quantile normalised (B), pseudo-modal shifted (C) and background scaled (D) averaged no UV treatment (black) and 60 min post-UV treatment (red) data over a section of chromosome 5. Note the changes of y-axis scales between the different plots.

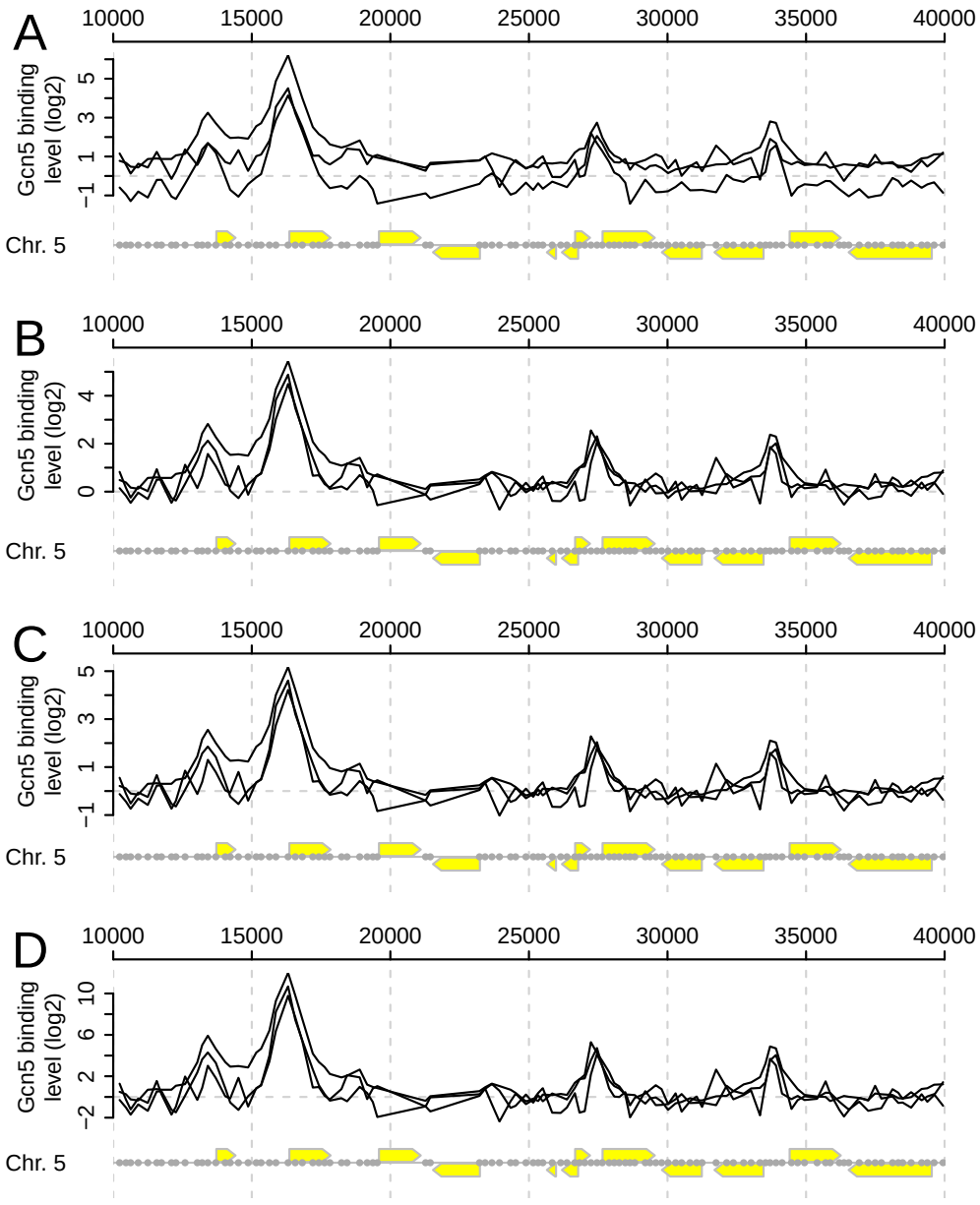


Figure 4.12: Profiles of replicate Gcn5 binding data undergoing normalisation: Preprocessed (A), quantile normalised (B), pseudo-modal shifted (C) and background scaled (D) replicate no UV treatment data over a section of chromosome 5. Note the changes of y-axis scales between the different plots.

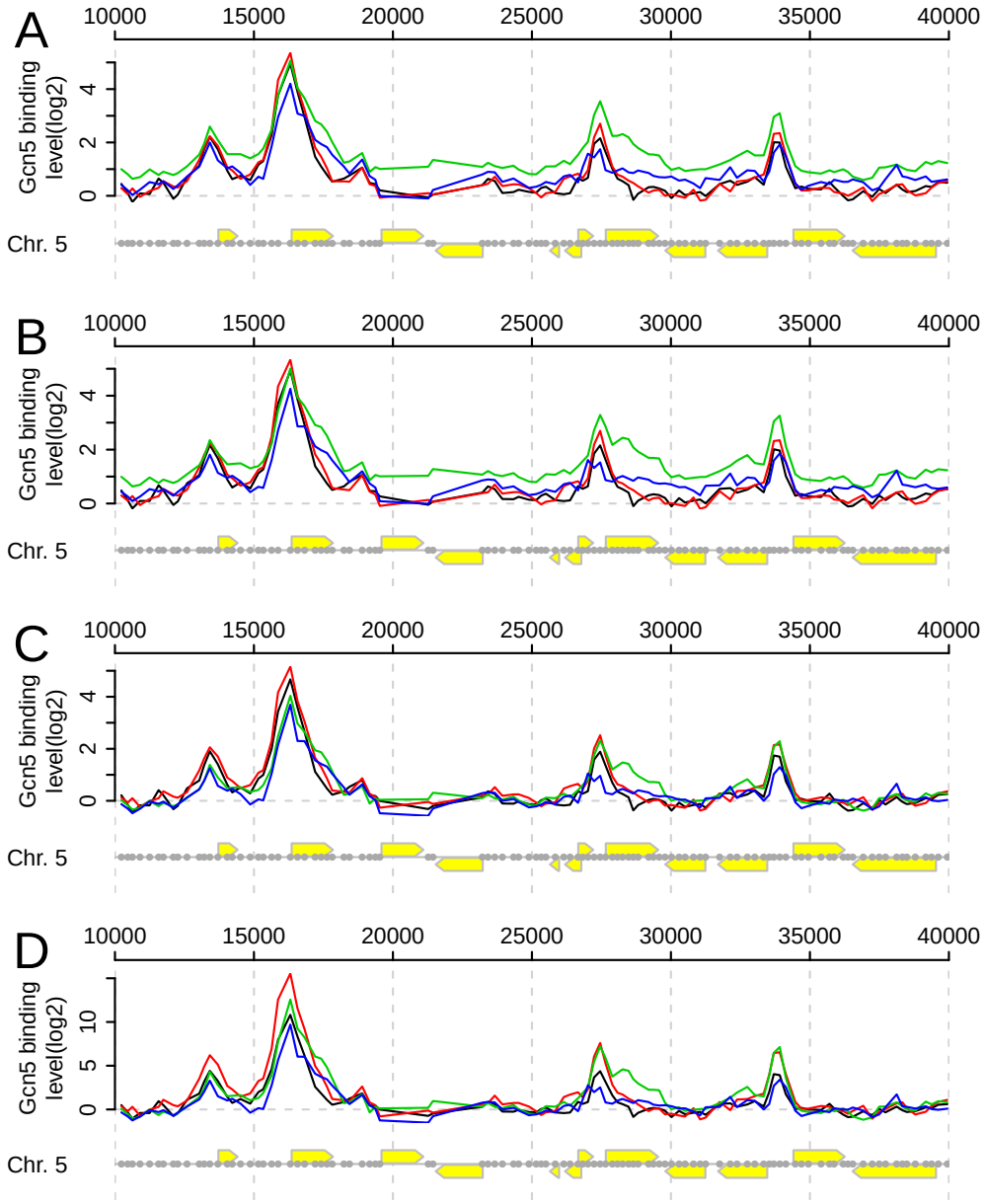


Figure 4.13: Profiles of averaged Gcn5 data undergoing normalisation: Pre-processed (A), quantile normalised (B), pseudo-modal shifted (C) and background scaled (D) averaged no UV treatment (black), 0 min post -UV treatment (red), 15 min post-UV treatment (green) and 60 min post UV treatment (blue) data over a section of chromosome 5. Note the changes of y-axis scales between the different plots.

being removed from the datasets. The range of the datasets decreases following preprocessing (the limits of the red lines are less than the black) because the quality of the probes removed is poor, meaning that they generally have outlying values at the extremes of the distributions.

The quantile normalisation has a slightly larger effect on the data, bringing all of the replicate datasets to the same distribution: all densities overlap as seen by the single red line on the plots (parts B of Figures 4.8 and 4.9). This does not however bring them all to the same profile (Figures 4.11 and 4.13), as the process is independent of genomic position. That is, although the points at any given point in the distribution are equal, these are not necessarily at identical points on the genome. Therefore while each dataset has, for example, identical maximum and minimum values, meaning the distributions start and end at the same points, these maxima and minima can (and do) appear at different probes. The process does however reduce the variation between the probes of the different datasets, shown by the profiles of the quantile normalised datasets being more similar than those of the raw data. This process does not take any variation between datasets from different experimental conditions into account.

The pseudo-modal shift can have a small or large effect on the datasets (parts C of Figures 4.8 and 4.9), depending on the original position of the pseudo-mode. Datasets with the pseudo-mode already centred at or near zero are shifted on the horizontal axis a small amount (Figure 4.9 C_1, for example, where the red line is almost on top of the black line), thereby having a small effect on the dataset. The further the pseudo-mode is from zero, the greater the effect on the dataset, as a greater shift is required (Figure 4.9 C_3, for example, where the red line is further to the left of the black line). These can be seen in the genome plots as shifts on the vertical axis. Replicate datasets are shifted by the same amount (Parts C of Figures 4.10 and 4.12), maintaining the same profile, while different datasets are shifted by different amounts (Parts C of Figures 4.11 and 4.13), changing the relative positions of the datasets.

As with the pseudo-modal shift, the effect of the background scaling is dependent on the original distribution of the background sub-population,

with the further the sub-population distribution from the standard normal distribution the greater the effect on the dataset as a greater scaling is required. Unlike the pseudo-modal shift, which influences all values equally, the background scaling has a larger effect on probes with higher values, as it is multiplicative. Therefore even a small scaling can potentially have a large effect on a dataset that contains large values in the enriched sub-population. The density plots (parts D of Figures 4.8 and 4.9) show this scale extends the limits of the datasets, reducing the density accordingly. This change can be seen more clearly in the the y-axis scale of the genome plots of replicate datasets (Parts D of Figures 4.10 and 4.12). As with the pseudo-modal shift, the scaling is different with each different set of replicate datasets, which can be seen as further relative changes between the datasets (Parts D of Figures 4.11 and 4.13).

As previously discussed, the aim of the normalisation procedure is to make the distributions of estimated background regions of different datasets the same, so as to enable comparisons to be made between the enriched regions. This objective was investigated by creating QQ-plots of the different datasets. Figure 4.14 shows pre-and post-UV treatment H3Ac data after preprocessing (A), quantile normalisation (B), the pseudo-modal shift (C) and background scaling (D). The grey box shows values less than zero, that is, the estimated background region. Alignment of the data points on the line $y = x$ in this region shows equal background distributions, while data points lying around this line show similar distributions. The distributions of the fully normalised data in these background regions are similar (shown in parts D) having the bulk of their data points lying on or around this line. As discussed previously, the estimated background regions of the datasets do not need to follow exactly the same distribution for the normalisation to be effective, they need only approximate the same distributions. This makes the enriched regions comparable. Similar QQ-plots are shown for the Gcn5p datasets at the three time points following UV irradiation plotted against the untreated, showing results pre- and post-normalisation (Figure 4.15). These plots also show the approximate alignment of the background regions following the normalisation procedure.

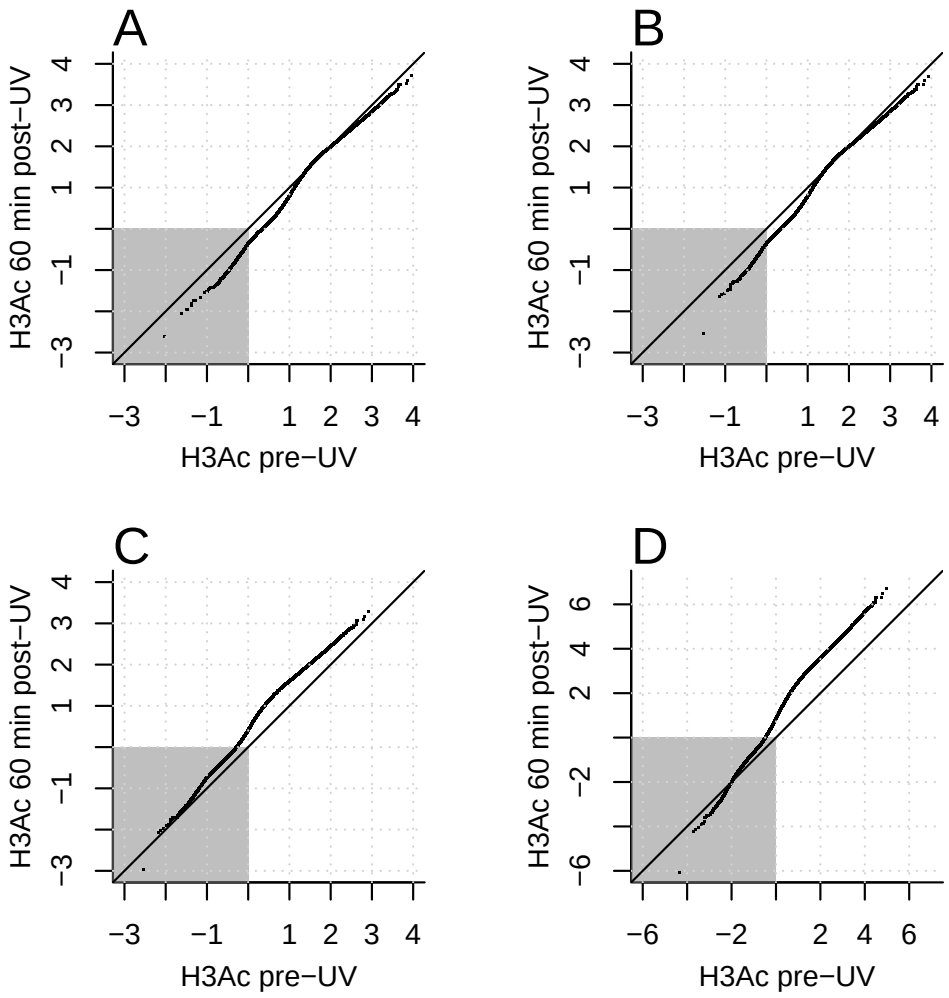


Figure 4.14: Q-Q plots of H3Ac data undergoing normalisation: Pre- and post-UV datasets following preprocessing (A), quantile normalisation (B), the pseudo-modal shift (C) and background scaling (D). The grey region shows values less than zero, that is, the estimated background region. The black line shows $y = x$, that is, equality between the two datasets. Note that part D is shown at a different scale.

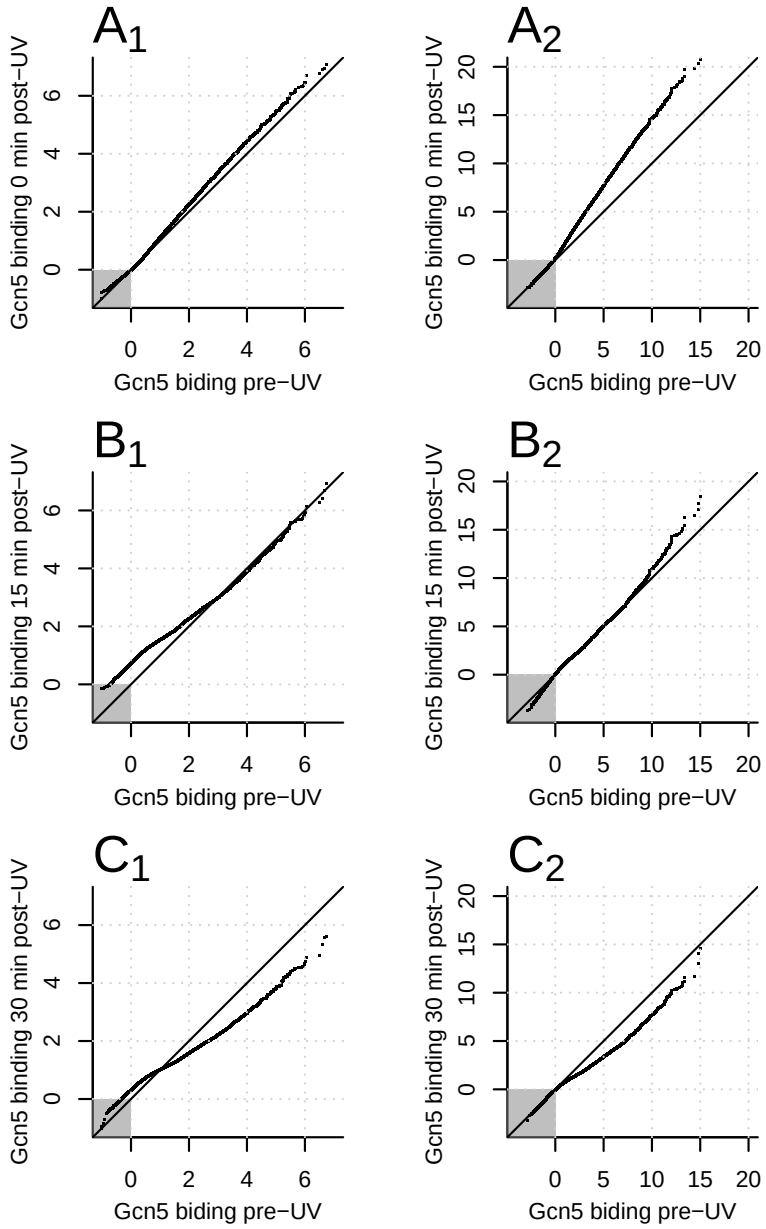


Figure 4.15: Q-Q plots of Gcn5p data pre- and post-normalisation: Pre-UV datasets plotted against the three post-UV datasets(0 min (A), 15 min (B) and 60 min (C)) before (1) and after (2) normalisation. The grey region shows values less than zero, that is, the estimated background region. The black line shows $y = x$, that is, equality between the two datasets. Note the changes of scales between the pre- and post-normalisation data plots.

4.3.1 Validation

A method independent of hybridisation to microarrays was used to validate the results of the normalisation procedure, namely Q-PCR. The fully normalised datasets were used to choose regions of the genome to test by this method. Six probes were chosen (details in "Q-PCR probes.pdf" in the electronic appendix; see Page 367) showing a range of different values in both the treated and untreated datasets. The chosen regions are shown in Figure 4.16, with black lines showing averaged untreated and red lines showing averaged UV treated. Two probes show similar low values both before and after treatment (A), two show high enrichment before and after treatment (B and E) and two show low enrichment before treatment and high afterwards (C and D). PCR primers were designed around these probes and Q-PCR reactions performed by Dr. Katie Evans.

The values for these probes throughout the normalisation process are shown as bar charts in Figure 4.17, showing the values from preprocessed (A), quantile normalised (B), pseudo-modal shifted (C) and background scaled (D) data. This shows that the values at the end of the procedure are different from those at the start, with the whole process required to bring out the differences between the values from the different conditions. These values are shown (out of log scale for comparison to the Q-PCR data) in Figure 4.18A along with the Q-PCR results in Figure 4.18B. Untreated values shown as dark grey and treated values as light grey. These values are shown together in Figure 4.18C, with microarray values scaled relative to the untreated Q-PCR values. Shaded bars show Q-PCR results against the corresponding unshaded bars of the microarray results. It is clear that there is a strong relationship between all Q-PCR and microarray data. T-tests performed on these values show that the microarray values are not significantly different to the Q-PCR values following the normalisation procedure (raw and FDR corrected P-values are shown in Table 4.4). This is in contrast to the comparison of raw values, the results of which produce smaller P-values, with one statistically significantly different set of values at the 95% significance level (raw and FDR corrected P values are shown in Table 4.5). Corre-

lating the two sets of values (Figure 4.19) shows a similar result: the raw microarray data (black points) show no correlation with their corresponding Q-PCR values, with a Spearman's Rank Correlation Coefficient rho value of 0.314, while the normalised data (red points) show a high correlation, with a rho value of 0.943. This shows that the normalisation procedure is required to convert the microarray data to a scale that accurately reflects the true biological state. Together this shows that the normalisation procedure works as expected and transforms the data into a format that allows relative comparisons to be made between different datasets, here untreated and UV treated H3Ac levels. Similar validation work is currently being performed at a number of sites from the Gcn5 binding datasets.

4.4 Alternative process

In situations where this normalisation process cannot be applied, such as where a background population is absent (if a whole genome is enriched), or indistinct (if there are too few probes in the background to form a clear subset), artificial DNA may be added to fulfill a similar function.

4.4.1 DNA spikes

Spikes are fragments of genetic material of known, varying concentrations added to the genetic material being assayed with a microarray, which have unique corresponding probes on the microarray. There are several examples of experimental techniques where spiked in DNA have been used. Several of these investigations have been into gene expression microarray normalisation techniques, where the addition of DNA of known concentrations allows various normalisation processes to be compared against a set of constant reference values. Chua et al. (2006) for example compare five normalisation methods with spike ins representing 200 differentially expressed (DE) genes. Similarly, Rydén et al. (2006) compare 252 normalisation methods with 8 DE genes and 12 non-differentially expressed (NDE) genes at varying concentrations, each represented 480 times on the microarray. McCall and Irizarry

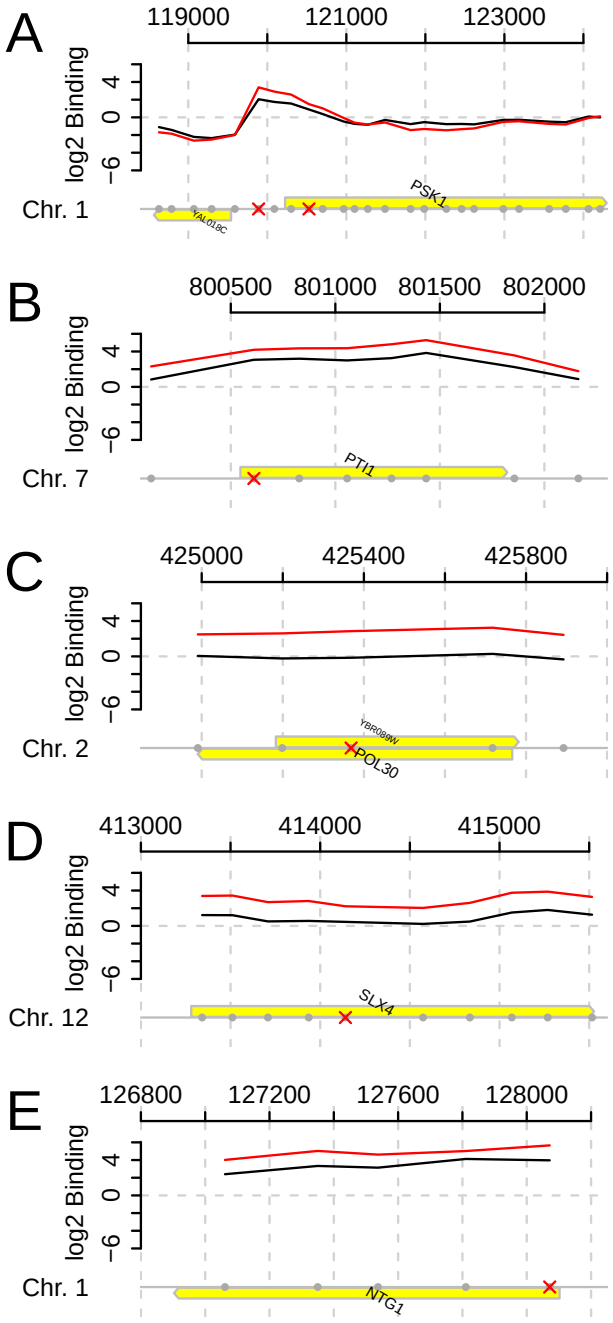


Figure 4.16: Probes chosen for Q-PCR analysis: Six probes showing a range of H3Ac values were selected from five regions. Probe positions are highlighted with red crosses. Black lines show averaged untreated data, red lines show averaged 60 min post-UV treatment data.

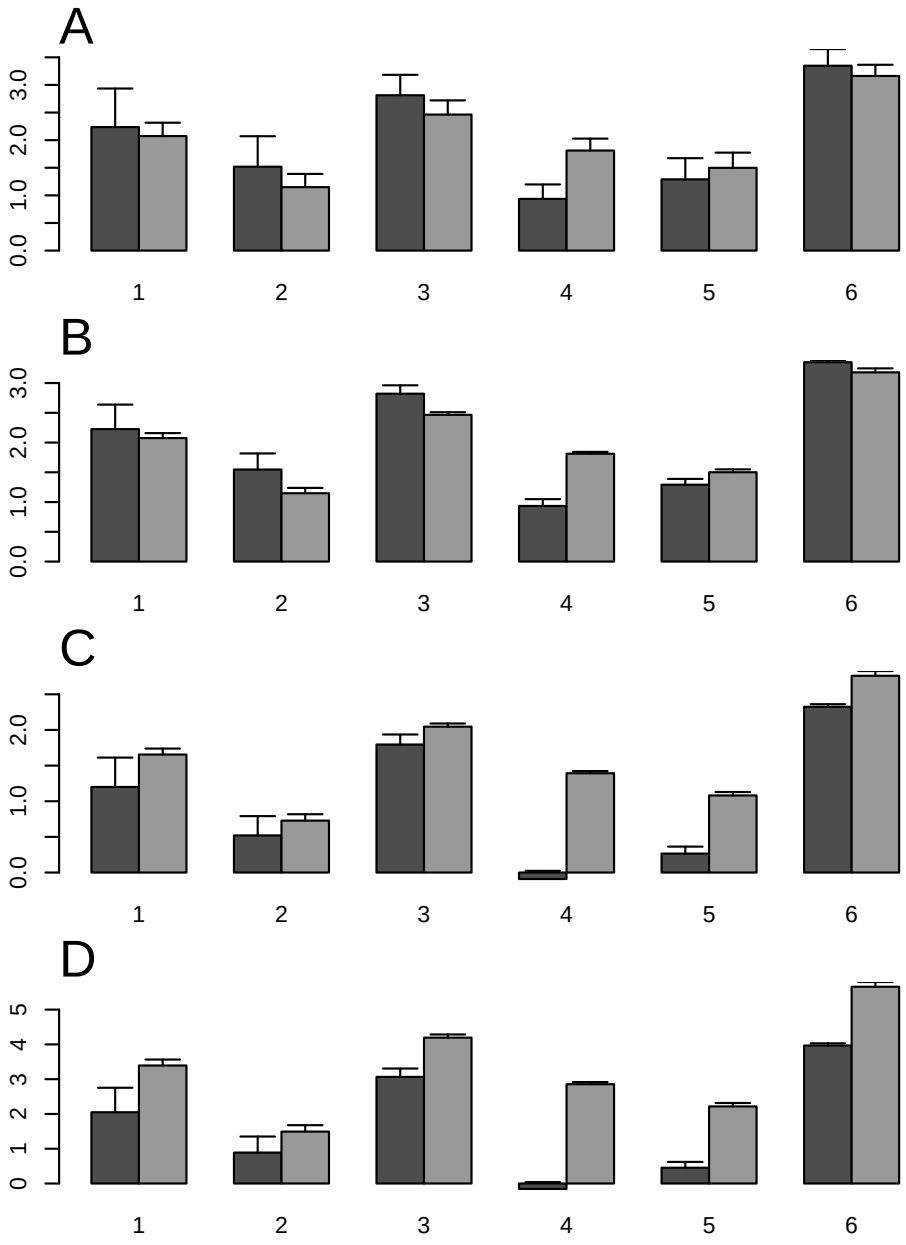


Figure 4.17: Bar charts of data from probes chosen for Q-PCR: Preprocessed (A), quantile normalised (B), pseudo-modal shifted (C) and background scaled (D) data for the six probes chosen for Q-PCR. Error bars show standard errors.

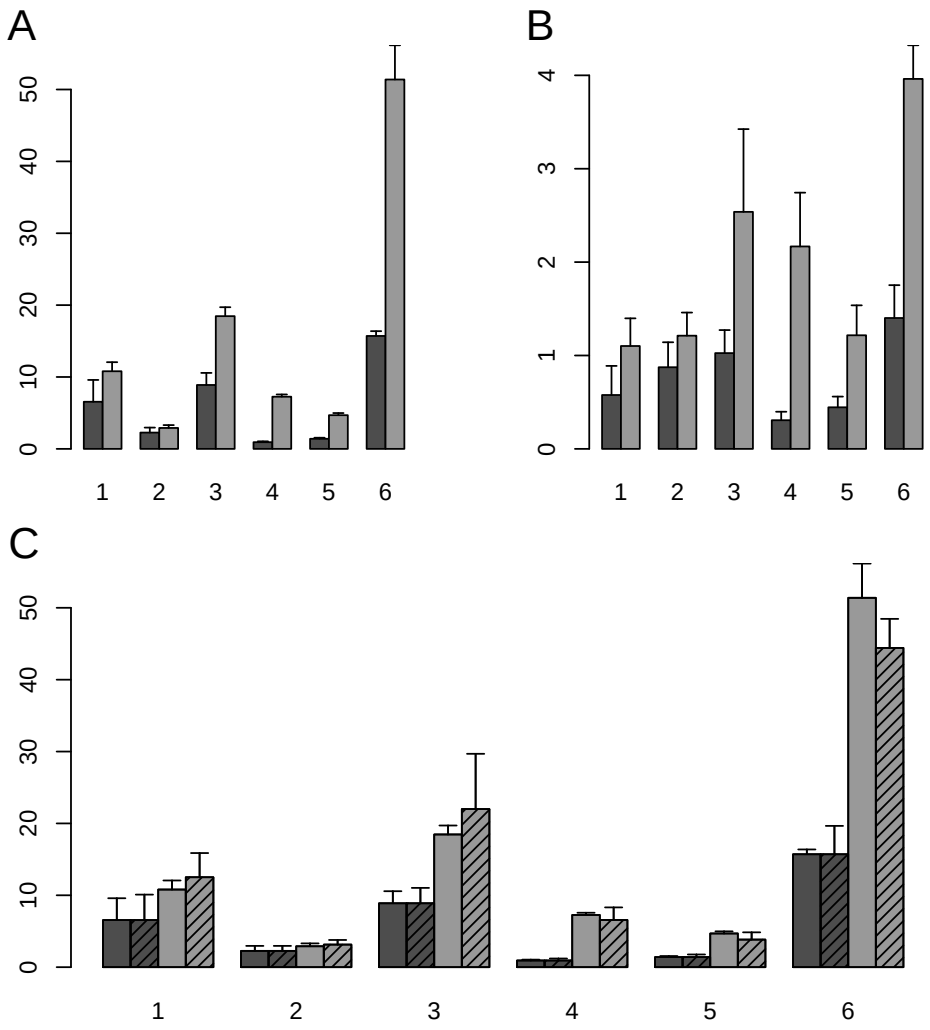


Figure 4.18: Bar charts of microarray and Q-PCR data: Untreated (dark grey) and treated (light grey) values from the microarrays (A) and Q-PCRs (B) for the six tested probes. Microarray values have been taken out of log scale to enable comparisons with Q-PCR values. Combined data (C) shows microarray (unshaded) and Q-PCR (shaded) values are very similar across all values tested. In C the Q-PCR values are scaled so as to bring the untreated values to the respective microarray untreated value. Error bars show standard errors.

Site	P-value	FDR value
1	0.653	0.813
2	0.769	0.813
3	0.643	0.813
4	0.813	0.813
5	0.510	0.813
6	0.410	0.813

Table 4.4: Normalised microarray and Q-PCR comparison P-values: Raw and FDR corrected P-values from t-tests comparing normalised microarray and Q-PCR results for the six tested values.

Site	P-value	FDR value
1	0.789	0.789
2	0.729	0.789
3	0.276	0.413
4	0.074	0.164
5	0.082	0.164
6	0.005	0.032

Table 4.5: Raw microarray and Q-PCR comparison P-values: Raw and FDR corrected P-values from t-tests comparing raw microarray and Q-PCR results for the six tested values.

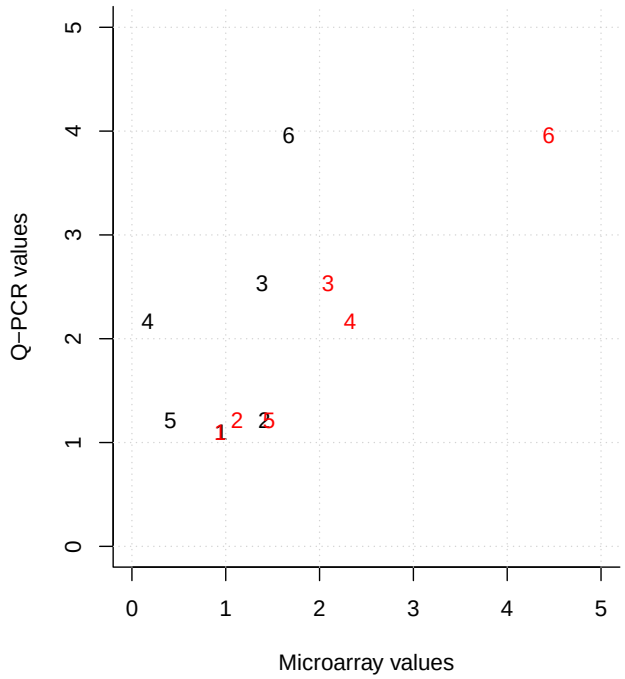


Figure 4.19: Microarray and Q-PCR values correlation: Raw (black) and normalised (red) microarray data plotted against their corresponding Q-PCR values for the six tested values. Each point shows the number of the tested site.

(2008) use spike ins to compare gene expression results from Affymetrix, Agilent and Illumina microarrays. These methods may have provided useful information about different normalisation procedures, and demonstrated the usefulness of spike in DNA, but were not designed to be used alongside real data as a full normalisation procedure.

Fardin et al. (2007) addressed this problem by suggesting using spiked in DNA to normalise low density gene transcription microarrays. These are microarrays containing only a selection of genes of interest to a particular investigation. As such they do not have the large population of NDE genes used in global normalisation and so require a different normalisation method. They used 8 RNA spikes to normalise 178 genes using the composite loess method. This constructs a curve through the spike in values and normalises all values based on this curve (Smyth and Speed, 2003).

Johnson et al. (2008) used DNA spikes to test various microarray platforms and normalisation methods with ChIP-chip data. A total of 100 sequences at varying concentrations were spiked into reference DNA and sent to 7 different laboratories for analysis. This shows that spiked in DNA is detectable and useful in ChIP-chip as well as gene expression microarray investigations.

To the best of my knowledge, there have been no attempts to normalise ChIP-chip data using spiked material. We attempted to develop a method to normalise any ChIP-chip datasets that are not suited to the novel procedure presented above because they do not have a distinct background region. DNA damage is one such example, which can occur at sufficiently high a level throughout a genome that no background region sub-population exists to be used for normalisation. Therefore the comparison of damage levels at different time points is not possible without the addition of artificial constants in the form of spikes.

The *Escherichia coli* genome was chosen as a source to design spike DNA probes from. This would reduce the likelihood of significant sequence similarity to the yeast and human genomes used experimentally in our laboratory, and any other eukaryotic genome that may be analysed the future. The first ~700,000 nt of the *E. coli* genome was split into 60 nt sequences (the length

of the probes on the Agilent microarrays). These ~12,000 sequences were uploaded to Agilent's eArray program (`https://earray.chem.agilent.com/earray/`) and analysed using this software to assign a score to each probe. Full information on how these scores are calculated is not provided, but factors such as melting temperature, GC content, hairpin ΔG formation, sequence complexity and homology to the reference genome are taken into account (Agilent Technologies Inc., 2010a). The scores range from 0 to 1, with higher scores representing a greater "likelihood that a probe will produce a good log ratio response."

All probes with scores greater than 0.95 (1020 total) were analysed for sequence similarity to the yeast and human genomes with BLAST searches (Altschul et al., 1990). The blastn program was used, which accommodates short sequences, and the "Automatically adjust parameters for short input sequences" box checked to ensure that accurate results were generated with the short sequences being tested. The 100 probes with the least similarity to both genomes were taken and used as the initial set of spikes. The statistics for these probes are summarised in Table 4.6 and shown in full in "Spike probe information.pdf" in the electronic appendix (see Page 367).

Statistic	Human	Yeast
No similarity	73	27
Longest similar stretch	30	31
Overall mean similarity	6.3	15.1
Mean similarity with some match	23.2	20.7

Table 4.6: Spike probes summary: Lengths of continuous sequence matches between the spike probes and the human and yeast genomes.

These probes were added to two custom microarray designs, one each for yeast and human. The yeast design is the same as that on the G4493 microarrays, with all probes for the mitochondria removed. The human design contains 39,517 probes over a 5,000,000 region of chromosome 17, from positions 10,000,000 to 15,000,000, giving an average resolution of one probe every 126.5 nt.

Work to use these spike probes as part of a full normalisation procedure is ongoing in our laboratory.

4.5 Discussion

The novel normalisation procedure presented here allows for the comparison of data from different microarray experiments to facilitate the discovery of biologically relevant results beyond the currently detectable large, wholesale changes, greatly expanding the use of ChIP-chip datasets. Changes in levels of protein binding, histone modifications or other biological factors can now be detected and compared under a variety of experimental conditions, where previously only the presence or absence of binding could be reliably inferred. This has been demonstrated with H3Ac data from ten microarrays under two different experimental conditions, revealing genome wide variations in levels between the different conditions.

Previously, the primary motivation for creating ChIP-chip datasets was to determine where in the genome a particular feature of interest is present, be that a protein, epigenetic modification, or other factor that can be identified by immunoprecipitation. Once this had been determined, by use of some enrichment or peak detection algorithm, the dataset becomes largely redundant, being replaced by a list of locations or regions identified as containing the feature of interest. For example, Schlecht et al. (2008) perform an analysis of Abf1 binding under three different conditions (fermentation, respiration, and sporulation) in this way, determining sites of changed occupancy between the three conditions. The data from the probes at these locations may be analysed further, to examine relative binding levels within the same dataset, or the list may be left as it is, reducing the data to a simple boolean representation of the genome, showing whether binding is present (TRUE) or absent (FALSE) at each probe of the microarray. Comparisons between different datasets was limited by this treatment of the data, and could only determine whether or not the presence of the factor of interest appeared, disappeared or stayed the same at a given location. If it stayed the same, the relative binding level between the two conditions could not be determined

because of the lack of a normalisation procedure that could be applied to the different datasets to allow comparisons to be made between them. The novel normalisation procedure presented here overcomes this problem, revealing an extra dimension of analysis of ChIP-chip data by giving new meaning to the binding values of the datasets.

This is not a new concept in the microarray field, with gene expression microarrays — the most popular use of microarray technology — allowing comparisons to be made between the levels of mRNAs present in cells from different conditions. This is possible due to the normalisation of the different datasets, of which there are many methods. These methods are not applicable to ChIP-chip data, and no alternative existed, constraining the application of the technology. The method presented here removes this constraint, opening ChIP-chip technology up to a wealth of possible new applications. Rather than reducing the datasets to lists of positions, more robust comparisons can be performed to determine relative increases or decreases of binding between the different datasets, coupled with appropriate statistical tests.

While a useful tool for researchers wishing to compare microarrays from different conditions, there are a number of inherent caveats which should be taken into account. The method relies on the previously noted expectation that the background sub-populations of data approximately follow a normal distribution. If this assumption is not met then the scaling part of the normalisation method may fail, as it will be unable to create the standard normal distribution from non-normally distributed data. In practice, small deviations from the normal distribution will not have a large effect on the results of the normalisation procedure as it will still enable this portion of the data to approximate the standard normal distribution. In our laboratory we have not seen any examples of datasets that have a background sub-population that does not approximate a normal distribution. Situations where the background sub-population is very different to the normal distribution are likely to represent poor quality across the whole dataset and so it may not be suitable for inclusion in further analyses.

The method requires a clear maximum in the sub-population of background data at which to assign the pseudo-mode to centre on zero. If this is

not present then this shift cannot be applied. It is worth noting that while in most assays this is usually the maximum of the whole population, and this is what the algorithm automatically searches for, it is not a requirement that this is the case. If the enriched portion of the data is larger than the background, this will have the largest peak in the distribution. However, provided there is still a discernible peak in the background sub-population this can be manually identified and specified to the algorithm. This allows for accurate normalisation even when more than 50% of the probes represent enrichment. The method cannot be applied when all or the majority of probes are enriched as no estimate of the background will be possible, such as the case with CPD damage (see Chapter 6). In this scenario, data from spike probes may be used in place of the background. This methodology is currently being developed in our laboratory.

This shifting method may introduce small errors, as it is based on an estimated pseudo-mode. However, there are many other sources of variation in microarray experiments which limit the accuracy of results and so any further small variations will not adversely affect the conclusions that can be drawn. Any introduced variation should be borne in mind along with the other sources of variation and taken into account when performing analyses. Microarray data should not be treated as a definitive results, rather a platform from which to generate hypotheses which can be tested by more sensitive techniques.

The method has been developed to allow comparisons of differences in a single factor due to changes in experimental conditions. It cannot reliably compare between data with other sources of variation as these will introduce undetectable variations. For example, datasets generated with different antibodies used in the immunoprecipitation stage will potentially have variations due to differing efficiencies of the antibodies. Therefore when comparing between them, even after normalisation, it will be impossible to say whether any changes are due to genuine differences in binding levels or differences in immunoprecipitation efficiencies.

The nature of the normalisation method means that all assays to be compared do not have to be normalised at the same time. Because each

background sub-population is scaled to the standard normal distribution, which is unvarying, each set of replicates can be normalised independently of the others and therefore the assays do not all have to be carried out at the same time. However, in the interests of minimising all sources of variation it is recommended that as many assays as possible are carried out together.

The computational process is very fast, allowing full normalisation of multiple datasets in a matter of seconds.

Chapter 5

Development of a novel enrichment detection method

5.1 Introduction

ChIP-chip is a technique that has been used primarily for investigating the binding locations of proteins on a genome wide scale (Buck and Lieb, 2004). This has necessitated the development of computational tools to identify those binding sites. ChIP-chip data can range from thousands to millions of individual values and so automated methods of peak detection are essential. Various methods have been developed to perform this function, which are shown in Table 5.1. Not all are applicable to the data analysed in this investigation because some have been designed to work only on data from different microarray formats. The earlier methods have been shown to be outperformed by the newer methods, which means only those towards the bottom of the list are currently relevant. Several of these are no longer publicly available and so the currently available tools for accurate enrichment detection of any format of ChIP-chip data are limited, potentially limiting the processing of data. The objective of the work presented in this chapter was to develop a new enrichment detection procedure to fill this gap, that could work with any format of ChIP-chip microarray data and detect peaks or extended regions of enrichment as required. The method developed is able to utilise multiple replicate datasets to increase the power of detection of enrichment over analysing the datasets individually. It does this in such a way that

the detection threshold is dynamically adjusted throughout the procedure to maintain the same overall probability level, meaning that the application of a multiple testing correction is not required on the final results, thereby removing any biases that this may introduce. The performance is indicated to be more powerful that the previously published methods, without having those methods available to test.

5.1.1 Existing methods

The first application of what is now referred to as ChIP-chip used a single array error model to identify enriched probes (Ren et al., 2000; Roberts et al., 2000). This early microarray had single probes in regions of interest, predominantly promoters. The error model gives a significance value to each probe based on the signal intensities of the two channels, uncertainties due to background subtraction and other non-uniformities such as hybridisation efficiency variations, taking into account values from replicate arrays.

MDScan (Liu et al., 2002) attempts to find DNA binding sites by identifying common sequence motifs in enriched areas. This method analyses sequences of highly enriched regions and uses these results to find additional sequences from regions of lower enrichment. It is therefore more applicable to consensus motif identification than peak finding. The final result is reliant upon there being a consensus motif present.

Median percentile ranking is suggested as an analysis method by Buck and Lieb (2004). This is a simple statistical procedure reliant on a number of repeats for each experiment. The probe values are converted to ranks, scaled to between 0 and 1 and the medians of these for each probe are analysed. If all binding values are random the medians will fall into a normal distribution centered on 0.5 and bounded by 0 and 1. If a sub-population of probes are consistently enriched their ranks will be consistently high and so their median values will fall at the top end of the range, creating a bimodal distribution. The trough of this distribution can then be used as a cutoff to define enriched probes. The advantage of this method is values are converted to ranks, so the original values become irrelevant and normalisation is not required. It

Name	Citation	Program readily available	Works with any data format
SAEM	Ren et al. (2000)	No	Yes
MDScan	Liu et al. (2002)	No	-
Median percentile rank	Buck and Lieb (2004)	No	Yes
Peakfinder	Glynn et al. (2004)	Yes	Yes
No name	Cawley et al. (2004)	No	No
Chipotle	Buck et al. (2005)	Yes	Yes
HMM	Li et al. (2005)	No	-
Chipper	Gibbons et al. (2005)	Yes	Yes
No name	Kim et al. (2005)	No	-
TileMap	Ji and Wong (2005)	No	-
JBD	Qi et al. (2006)	No	-
TAMALPAIS	Bieda et al. (2006)	Yes	No
MAT	Johnson et al. (2006)	Yes	No
Permuta	Lucas et al. (2007)	No	-
MA2C	Song et al. (2007)	Yes	No
Mpeak	Zheng et al. (2007)	No	-
Tilescope	Zhang et al. (2007)	No	-
Poisson approximation	Zhang (2008)	No	Yes
Splitter	Johnson et al. (2008)	No	-
JAMIE	Wu and Ji (2010)	No	-
DECODE	Barrett et al. (2011)	No	-
Wavelet	Karpikov et al. (2011)	No	-

Table 5.1: Peak detection methods: names and publications of ChIP-chip data analysis programs, whether or not they are currently readily available for use (as a downloadable program or web server) and whether or not they are able to process data from any microarray format (where information is available).

does however require enough repeats to allow the bimodal distribution to be perceived, which may be a large number if only a small number of sites are enriched.

