Next-generation control strategy for biomanufacture

G. Lorenzon

Supervisors – Nigel Scrutton, Sam Hay, Eriko Takano
Master of Philosophy Thesis - 2020

A thesis submitted to the University of Manchester for the degree of
Master of Philosophy in the Faculty of Science & Engineering
School of Natural Science
Department of Chemistry

Table of Contents

(Word count: 32673)

Table of Figures

Abstract

The development of effective and reliable control strategies plays a critical role in the optimisation of all bio-based processes, especially with respect to their scaling-up and general transition towards industrial-scale manufacturing. The complexity of the reactions occurring and the sensitivity of the biological components involved place stringent demands on the availability of fine-tuning capabilities and responsiveness of bioprocess control systems. Nevertheless, the implementation of powerful control techniques is hindered by the lack of accurate measurements. In fact, it is fundamental to possess accurate metrics in order to monitor both insightful process parameters and outputs of the implemented control strategies. The collection of such data usually follows the compromise: high accuracy – long measurement time, as opposed to low accuracy – short measurement time.

This project aims to build and validate an innovative monitoring platform, that will allow the implementation of advanced control techniques to a number of bioprocesses. This platform, called BioControl 1.0, consists in a Graphic User Interface (GUI) programmed in MATLAB language and it allows the user to collect and interpret data from a Proton Transfer Reaction Time of Flight Mass Spectrometer (PTR-ToF-MS). The PTR-ToF-MS will sample the headspace of a bioreactor, providing a real-time measurement of the compounds contained in the gaseous phase – in equilibrium with the subjacent liquid one, the concentration of whose solutes is the target of this measurement method.

Despite the applicability of such setup to several processes, this study targets the production of linalool production, a terpene with the potential of being transformed into biofuel. Bacterial synthesis of linalool via *E. coli* has been studied through plasmid engineering: repeated DNA sequences over different areas of the plasmid have been selectively removed, enhancing production consistency (from 13% successfully transformed colonies to 100%) while attaining maximum average titres around 165mg/$L_{overlay}$ in small-scale batches (i.e. 5mL).

BioControl detection capability towards linalool has been tested on scaled-up *E. coli* fermentations carried out in 1L bioreactors (500mL of working volume), resulting in accurate tracking of linalool concentration profile in the liquid phase over a 30h operation time. Linalool titres as low as 100ppb-vol have been successfully detected, while scaled fermentation led to the accumulation of a maximum of 190mg/ $L_{overlay}$.

Feasibility of linalool production control through optogenetics has been tested through MATLAB and Simulink simulations, indicating that stable control is viable.

Declaration

I declare that no portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright statement

Acknowledgments

I want to thank Nigel and Sam for being enthusiastic, affable, and dedicated supervisors: I am honored to have had the possibility to work with both.

A special thanks goes to Mauro, who taught me patiently about molecular biology and who constantly offers his valuable help. Similarly, I want to thank Kat, Clara, Robin, Matt and Lucy for having been the most available and open with me.

I am also truly grateful to our wonderful lab technicians Michi and Shirley, who helped me literally uncountable times.

Lastly, thanks to all of the members of the Scrutton Group for being such a positive and inspiring group of people.

1. Introduction

1.1. Bioprocesses and control

Industrial processes involving biological vectors include a number of different operations, spanning chemical synthesis by means of simple enzymatic systems to multi-organism co-culture anaerobic digestion systems for bio-waste treatment. The variety of applications belonging to this field makes it difficult to categorize the use of core technologies relevant to bioprocessing, and the benefits of making such distinctions is debatable. Nevertheless, it is important to define boundaries for the topic being discussed and a bioprocess definition is therefore needed. Throughout this work, the term bioprocess refers to all of those processes that envision i) the treatment of a biological feedstock, ii) the use of a bio-based catalyst, and iii) the manipulation of microorganisms (Liu, 2017). These descriptors unite bioprocesses that share some fundamental aspects such as feedstock variability, biological systems modelling and microbial homeostasis.

High yield and selectivity, process intensification, reduction in toxic and expensive catalysts, use of mild reaction conditions, and the exploitation of waste materials represent some of the main benefits of bioprocesses over traditional manufacturing approaches (Liguori & Faraco, 2016; Sy et al., 2018; Xie et al., 2019). As such, industrial bioprocessing is one of the fastest growing industrial sectors, with estimates for projected global market value around 360 billion GBP by 2025 (*National Plan for Industrial Biotechnology*, 2013). Evidence of the increased importance of this field is given by the many bioprocesses already adopted in industrial manufacturing, including the synthesis of biofuels and bioplastics ("Industry progress on UCO sustainability for UK biodiesel production", 2019; Sanctis, 2016), waste recovery from different types of residues ("Biodiesel", 2019; "Industry progress on UCO sustainability for UK biodiesel production", 2019) and wastewater purification.

Notwithstanding the above, growth of biomanufacturing industries has been hindered by multiple problems intrinsic to bioprocesses. Inconsistent product yields, non-uniform operational conditions, and maintenance of microorganism homeostasis need urgent attention to allow wider adoption of bioprocesses across the industrial sector. The efficiency of existing technologies has increased dramatically in recent years (Goldrick at al., 2019), but issues such as process consistency and long operation times remain as major challenges. Such problems often arise from the physiological limits of the microorganisms and biocatalytic systems employed, but equally they are also due to the administration of poor control strategies for the overall bio-production process. For example, bioprocess systems rely on fine-tuning of process conditions such as temperature, pH, dissolved oxygen, by-product accumulation, etc. (Fernández-Naveira et al. 2017; Gehan et al., 2019; Pachauri et al., 2017). Similarly, modified microorganisms require metabolic control, since the biological burden imposed by the expression of additional genes inevitably leads to the formation of low-(or non-)producing variants (Rugbjerg & Sommer, 2019). For these reasons, further development of current control technology is mandatory for the scale-up of industrial biomanufacture.

The following work focuses on two elements playing a key role in the design of effective bioprocess control strategies: measurement systems and genetic-level control. Specifically, the possibility of combining PTR-ToF-MS with optogenetic control is addressed: the former possesses high sensitivity and specificity over real-time measurements, while the latter allows for externally tunable genetic control. Proving the feasibility of this type of approach is the ultimate goal of this dissertation.

1.2. Innovative measurement techniques

Robust and reliable measurement techniques are essential to the design and implementation of a successful control strategy. Real data availability dictates control strategy structure and the lack of measurable parameters undermines the efficiency of any conceivable control solution (Holzberg et al., 2018). From the measurement perspective, bioprocesses are inherently challenging systems and are characterized by troublesome features. For example, bioprocesses have specific strict sterility requirements, operate under high levels of agitation, often have gritty/opaque environments with suspended solids, can be run with multiple phase configuration, and are generally ruled by stricter regulation policies than traditional chemical processes (Alford, 2006; Holzberg et al., 2018). In addition to this, many of the required measurements involve quantification of intermediates, enzymes, and inhibitors related to microbial activity. Also, in order to be suitable for control implementation, measurements must ensure high levels of accuracy and (quasi-) real-time acquisition. For all of these reasons, the adoption of control routines for bioprocesses based on novel measurements has been slow to emerge. Examples of successful case studies are available in the literature (Chauvatcharin, 1995; Ferreira et al., 2001; Lidgren et al., 2006; Maciejewska et al., 2006; Moeller et al., 2011; Sagmeister et al., 2013; Yuhong, 2003), but recent contributions focus on sensors for bioprocess monitoring rather than sensors for controlling them (although they may be useful for both purposes).

Holzberg and his co-workers (Holzberg et al., 2018) reviewed the present state of sensor technology in the biomanufacturing industry, with a focus on the transition from batch to continuous processes. Their work analyses the most established sensors available, categorizing them by their measurement rationale (i.e. either optical or electrochemical).

1.2.1. Proton Transfer Reaction Mass Spectrometry (PTR-MS)

Mass Spectrometry is a well-established analytical technique that has become the mainstay analysis approach for numerous applications across various research landscapes. New methodologies for the use of mass spectrometry are constantly under development, widening the range of applicability of this technology.

One promising research field is represented by PTR-MS. This specific MS configuration is based on a mild ionization reaction carried out by hydronium ions through proton transfer, resulting in low levels of sample fragmentation and thus allowing for the detection of the molecular ion and aiding identification of unknown species (see **Figure *1.1***). Proton transfer ionization with hydronium ions is governed by a selection rule which states that only analytes with a proton affinity greater than that of the ion source (i.e. water in most cases) can be ionised. A great advantage of PTR-MS is that common air constituents (N2, O2) have lower proton affinity than water and do not react, thus no diluting buffer gas is required.

This makes PTR-MS highly applicable to the detection of volatile compounds within bioreactor headspace through direct extraction of the 'air' to the mass spectrometer detector and has considerable potential for bioprocess control purposes (Blake, 2009; Romano, Capozzi, Spano, & Biasioli, 2015), leading to interesting applications of this technology, especially if coupled to a Time of Flight (ToF) mass analyser, which offers high sensitivity for the detection of a broad range of masses.

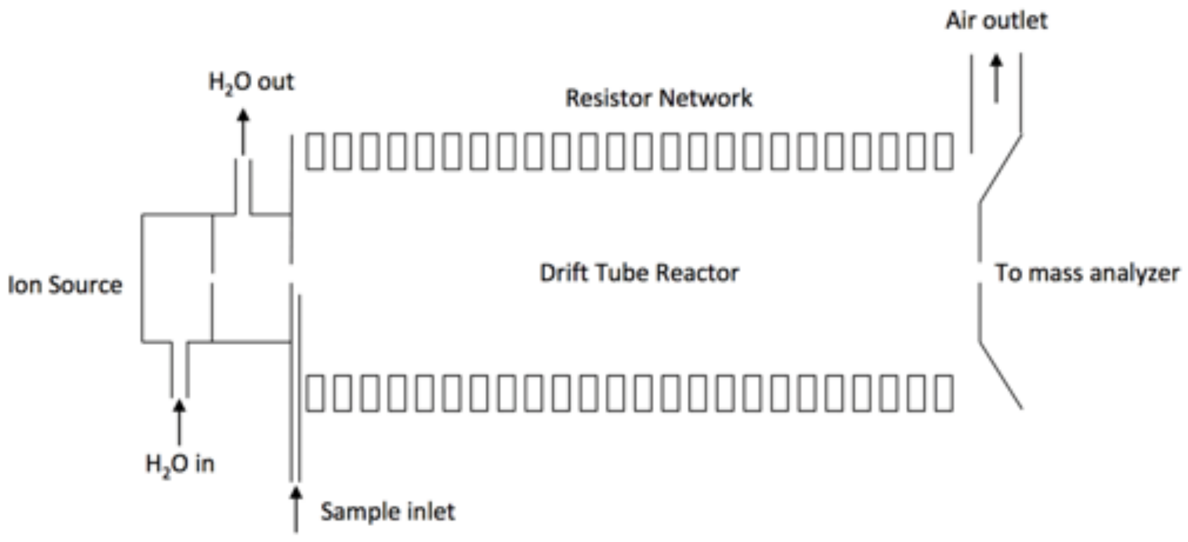


Figure 1.1. *Schematics of a traditional PTR-MS instrument consisting of an ion source, an ion source gas (H_2O) port and a drift reactor constructed by a series of ring electrodes. Adapted from TOFWERK AG website (Tofwerk AG, 2020).*

Benozzi and co-workers (Benozzi et al., 2015) employed PTR-ToF-MS to monitor a yogurt fermentation process. Over 300 substance-related peaks were detected during the analyses, although just 13 showed a statistically significant difference between the tested batches. Most of these compounds strongly contribute to yogurt flavor, and therefore have an important role in product quality evaluation. PTR-ToF-MS is a promising method to screen fermentation progression; it also has potential applications in the control process trajectory to adjust final quality. Specific interest should be given to the detection limits achieved, which resulted in tens of ppbV without the need of any chromatographic separation.

As for what is reported in literature, PTR-ToF-MS technology has not yet been exploited in control engineering, but this is a promising area for future investigation.

1.3. Genetic embedded control

Biological systems are highly regulated. Consequently, all living cells have countless control systems that enable preservation of homeostatic conditions. Deeper insight of natural control strategies and an ability to implement iterative design, build, test and learn (DBTL) synthetic biology cycles should enable molecular level bioengineering of new control loops for industrial bioprocesses (Chen et al., 2013). This “hard wiring” in the cell of next generation control systems – for example, the insertion of genetic switches that respond to internal cell-based stimuli – will open up new opportunities for the control of industrial bioprocesses.

Kobayashi and co-workers have designed a hard wired, genetically encoded control system (Kobayashi et al., 2004). Cell programmability was achieved by borrowing a genetic toggle switch design and rearranging it in a modular fashion. Their system comprised three modules: 1) a signal detection module, 2) a regulatory module, and 3) an output module. In variants of the basic design, different elements can be used in each module. This not only makes the cells programmable but also enables implementation of this modular design in a number of different settings. An important aspect of the overall design strategy is the response of the system to endogenous stimuli. In other words, the signal detection modules respond to endogenously synthesized molecules. This is distinct from prior studies in which genetic switches were created that respond to external stimuli, for example externally added inducer molecules (Gardner et al., 2000). Signal detection modules that respond endogenously to either i) DNA damage or ii) quorum-sensing activated at a specific population density, were used to demonstrate the regulatory potential of these next generation internal control systems. A bimodal genetic toggle switch, which is resistant to noise in the system, was used as the regulatory module (Gardner et al., 2000). Output modules used were either green fluorescent protein (GFP) expression or biofilm production. High sensitivity to DNA damage (based on the detection of single-stranded DNA) and the ability to switch targeted output production back and forth depending on population density, were characterized by these control systems. This is an important milestone in the design of next generation control systems, setting out design principles for building generic tools that enable implementation of internal cellular control mechanisms. In principle, these systems offer feedback control with an on/off policy: when the controlled parameter overcomes a fixed threshold level production is deactivated; then, it is restored once the control parameter falls below this threshold level. However, outputs were found to be sensitive to initial system conditions and the output profiles observed did not attain a constant value.

A stable profile of the controlled output was obtained in a similar control strategy implemented by You and co-workers (You, 2004). They exploited the same signal detection module used in the quorum-sensing system described above in order to track population density. The output module was designed to produce a “killer gene” (CcdB, a cytotoxic protein). Cell growth is prevented by the synthesis of CcdB once the population density reaches a fixed threshold. This provides a constant level of microbial cells and thereby stabilizes the microbial population. The effectiveness of the strategy was demonstrated against an uninduced OFF-circuit control. In this control, the microbial growth curve has a typical exponential profile, followed by a stationary phase when nutrients are depleted. The induced ON-circuit culture overlaps perfectly with the OFF-circuit control in the first part of the exponential phase. However, once population density reaches about 10^8 colony forming units per milliliter of culture (CFU/mL), the rate of growth slows and the growth curve enters an oscillation phase. Over a 17 h period, these oscillations are slowly damped and eventually disappear, giving way to a constant population level for the remaining experiment (conducted over about 30 h). Any deviation from expected cell densities is below 5% during this phase illustrating the attainment of a constant microbial cell population. Similar control strategies have been reported (Atkinson, 2003; Becksei, 2000), but without prolonged stability of the controlled parameter. The observed oscillations are significant – the population density increased two-fold from floor to peak value – and the time delay to reach a stable cell count is high (several hours). Further DBTL cycles could in principle improve and/or modulate these features. Also, the study focuses on population control, without offering any aperture on controlling the expression of specific genes. The latter will likely be important in the design of control systems for industrial bioprocesses.

The work of Aoki and co-workers (Aoki et al., 2019) explores further the potential application of *in vivo* control mechanisms. They regulate either a protein of interest or biomass accumulation by means of a genetically encoded feedback loop. Important to this design is the use of a pair of σ and anti-σ factors SigW and RsiW (**Figure *1.2***). RsiW has the capacity to bind to – and inactivate – SigW. Both SigW and RsiW can be used to create an antithetic feedback motif (i.e. the two factors play opposite roles, the former up-regulating expression, the latter down-regulating it), which the authors argue is needed for robust adaptation to disturbances and set-point changes. They studied two targeted gene expression systems: i) fluorescent protein expression and ii) metE expression, which is required for methionine biosynthesis, cell viability and consequent biomass accumulation. Expression of these systems was directly proportional to the synthesis of the positive regulator of expression RsiW, which annihilates its antithetic SigW. As SigW works as a positive regulator for the expression of the target gene(s) of interest (GoI), this system acts as a stabilizer for productivity. It successfully rejects applied disturbances: upon a 50% output change in the open loop system, the closed loop ensures deviation < 25% and full oscillation damping in about 7 h. In principle, should it be required, this system could also be modulated through the use of inducer molecules (**Figure *1.2***), for example to impart control of TF1 synthesis using external inducers. This could be used for adjusting the position of equilibrium of the overall control system.

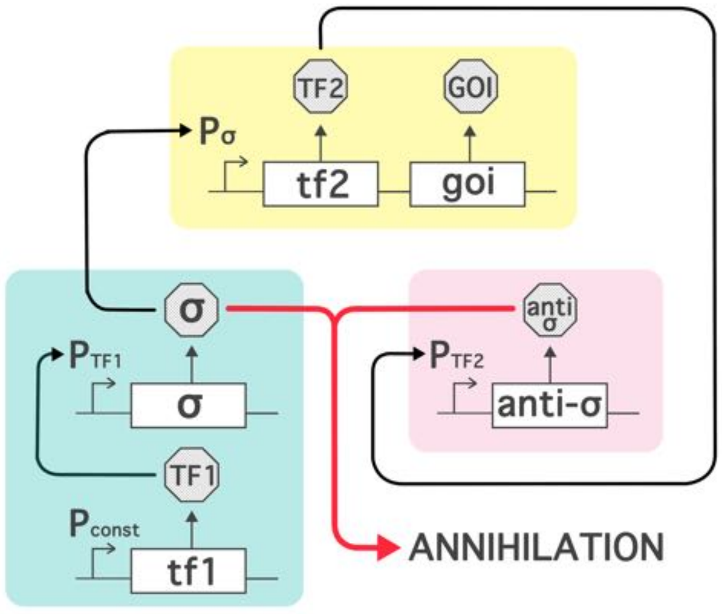


Figure 1.2. *Generalized Aoki and co-workers approach to achieve genetically-encoded in vivo feedback control (Aoki et al., 2019). A constitutively expressed transcription factor (TF1) regulates the expression of SIGW. In turn, SIGW induces the expression of both a second transcription factor (TF2) and a gene of interest (GOI). To close the loop, TF2 regulates the expression of RSIW, which annihilates with SIGW. The rationale is that the more SIGW is synthesized, the more GOI and RSIW are produced – up to the level that all SIGW is annihilated and GOI concentration reaches stability. In principle, the approach could be modified to impart control of TF1 synthesis using external inducers. This could be used for adjusting the equilibrium of the overall control system.*

1.3.1. Optogenetics

The use of optogenetics has emerged as one of the most promising recent strategies to implement genetic embedded control. The triggering of photosensitive switches that respond in an accurate and precise manner to specific wavelengths of light in principle could enable the coupling of *in vivo* and *in silico* control mechanisms. There are numerous advantages in implementing such an approach, including the high temporal and spatial precision obtained, the ability to achieve target specificity and good reversibility (i.e. the capability of the optogenetic switch to revert to the initial state after light exposure) (Bacchus & Fussenegger, 2012; De Mena et al., 2018; Olson et al., 2014; Zhang & Cui, 2015). Especially important for the high-level production of small molecules and other products is the potential to gain “real-time” control of production using optogenetics coupled to real time monitoring of product build up. This could be used to protect against over-accumulation of toxic compounds. These would be important milestones in the implementation of tunable *in vivo* control strategies that would pave the way to numerous further developments. The tuning of such systems, however, would be complex and case dependent. We consider below investigations carried out on *in silico* control systems applied to genetically embedded switches as the first steps towards these goals.

Zhao and co-workers (Zhao et al., 2018) employed a light-activated transcription factor and its corresponding promoter in a system termed "OptoEXP". This was used to gain control for the expression of a particular gene of interest (GoI). The strong dark-to-light expression difference is a specific advantage of this configuration. Studies conducted with green fluorescent protein demonstrated a 43-fold increase in expression when light is activated with respect to dark conditions. Of particular interest is the realization of both inducible and repressible systems starting from an inducible one. Using an antithetic motif (similar to that discussed above for Aoki and co-workers decsibed above), light is used both to induce expression of a gene and to halt expression of another. Effectively, two different systems are generated: i) an *inducible-only system* by placing the GoI after the OptoEXP promoter, inducing its expression when light is switched on; (ii) an *inducible-repressible system* by placing under the regulation of OptoEXP a GoI and an antithetic transcription factor that annihilates the one inducing the expression of a second GoI. In this way, light induces the expression of the first GoI and prevents expression of the second GoI, while darkness acts the other way round (**Figure *1.3***).

The second configuration was applied to the enhancement of isobutanol production by *S. cerevisiae*. As ethanol is one of the main by-products, repression of pyruvate decarboxylase (PDC) is expected to reduce its production. However, total repression of that pathway would prevent growth of yeast on glucose, with consequent impact on biomass accumulation. The inducible-repressible system was therefore exploited to express PDC under light conditions, and favor isobutanol production under dark conditions. This strategy achieves the balancing of microbial growth and isobutanol production. During the dark period, light pulses allowed for energy production to maintain metabolism. By optimizing such pulses and other growing conditions, this approach achieved a production titer of 8.49 g/L isobutanol, 5-fold higher production compared to the best previous strategies.

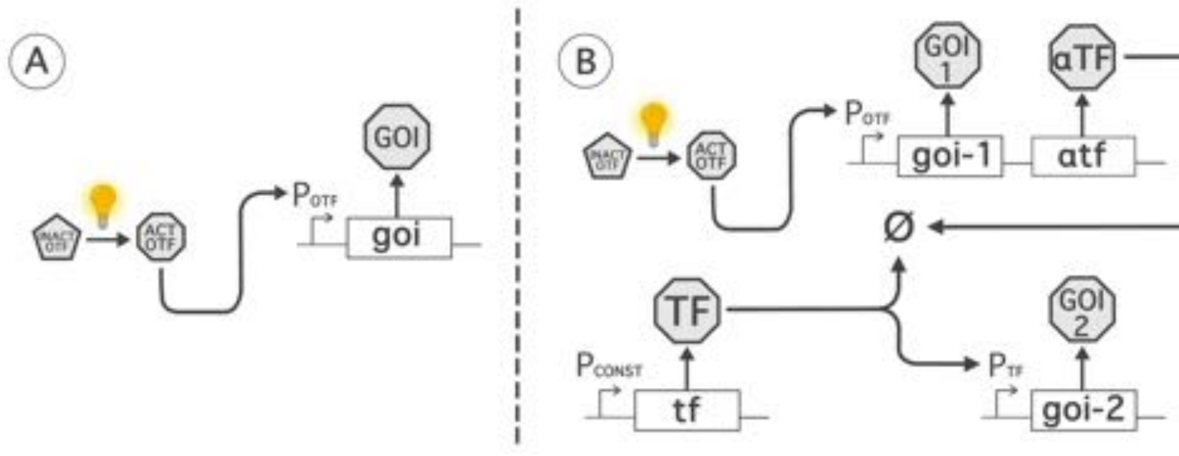


Figure 1.3. *Generalized Zhao and co-workers control strategy through optogenetic switch (Zhao et al., 2018). (A) Inducible-only system: light input allows for the activation of the optogenetic transcription factor (OTF); this induces the expression of the gene of interest (GOI). (B) Inducible-repressible system: light induces the production of a first GOI (GOI1) and the anti-transcription factor (αTF); the latter annihilates the constitutively expressed transcription factor (TF), so that the expression of a second GOI (GOI2) is not induced. Under dark conditions, GOI1 and αTF are not synthesized, hence GOI2 can be expressed.*

The work of Zhao and colleagues demonstrates the potential application of optogenetics to bioprocess regulation, proving that it is possible to implement and articulate modulation strategies at the same time. The aim of the work was to maximize productivity levels, but no hybrid *in vivo – in silico* control policy has yet been applied to this system.

Currently, the only available example of a hybrid *in vivo – in silico* control is that reported by Milias-Argeitis and co-workers (Milias-Argeitis et al., 2016; Milias-Argeitis et al., 2011). Their innovative approach led to the design of the first optogenetic-based feedback control to fully regulate the production of a gene of interest in *E. coli*, in this case a fluorescent protein. The adopted configuration employs a switch triggered by green and red light, resulting in expression and repression of the gene of interest, respectively. Control policy was implemented with fluorescence signal and optical density as the controlled variables, and LED light activation as the manipulated variable (**Figure *1.4***). Standard Proportional-Integral (PI) control and Model Predictive Control (MPC) techniques were compared, proving the efficiency of both during disturbance rejection and the superiority of the latter for set-point tracking. When the desired output is modified over time, its profile was successfully tracked by MPC policy (< 5% deviation), whereas PI fails to achieve the same level of accuracy. MPC was better than PI control because there is a lag from the light signal until the point that the fluorescence signal accumulates sufficiently to enable it to be measured; MPC is able to predict that lag and respond in a timely fashion.

This last application demonstrates a fully genetically-encoded control strategy. It also serves to illustrate the benefit of combining different biological and non-biological tools. Optogenetics is paired with an accurate measurement system (the flow cytometer) and a powerful control technique (MPC). The combination of these elements allows for an effective strategy, able to keep a complex biological system under full control, even when tracking a variable set-point.

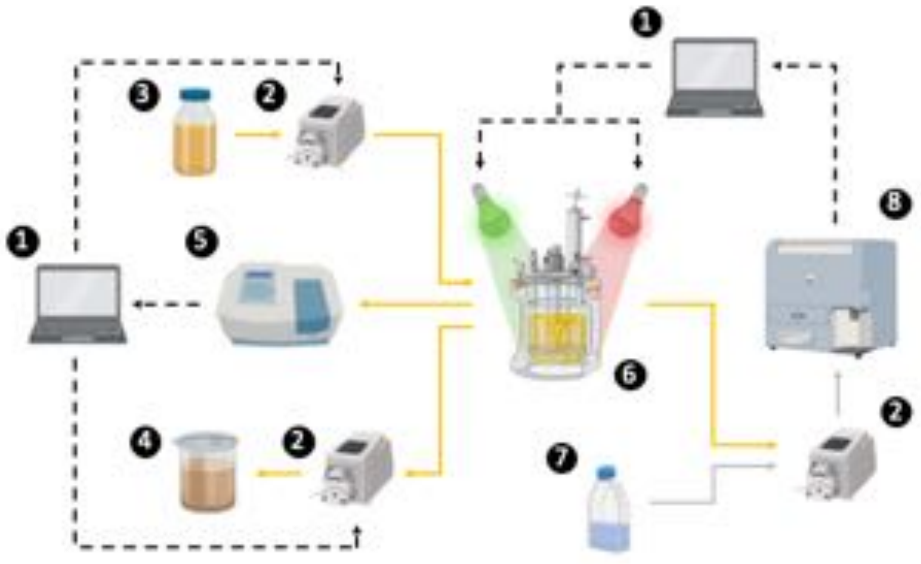


Figure 1.4. *A genetically-encoded in silico feedback control loop. The numbers in the Figure identify individual components as follows: (1) – Controller; (2) – Peristaltic pump; (3) – Fresh culture media; (4) – Removed culture media; (5) – UV-Vis; (6) – Bioreactor; (7) – PBS; (8) – Flow cytometer; (Yellow lines) – Culture media streams; (Dashed lines) – Signal streams. Automatic sampling of culture media allows for optical density measurement of the culture media and fluorescence measurement of the targeted fluorescent protein expression. Set point for both parameters is then tracked by switching LED light from green (protein expression) to red (protein repression) by means of an external computer-based controller. Figure based on reported studies using in silico feedback control (Milias-Argeitis et al., 2016).*

1.4. Terpenoids

Terpenoids (or isoprenoids) contribute to more than half of all of the naturally synthesised compounds discovered to this day, with over 55,000 described structures (Bian et al., 2017; Leferink et al., 2019; Leferink et al., 2016; Oldfield & Lin, 2012; Xiao et al., 2019). Their presence has been detected amongst the majority of life forms and their diversity accounts for the many key biochemical roles they fulfil – from electron transport, to photosynthesis, to membrane structure (Bian et al., 2017; Kempinski, 2015; Lange et al., 2000; Lombard & Moreira, 2011).

Because of the many possible applications they are prone to (see **Table *1.1***), terpenoids represent a commercially interesting product, with a global market worth more than $5 billion

(Leferink et al., 2016). They are commonly used as drugs, supplements, food additives, fragrances, pesticides and as platform chemicals for further transformation (Ashour et al., 2018; Li et al., 2020; Tetali, 2019; Vickers et al., 2017). Moreover, recent studies proved their potential as biofuels precursors, satisfying fundamental requirements like low freezing point, low temperature viscosity, high energy density and high volumetric net heat of combustion (Mendez-Perez et al., 2017; Mewalal et al., 2017).

Table 1.1. *Examples of largely manufactured terpenoids.*

Sector	**Products**
Pharmaceutical	Artemisin
	Cannibidiol
Nutraceutical	Coenzyme Q10
	Squalene
	Vitamin K
	Vitamin E
Food	Valencene
	Nootkatone
	Lycopene
	Menthol
	Beta-carotene
Commodity polymers	Isoprene
	Farnesene
Fragrances	Limonene
	Linalool
	Eucaliptol
	Geraniol
Fuels	Pinene
	Limonene
	Linalool
	Farnesene

Terpenoids are particularly abundant in plants, where they mostly serve as attractors or repellents of animals and insects (Ashour et al., 2018; Kempinski, 2015). Nevertheless, the extraction of these compounds from plants is inefficient, subject to seasonality and unsustainable (Wang et al., 2017; Zhang, 2013). Currently, one of the main industrial sources of terpenoids is turpentine oil, a by-product of the paper industry, from whose distillation many pinenes are recovered – which can be subsequently transformed into other terpenes (Behr & Johnen, 2009; Schwab et al., 2013). The second biggest source is represented by citrus oil from citrus juice production, from which limonene is isolated by distillation (Schwab et al., 2013).

Due to the low efficiency of terpenoids extraction from plants, alternative production pathways are being investigated and a promising route is given by microbial synthesis (Wang et al., 2017). Engineered bacterial chassis can feed on an inexpensive, renewable carbon source and exploit a specifically designed enzymatic pathway to synthesise several terpenoids (Gupta & Phulara, 2015; Tippmann et al., 2013).

Terpenoids are naturally synthesised from the C5 isoprenoid precursor isopentanyl diphosphate (IPP) and its isomer dimethylallyl pyrophosphate (DMAPP). These compounds are the final products of either the methylerythritol 4-phosphate pathway (MEP) or the mevalonate pathway (MVA), which are reported in **Figure *1.5*** (Leferink et al., 2016; Oldfield & Lin, 2012). Condensation of either of IPP or DMAPP by means of a prenyltransferases leads to the formation of geranyl pyrophosphate (GPP), which is the precursor of a number of monoterpenes (i.e. characterised by a C10 structure) (Gao et al., 2012; Oldfield & Lin, 2012). Further condensation with additional IPP/DMAPP molecules allows for the synthesis of precursors for sesquiterpenes, diterpenes, triterpines, etc. (Ashour et al., 2018; Oldfield & Lin, 2012).

To conclude the biochemical pathway to terpenes, enzymes belonging to the class of terpene synthases carry out the last transformation step, bringing about several modifications to the precursors' chemical structure, such as cyclisation, hybrid shift, alkyl shift, deprotonation and reprotonation (Ashour et al., 2018; Kempinski, 2015).

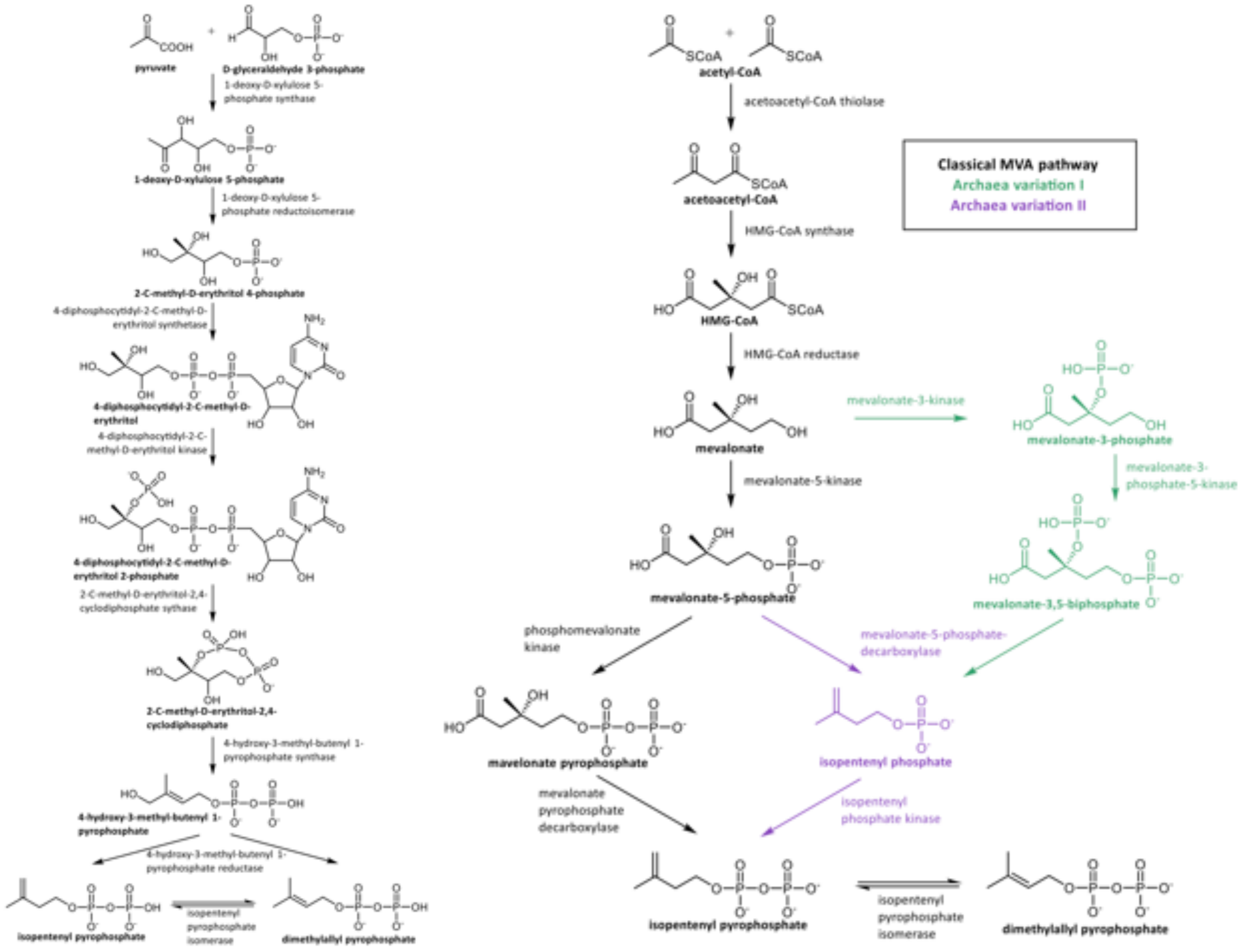

pyruvate
D-glyceraldehyde 3-phosphate
1-deoxy-D-xylulose 5-phosphate synthase
1-deoxy-D-xylulose 5-phosphate
1-deoxy-D-xylulose 5-phosphate reductoisomerase
2-C-methyl-D-erythritol 4-phosphate
4-diphosphocytidyl-2-C-methyl-D-erythritol synthetase
4-diphosphocytidyl-2-C-methyl-D-erythritol
4-diphosphocytidyl-2-C-methyl-D-erythritol kinase
4-diphosphocytidyl-2-C-methyl-D-erythritol 2-phosphate
2-C-methyl-D-erythritol-2,4-cyclodiphosphate sythase
2-C-methyl-D-erythritol-2,4-cyclodiphosphate
4-hydroxy-3-methyl-butenyl 1-pyrophosphate synthase
4-hydroxy-3-methyl-butenyl 1-pyrophosphate
4-hydroxy-3-methyl-butenyl 1-pyrophosphate reductase
isopentenyl pyrophosphate
isopentenyl pyrophosphate isomerase
dimethylallyl pyrophosphate
acetyl-CoA
acetoacetyl-CoA thiolase
acetoacetyl-CoA
HMG-CoA synthase
HMG-CoA
HMG-CoA reductase
mevalonate
mevalonate-5-kinase
mevalonate-5-phosphate
Classical MVA pathway
Archaea variation I
Archaea variation II
mevalonate-3-kinase
mevalonate-3-phosphate
mevalonate-3-phosphate-5-kinase
mevalonate-3,5-biphosphate
phosphomevalonate kinase
mevalonate-5-phosphate-decarboxylase
mavelonate pyrophosphate
isopentenyl phosphate
mevalonate pyrophosphate decarboxylase
isopentenyl phosphate kinase
isopentenyl pyrophosphate
isopentenyl pyrophosphate isomerase
dimethylallyl pyrophosphate

Figure 1.5. *MEP (left) and MVA (right) pathways for isoprenoid precursors IPP/GPP synthesis.*

Since the MVA and the MEP pathways are included in the genome of several bacteria, both of these biochemical routes have been heavily investigated in order to engineer microbial terpenes production (Miziorko, 2011; Oldfield & Lin, 2012). Many successful examples are available in literature, though the complexity of these pathways can place a hurdle on further improvements (Li et al., 2020; Rico et al., 2019; Yang et al., 2016). As a matter of fact, the route from glucose to IPP/DMAPP is formed by a cascade of 18 enzymatic steps: unbalanced/toxic intermediates and rate-limiting reactions easily compromise the final yield (Liu et al., 2020). As a consequence, many studies are focusing on the evaluation of alternative paths, such as the isoprenol mediated or prenol/isoprenol based. Promising results have been achieved for what regards the synthesis of both hemiterpenes and monoterpenes, with titres as high as 620mg/L for isoprene and 500mg/L for limonene (Clomburg et al., 2019; Yang et al., 2016). In addition to this, other studies are investigating on the improvement of terpene synthases, in order to enhance the selectivity towards the production of single terpenes (Leferink et al., 2016).

1.4.1. Linalool

Linalool is a monoterpene alcohol found across many plant species (e.g. basil, coriander, etc.). It naturally occurs in two enantiomeric forms (see **Figure *1.6***), which are characterised by different aromas as well as physico-chemical properties (Dudareva et al., 2006; Kamatou, 2008; Nakano et al., 2011).

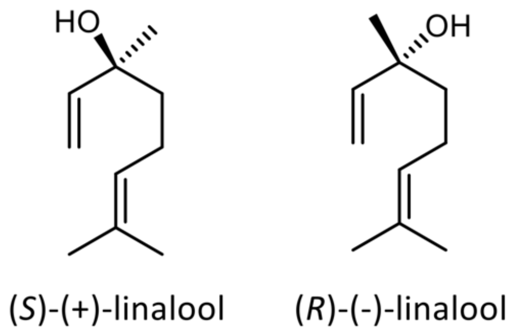


Figure 1.6. *The two naturally occurring enantiomeric forms of linalool.*

Naturally, it is employed by plants as insect repellent and germicide, while industrially it is used in fragrances, hygiene products, household cleaners, food, and beverages. Linalool has also been used in traditional medicine, proving to act as an anti-inflammatory, anti-oxidant, anti-nociceptic, anti-depressant and potential anti-tumoral compound (Cao et al., 2017; Karuppiah et al., 2017; Lapczynski et al., 2008; Zhang, 2013). In addition to this, more recent studies are benchmarking its potential use as a bio-based fuel precursor. As a matter of fact, linalool's structure is suitable to undergo ring-closing metathesis reaction, yielding to rocket and jet engine propellant RJ-4 (Hoye, 1999; Meylemans et al., 2011).

Especially due to this newly discovered potential, studies have focused on designing an innovative production process. Currently, linalool's most common sources are: (i) recovery from linalool-producing plants, (ii) synthesis from pinene isolated from turpentine oil, and (iii) synthesis from petroleum derivatives (Kamatou, 2008; Pommer, 1975). Novel microbial pathways have been explored, engineering bacteria, yeasts, and fungi (e.g. *E. coli, S. cerevisiae, Yarrowia lipolytica*) (Cao et al., 2017; Karuppiah et al., 2017; Mendez-Perez et al., 2017; Zhang, 2013). According to literature, the highest production value was achieved by Mendez-Perez and collaborators, who designed a linalool production strategy exploiting heterologous MVA pathway combined with linalool synthase from *Mentha citrate* and reported titres as high as 505mg/L over a 72h fermentation period (Mendez-Perez et al., 2017).

Despite not achieving commercially feasible productivity, such technology represents a reliable and well-known pathway, with a great potential to future improvements. Live measurement of intermediates and genetically tuneable control over the expression of selected parts of the enzymatic cascade could dramatically improve the overall yield. For these reasons, linalool has been selected as the most suitable candidate for the following work.

1.5. Summary

Industrial biomanufacture is a fast-growing market, but its further development is hindered by process inconsistency and requirement of strict supervision. Adequate process control strategies need to be developed to mitigate these issues, nevertheless accurate monitoring of bioprocesses is difficult to achieve, along with a fine control of the biological vectors.

In this thesis, it is proposed to combine real-time monitoring with PTR-ToF-MS on one side, with DNA-embedded optogenetic control on the other. The former allows for sensitive live headspace measurement of volatile compounds (down to ppt concentrations), while the latter ensures light-controlled gene expression. The feasibility of this approach will be tested on *E. coli* based synthesis of linalool, a monoterpene with widespread applications that has the potential to be used as a biofuel precursor.

2. Project aims

The aims of this project can be summarised as follows:

- Optimise the plasmid used for linalool pathway integration in *E. coli* by removing homologous regions, in order to enhance production consistency;
- Execute preliminary tests to evaluate detection capability of PTR-ToF-MS towards linalool both in vials and in 1L bioreactors;
- Understand PTR-ToF-MS data output and develop suitable tools to manage, interpret and store it;
- Build a data pipeline to handle live measurement of bioprocess headspace;
- Design a User Interface (UI) to manage measurement data;
- Simulate optogenetics control of linalool producing bacterial cultures and test stability of a Proportional Integral Derivative (PID) control routine;

3. Materials and methods

3.1. Strain engineering

3.1.1. Reagents

Ethyl acetate, sec-butyl benzene, nonane, anhydrous magnesium sulphate, *R*-(-)-linalool and terpene mixture were purchased from Sigma Aldrich. Agarose, carbenicillin, isopropyl β-D-1-thiogalactopyranoside (IPTG), Terrific Broth (TB) phosphate buffered and Luria Broth (LB) were purchased from Formedium. SOC outgrowth medium and nuclease-free water were purchased from New England Biolabs (NEB). Glucose was purchased from Fisher Chemical.

All media were prepared according to supplier recommendation and autoclaved before use. Solution stocks of carbenicillin (100 mg/mL), IPTG (100 mM) and glucose (40 %) were prepared, filter sterilized (Sartorius MiniSart 0.22μm) and kept at -20 °C freezer (except for glucose). Carbenicillin was used at a concentration of 100 μg/mL both in liquid media and in agar plates.

Monarch Plasmid DNA Miniprep kit from NEB was used for plasmid isolation, NucleoSpin Gel and PCR Clean-up kit from Macherey-Nagel was used for DNA purification from agarose gel. Standard suppliers' protocols have been followed.

3.1.2. Equipment

Eppendorf MiniSpin centrifuge was used for pelleting 2 mL tubes, Eppendorf ThermoMixer C was used for shaking and heating 2 mL tubes, Biometra TRIO thermocycler was used for all Polymerase Chain Reactions (PCRs), Thermo Scientific NanoDrop 2000 was used for determining DNA concentrations, Infors HT Multitron incubation shaker was used for culture growth and linalool assays, BioRad Gel Doc EZ imager was used for imaging DNA agarose gels, Varian Cary 50 Bio Spectrophotometer was used for measuring Optical Density measurements at 600nm (OD600), Agilent Technologies 7890B GC equipped with an Agilent Technologies 5977A MSD GCMS was used to assess linalool concentration in cultures.

3.1.3. Genes

Linalool pathway – comprising MVA, GPP synthase (GPPS) and linalool synthase (LinS) – was cloned in a single-plasmid, as reported in Leferink at al. (Leferink et al., 2016). This plasmid is reported in **Table *3.1***. Primers employed are reported in **Table *3.2***.

Table 3.1. *Employed plasmids.*

Plasmid name	Origin of replication, antibiotic marker, promoter	Genes	Reference
pMVALinS	p15A, ampicillin, lacUV5	EcAtoB, SaHMGS, SaHMGR, ScMK, ScPMK, ScPMD, EcIDI, AgtrGPPS, bLinS	Leferink et al., 2016

Table 3.2. *Employed primers.*

Primer name	Sequence
pMVALinStoNR2frag1.F	CCATGTCTCTGCCATTCC
pMVALinStoNR2frag1.R	GTCGGCGAAAAAACCCC
pMVALinStoNR2frag2.F	GGTTTTTTCGCCGACTGCA
pMVALinStoNR2frag2.R	AAGGGAGAGCGTCGAGATCC
pMVALinStoNR2frag3.F	TCGACGCTCTCCCTTATGC
pMVALinStoNR2frag3.R	CTTATTGTTGTCTAATTTCTTGTAAAATATGTTCC
pMVALinStoNR2frag4.F	GACAACAATAAGGCGAATTGATCTGGTTTGACAGC
pMVALinStoNR2frag4/5.R	ATGGCAGAGACATGGTTATTTCCTCC
pMVALinStoNR2frag5.F	TAGACAACAATAAGGCTGTTGACAATTAATCATCC
pMVALinStoNR1frag6.F	CAGCGGTTAAGGATCCTTCCTCGCTCACTGACTCG
pMVALinStoNR1frag7.R	TGATCTCCCTCCTAGATCCTTAAGACG
pMVALinStoNR1frag8.F	CTAGGAGGGAGATCATATGAGC

pMVALinStoNR1frag9.R	GATCCTTAACCGCTGCTACG
pMVALinStoNR2isol.F	GCGTTACGATCAAGATCGTCAA
pMVALinStoNR2isol.R	GCTGAACTTGATGTTCTTTGCG
check.F	CGTGTTTGTGATCTGGTGC
check.R	GTATCTTCCTGGCATCTTCCAGG

3.1.4. Strains

Competent *E. coli* NEB-DH5α cells (T1 phage resistant and endA deficient) from NEB were used both for plasmid amplification and expression.

3.1.5. Agarose gel protocol

DNA separation was carried out by agarose gel electrophoresis. Agarose gel was prepared at a concentration of 2% for DNA fragments shorter than 1500bp, 0.8% for longer fragments. Safe View from NBS Biologicals was used as nucleic acid stain in a concentration of 0.01% v/v. Subsequently, DNA samples were dyed using Purple Loading Dye from NEB in a proportion of 1:6 of dye and sample. Gel placed in TAE buffer (4.84g of Tris, 1.142mL of acetic acid, 2mL of 0.5M EDTA in 1L of water at a controlled pH 8.0) was loaded and run for 1h at 100V. ‘100bp’ DNA ladder was used for DNA fragments shorter than 1500bp, ‘1 kb plus’ for longer fragments (both from NEB).

3.1.6. Gene sequencing

DNA fragments and plasmids were sequenced using Eurofins Genomics Tubeseq service. For every 800bp to be sequenced, two 2mL vials were filled with 15μL of plasmid DNA in a concentration of 50-100ng/μL mixed with 2μL of either sequencing primer at 15pmol/μL.

3.1.7. Plasmid preparation protocol

Variations on pMVALinS to remove homologous regions were obtained through PCR and in-fusion cloning from the original plasmid. Three variations were designed: (i) pMVALinS.NR1, (ii) pMVALinS.NR2a, and (iii) pMVALinS.NR2b. The cloning protocols to synthesise them are reported in **Figure *3.1***.

PCRs were executed in 200μL PCR tubes employing Q5 High-Fidelity Hot Start DNA Polymerase Kit from NEB. Reagents amounts and thermocycling conditions are a reported in **Table *3.3*** and **Table *3.4***, respectively. PCR products concentration was checked to be higher than 20ng/μL.

In-fusion cloning was carried out with In-Fusion HD cloning Kit from Takara Bio. DNA fragments were designed to have 15bp overlap and were obtained by PCR amplification, agarose gel electrophoresis and final gel extraction. An aliquot of DNA material was then transferred into a 200μL PCR tube, following kit supplier's recommended amounts (~10ng for fragments < 0.5kbp, or ~50ng for fragments 0.5kbp to 10kbp). The mix was combined with 2μL of In-Fusion HD Enzyme Premix and volume was adjusted to 10μL using nuclease-free water. The tube was incubated for 15min at 50°C and eventually placed on ice.

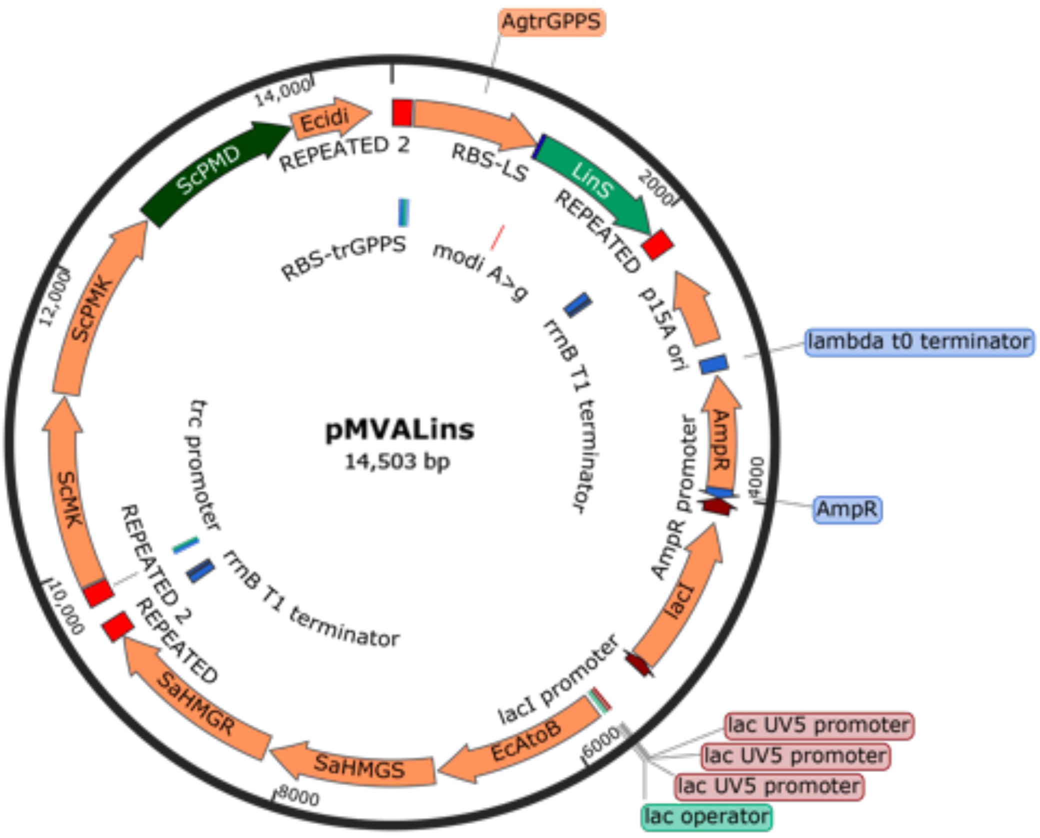
In-fusion cloned plasmids were used to transform competent *E. coli* cells (see **3.1.8 below**) and plate-cultured overnight. Colony PCR was executed on plate colonies to check for appropriate sequence removal and positive colonies were used to carry out the linalool assay. The primers used to carry out the colony PCRs are: check.F + check.R for pMVALinS.NR1, pMVALinStoNR2isol.F + pMVALinStoNR2isol.R for pMVALinS.NR2a, and pMVALinStoNR2isol.F + pMVALinStoNR2isol.R for pMVALinS.NR2b. Gel images and expected amplified fragment sizes are reported in **Figure *3.2***.

Table 3.3. *PCR reagents amounts.*

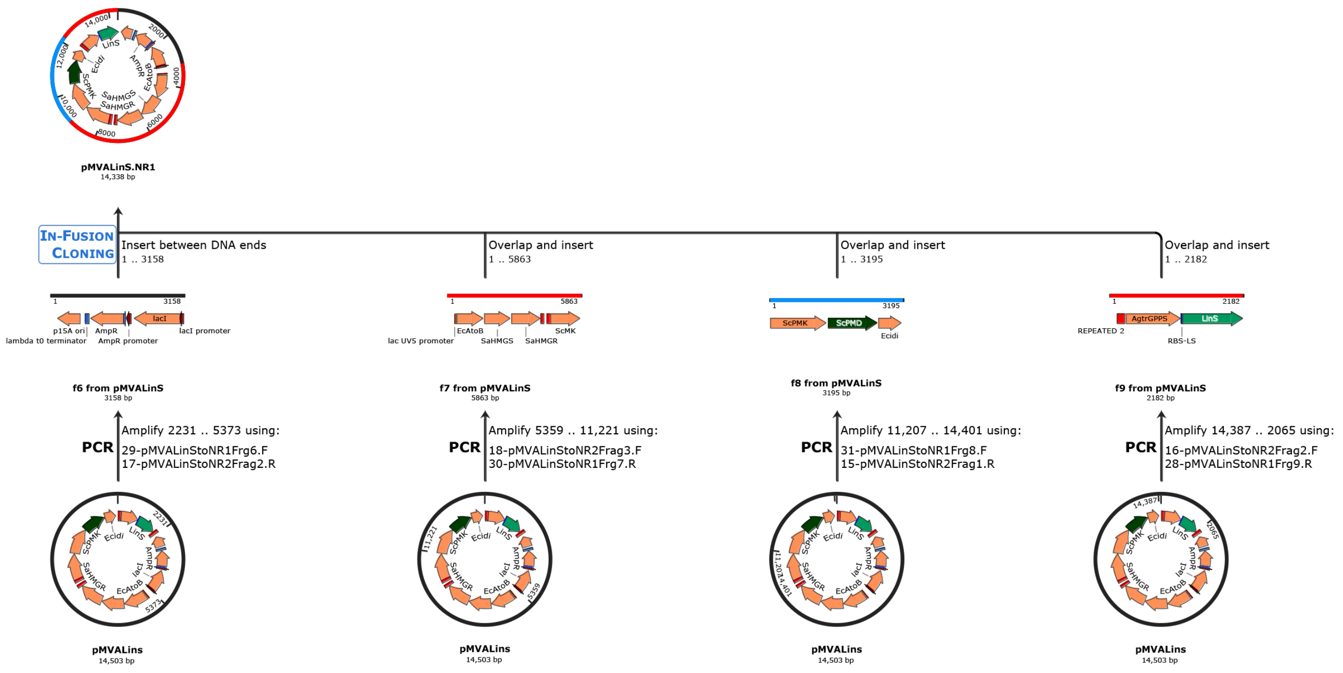
Component	Amount (for 50μL volume)
5x Q5 Reaction Buffer	10μL
10mM NTPS	1μL
10μM Forward Primer	2.5μL
10μM Reverse Primer	2.5μL
Template DNA	0.1 - 0.5ng
Q5 High Fidelity DNA Polymerase	0.5μL
Nuclease-free water	To 50μL

Table 3.4. *3-steps PCR thermocycling conditions. $T_{ANNEALING}$ was calculated with T_M calculator online from NEB.*

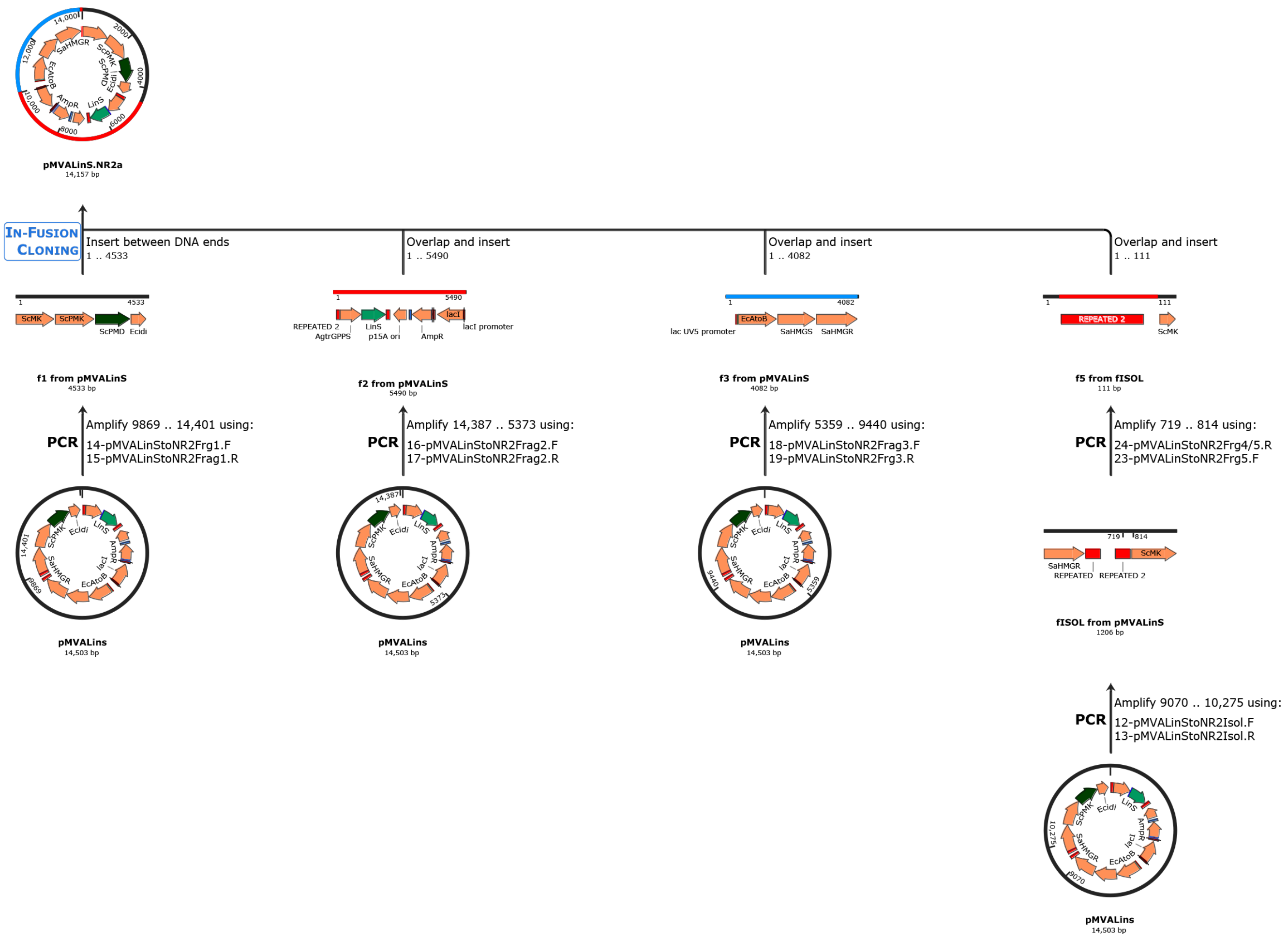
Step name	Temperature (°C)	Time
Initial denaturation	98	30s
Cycling (30x):		
➢ Denaturation	98	10s
➢ Annealing	$T_{ANNEALING}$	30s
➢ Extension	72	20s/kbp
Final extension	72	2min



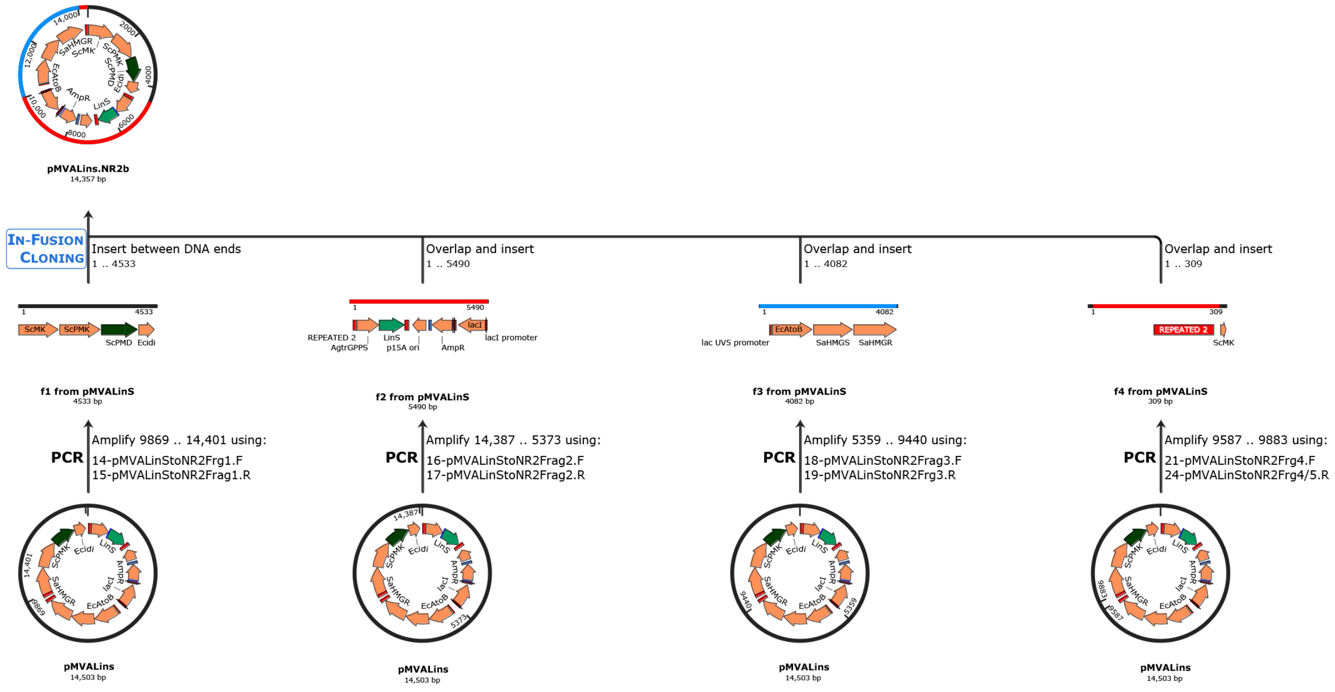
(a)


pMVALinS.NR1
14,338 bp
IN-FUSION CLONING
Insert between DNA ends
1 .. 3158
Overlap and insert
1 .. 5863
Overlap and insert
1 .. 3195
Overlap and insert
1 .. 2182
f6 from pMVALinS
3158 bp
f7 from pMVALinS
5863 bp
f8 from pMVALinS
3195 bp
f9 from pMVALinS
2182 bp
PCR
Amplify 2231 .. 5373 using:
29-pMVALinStoNR1Frg6.F
17-pMVALinStoNR2Frag2.R
PCR
Amplify 5359 .. 11,221 using:
18-pMVALinStoNR2Frag3.F
30-pMVALinStoNR1Frg7.R
PCR
Amplify 11,207 .. 14,401 using:
31-pMVALinStoNR1Frg8.F
15-pMVALinStoNR2Frag1.R
PCR
Amplify 14,387 .. 2065 using:
16-pMVALinStoNR2Frag2.F
28-pMVALinStoNR1Frg9.R
pMVALins
14,503 bp
pMVALins
14,503 bp
pMVALins
14,503 bp
pMVALins
14,503 bp
p15A ori
lambda t0 terminator
AmpR
AmpR promoter
lacI
lacI promoter
EcAtoB
lac UV5 promoter
SaHMGS
SaHMGR
ScMK
ScPMK
ScPMD
Ecidi
AgtrGPPS
LinS
REPEATED 2
RBS-LS

(b)


pMVALinS.NR2a
14,157 bp
IN-FUSION CLONING
Insert between DNA ends
1 .. 4533
Overlap and insert
1 .. 5490
Overlap and insert
1 .. 4082
Overlap and insert
1 .. 111
f1 from pMVALinS
4533 bp
f2 from pMVALinS
5490 bp
f3 from pMVALinS
4082 bp
f5 from fISOL
111 bp
PCR
Amplify 9869 .. 14,401 using:
14-pMVALinStoNR2Frg1.F
15-pMVALinStoNR2Frag1.R
Amplify 14,387 .. 5373 using:
16-pMVALinStoNR2Frag2.F
17-pMVALinStoNR2Frag2.R
Amplify 5359 .. 9440 using:
18-pMVALinStoNR2Frag3.F
19-pMVALinStoNR2Frg3.R
Amplify 719 .. 814 using:
24-pMVALinStoNR2Frg4/5.R
23-pMVALinStoNR2Frg5.F
fISOL from pMVALinS
1206 bp
Amplify 9070 .. 10,275 using:
12-pMVALinStoNR2Isol.F
13-pMVALinStoNR2Isol.R
pMVALins
14,503 bp

(c)



(d)

Figure 3.1. *Original pMVALinS plasmid map and cloning strategies to obtain 3 homologous-region-free alternatives. (a) Original pMVALinS map. (b) Cloning strategy to obtain pMVALinS.NR1. (c) Cloning strategy to obtain pMVALinS.NR2a. (d) Cloning strategy to obtain pMVALinS.NR2b.*

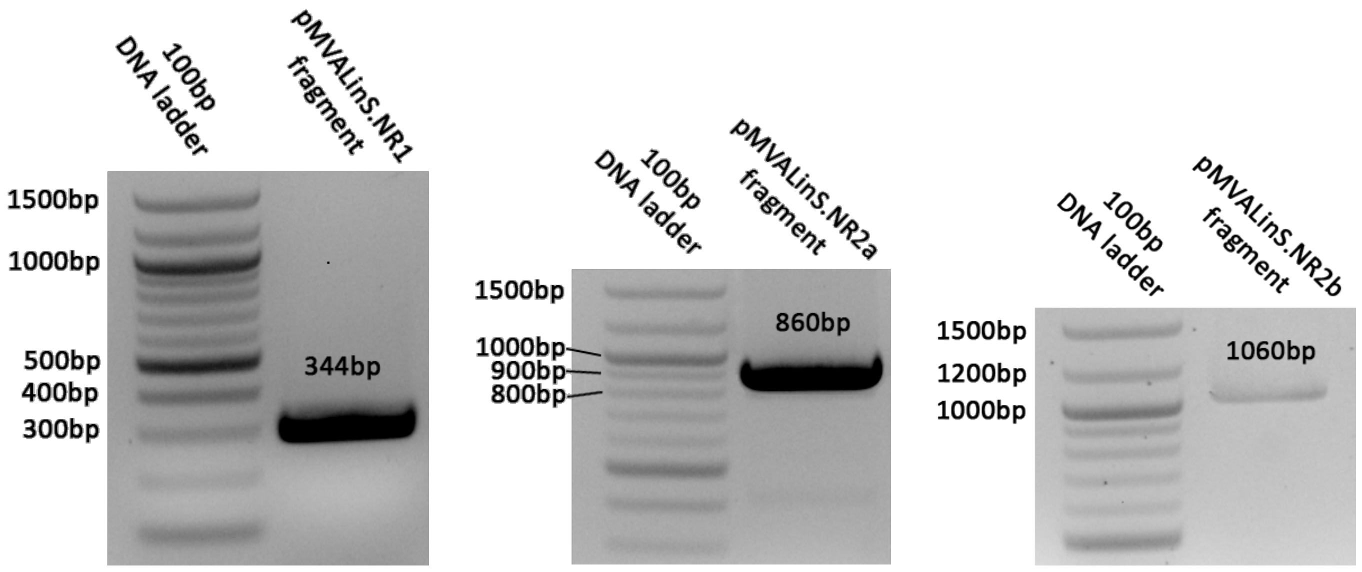


Figure 3.2. *Colony PCR gels of the 3 pMVALinS variations. Ladders are on the left side, while fragments on the right. Expected fragment length is reported.*

3.1.8. Transformation protocol

Plasmids were transformed into *E. coli* strains according to supplier's protocol. Briefly, cells from -80°C freezer were thawed on ice for 10min and approximately 50ng of plasmid DNA was added to 50µL of cell mixture. Cells were placed on ice for 30min, heat shocked at 42°C for 30s and placed on ice for 5min. 250µL of room temperature SOC outgrowth medium was added into the mixture and cells were incubated at 37°C for 1h at 190rpm. Mixture was plated on 2 LB-agar plates with 1‰ carbenicillin in the amounts of 25µL and 250µL and incubated overnight at 37°C.