Peakfinder (Glynn et al., 2004) smooths data and identifies peaks from the resulting first derivative. The data is smoothed to remove spurious peaks caused by noise but keep peaks caused by genuine enrichment. Peaks are then identified as regions where the first derivative is zero.

Cawley et al. (2004) present a method based on the Wilcoxon Rank Sum test, comparing treated and untreated datasets to identify differences, which are taken to be regions of enrichment. Datasets are quantile normalised within groups and then all scaled to have a median feature intensity of 1000. Probes within sliding windows of $\pm$ 500 bp are tested against the null hypothesis of equality between the datasets. A p-value cutoff of 10^{-5} is used to define enriched regions. This method has the disadvantage that a number of untreated control datasets have to be produced, at extra time and cost.

ChIPotle (Buck et al., 2005) uses a sliding window (default 1 kb in length with 0.25 kb steps). At each step the average of all points is calculated which smooths the data, aiming to remove spurious peaks and retain genuine peaks. A p-value is calculated from the standard error function for each window under the null hypothesis that the observed ratios are independent, identically distributed random variables having a Gaussian distribution with a mean of zero. These p-values are then corrected by the Bonferroni method.

Li et al. (2005) present a hidden Markov model (HMM) approach as an alternative to the method used by Cawley et al. (2004). This has two hidden states: ChIP-enriched and non-enriched. The method analyses probes to determine which of these two states all probes are in, calculating probabilities based on an estimate of the total number of binding sites and the total number of probes.

Chipper (Gibbons et al., 2005) uses variance stabilisation to identify protein binding sites. Data are transformed using `vsn` (see Chapter 4) and these scores used to determine p-values based on the null hypothesis of no binding.

Kim et al. (2005) use a two stage approach to identify binding sites in their datasets. Firstly, microarrays of ~14.5 million probes at ~100 bp resolution

were used to identify potential binding locations. The resulting data was smoothed by median filtering with a window size of 3 probes. Enriched areas were then defined as regions with a minimum of 4 probes separated by a maximum of 500 bp with values greater than 2.5 standard deviations from the mean. These regions were used to design a second microarray containing ~400,000 probes covering ~10,000 regions at 100 bp resolution. This was used to more precisely define binding sites. A double regression model is used to fit neighbouring log ratio signals to asymmetric triangles centred on candidate binding sites using a sliding window approach. Local residual minima are defined as peaks. This method requires a minimum separation of 500 bp between peaks. As well as this inability to find peaks closer than 500 bp, the main drawback to this technique is that two microarrays need to be used, with a design stage between the two, increasing the time and cost of any experiments.

TileMap (Ji and Wong, 2005) applies a two stage approach. Firstly a Bayes model is used to calculate a test statistic for each probe. These statistics are then used to to infer peaks. Neighbouring probes are analysed through a moving average or hidden Markov model. This approach allows multiple datasets to be analysed at the same time by calculating the test statistics from the multiple probes.

Joint binding deconvolution (JBD; Qi et al., 2006) reconstructs binding events from ChIP-chip at a higher spacial resolution than the underlying microarray probe spacing. This is achieved by deconvolving the predicted probe intensity peak shape from the observed peak shape to infer the genuine binding event location. This allows pairs of nearby events to be distinguished as multiple binding locations. This is linked to sequence information to further refine the predicted binding sites by consensus motif analysis.

TAMALPAIS (Bieda et al., 2006) uses a similar methodology to that described by Kim et al. (2005). Rather than requiring a single user defined threshold value for an array, the 95^{th} and 98^{th} percentiles of the ratio values are used. This 'normalises' the threshold values for each array to reflect the amplitude and distribution of the signal. The run size is determined by the number of probes having a p-value < 0.0001. This corresponds to 6 and 8

consecutive points above the 98^{th} and 95^{th} percentiles respectively.

Model based analysis of tiling-arrays for ChIP-chip (MAT; Johnson et al., 2006) is a method created for the analysis of Affymetrix microarray data, which takes into account the sequence of each probe to apply a correction, before estimating

Lucas et al. (2007) present a sliding window approach based on window size, ratio cutoff and the percentage of probes in the window over the cutoff. A false positive rate was estimated for each window. Peaks were called in windows with 40–100% of the probes above the defined cutoff and a false positive rate $\leq 10\%$ in at least 3 of the 4 datasets. The false positive rate was estimated by 20 repetitions of randomising the data and estimating the number of peaks found by chance at each cutoff value.

MA2C (Song et al., 2007) is a normalisation method which takes into account GC content, also containing a peak detection method based on MAT (Johnson et al., 2006). A sliding window of defined length is centred on each probe and a score assigned based on the median, pseudo-median, median polish or trimmed mean of the probes in the window. The median and trimmed mean values are calculated from all replicate datasets, where present. P-values are assigned to the windows and a cutoff based on p-values or FDR is applied.

Mpeak (Zheng et al., 2007) uses a model based method to recognise peak shapes in data. It looks for the truncated triangle shape of peaks by fitting a multiple regression model to a window around a central probe. All local maxima are first found, defined as the largest value in a 200 bp region. These are ordered from largest to smallest and, working down this list, the method fits the model to the window around the probe to find the point with the smallest residual variance. This is repeated for neighbouring probes with the lowest value indicating the estimated binding site.

Tilescope (Zhang et al., 2007) is a set of programs to analyse ChIP-chip data. Three peak detection methods are included, one based on the method of Cawley et al. (2004), one on HMM (Li et al., 2005) and one developed by the authors. This identifies local signal peaks in an iterative fashion by finding points that correspond to peaks and meet a predefined p-value threshold.

All points within a predefined distance are removed so as to prevent the detection of secondary peaks from the same feature. This is repeated until the signal being analysed is below the cutoff threshold.

Zhang (2008) propose a Poisson approximation approach which aims to accurately approximate the statistical significance of peaks in a manner better than calculating significance values and applying multiple testing corrections. This uses Poisson clumping on suitably modified data to calculate p-values, taking into account multiple datasets.

Splitter (Johnson et al., 2008) is available as a web server which also contains basic normalisation and averaging functions. The algorithm dynamically defines the cutoff values for peak detection. This cutoff is increased over a defined number of steps and the number of hits before and after the increment are compared. If this ratio is smaller than a defined 'break ratio' all of the hits are reported.

JAMIE (joint analysis of multiple ChIP-chip experiments) (Wu and Ji, 2010) is an algorithm which aims to 'borrow' information from related datasets to improve peak detection. Correlations between datasets are found using a hierarchical mixture model. A sliding window approach is used to determine binding sites in each dataset based on a defined threshold. These results are then compared across the related datasets to improve the determination of binding sites.

DECODE (Barrett et al., 2011) identifies potential binding regions as those at least 400 bp in length with a value greater than 1, after setting the histogram maxima to 1. The signal in these regions is then smoothed and these values analysed further. The first three derivatives of these values are used to identify local maxima. These maxima are then analysed to estimate peaks by two methods; minimising the differences between the transformed and original enrichment signals and maximising the entropy of the probes. P-values are then assigned to the peaks and a FDR applied to remove false results.

Wavelet (Karpikov et al., 2011) applies a wavelet transformation to data, whose fundamental aim is to separate data based on its scale. The method therefore attempts to separate binding signals from background noise. The

transformation is applied to both the red and green signals and the log ratio of these values is analysed. Thresholding allows peaks of varying sizes to be detected at the same confidence level and a FDR applied to the final results.

Methods produced by the microarray manufacturers for use on their own datasets also exist (Johnson et al. (2006) for Affymetrix and Scacheri et al. (2006) for Nimblegen).

5.1.2 Motivation for creating a new method

The majority of the above methods are either not available for use or cannot be applied to all types of ChIP-chip data, including that analysed here, as shown in Table 5.1. All of the most recent methods, shown to outperform earlier methods, cannot be applied to the data analysed here. Most of the other methods are intended to be applied to single datasets, and those that can be applied to multiple datasets do not do so to increase the power of detection of peaks, causing the loss of much valuable data from repeated experiments. In addition, several methods only seek to find defined peaks, which is of no value when analysing data such as histone acetylation, which occurs over extended regions. It is widely agreed that ChIP-chip experiments should be carried out in replicate to reduce the likelihood of spurious results being deemed genuine binding sites. Generally, these replicate datasets are analysed separately and peaks found in several or all of these datasets are reported as genuine peaks. This methodology means that smaller binding peaks may be missed because the power of detection in each individual dataset is not great enough to find them. Combining all datasets increases the power of detection and increases the chances of these peaks being identified.

A new enrichment detection method has been developed here to overcome these problems. The method is able to work with data from any microarray format, being able to work with data loaded from simple tab-delimited text files (Section 3.2.1.3) as well as the Agilent Feature Extraction file format. It analyses all replicate datasets simultaneously to achieve an increased power of detection, thereby allowing more binding sites to be identified. It also dynamically adjusts the detection cutoff level to maintain a p-value that seeks

to find no false positive results. This eliminates the need for any multiple testing correction to be applied to the final results, as all potential false results are removed at the point of detection. This means that the final results are more likely to all represent genuine peaks in the data than other methods, where the multiple testing correction method that is applied may be over- or under-sensitive. This new method is later shown to outperform existing methods at detecting enrichment in datasets containing artificially enriched spike regions.

5.2 Algorithm

The `peakDetection` function performs the enrichment and peak detection processes (Script 5.1) using the following arguments:

object An `arrayData` object to be processed (no default).

annotation A `genomeAnnotation` object for the current genome to be used in calculating the chromosome end points (no default).

windowSize A numeric vector specifying the window size to be used in determining enrichment (default 600).

fdre A numeric vector specifying the false peaks to 'find', used in determining the statistical significance levels (default 0.9).

scale A numeric vector specifying the factor to scale the dataset by, improving the detection of peaks in datasets which do not fully meet the expectations of the data.

findPeaks A logical vector indicating whether or not to perform peak detection and return a `peakList` or return regions of enrichment, as a TRUE or FALSE value for each probe (default TRUE).

shearSize A numeric vector specifying the average chromatin shear size of the material hybridised to the microarray (default 600).

Script 5.1: peakDetection: script to perform the enrichment and peak detection of an `arrayData` object.

```
## peakDetection function ##
## arguments: object (an arrayData object), annotation (a genomeAnnotation object) , windowSize (size of windows to use), fdre (statistical value), findPeaks (perform peak detection), scale (dataset scale factor), shearSize (average chromatin shear size)
peakDetection<-function(object,annotation,windowSize=600,fdre=0.9,findPeaks=TRUE,shearSize=600,scale=1) { #define function
  for (n in 1:ncol(object)) { #loop through datasets
    if("shiftByMode" %in% object$status[[n]] & "stNormScale" %in% object$status[[n]] & "rmNAs" %in% object$status[[n]]) { #dataset has been shifted and scaled
      }else{ #stNorm and shiftByMode not applied
        warning(paste(colnames(object$ratios)[n]," has not been fully normalised",sep=""), call.=F) #warn if correct normalisation has not been applied
    }
  }
  peaks.keep<-!is.na(object$ratios[,1]) #define non-missing ratios in dataset 1
  object.full<-object #copy data
  object<-object[peaks.keep,] *scale #remove ratios missing in dataset 1 and scale all data
  windows.all<-list() #initialise list to store windows
  peaks<-rep(F,nrow(object)) #initialise vectors to store peaks
  peaks.full<-rep(F,nrow(object.full))
  cutoffs<-matrix(ncol=1,qnorm(1-(fdre/nrow(object))^(1/1:10000))) #calculate cut off values
  cutoffs[cutoffs < 0]<-0
  multFactor<-ncol(object) #define multiplication factor = the number of datasets
  previous<-0 #set previous to zero
  for (chr in unique(object$coordinates[,1])) { #loop through chromosomes
    chrEnd<-max(c(annotation$coordinates[annotation$coordinates[,1] == chr,2:3],object$coordinates[object$coordinates[,1] == chr,2:3]),na.rm=T) #get the maximum chromosome value
    objectChr<-object[object$coordinates[,1] == chr,] #get data on current chromosome
    coordinates<-ceiling(rowMeans(objectChr$coordinates[,2:3])) #get probe coordinate mid points
    ratios<-objectChr$ratios #get dataset ratios
    probes<-objectChr$annotations[,1] #get probe IDs
    nRows<-nrow(objectChr) #get number of probes
    for (n in 1:nRows) { #loop through probes
      gap<-0 #set gap to zero
      d<-n #set down value (d) to probe number (n)
      if (coordinates[n] > windowSize) { #if probe coordinate is greater than the windowsize
        while (gap < windowSize) { #loop while gap value is less than the windowsize
          d<-d-1 #decrease d by 1
          if (d > 0) gap<-coordinates[n] - coordinates[d] else gap<-windowSize+1 #calculate the gap between the current and downward probes if the downward value is greater than zero, other wise set gap to greater than the windowsize
        } #gap greater than windowsize = window found + 1 probe
        d<-d+1 #add 1 to d to reenter the window
        if (min(ratios[d:n,],na.rm=T) > cutoffs[(n-d+1)*multFactor]) peaks[(d+previous):(n+previous)]<-T #if probe is to be included, if all ratios in the window are greather than the cutoff, set probes
```

```
            in window to TRUE
37          gap<-0 #set gap to zero
38          d<-d+1 #add 1 to d
39          while (gap < windowSize) { #loop while gap value is less than the
                windowsize
40            d<-d-1 #decrease d by 1
41            if (d > 0) gap<-(coordinates[n]-1) - coordinates[d] else gap<-
                  windowSize+1 #calculate the gap between the current - 1 and
                  downward probes if the downward value is greater than zero,
                  other wise set gap to greater than the windowsize
42          } #gap greater than windowsize = window found + 1 probe
43          d<-d+1 #add 1 to d
44          if (d < n) if (min(ratios[d:(n-1),],na.rm=T) > cutoffs[((n-1)-d+1)*
                multFactor]) peaks[(d+previous):(n-1+previous)]<-T #if downward
                probe is less than the current probe, if probe is to be included
                , if all ratios in the window are greather than the cutoff, set
                probes in window to TRUE
45        }
46        if (coordinates[n] < (chrEnd - windowSize)) {#probe coordinate is not
              in the last windowsize of the chromosome
47          gap<-0 #set gap to zero
48          u<-n  #set up value (u) to probe number (n)
49          while (gap < windowSize) { #loop while gap value is less than the
                windowsize
50            u<-u+1 #add 1 to u
51            if(u < nRows) gap<-coordinates[u] - coordinates[n] else gap<-
                  windowSize+1 #calculate the gap between the current and upward
                   probes if the upward value is less than the number of probes,
                   other wise set gap to greater than the windowsize
52          } #gap greater than windowsize = window found + 1 probe
53          u<-u-1 #subtract 1 from u to reenter the window
54          if (min(ratios[n:u,],na.rm=T) > cutoffs[(u-n+1)*multFactor]) peaks[(
                n+previous):(u+previous)]<-T #if probe is to be included, if all
                 ratios in the window are greather than the cutoff, set probes
                in window to TRUE
55          gap<-0 #set gap to zero
56          u<-u-1 #subtract 1 from u
57          while (gap < windowSize) { #loop while gap value is less than the
                windowsize
58            u<-u+1 #add 1 to u
59            if (coordinates[n]  > (chrEnd - windowSize)) inc<-F #set probe
                  include to FALSE if probe coordinate is in the last windowsize
                   of the chromosome
60            if(u < nRows) gap<-coordinates[u] - (coordinates[n]+1) else gap<-
                  windowSize+1 #calculate the gap between the current + 1 and
                  upward probes if the upward value is less than the number of
                  probes, other wise set gap to greater than the windowsize
61          } #gap greater than windowsize = window found + 1 probe
62          u<-u-1 #subtract 1 from u to reenter the window
63          if (u > n) if (min(ratios[(n+1):u,],na.rm=T) > cutoffs[(u-(n+1)+1)*
                multFactor]) peaks[(n+1+previous):(u+previous)]<-T #if upward
                probe is less than the current probe, if probe is to be included
                , if all ratios in the window are greather than the cutoff, set
                probes in window to TRUE
64        }
65      } #finished searching all probes
66      previous<-previous+nRows #add number of probes to previous
67    } #finished all chromosomes
68    peaks.full[peaks.keep]<-peaks #store peaks in appropriate locations
69    if(!findPeaks) return(peaks.full) #return enriched regions if required,
          otherwise peak detection:
70    if(length(which(peaks.full)) < 1) { #no enrichment is found
```

```
71      message("No peaks found") #print message
72      return(NULL) #return NULL
73    }#some enrichment is found
74    con<-consecutive(cbind(which(peaks.full),object.full$coordinates[peaks.
          full,1])) #get consecutive enriched probes
75    ratios<-object.full$ratios #get all ratios
76    peaks.found<-peaks.found.to<-peaks.found.from<-scores.found<-numeric() #
          initalise vectors to store results
77    for (n in 1:nrow(con)) { #loop through consecutive regions
78      if (con[n,2]-con[n,1] > 0) { #region is longer than a single probe
79        peaks.i<-matrix(ncol=multFactor,nrow=(con[n,2]-con[n,1])+1,0) #
              initialise matrix
80        ratios.all<-matrix(ncol=multFactor,ratios[con[n,1]:con[n,2],]) #get
              ratios in region
81        for (m in 1:multFactor) { #loop through datasets
82          ratios.current<-ratios.all[,m] #get ratios of dataset
83          ratios.current<-cbind(ratios.current,c(min(ratios.current),ratios.
                current[1:(length(ratios.current)-1)]),c(ratios.current[2:length
                (ratios.current)],min(ratios.current))) #align ratios with
                previous + next for vectorised maxima searching
84          peaks.i[ratios.current[,2] < ratios.current[,1] & ratios.current[,3]
                 < ratios.current[,1],m]<-1  #assign maxima 1
85        }
86        ratios.means<-rowMeans(ratios.all) #get mean ratios
87        ratios.means<-cbind(ratios.means,c(min(ratios.means),ratios.means[1:(
              length(ratios.means)-1)]),c(ratios.means[2:length(ratios.means)],
              min(ratios.means))) #align means with previous + next for
              vectorised maxima searching
88        peaks.a<-ratios.means[,2] < ratios.means[,1] & ratios.means[,3] <
              ratios.means[,1] #identify maxima
89        peaks.found<-c(peaks.found,matrix(con[n,1]:con[n,2])[peaks.a]) #store
              found probe positions
90        scores.found<-c(scores.found,rowMeans(peaks.i)[peaks.a]) #store maxima
               scores
91        peaks.i<-consecutive(which(rowMeans(peaks.i) > 0))+con[n,1]-1 #get
              consecutive replicates maxima probes
92        for (p in (which(peaks.a)+con[n,1]-1)) { #loop through maxima
93          if (p %in% peaks.i) { #mean maxima in replicate maxima
94            for (r in 1:nrow(peaks.i)) { #loop through maxima
95              if(p %in% peaks.i[r,1]:peaks.i[r,2]) { #find replicate maxima
                    matching mean maxima
96                peaks.found.from<-c(peaks.found.from,peaks.i[r,1]) #store
                      lower boundary
97                peaks.found.to<-c(peaks.found.to,peaks.i[r,2]) #store upper
                      boundary
98                break() #exit from loop
99              }
100           }
101         }else{ #mean maxima not in replicate maxima
102           closestPeaks<-abs(peaks.i-p) #get replicate peaks
103           near<-unique(peaks.i[closestPeaks == min(closestPeaks)]) #get
                  replicate peaks closest to average peak
104           near[abs(mean(object.full$coordinates[p,2:3])-rowMeans(matrix(ncol
                  =2,object.full$coordinates[near,2:3]))) > 200]<-p
105           peaks.found.from<-c(peaks.found.from,min(c(p,near))) #store from
106           peaks.found.to<-c(peaks.found.to,max(c(p,near))) #store to
107         }
108       }
109     }else{ #region is a singe probe
110       peaks.found<-c(peaks.found,con[n,1]) #store single probe
111       scores.found<-c(scores.found,1) #store score as 1
112       peaks.found.from<-c(peaks.found.from,con[n,1]) #store lower boundary
```

```
113       peaks.found.to<-c(peaks.found.to,con[n,2]) #store upper boundary
114     }
115   }
116   coordinates<-matrix(ncol=3,nrow=length(peaks.found)) #create matrix for
          coordinates
117   IDs<-matrix(ncol=2,nrow=length(peaks.found)) #create matrix for IDs
118   stats<-matrix(ncol=2,nrow=length(peaks.found)) #create matrix for stats
119   colnames(coordinates)<-c("PBRchr","PBRstart","PBRend") #set column names
          for coordinates
120   colnames(IDs)<-c("ID","Position") #set column names for IDs
121   colnames(stats)<-c("Score","Height")  #set column names for stats
122   coordinates[,1]<-object.full$coordinates[peaks.found,1] #add peak
          chromosomes
123   stats[,2]<-rowMeans(object.full[peaks.found,])$ratios #add peak mean
          ratios
124   stats[,1]<-scores.found #add peak scores
125   coords<-matrix(object.full$coordinates[peaks.found.from,],ncol=3) #store
          lower boundary coords in matrix
126   adjustAddValues<-adjustSubValues<-FALSE #initialise adjustors
127   subValues<-peaks.found.from-1 #get lower boundary - 1 probes
128   if(min(subValues) < 1) { #lowest probe is less than 1
129     subValues[subValues < 1]<-1 #set lowest probe to 1
130     adjustSubValues<-T #set subtracted values to be adjusted
131   }
132   coordsDown<-matrix(ncol=3,object.full$coordinates[subValues,]) #get lower
          boundary - 1 probes
133   if(adjustSubValues) coordsDown[1,2:3]<-rep(-Inf,2) #set first coordDown as
           -Inf if to be adjusted
134   sameChrDown<-coords[,1]==coordsDown[,1] #identify adjacent peaks on the
          same chromosome
135   coordinates[which(sameChrDown),2]<-ifelse(rowMeans(matrix(ncol=2,coords[
          sameChrDown,2:3])) - rowMeans(matrix(ncol=2,coordsDown[sameChrDown
          ,2:3])) < 2*shearSize,(rowMeans(matrix(ncol=2,coords[sameChrDown,2:3])
          ) + rowMeans(matrix(ncol=2,coordsDown[sameChrDown,2:3])))/2,rowMeans(
          matrix(ncol=2,coords[sameChrDown,2:3])) - shearSize) #calculate
          downward coordinate boundaries
136   coords<-matrix(object.full$coordinates[peaks.found.to,],ncol=3)  #store
          upper boundary coords in matrix
137   addValues<-peaks.found.to+1 #get upper boundary + 1 probes
138   if(max(addValues) > nrow(object)) { #highest probe is greater than the
          number of probes
139     addValues[addValues > nrow(object)]<-nrow(object) #set highest probe to
            the number of probes
140     adjustAddValues<-T #set added values to be adjusted
141   }
142   coordsUp<-matrix(ncol=3,object.full$coordinates[addValues,]) #get lower
          boundary - 1 probes
143   if(adjustAddValues) coordsUp[nrow(coordsUp),2:3]<-rep(Inf,2) #set last
          coordUp as Inf if to be adjusted
144   sameChrUp<-coords[,1]==coordsUp[,1] #identify adjacent peaks on the same
          chromosome
145   coordinates[which(sameChrUp),3]<-ifelse(rowMeans(matrix(ncol=2,coordsUp[
          sameChrUp,2:3])) - rowMeans(matrix(ncol=2,coords[sameChrUp,2:3])) < 2*
          shearSize,(rowMeans(matrix(ncol=2,coords[sameChrUp,2:3])) + rowMeans(
          matrix(ncol=2,coordsUp[sameChrUp,2:3])))/2,rowMeans(matrix(ncol=2,
          coords[sameChrUp,2:3])) + shearSize) #calculate upward coordinate
          boundaries
146   coordinates[which(!sameChrDown),2]<-ifelse(rowMeans(matrix(ncol=2,coords[!
          sameChrDown,2:3])) > shearSize,rowMeans(matrix(ncol=2,coords[!
          sameChrDown,2:3])) - shearSize,0) #calculate downward coordinate
          boundaries at starts at starts of chromosomes
```

```
147   coordinates[which(!sameChrUp),3]<-rowMeans(matrix(coords[!sameChrUp,2:3],
          ncol=2)) + shearSize #calculate downward coordinate boundaries at
          starts at ends of chromosomes
148   IDs[,1]<-object.full$annotations[peaks.found,1] #store probe IDs
149   IDs[,2]<-peaks.found #store probe numbers
150   peakList<-new("peakList",list(coordinates=coordinates,IDs=IDs,stats=stats,
          from=colnames(object$ratios),windowSize=windowSize,fdre=fdre,grid_name
          =object$grid_name))
151   return(peakList) #return detected peaks matrix
152 }
```

The function first checks that all provided datasets have undergone the `rmNAs`, `shiftByMode` and `stNormScale` normalisation procedures, which are required for the enrichment detection to work correctly, and gives a warning message if any have not (L4–9). Ratios in the first dataset with NA values (and therefore all datasets following the `rmNAs` function) are identified (L10) for exclusion. A copy of the full data is made (L11) and a new scaled `arrayData` object created without the NA values for analysis (L12). A list and two vectors are created to store windows and peaks (L13–15). Cutoff values are calculated for up to 10,000 probes, far more than would occur in actual data sets (L16). Cutoff values less than zero are set to zero (L17). The number of datasets is set as a multiplication factor (L18) and a count started (L19). A loop through chromosomes is initiated (L120). The maximum chromosome coordinate is found (L21), the `arrayData` (L22), coordinates (L23), ratios (L24) and probe IDs (L25) for the chromosome extracted and the number of probes on the chromosome stored (L26). A loop through the chromosome probes is initiated (L27), where the windows will be determined. The gap size is initiated at zero (L28) and a value defining the probes to include 'downwards' is set as the current probe, that is, no probes downwards (L29). If the current probe is within the window size of the start of the chromosome the downwards window is not included (L30). While the gap size is less than the defined window size (L31) the 'downwards' value is decreased by one (L32). If the downwards value remains above zero (that is, on the chromosome) a new gap value is calculated to the downwards probe, otherwise the gap value is set to be bigger than the window size (L33). In this way the next downward probe will be sought if the current downwards probe is above zero and its gap value is less than the window size. Once the gap size has increased beyond the window size (L34) the downward probe number is

increased by one to bring it back to the last probe within the window size (L35). The minimum ratio value in the window is compared to the window cutoff value (based on the number of probes in the window) and the probes set to TRUE, representing enriched, if it is greater (L36). This finds probes within a window that includes the current probe. The process is repeated, starting with the last determined downwards probe, this time analysing windows downwards from but not including the current probe (L37–45). The same search processes are carried out analysing probes upwards of the current probe (L46–65). The count of probes is increased (L66). When all enriched probes have been identified (L67) enrichment statuses are combined with those probes not examined (L68). If enrichment states are required by the user these are returned here (L69) and the function ends, otherwise the peak detection part of the function begins. If no enrichment is found (L70) the function returns `NULL` with a message (L71–72). If enrichment is found (L73) the `consecutive` function is run with the enriched data (L74). Ratio values are extracted (L75) and vectors created for the peak detection process (L76). A loop through the consecutive regions is initiated (L77). For regions longer than one probe (L78) a matrix is created to store maxima and minima (L77) and the ratios of the region extracted (L78). A loop through the dataset is initiated (L81) and the ratios of the dataset extracted (L82) along with the two bordering ratios (L83). Maxima are located and indicated in the matrix (L84). Means of the ratios of the region are calculated (L86) along with the two bordering means (L87) and the maxima located (L88). These maxima are treated as peaks and stored (L89). A score is calculated based on the individual dataset maxima at those positions (L90). Consecutive replicate maxima probes are identified (L91). A loop through mean maxima probes is initiated (L92) and replicate maxima containing them identified (L93). A loop through these replicate maxima ranges is initialised (L94) and the first mean maxima in these ranges is identified and the range stored (L95–98). If the replicate maxima do not contain a mean maxima, the nearest to the mean peak are stored (L101–106). Single probe regions are stored without searches (L109–114). Matrices are created to store the results to put into the `peakList` (L116–121) and filled with identified peak coordinates (L122),

ratios (L123), scores (L124) and lower peak boundaries (L125). Vectors defining adjustments are created (L126). The lower boundaries are reduced by 1 (L127) and any less than 1 (L128) are increased to 1 (L129) and the adjustment set to TRUE (L130). Reduced lower boundary coordinates are extracted (L132) and the first one adjusted to -Inf if required (L133). Matching chromosome numbers are identified (L134) and lower boundaries calculated as the smaller of halfway to the previous probe or the window size (L135). The process is repeated to get upper boundaries (L136–144). Corrections are made for the starts and ends of chromosomes (L145–147). Probe IDs and numbers are stored (L148–149) and a new (peakList) containing the results returned (L150–151).

The `peakDetection` function requires the `consecutive` function (Script 5.2) which identifies consecutive regions of enrichment and reduces the list of TRUE/FALSE enrichment values to a matrix of numerical values defining 'from' and 'to' enriched probe regions. The process is vectorised for efficiency. It has the following argument:

object A list of logical values indicating probe enrichment.

Script 5.2: consecutive: script to condense consecutive numbers into a range. Used within the `peakDetection` function to find extended regions of enrichment.

```
1  ## consecutive function ##
2  ##arguments: object (vector or matrix of values)
3  consecutive<-function(object) { #define function
4    if(is.vector(object)) object<-cbind(object,rep(1,length(object))) #add
         second column if missing
5    con<-matrix(ncol=2,nrow=0) #initialise matrix to store results
6    for (chr in unique(object[,2])) { #loop through chromosomes
7      values<-object[object[,2] == chr,1] #get current chromosome values
8      diffs<-abs(diff(values)) #calculate consecutive differences
9      diffs<-diffs != 1 #convert differences to TRUE/FALSE
10     con<-rbind(con,cbind(values[which(c(T,diffs))],values[which(c(diffs,T))
           ])) #find and store boundaries
11   }
12   return(con) #return result
13 }
```

The function adds a second column of '1's if a vector is provided, simulating all values coming from the same chromosome (L4). A matrix is created to store results (L5) and a loop of chromosome numbers initialised (L6). Values

on the current chromosome are extracted (L7) and the differences between adjacent values calculated (L8). These differences are converted to TRUE and FALSE values, TRUE representing differences other than 1, that is, non-consecutive values (L9). Boundaries of consecutive values, representing their ranges, are stored (L10) and the results returned (L12).

The `peakList` object created by the `peakDetection` function contains coordinates of PBRs, IDs of probes at the top of peaks, statistics of the peaks, the file names the peak detection was performed on, and the windowSize and FDRE values used. It has a set of methods associated with it (Script 5.3). The `show` method displays the dataset names used to create the list, the number of peaks found and the windowSize and FDRE values. The `summary` method shows the number of peaks found and summaries of the peak scores and heights. The `dim` method shows the number of peaks detected.

Script 5.3: peakList: methods to show, calculate the dimensions of, produce a summary and extract data from a `peakList` object.

```
1  ## set peakList class##
2  setClass("peakList",representation("list"))
3
4  ## peakList show method ##
5  setMethod("show", "peakList", function(object) { #define function
6    message("A \"peakList\" object created from:") #print message
7    for(n in 1:length(object$from)) message(paste("\t",object$from[n])) #print
         dataset names
8    message("Number of peaks:") #print message
9    message(paste("\t",nrow(object$IDs))) #print number of peaks
10   message("\nwindowSize:") #print message
11   message(paste("\t",object$windowSize)) #print windiowSize value
12   message("fdre:") #print message
13   message(paste("\t",object$fdre)) #print fdre value
14 }
15 )
16
17 ## peakList dim method ##
18 setMethod("dim", "peakList", function(x) { #define function
19   return(dim(as.matrix(ncol=1,x$IDs))) #dimensions relate to IDs
20 }
21 )
22
23 ##peakList summary method ##
24 setMethod("summary", "peakList", function(object) { #define function
25   message("Summary of peakList object") #print message
26   message("Number of peaks:") #print message
27   message(paste("\t",nrow(object$IDs))) #print number of peaks
28   message("Peak score statistics: ") #print message
29   scores<-as.matrix(table(object$stats[,1])) #get scores table
30   rownames(scores)<-round(as.numeric(rownames(scores)),2) #rename rows
31   colnames(scores)<-"Count" #rename column
```

```
32    print(scores) #print scores table
33    message("Peak height statistics: ") #print message
34    heights<-as.matrix(summary(object$stats[,2])) #get peak heights summary
35    colnames(heights)<-"Value" #rename column
36    print(heights) #print peak heights summary
37  }
38  )
39
40  ## peakList extract method ##
41  setMethod("[", "peakList", function(x,i,...) { #define function
42    if (nargs() != 2) stop("One subscript required", call. = FALSE) #check
          only one subscript (for rows)
43    return(new("peakList",list(coordinates=matrix(x$coordinates[i,],ncol=3),
          IDs=matrix(x$IDs[i,],ncol=2),stats=matrix(x$stats[i,],ncol=ncol(x$
          stats)),from=x$from,windowSize=x$windowSize,fdre=x$fdre,add=x$add,grid
          _name=x$grid_name))) #return new genomeAnnotation object
44  }
45  )
```

The `peakList` class is defined (L2). The `show` method (L5) prints dataset names (L6), the number of peaks (L9) and the "windowSize" (L11) and "FDRE" (L13) values used. The `dim` method (L18) returns dimensions of the "IDs" slot (L19), with the number of columns always 1. The `summary` method (L24) prints the number of peaks (L27), a matrix containing peak score counts (L29–32) and a matrix containing a peak height summary (L34–36). The extract method (L41) checks one argument is provided (L42) and returns data for the specified peaks (rows) (L43).

5.2.1 Window determination

To examine a dataset for enrichment it must first be divided into a number of subsets of probes, termed windows. Each window contains one or more probes, depending on the probe resolution, and represents a genome region likely to show the effect of enrichment in its vicinity. This is due to there being multiple probes to which the chromatin fragments from a region can bind, resulting in all probes covering the region showing enrichment. This is demonstrated in Figure 5.1. All fragmented, immunoprecipitated chromatin fragments will contain the binding site, represented by the blue dot, but fewer fragments will cover regions at greater distances from the binding site. Therefore the greatest distance from a binding site that can be covered by a fragment is equal to the fragment length, and so the region of enrichment is equal to twice the fragment length. A window size equal to this length

should therefore be capable of containing only enriched probes when centred over an enriched region. As the fragmentation process is random, an average chromatin shear size is used to determine this length. In this way, anomalous peaks occurring at single probes can often be disregarded as the surrounding probes in the window will not show enrichment. Probe values in each window are analysed and called as positive if determined to represent enrichment.

Smaller window sizes encompass few probes and are therefore not very stringent: it is possible for a spurious high probe to be the only value in a window and so out of the context of the surrounding low values may be called as positive. This results in a low specificity. Larger window sizes encompass many probes and can therefore be too stringent: it is possible for high values as a result of real enrichment to be masked by the surrounding low values on either side. This results in a low sensitivity. The optimum window size will allow regions of genuine enrichment to fill the window while keeping anomalous high values in the context of their surrounding values. This is illustrated in Figure 5.2.

The peak detection method presented here employs a sliding window approach, whereby a window of defined length is 'slid' along the data *in silico*. At defined points processing of the data in the window is carried out. In other sliding window methods these points are termed the 'step size' and determine the distance the window is moved between sets of calculations (Chipotle, Buck et al., 2005, for example). Figure 5.3 shows a representation of these steps to create different windows, using the smallest possible step size of one (black lines) and, as is more commonly used in peak detection methods, an appropriate larger value (red lines). Sliding the window by a defined length like this has two significant disadvantages. Firstly, as the arrangement of probes is not uniform, some probe combinations may be bypassed by the window as it takes a step beyond the particular configuration. For example, in Figure 5.3 there are 6 possible unique probe combinations with the step size applied (15 nt), but the sliding window with a step size of 5 does not find all of these. Conversely, the step size may not bring about a new combination of probes and so the same window is 'found' multiple times, wasting computational resources. Additionally, at regions with no

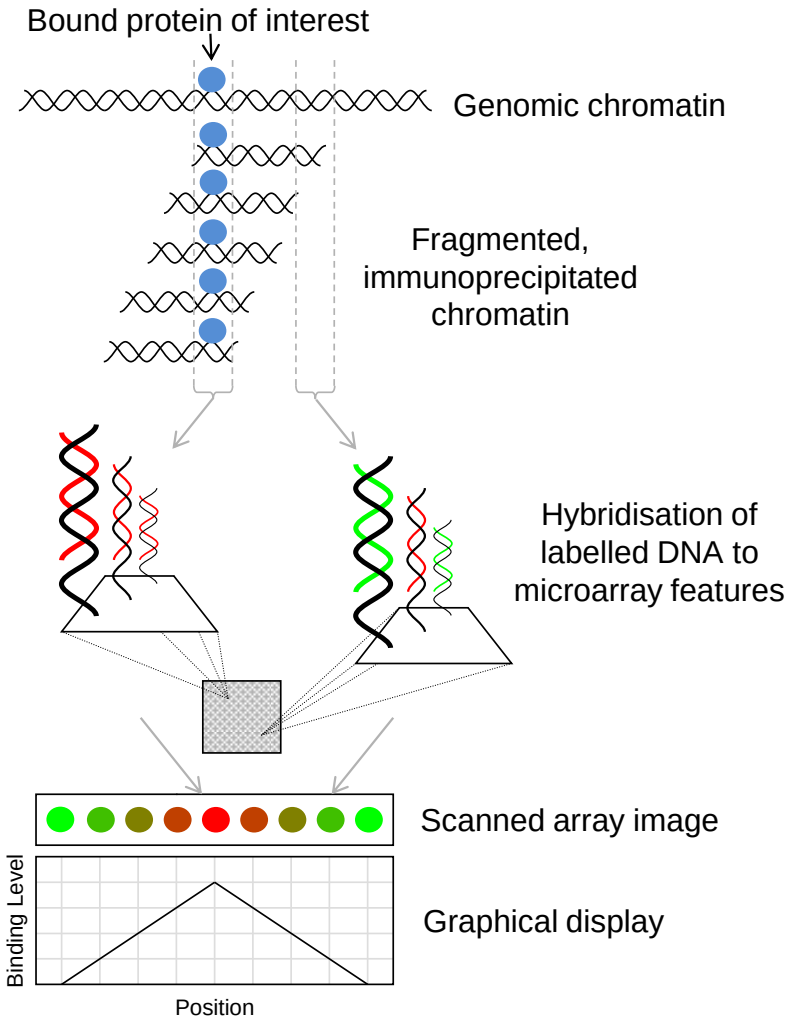


Figure 5.1: Representation of the formation of a peak shape: Overlapping chromatin fragments binding to several probes creating a peak shape in plotted data. Chromatin is shown as a DNA helix for clarity. Fluorescent labelling of the input(green) and immunoprecipitated (red) samples is represented by coloured molecules. An immunoprecipitated protein of interest (blue circle) can be present on a series of chromatin fragments. Following random binding to the microarray the probe closest to the protein binding site will have the most intense red signal because more immunoprecipitated fragments cover this region, with those further away having reduced red signals because fewer immunoprecipitated fragments cover the regions. Plotting the red/green ratios gives a peak shape centring on the protein binding site with a base approximately equal to twice the sheared chromatin length.

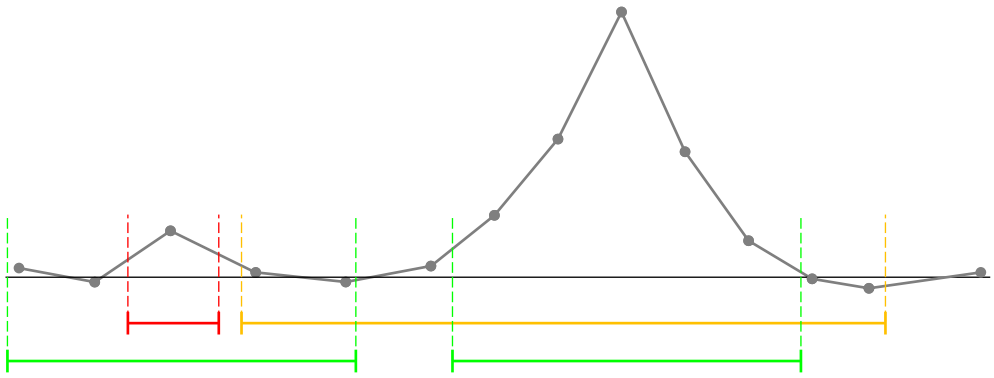


Figure 5.2: How different window sizes affect enrichment detection: Too small a window size (red) will allow anomalous high probe values in data to be detected, by taking them out of the context of their surrounding low-value probes. Too large a window size (orange) will prevent genuine regions of enrichment from being detected by extending beyond peaks to include low-valued probes. The correct window size (green) will overcome these problems, allowing only genuine peaks to be detected.

probe coverage the window will continue to search for probes where none will be found, again wasting resources. The first problem can be eliminated by setting a small step size but this greatly exacerbates the second. This is shown in Figure 5.3 where all possible windows and probe combinations are generated with the step size of one, with the creation of many unnecessary windows in the process.

To eliminate these problems the windows used by this algorithm do not follow a defined step size, but are instead calculated to analyse every possible unique window probe combination. This is equivalent to using a step size of 1, in the sense that all probe combinations will be found and analysed, but is much more computationally efficient than creating every possible window. This is achieved by creating windows only in the regions surrounding probes (Figure 5.4). All windows can be found by taking 4 windows at each probe: upwards from the probe coordinate; upwards from one above the probe coordinate; downwards from the probe coordinate and downwards from one below the probe coordinate. This creates two windows containing the probe and two not. This allows all unique probe combinations for any given window size to be found without the need to search through the whole genome. To test this, a script was written to determine every possible probe combination on the G4493A microarray with a window size of 600 bp by using a step size of 1 through the whole yeast genome. This brute force method took approximately 8,500 s to run (on a desktop PC with a 3.20 GHz Intel i7 processor and 24 GB of RAM) and found 82,465 unique windows. The script presented here, which finds windows in the way described above in a vectorised manner, takes less than 4 s, over 2,000 times faster, and finds the same 82,465 windows. These windows form the basis of the rest of the peak detection procedure.

For the purposes of peak detection the coordinate of each probe is calculated as the average of the start and end coordinates, that is, the mid-point of the probe. This means that during analyses a region is deemed to contain a probe if it contains this mid-point coordinate. In practice this means that at least half of a probe must overlap region for it to be considered 'in' the region. There are various ways a probe could be defined as in a region, such

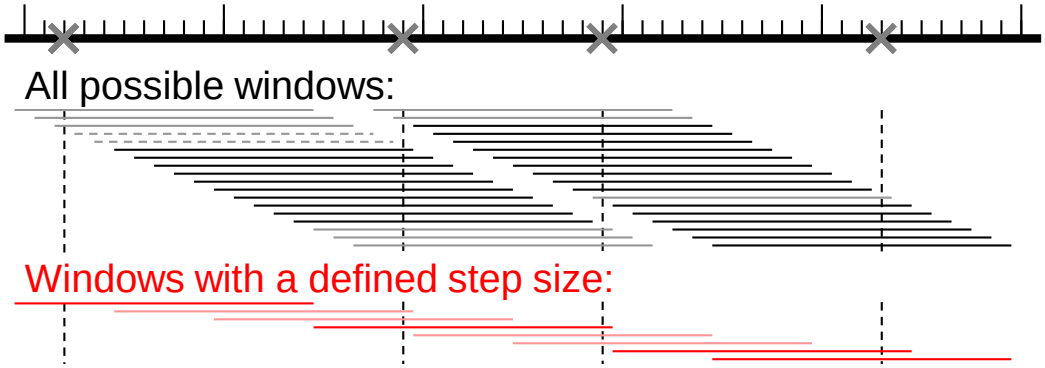


Figure 5.3: Examples of sliding windows: Representation of the windows that could be generated in the search for enriched probes. The thick black line shows a section of genome with each nucleotide position marked by a dash. Probe positions are shown with grey crosses. For clarity probes are shown at a much higher resolution than on the microarrays and a window size of 10 nucleotides is used, creating 6 unique windows. Probe positions are highlighted with vertical dashed lines. All possible window combinations (36 windows) find all unique windows with a lot of redundancy. A sliding window with a step size of 5 (8 windows) does not find all unique windows. Shading of windows highlights those with the same combinations of probes.

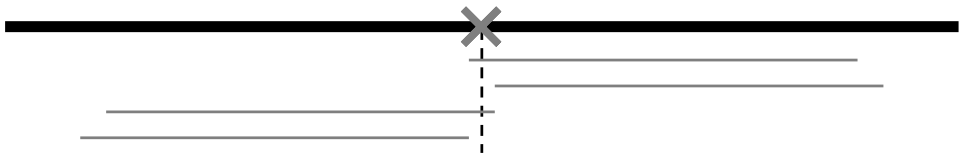


Figure 5.4: Representation of window determination: For each probe (represented by a cross) four windows are created: one upwards including the probe, one upwards adjacent to but not including the probe, one downwards including the probe and one downwards adjacent to but not including the probe.

as requiring the whole probe or at least a section to be within the region. In real data the effect of binding to a probe will gradually diminish with distance from it and so it is impossible to define a definitive cutoff point for these calculations. The mid-point was chosen as an average of these two extremes and for computational efficiency as only a single coordinate is needed to define the probe, not the two coordinates of the extremes.