3.1.9. Linalool production assay protocol

3.1.9.1. Small-scale fermentation

Linalool production assay was executed according to Leferink at al. (Leferink et al., 2016). Glass vials (28mL in volume) containing 3mL TB medium supplemented with 0.4% glucose and 100µg/L carbenicillin were inoculated with overnight plate-cultured colonies from freshly transformed cells. Vials were then placed in shaking oven at 37°C and 190rpm. Upon attainment of 0.4 OD600 (i.e. 4 to 5h), cells were induced with 50µM IPTG, a nonane overlay was inoculated in the proportion of 1:5 v/v, and vials were incubated for 72h at 30°C and 190rpm.

3.1.9.2. Scaled-up fermentation

-80°C glycerol stock of linalool producing strain was recovered overnight in 5mL TB with 100µg/L carbenicillin at 37°C. The next day, 500µL of recovered culture were used to prepare a 50mL TB with 0.4% glucose and 100µg/L carbenicillin inoculum, cultured at 30°C until 0.4 OD600. The so-prepared inoculum was then added to an Infors Multifors 2 1L bioreactor for a total culture volume of 500mL of TB with 0.4% glucose and 100µg/L carbenicillin. The reactor was left to operate overnight at 400rpm, 25°C, and $0.5L_{air}/L_{media}\cdot min$ aeration. The morning after, reactor temperature was increased to 30°C, until OD600 of 0.4 was reached. Upon attainment of the desired OD, the culture was induced with 50µM IPTG and another aliquot of carbenicillin was added to a final concentration of 100µg/L. Fermentation was then carried out for 30h.

3.1.9.3. Linalool quantification

For small-scale cultures, upon completion of the fermentation process vials were collected and centrifuged at 13,000rpm for 2min, in order to allow for phase separation of the water and organic layers. Nonane was then transferred into 2mL tubes and dried over anhydrous magnesium sulphate. Equal volumes of dried sample and 0.1% sec-butylbenzene (used at internal standard) in ethyl acetate were mixed in glass chromatography (GC) vials.

Linalool quantification was carried out through GC-MS analysis. The products were separated on a DB-WAX column (30m length x 0.32mm internal diameter, 0.25µM film thickness, Agilent Technologies). The injector temperature was set at 240°C with a split ratio of 20:1 (1µL injection). The carrier gas was helium with a flow rate of 1mL/min and a pressure of 5.1psi. The following oven program was used: 50°C (1min hold), ramp to 68°C at 5°C/min (2min hold), and ramp to 230°C at 25°C/min (2min hold). The ion source temperature of the mass spectrometer (MS) was set to 230°C and spectra were recorded from 50m/q to 250m/q.

Linalool concentration was extrapolated on the basis of the ratio between the peak areas of linalool and sec-butylbenzene, according to **Equation 3.1**:

$$c_{LIN} = \frac{A_{LIN}}{A_{S-BB}} \cdot m_{CAL} \tag{3.1}$$

where c_{LIN} is the concentration of linalool, A_{LIN} is the linalool peak area, A_{S-BB} is the sec-butylbenzene peak area, and m_{CAL} is the correlation coefficient obtained from calibration against spiked linalool samples (Guiochon & Guillemin, 1988). Effectively, m_{CAL} is the slope of the linear fitting of the calibration points reported in **Figure *3.3*** and its value is $4.6693 \cdot 10^{-4}$. Final concentration value is expressed in $\text{mg}_{\text{LINALOOL}}/\text{L}_{\text{OVERLAY}}$.

To quantify linalool concentration in scaled-up cultures, 1mL aliquots were drawn from the bioreactor and subsequently mixed with 200µL (i.e. 1:5 v/v) nonane to extract the terpene in the organic phase. Then, linalool analysis was carried out through GC-MS exactly as discussed above, although its value is representative of the whole culture media volume, so it must be expressed in $\text{mg}_{\text{LINALOOL}}/\text{L}$ (while if expressed in $\text{mg}_{\text{LINALOOL}}/\text{L}_{\text{OVERLAY}}$ its value would be 5-fold higher).

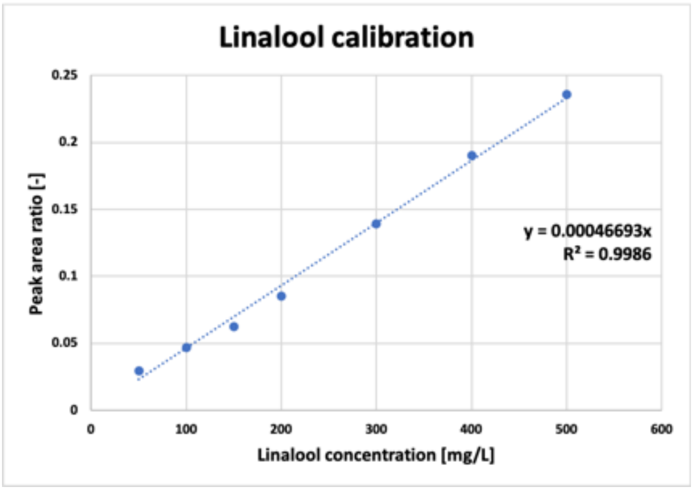


Figure 3.3. *Linalool calibration for GC-MS analysis.*

3.2. Terpenes measurement through PTR-ToF-MS

3.2.1. Experimental setup

Terpenes detection has been assessed by PTR-ToF-MS measurements. The equipment used was a Vocus PTR-ToF-MS from TOFWERK, which was used either alone or in combination with an ARI Modular GC from Aerodyne Research, while sample handling was carried out by means of a PAL RSI Series II Autosampler from PAL System. Both the chromatographic and sampling modules were controlled through Acquility, the Vocus GUI.

Vocus voltages are reported in **Figure *3.4***, reagent flowrate was set at 20sccm, and zero-gas flowrate was kept either at 0sccm for vial detection through GC-PTR-ToF-MS or at 40sccm for bioreactor detection via PTR-ToF-MS.

When using the GC module, samples were loaded on 20mL glass vials and separation was carried out through a DB-WAX column (30m length x 0.32mm internal diameter, 0.25μM film thickness, Agilent Technologies). Injection temperature was variable, with a split ratio of 10:1 (0.2SCCM for 60s of sample purging, for a total of 0.2cm^3 injection). The carrier gas was helium with a flow rate of 1.8SCCM and a pressure of 5.1psi. The following oven program was used: 40°C (1min hold), ramp to 60°C at 13.3°C/min, ramp to 175°C at 46°C/min, ramp to 205°C at 18°C/min, ramp to 225°C at 16°C/min, and ramp to 235°C at 12°C/min (4min hold). The ion source temperature of the mass spectrometer (MS) was set to 150°C and spectra were recorded from 1m/q to 216m/q.

When sampling bioreactor headspace directly with the Vocus (i.e. no GC step included), a heated transfer line was used to connect reactor and MS, whose temperature was kept at 50°C.

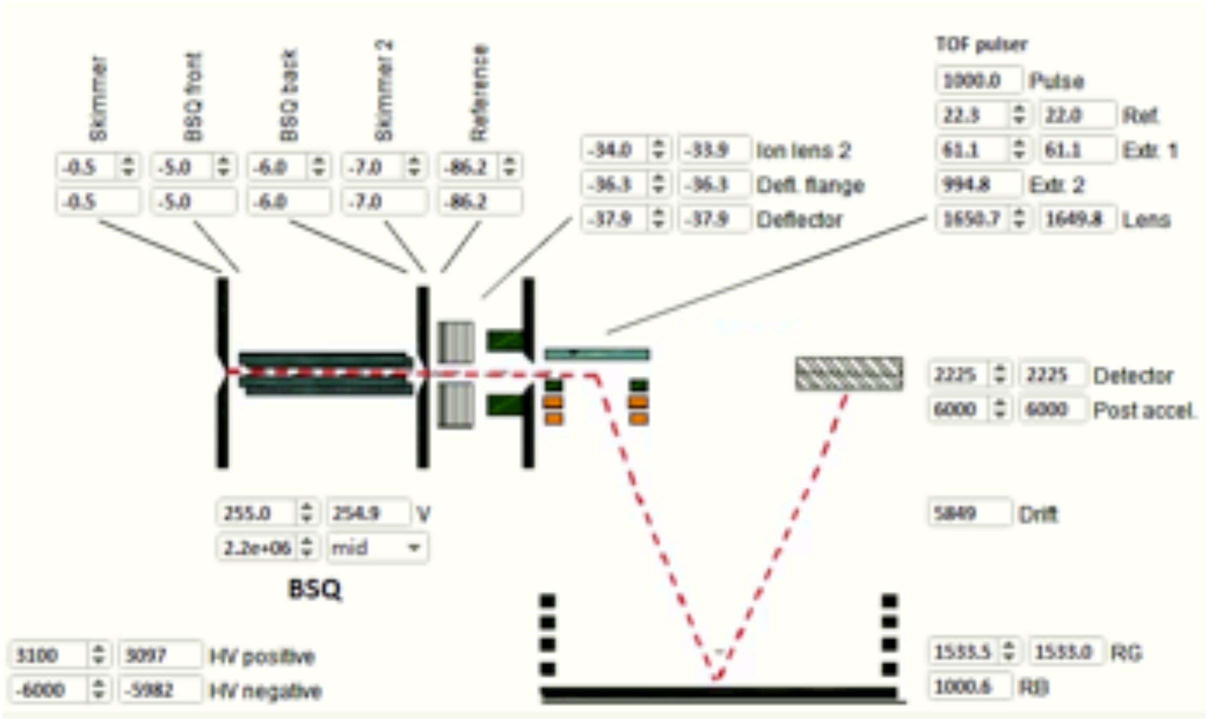

Skimmer -0.5 -0.5
BSQ front -5.0 -5.0
BSQ back -6.0 -6.0
Skimmer 2 -7.0 -7.0
Reference -86.2 -86.2
-34.0 -33.9 Ion lens 2
-36.3 -36.3 Defl. flange
-37.9 -37.9 Deflector
TOF pulser
1000.0 Pulse
22.3 22.0 Ref.
61.1 61.1 Extr. 1
994.8 Extr. 2
1650.7 1649.8 Lens
2225 2225 Detector
6000 6000 Post accel.
5849 Drift
255.0 254.9 V
2.2e+06 mid
BSQ
3100 3097 HV positive
-6000 -5982 HV negative
1535.5 1535.0 RG
1000.6 RB

Figure 3.4. *TOFWERK PTR-ToF-MS Vocus voltage settings.*

3.2.2. Design of Experiment

Design of Experiment (DoE) technique (Montgomery, 2013) was used to assess linalool measurement variability in vials through GC-PTR-ToF-MS. Face-centred central composite design was followed (Montgomery, 2013), employing 2 quantitative parameters – injection temperature and liquid volume – and 2 qualitative parameters – solvent type (TB, nonane, TB+nonane) and sample incubation (incubation, non-incubation). Employed solvents were TB, nonane, and TB + nonane (proportion 5:1 v/v), while 1h-incubation at 30°C was compared with no incubation (i.e. vial headspace sampled at room temperature). The measured output was signal/noise ratio of linalool, whose concentration was kept at a constant value of 10ppm throughout all of the runs.

11 runs were executed for each qualitatively different condition (i.e. 6 configurations), including a central point triplicate. Experimental design and summary of the experiments conducted are reported in **Figure *3.5*** and **Table *3.5***, respectively.

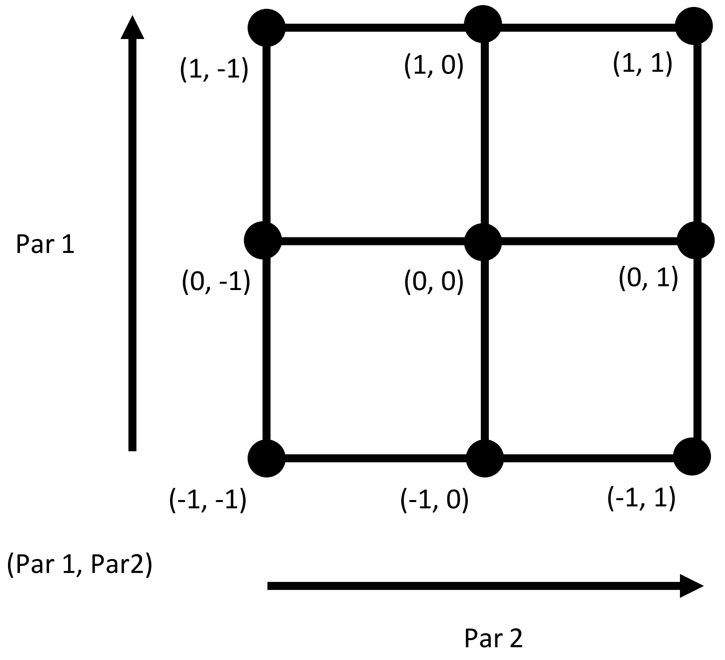


Figure 3.5. *Schematic of face-centred central composite design with 2 quantitative parameters. '-1' and '1' correspond to the lower and upper limits of the experimental range studied, while '0' is the mid-range value. Central point (0,0) run is carried out in triplicate.*

Table 3.5. *Summary of experimental runs executed for each qualitatively different configuration.*

Injection temperature [°C]	**Liquid volume [mL]**
30	0.5
30	2.75
30	5
50	0.5
50	2.75
50	2.75
50	2.75
50	5
70	0.5
70	2.75
70	5

Coefficient regression was carried out on coded variables, defined in **Equation 3.2**:

$$X_i = \frac{x_i - (UL_i + LL_i)/2}{(UL_i - LL_i)/2} \qquad (3.2)$$

where X_i is the coded variable, x_i is the corresponding natural variable (i.e. injection temperature, liquid volume), and LL_i - UL_i are the limits of the experimental range of the variable (e.g. 30°C and 70°C for the injection temperature) (Montgomery, 2013). Coded variables are quantitative variables that are normalised on the basis of their investigation range. The advantage of working with coded variables is given by an easier readability of the results analysis. As a matter of fact, when assessing the effect of a coded variable on an observed parameter, the magnitudes of the model coefficients are directly comparable, since they are dimensionless. Since two quantitative variables were investigated in this study (i.e. injection temperature and liquid volume), two coded variables are employed.

Linear regression based on least sum of squared errors was adopted, including single parameter effects, the interaction effect and the squared single parameter effects, following **Equation 3.3**:

$$\hat{y} = \beta_0 + \beta_1 \cdot X_1 + \beta_2 \cdot X_2 + \beta_3 \cdot X_1 X_2 + \beta_4 \cdot {X_1}^2 + \beta_5 \cdot {X_2}^2 \qquad (3.3)$$

where $\hat{y}$ is the predicted output, β_i are the regression coefficients, and X_i are the coded variables. Response surfaces are generated accordingly. In addition to standard single parameter main effects, It was decided to include interaction effects in order to study a potential influence of either of the quantitative parameter on the other. Additionally, quadratic effects were considered as well, since non linearity of the investigated parameters was suspected.

3.3. Monitoring platform and control simulations

BioControl 1.0, the control platform to handle output data from the PTR-ToF-MS, was designed in MATLAB R2020a, employing the AppDesigner tool. ‘Statistics and Machine Learning’ and ‘Bioinformatics’ toolboxes were used, along with ‘Natural-Order Filename Sort’ script available on MathWorks website.

Simulations of control stability of optogenetic-controlled bacterial systems were carried out in Simulink, with support scripts executed in MATLAB R2020a.

3.3.1. General description of *BioControl*

The main purpose of this project is the realisation of a monitoring platform that will be able to collect data coming from the live headspace analysis of a bioreactor and elaborate it in order to provide insight on the liquid concentration of selected target compounds. This information lays the foundation for future implementation of control strategies, so to improve general bioprocess performances. Since the use of the PTR-ToF-MS technology is innovative with respect to this particular application, it is necessary to build an integrated solution that will allow the user to fully exploit the potential of this setup.

BioControl has been developed with the purpose to offer a software that could import, store, manipulate and display raw data from a PTR-ToF-MS in real time. It has been programmed in MATLAB AppDesigner, which consents the development of standalone software programmes that might be even run remotely on a MathWorks server. So far, its design is able to execute the monitoring function only, which is also limited to one single target compound. All the reported functionalities have been tested against suitably generated mock-files first, and then over real experiments too.

3.3.2. Data structure and handling

The datafiles in output from the Vocus PTR-ToF-MS are organised in a Hierarchical Data Format (HDF) structure and possess the extension '.h5'. This type of data is suitable for the storage of large amount of information, organised in *Groups*, *Datasets* and *Attributes*. The downside of this type of files is the impossibility to gain access to the information gathered within them until definitive file closure. Therefore, file saving has been set to be triggered every 30s through Acquility, so to allow data reading at a rate close to real-time analysis.

Inside of each '.h5' file, ion intensities are stored in 4-dimensional matrixes of size [M x 1 x CS x TC], where M is the number of recorded masses, CS is the chunk size of the single time cycle (i.e. number of time points saved at every time cycle) and TC is the number of time cycles measured during the specific experiment. Usually, M comprises of ~150k mass points (average mass resolution of 0.0014m/q, over a 216m/q range), accounting for the sheer amount of data stored in any '.h5' file (circa 50MB/min of analysis). TP = CS x TC represents the total number of recorded timepoints (CS x TC to be intended as the mathematical product between the numbers indicating the lengths of CS and TC, not as a 2-dimensional matrix with size CS by TC), with an average time resolution of 0.3s. For every timepoint, an ion intensity value is registered at every measured mass. Despite the fact that CS should normally be fixed, it can variate between files – though normally attaining a value of 6. The reason why CS is allowed to change is not clear, nevertheless it must be considered as an unknown variable when querying an '.h5' file.

In order to interpret the data, the original matrix is rearranged in a bidimensional structure with size [M x TP]. On the basis of a user-defined mass target and relative neighbourhood, an interval of rows is extracted and integrated at each timepoint over the selected range of masses. In this way, a single intensity value is paired to a single time value, thus obtaining a final array with size [1 x TP].

To avoid speed problems and excessive task queuing, data manipulation is carried out only on selectively extracted data, so to avoid useless time-costing operations. This means that the packages of raw data imported from the PTR-ToF-MS are not extracted as a whole, but only accessed at the required spot (e.g. only the data related to the range of the mass target is extracted, maintaining the other masses' data compressed).

BioControl runs on the same computer controlling the PTR-ToF-MS, hence '.h5' files are directly imported and read. By employing a *timer callback function* (i.e. a MATLAB object capable to trigger the execution of specific functions in a timed manner), the software recursively scopes for new available datafiles, extracts the target data to interpret and visualise it, and eventually discards it. A data buffer of 5min is kept stored at a time, while additional files are deleted. Nevertheless, the intensity profile over time of the target compound is stored in MATLAB files (i.e. '.mat') for the whole duration of the experiment.

3.3.3. GUI structure and features

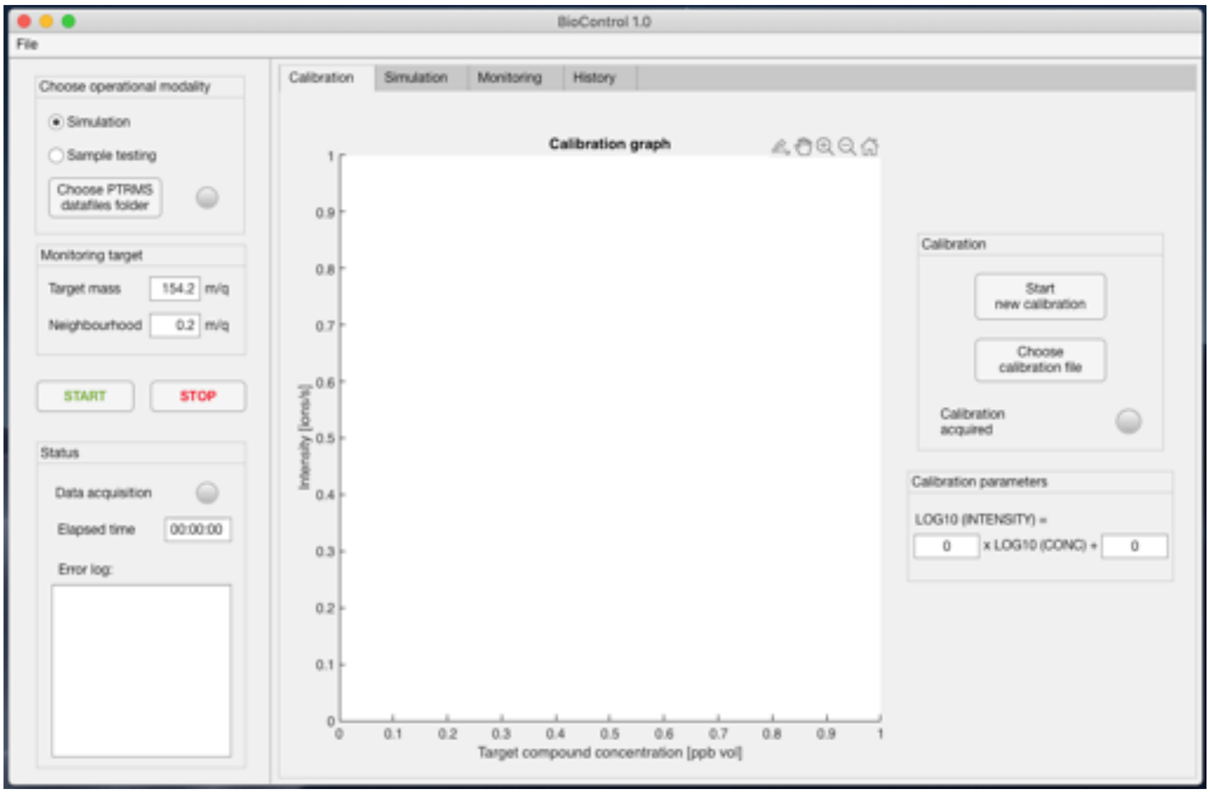
3.3.3.1. Sidebar

BioControl GUI, as reported in **Figure *3.6***, comprises of a sidebar where:

- It is possible to select which modality to operate with, which could be either 'Simulation' or real 'Sample testing'. Along with recursive data analysis, the former one triggers a timer for the repeated generation of mock files that will simulate the expected PTR-ToF-MS output. The latter allows for data analysis and display of real experimental data from the PTR-ToF-MS.

- Target compound mass needs to be entered, as well as its mass neighbourhood, along which to integrate the intensity. This allows for the monitoring of a specific compound (or multiple compounds sharing the same mass).
- A 'Status' panel, flagging the current data acquisition, the elapsed time since data acquisition started, and eventual error messages.
- 'START' & 'STOP' buttons to trigger data acquisition either from real files or from mock ones.

Fields in the 'Monitoring target' panel needs to be filled with suitable values before starting an acquisition. In the case of operating in 'Simulation' modality, it is required to provide more information on the 'Simulation' tab (see below). Error messages regarding any missing field are printed in the error log text area.


BioControl 1.0

Choose operational modality: (●) Simulation ( ) Sample testing

Monitoring target
Target mass 154.2 m/q
Neighbourhood 0.2 m/q

Status
Elapsed time 00:00:00
Error log:

Calibration parameters
LOG10 (INTENSITY) = 0 x LOG10 (CONC) + 0

Figure 3.6. *BioControl sidebar and 'Calibration' tab.*

3.3.3.2. Calibration

The first tab allows to include an experimental calibration, so that the software might directly convert headspace data into liquid phase concentration of the target compound to be analysed.

Pressing on the 'Start new calibration' button, an auxiliary application will be called and a second window will appear (see **Figure *3.7***). It features the possibility to define calibration points over previously recorded datafiles. Effectively, it is possible to pair a user-defined concentration value with one or more '.h5' files containing the headspace measurement of a sample spiked at that same concentration. Such experimental point will be then reported on the calibration graph (along with error bars generated accordingly). Base 10 logarithmic scale is used to plot data, in order to better visualise concentrations across a large scale (from ppb units to hundreds of ppm).

Once all of the points are included, it is possible to regress the parameters of a linear fit of the calibration data (pressing the button on the lower left 'Linear fit'), which will be shown in the dedicated text boxes (i.e. 'Slope', "Intercept' and 'R2' fields on the bottom of the window). Linear regression of the parameters is carried out on logarithmic values, as it results in a more balanced fit of all of the experimental points. A line showing the regressed fit will be also reported on the graph (see **Figure *3.8***). Again, it is necessary to specify a target mass and neighbourhood to allow for the extraction of data specific to the compound of interest ('Compound central mass (Da)' and 'Compound mass neighbourhood (Da)' fields on the right side of the window). Not providing such info will trigger suitable error messages reported on pop-up alert windows.

After regressing the fitting parameters, it is possible to store them along with experimental points data in an '.xlsx' spreadsheet. Furthermore, in order to test platform functionality, it is possible to generate a random calibration set by pressing the button 'Generate mock calibration'. The mass of the mocked mass will be requested upon generation (see **Figure *3.9***).

Once an '.xlsx' calibration file is available, it is possible to load it via the *BioControl* 'Calibration' tab, pressing on the button 'Choose calibration file' (see **Figure *3.6***). Importing that, will populate both the calibration graph and the 'Calibration parameters' panel, along with resulting in a green 'Calibration acquired' lamp (see **Figure *3.10***) – mandatory to proceed with data acquisition.

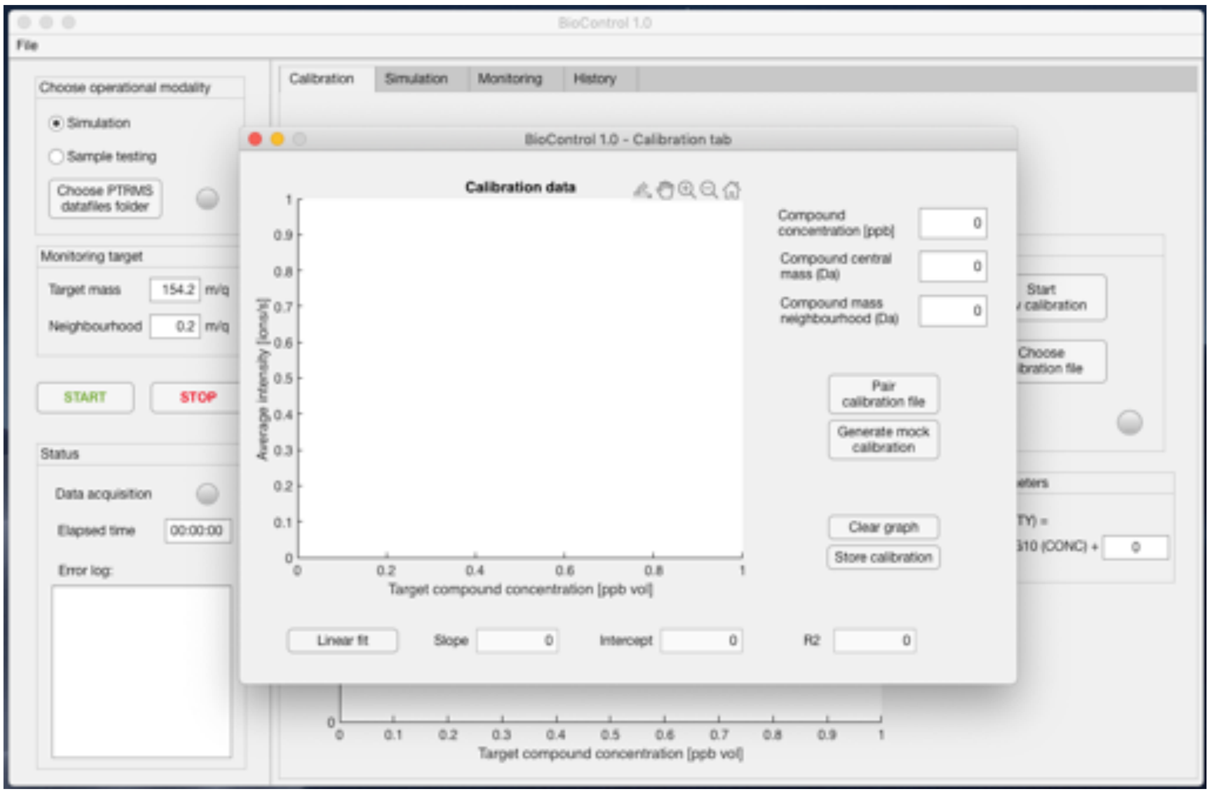
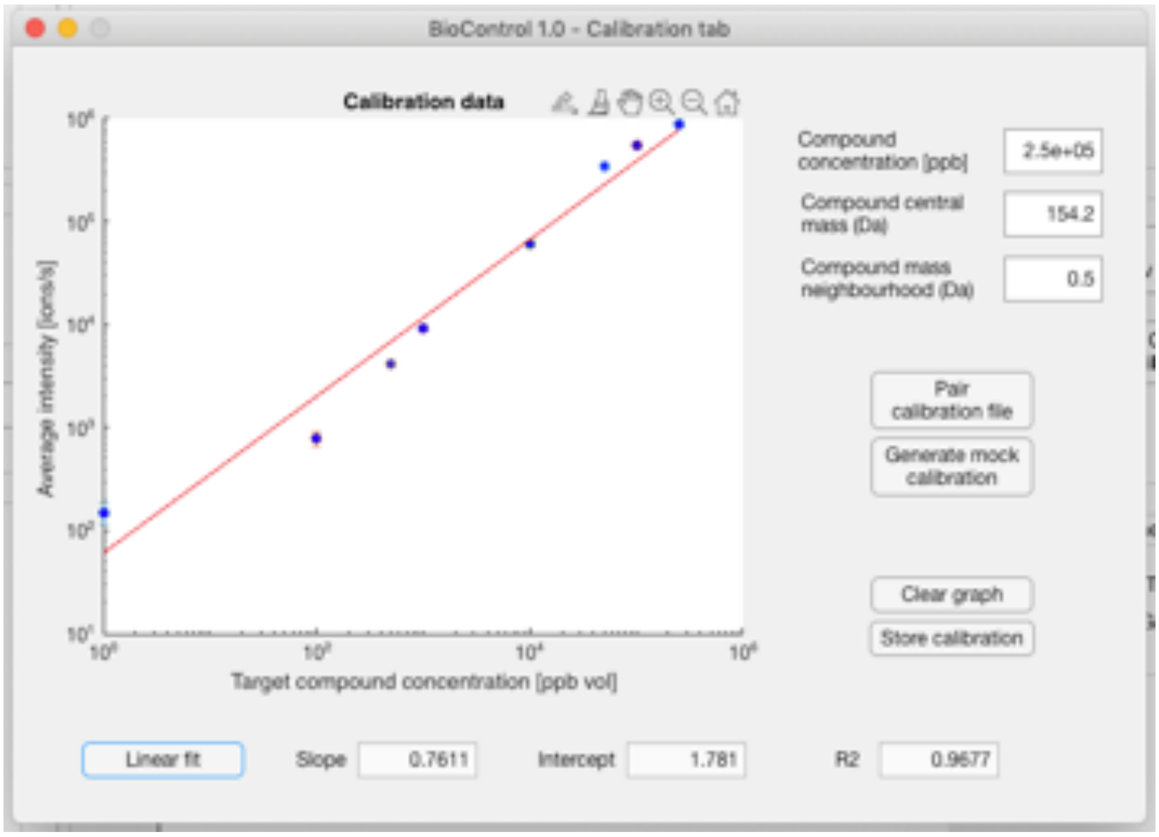


Figure 3.7. *BioControl calibration app window – general outlook.*


BioControl 1.0 - Calibration tab
Calibration data
Average intensity [ions/s]
Target compound concentration [ppb vol]
Pair calibration file
Generate mock calibration
Clear graph
Store calibration
Linear fit

Compound concentration [ppb] 2.5e+05
Compound central mass (Da) 154.2
Compound mass neighbourhood (Da) 0.5
Slope 0.7611
Intercept 1.781
R2 0.9677

Figure 3.8. *BioControl calibration app window – experimental calibration points and linear fitting.*

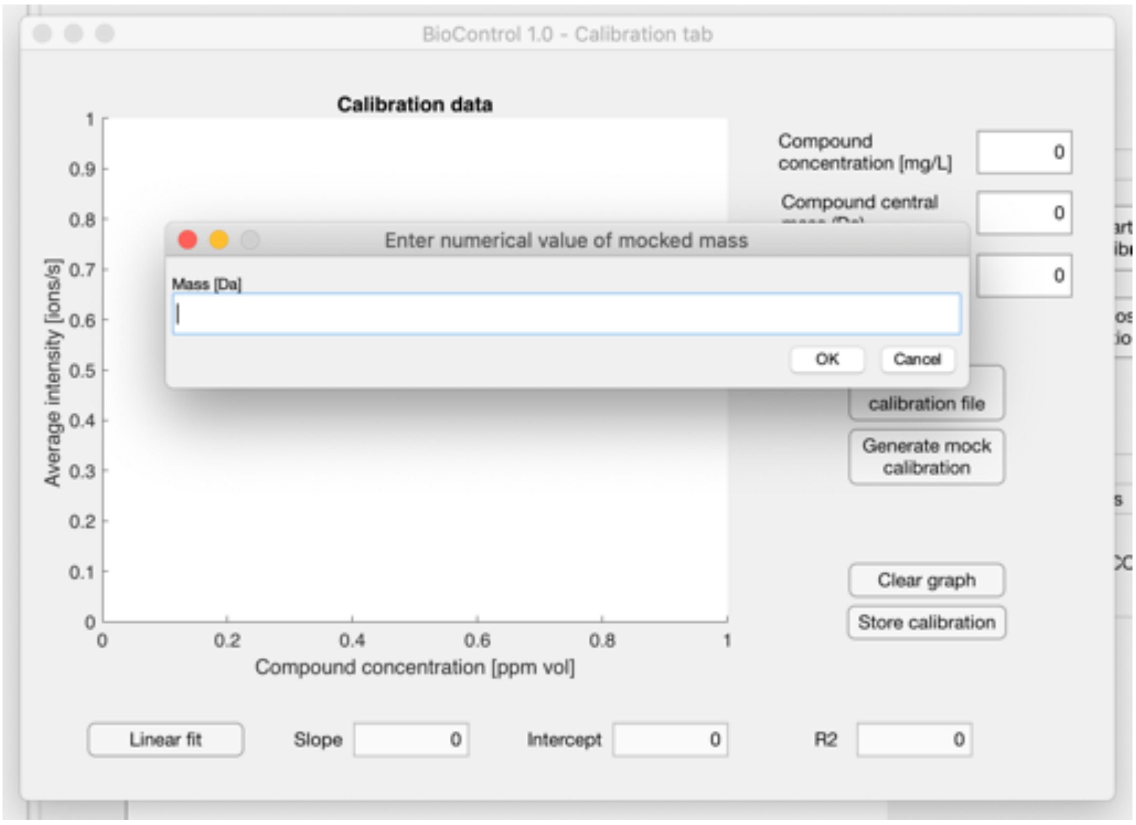

BioControl 1.0 - Calibration tab
Calibration data
Average intensity [ions/s]
Compound concentration [ppm vol]
Compound concentration [mg/L]
0
Compound central
0
0
Enter numerical value of mocked mass
Mass [Da]
OK
Cancel
calibration file
Generate mock calibration
Clear graph
Store calibration
Linear fit
Slope
0
Intercept
0
R2
0

Figure 3.9. *BioControl calibration app window – prompt window to provide mass value for the mock calibration set generation.*

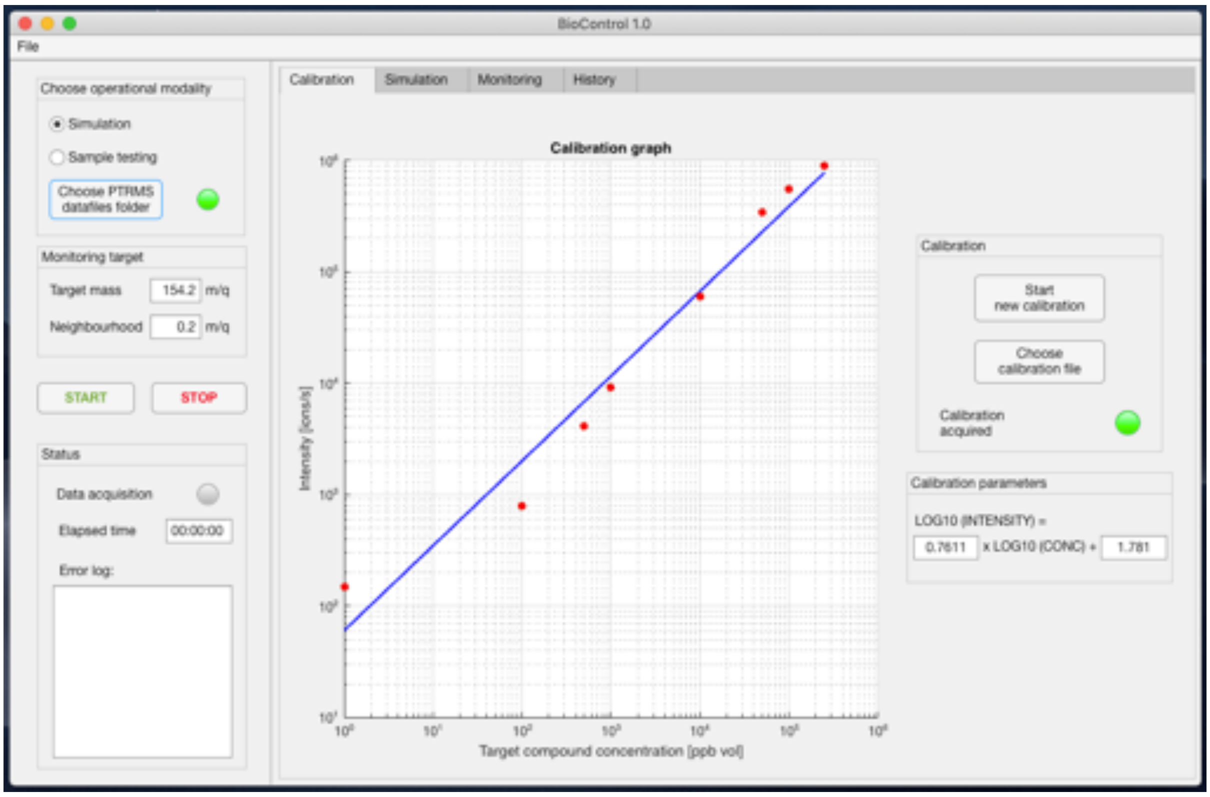
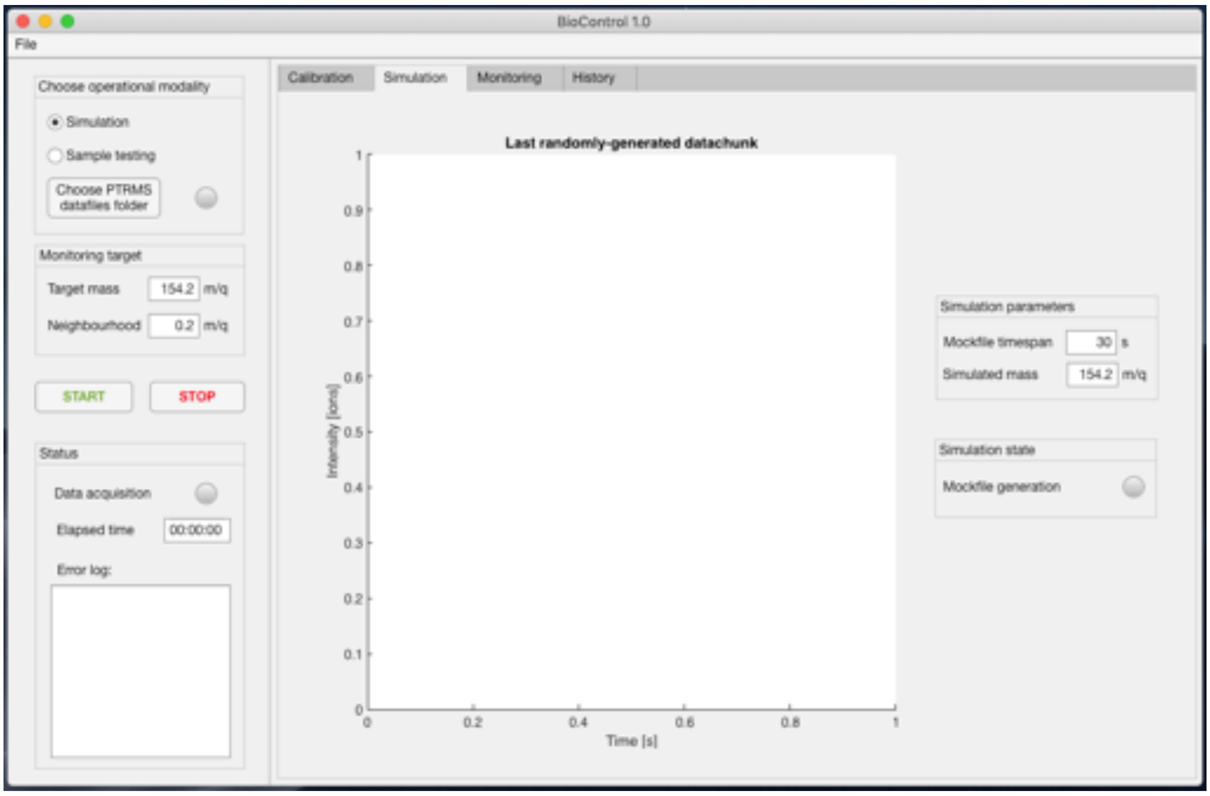


Figure 3.10. *BioControl 'Calibration' tab populated with imported calibration data.*

3.3.3.3. Simulation

In the case of the execution of a simulated data acquisition, it is necessary to include additional data on which mass value would be simulated. Through the ‘Simulation’ tab (see **Figure *3.11***), in the ‘Simulation parameters’ panel, the user must include the desired time range that each mock-file will cover (‘Mockfile timespan’) and the mass to be simulated (‘Simulated mass’). As a matter of fact, as previously mentioned, the expected output of the PTR-ToF-MS consists of a series of ‘.h5’ files, each containing data of about 30s of operation. This value can be customised by the user to have smaller or larger mock-files (useful to test performances).

Once data acquisition is started (and therefore mock-file generation is too), intensity data from each simulated timespan will be recursively reported on the simulation graph (see **Figure *3.12***).



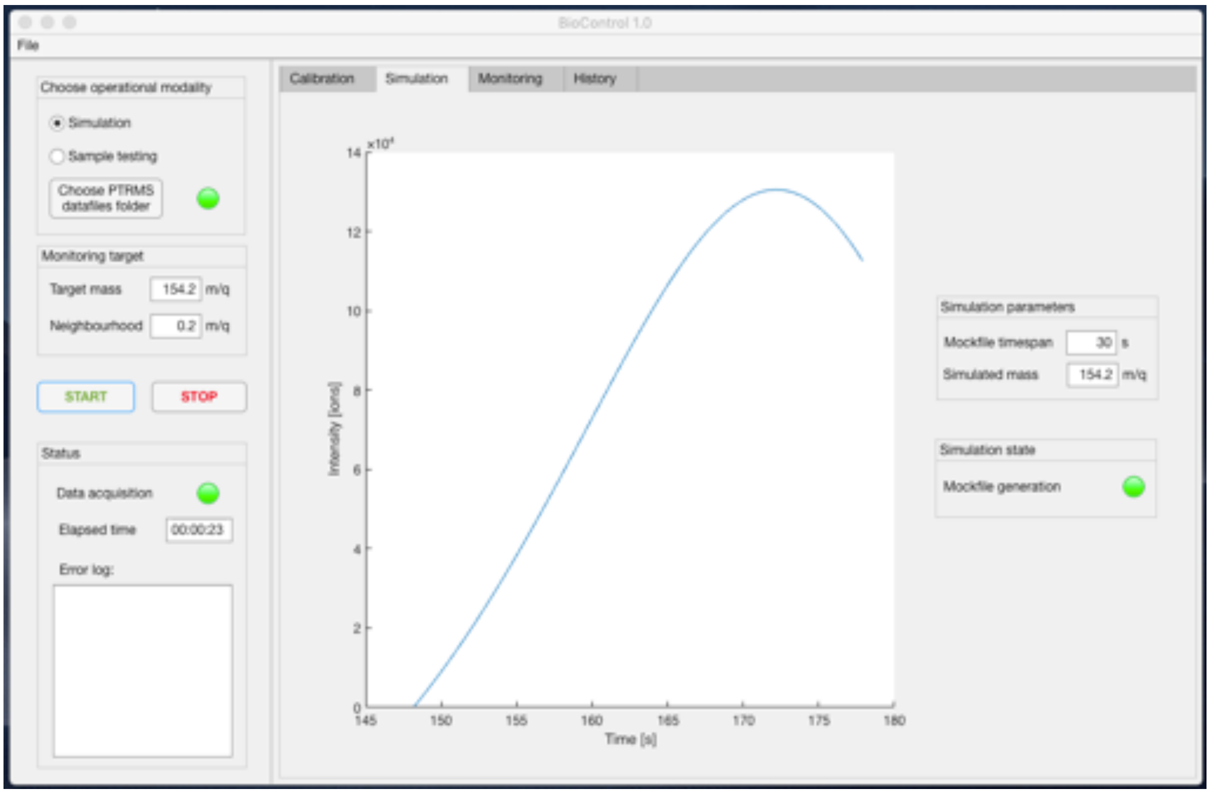
(●) Simulation
( ) Sample testing

Target mass 154.2 m/q
Neighbourhood 0.2 m/q

Elapsed time 00:00:00

Mockfile timespan 30 s
Simulated mass 154.2 m/q

Figure 3.11. *BioControl ‘Simulation’ tab – general outlook.*


BioControl 1.0

File

Choose operational modality

[✓ Simulation] Sample testing

Choose PTRMS datafiles folder

Monitoring target

Target mass 154.2 m/q

Neighbourhood 0.2 m/q

START STOP

Status

Data acquisition

Elapsed time 00:00:23

Error log:

Calibration | Simulation | Monitoring | History

Simulation parameters

Mockfile timespan 30 s

Simulated mass 154.2 m/q

Simulation state

Mockfile generation

Figure 3.12. *BioControl 'Simulation' tab – outlook during simulation.*

3.3.3.4. Live monitoring

In the 'Monitoring' tab (see **Figure *3.13***), the acquired data is reported. The top graph shows the 'Raw headspace ion count', which consists of the ion intensity profile for the target mass to be analysed (each timepoint representing the mass spectrum integrated over the predefined neighbourhood of the target mass). The bottom graph exploits the provided calibration to display the concentration in the liquid phase of the target compound, which is displayed as a 2-minutes-rolling average. Examples of these graphs during acquisition over real data and mock-files are reported in **Figure *3.13*** and **Figure *3.14***, respectively

.

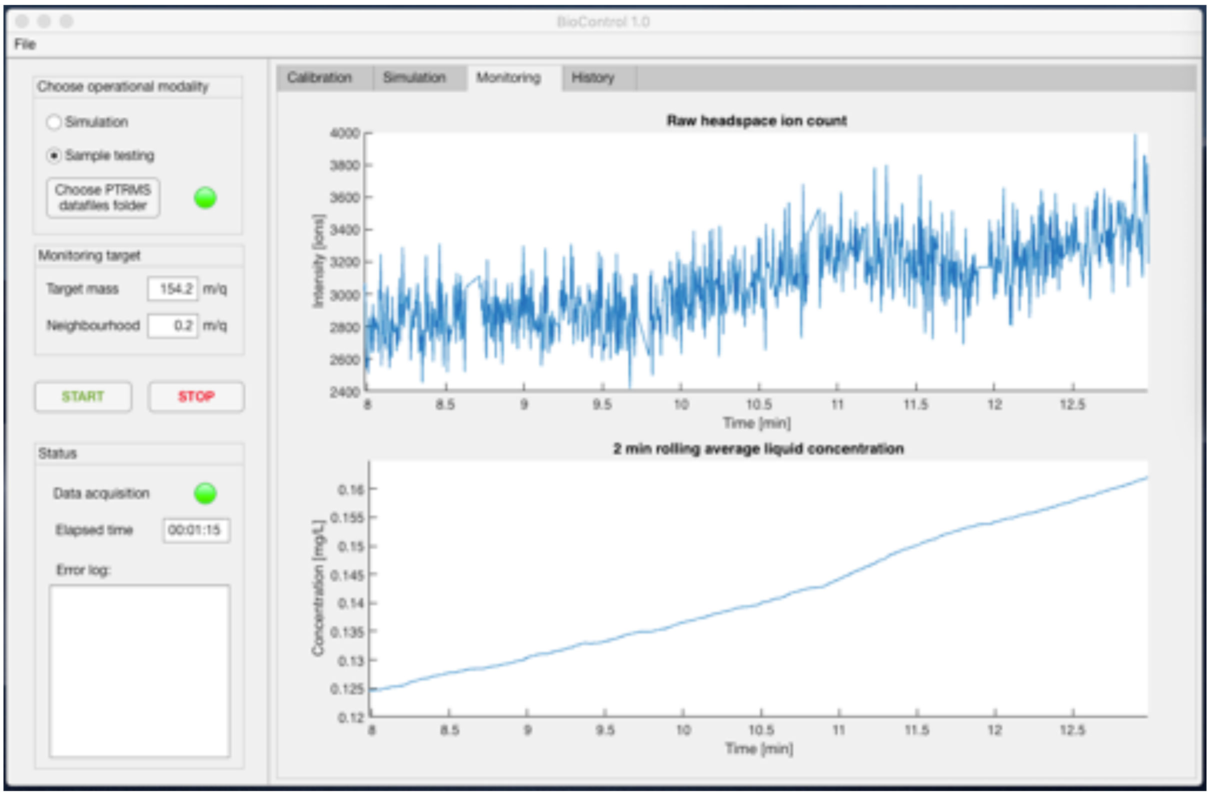


Figure 3.13. *BioControl 'Monitoring' tab – outlook during live PTR-ToF-MS acquisition.*

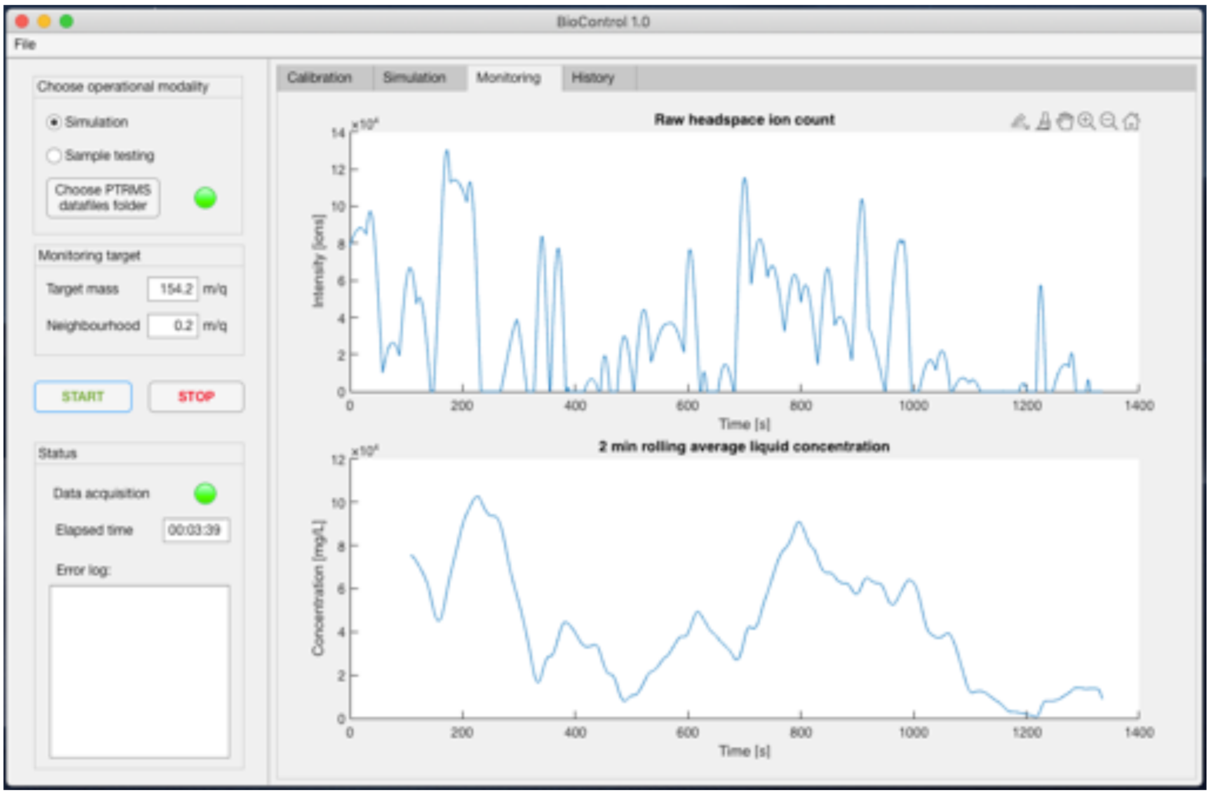


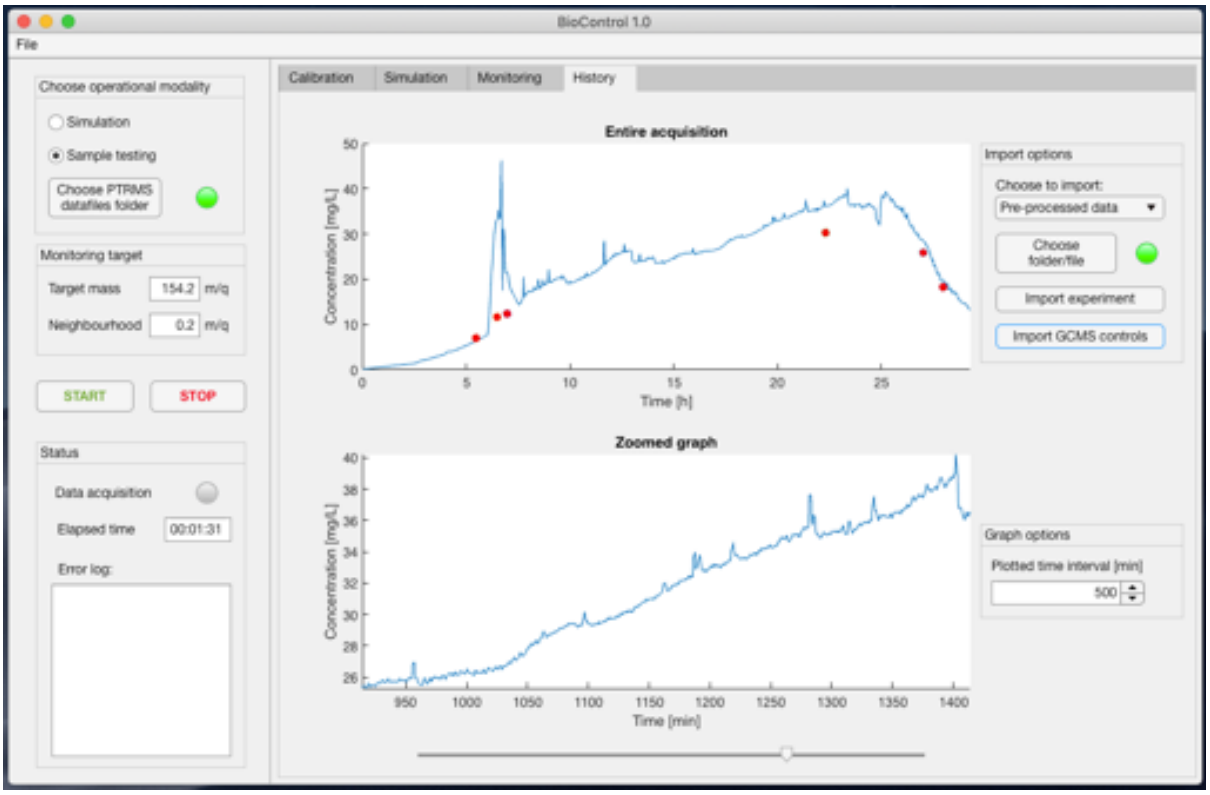
Figure 3.14. *BioControl 'Monitoring' tab – outlook during simulation.*

3.3.3.5. History import

One additional feature of BioControl is the ‘History’ tab, in which it is possible to import previous experiments. As a matter of fact, the ‘Monitoring’ tab displays current headspace data over a window of 10 minutes only, so that this feature is ideal to examine acquisitions in their entirety.

It is possible to import either raw PTR-ToF-MS data (i.e. ‘.h5’ files) or pre-processed data from previous acquisitions run with BioControl (see upper graph in **Figure *3.15***), which automatically saves compressed information about time-signal intensity profile in ‘.mat’ files while operating. In addition to this, it is possible to analyse more in detail the imported acquisition by means of a zoomed graph coupled with a scroller and a selector for the time interval one wants to scope (see lower graph in **Figure *3.15***).

Ultimately, it is possible to add GC-MS controls to the acquisition history, which can be manually imported through the button ‘Import GCMS controls’ (see **Figure *3.16***).



Operational modality: Sample testing

Target mass: 154.2 m/q

Neighbourhood: 0.2 m/q

Elapsed time: 00:01:31

Choose to import: Pre-processed data

Plotted time interval [min]: 500

Figure 3.15. *BioControl ‘History’ tab – outlook with imported acquisition and GC-MS controls.*

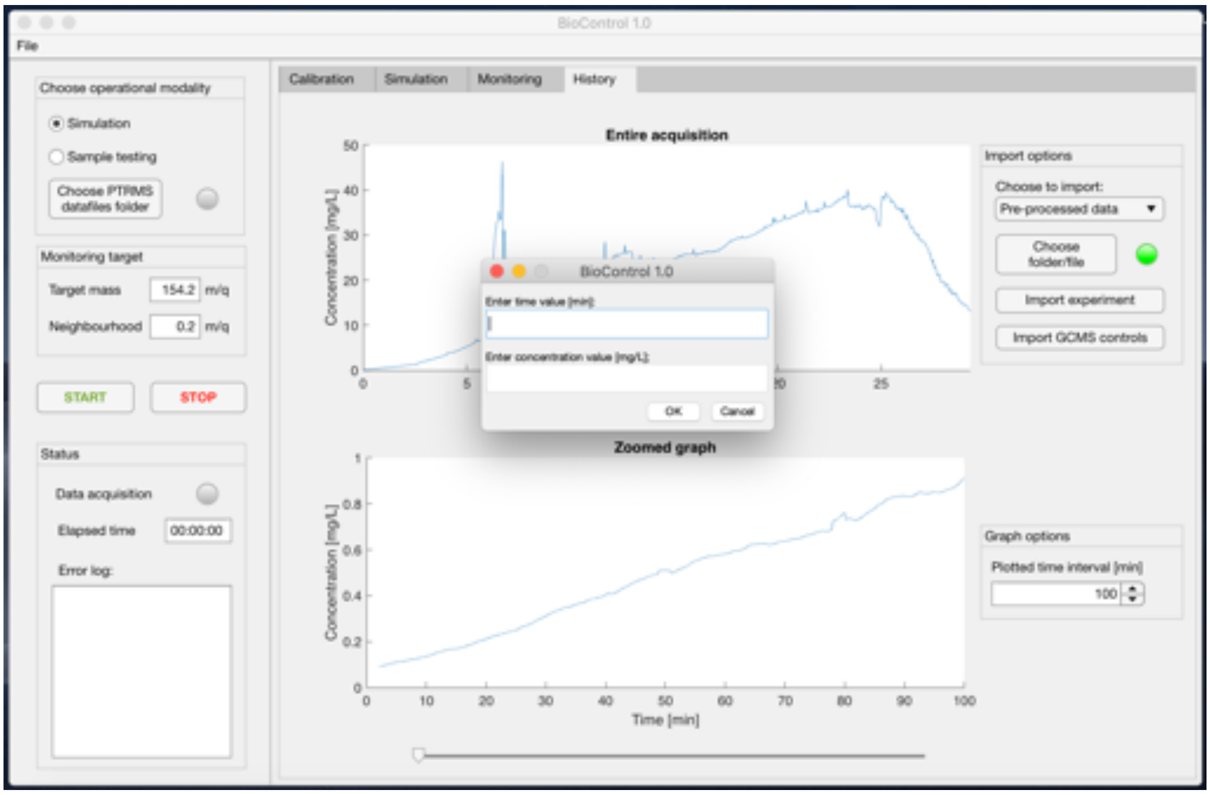


Figure 3.16. *BioControl 'History' tab – detail of pop-up window to manually enter GC-MS controls.*

4. Results and discussion

4.1. Plasmid engineering

Linalool production by *E. coli* transformation with plasmid pMVALinS has proved to be successful, nevertheless it seems to lead to inconsistent synthesis. As a matter of fact, despite the adoption of a uniform transformation protocol and a standard analytical assay, each fermentation round seems to strongly deviate from the others. Data from previous experiments (following the exact same transformation and fermentation protocols adopted for this work) has been analysed and it comes to light that standard deviations can be almost as high as the average output values, as shown in **Figure *4.1***. Considering the reported 10 batches (three biological replicates), the overall productivity value lies around 240mg/L over the course of 72h of fermentation, with a standard deviation of 196mg/$L_{OVERLAY}$.