The speed of the script has been increased by improving the efficiency of the determination of which probes to include in each window. The most simple way of achieving this in R is with the `which` function in the form `which (probeCoordinates >= windowStart & probeCoordinates <= windowEnd)`. This performs a search of all probe coordinates and returns those that match the criteria, in this case those that are in the specified window. Although this `which` command is efficient, repeating it four times for every probe on a microarray to get all windows slows the function down considerably. This process was therefore replaced with a series of loops which limit the search process to the region around each probe, rather than the whole chromosome. The loop starts at the coordinate of the probe being examined to find the window starting at and including the probe. The gap between this and the coordinate of the next probe is calculated and compared to the window size. If it is less, the gap to the next probe is calculated, and so this loop continues. When the gap is greater than the window size the loop stops and the binding values from the range of the probe being examined to the probe preceding that with a gap greater than the window size is analysed. This is then extended a further step to get the window starting immediately adjacent to the probe being examined. The same process is repeated in the opposite direction to get windows upstream and downstream of the probe being examined. The ratios in each window are analysed to determine whether or not they represent an enriched region, based on the cutoff value for the window, as described in the following sections. When all four windows have been analysed the probe being examined is incremented and the process repeated. This computational process is shown as a flow chart in Figure 5.5 and represented graphically in Figure 5.6.

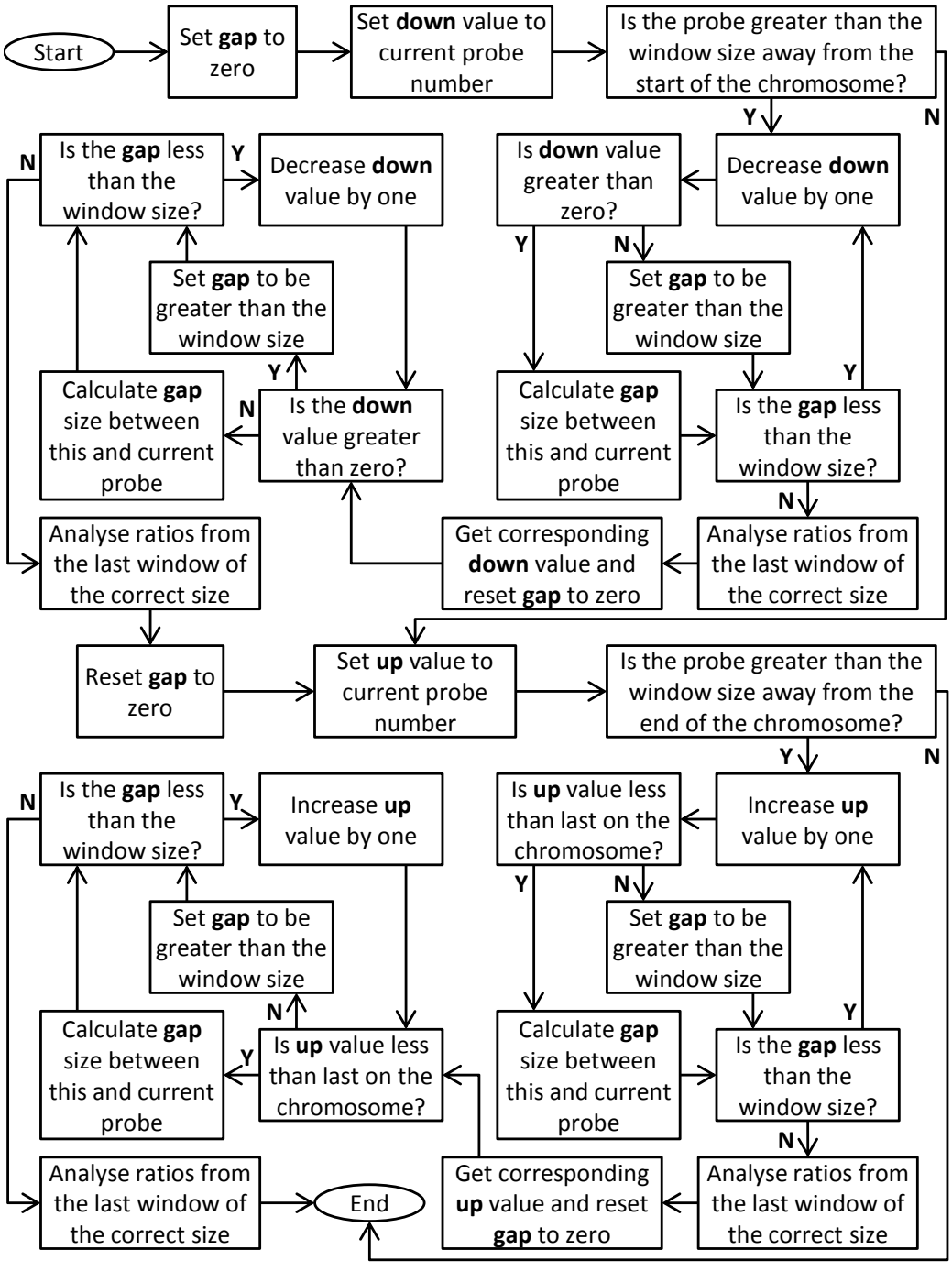


Figure 5.5: Window determination and enrichment detection process flow chart: The computational processes of the enrichment detection process, including the window determination and ratio cutoff analysis.

Step	d	u	gap	Action
1	5		<	Decrease
2	4		<	Decrease
3	3		<	Decrease
4	2		<	Decrease
5	1		>	Return
6	2		NA	Analyse (2,3,4,5)
7	2		<	Decrease
8	1		<	Decrease
9	0		>	Return
10	1		NA	Analyse (1,2,3,4)
11		5	<	Increase
12		6	<	Increase
13		7	<	Increase
14		8	<	Increase
15		9	>	Return
16		8	NA	Analyse (5,6,7,8)
17		8	<	Increase
18		9	>	Return
19		8	NA	Analyse (6,7,8)

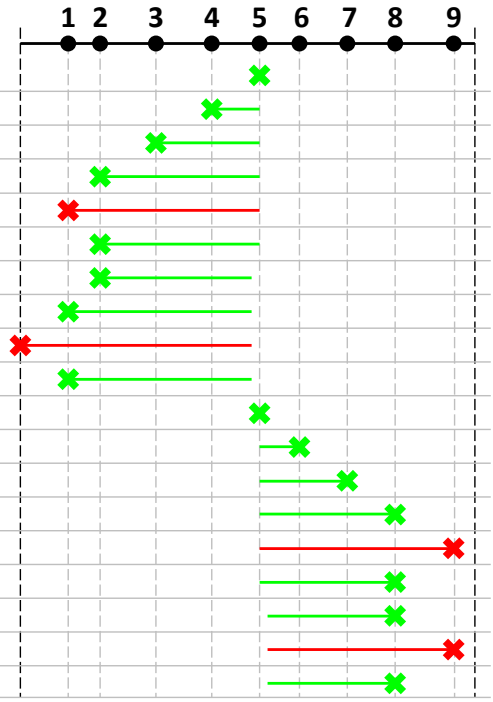


Figure 5.6: Representation of window determination and enrichment detection: The grid on the left represents the values and actions of the algorithm, displayed in the graphic on the right. 'd' is the probe number being tested downwards from the current probe. 'u' is the probe number being tested upward from the current probe. 'gap' shows if the gap between the probes is greater or less than the window size. 'Action' shows the step taken by the algorithm. The 'd' value is initially set to the current probe value (demonstrated here with probe 5), shown with a green cross. In step 1 the gap is less than the window size and so the action is to decrease the 'd' value. This is repeated for steps 2 to 4 with the window being tested shown with a green line. In step 5 the 'd' value gives a gap larger than the window size, indicated by a red line, and so the action is to return to the previous 'd' value. The values of all probes in this range are then analysed (step 6). The 'd' value is then decreased again in step 7, this time to calculate the gap to the point immediately preceding the current probe. The process is repeated, analysing a new window in step 10. Once both downward windows have been determine, the process is repeated in the upward direction, starting with a 'u' value at the current probe (step 11).

5.2.1.1 Cutoff calculation

The algorithm requires a user defined false discovery rate equivalent (FDRE) value of the number of false peaks to 'find'. Setting this value at less than one means that the cutoff values are maintained at a level that, statistically, no false peaks should be found. The default value for this argument is therefore 0.9. In theory, there will be a limit to the genuine peaks that can be detected in a dataset, due to its properties, including the sizes of the genuine peaks and the level of noise in the background. The default FDRE value of 0.9 is intended to find the maximum number of detectable genuine peaks without detecting any false peaks. Smaller values will result in fewer genuine peaks being detected, reducing the sensitivity, without affecting the specificity. Larger values will result in more false peaks being detected, reducing the specificity, without affecting the sensitivity. This process works in the reverse manner to traditional multiple testing corrections, where a series of tests produce a series of probability values to which a multiple testing correction is applied. The aim of this correction is to remove any significant results caused by chance alone, maintaining only those that occur due to genuine biological factors. Different corrections have varying levels of conservation and so remove varying numbers of results. This means that, even after applying the correction, some of the remaining results may still be false positives and some true positives may be removed. Adjusting the cutoff value for each window removes the need to apply any correction to the final results, meaning they can be treated with greater confidence.

The defined number of false peaks it is statistically acceptable to find is adjusted to take into account the total number of probes in the dataset, such that the overall level stays at the specified level regardless of how many probes are being analysed. For example, in a dataset of a single probe, that probe must have a value over -1.281552 to be called a peak at the level of significance which seeks to find 0.9 false peaks. In a dataset of 44,000 probes, a single probe in a window containing only that probe must have a value over 4.102284 to meet the same statistical requirements. This larger value reflects the fact that, statistically, more false peaks will be found in 44,000 probes

than 1 probe. The higher cutoff counteracts this as fewer probes will be over the higher value, so all the peaks found from the 44,000 probes should be statistically genuine.

As the number of probes in a window increases, the cutoff value is reduced while maintaining the same probability level, because the individual levels are multiplied together to give the overall probability. In the single-probe window from 44,000 probes example above, the probability level is 2.045452×10^{-5}, that is, the probability that the value is from the normal background population is 2.045452×10^{-5}, so it is actually very unlikely to have come from that population and is more likely to have come from the population of genuine peaks. If the window contained two probes, the cutoff value would reduce to 2.610336 which has a probability of 4.522667×10^{-3}. $4.522667 \times 10^{-3} \times 4.522667 \times 10^{-3} = 2.045452 \times 10^{-5}$, that is, the overall probability level of the cutoff value of the window containing two probes is the same as the window containing a single probe.

Replicates can also contribute to this. For example, a window containing 3 probes, each of which has 3 replicates is treated as containing a total of 9 probes. The cutoff value for this is 0.52071 which has a probability of 0.3012844. $0.3012844^9 = 2.045452 \times 10^{-5}$, that is, the same overall probability as before. So the more probes in a window (from probe dense regions or/and replicates) the lower the cutoff value can be to achieve the same probability level, thus increasing the detection sensitivity. In doing this the overall probability of finding a false peak is maintained at the level originally specified and so no correction needs to be applied to the final results. The results presented later show that this may not be beneficial in all datasets.

Cutoff values are calculated at the beginning of the algorithm so as to save computational time by not having to perform the calculation for every window. Cutoff values less than zero are set to zero, so only positive values can be found as enriched.

5.2.2 Enrichment detection

Enrichment detection takes place by examining the probe ratios in each window and recording which windows, if any, contain probes over the cutoff. It is likely that many probes will appear in more than one window. These probes need only be in one window with all values over the cutoff to be recorded as enriched. All probes in the window are required to be over the cutoff to avoid the detection of spurious high values caused by events other than enrichment (see Figure 5.7). At sites of genuine enrichment, a region approximately double the average chromatin shear size will be immunoprecipitated and the resulting peak will have a base approximately the same width (assuming a high probe coverage, otherwise the peak is theoretical). There will therefore be a region of values higher than background over this region. However, the smaller values at the two extremes of this region may be indistinguishable from the background. The window size is therefore set to the average chromatin shear size so that it can cover the central portion of the peak and analyse only these larger values, without the influence of the smaller values. Therefore windows over the centres of genuine peaks should contain values which are all above the cutoff value and be recorded as enriched. It may also be the case that windows covering the regions either side of the centre of the peak will contain values over the cutoff and so these too will be recorded as enriched, resulting in the detection of a larger enriched region. High values due to other factors are likely to affect single probes only, possibly extending to multiple probes in rare circumstances. It is unlikely that these will fill a whole window with values over the cutoff and so these will not be recorded as enriched by this method. It is also very unlikely that these spurious high values will occur at the same points in multiple datasets and so combining all data in the same analysis provides another method by which these regions are not detected as enriched. Figure 5.7 demonstrates this process over a short section of data, showing four windows detecting four probes as enriched. The final probe is not detected as enriched, even though it has a higher value that some of the other detected probes, because there are fewer probes in the window and so the cutoff is higher.

Following this stage of the algorithm every probe is assigned a `TRUE` or `FALSE` status representing whether or not it has been detected as enriched. If only enrichment detection is required by the user these statuses are returned, indicating all probes which the algorithm has determined to be showing enrichment. This is useful for conditions which do not exhibit distinct binding sites and therefore do not contain a series of peaks, such as histone modifications. These modifications may span regions of hundreds or thousands of nucleotides and so reducing this information to the location of a single peak is not biologically informative. Rather, the whole modified region is required which can then be analysed further by the user, depending on the aims of their investigation.

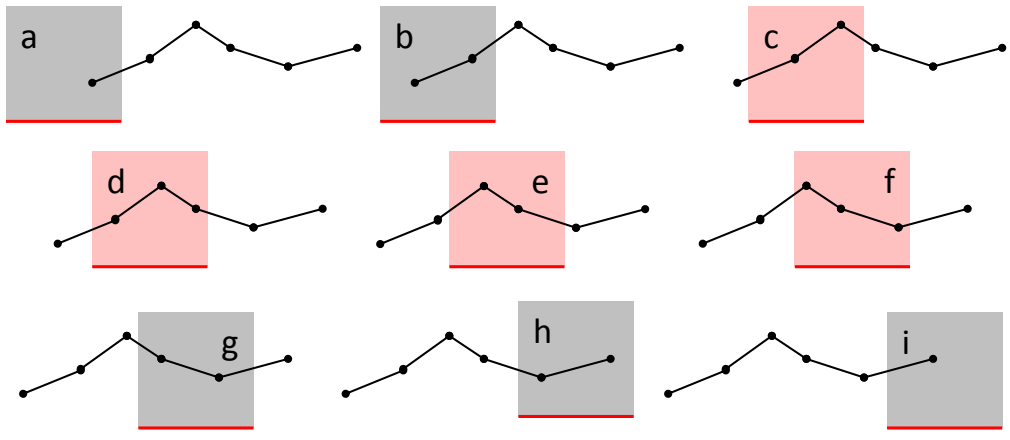
Because this method requires all of the values in a window to be above the cutoff it is only suited to good quality data with consistent replicates. Only a single probe need be less than the cutoff for the window not to be found, so a peak present in some datasets but not others will not be found by this method. This can be overcome by analysing the means of the datasets, which should create peaks above the cutoff value in area where enrichment is present in some of the datasets. This will result in a reduced ability to detect small peaks, as fewer probes will be analysed and so larger cutoff values will be calculated. However, it is unlikely that any such peaks could be reliably determined in poor quality data and so this averaging should not reduce the overall ability to detect regions of enrichment.

For investigations requiring peak detection the next stage of the algorithm can be invoked.

5.2.3 Peak detection

The peak detection stage of the algorithm is used to identify likely binding sites of immunoprecipitated molecules that bind at distinct sites, including proteins. All regions detected as enriched are analysed and a series of binding region coordinates are produced. The coordinates are based on the probe coordinates at the peak.

The enriched probes are first broken down into regions, each one repre-



Window	Number of probes	Lowest value	Window cutoff	Enriched?
a	1	1	4.1	N
b	2	1	2.6	N
c	2	3	2.6	Y
d	3	3	1.9	Y
e	2	4	2.6	Y
f	3	2	1.9	Y
g	2	2	2.6	N
h	2	2	2.6	N
i	1	4	4.1	N

Figure 5.7: Enrichment detection representation: A short section of example data showing 9 possible windows. For each window the number of probes, its lowest value and the calculated cutoff is shown. From these the enriched windows (those with their lowest values above the window cutoff) are found, which are highlighted as red.

senting each run of enriched probes. These may therefore range in length from one to scores of probes. Each of these regions is analysed in turn for binding peaks. The algorithm analyses all individual datasets as well as the means of these datasets when determining peak positions. Peak calling is based on the average values of the region, where every probe at a peak, that is, a probe with a value higher than its two adjacent probes, is recorded as a potential binding probe. This is based on the assumption that a genuine binding peak will occur in all replicate datasets and also therefore in the average of the datasets. Spurious, small peaks occurring in a small number of datasets are unlikely to also occur in the averaged dataset, depending on the number of replicates, and so these regions will not be detected as peaks.

Following this initial process all individual datasets are analysed at the detected sites to determine whether or not they also contain peaks at or near to the same probe. Peaks that occur at the same probe in the averaged and all individual datasets are given a score of 1, representing the fact that this is very likely to represent a probe near a genuine binding site. The process is outlined as a flow chart in Figure 5.8 and represented graphically in Figure 5.9.

It is possible, due to the resolution of the technology, that a genuine binding site will manifest as peaks at close, adjacent probes in different datasets. This fact is taken into account in situations where not all replicate datasets contain a peak at the same site as the averaged dataset. Here, probes within the average chromatin shear size of the probe detected from the averaged dataset are also analysed for peaks. If peaks are present in these regions the probe detected in the averaged dataset is recored as a peak. The score is calculated as a fraction of the number of individual datasets also present at the same site. For example, if two out of three replicates contain a peak at the same probe the score is $2/3 = 0.\dot{6}$.

As well as detecting and scoring peaks, the algorithm calculates a range over which the genuine binding site is likely to be located based on probe coordinates and the average chromatin shear size, termed the potential binding region (PBR). For peaks with a score of 1 this is calculated as the smaller of the average chromatin shear size or half the distance to the previous/next

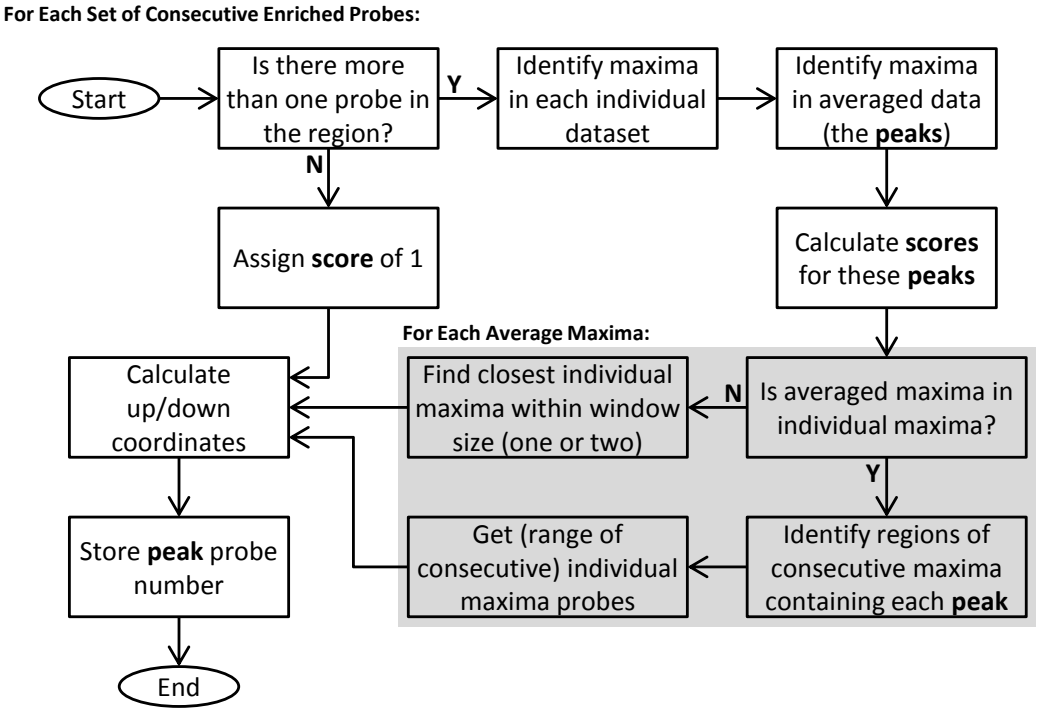
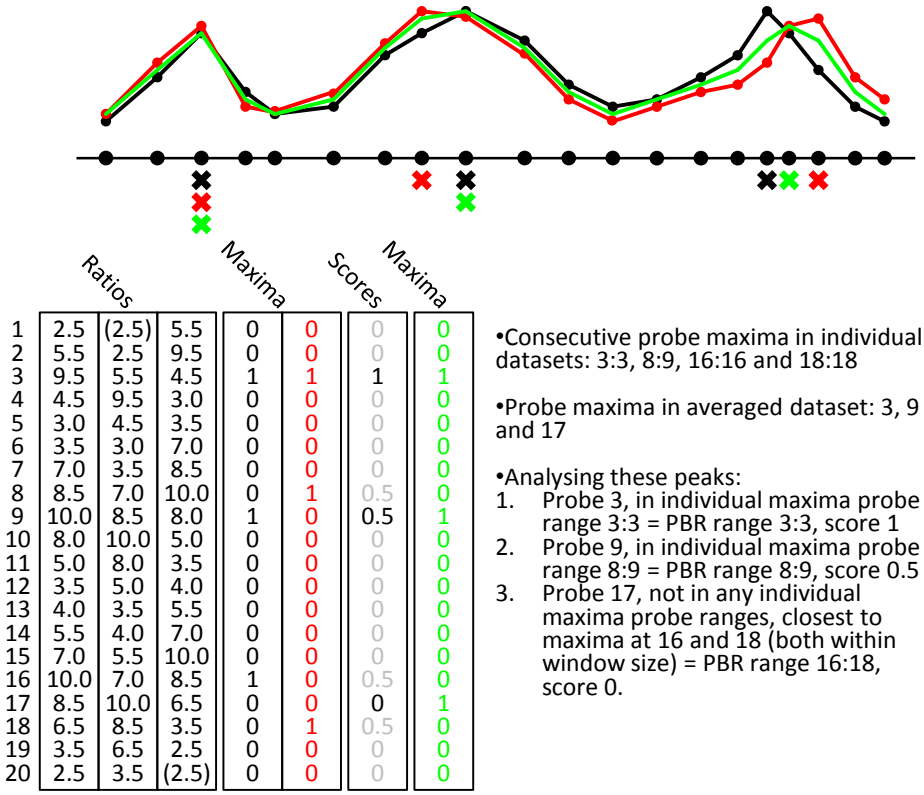


Figure 5.8: Peak detection process flow chart: The computational processes of the peak detection process, which takes the results from the enrichment detection to find potential binding regions.



1	2.5	(2.5)	5.5	0	0	0	0
2	5.5	2.5	9.5	0	0	0	0
3	9.5	5.5	4.5	1	1	1	1
4	4.5	9.5	3.0	0	0	0	0
5	3.0	4.5	3.5	0	0	0	0
6	3.5	3.0	7.0	0	0	0	0
7	7.0	3.5	8.5	0	0	0	0
8	8.5	7.0	10.0	0	1	0.5	0
9	10.0	8.5	8.0	1	0	0.5	1
10	8.0	10.0	5.0	0	0	0	0
11	5.0	8.0	3.5	0	0	0	0
12	3.5	5.0	4.0	0	0	0	0
13	4.0	3.5	5.5	0	0	0	0
14	5.5	4.0	7.0	0	0	0	0
15	7.0	5.5	10.0	0	0	0	0
16	10.0	7.0	8.5	1	0	0.5	0
17	8.5	10.0	6.5	0	0	0	1
18	6.5	8.5	3.5	0	1	0.5	0
19	3.5	6.5	2.5	0	0	0	0
20	2.5	3.5	(2.5)	0	0	0	0

Figure 5.9: Representation of peak determination: The top graphic shows a genome section with probe positions indicated by crosses. Black and red lines indicate two datasets with their mean shown in green. The columns of values represent the computational process of finding peaks or peak regions. The first 'Ratios' column shows the values of the black data line. The next two show the same values offset by one position in both directions, with the values in brackets being the resulting gaps filled with the original value. These three columns are analysed for maxima, occurring where a value in the first column is larger than the equivalent value in the next two columns. These sites are indicated with values of 1 in the black 'Maxima' column. The red 'Maxima' column shows the same results for the red data. Values in the 'Scores' column are calculated as the mean of these values. Peak sites are determined by the positions of maxima in the averaged data, shown in the green 'Maxima' column and highlighted in the 'Scores' column. Consecutive probe maxima in individual and averaged datasets are calculated and each averaged dataset maximum is analysed in the context of the individual dataset maxima to determine the potential binding region which is stored with the score, as shown in the text.

probe (Figure 5.10). This is based on two assumptions. The first is that the genuine binding site must lie within one average chromatin shear size length of the peak probe as beyond this range the probe will not detect the enrichment. This is relevant in situations where the distance between probes is greater than the average chromatin shear size and this lack of resolution is reflected in the large potential binding region. The second assumption is relevant where the distance between probes is less than the average chromatin shear size and is that the genuine binding site must lie closer to the peak probe than its adjacent probes. If this were not the case the adjacent probe would be at the top of the peak.

For peaks with a score less than 1, the same procedure is applied in calculating the likely binding region taking into account all of the probes from all replicate datasets that are considered to make up the peak (Figure 5.11). This extends the region, reflecting the fact that the peaks in differing positions reduce the certainty in determining the likely binding region.

The final result of this peak detection process is a six column matrix containing the chromosome, start and end coordinates of the potential binding region, the unique probe ID and $\log_2$ binding value of the probe at the top of the peak (taken from the averaged dataset).

5.3 Testing the performance of the algorithm

Testing the performance of enrichment detection algorithms, and comparing the results of different algorithms, poses a problem. The nature of ChIP-chip datasets means that many, if not all, of the genuine biological binding sites are not known. Therefore there is no way of comparing the performance of one algorithm with another, because it is not possible to know which results from the two are correct or incorrect. The accuracy of different methods is therefore difficult to assess. Although a subset of microarray results should always be validated by other techniques, it is impractical to validate every single result, both positive and negative, and so it cannot be known with absolute certainty which results are correct or incorrect.

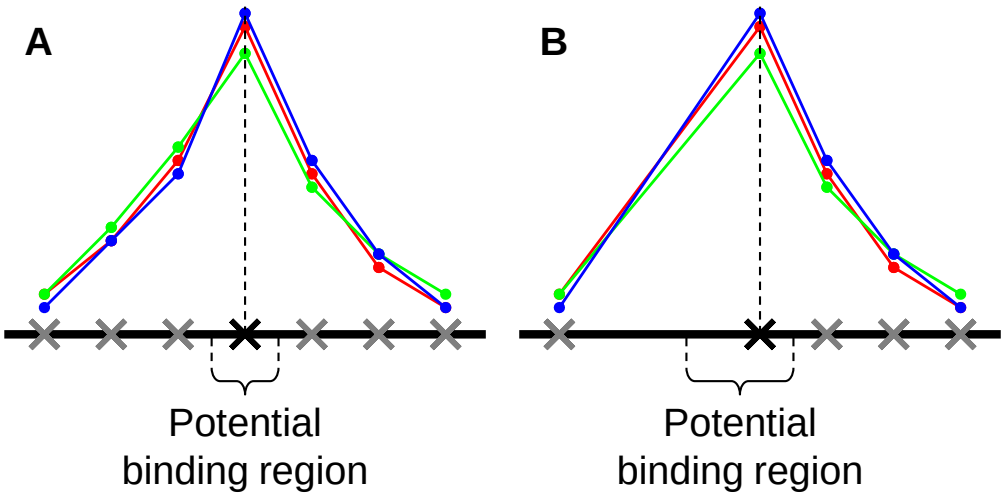


Figure 5.10: Calculating the PBR with consistent peaks: Crosses represent probes, the probe at the peak is highlighted; coloured lines represent individual datasets. A - Where adjacent probes are closer than the average chromatin shear size the distance is set as half the distance to the adjacent probe. B - Where adjacent probes are further than the average chromatin shear size the distance is limited to the average chromatin shear size.

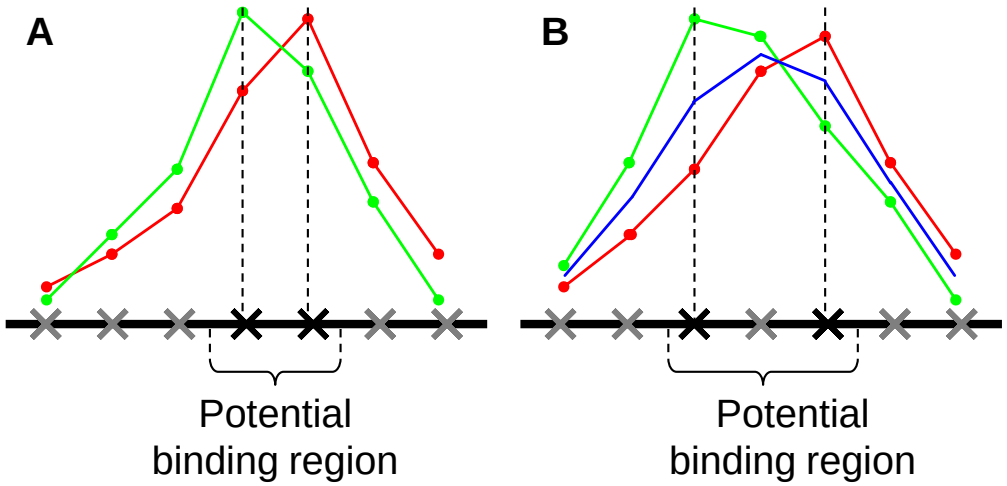


Figure 5.11: Calculating the PBR with inconsistent peaks: Crosses represent probes, the probes at the peaks are highlighted; green and red lines represent individual datasets. The region is calculated as in Figure 5.10 from the probes at the two extremes of the range of peaks. A - peaks occurring at adjacent probes form a potential binding region spanning those probes. B - peaks occurring at non adjacent probes are identified by a peak in the averaged data (blue line) and the potential binding region spans several probes.

5.3.1 Data

Two sets of data have been used here in an attempt to overcome the above problem and assess the performance of this algorithm compared to other published algorithms. The first is the creation of artificial datasets, designed to mimic the expected output of a microarray by simulating binding sites at known locations. The second is the use of a series of spike datasets produced by various labs, presented by Johnson et al. (2008), where a number of known locations have been artificially enriched to varying degrees. These datasets were created to evaluate variability in ChIP-chip experiments and peak detection algorithms. As locations of genuine enrichment (both simulated and real) are known for these datasets, the performance of the enrichment detection algorithm can be assessed. Comparing the regions detected by the algorithm with the genuine regions of enrichment allows the numbers of true and false positive and negative results to be calculated. These values can be used to determine the sensitivity and specificity of the algorithm (outlined later), giving a measure of its accuracy. They are then compared to the results created by other algorithms analysing the same datasets, to compare their performances.

5.3.1.1 Creating simulated ChIP-chip data

Simulated ChIP-chip data were created to test the performance of the enrichment detection process, based on the probe arrangement of the G4493A microarray. From this layout, 2000 probes were randomly selected to represent 'peaks'. The `predictProfile` function (Chapter 6) was used to generate a dataset based on these positions, with randomly generated values representing peak heights. A window size of 600 was used. Properties of the Abf1 datasets (no UV treatment; see Chapter 7 for details) were used to make the simulated data represent a set of real ChIP-chip data as accurately as possible. The largest value in these raw datasets is around 6.8, so this was used as an estimate of the maximum peak height to simulate. Four sets of height values were generated: small (0.5 to 2.5), medium (2.5 to 4.5), large (4.5 to 6.5) and a combination of all of these (0.5 to 6.5). Noise, in the form

of randomly generated normally distributed values, was added to the data, to simulate the noise associated with real data. The standard deviation of the estimated background data from the Abf1 datasets was calculated and a normal distribution with a mean of zero and standard deviation of this value, 0.31, applied.

For each set of randomly generated peaks, five different sets of random normally distributed values were applied. This was intended to simulate five replicate datasets, which all have the peaks present at the same positions but are subject to different random noise. As the noise is applied to peaks as well as the background, the heights and shapes of the peaks will be different in each dataset, depending on the random values applied to them. In addition, for each dataset 500 randomly generated probes had their values increased by a randomly generated value between 0 and 3, to simulate small, spurious peaks which may occur in data. This allowed the performance of the peak detection to be assessed with single and multiple datasets. Peak positions, heights and noise were applied blind, so as not to introduce any user bias in the analyses. The full normalisation procedure was applied to each dataset before peak detection was applied, to replicate the same procedure real ChIP-chip data is subject to. Peak detection was set to find peaks, resulting in a list of probes determined to be closest to the binding region.

5.3.1.2 Using spike datasets

Spike datasets, produced as a means to test the performance of different microarray platforms and peak detection methods, is presented by Johnson et al. (2008). These microarrays contain probes covering regions selected by the ENCODE consortium, covering 1% of the human genome. The spikes consist of 100 samples of cloned genomic DNA sequences of average length 497 bp, added at concentrations ranging from 1.25- to 196-fold above the background — a commercial human genomic DNA preparation. Datasets were downloaded from the Gene Expression Omnibus (`http://www.ncbi.nlm.nih.gov/geo`) under accession number GSE10114. In total, 7 individual Agilent datasets were produced: 3 produced by Myers et al.

containing undiluted spikes and 4 produced by McCuire et al., 2 containing undiluted spikes and 2 containing diluted spikes which went through a PCR amplification procedure prior to their application to the microarray. These datasets were analysed individually and in various combinations, shown in Table 5.2 along with the amplification status of each.

The consistency of the datasets was investigated by creating scatter plots of all replicates, shown in Figure 5.12 along with Spearman's rank correlation values. These show that there is very little correlation between any two datasets, with the best correlation value being 0.5. While the bulk of the data points should not show any correlation, under the assumption that these are from the background sub-population, there should be a small but significant subset of probes from the enriched sub-population which do show a correlation. While this is partially apparent in some comparisons, all plots also show higher values in one dataset which are not present in the other. Compared to the Abf1 datasets (see Figure 7.7), which have good correlations between replicates in their enriched sub-population regions, the plots created here suggest that the spike datasets may not be well suited to the requirements of this algorithm, as outlined in Section 5.2.2. Averaged datasets were therefore analysed along with combined datasets. As previously discussed, this can increase the ability of the algorithm to detect the larger peaks, possibly at the expense of smaller ones.

5.3.2 Optimisation of the algorithm

The algorithm was applied to the simulated and spike datasets with a number of different settings, to test its performance under different conditions and determine the optimal values for the different arguments. There are three user-modifiable arguments that can influence the results of the algorithm: the window size, FDRE value and scale factor. The algorithm was run with a range values for each of these, centred around the value expected to give the optimal results, and extending to values thought likely to give poor results. This range of results allowed ROC plots to be created and an estimate of the value producing the best results to be made, be this the expected optimum

Number	Dataset(s)	Amplified
1	Myers 1	No
2	Myers 2	No
3	Myers 3	No
4	Myers 1 & 2	No
5	Myers 2 & 3	No
6	Myers 1 & 3	No
7	Myers 1, 2 & 3	No
8	McCuire 1	No
9	McCuire 2	No
10	McCuire 1 & 2	No
11	McCuire 3	Yes
12	McCuire 4	Yes
13	McCuire 3 & 4	Yes
14	Myers 1, 2 & 3 and McCuire 1 & 2	No

Table 5.2: Johnson et al. (2008) spike datasets: Combinations of Agilent spike datasets presented by Johnson et al. (2008), showing the group that created the data and the amplification status. Colours are as used in later plots of the data and indicate single (grey), pairs (black, Myers only), full (red) and combined (blue) datasets.

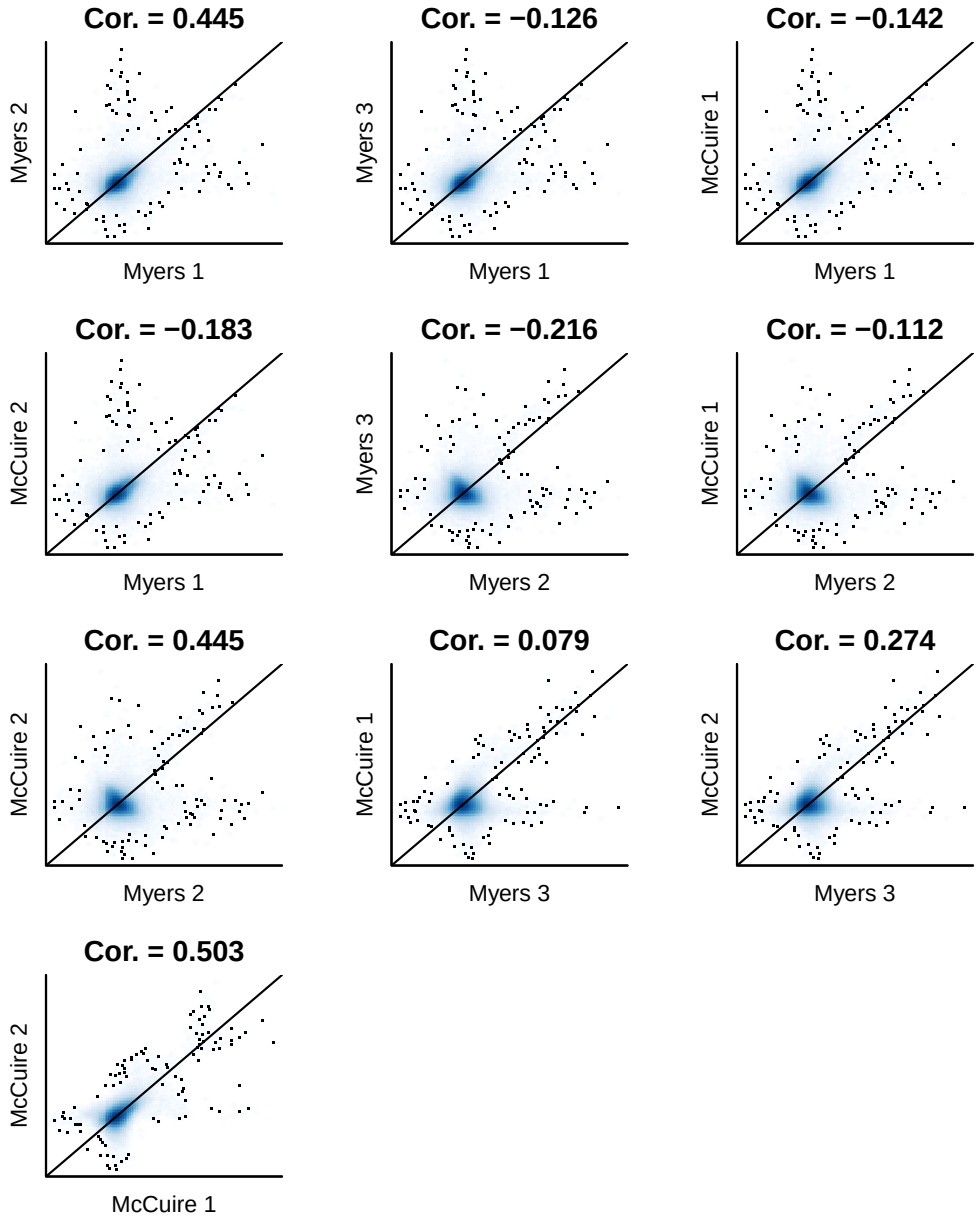


Figure 5.12: Johnson et al. (2008) data correlations: The darker the blue colour the more points occur in the region. Individual points outside of the central region are shown with dots. The black line shows $y = x$. The Spearman rank correlation value is shown above each plot.

or not.

The algorithm was first applied with a range of different window sizes, centred on the expected optimal values for the two sets of data: 600 for the simulated and 150 for the spike datasets. As previously outlined (Section 5.2.1), windows that are too small or too large are likely to have a detrimental impact on the ability of the algorithm to resolve sites of genuine enrichment. The optimal values found from these tests were used to test the algorithm's performance with different FDRE values, centred around the expected optimum of 0.9. As previously outlined, values that are too small or too large will create unsuitable cutoff values, limiting the ability of the algorithm to correctly extract values representing genuine enrichment. The optimal values found from these two runs were finally used to test the effect of a range of scale values on the algorithm's performance. If the properties of the estimated background sub-population vary from those of the assumed normal distribution, genuine enriched regions may be thought to represent background regions, or genuine background regions may be thought to represent enriched regions. The "scale" argument allows the properties of the data to be modified, by scaling the whole dataset by the given factor, to improve the ability of the algorithm to correctly identify enriched and background regions.

The function was applied to the simulated data with the following settings:

object `arrayData` objects containing normalised simulated dataset in the combinations outlined in Section 5.3.1.1.
annotation The `genomeAnnotation` object for the yeast genome.
windowSize A range of values, shown in Table 5.3
FDRE A range of values, shown in Table 5.5.
findPeaks `TRUE`, in order to identify probes at the tops of peaks.

The function was applied to the spike data with the following settings:

object `arrayData` objects containing normalised spike dataset in the combinations shown in Table 5.2.

annotation The `genomeAnnotation` object for the human genome.
windowSize A range of values, shown in Table 5.4.
FDRE A range of values, shown in Table 5.5.
findPeaks `FALSE`, in order to identify regions of enrichment.

Peak detection was applied to the simulated data, returning a list of probes deemed to be at the tops of peaks. These were compared to the 2,000 randomly selected probes at the tops of the genuine, simulated peak sites. Only enrichment detection was applied to the spiked datasets, without peak detection, because of the nature of the data. A protein immunoprecipitation procedure creates a triangular region of enrichment, because more DNA is present from the site of the protein than the surrounding regions (Figure 5.1), the top of which the peak detection aims to identify. The spike probes are present at a consistent level across their length, creating rectangular regions of enrichment (these can be seen in the "Enriched regions (combined).pdf" file in the electronic appendix; see Page 367). The probes in this region can be identified by the enrichment detection procedure, but as there is no actual peak the peak detection procedure will not provide useful additional information. Probes detected as enriched were compared to probes in the regions containing spikes.

Receiver operating characteristic (ROC) curves are a method of displaying the performance of signal detection methods, used here to display the performance of the peak detection method. The curves show the relationship between sensitivity — the ability to correctly detect genuine results — and specificity — the ability to correctly ignore false results. In other words, sensitivity measures the proportion of true positives identified as positive, that is, detected, and specificity measures the proportion of true negatives identified as negative, that is, not detected. Therefore a detection method with high sensitivity is able to correctly identify a high proportion of true positives, and a method with high specificity is able to correctly ignore a high proportion of true negatives. Both of these characteristics are required in a reliable and informative detection method. A ROC curve shows how the sensitivity and specificity change as a given parameter of the method is altered, in this case the window size, FDRE and scale values. Sensitivity is

shown on the y axis and 1-specificity on the x axis, so that the top left hand corner of the plot represents the 'perfect' scenario: maximum sensitivity and specificity. The format of a ROC plot, the relationships between sensitivity and specificity and example curves are shown in Figure 5.13.

The sensitivity and specificity values for a ROC curve are calculated from the numbers of true positives (detected known positives; TP), false positives (detected known negatives; FP), true negatives (undetected known negatives; TN) and false negatives (undetected known positives; FN), using the following Equations 5.1 and 5.2.

$$\text{sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.1}$$

$$\text{specificity} = \frac{\text{TN}}{\text{FP} + \text{TN}} \tag{5.2}$$

Johnson et al. (2008) use a variation on this format to create ROC-like plots from the results of their peak detections. Because most probes on these microarrays are are true negatives (over 99%), large numbers of detected false positives can be masked. This can create apparently good ROC curves even with large numbers of incorrectly called positive results. Therefore they calculate results based on the number of spikes; a much smaller value than the number of true negatives. This methodology was applied to the simulated data with a variation applied to the spike data, as this did not have peak detection process applied. For this variation, detected probes were split into consecutive regions and each of these compared with the genuine enriched regions. Those overlapping were deemed to be true positives. This principle is demonstrated in Figure 5.14. This format also reduces misleading results that can be produced when probes around, but not including, enriched regions are called as positive, or some, but not all, probes in enriched regions are called as positive. This can cause probes to be called as FP when they are immediately adjacent to enriched regions, and probes to be called as FN when they occur in enriched regions where a number of probes have been correctly called as TP. These modified sensitivity and specificity values were calculated using the following Equations 5.3 and 5.4.