In addition to this huge inter-batch variability, at times cultures yield titres sensibly lower with respect to the other biological replicates of the group (<30% of the average batch value). In general, around 13% of the samples show this behaviour, but for some cloning rounds this percentage could be even higher, even up to 100%.

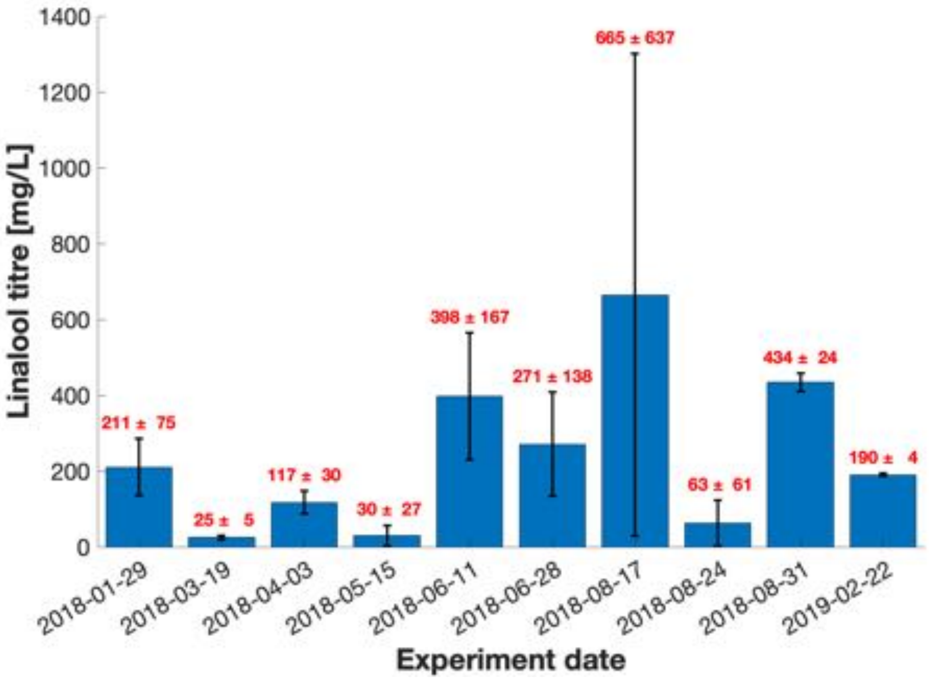


Figure 4.1. *Previous assays for linalool production in* E. coli *transformed with pMVALinS. Averages and error bars are calculated over 3 biological replicates.*

A possible reason for the high variability could be given by the presence of homologous regions on pMVALinS. As a matter of fact, there are two pairs of repeated sequences: (1) between 2076-2213bp and 9449-9586bp, (2) between 1-134bp and 9724-9856bp. These regions could cause homologous recombination, which would lead to the formation of two separate plasmids, each one containing part of the linalool pathway. Eventually, the plasmid not containing the antibiotic resistance would end up being rejected by the cell, thus compromising linalool production.

Region no.1 measures 138bp and contains the terminators rrnbB T1 and T7Te. Its two homologous parts are located between the LinS gene and the origin of replication (2076-2213bp), and between the end of a group of genes promoted by the lac UV5 promoter and the beginning of another group of genes promoted by the trc promoter (9449-9586bp). Despite termination sequences positioned at 2076-2213bp (which might help improving the expression rate of the linalool synthase), their removal is feasible without causing any major harm to cell metabolism, nor compromising linalool synthesis.

Region no.2 measures 134bp and contains the trc promoter and the lac operator, plus an additional non-coding sequence. Its two homologous parts are located before the AgtrGPPS gene (1-134bp) and before the ScMK gene (9724-9856bp). Of course, it is impossible to remove the entirety of this sequence, the promoters being of fundamental importance for correct expression. Nevertheless, it is possible to remove a non-coding homologous region as big as 65bp.

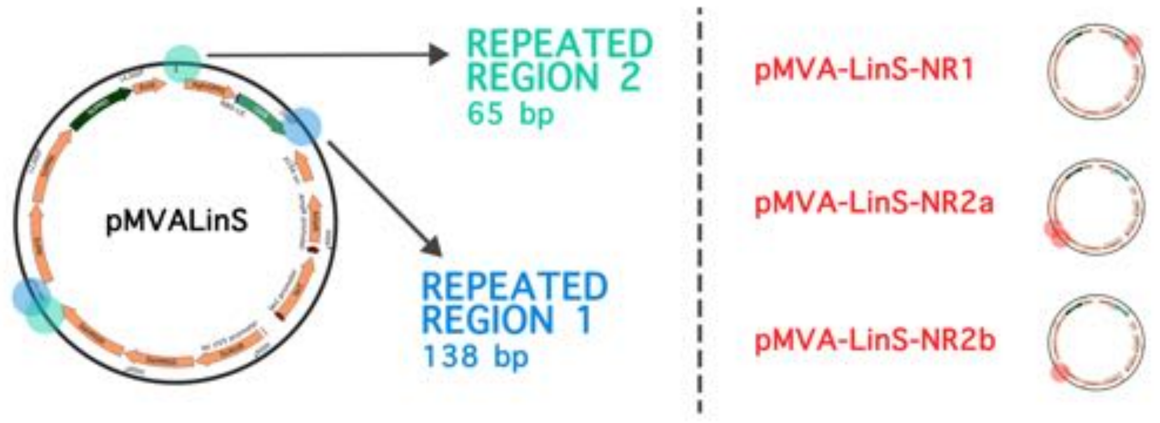


Figure 4.2. *On the left, schematic of the repeated regions. On the right, list of the pMVALinS variations produced with indication of the removed sequences.*

Three variations of pMVALinS have been designed, following the schematic reported in **Figure *4.2***. Removal of the repeated sequence close to the GPP synthase has not been considered yet, because of the higher complexity of gene editing in that region.

Linalool assay has been conducted for a first batch in order to compare production consistency for cells transformed with pMVALinS and pMVALins.NR1. The results reported in **Figure *4.3*** represent 15 biological replicates for each plasmid. It is possible to notice that the majority of the cultures transformed with pMVALinS didn't produce linalool (i.e. ~87%), while all those transformed with pMVALins.NR1 had measurable output. Productivity is very low if compared to previous results, but production repeatability is strongly enhanced.

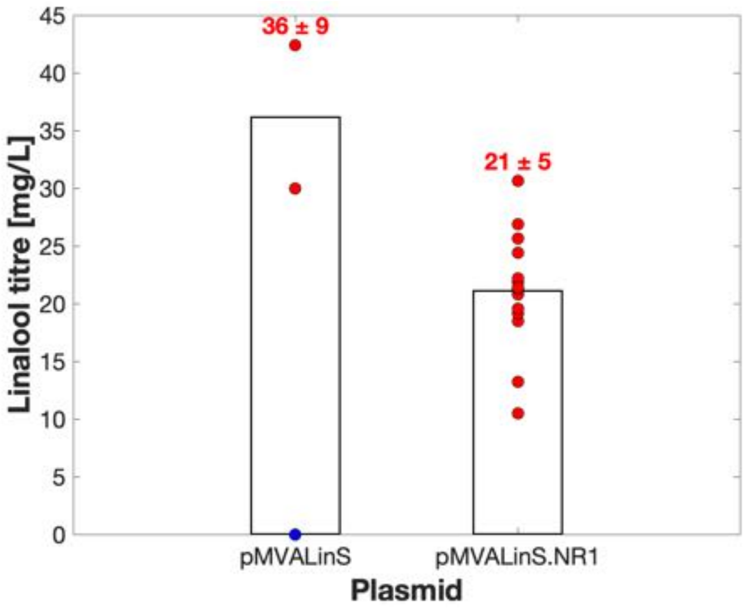


Figure 4.3. *Linalool assay results for pMVALinS – pMVALinS.NR1 comparison (15 biological replicates each). Red dots are non-null titre replicates, blue dots are null titre ones. Averages and standard deviations are calculated on the basis of non-null values.*

A second test has been conducted to compare the productivity of all the designed plasmids and results are reported in **Figure *4.4***. Variants NR2a and NR2b didn't seem to lead to any positive result, but NR1 yielded to production levels comparable to previously obtained data. Moreover, production consistency seems to be retained, as all of the NR1 replicates synthesised linalool.

Clearly, more rounds of testing are needed, but pMVALinS.NR1 pledges to be a promising candidate for obtaining more consistent linalool production levels in the future.

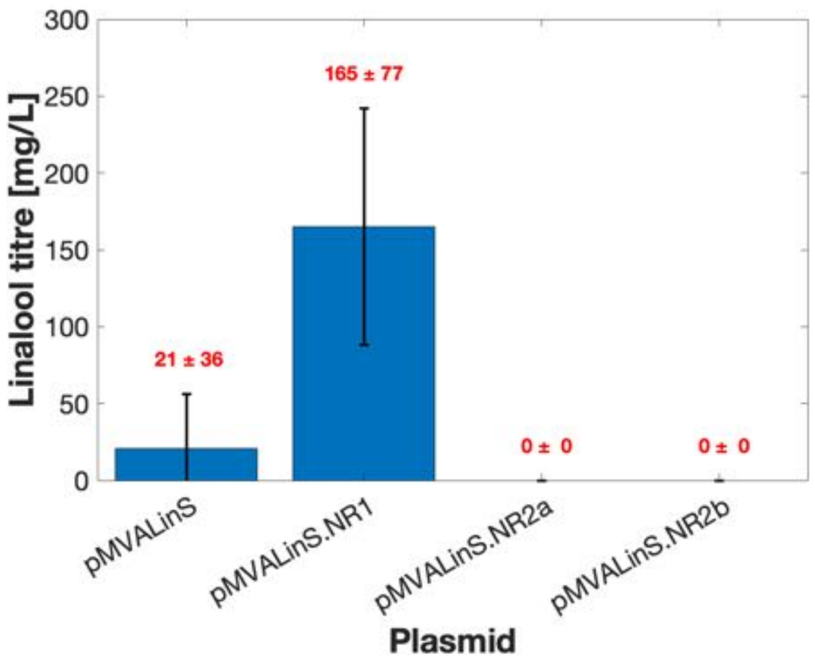


Figure 4.4. *Linalool assay results for pMVALinS – pMVALinS.NR1 – pMVALinS.NR2a – pMVALinS.NR2b comparison. Averages and error bars are calculated over 3 biological replicates.*

4.2. Linalool detection limits evaluation

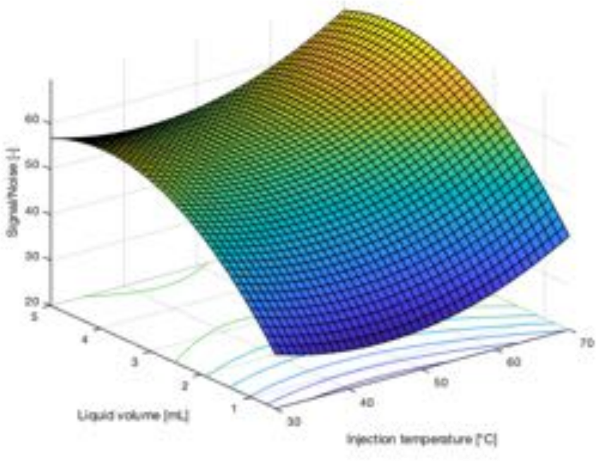
Assessment of the detection limit for linalool was carried out not only keeping in mind the fermentation conditions at which its synthesis is carried out by the previously discussed engineered *E. coli,* but also the experimental setup of the PTR-ToF-MS setup. After an initial adjustment of the GC protocol to allow for clear separation of linalool, tests were conducted varying sample volume, sample temperature, incubation time, injection temperature, and solvent. The analysis is based on the signal/noise ratio, for which a threshold value of 5 represents the minimal detectability of a compound.

It was decided to carry out a DoE to evaluate the effect of 2 quantitative factors and 2 qualitative factors: injection temperature, liquid volume, solvent, and sample incubation, respectively (the last one meant as storage of the sample at a fixed temperature for a specific amount of time prior to its injection into the PTR-ToF-MS). Injection temperature was varied between 30-70°C, a temperature range which was suggested by the operability range of the heated transfer line connecting bioreactor and MS. Liquid volume was varied between 0.5-5mL, the latter value being a safe liquid level for when vials are purged with the autosampler needle. Moreover, it was decided to investigate the effect caused by employing different solvents. Specifically, (i) TB, (ii) nonane, and (iii) TB mixed with nonane, as these media are used in the standard linalool assay (especially the TB-nonane pairing). Lastly, incubation at 30°C for 30 minutes was evaluated, comparing incubated samples against non-incubated ones, as direct sampling of the bioreactor would be taken on temperature-controlled cultures.

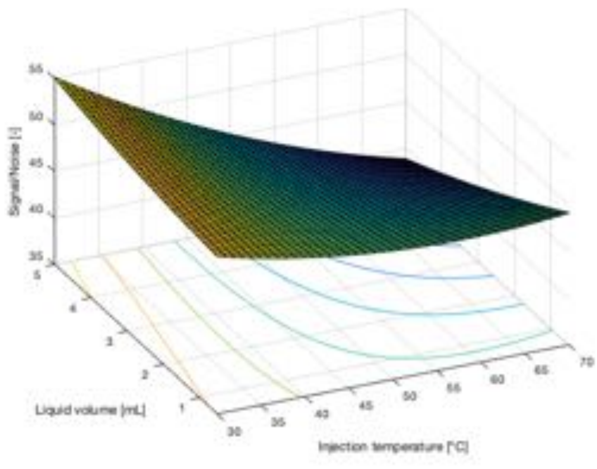
Below are reported the response surfaces for each pair of non-incubated/incubated solvent (see **Figure *4.5***). The general trend is that incubated samples produce a better signal/noise ratio, which is the expected effect. As a matter of fact, incubation at a temperature higher than the room one should push the equilibrium towards the gas phase, therefore having more linalool available in the headspace. Nevertheless, it is possible to notice that there is not a substantial difference between the two conditions in none of the examined solvents.

Highest values of signal/noise ratio were obtained from TB alone, which was strongly influenced by both volume and injection temperature in the non-incubated experimental round, while its response flattened when incubated. Lowest values were shown for TB-nonane mix, indicating that the linalool drawn to the organic layer (high partition coefficient towards nonane) is likely to interact with both phases – the aqueous one below and the gas one above. In this way the amount of linalool passing to the headspace is reduced. Nonetheless, high injection temperature and volume positively influenced detectability, as it is possible to notice a strong increase in signal/noise ratio for the TB-nonane mix at the upper limit conditions of both parameters.

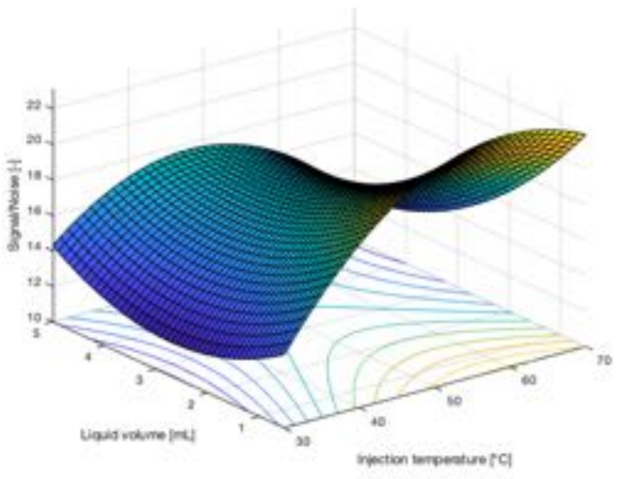
As for pure nonane, neither of the two parameters taken into account had a particularly strong effect on detectability, although higher volumes increased signal/noise ratio values in the incubated run.



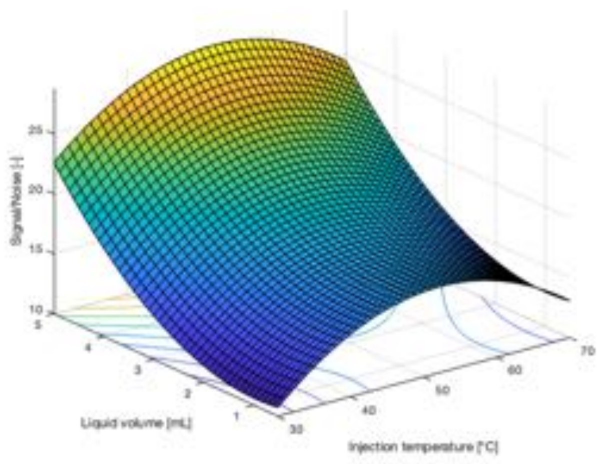
(a)



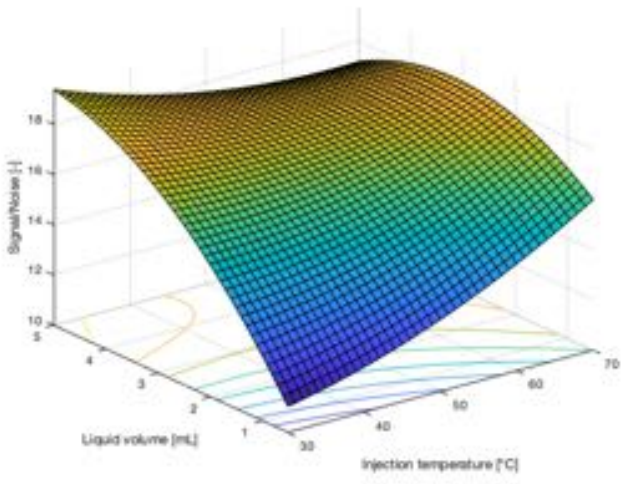
(b)



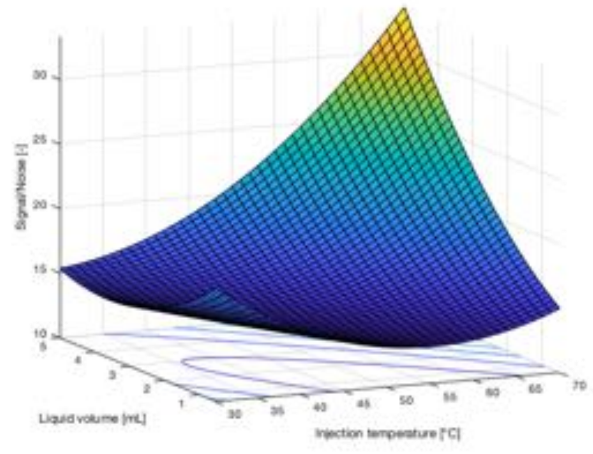
(c)



(d)



(e)



(f)

Figure 4.5. *Response surfaces generated from central composite experiment design. (a) & (b) are non-incubated and incubated TB; (c) & (d) are non-incubated and incubated nonane; (e) & (f) are non-incubated and incubated TB+nonane.*

Given the results of the DoE, it was then assessed the detection limit for linalool in both TB and nonane, using the standard 20mL vials with GC-PTR-ToF-MS, 1h incubation at 30°C, 50°C injection temperature, and 5mL of liquid volume. As reported in **Figure *4.6***, it is possible to see that the detection limit for TB lies around 16ppb, while for nonane is close to 400ppb (corresponding to 14μg/L and 343μg/L, respectively).

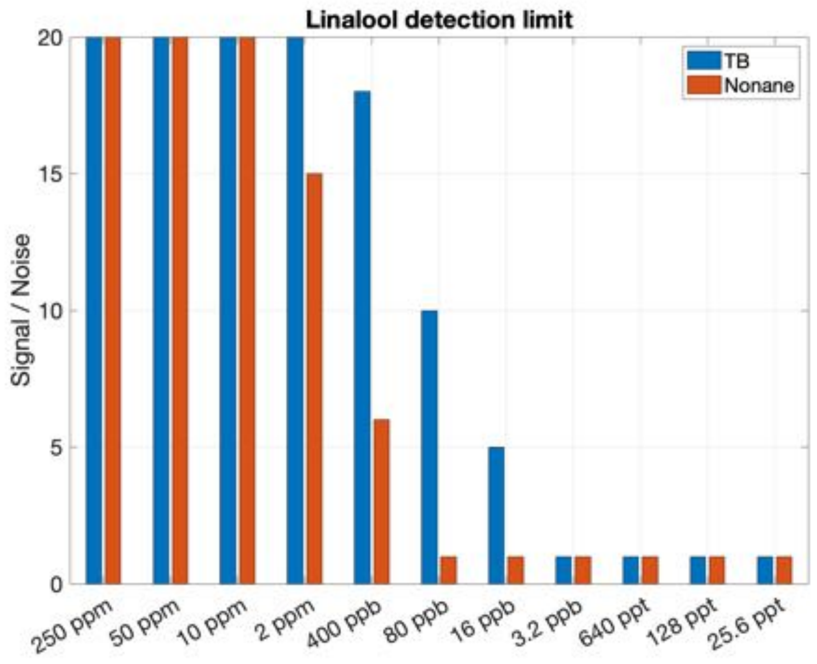
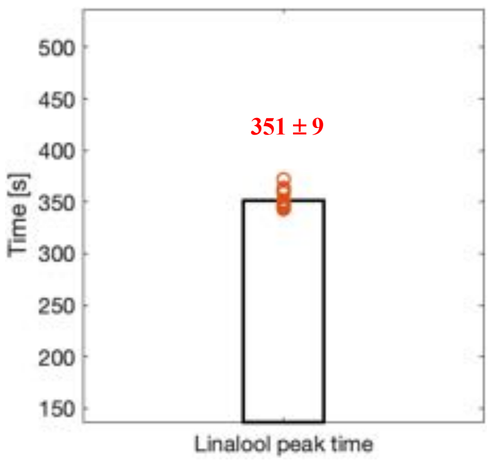
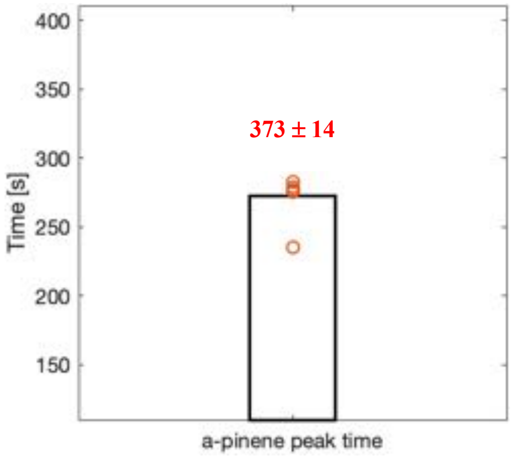


Figure 4.6. *Linalool detection limit in TB and nonane according to GC-PTR-ToF-MS. Samples prepared in 20mL vials, 1h incubation at 30°C, 50°C injection temperature, and 5mL of liquid volume*

Nevertheless, after the execution of these measurements, it has emerged that chromatographic separation seemed to be affected by anomalies. Strong retention time deviations from the expected have been registered not only for linalool, but also for other terpenes (see **Figure *4.7***). In fact, a fault with the main oven control thermocouple has recently been detected and it is currently being fixed.



(a)



(b)

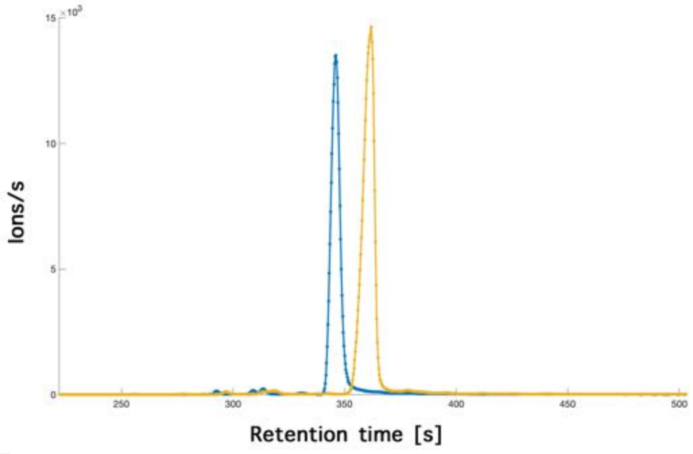


Figure 4.7. *Peak retention time shift. Above: linalool (a) and a-pinene (b) retention time variation for 10 replicates, average value and standard deviation reported. Below: highlight of two replicates of pure linalool, 2μL sample volume, ambient temperature incubation, 37°C injection temperature.*

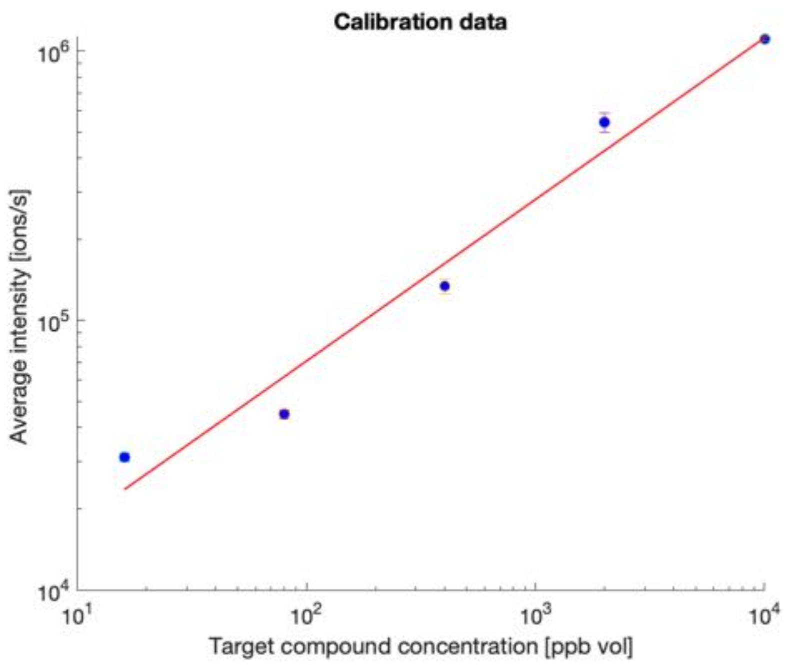
4.3. Platform testing

4.3.1. Calibration

The most fundamental step in testing BioControl operation over a real-time sampling of a bioreactor headspace was the acquisition of reliable linalool calibrations. Initial spiking tests were conducted on 500mL LB media in 1L reactor, set at 37°C, 400rpm stirring, and 0.5L_{air}/L_{media}·min. PTR-ToF-MS zero-gas flowrate was set at 0sccm, so to be as sensitive as possible towards linalool detection. Spiking was carried out by injecting 1mL of linalool in LB at a conveniently chosen concentration, so to bring the titre inside of the reactor to the desired value.

The objective of the calibration is to evaluate both the lower and upper detection limit for the analysis method. As a matter of fact, the sensor of the PTR-ToF-MS presents a physical limit related to the maximum number of ions it can detect at a time (between 10^6-10^7), and when that threshold is overcome the instrument "saturates" and it generates acquisition errors and anomalous output files.

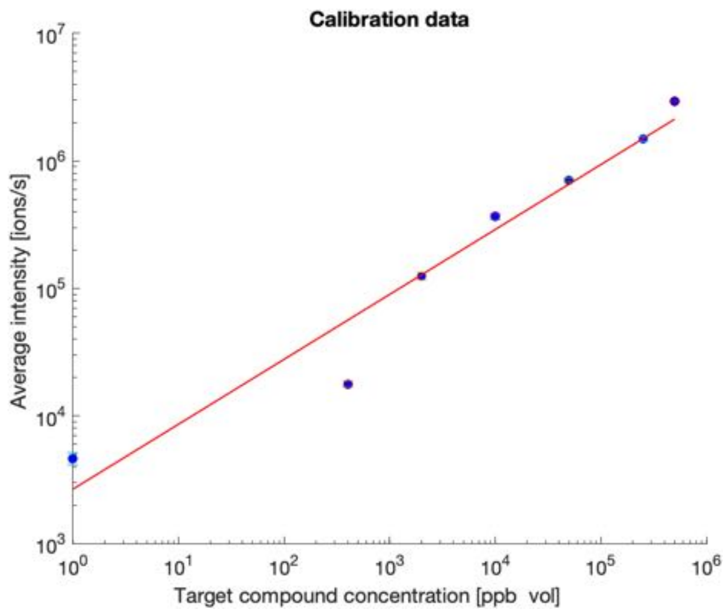
Results of the first tests are reported in **Figure *4.8***, for which values from 16ppb to 10ppm linalool are plotted. Despite leading to a good fit, this calibration is not ideal as it saturates PTR-ToF-MS detector at the highest concentration tested, 10ppm. This value is not ideal as an upper detection limit, as desired detectability range should include the concentrations values that have been observed at small-scale. Since we are expecting concentrations around tens of mg/L_{MEDIA} (i.e. 165 $mg_{LINALOOL}/L_{OVERLAY}$ = 33 $mg_{LINALOOL}/L_{MEDIA}$), an acceptable upper limit would lie around 50ppm at least (though concentrations as high as 200ppm would represent an ideal target).



SLOPE	0.5996
INTERCEPT	3.6508
R^2	0.97

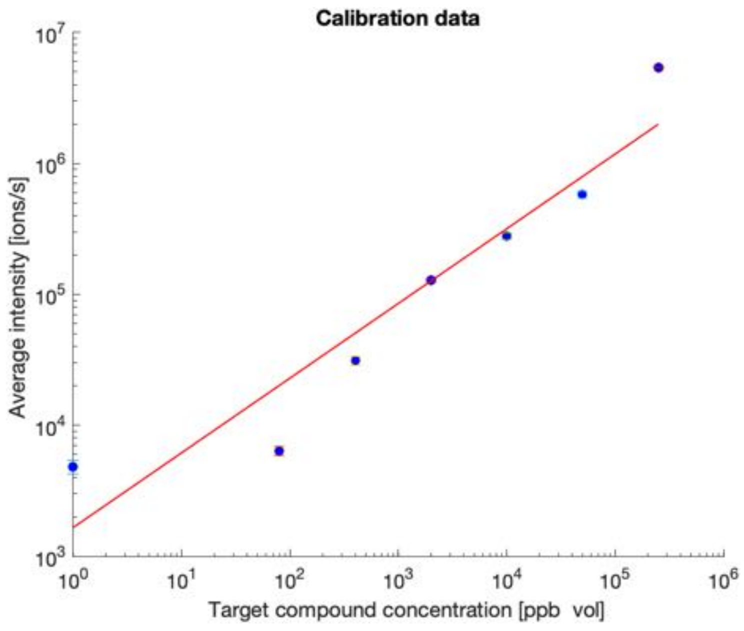
Figure 4.8. *PTR-ToF-MS linalool calibration in 1L bioreactor (conditions replicated in triplicate to obtain each point and corresponding error bar). LB media, 500mL liquid volume, 37°C, 400rpm stirring, 0.5L_{air}/L_{media}·min, 0sccm zero-gas flowrate.*

After some additional tests, it was decided to proceed with identical setup conditions, but operating with a zero-gas flowrate of 40sccm. In the graphs below (**Figure *4.9***) it is possible to see the results for LB, TB, and TB spiked with *E. coli*. It is easy to notice that all of these calibrations show very similar coefficients, leading to Vocus detector saturation around 250-500ppm, which are concentrations substantially higher than the range we are currently expecting to work with. The fact that different media show strong similarity is very positive too, as it gives way for this method to be used with different cultures. Furthermore*, E. coli* growth inside of the reactor didn't appear to hinder linalool detection.



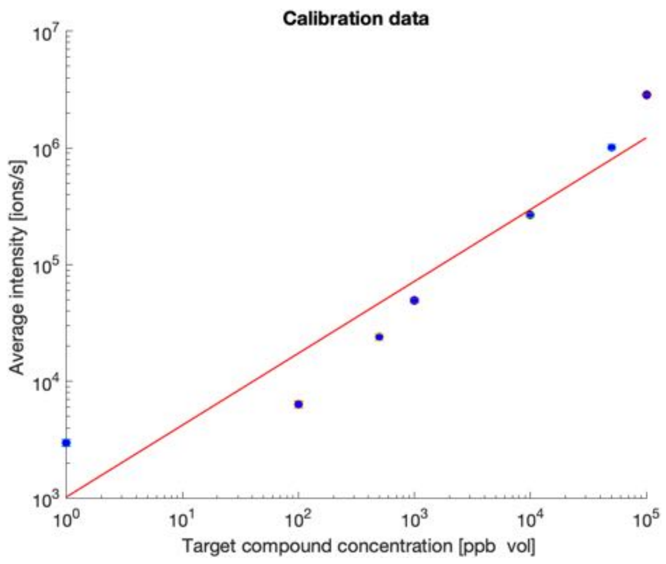
SLOPE	0.5089
INTERCEPT	3.4270
R^2	0.95

(a)



SLOPE	0.5704
INTERCEPT	3.2200
R^2	0.90

(b)

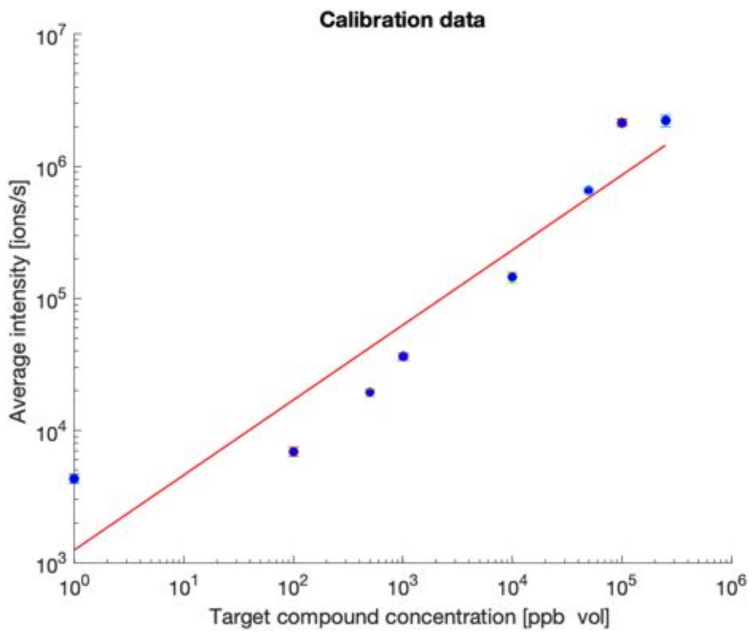


SLOPE	0.6155
INTERCEPT	3.0102
R^2	0.91

(c)

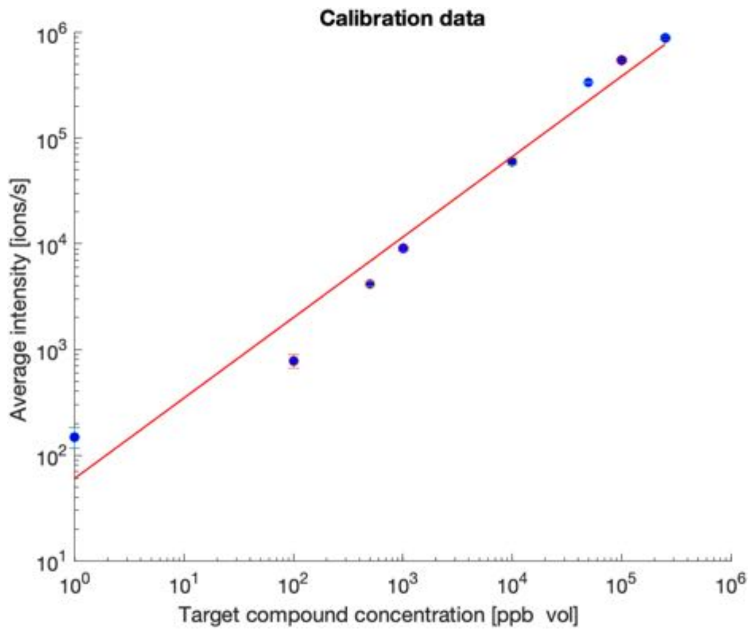
Figure 4.9. *PTR-ToF-MS linalool calibrations in 1L bioreactor (conditions replicated in triplicate to obtain each point and corresponding error bar). 500mL liquid volume, 37°C, 400rpm stirring, 0.5L_{air}/L_{media}·min, 40sccm zero-gas flowrate. (a) calibration in LB media; (b) calibration in TB media; (c) calibration in TB media spiked with* E. coli*.*

Further testing regarded operational conditions of the fermentation. It was decided to test a higher agitation speed (1000rpm) and a lower bioreactor temperature (30°C), since similar assays are executed at these conditions (e.g. linalool standard fermentation protocol comprises 30°C incubation after IPTG induction). Results for these two additional tests are reported in **Figure *4.10***, where it Is possible to notice that while stronger stirring did not lead to substantially different regression coefficients, lower temperature did: calibration line is steeper and intercept is much lower. This is positive, as it implies that in the tested conditions the system has a higher detection sensitivity (higher signal-to-concentration ratio).



SLOPE	0.5674
INTERCEPT	3.0969
R^2	0.90

(a)



SLOPE	0.7611
INTERCEPT	1.7812
R^2	0.97

(b)

Figure 4.10. *PTR-ToF-MS linalool calibrations in 1L bioreactor (conditions replicated in triplicate to obtain each point and corresponding error bar). TB media, 500mL liquid volume, 0.5L_{air}/L_{media}·min, 40sccm zero-gas flowrate. (a) calibration at 1000rpm and 37°C; (b) calibration at 400rpm and 30°C.*

In **Figure *4.11***, it is possible to compare the calibrations discussed above and it is evident that the only parameter strongly influencing them is the bioreactor temperature. This is a positive result, meaning that calibrations are reliable across a range of operational conditions.

Finally, one last calibration was executed with a nonane overlay in the proportion of 1:5 v/v with respect to TB media, which was additionally spiked with *E. coli*. Results reported in **Figure *4.12*** show that no linalool was detected under a threshold of 250ppm, meaning that this measurement method is not compatible with the standard fermentation assay carried out at small-scale currently.

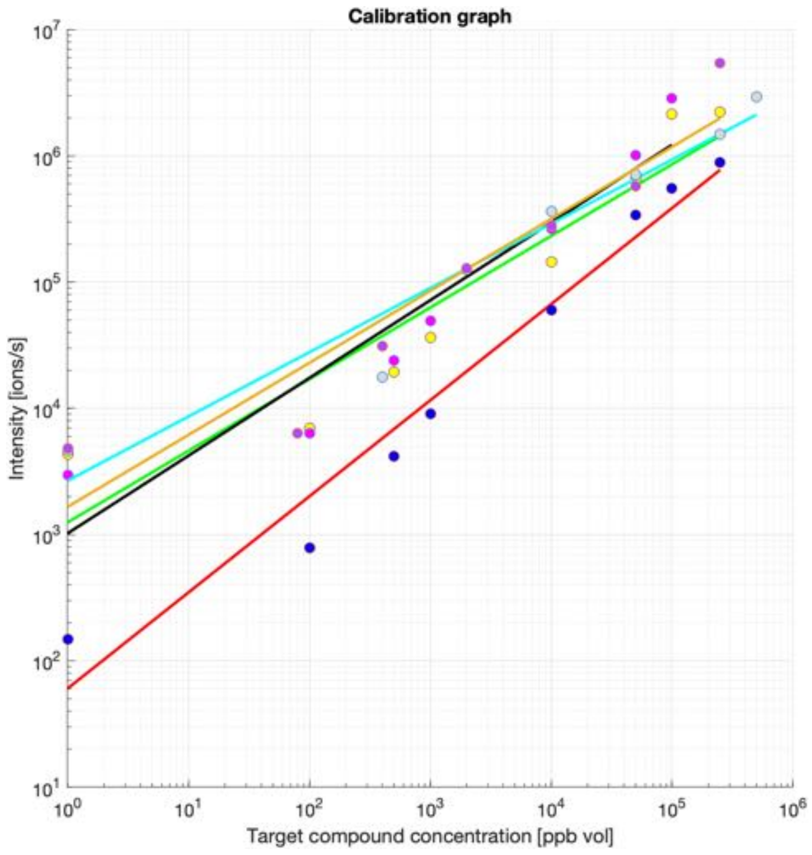


Figure 4.11. *PTR-ToF-MS linalool calibrations in 1L bioreactor (conditions replicated in triplicate to obtain each point and corresponding error bar). 500mL liquid volume, 0.5L_{air}/L_{media}·min, 40sccm zero-gas flowrate. (cyan line) LB media, 400rpm and 37°C; (orange line) TB media, 400rpm and 37°C; (green line) TB media, 1000rpm and 37°C; (black line) TB media spiked with* E. coli*, 400rpm and 37°C; (red line) TB media, 400rpm and 30°C.*

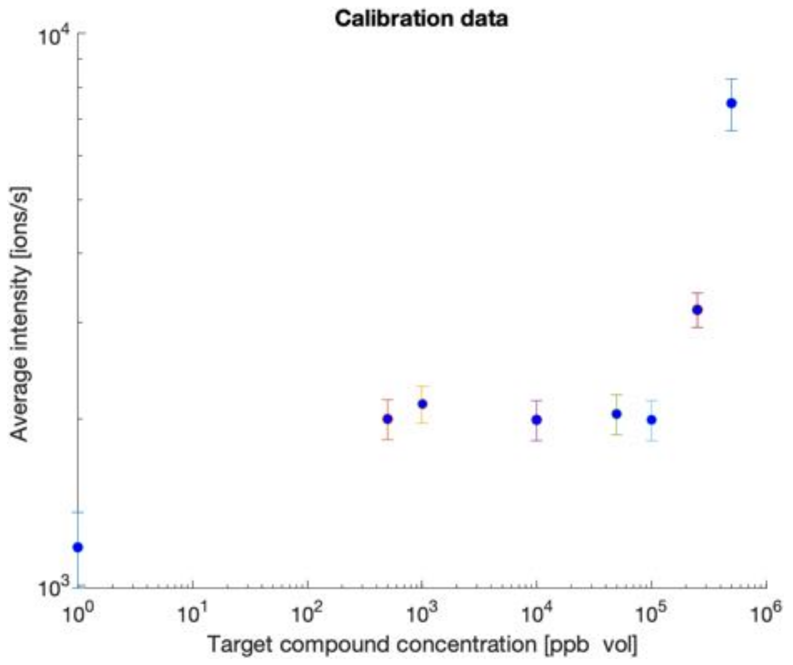


Figure 4.12. *PTR-ToF-MS linalool calibrations in 1L bioreactor (conditions replicated in triplicate to obtain each point and related error bar). TB media spiked with* E. coli*, 1:5 v/v nonane overlay, 500mL liquid volume, 37°C, 400rpm stirring, 0.5L$_{air}$/L$_{media}$·min, 40sccm zero-gas flowrate.*

4.3.2. Scaled-up linalool fermentation

Once BioControl had been successfully tested against mockfiles and suitable calibrations had been obtained, it was used to monitor a scaled-up linalool fermentation assay. Enhanced consistency strain transformed with previously descripted pMVALinS.NR1 was used to scale up fermentation to 500mL, without the use of any nonane overlay – as it was observed that would have strongly compromised linalool detection. BioControl-acquired results are reported in **Figure *4.13*** and represent a 30h fermentation assay.

The general outlook of the test is extremely positive, as linalool was successfully detected over the whole fermentation time and GC-MS controls (red dots in the graph) conducted on culture broth samples confirm that the detected concentrations are correct. Moreover, despite small local oscillations, the obtained profile is extremely consistent and not prone to particularly strong variations (blue line represents a 2-minutes-rolling average, which is not a large window of time). Furthermore, the acquisition provides useful insight on the real-time profile of the fermentation, which can be used to optimise the standard assay (e.g. a fermentation time reduction will be considered for future experiments).

In addition, concentrations achieved in this experiment are very promising and in line with previous results. Peak concentration was around 38mg/L$_{MEDIA}$, which would be the equivalent of

190mg/$L_{OVERLAY}$, as no nonane overlay was used. This is very promising, especially because of the absence of any organic overlay. As a matter of fact, Its purpose is to avoid linalool accumulation in the culture media, which would have cytotoxic effects on cells. Nevertheless, the lack of nonane did not seem to compromise the fermentation process significantly.

On the other hand, it is possible to notice how after 24h of fermentation, linalool concentration steeply decreased. Although no definitive conclusion has been drawn yet, it was confirmed that after 24h no linalool-producing plasmid was retained inside of the cells. Nevertheless, this isn't enough to explain linalool depletion. A possible reason for this would be linalool stripping caused by air bubbling. As a matter of fact, in order to avoid pressure build-up inside of the reactor due to compressed air being pumped inside of it, a ventilation outlet is left open. Despite the presence of a 5°C water condenser on the outlet, it could be that droplets of linalool were stripped away from the culture media. This could be problematic, as it would imply that also during the whole operation time linalool had been constantly removed from the bioreactor.

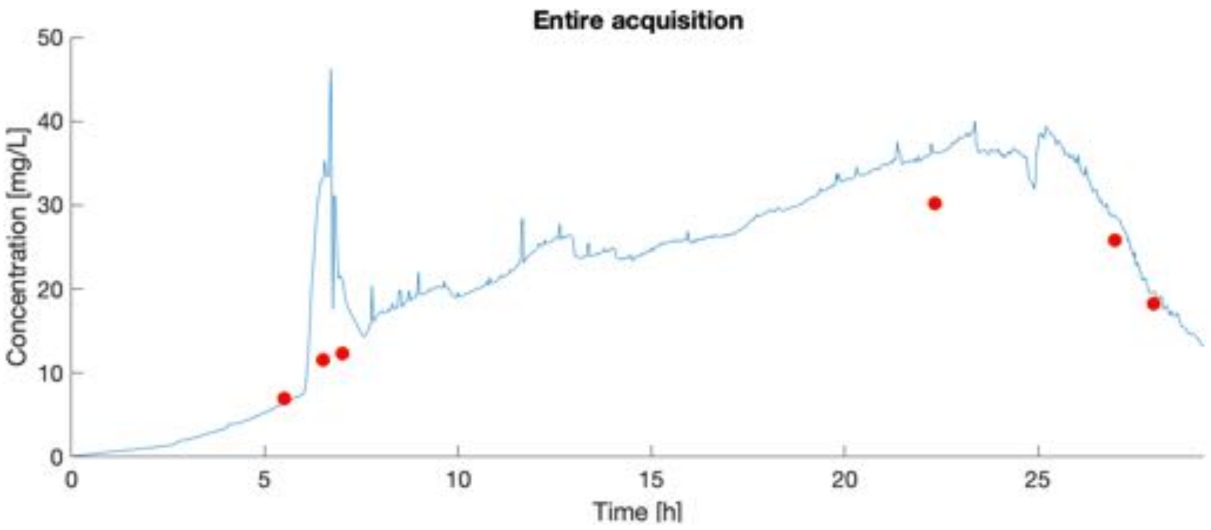


Figure 4.13. *BioControl-acquired linalool concentration profile over a 30h fermentation assay carried out in 1L bioreactor using pMVALinS.NR1 plasmid in* E. coli *chassis. Red dots represent GC-MS controls. Anomalous concentration spike around 6-7h was due to broth droplets accumulation in the PTR-ToF-MS transfer line, which was purged restoring measurement accuracy.*

4.4. Control loop simulation

4.4.1. System response modelling

Since the aim of this work is to lay the foundations for future investigations on fine control of microbial systems, the feasibility of an optogenetics-based control routine implementation on linalool production has been assessed by software simulation.

To evaluate control efficacy and stability it is necessary to design a closed-loop system, define the necessary transfer functions (TFs), and finally tune the controller element. In **Figure *4.14***, the closed-loop system employed in this work is reported. It was chosen to adopt a simple feedback control routine, based on the deviation from a fixed setpoint value of the controlled variable. Controlled variable was linalool productivity, while manipulated variable was light intensity (since we are assuming to simulate linalool production by means of an optogenetics-regulated biopathway), whose value ranges from 0 to 1 (i.e. dark condition vs. highest light intensity).

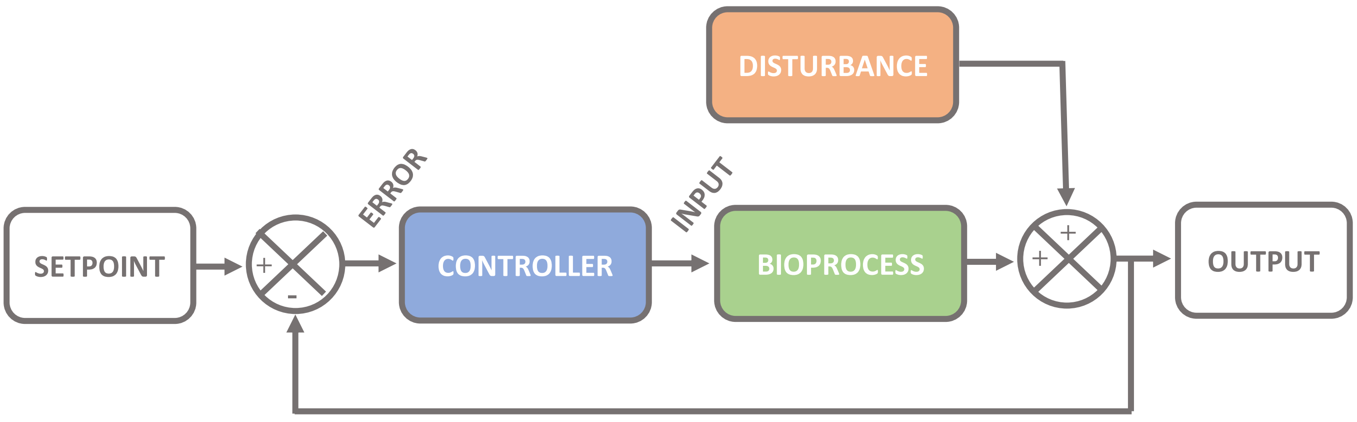


Figure 4.14. *Feedback control loop schematic. Output corresponds to the controlled variable (linalool productivity), which can be influenced by some external disturbance. The value of the controlled variable is assessed and compared to a user-defined setpoint. Deviation from that value is sent to the controller, which elaborates an adequate manipulated variable (light intensity) input, so to bring the bioprocess to the desired state.*

Since optogenetics-controlled biosynthetic pathways are not available yet, the necessary data to extrapolate the biological system's TF were obtained from standard small-scale batch fermentation experiments. Linalool concentration was measured at several time points through GC-MS analysis, so to map production profile over time. It was assumed that IPTG induction would have quite ideally approximated a light intensity step change from 0 to 1, since both induce expression of linalool metabolic pathway.

Additionally, batch fermentation doesn't involve substrate make-up to the culture, therefore substrate consumption must be taken into account. In other words, the dynamic of the system is not merely given by the effect of the IPTG induction, because substrate consumption plays a fundamental role too. As a matter of fact, linalool productivity increases due to the addition of IPTG, but it is also held back by the progressive reduction of available substrate. Consequently, it was assumed that two different effects would have contributed to the studied system's behaviour: (i) productivity increase due to IPTG induction, and (ii) productivity decrease due to decreasing substrate availability. Both of these effects would have simultaneously taken place since the beginning of the fermentation assay, nevertheless our interest was to test feasibility over a semi-batch system, where a control routine would generate the most significant benefit. Therefore, fermentation data were used to regress a TF model including both effects, the first one of which would have been used to test control stability later.

The transfer function that was used to determine the model coefficients by regression is reported in **Equation 4.1**:

$$G_P = \frac{Y'(s)}{X'(s)} = \frac{K}{\tau_1 \cdot s + 1} - \frac{K}{\tau_2 \cdot s + 1} \tag{4.1}$$

where G_P stands for transfer function of the process, $\frac{Y\prime(s)}{X\prime(s)}$ is the deviation of the controlled variable $Y'(s)$ (i.e. linalool productivity) due to the change of the manipulated variable $X'(s)$ (i.e. light intensity ≅ IPTG induction) in the Laplacian domain, K is the process gain (assumed to be equal and of opposite sign in the two effects, as their combination would ultimately lead to 0mg/L·h productivity within the end of the fermentation assay, due to substrate consumption), τ_1 is the time constant of the first effect (i.e. productivity increase due to induction), and τ_2 is the time constant of the second effect (i.e. productivity decrease due to substrate depletion). The two time constants are assumed to have different values, the first one being much smaller with respect to the second one, as productivity increase due to induction is expected to be a much quicker effect than productivity decrease due to substrate consumption, likely to be abundant during the first part of the fermentation.

In order to regress the coefficients, experimental data were fitted with the time domain transposed function reported in **Equation 4.2**:

$$y(t) = m \cdot K\left[\left(1 - e^{-t/\tau_2}\right) - \left(1 - e^{-t/\tau_1}\right)\right] \tag{4.2}$$

where $y(t)$ is the linalool productivity profile over time, and m is the change of the manipulated variable (= 1, as it is assumed that it is a step change from dark conditions to maximum intensity light).

Simple fitting of this function based on least sum of squared errors would not produce physically feasible results, as the value of K would be pushed as high as possible. Since K represents the maximum level of productivity the system would achieve if substrate was constantly replenished, values higher than 30-40mg/L·h are likely to be strongly unphysical. Moreover, the two time constants would likely assume very similar if not identical values if not bounded by a constraint.

Therefore, a second criterion was added to the least sum of squared error to optimise coefficient regression. Optimum search had to stop once variation in the sum of squared error reached a value < 1% of the last figure. In addition to this, τ_2 was modelled so to represent τ_1 plus an additional positive value, so to push τ_2 value to be larger than τ_1. The modified function is reported in **Equation 4.3**:

$$y(t) = m \cdot K\left[\left(1 - e^{-t/\tau_1+\tau_A}\right) - \left(1 - e^{-t/\tau_1}\right)\right] \tag{4.3}$$

where τ_A is the additional term to be added to τ_1 to obtain $\tau_2 = \tau_1 + \tau_A$. In this way, number of coefficients to be regressed did not increase, while helping to push the regression model towards more physical values. Boundaries on both τ_1 and τ_A were set to [500, 4500] min, range that includes values that are compatible with the process to be modelled. K upper boundary was increased step by step with the iterations of the fitting function, starting from a value of 5 mgL^{-1}h^{-1}l^{-1}.

Using this approach, values within a reasonable range were obtained for K, τ_1, τ_2, while guaranteeing a good fit of the experimental data (reported in **Figure *4.15*** and **Table *4.1***).

Table 4.1. *Results of the TF model fitting employing* ***Equation 4.3****. Process gain is expressed in productivity over light intensity.*

K [mgL^{-1}h^{-1}l^{-1}]	23
τ_1 [min]	1220
τ_A [min]	857
τ_2 [min]	3077
SSE [mgL^{-1}h^{-1}]	9.6

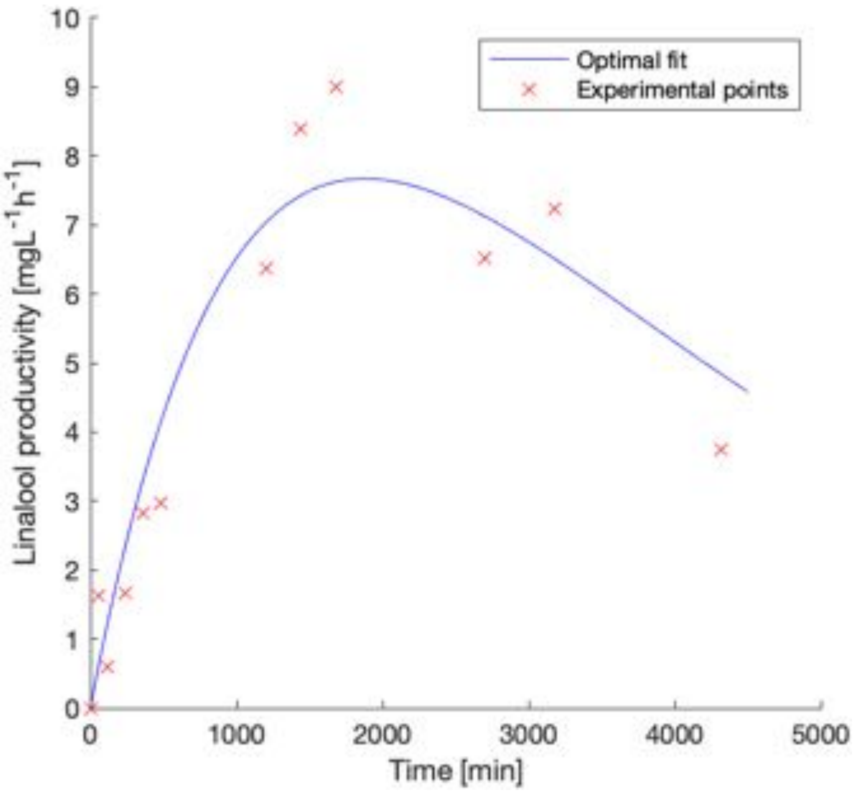


Figure 4.15. *Fitting of TF model for the biological system. Red crosses represent experimental points for linalool productivity obtained by GC-MS analysis (each point is the average of three biological replicates). Blue line is the fitted productivity profile obtained from* ***Equation 4.3****.*

4.4.2. PID tuning and stability analysis

Once having obtained sensible parameters for the bioprocess TF, it was decided to use them to tune a PID controller. As mentioned before, only the positive effect of the modelled TF has been considered, as the desired process to be modelled would be a fed-batch with constant substrate make up.

A Simulink representation of the closed loop system reported in **Figure *4.14*** has been designed and MATLAB tool for automatic PID tuning has been employed (considering a parallel PID configuration). PID controller transfer function is reported in **Equation 4.4**,

$$G_C = \frac{I'(s)}{e'(s)} = K_p + \frac{K_i}{s} + \frac{K_d \cdot s}{\tau_f \cdot s + 1} \qquad (4.4)$$

where G_C stands for the transfer function of the control element, $\frac{I'(s)}{e'(s)}$ is the controller output (i.e. bioprocess input) value over the controlled variable error in the Laplacian domain, K_p is the proportional gain, K_i is the integrator gain, K_d is the derivative gain, and τ_f is derivative filter time.

Standard PID formulation doesn't include the denominator on the derivative term as reported in **Equation 4.4** (i.e. $\tau_f \cdot s + 1$), which is called derivative filter. It was decided that a filter would have been appropriate, given the nature of the system in analysis. As a matter of fact, the derivative term of a PID contributes to reducing the settling time (i.e. the time it takes to reach steady state). Nevertheless, in case of noisy measurement of the controlled variable, the derivative term can easily generate of oscillations (Seborg et al., 2011). Therefore, since the measurement of linalool productivity through PTR-ToF-MS is prone to noise, the derivative filter was included, consisting of a simple first order dynamics. Tuned parameters are reported in **Table *4.2***.

Table 4.2. *Tuned coefficients for PID controller.*

K_p [mg^{-1}L h l]	27.7036
K_i [mg^{-1}L h l]	2.6615
K_d [mg^{-1}L h l]	-12.6155
τ_f [min]	1.4062

Final step was testing stability of the control routine over the tuned parameters. It was verified following Nyquist stability criterion (Seborg et al., 2011), which states that a closed-loop control system is stable if all of the poles of its equation are negative or have negative real part. The generic term for the closed loop equation of the system in analysis is reported in **Equation 4.5**:

$$Y = \frac{G_C G_P}{1 + G_{OL}} Y_{SP} + \frac{G_D}{1 + G_{OL}} D \tag{4.5}$$

Where Y is the controlled variable, Y_{SP} is the setpoint change, G_D is the transfer function of the disturbances entering the system, D is the disturbance, and G_{OL} is the open loop transfer function (being $G_{OL} = G_C G_P$). Disturbances have not been investigated specifically and they have been included only for completeness. Neither their origin (i.e. the variable affected by unexpected variation) nor their impact (i.e. the transfer function that describes their effect on the controlled variable) is known, therefore they have not been included in the formal stability analysis, leading to the equation reported in **Equation 4.6**:

$$\frac{Y}{Y_{SP}}=\frac{G_D}{1+G_{OL}}=\frac{G_C G_P}{1+G_C G_P}=\frac{\left(K_p+\frac{K_i}{s}+\frac{K_d\cdot s}{\tau_f\cdot s+1}\right)\left(\frac{K}{\tau_1\cdot s+1}-\frac{K}{\tau_2\cdot s+1}\right)}{1+\left(K_p+\frac{K_i}{s}+\frac{K_d\cdot s}{\tau_f\cdot s+1}\right)\left(\frac{K}{\tau_1\cdot s+1}-\frac{K}{\tau_2\cdot s+1}\right)} \qquad (4.6)$$

To verify its poles are negative, it is sufficient to verify that the characteristic equation (i.e. $1+G_{OL}$, the denominator) has negative roots. In order to do this, the Routh-Hurwitz stability criterion is employed, which states that a necessary and sufficient condition for all roots of the characteristic equation of a system to have negative real parts is that all of the elements in the left column of the Routh array are positive (Seborg et al., 2011). The stability criterion is based on a characteristic equation that has the form of:

$$a_n s^n + a_{n-1}s^{n-1} + a_{n-2}s^{n-2} + \cdots + a_1 s + a_0 = 0 \qquad (4.7)$$

Rearranging the denominator in **Equation 4.6**, we obtain the expression reported in **Equation 4.8**:

$$\begin{aligned}
&s^4\left(\tau_f\tau_1\tau_2\right)+\cdots\\
&s^3\left(\tau_1\tau_2+\tau_f(\tau_1+\tau_2)+K\left(K_p\tau_f+K_d\right)(\tau_2-\tau_1)\right)+\cdots\\
&s^2\left(\tau_1+\tau_2+\tau_f+K\left(K_p+K_i\tau_f\right)(\tau_2-\tau_1)\right)+\cdots\\
&s(1+KK_i(\tau_2-\tau_1))>0
\end{aligned} \qquad (4.8)$$

Therefore, it is possible to define the coefficients according to the desire form of the equation, as in **Equation 4.9** below:

$$\begin{aligned}
&a_4=\tau_f\tau_1\tau_2\\
&a_3=\tau_1\tau_2+\tau_f(\tau_1+\tau_2)+K\left(K_p\tau_f+K_d\right)(\tau_2-\tau_1)\\
&a_2=\tau_1+\tau_2+\tau_f+K\left(K_p+K_i\tau_f\right)(\tau_2-\tau_1)\\
&a_1=1+KK_i(\tau_2-\tau_1)
\end{aligned} \qquad (4.9)$$

Then, the Routh array is built, following the structure reported in **Figure *4.16***:

$$
\begin{array}{c|llll}
\textit{Row} & & & & \\
1 & a_n & a_{n-2} & a_{n-4} & \cdots \\
2 & a_{n-1} & a_{n-3} & a_{n-5} & \cdots \\
3 & b_1 & b_2 & b_3 & \cdots \\
4 & c_1 & c_2 & \cdots & \\
\vdots & \vdots & & & \\
n+1 & z_1 & & &
\end{array}
$$

$$b_1 = \frac{a_{n-1}a_{n-2} - a_n a_{n-3}}{a_{n-1}}$$

$$b_2 = \frac{a_{n-1}a_{n-4} - a_n a_{n-5}}{a_{n-1}}$$

$$\vdots$$

$$c_1 = \frac{b_1 a_{n-3} - a_{n-1} b_2}{b_1}$$

$$c_2 = \frac{b_1 a_{n-5} - a_{n-1} b_3}{b_1}$$

$$\vdots$$

Figure 4.16. *Structure of the Routh array and terms calculation rule (Seborg et al., 2011).*

Substituting all the coefficients (i.e. K, $\tau_1, \tau_2, K_p, K_i, K_d, \tau_f$), the following values are obtained for the left column of the array:

Table 4.3. *Left column of Routh array value for the system in exam.*

a_4	5278790.428
a_3	4885045.805
b_1	1224558.240
c_1	113676.326
d_1	0

As all of the values are positive, the system is then assumed stable.

Additionally, an empirical verification of the stability was carried out by simulating the response of the feedback system in Matlab over a servo control scenario (**Figure *4.17***). To assess the stability of the system, a sensitivity analysis was run on the coefficients, allowing a random ±20% maximum variation for each single parameter. A total of 1000 cases were generated and are below illustrated. It is possible to notice how, despite substantial deviation from optimal tuning conditions, the controlled variable responds in a stable and timely manner. As a matter of fact, upon a unit step change in setpoint (from 0 to 1, equal to darkness to light condition), the new value of the controlled variable is reached within 30min, with a 20% overshoot for the worst-case scenario.