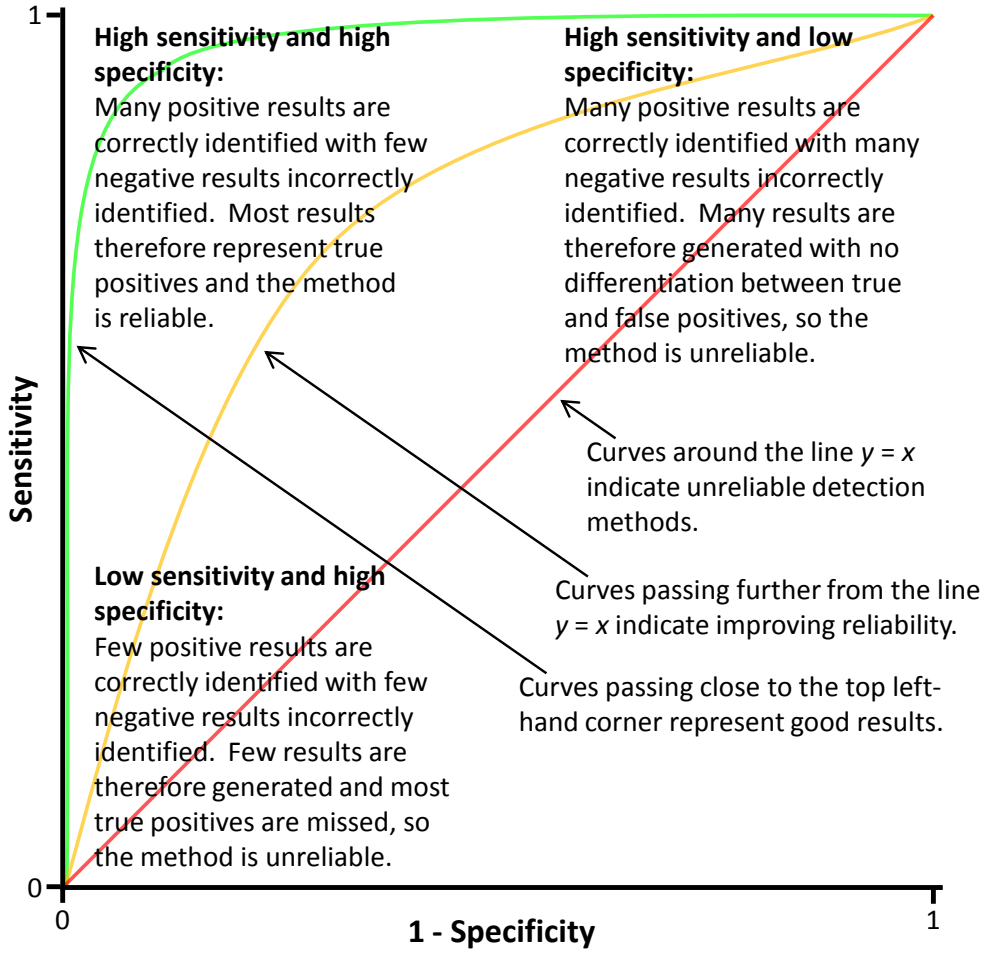


Figure 5.13: ROC plot properties: Examples of poor (red), intermediate (orange) and good (green) curves are shown.

$$\text{Modified sensitivity} = \frac{\text{Number of correct sites found}}{\text{Total number of sites}} \tag{5.3}$$

$$\text{Modified specificity} = \frac{\text{Number of incorrect sites found}}{\text{Total number of sites}} \tag{5.4}$$

5.3.2.1 Optimising the window size selection

The window size used by the algorithm can potentially have a large effect on the results generated. Too large or too small a window can reduce the sensitivity or specificity of the algorithm, reducing the reliability of the results (Figure 5.2). Optimal results should be generated when the window size is large enough to include enough probes to mask the effects of single, anomalous values and small enough to be entirely filled by regions of genuine enrichment. The average chromatin shear size was therefore expected to be the best value to use, as it meets these two criteria and is easily determined for each assay in the laboratory. The simulated datasets were created with a simulated chromatin fragment size of 600. A range of window sizes, based on this expected optimum, were tested (shown in Table 5.4). No information about the lengths of fragments produced by the sonication procedure for the spiked datasets is provided in the publication or supplementary information from Johnson et al. (2008). The average spike in length is around 500, suggesting the optimal window size to be 250. A range of window sizes, centred on this expected optimum were tested (shown in Table 5.3). All of these were carried out with the expected optimal FDRE value of 0.9.

The results of these tests are shown in Figure 5.15, with the first column showing ROC curves and the second ROC-like curves. The first row shows results from the simulated datasets. Green crosses indicate the results from the expected optimal value of 600. Single datasets are shown with grey lines, triplicates with black and all five with blue. The curves for the combined datasets (black and blue) are an unexpected shape, revealing an unusual result. They show that while the expected optimal window size produces the best sensitivity values, the specificity can be improved by reducing the window size to the smallest value tested (1). This is due to the fact that

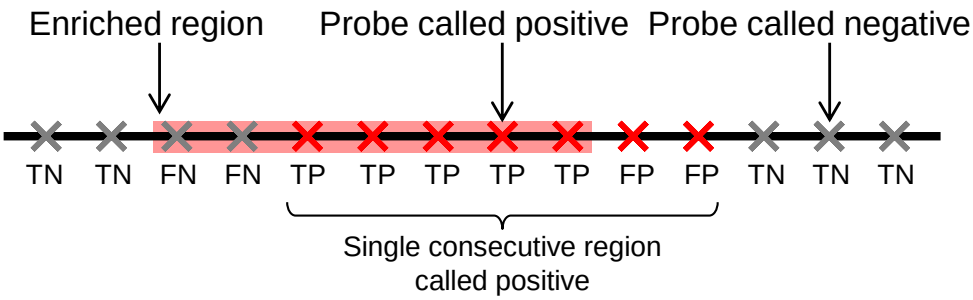


Figure 5.14: Labelling of consecutive enriched regions: Probes are indicated by crosses, with those called positive shown in red. The real enriched region is indicated with a pink box. Statuses, under each probe, show how individual probes are labelled as for use by Equations 5.1 and 5.2. Converting results to consecutive regions and comparing these positions to regions of real enrichment means this single region of consecutive probes is called as positive for use by Equations 5.3 and 5.4.

Number	Window Size
1	1
2	100
3	200
4	300
5	400
6	500
7	**600**
8	700
9	800
10	900
11	1,000

Table 5.3: Window sizes for testing simulated data: Window sizes used by the enrichment detection procedure with the simulated datasets to generate results used to create ROC and ROC-like curves. The expected optimal value is shown in bold.

Number	Window Size
1	1
2	50
3	100
4	150
5	200
6	**250**
7	300
8	350
9	400
10	450
11	500

Table 5.4: Window sizes for testing spike data: Window sizes used by the enrichment detection procedure on the spike datasets to generate results used to create ROC and ROC-like curves. The expected optimal value is shown in bold.

combined datasets are being analysed, which on its own is enough to remove the effects of any spurious results occurring in single datasets, without the need for the added sensitivity gained by combining probes in larger windows. Using smaller window sizes does not improve the sensitivity, meaning all the peaks that can be detected are detected with the expected optimal window size, but it does improve the specificity, meaning fewer incorrect peaks are detected. This means that the lower cutoff values — produced when analysing multiple probes in a window — are detrimental in this situation, allowing more regions to be incorrectly called as enriched. The curves for the single datasets (grey) are a more usual shape, with the window size of 600 producing the best combination of sensitivity and specificity. The curves show that values around this also produce good combinations of sensitivity and specificity and so the value does not have to be determined with a high degree of accuracy in the laboratory. With single datasets, the extra sensitivity gained by analysing combinations of probes in windows is still beneficial. Therefore in all further analyses of these datasets a window size of 600 was used for single datasets and 1 for combined datasets.

The results of the spiked datasets are shown in the next two rows, with combined datasets in B and averaged datasets in C. Single datasets are shown with grey lines, triplicates with black, total datasets with red and combined (Myers and McCuire) blue. These also produce some unusually shaped curves, but the green crosses, highlighting the expected optimal value of 250, show that this generally produces the best combination of sensitivity and specificity across all datasets. Unlike the simulated datasets, a window size of 1 does not improve the results from the combined datasets. This is likely due to the different format of this data, which lacks true peak shapes, and the resulting different analysis method, which analyses single probes called as enriched, rather than probes at the tops of peaks. All further analyses of these datasets use a window size of 250. Comparing the combined and averaged dataset curves shows mixed results. Generally, better sensitivity values are produced with averaged data, but better specificity values are produced with combined data.

These results show that in most cases the best balance of sensitivity

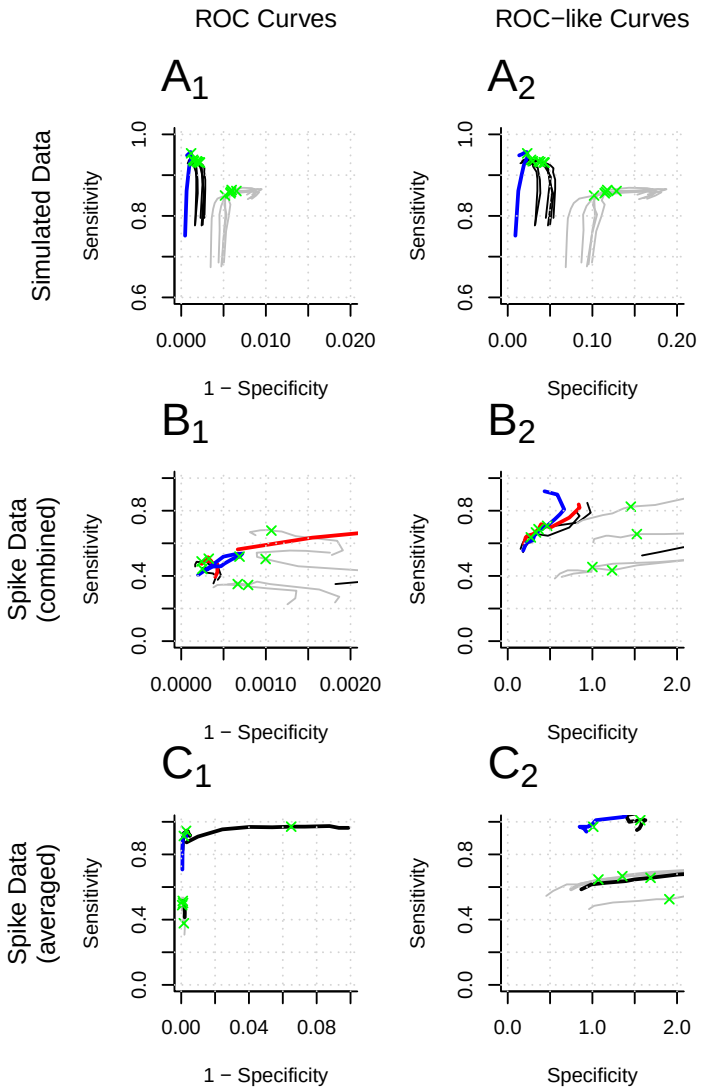


Figure 5.15: ROC curves from window size variations: Results produced by varying the window size used by the enrichment detection algorithm. The left hand column shows ROC curves created using Equations 5.1 and 5.2 and the right hand column shown ROC-like curves created with Equations 5.3 and 5.4. The first row shows results from the simulated datasets, with green crosses showing the expected optimal window size of 600. The second and third rows show results from the combined and averaged spiked datasets respectively, with green crosses showing the expected optimal value of 250. Note the differences in axis scales.

and specificity are at the expected window sizes, with the exception of the combined simulated datasets. There is scope to vary the value used without adversely affecting the results, depending on the nature of the datasets. Those with little background variation will produce good results even with a smaller window size, whereas those with increased levels of background variation may not. When performing peak detection on the results of the enrichment, only the probe at the centre of the peak need be found for the peak to be called. Therefore, changing the window size, resulting in a change in the probes called as enriched, will only affect the final result if this changes whether or not the probe at the centre of the peak is called as enriched. The average chromatin shear size can readily be determined in the laboratory, and these results show that this value is appropriate to use in the algorithm. They also show that this value does not need to be determined with a high level of accuracy, as small deviations from the optimal value do not have a large effect on the final results.

5.3.2.2 Optimising the FDRE value selection

The FDRE value used by the algorithm determines the cutoffs for each window and so can influence the performance of detection. Too small a value will give high cutoff values, enabling the detection only of enriched probes with large values and reducing the sensitivity. Too large a value will give low cutoff values, enabling the detection of probes with small values which may not be due to genuine enrichment, reducing the specificity. Optimal results should be generated with an FDRE value of around 0.9. The algorithm was applied, with the window sizes determined in the previous section, using a range of FDRE values based on this expected optimum value, shown in Table 5.5.

The results of these tests are shown in Figure 5.16, with the first column showing ROC curves and the second ROC-like curves. Green crosses indicate the results from the expected optimal value of 0.9. The first row (A) shows results from the simulated datasets. These plots confirm that sensitivity and specificity generally increase as the enrichment detection is performed on

FDRE	Value
1	0.1
2	0.2
3	0.3
4	0.4
5	0.5
6	0.6
7	0.7
8	0.8
9	**0.9**
10	1.0
11	1.1
12	1.2

Table 5.5: FDRE values for testing simulated data: FDRE values used by the enrichment detection procedures on the simulated and spike datasets to generate results used to create ROC and ROC-like curves. The expected optimal value is shown in bold.

more datasets, with the grey lines of single datasets being lower and further to the right than the coloured lines of combined datasets. The results of the spiked datasets are shown in the next two rows, with combined datasets in B and averaged datasets in C. Single datasets are shown with grey lines, triplicates with black, total datasets with red and combined (Myers and McCuire) blue. There is a general trend for improved sensitivity and specificity with the analysis of more datasets, but this is not as clear cut as the simulated data. For example, the first plot (B_1) shows a grey line, representing a single dataset, to have a higher sensitivity than the combined datasets at lower specificity levels. The curves show that generally the best balance of sensitivity and specificity for each curve is at the expected, highlighted points. Some improvement is possible with some curves, but good results are achieved with all. This shows that under the conditions of normal ChIP-chip assays, where genuine results are not known, this value would be able to provide reliable results for a range of datasets, confirming the expectation that this value is the best to use in the algorithm. Using lower values generates fewer correct results, with little change in the number of incorrect results, so does not have any additional benefit. Using higher values generates more incorrect results with little change in the number of correct results, reducing the usefulness of those results. Therefore this value of 0.9 was used for all further analyses.

5.3.2.3 Optimising the scale value selection

The results in the previous sections have been generated from datasets assumed to fully meet the expectations for enrichment detection (see Section 4.2.1). Although the expected optimal values are generally at the optimal points on the curves, some of the sensitivity and specificity values are poor, showing that there is still scope for improvement in the detection method. The performance of the algorithm was therefore assessed by varying the scale values, using the optimal window size and FDRE values.

The results of the peak detection using these values on the simulated datasets are shown in Figure 5.17. Each row shows the results of datasets with the small, medium, large and combined peak heights. The results show

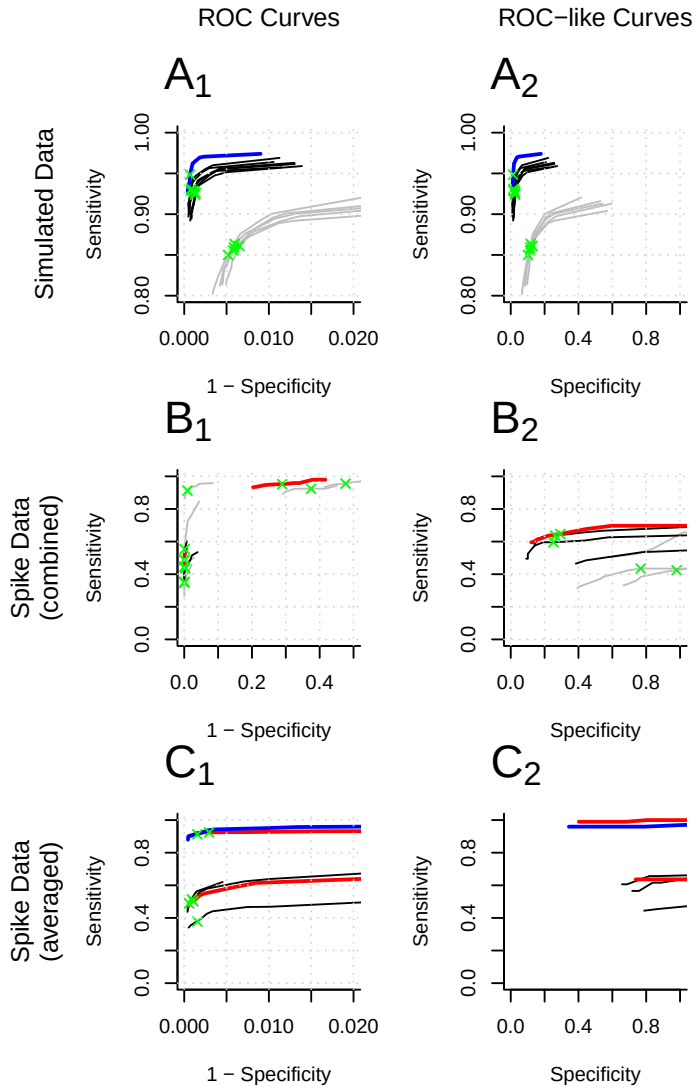


Figure 5.16: ROC curves from FDRE variations: Results produced by varying the FDRE value used by the enrichment detection algorithm. The left hand column shows ROC curves created using Equations 5.1 and 5.2 and the right hand column shown ROC-like curves created with Equations 5.3 and 5.4. The first row shows results from the simulated datasets, the second and third show results from the combined and averaged spiked datasets respectively. Green crosses show the expected optimal FDRE value of 0.9. Note the differences in axis scales.

the method generally has a good sensitivity and specificity.

The first row (A) shows good results to be generated from the datasets with small peaks. The optimum result corresponds to around 80% of the total peaks being found, with 50–100 incorrect peaks. This result shows that the detection method produces reliable results even with low levels of enrichment, suggesting it is suitable for use on genuine ChIP-chip data containing only low levels of enrichment. The second (B) and third (C) rows, showing the results from the medium and large peaks, give near perfect results, especially with the combined datasets, with the correct peaks found approaching 100% with the number of incorrect peaks approaching zero. However, as these lack any small peaks, they may not be representative of genuine ChIP-chip data. The fourth row (D), showing results of datasets with a range of combined peak heights, is likley to most accurately represent genuine ChIP-chip data. The optimal results here corresponds to around 90–95% of the total peaks being found, with around 50 incorrect peaks.

The sensitivity and specificity values are markedly improved by analysing combined datasets compared to analysing datasets individually. The best results come from the combination of all five datasets, shown with red lines. Many biological experiments are carried out in triplicate, represented with black lines, which still give a large improvement in sensitivity and specificity over the individual datasets, shown with grey lines. The analyses of individual datasets are reasonably robust, with the ROC-like curves of combined peak height datasets (Figure 5.17D$_2$) showing around 85% of the total genuine peaks are found, along with around 200 incorrect peaks (a specificity value of 0.1). This shows the method is reliable even when only single datasets have been created, although it is clearly preferable to have multiple datasets to analyse together.

Green crosses mark the results from the datasets with no scaling applied. These generally occur at the point on the curve closest to the top left hand corner, indicating the optimal result. Many are on the 'bend' of the curves, that is, the point at which the increase in sensitivity slows down while the decrease in specificity speeds up. Results before this point tend to show large increases in sensitivity with small decreases in specificity, meaning the quality

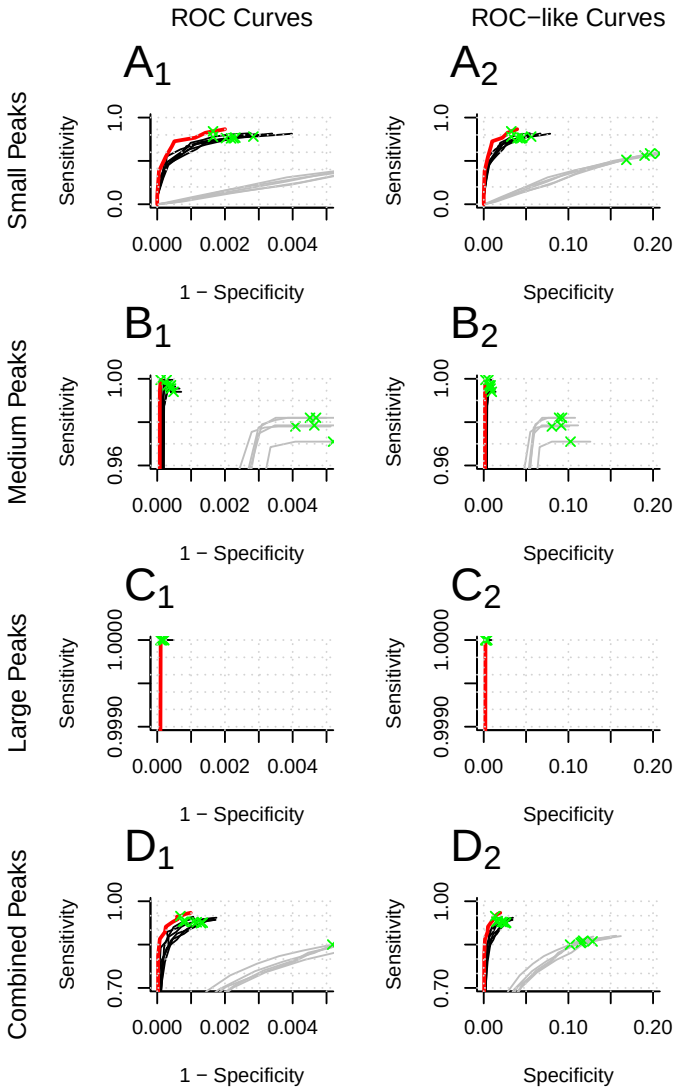


Figure 5.17: ROC curves from simulated data: Results produced by varying the scale value used by the enrichment detection algorithm on the simulated datasets. The left hand column shows ROC curves created using Equations 5.1 and 5.2 and the right hand column shown ROC-like curves created with Equations 5.3 and 5.4. The first row show results from small peak heights, the second medium, the third large and the fourth all combined. Grey lines show results from individual datasets, black groups of three and blue all five. Green crosses show the expected optimal scale value of 1, that is, no scaling. Note the differences in axis scales.

of the results is increasing. Results beyond this point tend not to show large increases in sensitivity but do have large decreases in specificity, meaning the quality of the results is decreasing. The results of the medium and large peak datasets (B and C) show that some reduction in scale can improve results, but this is not the case with the small and combined datasets (A and D), most likely to represent real ChIP-chip data. As these data are known to have a normally distributed background, no scaling would be expected to be required and so these results confirm the assumptions of the algorithm.

The results of the peak detection using the scale values on the combined and averaged spiked datasets are shown in Figures 5.19 and 5.18 respectively. For each the first row shows results from the unamplified datasets, showing the Myers (black), McCuire (red) and combined (blue) data, and the second the amplified (McCuire only) datasets. The left hand column shows ROC curves and the right hand column shown ROC-like curves. These curves show that the performance of the algorithm is poor with unscaled datasets, the results of which are highlighted with green crosses (some crosses are beyond the scale of the plots and are therefore not visible). This is as a result of the properties of the background distributions of the datasets not meeting the assumptions of the algorithm, that is, they do not form normal distributions. This means that the algorithm interprets some probes as enriched when they are in fact part of the background distribution, resulting in poor specificity values. Scaling the data down means these regions become correctly interpreted as background, while the genuine regions of enrichment are still detected, thus improving the sensitivity and specificity.

Curves created with results from averaged data (Figure 5.18) show a better performance of the algorithm than the combined individual datasets (Figure 5.19), evidenced by the fact that they pass closer to the top left hand corner of the plot regions. The reasons for this are outlined in Section 5.2.1.1, and stem from the fact that the datasets are not very consistent between repeats. The point closest to the top left hand corner of the plot regions, showing the best balance between sensitivity and specificity, was created by a scale factor of 0.6.

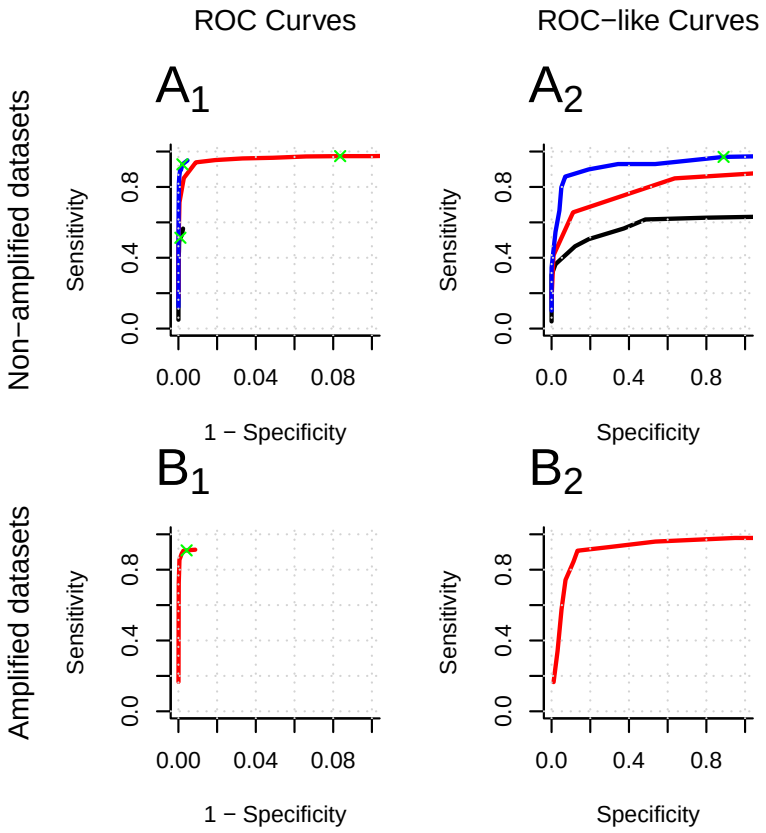


Figure 5.18: ROC curves from averaged spiked data: Results produced by varying the scale value used by the enrichment detection algorithm on the averaged spiked datasets. The left hand column shows ROC curves created using Equations 5.1 and 5.2 and the right hand column shown ROC-like curves created with Equations 5.3 and 5.4. The first row show results from enriched datasets (Myers (black), McCuire (red) and combined (blue)) and the second non-enriched datasets (McCuire only). Green crosses show the expected optimal scale value of 1, that is, no scaling.

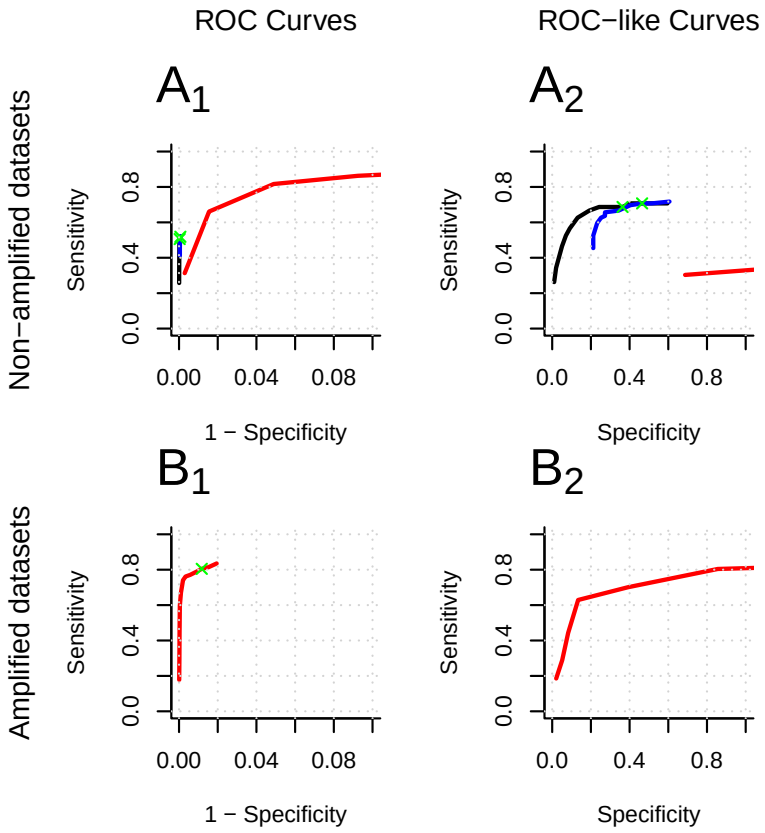


Figure 5.19: ROC curves from combined spiked data: Results produced by varying the scale value used by the enrichment detection algorithm on the combined spiked datasets. The left hand column shows ROC curves created using Equations 5.1 and 5.2 and the right hand column shown ROC-like curves created with Equations 5.3 and 5.4. The first row show results from enriched datasets (Myers (black), McCuire (red) and combined (blue)) and the second non-enriched datasets (McCuire only). Green crosses show the expected optimal scale value of 1, that is, no scaling.

5.3.2.4 Summary

Plots were created of every region detected as enriched, along with the regions of genuine enrichment ("Enriched regions (averaged).pdf" and "Enriched regions (combined).pdf" files in the electronic appendix; see Page 367). Of the non-enriched regions that were detected by the algorithm as enriched, all appear to be reasonably large peaks containing several probes, so ordinarily there would be no way of distinguishing between them and genuine peaks from the data alone. There could be several explanations for the formation of these peaks. They occur in all datasets and so their formation is unlikely to be due to chance events in a single experimental run. They may be genuine peaks, that is, caused by genuine enrichment, either because of increased material over the regions in the commercial human genomic DNA preparation or the spike in probes used being able to hybridise to other regions of the genome. Alternatively, they may not be due to genuine increased enrichment but inconsistencies with the probes on the microarray causing high signals to be produced from small amounts of bound DNA. There is no way of distinguishing between these and genuine peaks, and so there is no way of improving further on the specificity of the algorithm without finding the reasons for these spurious peaks.

The plots show that the scale value can have a large effect on the results of the algorithm and so it is important to provide the correct value. Unlike the expected optimal values for the window size and FDRE parameters, there is no way to estimate what the scale value should be for a given dataset. The lowest value of the datasets, used in estimating the background population, may provide a method of estimating the optimal scale value. The lowest value of a standard normal distribution is around 4. The lowest value of the spiked datasets is -7.3. The ratio of these numbers is around 0.6, that is, the optimal scale value for the datasets. This may be coincidental, or may represent a way of estimating the scale factor to use in the enrichment detection. Without more spiked datasets with known enrichments it is not possible to say with certainty that this provides a method of determining this value.

These results show that the expected optimum FDRE value of 0.9 has proved to produce the best results, meaning there is generally no need for users to modify this. The optimum window size was at the expected level for single datasets, but smaller values were shown to produce better results with combined datasets when detecting peaks. Cutoff values may be applied to the sensitivity and specificity values of these results, above which all values may be considered optimal. However, the way the algorithm presented here works means the theoretical optimal values for the FDRE and window size are known before hand, and these have been shown in this analysis to provide optimal results, and so reporting a range of values in this way provides no additional useful information. The expected optimal values of 0.9 for the FDRE and the average chromatin shear size (estimated from a gel) for the window size should be used under normal circumstances. The scale value can have a large effect on the performance of the algorithm when the data do not fully meet the expectations of the algorithm. This value is dataset dependent, specifically relating to the distribution of each dataset, which influences the estimation of the background region by the algorithm. Therefore the optimal value will vary for real datasets which do not meet the expectations of the algorithm, in a manner that cannot currently be predicted. The setting of the optimum value for this argument therefore needs more investigation.

The fact that the enrichment detection procedure classifies probes as being from the enriched or background sub-populations means that these can be compared to the overall and estimated background distributions. How well the density of the detected background sub-population matches the density of the estimated background sub-population may provide a way of estimating the accuracy of the results of the procedure with real ChIP-chip datasets, for which the correct results are not known *a priori*. If the densities closely match then it would be reasonable to assume that the detected enriched sub-population is accurate and therefore the procedure has performed well. This may provide a method of 'fine tuning' the algorithm to determine the best parameters to use for each dataset being analysed. An example plot is shown in Figure 5.20, created from the simulated data, showing how the densities of the results of the procedure compare to the estimated back-

ground region. The detected background region (red line) closely follows the estimated background region (dashed black line). The detected enriched sub-population (green line) follows the line expected in the enriched region of the overall dataset density (dashed grey line). There is a discrepancy between the estimated and detected background sub-populations towards the high end of the densities, which may be able to be corrected with further adjustments of the parameters.

5.3.3 Comparison with other methods

As previously stated, the availability of reliable enrichment detection software for ChIP-chip data is limited. Of the most recent publications that can be applied to Agilent data, the DECODE and JAMIE software packages are no longer available, Wavelets is not provided as software and SPLITTER is available but not functional.

SPLITTER is available as a web server (`zlab.bu.edu/yf/anchor/web/splitter.cgi?step=0`), where data to be analysed is uploaded in the form of tab delimited text files. To compare this algorithm with the one presented here, the simulated datasets were uploaded for analysis. Default settings and settings applied by Johnson et al. (2008) were used, but the algorithm was not able to return any results. A range of additional settings were also tried, adjusting all available parameters, but no results could be generated. The Abf1 binding dataset was then uploaded to test the algorithm with genuine ChIP-chip data, as opposed to the artificial nature of the previous two datasets, but it was not possible to generate any results from this data either. All data were uploaded correctly, as verified by histograms shown in the SPLITTER output. As this algorithm is available only as a web server, it is difficult to determine the reason for the lack of result generation. The performance of this algorithm could not therefore be accurately compared.

In addition, the MA2C package was downloaded, which is written to process only Nimblegen data files. Although the help forum for this program includes references to converting Agilent data to the required format, the programs required to achieve this are not publicly available.

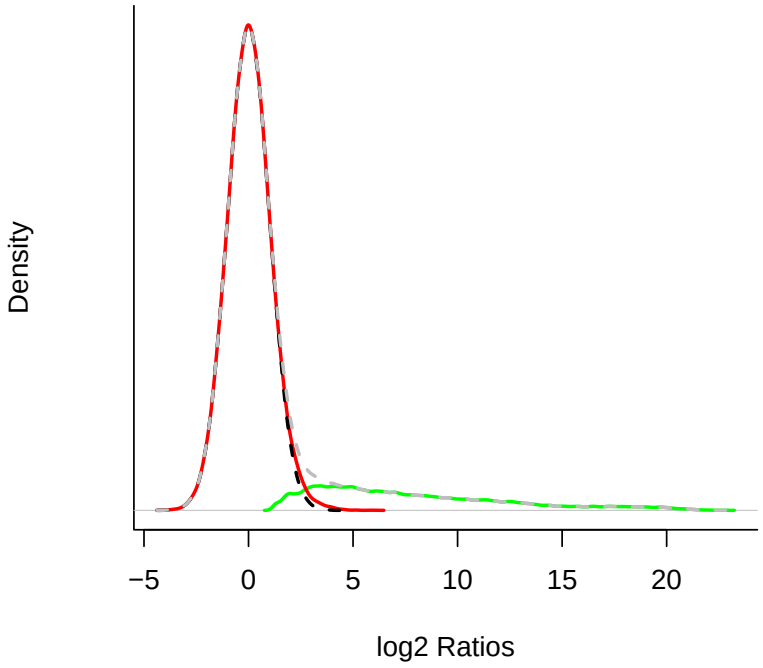


Figure 5.20: Example analysis of peak detection results: The green and red lines show the density of the detected enriched and background sub-populations respectively. The dashed grey line shows the overall data density and the dashed black line the estimated background sub-population.

Therefore the only way of assessing the performance of this algorithm relative to those previously published is by the visual comparison of published ROC and ROC-like curves. ROC-like curves created by Johnson et al. (2008) for the spiked datasets are shown in Figure 5.21. Here the x-axis value of 0.05 has been used to determine sensitivity values, the Agilent values of which can be seen to be around 0.5 for the amplified and unamplified datasets. Comparing these to the equivalent values in Figure 5.18 shows higher values created from the algorithm presented here, with the combined dataset (blue curve) reaching a sensitivity value of around 0.8 at this point.

The limited availability and applicability of many of these peak detection programs highlights the difficulties that users can face when attempting to analyse their own data. This is why the algorithm created here has been written in such a way that it can be used on any type of data, provided the values can be written to a tab delimited text file. The accompanying help files provide basic instructions on how to run the process from the very start, as well as more detailed information for more advanced users. It is hoped that, when published, this will allow a range of people to easily analyse their data in a way that suits their needs.

5.4 Discussion

Like the normalisation method, the peak detection method presented here relies on the background sub-population approximately following a normal distribution. Some other methods of peak detection also rely on this assumption (Buck et al., 2005, for example), which is generally held for ChIP-chip datasets. Unlike many other methods, it has the advantage of being able to return estimated binding sites (for example, for protein binding datasets) or regions of enrichment (for example, for histone modification datasets). This distinction allows relevant data to be analysed for any given dataset. It also has the advantage of utilising replicate datasets and overlapping probes to increase the power of detection. Therefore with several repeats of a dataset being analysed together, smaller binding peaks can detected than by analysing all the datasets separately and combining the results. Similarly, smaller peaks

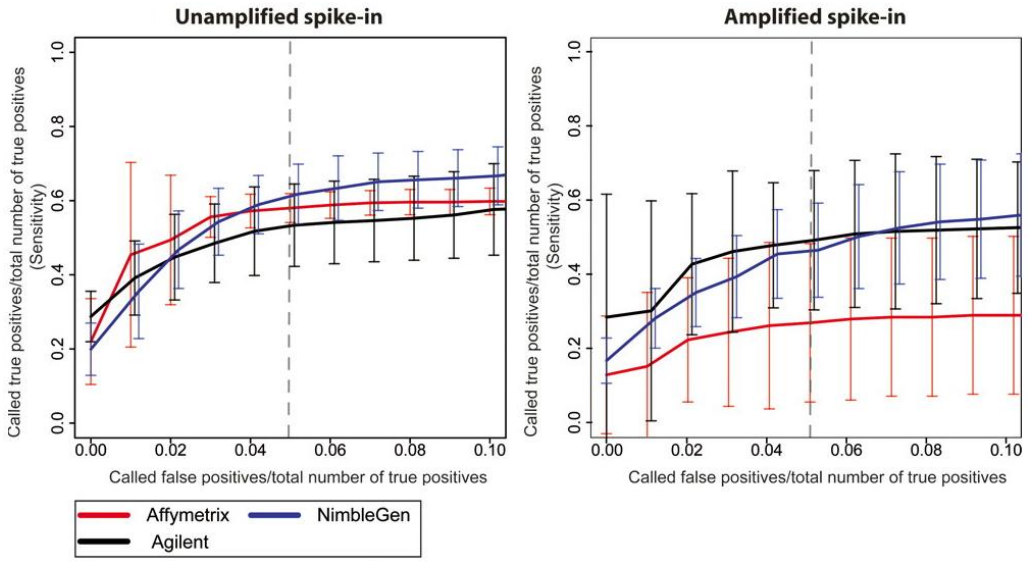


Figure 5.21: ROC-like curves created by Johnson et al. (2008): The combined results of enrichment detection procedures applied to the three microarray formats are shown. Adapted from Johnson et al. (2008).

can be detected in regions with higher probe coverage. Therefore on high resolution microarrays with multiple repeats it should be possible to detect the smallest of binding sites.

Users can define several variables, but the defaults for the window size and FDRE value have been shown here to work well, and so little time need be spent adjusting these to get the optimal results. The scale value has been shown to have a large effect on some data and so more time may need to be spent optimising this. More work needs to be done to determine if there is any link between a dataset and the optimal scale value, which would provide an easy way of setting this parameter. The algorithm is incorporated into the package of functions detailed in Chapter 3 and takes only a matter of seconds to run, meaning users can easily load data and obtain results from this algorithm in very little time.

The results of enrichment detection on publicly available spiked datasets (Johnson et al., 2008) suggest that the algorithm is able to outperform these published methods. Sensitivity and specificity values were calculated from a range of window size, FDRE and scale values, the results of which are shown in ROC and ROC-like curves. Visual comparison of these to published curves show that this algorithm produces higher sensitivity values at a given specificity value. The existing algorithms are not available for testing and so this visual comparison is the only way of gauging their performances relative to the algorithm presented here. The algorithms themselves would be required to perform an objective comparison of results and this could be carried out as a follow up study should they become publicly available in the future.

In addition to this, detection methods that can be applied to any format of ChIP-chip data are not readily available and so this algorithm is well placed to fill this gap.

Comparisons of peaks or enriched areas from different arrays can be made by statistical methods, such as T-tests, or graphical methods, using the tools provided in Chapter 3. Changes in the shapes of profiles may not be detected statistically, while being clear to the eye. These may still represent interesting biological results and so statistics should not be relied upon as the only definitive result.

This new enrichment detection method was created to overcome the limitations of some current algorithms, in the type of data that they can analyse, the type of detection that they can perform, their ability to analyse multiple datasets and their current availability for use (outlined in Table 5.1). The method presented here can analyse data from any platform (by converting data to the defined tab-delimited file format where necessary), can be set to find regions of enrichment or peaks (depending on the type of data being analysed) and is able to utilise multiple replicate datasets to increase the power of detection over analysing datasets individually.

Chapter 6

Development of a method to predict sequence specific damage events

6.1 Introduction

Many DNA damaging agents induce damage at specific nucleotide sequences. There are many examples in the literature of particular chemicals being shown to induce damage at particular sequences, such as reactive oxygen species (Oikawa, 2005), reducing sugars (Morita et al., 1985), lipid peroxidation products (Ueda et al., 1985) and ozone (Ito et al., 2005). In our laboratory we investigate primarily UV damage, which occurs with known frequency at the four possible dipyrimidine sequences. UV radiation, specifically at or near to 254 nm, is close to the maximum absorption wavelength of DNA. This can generate many lesions, mainly at adjacent pyrimidine sites. Two adjacent pyrimidines in the same polynucleotide chain can absorb UV energy to form a four-membered ring structure, a CPD, resulting from saturation of the C=C double bonds (Figure 1.5). CPDs form solely at adjacent pyrimidine sites, and as such their position is determined by DNA sequence. The quantitative ratio of CPD formation after UV irradiation at TT, TC, CT and CC sites is 68:16:13:3 as determined by measuring these lesions in plasmid DNA (Mitchell et al., 1992) and DNA from human cells (Tornaletti et al., 1993).

When developing assays to measure these damages throughout an entire genome it is useful to be able to generate a predicted profile of DNA damage expected immediately after a treatment. This offers an important quality control function in confirming that the assays are performing as expected. This gives greater confidence in analysing datasets generated at later time points, from which repair rates can be calculated.

It also has the potential to enable the identification of 'hot' and 'cold' spots of damage, by identifying regions of consistently lower or higher damage than the prediction suggests. The ability to detect and identify these outliers from the predicted pattern is especially useful for certain types of analysis (see Section 6.6.1).

6.2 Motivation

The technology developed in our laboratory to measure DNA damage genome wide (Teng et al., 2011) required a way of testing whether or not the signals from the microarrays represent genuine UV induced damage or general background noise (see Section 1.1.3). Damage occurs across the whole genome and so the data cannot be split into background and enriched subpopulations (shown in Figure 6.1 and described previously in Section 4.2.1): every probe represents some amount of damage and so the data consists of a single enriched population. Without this comparison between background and enriched subpopulations it is not possible to tell if the data show a signal resulting from enrichment following immunoprecipitation of damage or background noise as a result of non-specific immunoprecipitation and/or hybridisation.

Comparing the microarray data to a predicted profile allows their accuracy level to be estimated.

6.3 Methodology

As previously described (see Figure 5.1), a peak is centred on a binding site with a base width approximately twice the average chromatin shear size.

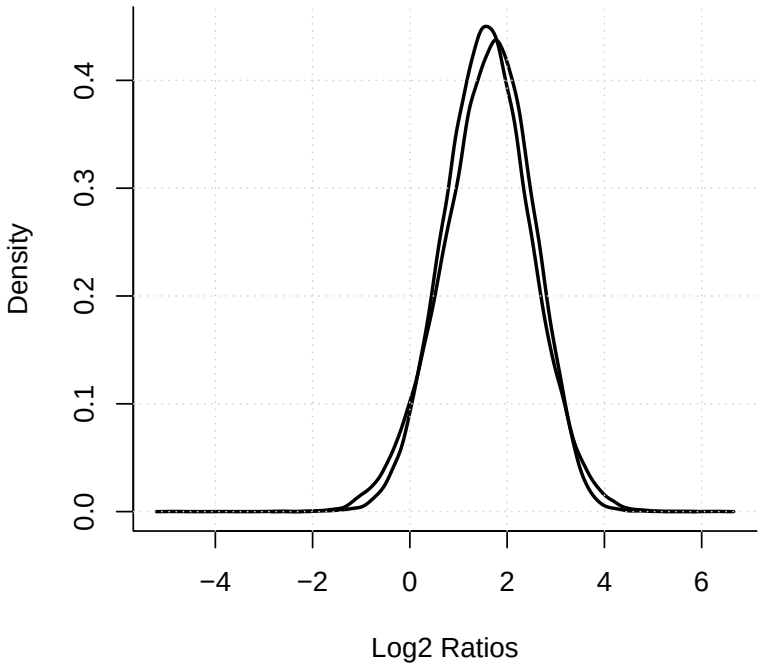


Figure 6.1: CPD dataset density plots: CPDs occur throughout the genome and so the density plots of the two replicates show a distribution of enrichment values without a distribution of background values.

The height of the peak is determined by the fluorescence signal, which is determined by the amount of material bound to the probe. Knowing these properties allows a peak shape to be predicted, given the average chromatin shear size. The fluorescence signal, or amount of material bound, does not need to be known in order to predict a peak shape, as the height can be treated as being in arbitrary units. The value of these units does not affect the peak shape as all points forming the peak will always have the same relationship to each other, regardless of their actual values. In the formula presented here the predicted values range from 0 to 1.

Theoretically, the value at any point of the peak is directly linked to its distance from the binding site. The point at the centre of the peak will have the largest predicted value, decreasing linearly as the distance away from the centre extends to the window size, at which point the predicted binding value becomes zero. In practice, the peak shape is more likely to form a Gaussian distribution due to the varying lengths of the sheared chromatin fragments. These will cause variations in the maximum extent of binding at a given site, influencing the width of the base of the peak. However, this effect is likely to be relatively small, with triangular peaks capturing much of the information of a binding site. Therefore for computational efficiency this peak shape is used in this prediction method. To calculate a predicted binding value at any given point one needs to know the location of the actual binding site, the location of the point at which the prediction is being made (or its distance from the actual binding site, as whether it is up- or downstream of this does not affect the binding value) and the window size. The predicted binding value can then be calculated as the difference between the fragment length and distance from the actual binding site divided by the fragment length. In this way if the site being predicted is at the actual binding site the distance between the two is zero. The calculation is therefore the window size minus 0 (equals the window size) divided by the window size, which equals 1. If the site being predicted is the distance of the window size away from the actual binding site, the distance between the two is the window size. The calculation is therefore the window size minus the window size (equals 0) divided by the window size, which equals 0. At all other points the predicted value will lie

between these two extremes. These predicted values at each probe can be calculated with the following Equation 6.1, with a single binding site. Here w is the window size and $|d|$ is the positive distance between the binding site and the point being calculated.

$$\frac{w - |d|}{w} \tag{6.1}$$

This same formula can be used to calculate the binding value from a damage event, where the term binding site can be replaced by damage site. In reality there are likely to be multiple damage sites surrounding each probe on the array, all contributing to their binding values. The equation can therefore be extended to take all these sites into account, by adding together their various values, shown in the following Equation 6.2. Here s represents each of n damage sites, the calculations for all of which are added together.

$$\sum_{s=0}^{n} \frac{w - |d|_s}{w} \tag{6.2}$$

In the event that different damages occur at different frequencies, such as CPDs, the probability value of any site can be incorporated to reflect the likelihood of it contributing to the overall value. This reduces the maximum predictable value from 1 to the probability value of the particular damage. This is shown in the following Equation 6.3. Here p is the probability value of binding site s.

$$\sum_{s=0}^{n} p_s \frac{w - |d|_s}{w} \tag{6.3}$$

This equation is applied to the region the size of the window size up- and downstream of every point in the genome at which a predicted value is required. It has been published and used to produce a predicted CPD damage profile for yeast by calculating a value at each of the probes on the G4493A microarrays (Teng et al., 2011).

6.4 Algorithm

The equation described here was used as the basis of a script to allow predictions to be made throughout whole genomes. This was originally written in Perl, as referred to in Teng et al. (2011), and was subsequently rewritten in R. The reason for this transition was the ease of access to any genome sequence in R via the `BSgenome` (Pages, 2012a) package. The alternative Perl version required each genome to be manually downloaded in the correct format and loaded into Perl.

Before running the prediction script, the relevant genome must first be loaded into R. This is then passed to the `predictProfile` function along with coordinates of the points to be predicted, the window size, damage site sequences and damage site sequence probabilities. The sequence around each site is taken and the `Biostrings` (Pages et al., 2009) package used to determine the locations of all sequences being investigated. The distance of these sites to the site being scored is calculated and the predicted score calculated as Equation 6.3. This is repeated for each damage sequence and all the predicted scores for the site are added together.

6.5 Alternative algorithm

This published method is computationally intensive, having as it does to perform calculations at every dipyrimidine site in the genome. It became apparent during investigations of the Kernel density estimate for the normalisation procedure (Chapter 4) that this may also be used to perform damage predictions. The profile of any ChIP-chip assay can be considered to be a representation of the density of the number of immunoprecipitation events along the genome. The density function also has the advantage of being able to calculate densities from different underlying distributions, potentially enabling a closer match to that in ChIP-chip. Using the triangular Kernel is essentially identical to the published method, and so this can be used to make a comparison between the two approaches. Using the Gaussian Kernel may provide a method of increasing the accuracy of the method, by

more closely representing the underlying peak distributions, without any loss of computational efficiency. It was therefore investigated whether this computationally efficient algorithm could be used to replace the existing CPD prediction algorithm.

The major advantage of using the `density` function is that looping of each dipyrimidine site to calculate individual profiles which are then added together is not necessary. Instead, the function calculates a density from all dipyrimidine sites on a chromosome at once. The arguments of the function are modified to increase the resolution of the density to a level similar to the arrays, as the default values smooth the data too much to be of any use. Changing these parameters will influence the final prediction as they perform a similar role to the window size (w) in the equations in Section 6.3.

The locations of each combination of dipyrimidine sites are calculated with the `Biostrings` package as previously, which the `density` function uses to calculate the density. By default this is done at 512 points, which is too few across a chromosome consisting of tens or hundreds of thousands of nucleotides. Therefore one tenth of the number of nucleotides is specified, which is rounded up to the next power of two in the calculations. There is no real advantage to using a higher resolution, which is more computationally expensive for little gain in resolution. The values calculated by the `density` function are dependent on the number of points being analysed, which is different for each dipyrimidine sequence and chromosome. To remove this discrepancy the results are adjusted to make the maximum density value for all calculations 1, before being multiplied by the relevant probability value. The density values are calculated at the specified number of regular intervals across each chromosome, rather than at probe sites as with the equations in Section 6.3. The `approx` function is therefore utilised, which interpolates the density values at the sites of probes, based on the surrounding density values.

The results of this procedure provide near identical results to using the equations in Section 6.3, in a much shorter time. This method is therefore used in place of that described in Teng et al. (2011). The `predictProfile` function (Script 6.1) performs this procedure. In addition to using sequence

information, the function can create predictions based on coordinates. This can be useful to predict, for example, the binding profile from the immunoprecipitation of proteins, where the coordinates of expected binding sites are known. The function has the following arguments:

arrayData An `arrayData` object containing the probe positions from which to create the predicted profile (no default).

seqs A character vector specifying sequences to calculate predictions from (defaults dipyrimidine sites).

probs A numeric vector specifying probability values for each of the sequences provided in "seqs" (defaults for dipyrimidine sites).

genome A previously downloaded `BSgenome` genome sequence to analyse (no default).

coordinates A three column matrix containing the chromosome number, position and probability value for each site to predict (no default).

windowSize A numeric vector specifying the window size to use when calculating predictions from the "coordinates" matrix.

masked A logical vector indicating whether or not the "genome" object is masked.

In addition, the `width` argument should be specified as the window size, which is passed to the `density` function and used in its calculations.

Script 6.1: predictProfile: script to predict the profile of ChIP-chip data based on immunoprecipitation events at defined sites or sequences.

```
1  ## profilePlot function ##
2  ## arguments: object (an arrayData object), seqs (sequences at which to
       predict enrichment), probs (probability values to apply to sequences),
       genome (a BSgenome sequence), coordinates (coordinates of sites to
       predict binding), windowSize (windowsize to predict sites of binding),
       masked (whether the genome is masked)
3  predictProfile<-function(object,seqs=c("TT","AA","TC","GA","CT","AG","CC","
       GG"),probs=c(0.68,0.68,0.16,0.16,0.13,0.13,0.03,0.03),genome,coordinates
       ,windowSize,masked=T,...) {
4    require(Biostrings) #load package if required
5    require(BSgenome) #load package if required
6    probeValues<-rep(0,nrow(object)) #set all values to 0
7    if(!missing(seqs)) if(length(seqs) != length(probs)) stops("Different
         numbers of sequences and probabilities",call.=F) #check lengths
8    for (chr in unique(object$coordinates[,1])) { #loop through chromosomes
9      current<-object$coordinates[,1] == chr #get current object data
10     probes<-round(rowMeans(object$coordinates[current,2:3]),0)  #probe
           middles
```

```
11      if(missing(coordinates)) { #coordinates not provided
12        if(masked) currentSequence<-unmasked(genome[[chr]])[min(probes):max(
              probes)] else currentSequence<-genome[[chr]][min(probes):max(
              probes)] #get chromosome sequence
13        for (s in 1:length(seqs)) { #loop through sequences
14          pos<-as.matrix(matchPattern(seqs[s],currentSequence,fixed=F))[,1] #
                find sequences
15          x<-density(pos,n=(max(probes)-min(probes))/10,...) #kernel densities
16          x$y<-x$y-min(x$y) #scale to max=1
17          x$y<-x$y/max(x$y) #scale to max=1
18          x$x<-x$x+min(probes) #correct for genome position
19          probeValues[current]<-probeValues[current]+(approx(x$x,x$y,probes)$y
                * probs[s]) #interpolate probe values
20        }
21      }else{ #coordinates are provided
22        if(!is.matrix(coordinates)) stop("Coordinates must be matrix",call.=F)
              #check coordinates
23        if(ncol(coordinates) != 3) stop("Coordinates must contain three
              columns",call.=F) #check coordinates
24        pos<-coordinates[coordinates[,1] == chr,2] #get current coordinates
25        if(length(pos) > 1) { #if positions to predict
26          x<-density(pos,n=(max(object$coordinates[current,3])-min(object$
                coordinates[current,2]))/10,...) #calculate kernel density
27          x$y<-x$y-min(x$y) #scale to max=1
28          x$y<-x$y/max(x$y) #scale to max=1
29          x$x<-x$x+min(probes) #correct for genome position
30          probeValues[current]<-probeValues[current]+(approx(x$x,x$y,probes)$y
                ) #interpolate probe values
31          probs<-coordinates[coordinates[,1] == chr,3] #get probabilities
32          for (n in 1:length(pos)) { #loop through positions
33            probeValues[current][probes > (pos[n]-windowSize) & probes < (pos[
                  n]+windowSize)]<-probeValues[current][probes > (pos[n]-
                  windowSize) & probes < (pos[n]+windowSize)]*probs[n] #scale
34          }
35        }
36      }
37    }
38  probeValues[is.na(probeValues)]<-0 #set NAs to 0
39  prediction<-object[,1] #create arrayData object
40  prediction$ratios<-matrix(ncol=1,probeValues) #put in predicted values
41  colnames(prediction$ratios)<-"Prediction" #set column name
42  prediction$status<-list("Prediction") #set status
43  return(new("arrayData",prediction)) #return prediction
44  }
```

The function loads the required packages if not already done so (L4–5) and assigns a vector to store the predicted probe values (L6). Where provided, that the sequences have corresponding probability values is checked (L7). A loop through the chromosomes is initiated (L8) and the corresponding probe data extracted (L9–10). If coordinate values are not provided (L11) the sequence for the current chromosome is extracted, unmasking it where required (L12), and a loop through the provided sequences initiated (L13). The positions of the occurrences of the sequences are found (L14) and used to calculate the density values (L15). These density values are scaled to have a

maximum value of 1 (L16–17) and adjusted to have the correct chromosomal coordinates (L18). Values at probe positions are calculated, multiplied by the relevant probability value and added to the probe score (L19). If coordinates are provided (L21) their format is checked (L22–23) and the positions for the current chromosome extracted (L24). If positions for the current chromosome are present (L25) the density is calculated (L26) and adjusted and stored as for sequences (L27–30). Probability values are extracted (L31) and used to adjust each of the predicted peaks (L32–33). Any probes containing NA values are assigned a zero value (L38) and the values stored in an `arrayData` object (L39–40). The column names and status are updated (L41–43) and the results returned (L43).

6.6 Application

The script was used to predict the CPD induction profile of the yeast genome at the probe locations of the G4493A microarray. As the predicted values are arbitrary, the predicted data is adjusted to make it comparable to the microarray data. This is achieved by calculating the equation of the line of best fit ($y = mx + c$) through the two datasets and adjusting the predicted data to make this the line $y = x$. The values m and c of the equation of the line of best fit are found from the function `lm` in `R`. The c value is subtracted from the predicted values, which are then divided by the m value. This does not affect the correlation between the two sets of data and so does not skew any further analyses of the data.

A short section of chromosome 1 is shown in Figure 6.2 (the whole genome is shown in "CPD predicted and actual profile.pdf" in the electronic appendix; see Page 367), showing the profile of the predicted and detected levels of DNA damage induction are in good agreement. Throughout the whole genome the Spearman's correlation coefficient is 0.77 (Figure 6.3), indicating that the data from the whole microarray are a good reflection of the damage induction throughout the genome. This confirms the efficacy of the microarrays in detecting genome wide damage events.

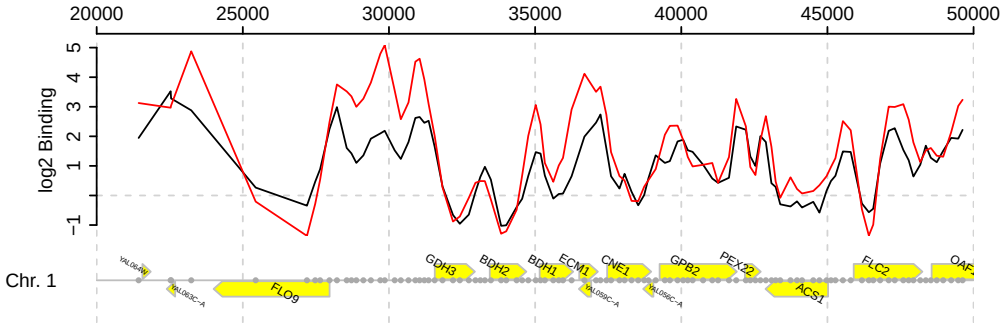


Figure 6.2: CPD damage profile: A section of chromosome 1 showing the predicted (red) and actual (black) microarray CPD values. Probe positions are shown with grey circles. ORFs are shown as yellow boxes with the arrow indicating the direction of transcription.

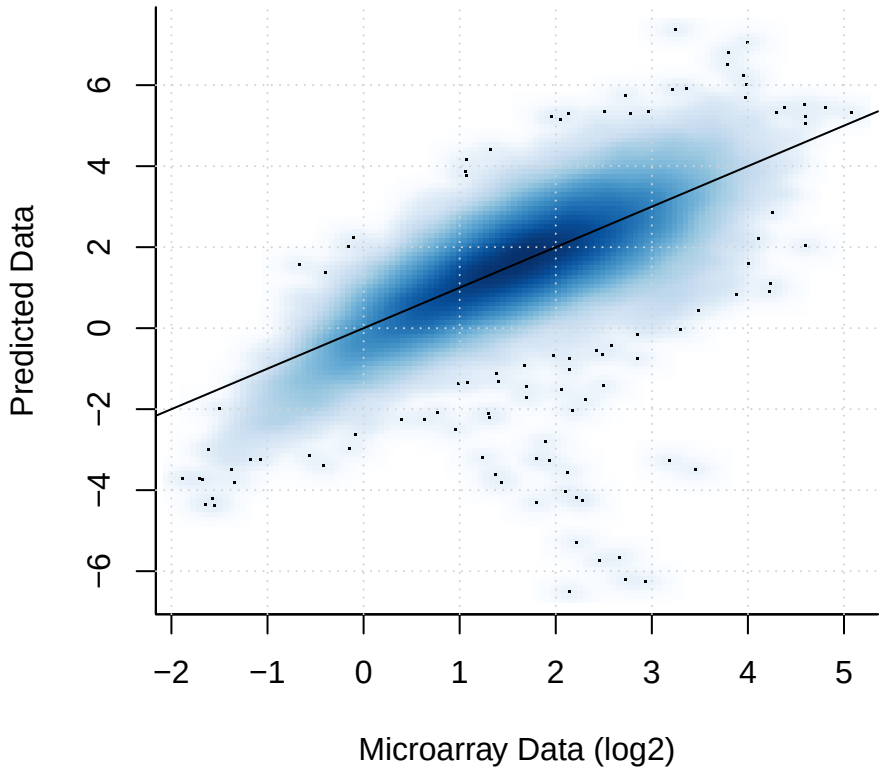


Figure 6.3: CPD damage scatter plot: The relationship between the predicted and actual microarray CPD values for each of the probes on the G4493A microarray. The darker the blue colour the more points occur in the region. Individual points outside of the central region are shown with dots. The black line shows $y = x$.

6.6.1 Comparisons

There are several possible reasons the correlation value is not higher than 0.77. One is that there are genuine biological differences between the actual and predicted values. Another is that the level of inherent variation in the microarrays is such that even if the true biological values are exactly predicted by the algorithm, they may not be shown as such on the microarrays. In this case, taking the average of several normalised datasets will improve the correlation as some of the variation will be cancelled out. This only provides a limited solution with the two currently available datasets. Another reason may be the inverse of this; that the microarray data are accurate but the prediction is not, due to factors such as variations in the ratio of induction of DNA damage at the different dipyrimidine sites. These scenarios are not mutually exclusive and so both could be contributing. These ideas were tested by examining the variation between the actual and predicted values. Working under the assumption that the prediction is fully accurate throughout the whole genome and the microarray data vary, these variations will cause approximately equal numbers of predicted values to be lower and higher than the true biological values, thereby reducing the observed correlation coefficient. The same would be true under the assumptions that the microarray data are fully accurate and the prediction varies, or that both have some level of variation, as the variations would occur randomly throughout the datasets. Alternatively, it is possible that there are some regions of the genome which, for various biological reasons, have higher or lower levels of *in vivo* damage induction. These will therefore deviate from the predicted values, reducing the observed correlation coefficient. In this case the differences are unlikely to all be random, with some focussed in regions representative of hot or cold spots of damage.

To identify any such regions, the differences between the predicted and actual damage values were calculated. If the differences between the two sets of values are due to random variations as a result of inherent variations in the microarray technology, random variations in the locations of damage throughout the genome, and inaccuracies in the prediction method, it is

reasonable to assume these will follow a normal distribution. Specific regions of increased or decreased damage are likely to fall outside of this normal distribution and so can be found by standard outlier detection methodologies.

The differences are shown as a histogram in Figure 6.4. The normal distribution curve added to this histogram with the mean and standard deviation calculated from the differences (0 and 0.7 respectively) shows that the majority of the difference values approximate the normal distribution. This suggests that the majority of damage events throughout the genome occur at the expected rate and are not in areas of specifically increased or decreased damage.

Plotting the differences as a Q-Q plot shows that some difference values at the ends of the range deviate from a normal distribution (Figure 6.5). This suggests that there are probes with variations in the levels of damage beyond that which would be expected if the variations arose entirely at random. The plot shows that most of these probes have higher values, representing more damage, than the prediction.

Outlier detection was used to identify the probes that deviate away from the expected normal distribution. The "extremevalues" package (van der Loo, 2010) was used to determine these values, which estimates the underlying distribution of the values and uses the properties of this to determine outliers. A Q-Q plot highlighting these detected outliers (Figure 6.6, produced using the extremeValues package) shows that the the remainder closely follow the expected normal distribution. The outlier detection identified 115 probes (0.28% of the total). Visual analysis of the positions of these probes showed that the majority occur at the beginnings and ends of chromosome probe regions, where the prediction is consistently low, suggesting that it is not acurate in these regions (all detected outlying probes are highlighed with red crosses in the plot of predicted and actual CPD values in "CPD predicted and actual profile.pdf" in the electronic appendix; see Page 367). It is also possible that the probe quality is lower in these regions because telomeres contain repetative sequences, potentially reducing the reliability of some of the probe values. Other deviating probes tend to occur individually or in short consecutive regions, which appear to cluster together. Chromosomes

1 and 6, for example, have several such sites, whereas those betwen have none. It is possible that the deviation from the prediction at these points is due to genuine biological changes in the induction of damage at these sites. However, more testing would be required to gain further evidence for this.

A similar analysis was undertaken with 2 replicate cisplatin induced DNA damage microarray datasets, produced by James Powell (PhD thesis in production). This is a chemotherapeutic drug which produces adducts at a number of sequence specific sites, of which an antibody against GG adducts was employed. The prediction was run for GG sites accordingly, which gave a Spearman's rank correlation value of 0.55. A total of 57 outlying difference values were found for these data. As with the CPD data, many outlying differences were found at the ends of chromosomes, further suggesting the prediction and/or the microarrays are unreliable in these regions. The remainder are isolated sites of small numbers of probes, different to those identified in the CPD data. Once again, these may be indicative of regions at which the damage induction rate is different to the rest of the genome, but more evidence is required. The whole genome plot of predicted and actual damage, highlighting probes detected as outliers, is provided in "Cisplatin predicted and actual profile.pdf" in the electronic appendix (see Page 367).

These findings suggest that generally CPD and cisplatin DNA damage induction occurs at a uniform level throughout the genome (taking into account sequence variations), and that any potential hot or cold spots of damage are at a level below the detection threshold of the microarrays. Many of the outlying differences between the predicted and actual microarray values appear to be as a result of inconsistencies in the prediction, the microarray probe values, or a combination of the two. A higher resolution microarray may enable a more sensitive analysis, as would some additional biological replicate datasets (work which is currently being undertaken in our laboratory). These data may be combined with a more stringent outlier detection method to provide a more robust analysis. It may also be possible to use the microarray data to refine the ratios of damage induction used in the prediction to improve its accuracy.

Aside from the outlying difference values, which requires more data for

further investigation, it is evident that the prediction method developed here is able to generate accurate representations of genome wide DNA damage induction. It is also clear that this novel use of microarrays is able to detect varying damage induction throughout whole genomes and therefore has the potential to be able to detect variations in damage levels from a set of predicted values, which may have important biological implications.

6.6.2 Uses

The primary application for this type of assay is to measure DNA damage levels and analyse repair rates throughout whole genomes. There are many additional potential uses for the technology. In pharmacology, analysing how DNA and chromatin binding drugs localise, interact and perform in response to damaging agents in the human genome could indicate how they operate, whether or not they reach their intended targets, or if there is any non-specific or off-target binding, all in the context of local DNA repair rates. In pharmacogenomics, analysing drug target sites and the relationship with repair in individual genomes could show whether or not a given drug is suitable for a particular person (stratified medicine). In drug discovery, analysing the effects of novel compounds on repair rates could reveal new therapeutic agents to be used in the fight against cancer. Analysing where different proteins bind, or are prevented from binding, and how this affects DNA repair rates in cancer genomes may provide useful prognostic tools, cancer biomarkers and the determination of the response to the drug.

Analysing repair rates alone could also help to identify coding and other important regions of the human genome. Currently the total number of genes and their respective positions in the genome are not known. Additionally, there may be important regulatory or other non-coding regions yet to be identified. Regions of fast repair identified by this technology may be indicative of these regions, either due to TC-NER operating in undiscovered transcribed regions or as the cell prioritises repair by some other mechanism in important non-coding regions.

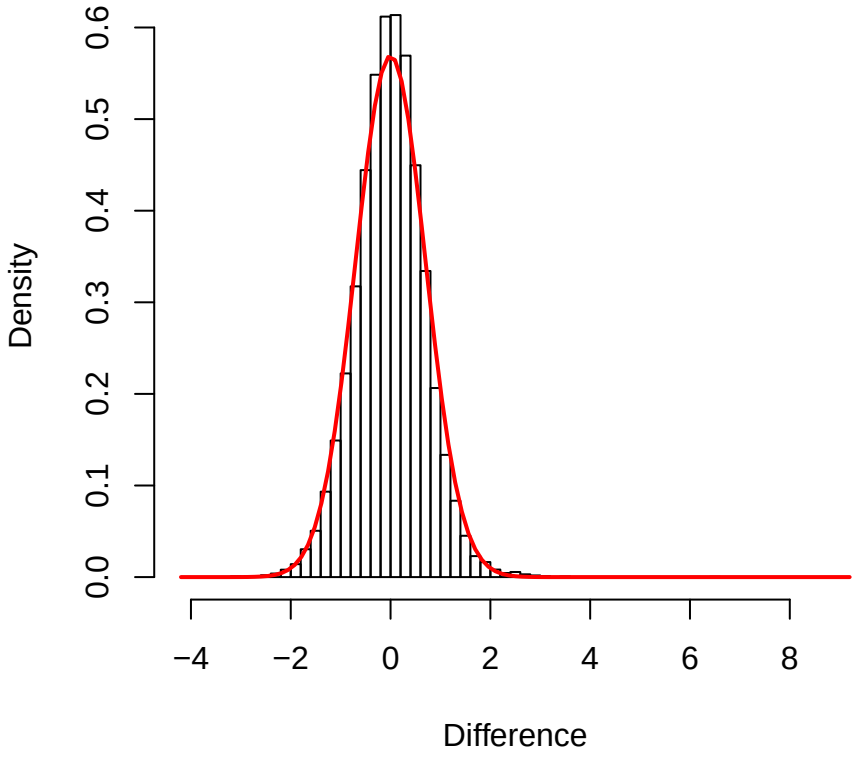


Figure 6.4: Histogram of predicted and actual differences: The frequencies of difference values between the predicted and actual CPD microarray values. The red line shows the normal distribution calculated from the mean and standard deviation of the difference values.

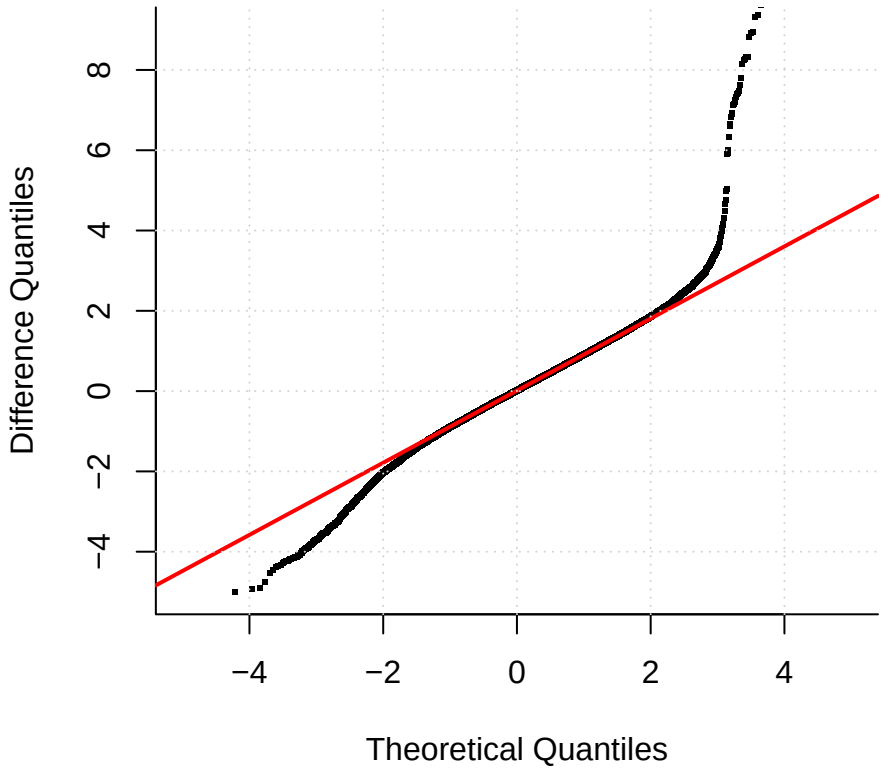


Figure 6.5: Q-Q plot of predicted and actual differences: The difference quantiles and theoretical normal distribution quantiles are shown with dots. The red line represents normally distributed data. The majority of the difference values follow this line, with a small deviation at the low end and a large deviation at the high end.

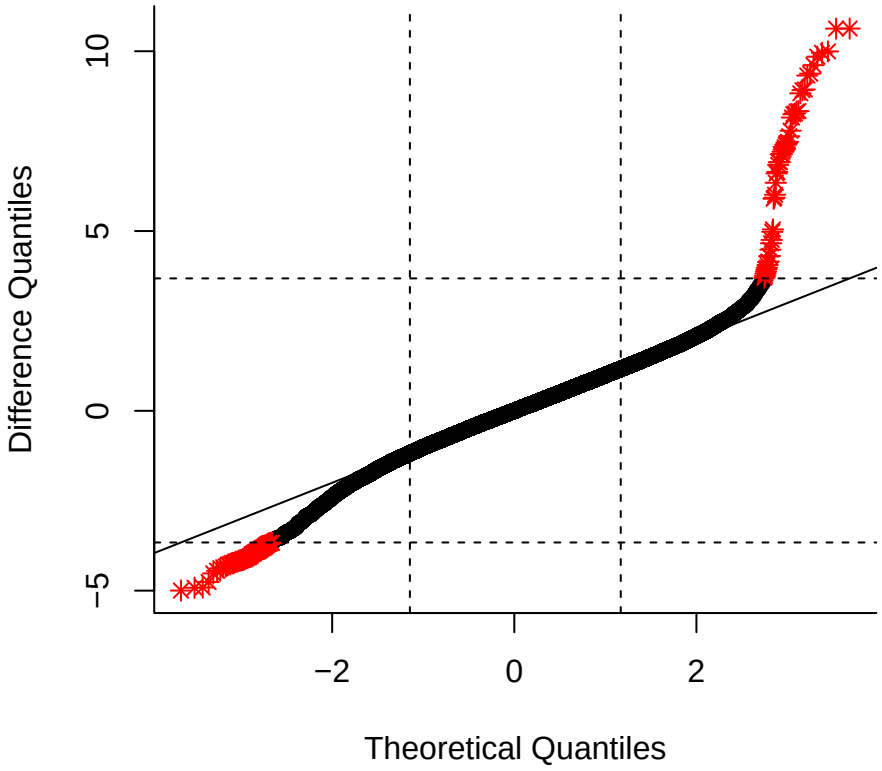


Figure 6.6: Q-Q plot of predicted and actual non-outlier differences: The difference quantiles and theoretical normal distribution quantiles. The black points represent normally distributed data while the red points show detected outliers. The black line represents normally distributed data. The majority of the difference values follow this line showing they approximate a normal distribution.

6.7 Discussion

The formula and its application to predict the profile of damage created by a sequence specific damaging agent presented here has enabled the confirmation that microarrays can detect genome wide damage levels. This has been shown for CPDs (Teng et al., 2011) and is also currently being used to investigate damage induced by chemotherapeutic platinating agents, with similar results. This validation of microarray technology for DNA damage detection is important in the field of DNA damage research as it allows a new way to analyse damage events at a high resolution throughout a genome.

Comparisons between the predicted CPD levels following UV irradiation and the actual values from the microarrays allowed an estimation of significant differences between the two to be determined. This was based on the assumption that random differences between the two, due to noise in the microarray data, inaccuracies in the prediction, or a combination of the two, would follow a normal distribution. Non-random, and therefore potentially biologically relevant, differences may have indicated regions of damage above or below the expected level. Outlier detection was used to identify any such regions, which found only a relatively small number of probes, which were not linked to biologically significant variations. The same conclusions were drawn from a similar analysis of cisplatin induced DNA damage data. These analyses need to be repeated with more replicate datasets to confirm the findings.

Chapter 7

Genome wide analysis of the binding site locations of the Abf1 protein

7.1 Introduction

Abf1 is an essential yeast general regulatory factor (GRF) with multiple roles in the cell (Buchman et al., 1988; Fourel et al., 2002; Lascaris et al., 2000; Loo et al., 1995; Miyake et al., 2002; Rhode et al., 1992), including GG-NER (Reed et al., 1999). Genome wide binding site investigations have been undertaken previously (Lee et al., 2002; Harbison et al., 2004; Schlecht et al., 2008) which have revealed around 1,500 sites. The microarrays employed in these investigations were of a lower resolution than the Agilent G4493A microarrays, and so these have been used here to to conduce a higher resolution analysis than previously undertaken. Genome wide Abf1 binding ChIP-chip datasets from before and after UV-irradiation were created and investigated by Dr. Matthew Leadbitter (Leadbitter, 2011). Those same datasets have been reanalysed here with a focus on the unirradiated datasets. The investigation here uses the previously described bioinformatic tools to analyse Abf1 binding. The objective of this analysis was to demonstrate the tools' applications to real ChIP-chip data, to analyse these results to identify any novel information about the genome wide binding properties of Abf1 and to compare these results to those of previously published investigations.

7.2 Methods

7.2.1 Generation of data

All ChIP-chip microarrays analysed in this chapter were created by Dr. Matthew Leadbitter. This process is detailed in Leadbitter (2011) and outlined in Chapter 2. Briefly, an antibody against the Abf1 protein was used in the immunoprecipitation procedure, which was carried out on yeast BY4742 cells with no UV treatment (U), immediately following UV treatment (0) and 30 min following UV treatment (30). Three biological replicates were carried out for each time point, generating nine datasets. The file names of these are shown in Table 7.1 and the raw data files are provided in the "Abf1 microarray datasets" folder in the electronic appendix (see Page 367).

File name	Description
Abf1_U_1.txt	No UV treatment replicate 1
Abf1_U_2.txt	No UV treatment replicate 2
Abf1_U _3.txt	No UV treatment replicate 3
Abf1_0_1.txt	0 minutes after UV treatment replicate 1
Abf1_0_2.txt	0 minutes after UV treatment replicate 2
Abf1_0_3.txt	0 minutes after UV treatment replicate 3
Abf1_30_1.txt	30 minutes after UV treatment replicate 1
Abf1_30_2.txt	30 minutes after UV treatment replicate 2
Abf1_30_3.txt	30 minutes after UV treatment replicate 3

Table 7.1: Abf1 ChIP-chip datasets: dataset file names used in this investigation and available in the accompanying electronic appendix (see Page 367).

7.2.2 Data validation

The `checkData` function was used to check the quality of the microarray datasets. The results of this are provided in "check.pdf" in the electronic appendix (see Page 367) and show that the data are of good quality, with no suggestions of any reasons not to use any of the datasets. All nine files were therefore loaded into `R` and used throughout the remainder of this analysis.

7.2.3 Data normalisation

All data were normalised using the procedure described in Chapter 4. This process removes probe values known to be irrelevant, namely from the mitochondrial genome and deleted genes, quantile normalises replicate datasets, shifts the new distributions' pseudo-modes to zero and scales the distributions to make the negative part of the distribution approximate the equivalent part of the normal distribution.

7.2.4 Peak detection

Peak detection was carried out with each set of normalised replicate datasets using the enrichment detected method described in Chapter 5 to determine potential sites of Abf1 binding. All datasets for each condition were analysed together using the optimal settings determined in Chapter 5: a window size of 1 (shown to produce the best peak detection results with multiple datasets), an FDRE value of 0.9 (the optimal balance of sensitivity and specificity) and no scaling (unnecessary because the data appear to meet the expectations of the enrichment detection algorithm). The function was set to find peaks and therefore the results were returned as three new `peakList` objects; one for each of the three experimental conditions investigated.

7.2.5 Hypergeometric distribution

The hypergeometric probability function `phyper` in `R` was used to calculate the significance of overlaps between different groups of genes, based on the total number of ORFs loaded by the `loadAnnotation` function, which is 7,071. The four arguments of the function (q, m, n, and k) were provided as the number of overlapping gene names between this and the published investigation being compared, the total number of genes in the published investigation, 7,071 - the total number of genes in the published investigation (such that m + n = 7,071) and the number of gene names found in this investigation, respectively. The result was subtracted from 1 to get the p-value.

7.2.6 Sequence extraction

The enrichment detection function, when set to detect peaks, provides an estimate of the PBR — the region likely to contain the genuine binding site leading to the creation of the peak. These estimates (chromosome numbers and start and end coordinates) were used to extract sequences for each PBR with the `getSequences` function. The UCSC sacCer3 (April 2011) genome release was used for this, downloaded using tools from the `BSgenome` package.

7.2.7 Sequence analysis

Extracted sequences were written to files in the FASTA format and analysed with BioProspector (Liu et al., 2001). All sequences are provided in "Sequences at detected PBRs.fasta" in the electronic appendix (see Page 367). For each FASTA file, the program was run twice, varying the setting to search for motifs present in all sequences. Running the program under the condition that the motif does not have to be present in every sequence allows the strongest motif results to be found, which may or may not be present in all of the sequences in the file. Running the program under the condition that the motif must be present in every sequence allows the motif represented the most in all sequences to be determined, which may be different from the strongest motif. Searches for a single, continuous sequence were carried out with a width of 15, 2 positions greater than the length of the current consensus. Searches carried out for two discrete sequences were carried out using two widths of 6 and a gap between them ranging from 3 to 7, making the maximum analysed sequence length 19. All other settings were left as defaults, each search returning the top 5 motif results.

7.2.8 Motif logo creation

Binding motifs discovered during the sequence analysis were converted to graphical forms for display using WebLogo version 3.3 (`http://weblogo.threeplusone.com`; Crooks et al., 2004), setting the sequence type to 'DNA', base composition to '*S. cerevisiae*', colour scheme to 'classic', and using all

other default settings. This allows the prevalence of the different bases at each site to be visualised. Each position shows a stack of bases, displayed in different colours, the heights of which represent the frequency of occurrence of each base at each position. The overall height at each position corresponds to the bit score (y axis), representing the sequence conservation at the position. Therefore if all bases appear randomly at a site the height of the stack will be at or close to zero, representing no sequence conservation at the site. If the same base(s) frequently occur(s) at a site the height of the stack will increase and the most frequent base(s) will be indicated in the stack in the proportion in which they are found. This enables more information about a binding site sequence to be visualised than a consensus sequence alone. Motifs are assigned a score, as described by Liu et al. (2001), with higher values representing more robust results.

7.2.9 Ganapathi data

The downloaded microarray data from Ganapathi et al. (2011) is provided in BAR file format. This is an Affymetrix file format (details available at `http://www.affymetrix.com/support/developer/powertools/changelog/gcos-agcc/bar.html`) which cannot be read by standard programs. They were therefore converted to tab-delimited text files using a tool available in the CisGenome package (Ji et al., 2008). This was downloaded as a ZIP file and extracted to a known location. The Windows command line editor was used to navigate to the 'bin' folder in this location. From here the `affy_bar2txt` program was run with each of the BAR files, converting them to text files. These text files contain three columns: the chromosome number, a single coordinate for the probe and the probe value. The data from these text files were used for all analyses.

The microarrays used in this investigation were a much higher resolution that those used to create the Abf1 binding data analysed here (3,115,004 probes compared to 41,775). In order to make comparisons between the two, the Affymetrix data was reduced to the locations of the probes one the Agilent G4493A microarrays. This was achieved with the `approx` function

in R, which interpolates values from a dataset at a given set of positions. In this way values from the Affymetrix data were approximated at the locations of the Agilent probes, allowing the two datasets to be compared optimally. As the resolution of the Affymetrix microarray is so much higher than the Agilent, the majority of these interpolated values should map very closely to actual probe values.

The two sets of Abf1 binding sites determined in this investigation and by Ganapathi et al. (2011) are presented as coordinates. These were used to compare the two sets of results. The Ganapathi et al. (2011) regions are much shorter than the PBRs determined in this investigation (median length of 65 compared to 307), as a result of the higher array resolution. Therefore any two sets of coordinates sharing any overlap were deemed to represent the same region and therefore the same Abf1 binding site.

7.3 Results

7.3.1 Data validation

The output of the `checkData` function for the first untreated dataset is shown in Figure 7.1 and are provided in "check.pdf" in the electronic appendix (see Page 367). All were visually examined and found to be within reasonable limits, as described in Section 3.2.2. The pseudo-images of the red and green channels do not show any signs indicative of artifacts on or damage to the microarray surface. The final dataset (Abf1_u_3.txt) shows a small area of reduced hybridisation in both channels in the lower left-hand corner, but not at a level that gives cause for concern. As both channels look to have the same pattern of reduced hybridisation the ratio between the two should not be adversely affected.

The box plots show the bulk of the signal intensity values to be within the expected range, around a $\log_2$ signal intensity of 15, which is consistent with all microarray datasets produced in our laboratory. There are no large differences between the red and green intensity values for each dataset, suggesting the labellings and hybridisations were equally effective in both channels. The

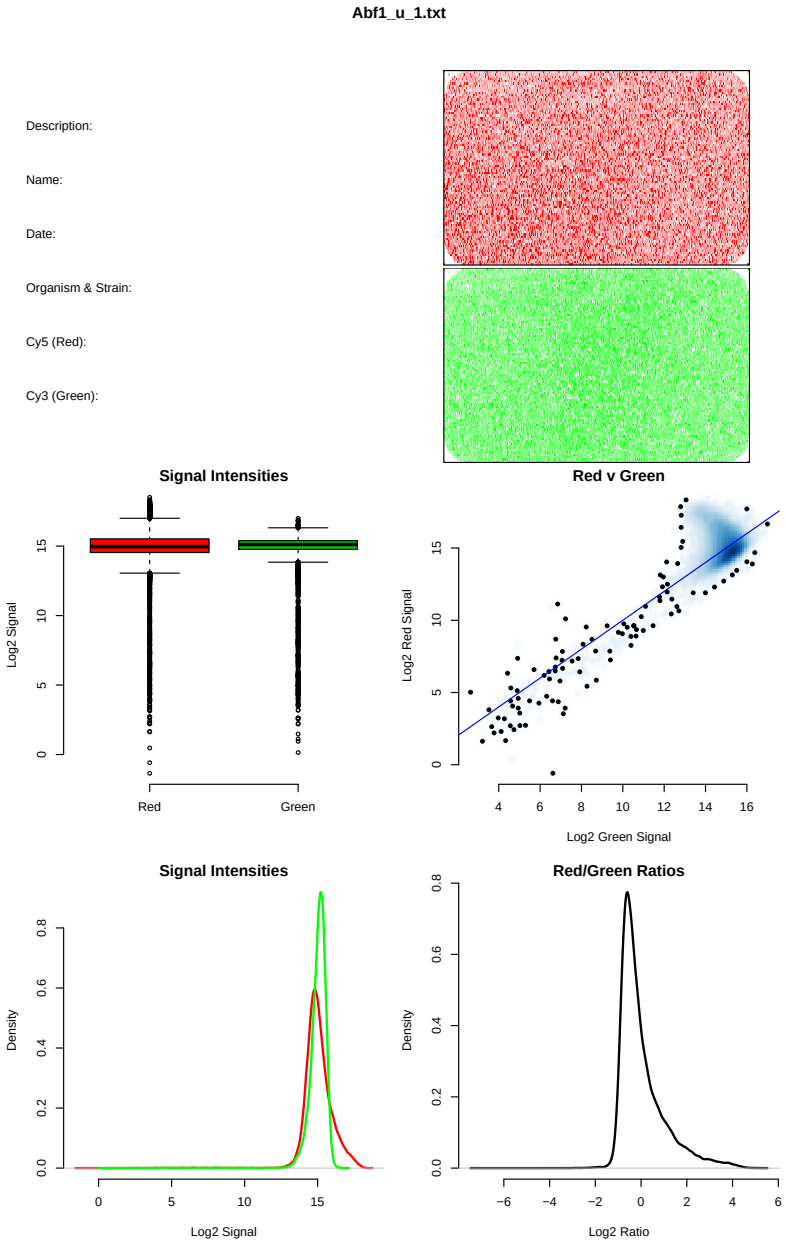


Figure 7.1: Abf1 `checkData` output: quality control plots for the first Abf1 dataset, showing the data to be of good quality suitable for further analysis. All plots are provided in the electronic appendix (see Page 367).

small number of outlying values, mostly at the lower end of the scale, are likely due to probes known to be unreliable, such as those for the mitochondrial genome, which will be removed as part of the normalisation process.

The scatter plots show consistent one-to-one relationships between the data from the two channels, with the bulk of the data around the $\log_2$ signal intensity value of 15, as before. This confirms the data from the box plots and suggests that there is not a large amount of dye bias present between the two samples. The region extending from the bulk of the data towards the red signal region of the graph represents binding sites of Abf1, where the red signal intensities are higher that the green. This area is present in all datasets and confirms the presence of a large number of binding regions, as would be expected.

The shapes of the distributions of the signal intensities are smooth, indicating no unusual effects. Once again, the medians are centred around the expected region of a signal intensity of 15. The input sample (green) distributions are uniform, as is expected for the total genomic DNA, while the IP sample (red) distributions are skewed to the right, as is expected because a subset of the genetic material is present in increased amounts.

The distributions of $\log_2$ ratios are all similar and of the expected shape, with no major distortions. The left hand skew is due to the higher signal intensities of immunoprecipitated material seen in the distribution of red signal intensities. Many datasets have the median centred around zero, which is to be expected as this region should represent the background, non-immunoprecipitated, regions. It does not pose a problem however that some are not centred around this region, as this is one of the factors corrected for in the normalisation procedure (see Chapter 4). On the basis of these QC checks it was decided that all nine datasets were suitable for loading into `R` for analysis.

7.3.2 Consequences of normalisation

The effect of the normalisation procedure on the shape of the density plots of all datasets can be seen by comparing the density distributions of the

raw and fully normalised datasets (Figures 7.2 and 7.3), where each set of replicates is shown as a different colour. The nine raw datasets are not all aligned together, with several pseudo-modes lying away from zero. The fully normalised data shows all pseudo-modes align at zero. The replicate datasets appear as single lines as the quantile normalisation procedure has caused them to follow the same distributions. The left hand side of the distributions, below zero, align closely together as a result of their being scaled to follow the standard normal distribution over this region. As there is only a small amount of variation between the raw datasets, the main effect of the procedure has been to increase the binding values as a result of the background scaling procedure.

Plots along a 30 kb section of chromosome 1 (Figures 7.4 and 7.5) show the effects on a small section of data in the context of the genome. In this instance the normalisation procedure has not caused large changes in the data, that is to say, they were close to the optimal state that the normalisation attempts to achieve before the process was applied.

The raw data points for each replicate dataset were plotted against their normalised values to visualise the changes caused by the normalisation (Figure 7.6 A-C). Values removed by the normalisation procedure are not plotted. All datasets produced similar, reasonably straight lines in these plots, showing them to follow similar distributions which the normalisation procedure did not change. They all deviate markedly from the line $y = x$ as a result of the scaling procedure. These results match with the expected result of the normalisation, given that the distributions of the raw datasets were very similar.

Plotting the averages of the three replicates of each dataset (Figure 7.6 D) shows the relationships between the different conditions before and after normalisation. Again, this shows little difference between the datasets, especially towards the upper half of the data. The red line (0 min after UV treatment) deviates from the other two in the lower half of the raw data, which is corrected for in the normalised data.