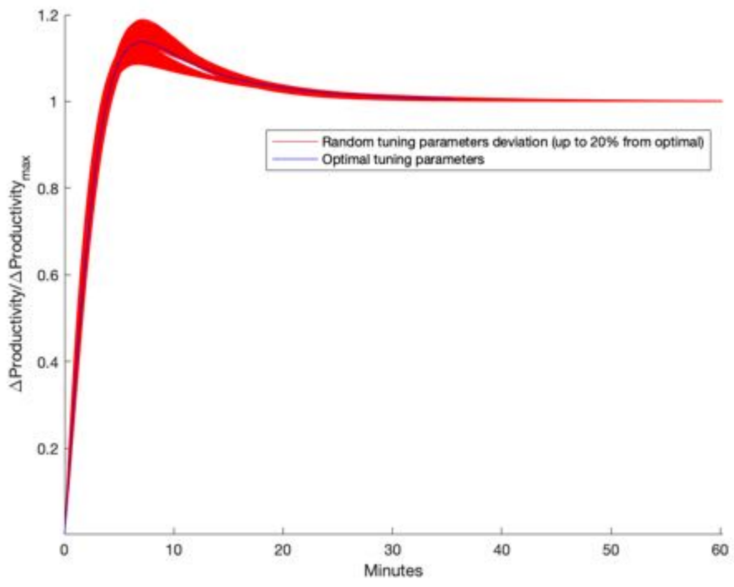


Figure 4.17. *Tuning parameters sensitivity test with 1000 random scenarios varying each parameter up to ±20% from optimal value.*

4.5. Summary

In this chapter the results of the different experiments and simulations are reported. Firstly, the design of the plasmid used for the production of linalool is addressed. Two homologous regions are highlighted in the plasmid and 3 alternative designs removing either one or both are investigated. Design NR1 is showed to improve production consistency, though at the same time yielding lower concentrations with respect to the average of previous testing.

Subsequently, sensitivity of PTR-ToF-MS towards linalool detection is tested against parameters like injection temperature, sample volume, solvent type and sample incubation (i.e. sample stabilisation at a fixed temperature before injection). Results show that in general higher injection temperature and sample incubation has a positive effect on signal intensity. TB media leads to noisier measurement than average, but at same time to the best detectability limit (14ppb).

Calibration of PTR-ToF-MS for linalool measurement in bioreactors is then addressed, considering multiple parameters. Calibrations with different conditions show very similar slope/intercept, indicating that the measurement method is stable and reliable across a range of operations. Moreover, it is highlighted that nonane overlay is not suitable for this type of setup as it strongly reduces the sensitivity range.

Next, real process monitoring through BioControl is tested over standard linalool fermentation assay. By comparison against standard GC-MS quantification at various time points, it is proved that the measurement of the monoterpene not only is feasible, but also accurate.

Eventually, simulation of the control loop to maintain linalool productivity is carried out. Block diagram of the closed-loop system is designed. Bioprocess transfer function is defined and its parameters are regressed. Lastly, a PID control element is used to simulate a feedback control of the system. After tuning, its stability is verified both theoretically and empirically.

5. Conclusions and future work

Microbial synthesis of linalool has been analysed with the perspective of integrating its productive process with control techniques based on PTR-ToF-MS headspace analysis.

Firstly, process consistency has been investigated, observing high variability in linalool titres. The presence of repeated sequences in the plasmid used to transform *E. coli* cells was suspected as a potential cause for production variability, as such configuration could cause homologous recombination. Three plasmid alternatives were designed and tested, amongst which one showed strong repeatability enhancement, despite not reaching the same productivity levels attained by previous experiments.

Then PTR-ToF-MS technology has been used to perform linalool headspace analysis on sample vials. The optimisation of conditions such as sample volume, injection temperature, and solvent allowed to perform successful linalool detection at levels far below regular GC-MS analysis and more than adequate to track linalool in batch fermentation assays.

An original software has been designed to operate as a monitoring platform for fermentative processes: data from PTR-ToF-MS headspace analysis are imported and interpreted, so to allow for live monitoring of a microbial culture batch; this information can then be used to implement control strategies to optimise productivity.

Lastly, feasibility of the implementation of control routines on microbial synthesis of linalool has been assessed through simulation of closed loop system response, whose dynamic has been modelled over experimental data. Stability of the control routine has been confirmed by means of PID tuning parameters sensitivity analysis.

Future work will focus on the execution of further tests for the optimisation of the modified *E. coli* strain: additional alternative plasmids without homologous regions will be designed and tested, so to guarantee a more stable production of linalool. Subsequently, optogenetic control will be added by means of a second plasmid, allowing for an externally triggered expression of linalool synthase with a light-induced genetic switch.

The control platform will be expanded so to feature the possibility to modulate an external light switch on the basis of the control strategy adopted. Eventually, different control routines will be designed and tested, aiming at a more consistent linalool production over time, along with a more balanced biological burden on microbial cells.

Bibliography

Alford, J. S. (2006). Bioprocess control: Advances and challenges. *Computers & Chemical Engineering, 30*(10-12), 1464-1475. doi:10.1016/j.compchemeng.2006.05.039

Aoki, S. K., Lillacci, G., Gupta, A., Baumschlager, A., Schweingruber, D., & Khammash, M. (2019). A universal biomolecular integral feedback controller for robust perfect adaptation. *Nature, 570* (7762), 533-537. doi:10.1038/s41586-019-1321-1

Ashour, M., Wink, M., & Gershenzon, J. (2018). Biochemistry of Terpenoids: Monoterpenes, Sesquiterpenes and Diterpenes. In *Annual Plant Reviews online* (pp. 258-303).

Atkinson, M. R. S., Savageau, M. A., Myers, J. T., Ninfa, A. J. (2003). Development of Genetic Circuitry Exhibiting Toggle Switch or Oscillatory Behavior in Escherichia coli. *Cell, 113*, 597-607. doi:https://doi.org/10.1016/S0092-8674(03)00346-5

Bacchus, W., & Fussenegger, M. (2012). The use of light for engineered control and reprogramming of cellular functions. *Current Opinion in Biotechnology, 23* (5), 695-702. doi:10.1016/j.copbio.2011.12.004

Becksei, A., & Serrano, L. (2000). Engineering stability in gene networks by autoregulation. *Nature, 405*, 590-593.

Behr, A., & Johnen, L. (2009). Myrcene as a natural base chemical in sustainable chemistry: a critical review. *ChemSusChem, 2*(12), 1072-1095. doi:10.1002/cssc.200900186

Benozzi, E., Romano, A., Capozzi, V., Makhoul, S., Cappellin, L., Khomenko, I., . . . Biasioli, F. (2015). Monitoring of lactic fermentation driven by different starter cultures via direct injection mass spectrometric analysis of flavour-related volatile compounds. *Food Research International, 76*(Pt 3), 682-688. doi:10.1016/j.foodres.2015.07.043

Bian, G., Deng, Z., Liu, T. (2017). Strategies for terpenoid overproduction and new terpenoid discovery. *Current Opinion in Biotechnology, 48*, 234-241. doi:10.1016/j.copbio.2017.07.002

Biodiesel. (2019). Retrieved from https://www.olleco.co.uk/green-fuels/biodiesel

Blake, R. S. M., Paul S., Ellis, Andrew M. (2009). Proton-Transfer Reaction Mass Spectrometry. *Chemical Reviews, 109* (3), 861-896. doi:doi:10.1021/cr800364q

Cao, X., Wei, L. J., Lin, J. Y., Hua, Q. (2017). Enhancing linalool production by engineering oleaginous yeast Yarrowia lipolytica. *Bioresource Technology, 245*(Pt B), 1641-1644. doi:10.1016/j.biortech.2017.06.105

Chauvatcharin, S. S., T., Fujiyama, K., Yoshida, T. (1995). On-Line Monitoring and Control of Acetone-Butanol Fermentation Membrane-Sensor Mass Spectrometry. *Journal of Fermentation and Bioengineering, 79*, 264-269. doi:doi:10.1016/0922-338X(95)90614-6

Chen, S., Harrigan, P., Heineike, B., Stewart-Ornstein, J., & El-Samad, H. (2013). Building robust functionality in synthetic circuits using engineered feedback regulation. *Current Opinion in Biotechnology, 24*(4), 790-796. doi:10.1016/j.copbio.2013.02.025

Clomburg, J. M., Qian, S., Tan, Z., Cheong, S., & Gonzalez, R. (2019). The isoprenoid alcohol pathway, a synthetic route for isoprenoid biosynthesis. *Proceedings of the National Academy of Science U S A, 116*(26), 12810-12815. doi:10.1073/pnas.1821004116

De Mena, L., Rizk, P., Rincon-Limas, D. E. (2018). Bringing Light to Transcription: The Optogenetics Repertoire. *Frontiers in Genetics, 9*, 518. doi:10.3389/fgene.2018.00518

Dudareva, N., Negre, F., Nagegowda, D. A., & Orlova, I. (2006). Plant Volatiles: Recent Advances and Future Perspectives. *Critical Reviews in Plant Sciences, 25*(5), 417-440. doi:10.1080/07352680600899973

Fernández-Naveira, Á., Veiga, M. C. Kennes, C. (2017). Effect of pH control on the anaerobic H-B-E fermentation of syngas in bioreactors. *Journal of Chemical Technology & Biotechnology, 92*(6), 1178-1185. doi:10.1002/jctb.5232

Ferreira, L. S., De Souza Jr., M. B., Folly, R. O. M. (2001). Development of an alcohol fermentation control system based on biosensor measurements interpreted by neural networks. *Sensors and Actuators, 75*, 166-171. doi:doi:10.1016/S0925-4005(01)00540-8

Gao, Y., Honzatko, R. B., Peters, R. J. (2012). Terpenoid synthase structures: a so far incomplete view of complex catalysis. Preceedings of the National Accademy of Sciences. *Nat Prod Rep, 29*(10), 1153-1175. doi:10.1039/c2np20059g

Gardner, T. S., Cantor, C. R., Collins, J. J. (2000). Construction of a genetic toggle switch in Escherichia coli. *Nature, 403*, 339-342. doi:doi:10.1038/35002131

Gehan, O., Pigeon, E., Menard, T., Mosrati, R., Pouliquen, M., Fall, L. M., & Reuter, J. (2019). Dissolved oxygen level output feedback control based on discrete-time measurements during a

Pseudomonas putida mt-2 fermentation. *Journal of Process Control, 79*, 29-40. doi:10.1016/j.jprocont.2018.10.004

Goldrick, S., Duran-Villalobos, C.A., Jankauskas, K., Lovett, D., Farid, S.S., Lennox, B. (2019). Modern day monitoring and control challenges outlined on an industrial-scale benchmark fermentation process. *Computational Chemical Engineering*, 130, 106471, 10.1016/j.compchemeng.2019.05.037

Guiochon, G., & Guillemin, C. L. (1988). Quantitative analysis by gas chromatography measurement of peak area and derivation of sample composition. *For laboratory analyses and on-line process control*, 42, 629-659. doi.org/10.1016/S0301-4770(08)70087-3

Gupta, P., & Phulara, S. C. (2015). Metabolic engineering for isoprenoid-based biofuel production. *J Applied Microbiology, 119*(3), 605-619. doi:10.1111/jam.12871

Holzberg, T. R., Watson, V., Brown, S., Andar, A., Ge, X., Kostov, Y., . . . Rao, G. (2018). Sensors for biomanufacturing process development: facilitating the shift from batch to continuous manufacturing. *Current Opinion in Chemical Engineering, 22*, 115-127. doi:10.1016/j.coche.2018.09.008

Hoye, T. R., & Zao, H. (1999). Some allylic substituent effects in ring-closing metathesis reactions: Allylic alcohol activation. *Organic Letters, 1*, 1123-1125.

Industry progress on UCO sustainability for UK biodiesel production. (2019). [Press release]

Kamatou, G. P. P., & Viljoen, A. M. (2008). Linalool – A Review of a Biologically Active Compound of Commercial Importance. *Natural Product Communication, 9*, 1183-1192.

Karuppiah, V., Ranaghan, K. E., Leferink, N. G. H., Johannissen, L. O., Shanmugam, M., Ni Cheallaigh, A., . . . Scrutton, N. S. (2017). Structural Basis of Catalysis in the Bacterial Monoterpene Synthases Linalool Synthase and 1,8-Cineole Synthase. *ACS Catalysis, 7*(9), 6268-6282. doi:10.1021/acscatal.7b01924

Kempinski, C. J., Z.; Bell, S., Chappell, J. (2015). Biotechnology of Isoprenoids. *Advances in Biochemical Engineering/Biotechnology, 148*.

Kobayashi, H., Kaern, M., Araki, M., Chung, K., Gardner, T. S., Cantor, C. R., & Collins, J. J. (2004). Programmable cells: interfacing natural and engineered gene networks. *Proceedings of the National Academy of Science, 101*(22), 8414-8419. doi:10.1073/pnas.0402940101

Lange, B. M., Rujan, T., Martin, W., Croteau, R. (2000). Isoprenoid biosynthesis: the evolution of two ancient and distinct pathways across genomes. *Proceedings of the National Academy of Science U S A, 97* (24), 13172-13177. doi:10.1073/pnas.240454797

Lapczynski, A., Bhatia, S. P., Letizia, C. S., Api, A. M. (2008). Fragrance material review on l-linalool. *Food and Chemical Toxicology, 46 Supplement 11*, S195-196. doi:10.1016/j.fct.2008.06.057

Leferink, N. G. H., Dunstan, M. S., Hollywood, K. A., Swainston, N., Currin, A., Jervis, A. J., . . . Scrutton, N. S. (2019). An automated pipeline for the screening of diverse monoterpene synthase libraries. *Scientific Reports, 9*(1), 11936. doi:10.1038/s41598-019-48452-2

Leferink, N. G. H., Jervis, A. J., Zebec, Z., Toogood, H. S., Hay, S., Takano, E., Scrutton, N. S. (2016). A 'Plug and Play' Platform for the Production of Diverse Monoterpene Hydrocarbon Scaffolds in Escherichia coli. *ChemistrySelect, 1*(9), 1893-1896. doi:10.1002/slct.201600563

Li, M., Hou, F., Wu, T., Jiang, X., Li, F., Liu, H., . . . Zhang, H. (2020). Recent advances of metabolic engineering strategies in natural isoprenoid production using cell factories. *Natural Product Reports, 37*(1), 80-99. doi:10.1039/c9np00016j

Lidgren, L., Lilja, O., Krook, M., Kriz, D. (2006). Automatic fermentation control based on a real-time in situ SIRE biosensor regulated glucose feed. *Biosensors and Bioelectronics, 21*(10), 2010-2013. doi:10.1016/j.bios.2005.09.012

Liguori, R., & Faraco, V. (2016). Biological processes for advancing lignocellulosic waste biorefinery by advocating circular economy. *Bioresource Technology, 215*, 13-20. doi:10.1016/j.biortech.2016.04.054

Liu, N., Santala, S., & Stephanopoulos, G. (2020). Mixed carbon substrates: a necessary nuisance or a missed opportunity? *Current Opinion in Biotechnology, 62*, 15-21. doi:10.1016/j.copbio.2019.07.003

Liu, S. (2017). Introduction. *Bioprocess Engineering* (pp. 1-20).

Lombard, J., & Moreira, D. (2011). Origins and early evolution of the mevalonate pathway of isoprenoid biosynthesis in the three domains of life. *Molecular Biology and Evolution, 28*(1), 87-99. doi:10.1093/molbev/msq177

Maciejewska, M., Szczurek, A., Kerényi, Z. (2006). Utilisation of first principal component extracted from gas sensor measurements as a process control variable in wine fermentation. *Sensors and Actuators B: Chemical, 115*(1), 170-177. doi:10.1016/j.snb.2005.08.036

Mendez-Perez, D., Alonso-Gutierrez, J., Hu, Q., Molinas, M., Baidoo, E. E. K., Wang, G., . . . Lee, T. S. (2017). Production of jet fuel precursor monoterpenoids from engineered Escherichia coli. *Biotechnology and Bioengineering, 114*(8), 1703-1712. doi:10.1002/bit.26296

Mewalal, R., Rai, D. K., Kainer, D., Chen, F., Kulheim, C., Peter, G. F., Tuskan, G. A. (2017). Plant-Derived Terpenes: A Feedstock for Specialty Biofuels. *Trends in Biotechnology, 35*(3), 227-240. doi:10.1016/j.tibtech.2016.08.003

Meylemans, H. A., Quintana, R. L., Goldsmith, B. R., & Harvey, B. G. (2011). Solvent-free conversion of linalool to methylcyclopentadiene dimers: a route to renewable high-density fuels. *ChemSusChem, 4*(4), 465-469. doi:10.1002/cssc.201100017

Milias-Argeitis, A., Rullan, M., Aoki, S. K., Buchmann, P., Khammash, M. (2016). Automated optogenetic feedback control for precise and robust regulation of gene expression and cell growth. *Nature Communications, 7*, 12546. doi:10.1038/ncomms12546

Milias-Argeitis, A., Summers, S., Stewart-Ornstein, J., Zuleta, I., Pincus, D., El-Samad, H., . . . Lygeros, J. (2011). In silico feedback for in vivo regulation of a gene expression circuit. *Nature Biotechnology, 29*(12), 1114-1116. doi:10.1038/nbt.2018

Miziorko, H. M. (2011). Enzymes of the mevalonate pathway of isoprenoid biosynthesis. *Archives of Biochemistry and Biophysics, 505*(2), 131-143. doi:10.1016/j.abb.2010.09.028

Moeller, L., Grunberg, M., Zehnsdorf, A., Aurich, A., Bley, T., Strehlitz, B. (2011). Repeated fed-batch fermentation using biosensor online control for citric acid production by Yarrowia lipolytica. *Journal of Biotechnology, 153*(3-4), 133-137. doi:10.1016/j.jbiotec.2011.03.013

Montgomery, D. C. (2013). *Design and analysis of experiments*. Eigth edition.

Nakano, C., Kim, H. K., Ohnishi, Y. (2011). Identification and characterization of the linalool/nerolidol synthase from Streptomyces clavuligerus. *Chembiochem, 12*(16), 2403-2407. doi:10.1002/cbic.201100501

National Plan for Industrial Biotechnology. (2013). Retrieved from https://ec.europa.eu/knowledge4policy/sites/know4pol/files/scotland-national_plan_for_industrial_biotechnology.pdf

Oldfield, E., & Lin, F. Y. (2012). Terpene biosynthesis: modularity rules. *Angewandte Chemie International Edition in English, 51*(5), 1124-1137. doi:10.1002/anie.201103110

Olson, E. J., Hartsough, L. A., Landry, B. P., Shroff, R., Tabor, J. J. (2014). Characterizing bacterial gene circuit dynamics with optically programmed gene expression signals. *Nature Methods, 11*(4), 449-455. doi:10.1038/nmeth.2884

Pachauri, N., Singh, V., Rani, A. (2017). Two degree of freedom PID based inferential control of continuous bioreactor for ethanol production. *ISA Transactions, 68*, 235-250. doi:10.1016/j.isatra.2017.03.014

Pommer, H. N., A. . (1975). Industrial Synthesis of Terpene Compounds. *Organic Synthesis.*

Rico, J., Duquesne, K., Petit, J. L., Mariage, A., Darii, E., Peruch, F., . . . Iacazio, G. (2019). Exploring natural biodiversity to expand access to microbial terpene synthesis. *Microbial Cell Factories, 18*(1), 23. doi:10.1186/s12934-019-1074-4

Romano, A., Capozzi, V., Spano, G., Biasioli, F. (2015). Proton transfer reaction-mass spectrometry: online and rapid determination of volatile organic compounds of microbial origin. *Applied Microbiology and Biotechnology, 99*(9), 3787-3795. doi:10.1007/s00253-015-6528-y

Rugbjerg, P., & Sommer, M. O. A. (2019). Overcoming genetic heterogeneity in industrial fermentations. *Nature Biotechnology, 37*(8), 869-876. doi:10.1038/s41587-019-0171-6

Sagmeister, P., Wechselberger, P., Jazini, M., Meitz, A., Langemann, T., Herwig, C. (2013). Soft sensor assisted dynamic bioprocess control: Efficient tools for bioprocess development. *Chemical Engineering Science, 96*, 190-198. doi:10.1016/j.ces.2013.02.069

Sanctis, F. D. (2016). Opening of the world's first industrial scale plant for the production of butanediol via fermentation of renewable raw materials [Press release]

Seborg et al. (2011). *Process dynamics and control*, 3rd edition, John Wiley & Sons.

Schwab, W., Fuchs, C., Huang, F.-C. (2013). Transformation of terpenes into fine chemicals. *European Journal of Lipid Science and Technology, 115*(1), 3-8. doi:10.1002/ejlt.201200157

Sy, C. L., Ubando, A. T., Aviso, K. B., Tan, R. R. (2018). Multi-objective target oriented robust optimization for the design of an integrated biorefinery. *Journal of Cleaner Production, 170*, 496-509. doi:10.1016/j.jclepro.2017.09.140

Tetali, S. D. (2019). Terpenes and isoprenoids: a wealth of compounds for global use. *Planta, 249*(1), 1-8. doi:10.1007/s00425-018-3056-x

Tofwerl AG (2020). Retrieved from https://www.tofwerk.com/

Tippmann, S., Chen, Y., Siewers, V., Nielsen, J. (2013). From flavors and pharmaceuticals to advanced biofuels: production of isoprenoids in Saccharomyces cerevisiae. *Biotechnology Journal, 8*(12), 1435-1444. doi:10.1002/biot.201300028

Vickers, C. E., Williams, T. C., Peng, B., Cherry, J. (2017). Recent advances in synthetic biology for engineering isoprenoid production in yeast. *Current Opinion in Chemistry and Biology, 40*, 47-56. doi:10.1016/j.cbpa.2017.05.017

Wang, C., Zada, B., Wei, G., Kim, S. W. (2017). Metabolic engineering and synthetic biology approaches driving isoprenoid production in Escherichia coli. *Bioresource Technology, 241*, 430-438. doi:10.1016/j.biortech.2017.05.168

Xiao, H., Zhang, Y., Wang, M. (2019). Discovery and Engineering of Cytochrome P450s for Terpenoid Biosynthesis. *Trends in Biotechnology, 37*(6), 618-631. doi:10.1016/j.tibtech.2018.11.008

Xie, J., Ping, H., Tan, T., Lei, L., Xie, H., Yang, X.-Y., Fu, Z. (2019). Bioprocess-inspired fabrication of materials with new structures and functions. *Progress in Materials Science, 105*. doi:10.1016/j.pmatsci.2019.05.004

Yang, J., Nie, Q., Liu, H., Xian, M., & Liu, H. (2016). A novel MVA-mediated pathway for isoprene production in engineered E. coli. *BMC Biotechnology, 16*, 5. doi:10.1186/s12896-016-0236-2

You, L. C. I., R. S.; Weiss, R.; Arnold, F. H. (2004). Programmed population control by cell–cell communication and regulated killing. *Nature, 428*(6985), 868-871. doi:10.1038/nature02468

Yuhong, W. D., H.; Dongjie, G.; Yihui, J. (2003). Wavelet Networks Based Soft Sensors and Predictive Control in Fermentation Process. *Process Systems Engineering*, 1222-1227.

Zhang, J. (2013). Metabolic engineering of Escherichia coli for the biosynthesis of alpha-pinene. *Biotechnology and Biofuels, 6*, 1-10.

Zhang, K., & Cui, B. (2015). Optogenetic control of intracellular signaling pathways. *Trends in Biotechnology, 33*(2), 92-100. doi:10.1016/j.tibtech.2014.11.007

Zhao, E. M., Zhang, Y., Mehl, J., Park, H., Lalwani, M. A., Toettcher, J. E., Avalos, J. L. (2018). Optogenetic regulation of engineered cellular metabolism for microbial chemical production. *Nature, 555*(7698), 683-687. doi:10.1038/nature26141

Appendix

All the code reported in the following pages is also available on GitHub at: https://github.com/giovanni92lorenzon/BioControl1.0

Custom functions:

geth5log.m

```
function [n_cycles, n_zeroes, chunk_size, error_msg] = geth5log(filename)
% ==========================================================================
% INPUTs
% 'filename' = Name of the .h5 file in output from the PTR-MS in GC-mode
%
% OUTPUTs
% 'n_cycles' = Number of completed acquisition cycles
% 'n_zeroes' = Number of zeroes in the last cycle
% 'chunk_size' = Chunk size of stored data
% 'error_msg' = Regarding unexpected n_cycles, n_zeroes, or chunk_size
%
%
% Function to extract number of completed acquisition cycles and number of
% zeroes in the final cycle FOR REGULAR PTR-MS FILES in GC MODE
% ==========================================================================


% ==========================================================================
% Initialisation and error handling
% ==========================================================================
assert(ischar(filename),'First input <filename> must be a char array.')

check = strcmpi(filename((end - 2):end),'.h5');
if ~check
        error('Filetype was not expected. Use .h5 file.')
end


% ==========================================================================
% Log data extraction
% ==========================================================================
name_log_dtst = '/AcquisitionLog/Log'; % Where the vector containing all
                                          % of the measured masses is
                                          % stored
cycles_pos = 27; % Position index for the beginning of the 'number of
                 % cycles' entry in the logfile of each .h5 file
shift_to_zeroes_pos = 17; % Shift from end of the 'cycles' entry to the
                          % beginning of the 'zeroes' entry
expected_chunk_size = 6;
%--------------------------------------------------------------------------
log = h5read(filename, name_log_dtst);
log = log.logtext';

length_check = find(log(2,:) == ' ');
if length(length_check) <= 3
    error_msg = ['MSmode_file'];
    n_cycles = 0;
    n_zeroes = 0;
    chunk_size = 0;
    return
end

spot = '';
cycles = '';
while ~strcmp(spot, ' ')
    spot = log(2,cycles_pos);
    cycles = [cycles spot];
    cycles_pos = cycles_pos + 1;
end

n_cycles = str2double(cycles(1:(end-1)));


% ========================================================================
% Double check with actual time array
% ========================================================================
info = h5info(filename, '/TimingData/BufTimes');
time_dtst_size = info.Dataspace.Size;

check1 = time_dtst_size(1) ~= expected_chunk_size;
check2 = time_dtst_size(2) ~= n_cycles;

if  check1 || check2
    n_cycles = time_dtst_size(2);
    chunk_size = time_dtst_size(1);

    start = [1 n_cycles];
    count = [chunk_size 1];
    last_cycle_data = ...
        h5read(filename, '/TimingData/BufTimes', start, count);
    indexes = find(last_cycle_data == 0);
    n_zeroes = length(indexes);
    if check1 && ~check2
        error_msg = ['UnexpectedChunkSize(' num2str(chunk_size) ')'];
    elseif ~check1 && check2
        error_msg = ['UnexpectedCycleNumber(' num2str(n_cycles) ')'];
    else
        error_msg = ['UnexpectedChunkSize(' num2str(chunk_size) ')' ...
            '&UnexpectedCycleNumber(' num2str(n_cycles) ')'];
    end
else
    n_zeroes = str2double(log(2,(cycles_pos + shift_to_zeroes_pos)));
    chunk_size = expected_chunk_size;
    error_msg = 'None';
end
```

<u>geth5mocklog.m</u>

```
function [n_cycles, n_zeroes] = geth5mocklog(filename)
% ========================================================================
% INPUTs
% 'filename' = Name of the .h5 mockfile to simulate live PTR-MS acquisition
%
% OUTPUTs
% 'n_cycles' = Number of completed acquisition cycles
% 'n_zeroes' = Number of zeroes in the last cycle
%
%
% Function to extract number of completed acquisition cycles and number of
% zeroes in the final cycle FOR MOCK PTR-MS FILES
% ========================================================================
% ==========================================================================
% Initialisation and error handling
% ==========================================================================
assert(ischar(filename),'First input <filename> must be a char array.')

check = strcmpi(filename((end - 2):end),'.h5');
if ~check
        error('Filetype was not expected. Use .h5 file.')
end


% ==========================================================================
% Log data extraction
% ==========================================================================
name_log_dtst = '/AcquisitionLog/Log'; % Where the vector containing all
                                          % of the measured masses is
                                          % stored
cycles_pos = 27; % Position index for the beginning of the 'number of
                 % cycles' entry in the logfile of each .h5 file
shift_to_zeroes_pos = 17; % Shift from end of the 'cycles' entry to the
                          % beginning of the 'zeroes' entry
%---------------------------------------------------------------------------
log = char(h5read(filename, name_log_dtst));

spot = '';
cycles = '';
while ~strcmp(spot, ' ')
    spot = log(cycles_pos);
    cycles = [cycles spot];
    cycles_pos = cycles_pos + 1;
end

n_cycles = str2double(cycles(1:(end-1)));
n_zeroes = str2double(log(cycles_pos + shift_to_zeroes_pos));
end
```