Plotting the three untreated datasets against each other (Figure 7.7) shows that the three replicates are very consistent. The Spearman's cor-

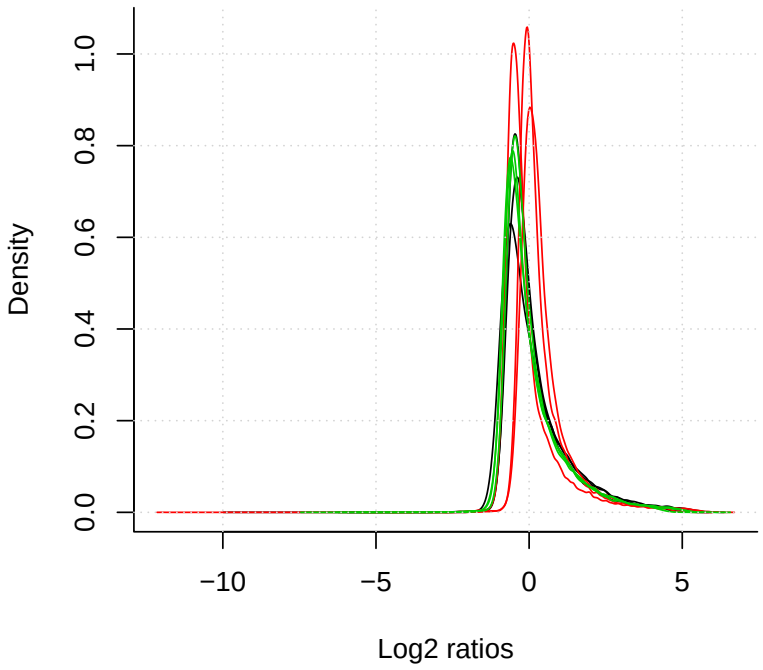


Figure 7.2: Raw Abf1 data density plots: The three biological repeats are shown for each of the no UV treatment (black), immediately following UV treatment (red) and 30 min following UV treatment (green) Abf1 binding datasets.

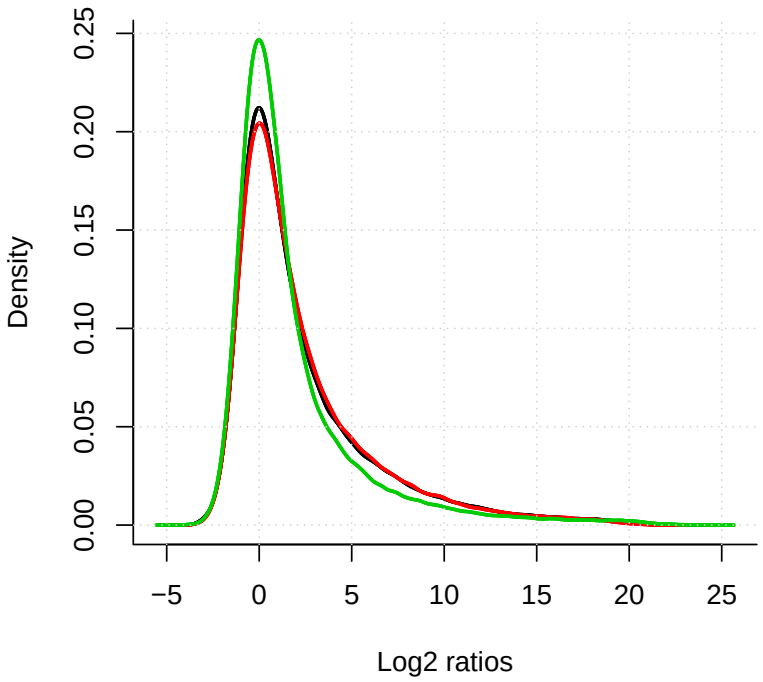


Figure 7.3: Normalised Abf1 data density plots: The three biological repeats are shown for each of the no UV treatment (red) and 30 min following UV treatment (green) Abf1 binding datasets. As quantile normalisation makes datasets follow the same distribution, the three lines for each dataset overlap each other and appear as a single line.

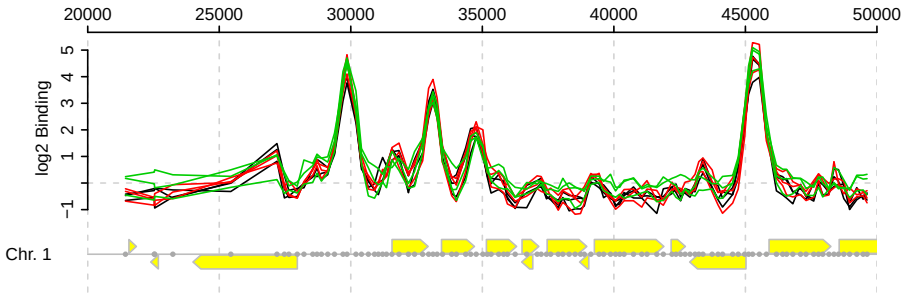


Figure 7.4: Raw Abf1 data profile: A section of chromosome 1 showing each of the no UV treatment (black), immediately following UV treatment (red) and 30 min following UV treatment (green) Abf1 binding datasets.

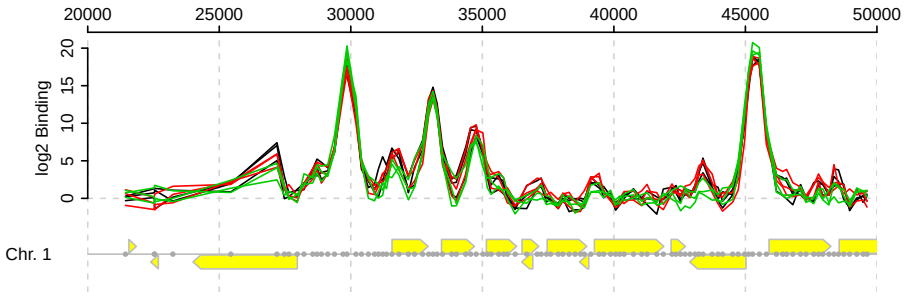


Figure 7.5: Normalised Abf1 data profile: A section of chromosome 1 showing each of the no UV treatment (black), immediately following UV treatment (red) and 30 min following UV treatment (green) Abf1 binding datasets. The normalisation procedure has reduced the variation in the background regions (coordinates ~40,000 to ~43,000 for example) and altered the order datasets appear at the top of some of the peaks. The scaling of the datasets has increased the binding values of the peaks by around 4 times.

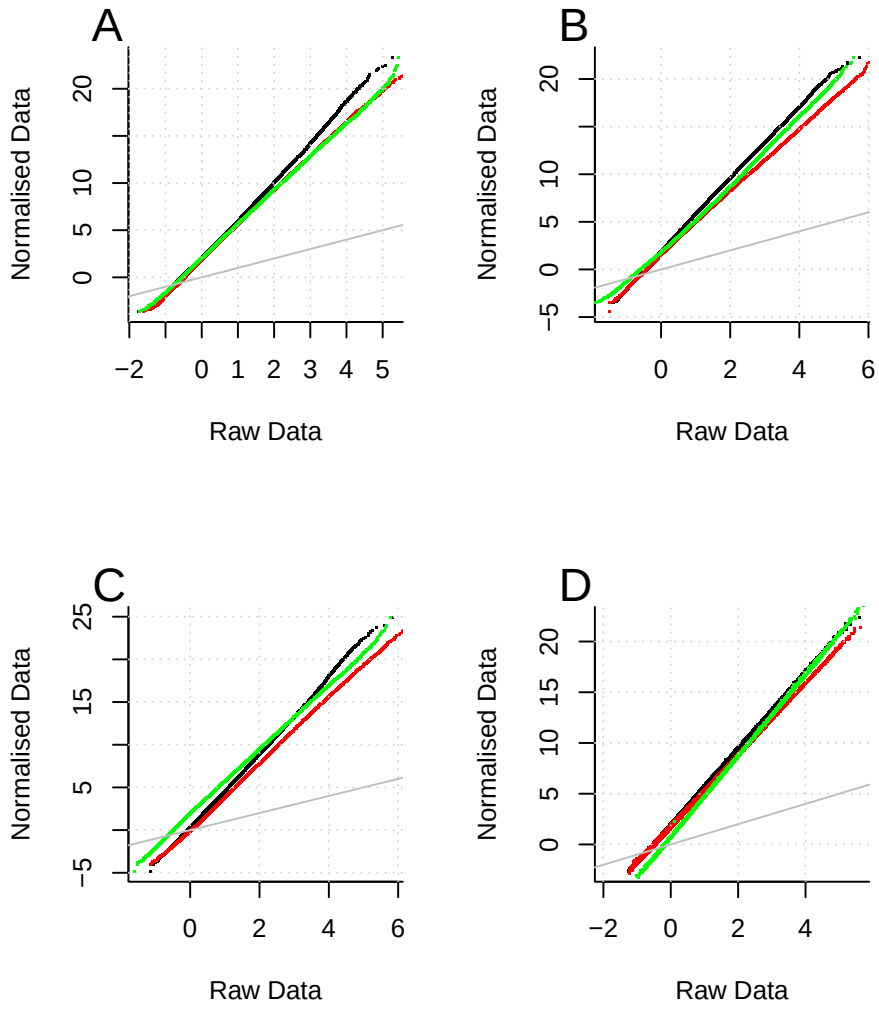


Figure 7.6: Effect of normalisation on averaged Abf1 data: Scatter plots of the three replicates each (black, red and green points) of no UV treatment (A), 0 min after UV treatment (B) and 30 min after UV treatment (C) datasets, and the averages of the no UV treatment (black), 0 min after UV treatment (red) and 30 min after UV treatment (green) dataset (D) before and after normalisation. The grey line shows $y = x$. The relatively straight lines of data indicate that the normalisation procedure has not had a large effect on the shape of most of the data, with only a small number of points deviating from this trend.

relation values show good reproducibility, suggesting the data are well suited to the enrichment detection procedure.

7.3.3 Peak detection

Peak detection was carried out for each set of replicates. The numbers of peaks found for each condition are shown in Table 7.2.

Venn diagrams were created to display the relationships between these datasets (Figures 7.8 and 7.9) with the `venn` function, which provides two methods for creating Venn diagrams from the results of the enrichment detection algorithm (Section 3.2.7.2). Briefly, the first method uses only the probes determined to be at the tops of the averaged peaks, created by averaging all replicate datasets. The numbers therefore represent the numbers of probes falling into each of the categories. The second method uses the PBRs of each peak, which may extend beyond the probe at the centre of the average peak. The numbers therefore represent regions falling into each of the categories, where an overlap between two regions, as well as a complete match, counts as a peak occurring in both datasets. The relative pros and cons of each method are discussed in Section 3.2.7.2. It is felt that for the purposes of this investigation the overlapping diagram is more informative as it is likely that peaks differing by a single probe will be caused by the same Abf1 binding site and therefore should be counted once.

Both diagrams show that the majority of peaks do not change between the different conditions. This matches a previous analysis of these datasets (Leadbitter, 2011) which concluded that there were no significant UV-induced changes in Abf1 binding sites. The peaks that are not common to all datasets were determined to be either present in all datasets, but at a level slightly below the detection threshold in one or more, or present in all datasets but at adjacent probes so that they appear as separate peaks but are likely caused by the same binding site. A full analysis and explanation is available in Leadbitter (2011) and these analyses will not be repeated here. The data are included to show the consistency of Abf1 binding across nine biological repeat datasets under three different experimental conditions. This suggests

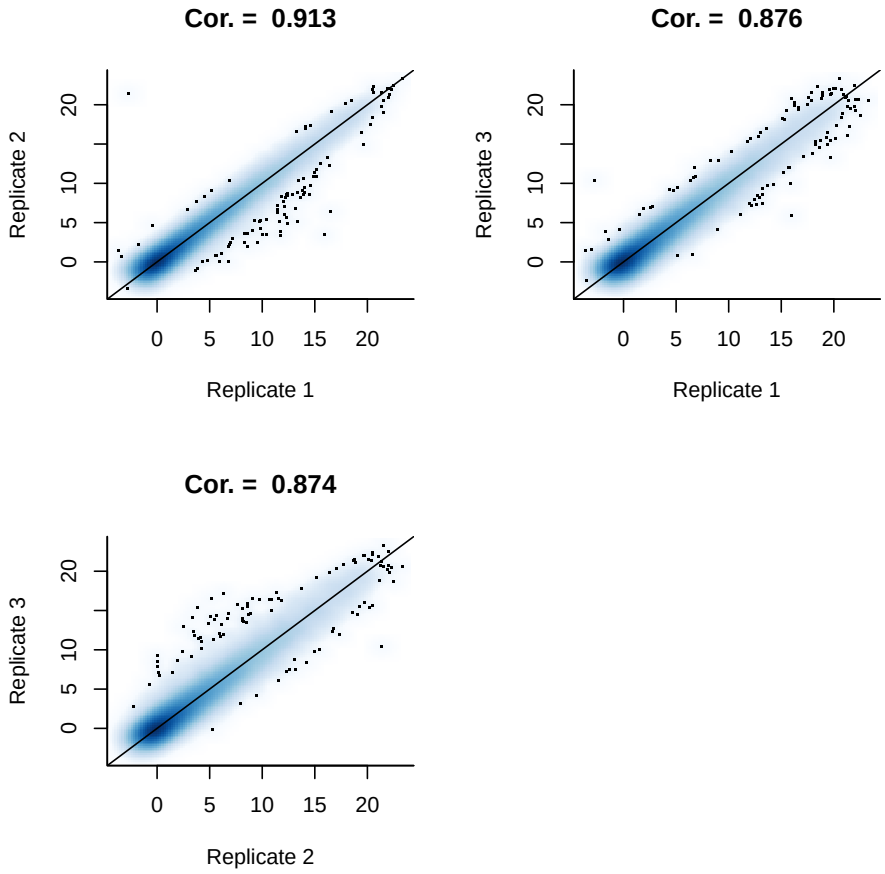


Figure 7.7: Abf1 data correlations: The darker the blue colour the more points occur in the region. Individual points outside of the central region are shown with dots. The black line shows $y = x$. The Spearman's rank correlation value is shown above each plot.

Condition	Number of peaks detected
No UV treatment	4369
0 minutes after UV treatment	4261
30 minutes after UV treatment	3489

Table 7.2: Numbers of Abf1 binding peaks detected

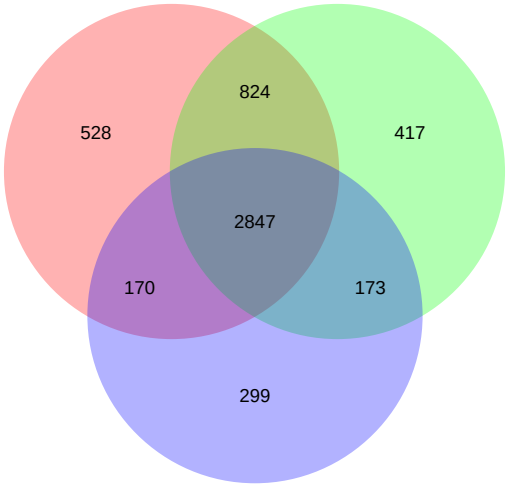


Figure 7.8: Venn diagram of Abf1 peaks: The relationship between the no UV treatment (red) 0 min after UV treatment (green) and 30 min after UV treatment (blue) Abf1 binding datasets based on the probes at the centres of the peaks as determined by the enrichment detection algorithm.

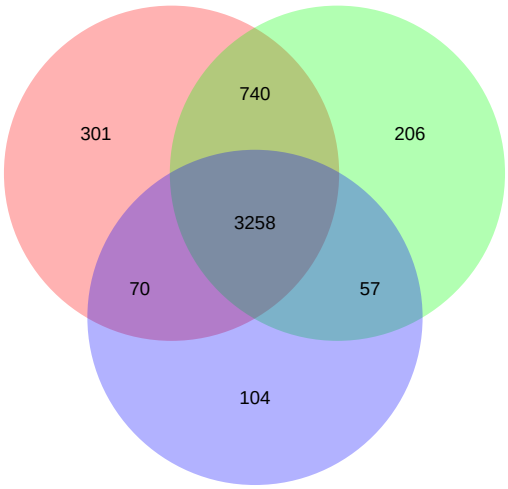


Figure 7.9: Venn diagram of overlapping Abf1 peaks: The relationship between the no UV treatment (red) 0 min after UV treatment (green) and 30 min after UV treatment (blue) Abf1 binding datasets based on overlaps between potential binding regions determined by the enrichment detection algorithm.

that Abf1 binding is targeted to certain points in the genome, the majority of which do not then vary under these different conditions.

Visualisation of the changes in peak heights, rather than positions, was achieved with a rainbow plot (Figure 7.10). Here each peak in the untreated dataset is represented by a line, which tracks the peak height across the next two time points. The no UV and 0 min after UV treatment data are very similar, visible by the colour transition being similar in both columns. Many peaks show a deviation 30 min after UV treatment, both up and down, visible by the expanded range of binding values in the final column. These results are confirmed with scatter plots which show that the no UV and 0 min samples are similar (Figure 7.11A), while there is more variation between these and the 30 min sample (Figures 7.11B and C). The general trend is for lower values in the 30 min sample.

To confirm that the peak detection process had worked correctly the `profilePlot` function was used to create a plot of all 4,369 detected regions (Figure 7.12A). A second profile plot of 4,369 randomly generated probes was also created for comparison (Figure 7.12B). Each individual line is shown in black and it can be seen that the detected regions show a clear peak shape centred around the probes determined to be at the tops of the peaks. The trend line, calculated from all lines and shown in red, follows the same pattern. This shows that the average peak has a height of a $\log_2$ binding value of ~7 and a width at the base of ~2000 nt. Beyond this, towards the extremes of the plot region, all values are centered around zero with no visible peaks. This shows that these are background, unbound regions. The plots of randomly generated sites do not show any pattern, with all black lines falling randomly across the width of the graph. The red trend line is flat across the width of the graph, showing no pattern is present. Taken together, these results show that the peak detection process has found only regions of genuine peaks.

The number of potential binding sites detected in this investigation is far greater than those identified in previous investigations, which prompted the analysis of the sequences around the binding sites to determine whether or not they contained the consensus binding sequence previously identified

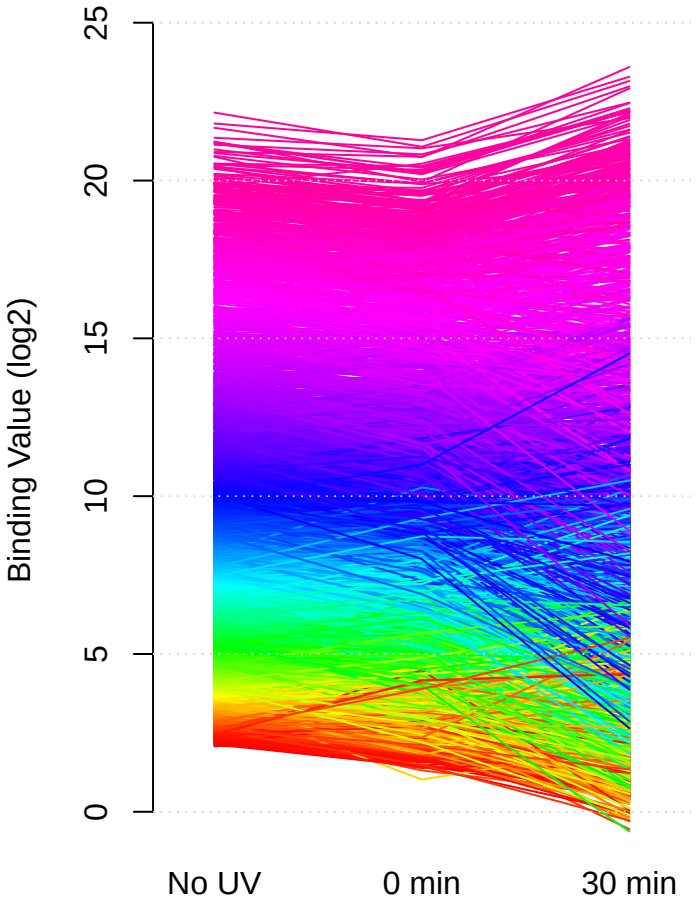


Figure 7.10: Rainbow plot of Abf1 peak changes: Representation of the changes in the peak heights between the untreated, 0 min after UV and 30 min after UV datasets. Each line represents one peak found in the untreated dataset and shows how it varies after 0 and 30 min. It can be seen that the 30 min dataset is very similar to the untreated dataset. There is some variation in the 0 min dataset, with many peak values increasing and decreasing by reasonably large amounts.

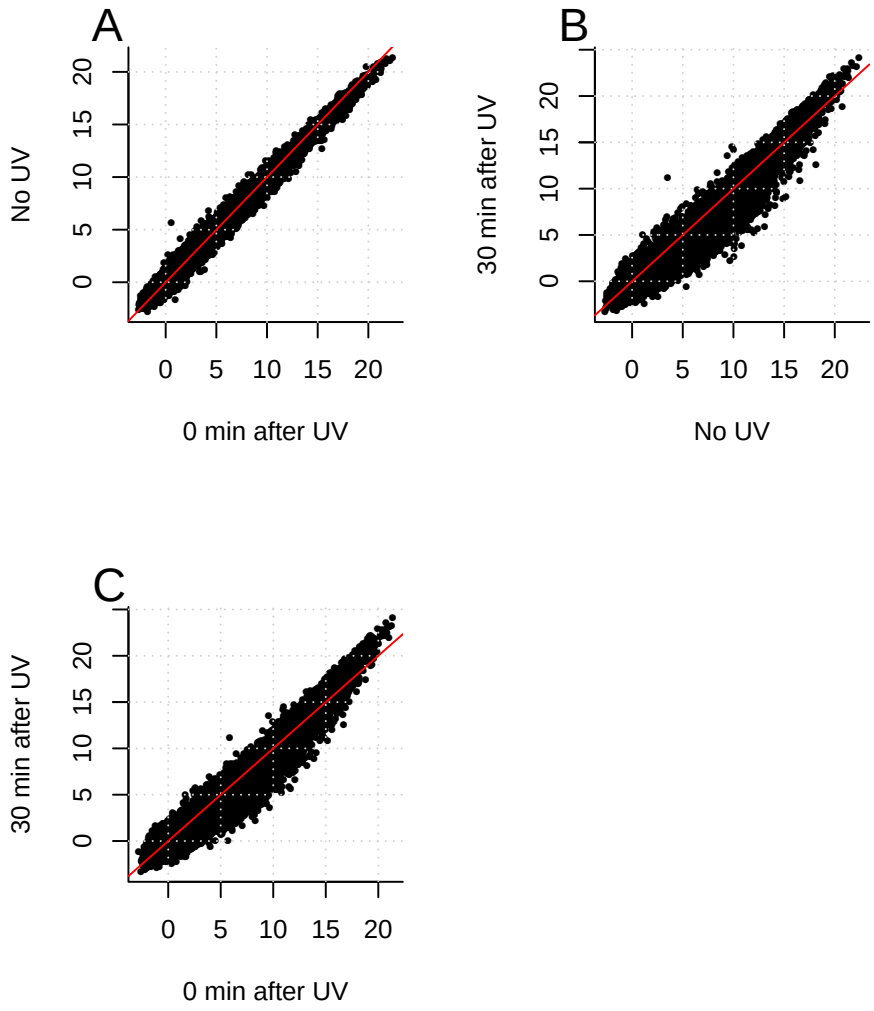


Figure 7.11: Scatter plots of Abf1 peak changes: The relationships between the peak heights in the untreated, 0 min after UV and 30 min after UV datasets. The red line shows $y = x$. The general trend is for lower values in the 30 min sample than the no UV or 0 min. The no UV and 0 min samples are very similar.

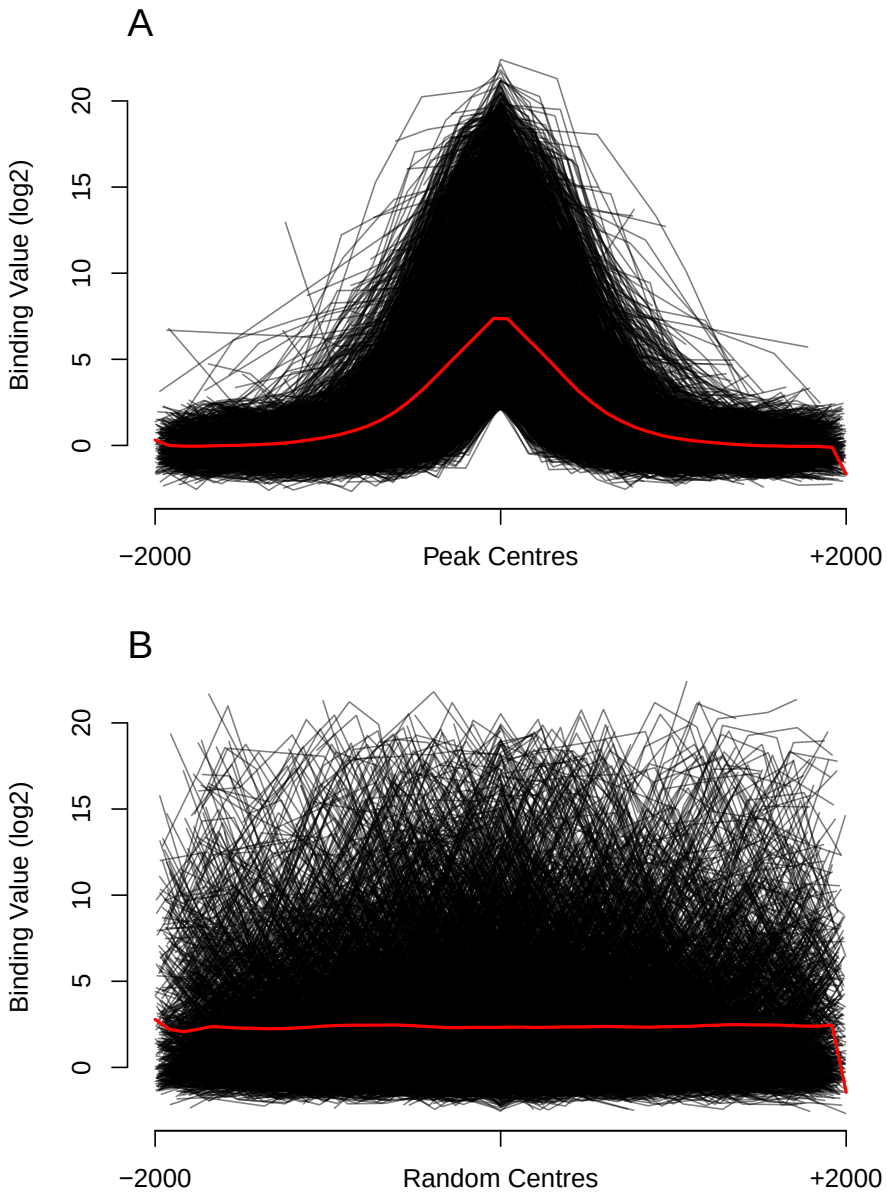


Figure 7.12: Abf1 peaks profile plot: All 4,369 detected peaks (**A**) and 4,369 random probes (**B**). All individual lines shown in black; trend line shown in red. Data are averages of the three replicates. The detected peaks are clearly visible, centred around the probe determined to be at the top of the peak. The random probes show no pattern.

(Rhode et al., 1992).

7.3.4 Genomic binding site locations

The previous analysis of this Abf1 binding data by Leadbitter (2011) showed that binding peaks occur preferentially in promoter regions of genes. This analysis was undertaken using the genome location information provided by Agilent in the results of the feature extraction files, which labels each probe according to its relation to its nearest ORF(s). This enabled a genome description of each peak to be extracted and analysed. Statistical analysis showed a significant over-representation of peaks in promoter regions, which was corroborated by visual analysis of the plots with respect to gene positions.

The analysis here improves on this with the use of the `findGene` function. This determines the genome position of each probe to return results similar to those listed above, but taken from up to date annotation data and alongside further positional information. This was used in conjunction with the `positionsPlot` function to create Figure 7.13, which shows the positions of the Abf1 binding sites relative to the start or end of the ORF they are determined to be located at or near. It can be seen by the large peak in the density plot around the ORF start position (0) that the majority of peaks occur at this location, with another significant proportion in downstream regions. Several peaks also occur inside ORFs, some several thousand nucleotides into the gene. It is possible that many of the genuine binding sites are in promoter regions but the limited resolution of ChIP-chip means that those close to the ORF start are recorded as falling withing ORFs. The bar chart shows ~75% of the peaks occur in promoters or inside genes. Around half of the promoter regions are divergent. The remaining ~25% fall in downstream regions. A smaller proportion of these regions (~25%) are divergent. It appears from this graph that most of the larger peaks fall in intergenic regions, and there is a trend for smaller peaks further into genes. There are still however a number of small peaks around the ORF start and end sites.

The results were filtered to examine the PBRs most likely to contain genuine Abf1 binding sites. Three strict criteria were used to extract only

the peaks most likely to be generated by strong Abf1 binding sites: peaks must be present at the same probe in all three replicate datasets, increasing the likelihood that the peak is centred around a genuine binding site; the height of the peak must be greater than 5, meaning it represents a region of reasonably high Abf1 binding; and the PBR length must be less than 200, meaning it comes from a region of reasonably high probe coverage where the data should be more reliable than regions covered by few probes. This left 160 peaks. The positional data of these is shown in Figure 7.14. As before, most peaks are located around ORF start sites with some at ORF end sites. The percentage of those inside ORFs has reduced while the percentage in promoter regions has increased, maintaining the same proportion of divergent at ~50%. The percentage in downstream regions has remained approximately the same. There are still some binding peaks several thousand nucleotides into ORFs. Nearly all of the tallest peaks occur in intergenic regions and once again the trend is for smaller peaks further into ORFs.

The method used to generate these results identifies the nearest ORF to a binding site. Many of the intergenic sites labelled as downstream may therefore be within promoter regions of other ORFs, the start sites of which are further away. To remove these potential discrepancies, the plots were created with only those sites labelled as divergent, for which this problem does not apply. These are shown in "positionsPlots.pdf" in the electronic appendix (see Page 367), for total and filtered PBRs respectively, which show a similar pattern to those in Figures 7.13 and 7.14. This shows that Abf1 binds throughout the genome, including inside ORFs and in non-promoter intergenic regions, which are likely related to its functions aside from transcriptional regulation.

7.3.5 Comparison with other datasets

Several previous studies have examined Abf1 binding genome wide. These are summarised in Table 7.3 and described in Section 1.4.4.

The results of this study were compared with these previously published data. It is immediately apparent that far more PBRs have been identified in

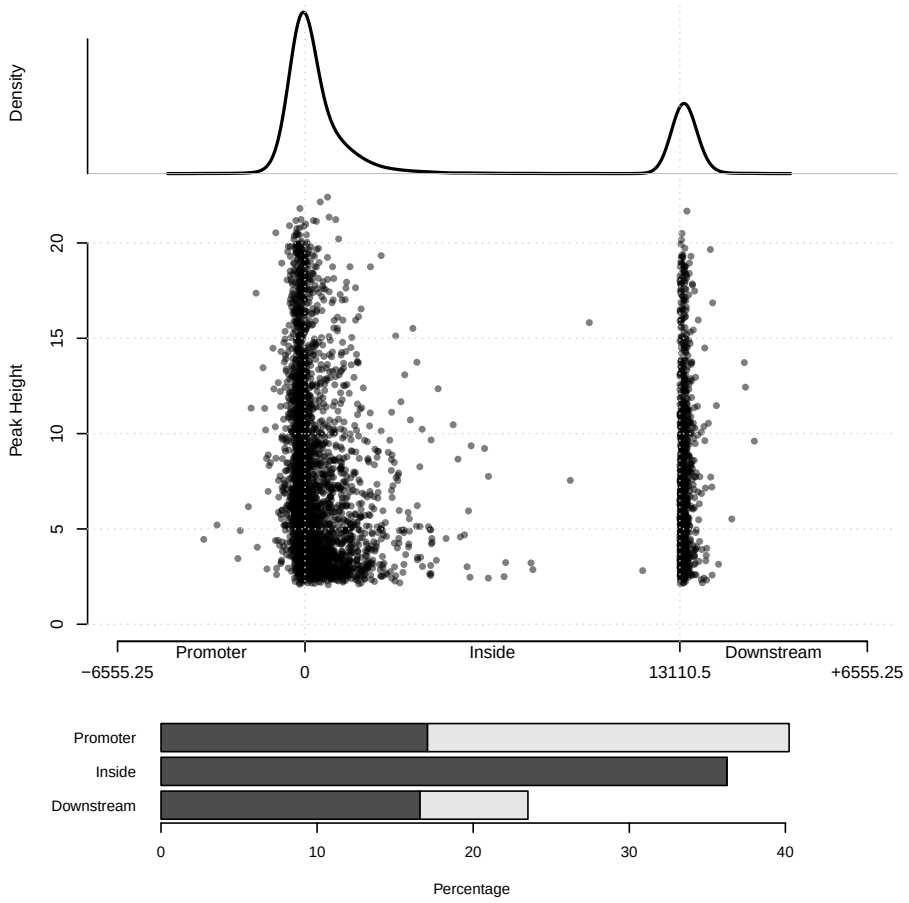


Figure 7.13: Abf1 binding site locations: All 4,369 Abf1 peak positions relative to their nearest gene. Top panel shows the density of the peak positions and percentage of peaks falling into the three categories of promoter, inside (an ORF) and downstream. The middle panel shows the peak positions plotted against the peak heights. Position 0 indicates the ORF start. Position 13,110 indicates the ORF end; this is the furthest into an ORF a peak is found. The bottom panel shows the percentage of probes falling into each category; light shading indicates divergent regions.

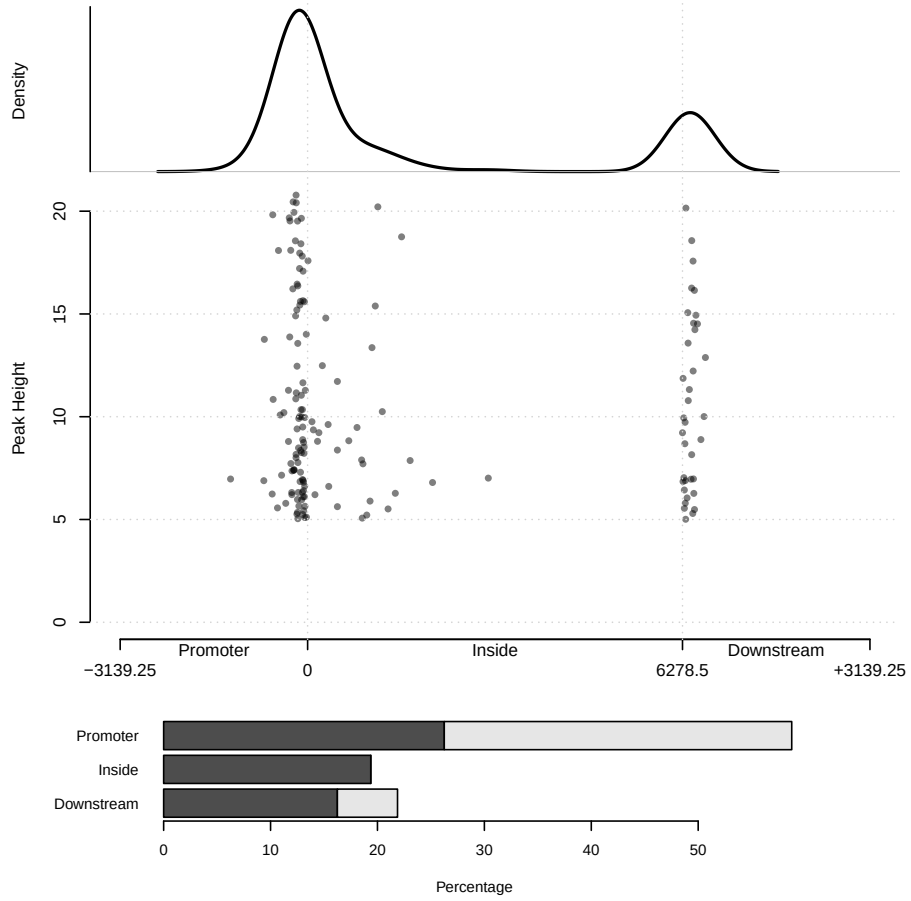


Figure 7.14: Filtered Abf1 binding site locations: The 124 filtered Abf1 peak positions relative to their nearest gene. Top panel shows the density of the peak positions and percentage of peaks falling into the three categories of promoter, inside (an ORF) and downstream. The middle panel shows the peak positions plotted against the peak heights. Position 0 indicates the ORF start. Position 6,278 indicates the ORF end; this is the furthest into an ORF a peak is found. The bottom panel shows the percentage of probes falling into each category; light shading indicates divergent regions.

this study than previously. This may be due to limited microarray resolution, especially so with earlier studies, and/or the application of strict statistical criteria causing the removal of many genuine binding sites. It is the case in some studies that results were filtered on the basis of the presence of the consensus binding sequence, potentially removing genuine binding sites lacking this. Direct comparisons of the results of the studies is difficult, as each presents their data in a different format. The earlier studies (Lee et al., 2002; Harbison et al., 2004; Schlecht et al., 2008) report only a gene name for each binding site, referring to the analysed promoters found to contain peaks. Comparisons of gene names can be difficult for many reasons. Genes can have multiple names (standard and systematic in the case of yeast) and so the genes being compared must be in the same format. Gene names can be changed in light of new findings, and so names reported in older studies may not now be correct. Additionally, genes can be very long, potentially with multiple binding sites associated with them, and so it may not be clear to which site a name refers.

These limitations notwithstanding, comparisons were made between the previous early datasets and the results from this study. The peaks found in this study were assigned a gene name using the `findGene` function, based on their closest ORF, which produced 3,589 unique gene names. Where necessary, gene names were converted to systematic names using the YeastMine tool on the SGD website (`http://yeastmine.yeastgenome.org/yeastmine/begin.do`). These systematic names were then compared between the datasets, with the resulting overlaps shown in Figure 7.15. Each dataset is represented in a different oval, and the number of gene names found in each possible overlapping category is shown. There is surprisingly little similarity between the datasets, with only 213 gene names appearing in all four. The overlaps between this study and those of Lee et al. (2002), Harbison et al. (2004) and Schlecht et al. (2008) are all 59%. The significance of the overlaps between the results of this study and the published three were determined by use of the hypergeometric distribution. This found that all overlaps are significant, with p values of 2.43×10^{-13} (Schlecht et al., 2008), 4.23×10^{-5} (Harbison et al., 2004) and 1.14×10^{-4} (Lee et al., 2002).

Study	Number of Abf1 binding sites found
Lee et al. (2002)	458
Harbison et al. (2004)	468
Schlecht et al. (2008)	1428
Ganapathi et al. (2011)	1035

Table 7.3: Abf1 binding sites found in previous studies.

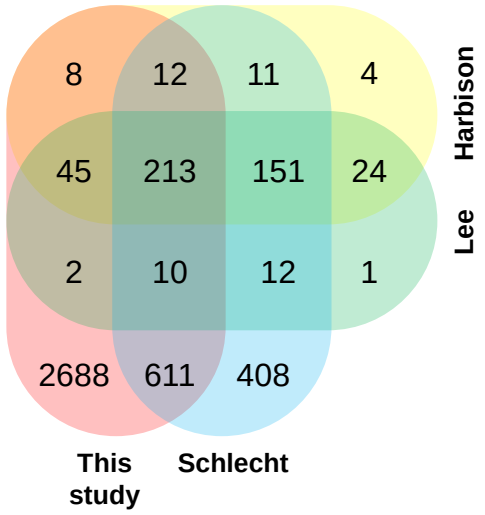


Figure 7.15: Previously published Abf1 comparisons: Diagram showing the relatedness between 3 previously published Abf1 binding datasets (Lee et al. (2002), Harbison et al. (2004) and Schlecht et al. (2008)) and this study. Note areas are not representative of dataset sizes.

The results of this study were next compared with those of Ganapathi et al. (2011), which report results as coordinates and provide all microarray data, allowing a higher resolution comparison to be made. All data were first compared visually with a scatter plot. Both sets of available data were plotted against the data from this study: actual microarray values (Figure 7.16A) and calculated p-values (Figure 7.16B). These plots indicate no correlations between either the microarray values or p-values, which is confirmed with Spearman's correlation coefficients of 0.05 and -0.04 respectively. In this investigation Abf1 binding sites were measured indirectly, by analysing nucleosome positions in the WT and temperature sensitive binding mutant *abf1-1*. Significant changes in nucleosome positions between the two were taken to be as a result of the loss of Abf1 binding and therefore indicative of Abf1 binding sites. As the datasets are so different they may not be directly comparable in this way which may explain the lack of any correlation.

The locations determined by the authors to be significant, and therefore representative of Abf1 binding, were next compared to the PBRs determined in this investigation. These locations are presented as coordinates, as are the PBRs, allowing direct comparisons between the two. Of the 1,035 regions provided by Ganapathi et al. (2011) in their list of significant sites, 446 (43%) overlap with probe positions on the Agilent microarray identified as binding sites here, which is statistically significant by the hyper geometric distribution with a p-value approaching zero. These points are highlighed in the scatter plots of Figure 7.16 with large black dots. These are spread randomly throughout the significant regions of the two datasets, with no association between the two. Many of the sites have high p-values as determined by Ganapathi et al. (2011) (Figure 7.16B), suggesting that the mapping of values from the Affymetrix to Agilent datasets carried out here may not be accurate. However, as with the previous analyses, there is a statistically significant overlap between the Abf1 binding sites identified in the two analyses.

It was shown in the investigation by Schlecht et al. (2008) that Abf1 binding sites can vary under different cellular conditions. It is also likely be the case that they vary between different strains of *S. cerevisiae*. Some of the

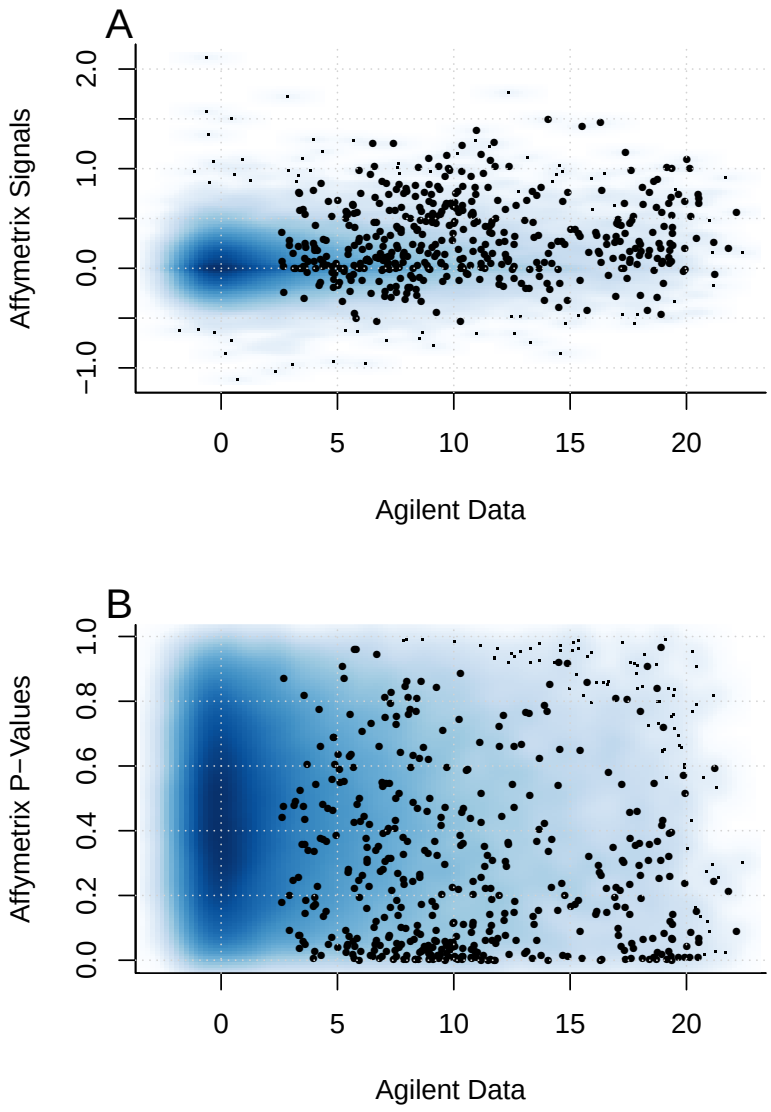


Figure 7.16: Ganapathi et. al.'s data: Scatter plots showing the relationship between the data of Ganapathi et al. (2011) (Affymetrix data) and this investigation (Agilent data). **A**: values from the microarrays. **B**: calculated p-values. Darker blue colours show more dense regions of points; isolated single points are shown with small black dots. Large black dots show regions determined to be statistically significant and therefore indicative of Abf1 binding sites.

discrepancies seen here may therefore be due to variations in the procedures between the different laboratories performing the investigations. This would mean that there are many more potential Abf1 binding sites in *S. cerevisiae* than have Abf1 bound at any given time.

7.3.6 Sequences at binding sites

An analysis of the yeast genome was undertaken to find the total number of occurrences of the consensus sequence RTCRYNNNNNACG. A script was written in `R` to search the UCSC sacCer3 (April 2011) genome assembly using tools from the `Biostrings` package (H et al.). This found 1,785 instances of the consensus; less than half the number of peaks found. Even taking into account the fact that some of the PBRs detected in this investigation may not represent genuine Abf1 binding sites, it is likely that not all of the Abf1 binding sites can contain the motif. An analysis of these regions was therefore undertaken to determine if they contain any additional, novel consensus sequences.

The sequences at PBRs were extracted, using the coordinates from the `peakList`, and examined for consensus sequences with BioProspector. The properties of the PBRs are shown in Table 7.4.

The sequences themselves were first examined for the presence of the consensus motif. This found 1,034 instances of the motif in 927 of the PBRs, detailed in Table 7.5. This shows that the majority of the PBRs (79%) do not contain the consensus motif. Additionally, a large proportion of the total motifs (42%) do not appear to be associated with an Abf1 binding peak. As discussed above, it is possible that some or all of these sites have the potential to be bound by Abf1, but are not under the conditions used here.

Position plots were created from the PBRs with and without the consensus motif, which show a similar pattern to those in Figures 7.13 and 7.14 (shown in "positionsPlots.pdf" in the electronic appendix; see Page 367). This shows that binding sites relative to ORFs are not associated with the presence of the consensus motif, or lack thereof.

BioProspector was first run with only the 927 sequences known to contain

Median length	Shortest length	Longest length
307	159	9472

Table 7.4: Abf1 PBR lengths: statistical summaries of all detected Abf1 PBRs.

Number of motifs	Number of sequences
0	3442
1	832
2	84
3	10
4	1

Table 7.5: Abf1 PBR motif counts: Numbers of detected Abf1 PBRs containing different numbers of the consensus motif sequence RTCRYNNNNNACG.

the consensus sequence, in order to test its performance. The top results of these runs are shown in Figure 7.17. The top logo shows the best result when not requiring the algorithm to find a motif in every sequence. The bottom logo shows the best result when the algorithm is required to find the motif in every sequence. These show that the program is reliable, since it is able to discern these known motifs from the remainder of the sequence, with motif scores of 4.444 and 4.415 respectively. There is little difference between the appearance of the two logos, showing that in this case the program works equally well with both settings. The remainder of the results (shown in "All BioProspector motifs.pdf" the electronic appendix; see Page 367) also all find the same motif.

The program was then run with all of the 4,369 PBRs to determine if any motifs are identifiable in these total sequences. The top logos are shown in Figure 7.18 and have scores of 4.346 and 2.162 respectively. This analysis did not find the previously identified consensus Abf1 binding site with either condition. All motifs found in this way contained variants of repeat TA regions, likely to be caused by TATA boxes present in promoter regions, the most common region of Abf1 binding. The remainder of the results (shown in "All BioProspector motifs.pdf" in the electronic appendix; see Page 367) show similar results, with no indication of any other motif(s) present in the sequences.

It is possible that other binding motifs may be present in the PBR sequences, but the number of additional sequences without them present prevents them being identified by this method. Of the 160 previously filtered PBRs thought most likely to represent genuine binding sites, 29% contain the motif (details in Table 7.6). All of these were therefore analysed for the presence of other motifs. The top sequence logos are shown in Figure 7.19 and have scores of 2.600 and 1.911 respectively. These show the consensus motif has been identified in this subset of sequences. The remainder of the results (shown in "All Bioprospector motifs.pdf" in the electronic appendix; see Page 367) repeatedly find the consensus, with no other motif identified. This suggested that even at the sites selected here to represent strong Abf1 binding, some other factor(s), independent of sequence, is(are) influencing

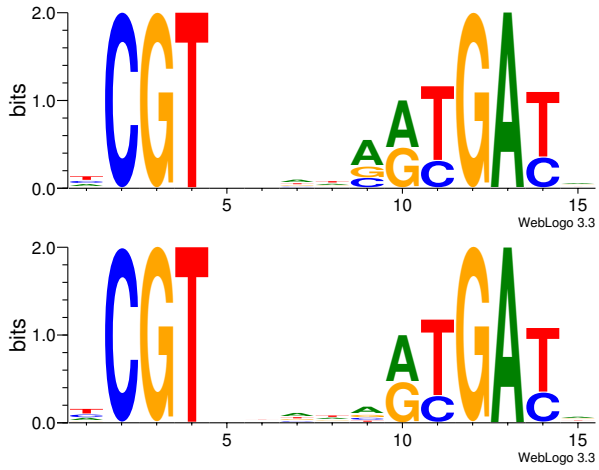


Figure 7.17: Sequence logos from PBRs containing the consensus: Results where motif is not (top) and is (bottom) required to be present in every sequence, created from 927 sequences known to contain the consensus Abf1 binding sequence RTCRYNNNNNACG.

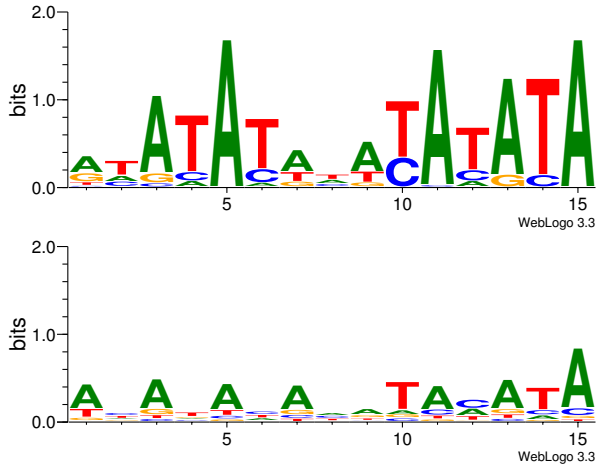


Figure 7.18: Sequence logos from all PBRs: Results where motif is not (top) and is (bottom) required to be present in every sequence, created from 4,369 sequences from every identified Abf1 PBR.

the binding.

Finally, sequences known not to contain the consensus motif were analysed, to see if any other motif(s) could be identified, without the presence of the currently known consensus to confound the results. All 3,442 PBRs were first analysed, and the logos (Figure 7.20 (scores of 4.165 and 2.312) and "All BioProspector motifs.pdf" in the electronic appendix) show similar results to those of the total sequences, in that only TATA box-like sequences appear to have been identified. The procedure was also carried out allowing a variable gap region between the two motif blocks, to allow any sequences of variable lengths to be identified, which with respect to the consensus motif would be in the format XRTCRY$[N]_{3-7}$ACGXXX, where N represents a non-conserved base and X a conserved base. (Figure 7.21 and "All BioProspector motifs.pdf" in the electronic appendix). This method produced the highest motif scores of all searches (5.063 and 4.439 respectively) but once again consists of conserved A and T bases, appearing to be relate to the TATA box.

The same analysis was carried out with the 113 filtered sequences without the consensus motif (Figures 7.22 and 7.23 and "All BioProspector motifs.pdf" in the electronic appendix). The fixed width sequence logos (scores of 2.305 and 1.866) show a high proportion of A/T nucleotides, but different from the previous TATA box-like sequences. There is no indication of a novel consensus. The sequence logos with variable gap regions (scores of 3.021 and 2.972) show more conservation at several sites, with many positions having high bit scores. However, these are not as high as the results from the PBRs containing the known motif, and although individual bases have high scores there do not appear to be any novel consensus motifs.

It was investigated whether there is a relationship between the number of motifs in a PBR and the height of the corresponding peak (Figure 7.24A). The sequences with more motifs tend to come from PBRs with higher peaks, as shown by the increasing positions of the box plots. However, there are still a large number of sites with no or few motifs that have high peaks. Adjusting for the length of the PBR (Figure 7.24B), to take account of the increasing chance of multiple motifs appearing in longer sequences, reduces

Number of motifs	Number of sequences
0	113
1	43
2	4

Table 7.6: Filtered Abf1 PBR motif counts: Numbers of filtered detected Abf1 PBRs containing different numbers of the consensus motif sequence RTCCTYNNNNNACG.

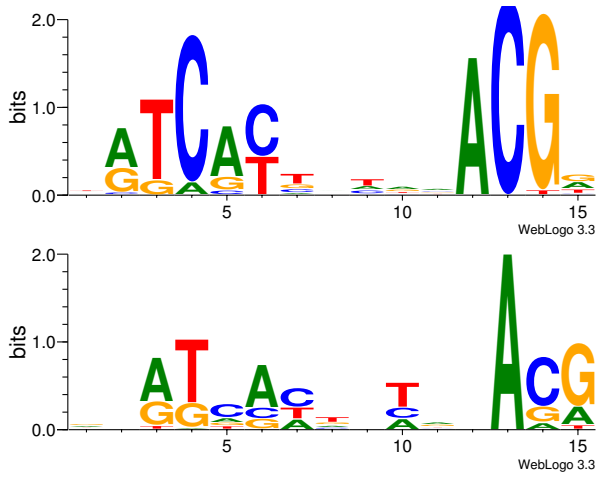


Figure 7.19: Sequence logos from filtered PBRs: Results where motif is not (top) and is (bottom) required to be present in every sequence, created from 160 filtered PBRs thought most likely to represent genuine Abf1 binding sites, showing the consensus Abf1 binding sequence RTCRYNNNNNACG has been identified.

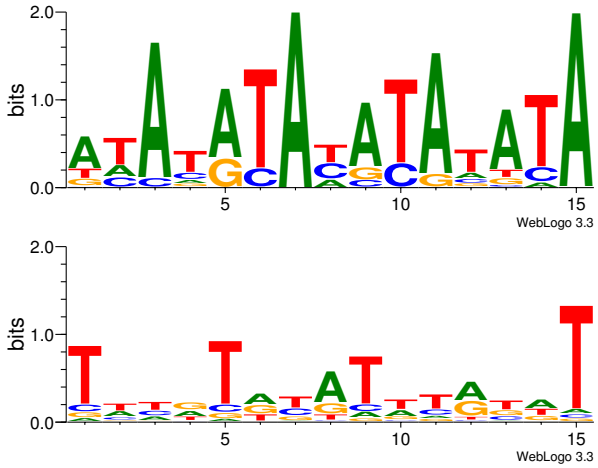


Figure 7.20: Sequence logos from PBRs without the consensus: Results where motif is not (top) and is (bottom) required to be present in every sequence, created from 3,442 sequences known not to contain the consensus Abf1 binding sequence RTCRYNNNNNACG.

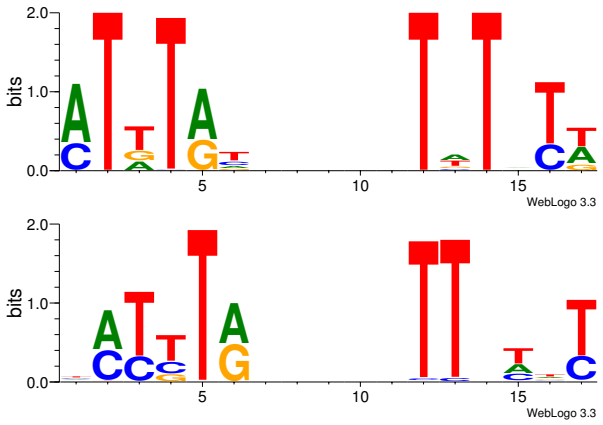


Figure 7.21: Sequence logos with variable gap regions from PBRs without the consensus: Results where motif is not (top) and is (bottom) required to be present in every sequence, allowing a gap region of 6–10 nt between blocks, created from 3,442 sequences known not to contain the consensus Abf1 binding sequence RTCRYNNNNNACG.

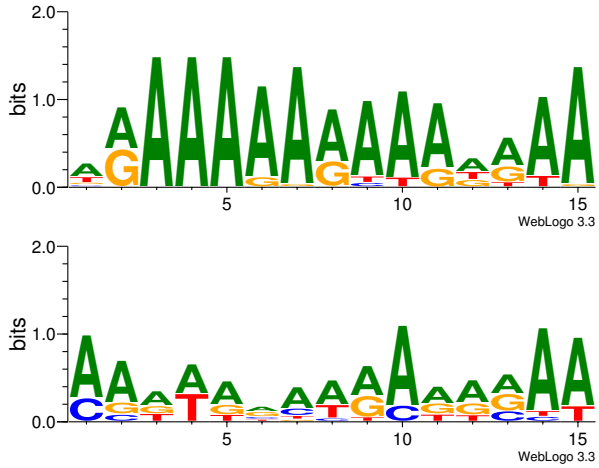


Figure 7.22: Sequence logos from filtered PBRs without the consensus: Results where motif is not (top) and is (bottom) required to be present in every sequence, created from 113 filtered sequences known not to contain the consensus Abf1 binding sequence RTCRYNNNNNACG.

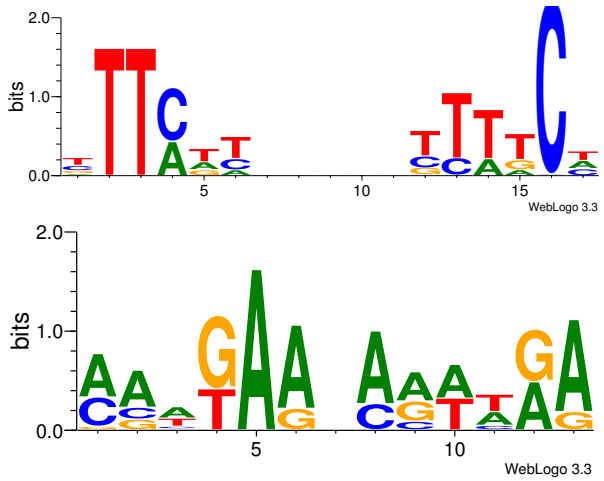


Figure 7.23: Sequence logos with variable gap regions from filtered PBRs without the consensus: Results where motif is not (top) and is (bottom) required to be present in every sequence, allowing a gap region of 6–10 nt between blocks, created from 113 filtered sequences known not to contain the consensus Abf1 binding sequence RTCRYNNNNNACG.

this relationship, but the general trend is still apparent. This shows that the PBRs containing motifs tend to contain peaks of greater height, but similar peaks also exist in the absence of the motif.

It is possible that the PBRs calculated by the peak detection script are too narrow, excluding some motif sites which contribute to Abf1 binding from the analysed sequence. It was therefore investigated whether or not motif sites close to, but maybe not within, PBRs have an influence on the peak. For each peak, the gap between the probe at its centre and the nearest motif site was calculated. These gap values are plotted against the peak heights in Figure 7.25. This plot shows that there are a large number of peaks, some with high binding values, far from a motif site. The 160 peaks determined by the strict filtering applied earlier are highlighted in red and can be seen to be scattered throughout the dataset, showing that many of these are also situated far from a motif site. Taken together these results show that many Abf1 PBRs do not contain the consensus motif, and this does not seem to have a detrimental effect on the ability of Abf1 to bind the region.

These results suggest that there are not other sequence motifs causing the binding of Abf1 in the regions examined. It also suggests that Abf1 binding is not influenced by another protein binding to its own consensus sequence, as it is likely this would have been identified. Previous studies have found Abf1 to bind to motifs similar, but not identical, to the consensus. Schroeder and Weil (1998), for example, showed Abf1 to bind to the sequence RTARYNNNNNACG, with the C changed to an A at the third position. Analysing the PBRs without the consensus sequence for this modified sequence found 263 with at least one instance. If Abf1 is able to bind to other sequences with a single base different from the consensus this will not be visually detectable in the logos. In this case no true consensus sequence would exist, because the sequence would be degenerate, in which every position could be any base, and so would be represented with an 'N'. The consensus provides the basis of this sequence. All possible combinations of the degenerate sequence with single base changes were searched for, both genome wide and within PBRs. Based on the fraction of the genome represented in PBRs, the expected number of these motifs in and out of PBRs

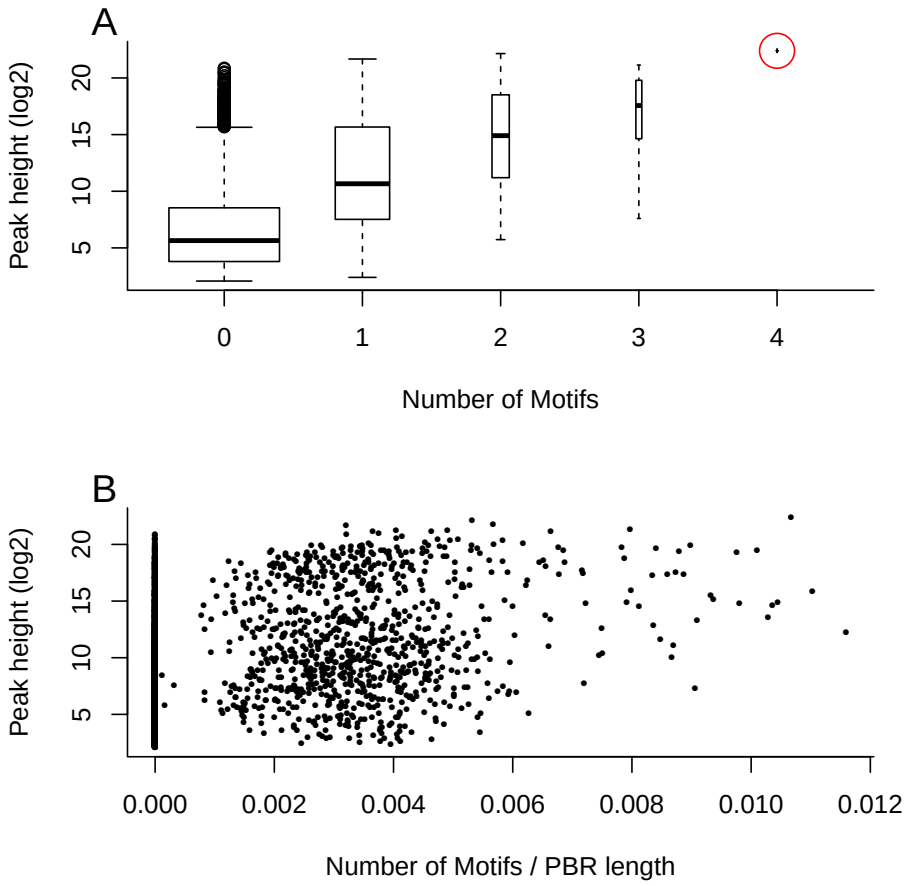


Figure 7.24: Abf1 peak heights and numbers of motifs: Box plots (A) showing the heights of peaks with different numbers of motifs present in their PBRs. The width of the box plot is proportional to the number of peaks present in the sample. For clarity the final box plot is highlighted with a red circle. The same data is shown with an adjustment the for the length of the PBR (B).

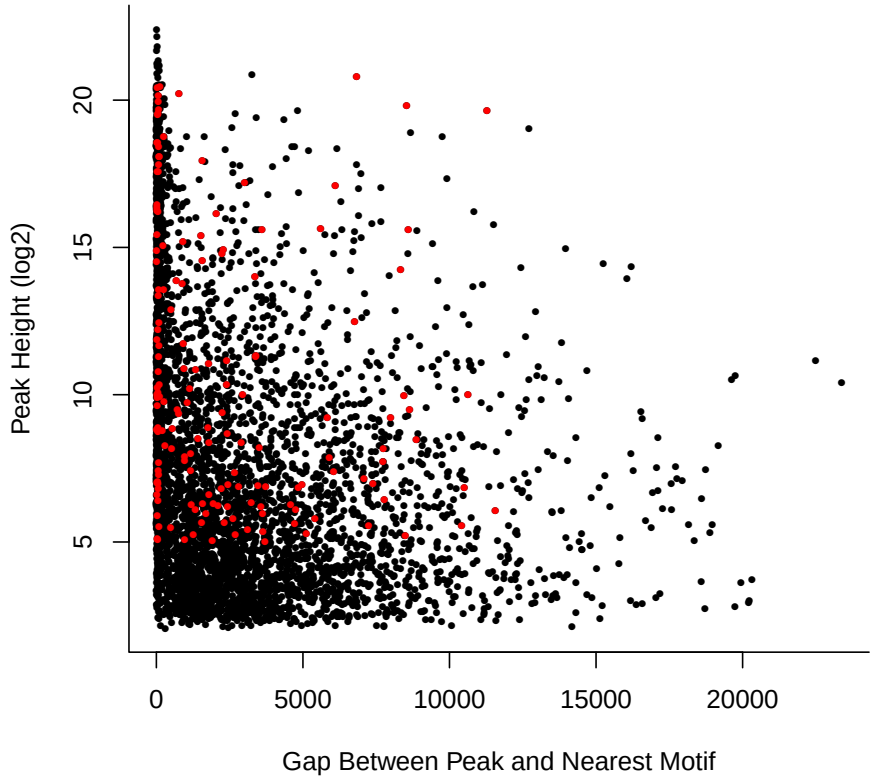


Figure 7.25: Gaps between Abf1 peaks and motifs: Scatter plot showing the size of gaps between Abf1 peaks and their nearest motif site against peak heights. Red points indicate 159 peaks identified by a strict filtering as the most likely to represent genuine binding sites.

was calculated. This was compared to the observed numbers with the Chi-squared test. This showed that all sequences, as well as the complete motif, are statistically significantly overrepresented in PBRs. By way of a comparison, random sequences in a similar format to the motif were generated, searched for and tested in the same way. These did not produce statistically significant results (data not shown).

7.4 Discussion

These datasets have previously been analysed to investigate the role of Abf1 in GG-NER (Leadbitter, 2011), where it was shown that Abf1 preferentially binds at promoters, the Rad16 protein colocalises to these sites and Rad16 dependent UV-induced H3Ac occurs at these sites. Preliminary data also suggest there may be changes in the DNA binding kinetics of Abf1 following UV irradiation, and a number of sites showed reduced binding levels 30 min after UV irradiation.

The investigation here used the same data to focus on Abf1 binding without any UV treatment, and has shown that there are many more Abf1 binding sites in the yeast genome than have previously been identified. Comparisons of this data with previously published genome wide investigations of Abf1 binding (Lee et al., 2002; Harbison et al., 2004; Schlecht et al., 2008; Ganapathi et al., 2011) have shown statistically significant overlaps, albeit through indirect comparisons, using either gene names, which may relate to different binding sites over the length of the gene, or by the mapping of an Affymetrix dataset onto the Agilent dataset created here, which may introduce errors into the data. There are still numerous sites found uniquely in the different investigations, suggesting that Abf1 binding is variable and dynamic, with thousands of potential binding sites throughout the genome, mainly in promoter regions but also within genes and in downstream regions, not all of which are bound by Abf1 in the conditions analysed in the different investigations.

Previous studies have shown that Abf1 is able to bind at sites other than those containing the consensus binding motif RTCRYNNNNNACG,

(Schroeder and Weil, 1998; Ganapathi et al., 2011, for example). Analysis of the sequences at binding sites in this investigation also showed that many do not contain this motif. In fact, the number of consensus sites in the whole genome is less than half of the number of peaks identified, clearly suggesting that Abf1 is able to bind at sites other than those containing this motif. No novel motif could be identified at the sites without the consensus motif. There may be many possible reasons for this. One is that a novel motif is present, but the BioProspector software used was unable to identify it. This is unlikely, as the software was shown to be able to correctly identify the current consensus sequence. The filtered sequences were also run through a second piece of software, MEME (`http://meme.nbcr.net/meme/cgi-bin/meme.cgi`; Bailey et al., 2009), which was not able to identify any other motif (results shown in "meme.pdf" in the accompanying electronic appendix; see Page 367). Another is that there is no novel motif, and a factor independent of sequence causes Abf1 binding. This may be related to the TATA box, whereby proteins binding at these sites, such as transcription factors, facilitate the binding of Abf1. Alternatively, Abf1 may be able to recognise and directly bind TATA-box like sequences. However, only ~20% of yeast promoters have been shown to contain a TATA box (Basehoar et al., 2004), which would not account for the numbers of Abf1 binding sites identified here. Finally, Abf1 may be binding to variations of the consensus sequence, as has been previously shown at a small number of sites. If Abf1 was able to bind at the consensus with a variation at any position, this may not show up in the sequence logos as there will not be any consistency accross the sequences. A statistical analysis of these sequences in the PBRs suggests that they are overrepresented and therefore may be having an influence on Abf1 binding. Visual analysis of the actual sequences detected by BioProspector does not indicate that this is the case in all PBRs, with many identified sequences varying from the consensus at multiple positions.

As previously discussed, ChIP-chip is a hypothesis generating technology whose results need to be confirmed by complementary techniques. Here the hypothesis is that Abf1 is able to bind to DNA at sites other than those

containing its known consensus binding motif RTCRYNNNNNACG. In order to gain evidence for this, binding at a number of these sites needs to be confirmed by other methods. These other methods would also allow a higher resolution analysis of the binding region to be determined, giving a better indication of the DNA sequence bound. There is evidence that some proteins are able to bind to degenerate binding sites bearing little resemblance to their consensus sequence (for example, Baumruker et al., 1988) and change their structural confirmation to fit different variations of their consensus sequence (for example, Phillips and Luisi, 2000). It may be that Abf1 has a similar propensity, which may account for its apparent ubiquitous binding profile and the inability here to identify a novel consensus binding sequence.

In the context of GG-NER, Abf1 is hypothesised to act to locate the Rad7/Rad16 complex at its binding sites in the absence of damage. The alternative, previously held view was that the complex only associated with DNA following the induction of damage. Having the complex bound to DNA at all times theoretically increases the speed at which the cell can respond to and repair damage. The complex may translocate from these binding sites either a short distance, having a distant effect, or a long distance, possibly to sites of damage, to initiate the repair process. Previous work suggests this is unidirectional. In this scenario, it is advantageous for Abf1 to be bound regularly throughout the genome, positioning the GG-NER machinery to act at any damage site. 20% of the genome is within 400 nt of an Abf1 binding site identified here which, not taking into account the orientation of the binding sites, means GG-NER over this proportion of the genome can be influenced by the previously identified 400 nt window of repair unidirectionally from an Abf1 binding site. It is possible that the domain of repair is longer than 400 nt at some Abf1 binding sites, increasing the proportion of the genome accessible to repair by the complex.

Chapter 8

Conclusions and future work

Tools

The bioinformatic tools presented in Chapter 3 provide a way of loading and analysing data in R. They are intended to provide a complete, integrated set of tools for these procedures, allowing basic analyses to be undertaken by people with little previous experience of bioinformatic data analysis or R, whilst maintaining the scope for more advanced analyses. They have been tested and used extensively in our laboratory, by people with a range of previous bioinformatic experience. The tools proved to be useful in generating results that have been used in publications, PhD theses, both complete and ongoing, and various other reports, posters and presentations. They have shown themselves to be simple enough to allow their use, along with the instructions, without significant additional support, thus achieving both of their key objectives.

The next step is to create a single R package containing these tools and present this as a publication, a draft manuscript for which has already been completed. Combining all the functions into a new package has many advantages over their current state, which are currently in separate files which have to be manually loaded into R. A single package can be easily downloaded and installed from, for example, Bioconductor, making the functions more accessible to the wider research community. Having this single entity makes maintaining and updating the functions easier and ensures that all

users have all of the most up to date functions at all times. This also allows new packages to use functions presented here in their processing, if required.

Normalisation

The normalisation method presented in Chapter 4 allows multiple ChIP-chip datasets from different conditions to be processed so as to allow comparisons to be made between them. Previously this was not possible and ChIP-chip investigations tended to either examine a single condition at a time, to determine where a factor of interest is present in a genome, or make limited comparisons between datasets from different conditions which could only distinguish the complete loss or gain of the factor at a given location from one condition to another. Normalising the data in the way described here allows relative changes in the level of the factor to be examined, opening up a new dimension of analysis.

Work is ongoing to further validate and refine the procedure with other quantitative technologies, such as Q-PCR. So far, two sets of data have been validated in this way, showing the normalisation to be robust. The Gcn5 protein binding dataset is currently undergoing this validation.

The alternative normalisation method, using spiked DNA samples of varying concentrations, is still under development. Several spike samples have now been created and these have been applied to several microarrays in order to optimise their concentrations and conditions (work primarily carried out by Dr. Katie Evans). Following this, samples of a range of optimal concentrations will begin to be analysed alongside real ChIP-chip data to develop a methodology for their use in normalisation.

Enrichment detection

The enrichment detection method presented in Chapter 5 was developed to fill a gap in the currently available enrichment detection software. There is no one method that is able to work with ChIP-chip data from any microarray platform, which the method presented here can do. It can also be used to detect either regions of enrichment or peaks, depending on the type of data

being analysed. It is fully integrated with the other functions presented here and has been optimised for fast performance, allowing for easy generation and presentation of results.

It has been tested with previously published spike datasets, developed for testing different aspects of ChIP-chip technology and data analysis. The calculated sensitivity and specificity values from these tests showed the method to outperform previously published enrichment detection methods tested on the same datasets. It was also tested with simulated ChIP-chip datasets, created here for this purpose, which also produced good sensitivity and specificity values.

Some problems with the available datasets for the testing of enrichment detection algorithms, such as that presented here, have been highlighted. There is only one published set of ChIP-chip data created for the purposes of testing and validating ChIP-chip procedures (Johnson et al., 2008), which was shown here to be inconsistent between repeats. A simulated dataset was also created here for testing the algorithm, but this will not necessarily accurately represent a genuine ChIP-chip dataset. It may be advantageous therefore to develop new datasets, where all sites of enrichment are known, possibly using spiked DNA samples, to enable the more accurate testing and refinement of enrichment detection algorithms.

Damage prediction

The damage prediction method presented in Chapter 6 has been invaluable in our laboratory for developing and testing a novel use of ChIP-chip technology, to detect DNA damage. Without the ability to compare microarray results with the predicted profiles, it would not be possible to determine whether or not the technology is working correctly, and displaying the results of damage, or displaying only random noise. The prediction has been used to validate microarray datasets examining damage created by UV radiation and the chemotherapeutic drugs cisplatin and oxaliplatin. The `R` function has been further optimised since its publication in 2011 and is now able to create these predictions in seconds, rather than the hours previously required.

Abf1 binding

The tools presented in this thesis have been used extensively to examine a range of ChIP-chip datasets generated in our laboratory. They were used in Chapter 7 to normalise, detect peaks, extract sequence information and create graphical displays from Abf1 binding datasets. This showed that Abf1 appears to bind at many more sites throughout the genome than have previously been identified. Many of these sites do not contain the current Abf1 binding consensus sequence RTCRYNNNNNACG.

As explained previously, ChIP-chip is a hypothesis generating technology which needs to be validated with other techniques. Some binding sites containing the consensus have previously been validated using PCR by Dr. Matthew Leadbitter. Some sites without the consensus also now need to be tested with a suitable technology, both to confirm the results of the microarrays and to determine the sequence(s) at which Abf1 is binding. This will help to answer the questions posed here as to whether there is a novel Abf1 binding motif or whether some other factor is able to influence Abf1's binding, which may have implications in all of the functions of Abf1. The information can also be incorporated into the current model of the mechanisms of GG-NER. Further tests can be carried out, both genome wide with the use of ChIP-chip and by other technologies, to determine if and how the various binding sites influence DNA repair.

The future

Several diverse but related projects in our laboratory are using the bioinformatic tools presented here to analyse data. These include clinical and industrial projects, as well as ongoing basic research. An investigation into the genome wide profile of damage induced by the platinating chemotherapeutic agents cisplatin and oxaliplatin, along with cellular responses to these drugs, is being undertaken to better understand the actions of the drugs, with a view being able to provide treatment programs tailored to individual patients based on their responses to these DNA damaging agents. This requires ways of analysing multiple large datasets, making comparisons between them

and extracting significant data, which might, for example, predict how a patient will respond to treatment. The tools described here allow analyses of this type. An investigation will begin shortly with industrial partners into understanding how histone deacetylase inhibitors, which affect the epigenome, lead to genotoxicity. This will also require the tools presented here, to be able to determine factors such as where the drugs localise, where they then act, how long they remain bound, what responses they induce and how they influence DNA repair rates. The previous methods of analysis, enabling only sites of binding to be identified, would not be sufficiently informative to allow these analyses. The basic research being undertaken in our laboratory is into the mechanisms of the GG-NER pathway. Multiple ChIP-chip datasets have been produced as part of this research, including histone acetylation, the chromatin binding of proteins involved in the repair process, and DNA damage itself. Further investigations will be carried out in this vein, in the context of newly generated genome wide datasets, to enable a model of the process to be made at the genome wide level, rather than at the short genomic regions that have been analysed previously. An analysis of this type has been published by Leadbitter (2011).

ChIP-seq (chromatin immunoprecipitation followed by next generation sequencing) is a developing technology which is likely to replace ChIP-chip in the coming years. ChIP-chip is currently more accessible to most laboratories as it is cheaper and less labour intensive. Some ChIP-seq data of DNA damage have been produced in our laboratory. The tools developed here can and have been adapted to analyse this type of data and can continue to be updated and modified to do so. There is an increasing focus on integrating different types of 'omics data produced on different platforms to be able to analyse them together (Cutts et al., 2012, provide a recent example). This is something that the tools presented here could be adapted to facilitate, given that they are already able to load data from any source.

It is anticipated that two publications will be created from the work presented here, in addition to the already published work from Chapter 6. The first will present the `R` scripts as a complete package, which will be made publicly available for others to use through a platform such as Bioconductor.

There is currently no other method available for the normalisation of ChIP-chip datasets from different conditions so as to allow relative comparisons to be made between them and so it is hoped that the tools presented here will allow the wider research community to perform this process on their own data to expand the current uses of ChIP-chip to produce novel results. The second will present the Abf1 binding data, which has implications in the variety of fields that Abf1 plays a role, including transcriptional regulation, genome partitioning, replication and GG-NER. Abf1 is an important general regulatory factor in the yeast genome and this updated, comprehensive analysis of genome wide binding sites will be significant for understanding how this protein functions in the cell.

Over the coming years bioinformatics will become increasingly important as larger and more detailed biological datasets are generated, which will require ever evolving computational tools for their analysis (Pepke et al., 2009; Ji, 2011). The tools presented in this thesis cannot be viewed as an end point, rather the foundations on which further tools can be based as new avenues of investigation are revealed. In this way the field of bioinformatics will increase both in depth, as the focus of investigations becomes ever more narrow, and breadth, as the number of types of investigation that can be carried out grows (Park, 2009). It is therefore important that the role of the bioinformatician is not simply to take over the analysis of data generated by biologists carrying out wet laboratory work, but to become actively involved in the design of experiments, having a say over what is investigated and how this is carried out, and any follow-up studies, in order to maximise the potential of any data generated. As genome wide technologies become more prevalent and next-generation sequencing becomes more accessible, computers should be viewed as another weapon in the biologists' arsenal, as essential as pipettes or gel rigs, rather than a specialist tool operated by people separate from those working at the bench. Thus the analysis of large datasets should become a routine and expected part of everyday experimental biology, with the distinction between 'biologist' and 'bioinformatician' becoming less and less defined.

Bibliography

M.E. Adriaens, M. Jaillard, L.M.T. Eijssen, C.D. Mayer, and C.T.A. Evelo. An evaluation of two-channel ChIP-on-chip and DNA methylation microarray normalization strategies. *BMC genomics*, 13(1):42, 2012.

Agilent Technologies Inc. Agilent SurePrint technology technical overview, January 2003. URL `http://www.chem.agilent.com/Library/technicaloverviews/Public/5988-8171en.pdf`.

Agilent Technologies Inc. Agilent microarray format technical overview (revision 1.0), January 2007. URL `http://www.chem.agilent.com/Library/technicaloverviews/Public/G4502-90001_MicroarrayFormat.pdf`.

Agilent Technologies Inc. eArray custom CGH microarrays FAQ, March 2010a. URL `http://www.chem.agilent.com/Library/brochures/5990-5520en_lo.pdf`.

Agilent Technologies Inc. Agilent Genomic Workbench Feature Extraction (version 10.10), September 2010b. URL `http://www.chem.agilent.com/Library/usermanuals/Public/G4460-90035_FeatureExtraction_QuickStart.pdf`.

B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell, fourth edition*. Garland Science, 2002.

D.G. Albertson, C. Collins, F. McCormick, J.W. Gray, et al. Chromosome aberrations in solid tumors. *Nature genetics*, 34(4):369–376, 2003.

S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

S. Andrews. ChIPMonk: software for viewing and analysing ChIP-on-chip data. *BMC Systems Biology*, 1(Suppl 1):P80, 2007.

G. Badis, E.T. Chan, H. van Bakel, L. Pena-Castillo, D. Tillo, K. Tsui, C.D. Carlson, A.J. Gossett, M.J. Hasinoff, C.L. Warren, et al. A library of yeast transcription factor motifs reveals a widespread function for Rsc3 in targeting nucleosome exclusion at promoters. *Molecular cell*, 32(6):878–887, 2008.

T.L. Bailey, M. Boden, F.A. Buske, M. Frith, C.E. Grant, L. Clementi, J. Ren, W.W. Li, and W.S. Noble. MEME SUITE: tools for motif discovery and searching. *Nucleic acids research*, 37(suppl 2):W202–W208, 2009.

D.D. Bang, R. Verhage, N. Goosen, J. Brouwer, P. van de Putte, et al. Molecular cloning of RAD16, a gene involved in differential repair in Saccharomyces cerevisiae. *Nucleic acids research*, 20(15):3925, 1992.

A.J. Bardwell, L. Bardwell, A.E. Tomkinson, and E.C. Friedberg. Specific cleavage of model recombination and repair intermediates by the yeast Rad1-Rad10 DNA endonuclease. *Science*, 265(5181):2082–2085, 1994.

C.L. Barrett, B.K. Cho, and B.O. Palsson. Sensitive and accurate identification of protein–DNA binding events in ChIP-chip assays using higher order derivative analysis. *Nucleic acids research*, 39(5):1656, 2011.

D.P. Bartel. MicroRNAs: genomics, biogenesis, mechanism, and function. *Cell*, 116(2):281–297, 2004.

A.D. Basehoar, S.J. Zanton, and B.F. Pugh. Identification and distinct regulation of yeast TATA box-containing genes. *Cell*, 116(5):699–709, 2004.

T. Baumruker, R. Sturm, and W. Herr. OBP100 binds remarkably degenerate octamer motifs through specific interactions with flanking sequences. *Genes & development*, 2(11):1400, 1988.

R. Beinoravičiūtė-Kellner, G. Lipps, and G. Krauss. In vitro selection of DNA binding sites for ABF1 protein from¡ i¿ Saccharomyces cerevisiae¡/i¿. *FEBS letters*, 579(20):4535–4540, 2005.

P.K. Bhatia, R.A. Verhage, J. Brouwer, and E.C. Friedberg. Molecular cloning and characterization of saccharomyces cerevisiae RAD28, the yeast homolog of the human Cockayne syndrome A (CSA) gene. *Journal of bacteriology*, 178(20):5977–5988, 1996.

M. Bieda, X. Xu, M.A. Singer, R. Green, and P.J. Farnham. Unbiased location analysis of E2F1-binding sites suggests a widespread role for E2F1 in the human genome. *Genome research*, 16(5):595, 2006.

G. Bindea, B. Mlecnik, WH Fridman, F. Pagès, and J. Galon. Natural immunity to cancer in humans. *Current opinion in immunology*, 22(2): 215, 2010.

B.M. Bolstad, R.A. Irizarry, M. Åstrand, and T.P. Speed. A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. *Bioinformatics*, 19(2):185, 2003.

A.J. Brookes. The essence of SNPs. *Gene*, 234(2):177–186, 1999.

AR Buchman and RD Kornberg. A yeast ARS-binding protein activates transcription synergistically in combination with other weak activating factors. *Molecular and cellular biology*, 10(3):887–897, 1990.

A.R. Buchman, W.J. Kimmerly, J. Rine, and R.D. Kornberg. Two DNA-binding factors recognize specific sequences at silencers, upstream activating sequences, autonomously replicating sequences, and telomeres in Saccharomyces cerevisiae. *Molecular and cellular biology*, 8(1):210–225, 1988.

M.J. Buck and J.D. Lieb. ChIP-chip: considerations for the design, analysis, and application of genome-wide chromatin immunoprecipitation experiments. *Genomics*, 83(3):349–360, 2004.

M.J. Buck, A.B. Nobel, and J.D. Lieb. ChIPOTle: a user-friendly tool for the analysis of ChIP-chip data. *Genome Biology*, 6(11):R97, 2005.

A. Butte. The use and analysis of microarray data. *Nature reviews drug discovery*, 1(12):951–960, 2002.

J. Cadet, N.E. Gentner, B. Rozga, and M.C. Paterson. Rapid quantitation of ultraviolet-induced thymine-containing dimers in human cell DNA by reversed-phase high-performance liquid chromatography. *Journal of Chromatography A*, 280:99–108, 1983.

M.F. Carey, C.L. Peterson, and S.T. Smale. Chromatin immunoprecipitation (ChIP). *Cold Spring Harbor Protocols*, 2009(9):pdb–prot5279, 2009.

S. Cawley, S. Bekiranov, H.H. Ng, P. Kapranov, E.A. Sekinger, D. Kampa, A. Piccolboni, V. Sementchenko, J. Cheng, A.J. Williams, et al. Unbiased mapping of transcription factor binding sites along human chromosomes 21 and 22 points to widespread regulation of noncoding RNAs. *Cell*, 116 (4):499–509, 2004.

M. Cesaroni, D. Cittaro, A. Brozzi, P.G. Pelicci, and L. Luzi. CARPET: a web-based package for the analysis of ChIP-chip and expression tiling data. *Bioinformatics*, 24(24):2918, 2008.

C.L. Chaffer and R.A. Weinberg. A perspective on cancer cell metastasis. *Science (New York, NY)*, 331(6024):1559, 2011.

D.I. Chasman, N.F. Lue, A.R. Buchman, J.W. LaPointe, Y. Lorch, and R.D. Kornberg. A yeast protein that influences the chromatin structure of uasg and functions as a powerful auxiliary gene activator. *Genes & development*, 4(4):503, 1990.

M. Chee, R. Yang, E. Hubbell, A. Berno, XC Huang, D. Stern, J. Winkler, DJ Lockhart, MS Morris, and SP Fodor. Accessing genetic information with high-density DNA arrays. *Science (New York, NY)*, 274(5287):610, 1996.

Y. Chen, C.A. Meyer, T. Liu, W. Li, J.S. Liu, X.S. Liu, et al. MM-ChIP enables integrative analysis of cross-platform and between-laboratory ChIP-chip or ChIP-seq data. *Genome Biol*, 12:R11, 2011.

G. Cho, J. Kim, H.M. Rho, and G. Jung. Structure-function analysis of the DNA binding domain of saccharomyces cerevisiae ABF1. *Nucleic acids research*, 23(15):2980–2987, 1995.

H. Choi, A.I. Nesvizhskii, D. Ghosh, and Z.S. Qin. Hierarchical hidden Markov model with application to joint analysis of ChIP-chip and ChIP-seq data. *Bioinformatics*, 25(14):1715–1721, 2009.

S.W. Chua, P. Vijayakumar, P.M. Nissom, C.Y. Yam, V.V.T. Wong, and H. Yang. A novel normalization method for effective removal of systematic variation in microarray data. *Nucleic acids research*, 34(5):e38–e38, 2006.

W.S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association*, pages 829–836, 1979.

A.R. Collins. The comet assay for DNA damage and repair. *Molecular biotechnology*, 26(3):249–261, 2004.

C.R. Contopoulou, V.E. Cook, and R.K. Mortimer. Analysis of DNA double strand breakage and repair using orthogonal field alternation gel electrophoresis. *Yeast*, 3(2):71–76, 1987.

C.T. Courcelle, K.H. Chow, A. Casey, and J. Courcelle. Nascent DNA processing by RecJ favors lesion repair over translesion synthesis at arrested replication forks in Escherichia coli. *Proceedings of the National Academy of Sciences*, 103(24):9154–9159, 2006.

D. Coverley, MK Kenny, M. Munn, WD Rupp, DP Lane, and RD Wood. Requirement for the replication protein SSB in human DNA excision repair. *Nature*, 349(6309):538, 1991.

F. Crick and J. Watson. A structure for deoxyribose nucleic acid. *Nature*, 171(737-738), 1953.

G.E. Crooks, G. Hon, J.M. Chandonia, and S.E. Brenner. WebLogo: a sequence logo generator. *Genome research*, 14(6):1188–1190, 2004.

R.J. Cutts, A.Z.D. Ullah, A. Sangaralingam, E. Gadaleta, N.R. Lemoine, and C. Chelala. O-miner: an integrative platform for automated analysis and mining of-omics data. *Nucleic Acids Research*, 40(W1):W560–W568, 2012.

W.L. De Laat, E. Appeldoorn, K. Sugasawa, E. Weterings, N.G.J. Jaspers, and J.H.J. Hoeijmakers. DNA-binding polarity of human replication protein A positions nucleases in nucleotide excision repair. *Genes & development*, 12(16):2598–2609, 1998.

F. Della Seta, I. Treich, JM Buhler, and A. Sentenac. Abf1 binding sites in yeast RNA polymerase genes. *Journal of Biological Chemistry*, 265(25): 15168–15175, 1990.

M.F. Denissenko, J.X. Chen, M. Tang, and G.P. Pfeifer. Cytosine methylation determines hot spots of DNA damage in the human P53 gene. *Proceedings of the National Academy of Sciences*, 94(8):3893, 1997.

M. Dizdaroglu. The use of capillary gas chromatographymass spectrometry for identification of radiation-induced DNA base damage and DNA baseamino acid cross-links. *Journal of Chromatography A*, 295:103–121, 1984.

J.H. Do, D. Choi, et al. Normalization of microarray data: single-labeled and dual-labeled arrays. *Molecules and cells*, 22(3):254, 2006.

B.A. Donahue, S. Yin, J.S. Taylor, D. Reines, and P.C. Hanawalt. Transcript cleavage by rna polymerase ii arrested by a cyclobutane pyrimidine dimer in the DNA template. *Proceedings of the National Academy of Sciences*, 91(18):8502, 1994.

T. Douki, S. Sauvaigo, F. Odin, J. Cadet, et al. Formation of the main UV-induced thymine dimeric lesions within isolated and cellular DNA as measured by high performance liquid chromatography-tandem mass spectrometry. *Journal of Biological Chemistry*, 275(16):11678–11685, 2000.

S. Durinck, Y. Moreau, A. Kasprzyk, S. Davis, B. De Moor, A. Brazma, and W. Huber. BioMart and Bioconductor: a powerful link between biological databases and microarray data analysis. *Bioinformatics*, 21(16):3439–3440, 2005.

AW Einerhand, W. Kos, W.C. Smart, A.J. Kal, H.F. Tabak, and T.G. Cooper. The upstream region of the FOX3 gene encoding peroxisomal 3-oxoacyl-coenzyme A thiolase in Saccharomyces cerevisiae contains ABF1- and replication protein A-binding sites that participate in its regulation by glucose repression. *Molecular and cellular biology*, 15(6):3405–3414, 1995.

M.M. Elkind and C.M. Chang-Liu. Repair of a DNA complex from x-irradiated Chinese hamster cells. *International Journal of Radiation Biology*, 22(1):75–90, 1972.

K. Erixon and G. Ahnström. Single-strand breaks in DNA during repair of UV-induced damage in normal human and xeroderma pigmentosum cells as determined by alkaline DNA unwinding and hydroxylapatite chromatography: Effects of hydroxyurea, 5-fluorodeoxyuridine and 1-β-d-arabinofuranosylcytosine on the kinetics of repair. *Mutation Research/-Fundamental and Molecular Mechanisms of Mutagenesis*, 59(2):257–271, 1979.

K.E Evans. *A genome wide study of histone H3 acetylation and its role in nucleotuide excision repair*. PhD thesis, Cardiff University School of Medicine, 2011.

P. Fardin, S. Moretti, B. Biasotti, A. Ricciardi, S. Bonassi, and L. Varesio. Normalization of low-density microarray using external spike-in controls: analysis of macrophage cell lines expression profile. *BMC genomics*, 8(1): 17, 2007.

C.R. Farthing, G. Ficz, R.K. Ng, C.F. Chan, S. Andrews, W. Dean, M. Hemberger, and W. Reik. Global mapping of DNA methylation in mouse promoters reveals epigenetic reprogramming of pluripotency genes. *PLoS genetics*, 4(6):e1000116, 2008.

W.J. Feaver, W. Huang, O. Gileadi, L. Myers, C.M. Gustafsson, R.D. Kornberg, and E.C. Friedberg. Subunit interactions in yeast transcription/repair factor TFIIH. *Journal of Biological Chemistry*, 275(8):5941–5946, 2000.

A.J. Fornace, K.W. Kohn, and H.E. Kann. DNA single-strand breaks during repair of UV damage in human fibroblasts and abnormalities of repair in xeroderma pigmentosum. *Proceedings of the National Academy of Sciences*, 73(1):39, 1976.

A.J. Fornace Jr. Measurement of M. luteus endonuclease-sensitive lesions by alkaline elution. *Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis*, 94(2):263–276, 1982.

G. Fourel, T. Miyake, P.A. Defossez, R. Li, and É. Gilson. General regulatory factors (grfs) as genome partitioners. *Journal of Biological Chemistry*, 277 (44):41736–41743, 2002.

E. C. Friedberg, G.C. Walker, and W. Siede. *DNA repair and mutagenesis, second edition*. ASM Press, 2006.

A. Fujita, J.R. Sato, L.O. Rodrigues, C.E. Ferreira, and M.C. Sogayar. Evaluating different methods of microarray data normalization. *BMC bioinformatics*, 7(1):469, 2006.

S. Fulda. Evasion of apoptosis as a cellular stress response in cancer. *International Journal of Cell Biology*, 2010, 2010.

D.J. Galas and A. Schmitz. DNAase footprinting a simple method for the detection of protein-DNA binding specificity. *Nucleic acids research*, 5(9): 3157–3170, 1978.

M. Ganapathi, M.J. Palumbo, S.A. Ansari, Q. He, K. Tsui, C. Nislow, and R.H. Morse. Extensive role of the general regulatory factors, Abf1 and Rap1, in determining genome-wide chromatin structure in budding yeast. *Nucleic acids research*, 39(6):2032–2044, 2011.

M.M. Garner and A. Revzin. A gel electrophoresis method for quantifying the binding of proteins to specific DNA regions: application to components of the escherichia coli lactose operon regulatory system. *Nucleic acids research*, 9(13):3047–3060, 1981.

Y. Gavrieli, Y. Sherman, and S.A. Ben-Sasson. Identification of programmed cell death in situ via specific labeling of nuclear DNA fragmentation. *The Journal of cell biology*, 119(3):493–501, 1992.

R. Gentleman. *Bioinformatics and computational biology solutions using R and Bioconductor*. Springer Verlag, 2005.

R.C. Gentleman, V.J. Carey, D.M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome biology*, 5(10):R80, 2004.

F.D. Gibbons, M. Proft, K. Struhl, and F.P. Roth. Chipper: discovering transcription-factor targets from chromatin immunoprecipitation microarrays using variance stabilization. *Genome Biology*, 6(11):R96, 2005.

E.F. Glynn, P.C. Megee, H.G. Yu, C. Mistrot, E. Unal, D.E. Koshland, J.L. DeRisi, and J.L. Gerton. Genome-wide mapping of the cohesin complex in the yeast Saccharomyces cerevisiae. *PLoS biology*, 2(9):e259, 2004.

A.J. Gossett and J.D. Lieb. Dna immunoprecipitation (DIP) for the determination of DNA-binding specificity. *Cold Spring Harbor Protocols*, 2008 (3):pdb–prot4972, 2010.

H.L. Govan III, Y. Valles-Ayoub, and J. Braun. Fine-mapping of DNA damage and repair in specific genomic segments. *Nucleic acids research*, 18 (13):3823–3830, 1990.

P.A. Grant, L. Duggan, J. Côté, S.M. Roberts, J.E. Brownell, R. Candau, R. Ohba, T. Owen-Hughes, C.D. Allis, F. Winston, et al. Yeast Gcn5 functions in two multisubunit complexes to acetylate nucleosomal histones: characterization of an Ada complex and the SAGA (Spt/Ada) complex. *Genes & development*, 11(13):1640–1650, 1997.

K.L. Gunderson, X.C. Huang, M.S. Morris, R.J. Lipshutz, D.J. Lockhart, and M.S. Chee. Mutation detection by ligation to complete n-mer DNA arrays. *Genome research*, 8(11):1142–1153, 1998.

S.N. Guzder, P. Sung, L. Prakash, and S. Prakash. Yeast DNA-repair gene RAD14 encodes a zinc metalloprotein with affinity for ultraviolet-damaged DNA. *Proceedings of the National Academy of Sciences*, 90(12):5433, 1993.

S.N. Guzder, P. Sung, L. Prakash, and S. Prakash. Nucleotide excision repair in yeast is mediated by sequential assembly of repair factors and not by a pre-assembled repairosome. *Journal of Biological Chemistry*, 271(15): 8903, 1996.

S.N. Guzder, P. Sung, L. Prakash, and S. Prakash. Yeast Rad7-Rad16 complex, specific for the nucleotide excision repair of the nontranscribed DNA strand, is an ATP-dependent DNA damage sensor. *Journal of Biological Chemistry*, 272(35):21665–21668, 1997.

S.N. Guzder, P. Sung, L. Prakash, and S. Prakash. Affinity of yeast nucleotide excision repair factor 2, consisting of the Rad4 and Rad23 proteins, for ultraviolet damaged DNA. *Journal of Biological Chemistry*, 273 (47):31541–31546, 1998.

Pages. H, P. Aboyoun, R. Gentleman, and S. DebRoy. *Biostrings: String objects representing biological sequences, and matching algorithms*. R package version 2.22.0.

Y. Habraken, P. Sung, L. Prakash, and S. Prakash. Yeast excision repair gene RAD2 encodes a single-stranded DNA endonuclease. *Nature*, 366(6453): 365, 1993.

Y. Habraken, P. Sung, S. Prakash, and L. Prakash. Transcription factor TFIIH and DNA endonuclease Rad2 constitute yeast nucleotide excision repair factor 3: implications for nucleotide excision repair and Cockayne syndrome. *Proceedings of the National Academy of Sciences*, 93(20):10718, 1996.

D. Hanahan and R.A. Weinberg. The hallmarks of cancer. *cell*, 100(1):57–70, 2000.

D. Hanahan and R.A. Weinberg. Hallmarks of cancer: the next generation. *Cell*, 144(5):646–674, 2011.

C.T. Harbison, D.B. Gordon, T.I. Lee, N.J. Rinaldi, K.D. Macisaac, T.W. Danford, N.M. Hannett, J.B. Tagne, D.B. Reynolds, J. Yoo, et al. Transcriptional regulatory code of a eukaryotic genome. *Nature*, 431(7004): 99–104, 2004.

J.J. Harrington and M.R. Lieber. Functional domains within FEN-1 and RAD2 define a family of structure-specific endonucleases: implications for nucleotide excision repair. *Genes & development*, 8(11):1344–1355, 1994.

C.C. Harris. p53 tumor suppressor gene: from the basic research laboratory to the clinic–an abridged historical perspective. *Carcinogenesis*, 17(6): 1187, 1996.

M.J. Heller. DNA microarray technology: devices, systems, and applications. *Annual review of biomedical engineering*, 4(1):129–153, 2002.

J.D. Hoheisel. Microarray technology: beyond transcript profiling and genotype analysis. *Nature reviews genetics*, 7(3):200–210, 2006.

P.P. Hsu and D.M. Sabatini. Cancer cell metabolism: Warburg and beyond. *Cell*, 134(5):703, 2008.

W. Huang, WJ Feaver, AE Tomkinson, and EC Friedberg. The N-degron protein degradation strategy for investigating the function of essential genes: requirement for replication protein A and proliferating cell nuclear antigen proteins for nucleotide excision repair in yeast extracts. *Mutation research*, 408(3):183, 1998.

W. Huber, A. Von Heydebreck, H. Sültmann, A. Poustka, and M. Vingron. Variance stabilization applied to microarray data calibration and to the quantification of differential expression. *Bioinformatics*, 18(suppl 1):S96–S104, 2002.

K. Ito, S. Inoue, Y. Hiraku, and S. Kawanishi. Mechanism of site-specific DNA damage induced by ozone. *Mutation Research/Genetic Toxicology and Environmental Mutagenesis*, 585(1-2):60–70, 2005.

H. Ji and W.H. Wong. TileMap: create chromosomal map of tiling array hybridizations. *Bioinformatics*, 21(18):3629–3636, 2005.

H. Ji, H. Jiang, W. Ma, D.S. Johnson, R.M. Myers, and W.H. Wong. An integrated software system for analyzing ChIP-chip and ChIP-seq data. *Nature biotechnology*, 26(11):1293–1300, 2008.

Hongkai Ji. Computational analysis of ChIP-chip data. *Handbook of Statistical Bioinformatics*, pages 257–282, 2011.

F. Johannes, R. Wardenaar, M. Colomé-Tatché, F. Mousson, P. De Graaf, M. Mokry, V. Guryev, H.T. Timmers, et al. Comparing genome-wide chromatin profiles using ChIP-chip or ChIP-seq. *Bioinformatics*, 26(8): 1000, 2010.

D.S. Johnson, W. Li, D.B. Gordon, A. Bhattacharjee, B. Curry, J. Ghosh, L. Brizuela, J.S. Carroll, M. Brown, P. Flicek, et al. Systematic evaluation of variability in ChIP-chip experiments using predefined DNA targets. *Genome research*, 18(3):393, 2008.

J.M. Johnson, J. Castle, P. Garrett-Engele, Z. Kan, P.M. Loerch, C.D. Armour, R. Santos, E.E. Schadt, R. Stoughton, and D.D. Shoemaker. Genome-wide survey of human alternative pre-mRNA splicing with exon junction microarrays. *Science*, 302(5653):2141–2144, 2003.

W.E. Johnson, W. Li, C.A. Meyer, R. Gottardo, J.S. Carroll, M. Brown, and X.S. Liu. Model-based analysis of tiling-arrays for ChIP-chip. *Proceedings of the National Academy of Sciences*, 103(33):12457, 2006.

C.J. Jones and R.D. Wood. Preferential binding of the xeroderma pigmentosum group A complementing protein to damaged DNA. *Biochemistry*, 32(45):12096–12104, 1993.

A. Karpikov, J. Rozowsky, and M. Gerstein. Tiling array data analysis: a multiscale approach using wavelets. *BMC bioinformatics*, 12(1):57, 2011.

W.J. Kent, C.W. Sugnet, T.S. Furey, K.M. Roskin, T.H. Pringle, A.M. Zahler, and D. Haussler. The human genome browser at UCSC. *Genome Research*, 12(6):996, 2002.

R.S. Kerbel. Tumor angiogenesis: past, present and the near future. *Carcinogenesis*, 21(3):505, 2000.

T.H. Kim, L.O. Barrera, M. Zheng, C. Qu, M.A. Singer, T.A. Richmond, Y. Wu, R.D. Green, and B. Ren. A high-resolution map of active promoters in the human genome. *Nature*, 436(7052):876–880, 2005.

K.W. Kohn and R.A. Grimek-Ewig. Alkaline elution analysis, a new approach to the study of DNA single-strand interruptions in cells. *Cancer Research*, 33(8):1849–1853, 1973.

H. Kohzaki, Y. Ito, and Y. Murakami. Context-dependent modulation of replication activity of Saccharomyces cerevisiae autonomously replicating sequences by transcription factors. *Molecular and cellular biology*, 19(11): 7428–7435, 1999.

M.F. Kramer and D.M. Coen. Enzymatic amplification of DNA by PCR: Standard procedures and optimization. *Current Protocols in Toxicology*, 2001.

S.K. Kurdistani, S. Tavazoie, and M. Grunstein. Mapping global histone acetylation patterns to gene expression. *Cell*, 117(6):721–733, 2004.

W.R. Lai, M.D. Johnson, R. Kucherlapati, and P.J. Park. Comparative analysis of algorithms for identifying amplifications and deletions in array CGH data. *Bioinformatics*, 21(19):3763, 2005.

M. Landfors, P. Philip, P. Rydén, and P. Stenberg. Normalization of high dimensional genomics data where the distribution of the altered variables is skewed. *PloS one*, 6(11):e27942, 2011.

R.F. Lascaris, E. De Groot, W.H. Mager, R.J. Planta, et al. Different roles for abf1p and a T-rich promoter element in nucleosome organization of the yeast RPS28A gene. *Nucleic acids research*, 28(6):1390–1396, 2000.

M. Leadbitter. *A genome-wide study to investigate the organisation of global genome nucleotide excision repair in Saccharomyces cerevisiae*. PhD thesis, Cardiff University School of Medicine, 2011.

T.I. Lee and R.A. Young. Transcription of eukaryotic protein-coding genes. *Annual review of genetics*, 34(1):77–137, 2000.

T.I. Lee, N.J. Rinaldi, F. Robert, D.T. Odom, Z. Bar-Joseph, G.K. Gerber, N.M. Hannett, C.T. Harbison, C.M. Thompson, I. Simon, et al. Transcriptional regulatory networks in Saccharomyces cerevisiae. *Science's STKE*, 298(5594):799, 2002.

W. Lee, D. Tillo, N. Bray, R.H. Morse, R.W. Davis, T.R. Hughes, and C. Nislow. A high-resolution atlas of nucleosome occupancy in yeast. *Nature genetics*, 39(10):1235–1244, 2007.

A.R. Lehmann and M. O'Driscoll. DNA repair: Disorders. *Encyclopedia of Life Sciences*, pages 1–9, 2010.

R. Li, S.Y. David, M. Tanaka, L. Zheng, S.L. Berger, and B. Stillman. Activation of chromosomal DNA replication insaccharomyces cerevisiae by Acidic Transcriptional Activation Domains. *Molecular and cellular biology*, 18(3):1296–1302, 1998.

S. Li and M.J. Smerdon. Rpb4 and Rpb9 mediate subpathways of transcription-coupled DNA repair in Saccharomyces cerevisiae. *The EMBO journal*, 21(21):5921–5929, 2002.

W. Li, C.A. Meyer, and X.S. Liu. A hidden Markov model for analyzing ChIP-chip experiments on genome tiling arrays and its application to p53 binding sequences. *Bioinformatics*, 21(suppl 1):i274, 2005.

X. Li, W. Gu, S. Mohan, and D.J. Baylink. DNA microarrays: their use and misuse. *Microcirculation*, 9(1):13–22, 2002.

C.G. Liu, G.A. Calin, B. Meloon, N. Gamliel, C. Sevignani, M. Ferracin, C.D. Dumitru, M. Shimizu, S. Zupo, M. Dono, et al. An oligonucleotide microchip for genome-wide microrna profiling in human and mouse tissues. *Proceedings of the National Academy of Sciences of the United States of America*, 101(26):9740, 2004.

X. Liu, D.L. Brutlag, and J.S. Liu. BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. In *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, page 127, 2001.

X. Liu, D.M. Noll, J.D. Lieb, and N.D. Clarke. DIP-chip: rapid and accurate determination of DNA-binding specificity. *Genome research*, 15(3):421–427, 2005.

X.S. Liu, D.L. Brutlag, and J.S. Liu. An algorithm for finding protein–DNA binding sites with applications to chromatin-immunoprecipitation microarray experiments. *Nature biotechnology*, 20(8):835–839, 2002.

C.M. Loch, N. Mosammaparast, T. Miyake, L.F. Pemberton, and R. Li. Functional and physical interactions between autonomously replicating sequence-binding factor 1 and the nuclear transport machinery. *Traffic*, 5(12):925–935, 2004.

D.J. Lockhart, H. Dong, M.C. Byrne, M.T. Follettie, M.V. Gallo, M.S. Chee, M. Mittmann, C. Wang, M. Kobayashi, H. Norton, et al. Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nature biotechnology*, 14(13):1675–1680, 1996.

S. Loo, P. Laurenson, M. Foss, A. Dillin, and J. Rine. Roles of ABF1, NPL3, and YCL54 in silencing in Saccharomyces cerevisiae. *Genetics*, 141(3):889, 1995.

I. Lucas, A. Palakodeti, Y. Jiang, D.J. Young, N. Jiang, A.A. Fernald, and M.M. Le Beau. High-throughput mapping of origins of replication in human cells. *EMBO reports*, 8(8):770–777, 2007.

Y. Marahrens and B. Stillman. A yeast chromosomal origin of DNA replication defined by multiple functional elements. *Science*, 255(5046):817–823, 1992.

J. Marmur and P. Doty. Determination of the base composition of deoxyribonucleic acid from its thermal denaturation temperature. *Journal of molecular biology*, 5:109, 1962.

C. Masutani, K. Sugasawa, J. Yanagisawa, T. Sonoyama, M. Ui, T. Enomoto, K. Takio, K. Tanaka, PJ Van der Spek, D. Bootsma, et al. Purification and cloning of a nucleotide excision repair complex involving the xeroderma pigmentosum group C protein and a human homologue of yeast RAD23. *The EMBO journal*, 13(8):1831, 1994.

M.N. McCall and R.A. Irizarry. Consolidated strategy for the analysis of microarray spike-in data. *Nucleic Acids Research*, 36(17):e108, 2008.

RA McGrath and RW Williams. Reconstruction in vivo of irradiated Escherichia coli deoxyribonucleic acid; the rejoining of broken pieces. *Nature*, 212(5061):534, 1966.

R.D. Miller, L. Prakash, and S. Prakash. Defective excision of pyrimidine dimers and interstrand DNA crosslinks in rad7 and rad23 mutants of Saccharomyces cerevisiae. *Molecular and General Genetics MGG*, 188(2):235–239, 1982.

D.L. Mitchell, C.A. Haipek, and J.M. Clarkson. (6-4) photoproducts are removed from the DNA of UV-irradiated mammalian cells more efficiently than cyclobutane pyrimidine dimers. *Mutation Research Letters*, 143(3): 109–112, 1985.

D.L. Mitchell, J. Jen, and J.E. Cleaver. Sequence specificity of cyclobutane pyrimidine dimers in DNA treated with solar (ultraviolet B) radiation. *Nucleic acids research*, 20(2):225–229, 1992.

T. Miyake, C.M. Loch, and R. Li. Identification of a multifunctional domain in autonomously replicating sequence-binding factor 1 required for transcriptional activation, DNA replication, and gene silencing. *Molecular and cellular biology*, 22(2):505–516, 2002.

T. Miyake, J. Reese, C.M. Loch, D.T. Auble, and R. Li. Genome-wide analysis of ARS (autonomously replicating sequence) binding factor 1 (abf1p)-mediated transcriptional regulation in Saccharomyces cerevisiae. *Journal of biological chemistry*, 279(33):34865, 2004.

P. Moore and BS Strauss. Sites of inhibition of in vitro DNA] synthesis in carcinogen-and UV-treated phi X174 DNA. *Nature*, 278(5705):664, 1979.

J. Morita, K. Ueda, S. Nanjo, and T. Komanol. Sequence specific damage of DNA induced by reducing sugars. *Nucleic acids research*, 13(2):449–458, 1985.

S. Mukherjee, M.F. Berger, G. Jona, X.S. Wang, D. Muzzey, M. Snyder, R.A. Young, and M.L. Bulyk. Rapid analysis of the DNA-binding specificities of transcription factors with DNA microarrays. *Nature genetics*, 36(12): 1331–1339, 2004.

M.H. Myllyperkiö, T.R.A. Koski, L.M. Vilpo, and J.A. Vilpo. Kinetics of excision repair of UV-induced DNA damage, measured using the comet assay. *Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis*, 448(1):1–9, 2000.

Y. Nakatsu, H. Asahina, E. Citterio, S. Rademakers, W. Vermeulen, S. Kamiuchi, J.P. Yeo, M.C. Khaw, M. Saijo, N. Kodo, et al. XAB2, a novel tetratricopeptide repeat protein involved in transcription-coupled DNA repair and transcription. *Journal of Biological Chemistry*, 275(45):34931, 2000.

R.K. Newton, M. Aardema, and J. Aubrecht. The utility of DNA microarrays for characterizing genotoxicity. *Environmental health perspectives*, 112(4): 420, 2004.

A. O'Donovan, A.A. Davies, J.G. Moggs, S.C. West, and R.D. Wood. XPG endonuclease makes the 3'incision in human DNA nucleotide excision repair. *Nature*, 371(6496):432, 1994.

OED. "microarray, n.". oed online. december 2011. oxford university press. 12 march 2012, December 2011. URL `http://www.oed.com/view/Entry/259001`.

S. Oikawa. Sequence-specific DNA damage by reactive oxygen species: Implications for carcinogenesis and aging. *Environmental health and preventive medicine*, 10(2):65–71, 2005.

P.L. Olive and J.P. Banáth. The comet assay: a method to measure DNA damage in individual cells. *Nature protocols*, 1(1):23–29, 2006.

P.L. Olive, J.P. Banáth, and R.E. Durand. Heterogeneity in radiation-induced DNA damage and repair in tumor and normal cells measured using the" comet" assay. *Radiation research*, 122(1):86–94, 1990.

H Pages. *BSgenome: Infrastructure for Biostrings-based genome data packages*, 2012a. R package version 1.22.0.

H. Pages. *BSgenome: Infrastructure for Biostrings-based genome data packages*, 2012b. R package version 1.22.0.

H. Pages, P. Aboyoun, R. Gentleman, S. DebRoy, and P.D.F.R.S.P.S. Alignments. String objects representing biological sequences, and matching algorithms. *Biostrings available at: http://www. bioconductor. org/packages/bioc/html/Biostrings. html*, 2009.

C.H. Park, T. Bessho, T. Matsunaga, and A. Sancar. Purification and characterization of the XPF-ERCC1 complex of human DNA repair excision nuclease. *Journal of Biological Chemistry*, 270(39):22657–22660, 1995.

H.J. Park, K. Zhang, Y. Ren, S. Nadji, N. Sinha, J.S. Taylor, and C.H. Kang. Crystal structure of a DNA decamer containing a cis-syn thymine dimer. *Proceedings of the National Academy of Sciences of the United States of America*, 99(25):15965, 2002.

P. J. Park. ChIP–seq: advantages and challenges of a maturing technology. *Nature Reviews Genetics*, 10(10):669–680, 2009.

S. Peng, A. Alekseyenko, E. Larschan, M. Kuroda, and P. Park. Normalization and experimental design for ChIP-chip data. *BMC bioinformatics*, 8 (1):219, 2007.

J. Penterman, D. Zilberman, J.H. Huh, T. Ballinger, S. Henikoff, and R.L. Fischer. Dna demethylation in the Arabidopsis genome. *Proceedings of the National Academy of Sciences*, 104(16):6752, 2007.

S. Pepke, B. Wold, and A. Mortazavi. Computation for ChIP-seq and RNA-seq studies. *Nature methods*, 6:S22–S32, 2009.

D. Perdiz, P. Gróf, M. Mezzina, O. Nikaido, E. Moustacchi, and E. Sage. Distribution and repair of bipyrimidine photoproducts in solar UV-irradiated mammalian cells. *Journal of Biological Chemistry*, 275(35):26732–26742, 2000.

AR Peterson, B.W. Fox, and M. Fox. Alkaline sucrose sedimentation studies of DNA from P388F lymphoma cells treated with difunctional alkylating agents. *Biochimica et Biophysica Acta (BBA)-Nucleic Acids and Protein Synthesis*, 299(3):385–396, 1973.

D. Pettijohn and P.C. Hanawalt. Evidence for repair replication of ultraviolet damage DNA in bacteria. *Journal of molecular biology*, 9:395–410, 1964.

K. Phillips and B. Luisi. The virtuoso of versatility: POU proteins that flex to fit. *Journal of molecular biology*, 302(5):1023–1039, 2000.

D. Pinkel and D.G. Albertson. Comparative genomic hybridization. *Annu. Rev. Genomics Hum. Genet.*, 6:331–354, 2005.

D. Pinkel, R. Segraves, D. Sudar, S. Clark, I. Poole, D. Kowbel, C. Collins, W.L. Kuo, C. Chen, Y. Zhai, et al. High resolution analysis of DNA copy number variation using comparative genomic hybridization to microarrays. *Nature genetics*, 20:207–211, 1998.

D.K. Pokholok, C.T. Harbison, S. Levine, M. Cole, N.M. Hannett, T.I. Lee, G.W. Bell, K. Walker, P.A. Rolfe, E. Herbolsheimer, et al. Genome-wide map of nucleosome acetylation and methylation in yeast. *Cell*, 122(4): 517–527, 2005.

R. Ponzielli, P.C. Boutros, S. Katz, A. Stojanova, A.P. Hanley, F. Khosravi, C. Bros, I. Jurisica, and L.Z. Penn. Optimization of experimental design parameters for high-throughput chromatin immunoprecipitation studies. *Nucleic acids research*, 36(21):e144, 2008.

L. Prakash. Defective thymine dimer excision in radiation-sensitive mutants rad10 and rad16 of saccharomyces cerevisiae. *Molecular and General Genetics MGG*, 152(2):125–128, 1977.

S. Prakash and L. Prakash. Nucleotide excision repair in yeast. *Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis*, 451(1-2):13–24, 2000.

L. Pray. DNA replication and causes of mutation. *Nature Education*, 1(1), 2008.

Y. Qi, A. Rolfe, K.D. MacIsaac, G.K. Gerber, D. Pokholok, J. Zeitlinger, T. Danford, R.D. Dowell, E. Fraenkel, T.S. Jaakkola, et al. High-resolution computational models of genome binding events. *Nature biotechnology*, 24 (8):963–970, 2006.

Y. Qiao, C. Wang, and L. Ma. Single cell DNA damage/repair assay using HaloChip. *Analytical Chemistry*, 2011.

J. Quackenbush et al. Microarray data normalization and transformation. *nature genetics*, 32(supp):496–501, 2002.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. URL `http://www.R-project.org/`. ISBN 3-900051-07-0.

B. Ramanathan and M.J. Smerdon. Enhanced DNA repair synthesis in hyperacetylated nucleosomes. *Journal of Biological Chemistry*, 264(19): 11026–11034, 1989.

R.E. Rasmussen and R.B. Painter. Evidence for repair of ultra-violet damaged deoxyribonucleic acid in cultured mammalian cells. *Nature*, 203:1360, 1964.

R. Redon, S. Ishikawa, K.R. Fitch, L. Feuk, G.H. Perry, T.D. Andrews, H. Fiegler, M.H. Shapero, A.R. Carson, W. Chen, et al. Global variation in copy number in the human genome. *Nature*, 444(7118):444–454, 2006.

S.H. Reed. Nucleotide excision repair in chromatin: Damage removal at the drop of a HAT. *DNA repair*, 2011.

S.H. Reed, M. Akiyama, B. Stillman, and E.C. Friedberg. Yeast autonomously replicating sequence binding factor is involved in nucleotide excision repair. *Genes & development*, 13(23):3052–3058, 1999.

B. Ren, F. Robert, J.J. Wyrick, O. Aparicio, E.G. Jennings, I. Simon, J. Zeitlinger, J. Schreiber, N. Hannett, E. Kanin, et al. Genome-wide location and function of DNA binding proteins. *Science*, 290(5500):2306, 2000.

P.R. Rhode, K.S. Sweder, K.F. Oegema, and J.L. Campbell. The gene encoding ARS-binding factor i is essential for the viability of yeast. *Genes & development*, 3(12a):1926–1939, 1989.

P.R. Rhode, S. Elsasser, and JL Campbell. Role of multifunctional autonomously replicating sequence binding factor 1 in the initiation of DNA replication and transcriptional control in saccharomyces cerevisiae. *Molecular and cellular biology*, 12(3):1064–1077, 1992.

C.J. Roberts, B. Nelson, M.J. Marton, R. Stoughton, M.R. Meyer, H.A. Bennett, Y.D. He, H. Dai, W.L. Walker, T.R. Hughes, et al. Signaling and circuitry of multiple MAPK pathways revealed by a matrix of global gene expression profiles. *Science*, 287(5454):873, 2000.

S.J. Russell, S.H. Reed, W. Huang, E.C. Friedberg, and S.A. Johnston. The 19S regulatory complex of the proteasome functions independently of proteolysis in nucleotide excision repair. *Molecular cell*, 3(6):687–695, 1999.

B. Rydberg, M. Löbrich, and P.K. Cooper. DNA double-strand breaks induced by high-energy neon and iron ions in human fibroblasts. i. pulsed-field gel electrophoresis method. *Radiation research*, 139(2):133–141, 1994.

P. Rydén, H. Andersson, M. Landfors, L. Näslund, B. Hartmanová, L. Noppa, and A. Sjöstedt. Evaluation of microarray data normalization procedures using spike-in experiments. *Bmc Bioinformatics*, 7(1):300, 2006.

P.C. Scacheri, G.E. Crawford, and S. Davis. [14] statistics for ChIP-chip and dnase hypersensitivity experiments on nimblegen arrays. *Methods in enzymology*, 411:270–282, 2006.

C. Schauber, L. Chen, P. Tongaonkar, I. Vega, D. Lambertson, W. Potts, and K. Madura. Rad23 links DNA repair to the ubiquitin/proteasome pathway. *Nature*, 391(6668):715–718, 1998.

M. Schena. *Microarray analysis*. John Wiley and Sons, 2003.

M. Schena, D. Shalon, R.W. Davis, and P.O. Brown. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, 270(5235):467, 1995.

U. Schlecht, I. Erb, P. Demougin, N. Robine, V. Borde, E. Van Nimwegen, A. Nicolas, and M. Primig. Genome-wide expression profiling, in vivo DNA binding analysis, and probabilistic motif prediction reveal novel abf1 target genes during fermentation, respiration, and sporulation in yeast. *Molecular Biology of the Cell*, 19(5):2193–2207, 2008.

S.C. Schroeder and P.A. Weil. Biochemical and genetic characterization of the dominant positive element driving transcription of the yeast TBP-encoding gene, SPT15. *Nucleic acids research*, 26(18):4186–4195, 1998.

A. Schulze and J. Downward. Navigating gene expression using microarraysa technology review. *Nature cell biology*, 3(8):E190–E195, 2001.

P. Sestili, C. Martinelli, and V. Stocchi. The fast halo assay: an improved method to quantify genomic DNA strand breakage at the single-cell level. *Mutation Research/Genetic Toxicology and Environmental Mutagenesis*, 607(2):205–214, 2006.

R.B. Setlow and W.L. Carrier. The disappearance of thymine dimers from DNA: an error-correcting mechanism. *Proceedings of the National Academy of Sciences of the United States of America*, 51(2):226, 1964.

J.W. Shay and S. Bacchetti. A survey of telomerase activity in human cancer. *European journal of cancer (Oxford, England: 1990)*, 33(5):787, 1997.

K. Shedden, W. Chen, R. Kuick, D. Ghosh, J. Macdonald, K. Cho, T. Giordano, S. Gruber, E. Fearon, J. Taylor, et al. Comparison of seven methods for producing Affymetrix expression scores based on false discovery rates in disease profiling data. *BMC bioinformatics*, 6(1):26, 2005.

D. Shore, D.J. Stillman, A.H. Brand, and K.A. Nasmyth. Identification of silencer binding proteins from yeast: possible roles in SIR control and DNA replication. *The EMBO journal*, 6(2):461, 1987.

M.J. Smerdon, S.Y. Lan, R.E. Calza, and R. Reeves. Sodium butyrate stimulates DNA repair in UV-irradiated normal and xeroderma pigmentosum human fibroblasts. *Journal of Biological Chemistry*, 257(22):13441–13447, 1982.

G. Smyth. Limma: linear models for microarray data. *Bioinformatics and computational biology solutions using R and Bioconductor*, pages 397–420, 2005.

G.K. Smyth and T. Speed. Normalization of cDNA microarray data. *Methods*, 31(4):265–273, 2003.

S. Solinas-Toldo, S. Lampel, S. Stilgenbauer, J. Nickolenko, A. Benner, H. Döhner, T. Cremer, and P. Lichter. Matrix-based comparative genomic hybridization: biochips to screen for genomic imbalances. *Genes, chromosomes and cancer*, 20(4):399–407, 1997.

J.S. Song, W.E. Johnson, X. Zhu, X. Zhang, W. Li, A.K. Manrai, J.S. Liu, R. Chen, and X.S. Liu. Model-based analysis of two-color arrays (MA2C). *Genome Biol*, 8(8):R178, 2007.

G. Speit and A. Hartmann. The comet assay: a sensitive genotoxicity test for the detection of DNA damage and repair. *Methods in molecular biology (Clifton, NJ)*, 314:275, 2006.

J. Sram, S.S. Sommer, and Q. Liu. Microarray-based DNA re-sequencing using 3'blocked primers. *Analytical biochemistry*, 374(1):41, 2008.

S.A. Stewart and R.A. Weinberg. Senescence: does it all happen at the ends? *Oncogene*, 21(4):627, 2002.

P. Sung, L. Prakash, and S. Prakash. Renaturation of DNA catalysed by yeast DNA repair and recombination protein RAD10. *Nature*, 355(6362): 743, 1992.

P. Sung, P. Reynolds, L. Prakash, and S. Prakash. Purification and characterization of the saccharomyces cerevisiae RAD1/RAD10 endonuclease. *The Journal of biological chemistry*, 268(35):26391, 1993.

Y. Takagi, H. Komori, W.H. Chang, A. Hudmon, H. Erdjument-Bromage, P. Tempst, and R.D. Kornberg. Revised subunit structure of yeast transcription factor IIH (TFIIH) and reconciliation with human TFIIH. *Journal of Biological Chemistry*, 278(45):43897–43900, 2003.

P.K. Tan, T.J. Downey, E.L. Spitznagel Jr, P. Xu, D. Fu, D.S. Dimitrov, R.A. Lempicki, B.M. Raaka, and M.C. Cam. Evaluation of gene expression measurements from commercial microarray platforms. *Nucleic acids research*, 31(19):5676, 2003.

K. Tanaka, N. Miura, I. Satokata, I. Miyamoto, MC Yoshida, Y. Satoh, S. Kondo, A. Yasui, H. Okayama, and Y. Okada. Analysis of a human DNA excision repair gene involved in group A xeroderma pigmentosum and containing a zinc-finger domain. *Nature*, 348(6296):73, 1990.

Y. Teng, S. Li, R. Waters, and S.H. Reed. Excision repair at the level of the nucleotide in the saccharomyces cerevisiae MFA2 gene: mapping of where enhanced repair in the transcribed strand begins or ends and identification of only a partial rad16 requisite for repairing upstream control sequences1. *Journal of molecular biology*, 267(2):324–337, 1997.

Y. Teng, H. Liu, H.W. Gill, Y. Yu, R. Waters, and S.H. Reed. Saccharomyces cerevisiae Rad16 mediates ultraviolet-dependent histone H3 acetylation required for efficient global genome nucleotide-excision repair. *EMBO reports*, 9(1):97–102, 2007.

Y. Teng, M. Bennett, K.E. Evans, H. Zhuang-Jackson, A. Higgs, S.H. Reed, and R. Waters. A novel method for the genome-wide high resolution analysis of DNA damage. *Nucleic Acids Research*, 39(2):e10, 2011.

J. Toedling and W. Huber. Analyzing ChIP-chip data using Bioconductor. *PLoS Computational Biology*, 4(11):e1000227, 2008.

J. Toedling, O. Sklyar, and W. Huber. Ringo–an R/Bioconductor package for analyzing ChIP-chip readouts. *BMC bioinformatics*, 8(1):221, 2007.

S. Tornaletti and P.C. Hanawalt. Effect of DNA lesions on transcription elongation. *Biochimie*, 81(1-2):139–146, 1999.

S. Tornaletti, D. Rozek, GP Pfeifer, et al. The distribution of UV photoproducts along the human p53 gene and its relation to mutations in skin cancer. *Oncogene*, 8(8):2051, 1993.

J.J. Truglio, D.L. Croteau, B. Van Houten, and C. Kisker. Prokaryotic nucleotide excision repair: The UvrABC system. *Chemical Reviews*, 106(2): 233–252, 2006.

C. Tuerk and L. Gold. Systematic evolution of ligands by exponential enrichment: RNA ligands to bacteriophage t4 DNA polymerase. *Science*, 249 (4968):505–510, 1990.

K. Ueda, S. Kobayashi, J. Morita, and T. Komano. Site-specific DNA damage caused by lipid peroxidation products. *Biochimica et Biophysica Acta (BBA)-Gene Structure and Expression*, 824(4):341–348, 1985.

M.P.J. van der Loo. *extremevalues, an R package for outlier detection in univariate data*, 2010. URL `http://www.cran.R-project.org,http://www.markvanderloo.eu`. R package version 2.0.

A.J. van Gool, R. Verhage, SM Swagemakers, P. van de Putte, J. Brouwer, C. Troelstra, D. Bootsma, and JH Hoeijmakers. RAD26, the functional S. cerevisiae homolog of the Cockayne syndrome B gene ERCC6. *The EMBO journal*, 13(22):5361, 1994.

R. Verhage, A.M. Zeeman, N. de Groot, F. Gleig, D.D. Bang, P. Van De Putte, and J. Brouwer. The RAD7 and RAD16 genes, which are essential for pyrimidine dimer removal from the silent mating type loci, are also required for repair of the nontranscribed strand of an active gene in Saccharomyces cerevisiae. *Molecular and cellular biology*, 14(9):6135–6142, 1994.

SS Walker, SC Francesconi, BK Tye, and S. Eisenberg. The OBF1 protein and its DNA-binding site are important for the function of an autonomously replicating sequence in Saccharomyces cerevisiae. *Molecular and cellular biology*, 9(7):2914–2921, 1989.

S.S. Walker, S.C. Francesconi, and S. Eisenberg. A DNA replication enhancer in Saccharomyces cerevisiae. *Proceedings of the National Academy of Sciences*, 87(12):4665, 1990.

J. Wang, G. Rivas, M. Ozsoz, D.H. Grant, X. Cai, and C. Parrado. Microfabricated electrochemical sensor for the detection of radiation-induced DNA damage. *Analytical chemistry*, 69(7):1457–1460, 1997.

J.F. Watkins, P. Sung, L. Prakash, and S. Prakash. The Saccharomyces cerevisiae DNA repair gene RAD23 encodes a nuclear protein containing a ubiquitin-like domain required for biological function. *Molecular and cellular biology*, 13(12):7757–7765, 1993.

J. D. Watson, T. A. Baker, S. P. Bell, A. Gann, M. Levine, and R. Losick. *Molecular Biology of the Gene, Fifth Edition.* Benjamin Cummings, 5 edition, December 2003. ISBN 080534635X.

M. Weber, J.J. Davies, D. Wittig, E.J. Oakeley, M. Haase, W.L. Lam, and D. Schübeler. Chromosome-wide and promoter-specific analyses identify sites of differential DNA methylation in normal and transformed human cells. *Nature genetics*, 37(8):853–862, 2005.

B.G. Wilson and C.W.M. Roberts. SWI/SNF nucleosome remodellers and cancer. *Nature Reviews Cancer*, 11(7):481–492, 2011.

S.M. Wolf and P. Vouros. Application of capillary liquid chromatography coupled with tandem mass spectrometric methods to the rapid screening of adducts formed by the reaction of N-acetoxy-N-acetyl-2-aminofluorene with calf DNA. *Chemical research in toxicology*, 7(1):82–88, 1994.

D.K. Wood, D.M. Weingeist, S.N. Bhatia, and B.P. Engelward. Single cell trapping and DNA damage analysis using microwell arrays. *Proceedings of the National Academy of Sciences*, 107(22):10008, 2010.

H. Wu and H. Ji. JAMIE: joint analysis of multiple ChIP-chip experiments. *Bioinformatics*, 26(15):1864, 2010.

W. Yang. Structure and mechanism for DNA lesion recognition. *Cell research*, 18(1):184–197, 2007.

A. Yarragudi, L.W. Parfrey, and R.H. Morse. Genome-wide analysis of transcriptional dependence and probable target sites for Abf1 and Rap1 in saccharomyces cerevisiae. *Nucleic acids research*, 35(1):193–202, 2007.

S. Yu, T. Owen-Hughes, E.C. Friedberg, R. Waters, and S.H. Reed. The yeast Rad7/Rad16/Abf1 complex generates superhelical torsion in DNA that is required for nucleotide excision repair. *DNA repair*, 3(3):277–287, 2004.

S. Yu, J.B. Smirnova, E.C. Friedberg, B. Stillman, M. Akiyama, T. Owen-Hughes, R. Waters, and S.H. Reed. ABF1-binding sites promote efficient global genome nucleotide excision repair. *Journal of Biological Chemistry*, 284(2):966–973, 2009.

S. Yu, Y. Teng, R. Waters, and S.H. Reed. How chromatin is remodelled during DNA repair of UV-induced DNA damage in Saccharomyces cerevisiae. *PLoS genetics*, 7(6):e1002124, 2011.

Y. Yu, Y. Teng, H. Liu, S.H. Reed, and R. Waters. UV irradiation stimulates histone acetylation and chromatin remodeling at a repressed yeast locus. *Proceedings of the National Academy of Sciences of the United States of America*, 102(24):8650, 2005.

Y. Zhang. Poisson approximation for significance in genome-wide ChIP-chip tiling arrays. *Bioinformatics*, 24(24):2825, 2008.

Z.D. Zhang, J. Rozowsky, HY Lam, J. Du, M. Snyder, M. Gerstein, et al. Tilescope: online analysis pipeline for high-density tiling microarray data. *Genome Biol*, 8(5):R81, 2007.

M. Zheng, L.O. Barrera, B. Ren, and Y.N. Wu. ChIP-chip: Data, model, and analysis. *Biometrics*, 63(3):787–796, 2007.

L. Zhou and J.F. Rusling. Detection of chemically induced DNA damage in layered films by catalytic square wave voltammetry using Ru (Bpy) 32+. *Analytical chemistry*, 73(20):4780–4786, 2001.

A. Zotter, M.S. Luijsterburg, D.O. Warmerdam, S. Ibrahim, A. Nigg, W.A. Van Cappellen, J.H.J. Hoeijmakers, R. Van Driel, W. Vermeulen, and A.B. Houtsmuller. Recruitment of the nucleotide excision repair endonuclease XPG to sites of UV-induced DNA damage depends on functional TFIIH. *Molecular and cellular biology*, 26(23):8868–8879, 2006.

Appendix A

Electronic Appendix Structure

- DVD
  - Chapter 3 - R scripts
    - instructions.pdf ..... See Section 3.2
    - R scripts
      - ..... All R script files
  - Chapter 4 - Normalisation
    - Gcn5 binding microarray datasets
      - ..... All Gcn5 binding data files
    - H3Ac microarray datasets
      - ..... All H3Ac data files
    - Spike probe information.pdf ..... See Section 4.4.1
    - Kernel density estimates.pdf ..... See Section 4.2.4.2
    - Q-PCR probes.pdf ..... See Section 4.3.1
  - Chapter 5 - Enrichment Detection
    - Enriched regions (averaged).pdf ..... See Section 5.3.2.4
    - Enriched regions (combined).pdf ..... See Section 5.3.2.4
    - key.txt ..... Information on the plots
  - Chapter 6 - CPDs
    - CPD microarray datasets
      - ..... All CPD data files
    - CPD predicted and actual profile.pdf ..... See Section 6.6
    - Cisplatin predicted and actual profile.pdf .... See Section 6.6.1
    - key.txt ..... Information on the plots
  - Chapter 7 - Abf1
    - Abf1 microarray datasets
      - check.pdf ..... See Section 7.3.1
      - ..... All Abf1 binding data files
    - Sequences at detected PBRs.fasta ..... See Section 7.2.7
    - positionPlots.pdf ..... See Section 7.3.4
    - All Bioprospector motifs.pdf ..... See Section 7.3.6
    - meme.pdf ..... See Section 7.4