<u>geth5logMS.m</u>

```
function [n_cycles, n_zeroes, chunk_size] = geth5logMS(filename)
% ==========================================================================
% INPUTs
% 'filename' = Name of the .h5 file in output from the PTR-MS in MS-mode
%
% OUTPUTs
% 'n_cycles' = Number of completed acquisition cycles
% 'n_zeroes' = Number of zeroes in the last cycle
% 'chunk_size' = Chunk size of stored data
%
%
% Function to extract number of completed acquisition cycles and number of
% zeroes in the final cycle FOR REGULAR PTR-MS FILES in MS MODE
% ==========================================================================


% ==========================================================================
% Initialisation and error handling
% ==========================================================================
assert(ischar(filename),'First input <filename> must be a char array.')

check = strcmpi(filename((end - 2):end),'.h5');
```

```
if ~check
        error('Filetype was not expected. Use .h5 file.')
end


% ==========================================================================
% Log data extraction
% ==========================================================================
info = h5info(filename, '/TimingData/BufTimes');
time_dtst_size = info.Dataspace.Size;

n_cycles = time_dtst_size(2);
chunk_size = time_dtst_size(1);

start = [1 n_cycles];
count = [chunk_size 1];
last_cycle_data = ...
    h5read(filename, '/TimingData/BufTimes', start, count);
indexes = find(last_cycle_data == 0);
n_zeroes = length(indexes);

end
```

geth5masses.m

```
function masses = geth5masses(filename)
% ==========================================================================
% INPUTs
% 'filename' = Name of the .h5 file in output from the PTR-MS (or mockfile)
%
% OUTPUTs
% 'masses' = Mx1 column array containing all of the masses detected by the
%            PTR-MS (or mock masses)
%
%
% Function to extract the whole array of the masses analysed by the PTR-MS
% ==========================================================================


% ==========================================================================
% Initialisation and error handling
% ==========================================================================
assert(ischar(filename),'First input <filename> must be a char array.')

check = strcmpi(filename((end - 2):end),'.h5');
if ~check
        error('Filetype was not expected. Use .h5 file.')
end


% ==========================================================================
% Mass data extraction
% ==========================================================================
name_mass_dtst = '/FullSpectra/MassAxis'; % Where the vector containing all
                                          % of the measured masses is
                                          % stored
%---------------------------------------------------------------------------
masses = h5read(filename, name_mass_dtst);
end
```

geth5times.m

```
function times = geth5times(filename)
% =========================================================================
% INPUTs
% 'filename' = Name of the .h5 file in output from the PTR-MS (or mockfile)
%
% OUTPUTs
% 'times' = 1xP row array containing all of the timepoints registered by
%           the current file
%
%
% Function to extract the timepoints measured by the PTR-MS. Storage
% structure within .h5 file is [S,C], where S is the chunk size (=6), and C
% is the number of completed acquisition cycles. Mind that the
% function extracts all of the values and rearranges them in a 1xP
% structure, where P is given by P = S*C - n_zeroes (zeroes are chunked)
%
% DEPENDANCIES: 'geth5log.m', 'geth5mocklog.m', 'geth5logMS.m'
% =========================================================================


% =========================================================================
% Initialisation and error handling
% =========================================================================
assert(ischar(filename),'First input <filename> must be a char array.')

check = strcmpi(filename((end - 2):end),'.h5');
if ~check
        error('Filetype was not expected. Use .h5 file.')
end


% =========================================================================
% Log data extraction
% =========================================================================
if contains(filename, 'mock')
    [n_cycles, n_zeroes] = geth5mocklog(filename);
    chunk_size = 6; % Storage file chunk size for time points
else
    [n_cycles, n_zeroes, chunk_size, error_msg] = geth5log(filename);
    if strcmp(error_msg, 'MSmode_file')
        [n_cycles, n_zeroes, chunk_size] = geth5logMS(filename);
    end
end


% =========================================================================
% Time data extraction
% =========================================================================
name_time_dtst = '/TimingData/BufTimes'; % Where all the times of the
                                         % single timesteps are stored
% chunk_size = 6; % Storage file chunk size for time points
%--------------------------------------------------------------------------
time_raw = h5read(filename, name_time_dtst);

times = zeros(1,n_cycles*chunk_size);
for i = 1:(n_cycles)
    times( ...
        (((i-1)*chunk_size)+1): ...
        (((i-1)*chunk_size)+chunk_size) ...
        ) = time_raw(:,i);
end
%--------------------------------------------------------------------------
for z = 1:n_zeroes
```

```
    times(:,end) = [];
end
end
```

geth5mrcumpeaks.m

```
function [cumpeakprof, mass_rng, times] = geth5mrcumpeaks( ...
    filename, ...
    central_mass, neighbourhood)
% =========================================================================
% INPUTs
% 'filename' = Name of the .h5 file in output from the PTR-MS (or mockfile)
% 'central_mass' = Mass target to be analysed.
%                  To be considered as --> !!!(MASS + 1)!!!
% 'neighbourhood' = Defines the mass range to be included in the
%                   calculation of the peak at the specified mass target
%                   [central_mass-neighbourhood,central_mass+neighbourhood]
%
% OUTPUTs
% 'cumpeakprof' = 1xP array containing the cumulative signal intensity
%                 profile over time (P) for the mass range defined in the
%                 inputs.
%                 Each column contains the  cumulative ions/s value
%                 calculated across all of the masses for a fixed
%                 timepoint.
% 'mass_rng' = Mrx1 column array containing the masses within the range
%              described by the central mass +- neighbourhood
% 'times' = 1xP row array containing all of the timepoints registered by
%           the current file
%
%
% Function to extract the cumulative signal intensity across the inputted
% mass range over time. Storage structure within .h5 file is [M,1,S,C],
% where M is the number of all of the detected masses, S is the chunk size
% (=6), and C is the number of completed acquisition cycles. Mind that this
% function extracts all of the values and rearranges them in a MrxP
% structure, where P is given by P = S*C - n_zeroes (zeroes are chunked),
% and Mr is equal to the number of mass points between the provided range
% to be analysed. Eventually, it sums up the values across each column, to
% get an output like 1xP.
%
% DEPENDANCIES: 'geth5log.m', 'geth5mocklog.m', 'geth5masses.m',
%               'geth5time.m'
% =========================================================================


% =========================================================================
% Initialisation and error handling
% =========================================================================
format long

assert(ischar(filename),'First input <filename> must be a char array.')

check = strcmpi(filename((end - 2):end),'.h5');
if ~check
        error('Filetype was not expected. Use .h5 file.')
end
%--------------------------------------------------------------------------
bool = isnumeric(central_mass) & isnumeric(neighbourhood);

assert(bool, ['Second and third inputs <central_mass> and ' ...
    '<neighbourhood> must be numeric values.']);
```

```
%-----------------------------------------------------------------------
% Gathers data on masses analysed to check if 'central_mass' and
% 'neighbourhood' are valid picks
masses = geth5masses(filename);

assert(~(central_mass < masses(1) | central_mass > masses(end)), ...
    ['Second input <central_mass> does not fall within the mass range ' ...
    'measured by the inputted file.'])


% ======================================================================
% Pair mass range to mass list indexes
% ======================================================================
cond = (central_mass - neighbourhood) < masses(1) | ...
    (central_mass + neighbourhood) > masses(end);

assert(~cond, 'Neighbourhood to be analysed exceeds mass range limits.')

[~,ix_min] = min(abs(masses - (central_mass - neighbourhood)));
[~,ix_max] = min(abs(masses - (central_mass + neighbourhood)));



% ======================================================================
% Log data extraction
% ======================================================================
if contains(filename, 'mock')
    [n_cycles, n_zeroes] = geth5mocklog(filename);
    chunk_size = 6; % Storage file chunk size for time points
else
    [n_cycles, n_zeroes, chunk_size, error_msg] = geth5log(filename);
    if strcmp(error_msg, 'MSmode_file')
        [n_cycles, n_zeroes, chunk_size] = geth5logMS(filename);
    end
end


% ======================================================================
% Extracting peak data
% ======================================================================
name_tofdata_dtst = '/FullSpectra/TofData'; % Where ions/s data are stored
                                            % for every time point and for
                                            % every recorded mass
% chunk_size = 6; % Storage file chunk size for time points

start = [ix_min 1 1 1];
x_count = length(masses(ix_min:ix_max));
count = [x_count 1 chunk_size n_cycles];
stride = [1 1 1 1];
%-----------------------------------------------------------------------
tofdata_raw = h5read(filename, name_tofdata_dtst, start, count, stride);

time_length = chunk_size*n_cycles;

tofdata = zeros(x_count,time_length);
for i = 1:x_count
    singlemass = zeros(1,time_length);
    for j = 1:(n_cycles)
        singlemass( ...
            (((j-1)*chunk_size)+1): ...
            (((j-1)*chunk_size)+chunk_size) ...
            ) = tofdata_raw(i,1,:,j);
    end
    tofdata(i,:) = singlemass;
end
%-----------------------------------------------------------------------
```

```
for z = 1:n_zeroes
    tofdata(:,end) = [];
end
%-------------------------------------------------------------------------
cumpeakprof = sum(tofdata);

mass_rng = masses(ix_min:ix_max);

times = geth5times(filename);

% cumpeakprof = zeros(1,length(times));
% for i = 1:length(times)
%     single = trapz(mass_rng,tofdata(:,i));
%     cumpeakprof(i) = single;
% end
end
```

geth5mtrcumpeaks.m

```
function [cumpeakprofrng, mass_rng, times_rng] = geth5mtrcumpeaks( ...
    filename, ...
    central_mass, neighbourhood, ...
    t_start, t_end)
% ========================================================================
% INPUTs
% 'filename' = Name of the .h5 file in output from the PTR-MS (or mockfile)
% 'central_mass' = Mass target to be analysed
% 'neighbourhood' = Defines the mass range to be included in the
%                   calculation of the peak at the specified mass target
%                   [central_mass-neighbourhood,central_mass+neighbourhood]
% 't_start' = Time (in seconds) to the define the starting point of the
%             output array
% 't_end' = Time (in seconds) to the define the end point of the output
%           array
%
% OUTPUTs
% 'cumpeakprofrng' = 1xPr array containing the cumulative signal intensity
%                    profile over a specific range of time (Pr) within that
%                    recorded by the file in analysis for the mass range
%                    defined in the inputs.
%                    Each column contains the  cumulative ions/s value
%                    calculated across all of the masses for a fixed
%                    timepoint.
% 'mass_rng' = Mrx1 column array containing the masses within the range
%              described by the central mass +- neighbourhood
% 'times_rng' = 1xPr row array containing the timepoints within the range
%               given by the last two inputs
%
%
% Function to extract the cumulative signal intensity across the inputted
% mass range over a specific range of time. Storage structure within .h5
% file is [M,1,S,C], where M is the number of all of the detected masses,
% S is the chunk size (=6), and C is the number of completed acquisition
% cycles. Mind that this function extracts all of the values and rearranges
% them in a MrxPr structure, where Pr is the selected range of times
% belonging to the original set given by P = S*C - n_zeroes (zeroes are
% chunked), and Mr is equal to the number of mass points between the
% provided range to be analysed. Eventually, it sums up the values across
% each column, to get an output like 1xPr.
%
% DEPENDANCIES: 'geth5masses.m', 'geth5time.m', 'geth5log.m'
% ========================================================================
% ========================================================================
% Initialisation and error handling
% ========================================================================
format long

assert(ischar(filename),'First input <filename> must be a char array.')

check = strcmpi(filename((end - 2):end),'.h5');
if ~check
        error('Filetype was not expected. Use .h5 file.')
end
%-------------------------------------------------------------------------
bool = isnumeric(central_mass) & isnumeric(neighbourhood);

assert(bool, ['Second and third inputs <central_mass> and ' ...
    '<neighbourhood> must be numeric values.']);
%-------------------------------------------------------------------------
% Gathers data on masses analysed to check if 'central_mass' and
% 'neighbourhood' are valid picks
masses = geth5masses(filename);

assert(~(central_mass < masses(1) | central_mass > masses(end)), ...
    ['Second input <central_mass> does not fall within the mass range ' ...
    'measured by the inputted file.'])
%-------------------------------------------------------------------------
% Gathers data on times analysed to check if 't_start' and 't_end'
% are valid picks
times = geth5times(filename);

assert(~(t_start < times(1) | t_end > times(end)), ...
    'Inputted time limits exceed those of the file in exam.')


% ========================================================================
% Pair mass range to mass list indexes
% ========================================================================
cond = (central_mass - neighbourhood) < masses(1) | ...
    (central_mass + neighbourhood) > masses(end);

assert(~cond, 'Neighbourhood to be analysed exceeds mass range limits')

[~,ix_min] = min(abs(masses - (central_mass - neighbourhood)));
[~,ix_max] = min(abs(masses - (central_mass + neighbourhood)));


% ========================================================================
% Extracts chunk size to be employed
% ========================================================================
if contains(filename, 'mock')
    chunk_size = 6;
else
    [~,~,chunk_size, error_msg] = geth5log(filename);
    if strcmp(error_msg, 'MSmode_file')
        [~, ~, chunk_size] = geth5logMS(filename);
    end
end


% ========================================================================
% Pair time range to times list indexes
% ========================================================================
[~,ix_st] = min(abs(times - t_start),[],'all','linear');
[~,ix_end] = min(abs(times - t_end),[],'all','linear');
```

```
cyc_st = ceil(ix_st/chunk_size);
rem_st = rem(ix_st,chunk_size);
cyc_end = ceil(ix_end/chunk_size);
rem_end = rem(ix_end,chunk_size);
rng = cyc_end - cyc_st;

times_rng = times(ix_st:ix_end);


% ==========================================================================
% Extracting peak data
% ==========================================================================
name_tofdata_dtst = '/FullSpectra/TofData'; % Where ions/s data are stored
                                            % for every time point and for
                                            % every recorded mass

start = [ix_min 1 1 cyc_st];
x_count = length(masses(ix_min:ix_max));
z_count = rng + 1;
count = [x_count 1 chunk_size z_count];
stride = [1 1 1 1];
%--------------------------------------------------------------------------
tofdata_raw = h5read(filename, name_tofdata_dtst, start, count, stride);

time_length = chunk_size*z_count;

tofdata = zeros(x_count,time_length);
for i = 1:x_count
    singlemass = zeros(1,time_length);
    for j = 1:(z_count)
        singlemass( ...
            (((j-1)*chunk_size)+1): ...
            (((j-1)*chunk_size)+chunk_size) ...
            ) = tofdata_raw(i,1,:,j);
    end
    tofdata(i,:) = singlemass;
end
%--------------------------------------------------------------------------
if rem_st == 0
    if rem_end == 0
        tofdata = tofdata(1:end,chunk_size:end);
    else
        tofdata = tofdata(1:end,chunk_size:(end-(chunk_size-rem_end)));
    end
else
    if rem_end == 0
        tofdata = tofdata(1:end,rem_st:end);
    else
        tofdata = tofdata(1:end,rem_st:(end-(chunk_size-rem_end)));
    end
end
%--------------------------------------------------------------------------
cumpeakprofrng = sum(tofdata);

mass_rng = masses(ix_min:ix_max);
end
```

genaddh5mock.m

```
function name_curr_mock = genaddh5mock(user_mass,timelength_mock,varargin)
% ==========================================================================
% INPUTs
% 'user_mass' = Mass value at which simulate a random intensity profile
% 'timelength_mock' = Timespan covered by the mockfile to be generated
%
% OPTIONAL INPUTs
% (1st ARG)'random distribution function' = The function used to generate
%                                           the peak intensity profile over
%                                           time.
%                                           DEFAULT = 'stable'
%                                           OPT1 = 'sinusoindal'
%                                           OPT2 = 'constant'
% (2nd ARG)'measured masses starting point' = The lowest mass value
%                                             detected by the PTRMS during
%                                             the current mock.
%                                             DEFAULT = 0
%                                             OPT = N > 0
%
% OUTPUTs
% 'name_curr_mock' = Name of the mock .h5 file in output
%
%
% Function that generates a mock .h5 file to emulate the output of the
% PTR-MS. The output file is generated exactly as the original one, except
% for the data logging regarding the number of completed acquisition cycles
% and the number of zeroes contained in the last time cycle. In addition to
% file generation, the function adds a random output profile at the mass
% value chosen by the user. This random profile is given by the combination
% of two distributions: (i) a distribution over time based on a userdefined
% distribution' which determines the intensity profile over time for the
% exact mass inputted by the user; (ii) a normal distribution across the
% masses to generate fictional peaks around the chosen mass (usually there
% is a neighbourhood of masses to be considered when detecting a specific
% one)
% ==========================================================================


% ==========================================================================
% Initialisation
% ==========================================================================
% delete 'PTRMSmocksequence*.h5'
% close all
% clear all
format long
% clc


% ==========================================================================
% Error handling & optional arguments evaluation
% ==========================================================================
switch nargin
    case 2
        assert(isnumeric(user_mass) & ...
            2 == ndims(user_mass) & ...
            216 > user_mass & ...
            0 < user_mass & ...
            1 == size(user_mass,1) & ...
            1 == size(user_mass,2), ...
            ['First input <user_mass> must be a positive scalar ' ...
            'lower than 216m/q.'])
        assert(isnumeric(timelength_mock) & ...
            2 == ndims(timelength_mock) & ...
            5 < timelength_mock & ...
            1 == size(timelength_mock,1) & ...
            1 == size(timelength_mock,2), ...
            ['Second input <timelength_mock> must be a positive ' ...
            'scalar higher than 5s.'])
            start_mass = 0;
            distr_fun = 'stable';
        case 3
            assert(isnumeric(user_mass) & ...
                2 == ndims(user_mass) & ...
                216 > user_mass & ...
                0 < user_mass & ...
                1 == size(user_mass,1) & ...
                1 == size(user_mass,2), ...
                ['First input <user_mass> must be a positive scalar ' ...
                'lower than 216m/q.'])
            assert(isnumeric(timelength_mock) & ...
                2 == ndims(timelength_mock) & ...
                5 < timelength_mock & ...
                1 == size(timelength_mock,1) & ...
                1 == size(timelength_mock,2), ...
                ['Second input <timelength_mock> must be a positive ' ...
                'scalar higher than 5s.'])
            assert(ischar(varargin{1}) & ...
                2 == ndims(varargin{1}) & ...
                1 == size(varargin{1},1), ...
                ['Third input <random distribution function> must be ' ...
                'a char row vector (1xN char).'])
            assert(strcmpi(varargin{1}, 'stable') | ...
                strcmpi(varargin{1}, 'sinusoidal') | ...
                strcmpi(varargin{1}, 'constant'), ...
                ['Third input <random distribution function> must match ' ...
                'the options <stable>, <sinusoidal> or <constant>.'])

            start_mass = 0;
            distr_fun = varargin{1};
        otherwise
            assert(isnumeric(varargin{2}) & ...
                2 == ndims(varargin{2}) & ...
                0 < varargin{2} & ...
                1 == size(varargin{2},1) & ...
                1 == size(varargin{2},2), ...
                ['Fourth input <measured masses starting point> must ' ...
                'be a positive scalar higher than 0m/q.'])

            start_mass = varargin{2};
            spectrum_extension = 216; % Span of detection range of PTRMS
            end_mass_rng = start_mass + spectrum_extension;

            assert(isnumeric(user_mass) & ...
                2 == ndims(user_mass) & ...
                end_mass_rng > user_mass & ...
                start_mass < user_mass & ...
                1 == size(user_mass,1) & ...
                1 == size(user_mass,2), ...
                ['First input <user_mass> must be a positive scalar ' ...
                'lower than ' num2str(round(end_mass_rng)) 'm/q.'])
            assert(isnumeric(timelength_mock) & ...
                2 == ndims(timelength_mock) & ...
                5 < timelength_mock & ...
                1 == size(timelength_mock,1) & ...
                1 == size(timelength_mock,2), ...
                ['Second input <timelength_mock> must be a positive ' ...
                'scalar higher than 5s.'])
            assert(ischar(varargin{1}) & ...
                2 == ndims(varargin{1}) & ...
                1 == size(varargin{1},1), ...
                ['Third input <random distribution function> must be ' ...
                'a char row vector (1xN char).'])
            assert(strcmpi(varargin{1}, 'stable') | ...
                strcmpi(varargin{1}, 'sinusoidal') | ...
                strcmpi(varargin, 'constant'), ...
                ['Third input <random distribution function> must match ' ...
                'the options <stable>, <sinusoidal> or <constant>.'])

            distr_fun = varargin{1};
    end


    % ==========================================================================
    % Constants
    % ==========================================================================
    avg_mass_step = 0.0014; % Mass resolution of the PTRMS
    avg_time_step = 0.3; % Time in between ions sampling from the PTRMS
    spectrum_extension = 216; % Span of detection range of PTRMS
    chunk_size = 6; % Storage file chunk size for time points
    cycles_pos = 27; % Position index for the beginning of the 'number of
                     % cycles' entry in the logfile of each .h5 file
    shift_to_zeroes_pos = 17; % Shift from end of the 'cycles' entry to the
                              % beginning of the 'zeroes' entry
    neighbourhood_mock = 0.14; % Neighbourhood of masses whose ion count still
                               % contributes to the evaluation of the central
                               % mass value the user wants to examine. This is
                               % not for scoping, but rather for the definition
                               % of the structure of the mockfile.


    % ==========================================================================
    % User specified data
    % ==========================================================================
    beg_mass_rng = start_mass + 1; % starting point of the examined mass range (starts
    from
                      % 1 because it's the mass of the single proton, the
                      % smallest detectable ion
    % user_mass = 154.25; % Target mass to be analysed
    mass_of_choice = user_mass + 1; % We're working with the shifted mass
                                    % (proton is attached, so +1 in mass)
    % timelength_mock = 30; % Length of time covered by the mockfile in seconds


    % ==========================================================================
    % Flags
    % ==========================================================================
    no_prev_files_flag = 0; % Marks if there is already a previous mock spectra
                            % file (0 = there are previous files; 1 = there is
                            % no previous file)


    % ==========================================================================
    % Check for previous mock-files / modifies the relative flag accordingly /
    % assess the file-number associated with the last mock-file available (the
    % one with the highest number in the name - not the most recent one)
    % ==========================================================================
    mocks = dir('PTRMSmocksequence*.h5');
    n_mocks = length(mocks);

    if n_mocks == 0
        n_last_mock = 0;
        no_prev_files_flag = 1;
    elseif n_mocks ==1
        n_last_mock = str2double(mocks.name(end-5:end-3));
    else
        ns_mocks = zeros(1,n_mocks);
        for i = 1:n_mocks
            ns_mocks(i) = str2double(mocks(i).name(end-5:end-3));
        end
        n_last_mock = max(ns_mocks);
```

```
end


% ==========================================================================
% Defines name for the current mockfile to be outputted (the final file
% name has to be identified by three digits '$$$' --> max n_mockfiles is
% therefore '999')
% ==========================================================================
n_curr_mock = n_last_mock + 1;

if n_curr_mock >= 100
    id_curr_mock = num2str(n_curr_mock);
elseif n_curr_mock < 100 && n_curr_mock >= 10
    id_curr_mock = ['0' num2str(n_curr_mock)];
else
    id_curr_mock = ['00' num2str(n_curr_mock)];
end

name_curr_mock = ['PTRMSmocksequence' id_curr_mock '.h5'];


% ==========================================================================
% Defines the exact number of timepoints for the current mockfile being
% produced (I want each mockfile to cover approximately 30s and being ~0.3s
% the average timestep in the MS data acquisition system it means that the
% number of timesteps is 100. As I want to be flexible and allow already
% for good adaptability towards the files the platform will be working with
% I randomly assign a number of timepoints between 96 and 102. This because
% the chunk size of the data storage system is 6. Every cycle consists of 6
% entries and if the data buffer is emptied at a point in which all 6
% points of the current cycles aren't yet acquired, the system fills the
% remaining entries with zeroes, so to have a valid array structure to
% insert inside of the .h5 file. Similarly, if the number of timepoints
% randomly generated here differs from 96 or 102, there'll be a number of
% zeroes in the final data chunk.
% ==========================================================================
rng('shuffle');
low_threshold = ...
    floor((timelength_mock/avg_time_step)/chunk_size)*chunk_size;
high_threshold = low_threshold + chunk_size;
n_timepoints_curr_mock = randi([low_threshold, high_threshold]);
n_timecycles_curr_mock = ceil(n_timepoints_curr_mock/6);
remainder = rem(n_timepoints_curr_mock,chunk_size);
if remainder > 0
    n_zeroes_curr_mock = chunk_size - remainder; % From 1 to 5
else
    n_zeroes_curr_mock = 0;
end

% ==========================================================================
% Creates all the .h5 structures for the current mockfile
% ==========================================================================
name_time_dtst = '/TimingData/BufTimes'; % Where all the times of the
                                         % single timesteps are stored
time_size_curr_mock = [6 n_timecycles_curr_mock];
h5create(name_curr_mock,name_time_dtst,time_size_curr_mock);
%---------------------------------------------------------------------------
name_mass_dtst = '/FullSpectra/MassAxis'; % Where the vector containing all
                                          % of the measured masses is
                                          % stored
masses_length = round(spectrum_extension/avg_mass_step);
mass_size_curr_mock = [masses_length 1];
h5create(name_curr_mock,name_mass_dtst,mass_size_curr_mock);
%---------------------------------------------------------------------------
name_tofdata_dtst = '/FullSpectra/TofData'; % Where ions/s data are stored
                                            % for every time point and for
                                            % every recorded mass
tofdata_size_curr_mock = ...
    [masses_length, ...
    1, ...
    chunk_size, ...
    n_timecycles_curr_mock];
h5create(name_curr_mock,name_tofdata_dtst,tofdata_size_curr_mock);
%---------------------------------------------------------------------------
name_log_dtst = '/AcquisitionLog/Log'; % Where the info about the number of
                                       % of completed cycles and zeroes in
                                       % the last cycle is stored
% log_size_curr_mock = [1 (91 + length(num2str(n_timecycles_curr_mock)))];
log_size_curr_mock = [1 1];
h5create(name_curr_mock,name_log_dtst,log_size_curr_mock, ...
    'Datatype', 'string');


% ==========================================================================
% Evaluates the starting timepoint based on the presence/absence of
% previous mockfiles
% ==========================================================================
if no_prev_files_flag == 1
    start_time_curr_mock = 0;
else
    % Identifies the file to open to extract the info related to time
    if n_curr_mock >= 2 && n_curr_mock <= 10
        id_last_mock = ['00' num2str(n_last_mock)];
    elseif n_curr_mock > 10 && n_curr_mock <= 100
        id_last_mock = ['0' num2str(n_last_mock)];
    else
        id_last_mock = num2str(n_last_mock);
    end

    % Opens time dataset and file log
    name_last_mock = ['PTRMSmocksequence' id_last_mock '.h5'];
    data_time_last_mock = h5read(name_last_mock, name_time_dtst);
    log_last_mock = char(h5read(name_last_mock, name_log_dtst));

    % Extracts no. of cycles & zeroes from log
    spot = '';
    cycles = '';
    while strcmp(spot, ' ') == 0
        spot = log_last_mock(cycles_pos);
        cycles = [cycles spot];
        cycles_pos = cycles_pos + 1;
    end

    n_timecycles_last_mock = str2double(cycles(1:(end-1)));
    n_zeroes_last_mock = ...
        str2double(log_last_mock( ...
        cycles_pos + shift_to_zeroes_pos ...
        ));

    % Extracts last recorded timepoint and adds timestep to get the
    % starting point of the current mockfile
    if n_zeroes_last_mock > 0
        start_time_curr_mock = ...
            data_time_last_mock((end - n_zeroes_last_mock), end) + ...
            avg_time_step;
    else
        start_time_curr_mock = ...
            data_time_last_mock(end,end) + ...
            avg_time_step;
    end

end
```

```
% ==========================================================================
% Switch between the functions for the creation of the mock intensity
% profile. Switch is based on the variable 'distr_fun'
% ==========================================================================
switch distr_fun
%---------------------------------------------------------------------------
    case 'stable'
% ==========================================================================
% Define a random distribution parameters over time of the ions count.
% This distributions defines the profile over time of the central
% mass to be analysed. It employs the probability density function 'pdf'
% based on the distribution given by 'makedist' (which in turn employs a
% 4-parameters stable distribution). Distribution parameters are randomly
% picked amongst pre-defined ranges that are found to be suitable for the
% desired profile to output.
% ==========================================================================
lim_alpha = [1 2];
lim_beta = [-1 1];
lim_gamma = [10 timelength_mock];

% Defines the delta par based on the time range sampled. Delta is related
% to the central position of the distribution, therefore it needs to
% account for the actually measured time chunk
if no_prev_files_flag == 1
    lim_delta = [0 30];
else
    lim_delta = [(start_time_curr_mock) ...
        (start_time_curr_mock + n_timepoints_curr_mock*avg_time_step)];
end

% Randomly defines the 4 parameters within the defined ranges
lims = [lim_alpha; lim_beta; lim_gamma; lim_delta];
for i = 1:4
    if i == 1
        alpha = (lims(i,2) - lims(i,1))*rand + lims(i,1);
    elseif i == 2
        beta = (lims(i,2) - lims(i,1))*rand + lims(i,1);
    elseif i == 3
        gamma = (lims(i,2) - lims(i,1))*rand + lims(i,1);
    elseif i == 4
        delta = (lims(i,2) - lims(i,1))*rand + lims(i,1);
    end
end

% Creates central distribution
scaling_factor = 100000; % Takes output of distribution function to values
                         % similar to those given by actual ions count on
                         % real measurement of PTRMS

times = linspace(start_time_curr_mock, ...
    (start_time_curr_mock + avg_time_step*(n_timepoints_curr_mock-1)), ...
    n_timepoints_curr_mock);
distribution = ...
    makedist('Stable','alpha',alpha,'beta',beta,'gam',gamma,'delta',delta);
central_ions_distr = pdf(distribution,times);
central_ions_distr = central_ions_distr*scaling_factor;


%---------------------------------------------------------------------------
    case 'sinusoidal'
% ==========================================================================
% Define a random distribution parameters over time of the ions count.
% This distributions defines the profile over time of the central
% mass to be analysed. It employs a 'sinusoidal' function, slowed down to
% guarantee a non-excessive oscillation. Distribution parameters are
% randomly picked amongst pre-defined ranges that are found to be suitable
% for the desired profile to output.
% ==========================================================================
lim_slow_factor = [2.5 4];
lim_scaling_factor = [500 5000]; % Takes output of distribution function to
                                 % values similar to those given by actual
                                 % ions count on real measurement of PTRMS

slow_factor = (lim_slow_factor(2) - lim_slow_factor(1))*rand + ...
    lim_slow_factor(1);
scaling_factor = (lim_scaling_factor(2) - lim_scaling_factor(1))*rand + ...
    lim_scaling_factor(1);

times = linspace(start_time_curr_mock, ...
    (start_time_curr_mock + avg_time_step*(n_timepoints_curr_mock-1)), ...
    n_timepoints_curr_mock);

central_ions_distr = scaling_factor*sin(times/slow_factor);

if no_prev_files_flag == 1
    central_ions_distr = central_ions_distr + lim_scaling_factor(2);
end


%---------------------------------------------------------------------------
    case 'constant'
% ==========================================================================
% Define a random distribution parameters over time of the ions count.
% This distributions defines the profile over time of the central
% mass to be analysed. It employs a 'sinusoidal' function, slowed down to
% guarantee a non-excessive oscillation. Distribution parameters are
% randomly picked amongst pre-defined ranges that are found to be suitable
% for the desired profile to output.
% ==========================================================================
times = linspace(start_time_curr_mock, ...
    (start_time_curr_mock + avg_time_step*(n_timepoints_curr_mock-1)), ...
    n_timepoints_curr_mock);

if no_prev_files_flag == 1
    prompt = 'Output level';
    dlgtitle = 'Enter numerical value of mocked output';
    dims = [1 100];
    output = inputdlg(prompt,dlgtitle,dims);

    central_ions_distr = linspace( ...
        str2double(output{1}), ...
        str2double(output{1}), ...
        n_timepoints_curr_mock);
else
    central_ions_distr = zeros(1, n_timepoints_curr_mock);
end
%---------------------------------------------------------------------------
end


% ==========================================================================
% Generates the orthogonal distribution related to the masses. Target mass
% isn't the only one that registers ions related to the target coupound,
% also a small neighbourhood records ions which are fundamental to the
% accurate measurement of the concentration of the target compound.
% Therefore, it is desirable to simulate the same profile in the mock file.
% To do this, a normal distribution is generated. Combining the ions
% distribution over time with the distribution over the masses, will
% generate the surface of peaks across both masses and timepoints.
% ==========================================================================
% Generates the whole list of masses that the PTRMS will analyse
```

```
data_mass_curr_mock = ...
    [beg_mass_rng:...
    avg_mass_step:...
    (beg_mass_rng + (masses_length - 1)*avg_mass_step)];
data_mass_curr_mock = data_mass_curr_mock';

% Extracts the mass range in which the mock profile will be generated
% according to the user defined mass value whose peak must be simulated
[~,ix_mass_choice] = min(abs(data_mass_curr_mock - mass_of_choice));
ixs_neighbourhood = round(neighbourhood_mock/avg_mass_step);
masses_mock = data_mass_curr_mock(...
    ix_mass_choice - ixs_neighbourhood:...
    ix_mass_choice + ixs_neighbourhood...
    );


% Generates the distribution
norm_distr = normpdf( ...
    masses_mock, ... % Created across the prev identified range of masses
    mass_of_choice, ... % Centre is on the target mass
    neighbourhood_mock/3); % Gives quite a narrow distr, okay for purpose
norm_distr = norm_distr/max(norm_distr); % Normalise to get 'height' values
                                         % of the distr to be 0:1


% =========================================================================
% Accounts for the eventual previous mockfile distribution and guarantees
% continuity across the next mockfile in the distribution of peaks. First,
% it finds the position of the last peak entry for the target mass in the
% previous mockfile, then it extracts it, and lastly it uses it to offset
% the currently generated distribution.
% =========================================================================
if no_prev_files_flag == 0
    ix_last_entry_last_mock = chunk_size - n_zeroes_last_mock;
    read_start = [...
        ix_mass_choice, ...
        1, ...
        ix_last_entry_last_mock, ...
        n_timecycles_last_mock ...
        ];
    read_count = [1 1 1 1];

    endvalue_tofdata_last_mock = ...
        h5read(name_last_mock,name_tofdata_dtst,read_start,read_count);

    central_ions_distr = ...
        central_ions_distr + ...
        (endvalue_tofdata_last_mock - central_ions_distr(1));

    if any(central_ions_distr < 0)
        central_ions_distr(central_ions_distr < 0) = 0;
    end
end


% =========================================================================
% Generates the mixed time-mass distribution of the ion peaks. Then it adds
% the zeroes to maintain proper chunk size, and lastly it structures the
% data in the same way they are stored in the original .h5 files.
% =========================================================================
cross_ions_distr = zeros(length(masses_mock),n_timepoints_curr_mock);
for i = 1:length(masses_mock)
    cross_ions_distr(i,:) = norm_distr(i)*central_ions_distr;
end

% Adds zeroes
```

```
if n_zeroes_curr_mock == 0
    cross_ions_distr_w_nulls = cross_ions_distr;
else
    cross_ions_distr_w_nulls = [ ...
        cross_ions_distr, ...
        zeros(length(masses_mock), n_zeroes_curr_mock) ...
        ];
end

% Rearrange data to be written to .h5 file
data_tofdata_curr_mock = ...
    zeros(length(masses_mock), 1, 6, n_timecycles_curr_mock);
for i = 1:length(masses_mock)
    for j = 1:n_timecycles_curr_mock
        data_tofdata_curr_mock(i,1,:,j) = ...
            cross_ions_distr_w_nulls( ...
            i, ...
            ((j-1)*chunk_size+1):((j-1)*chunk_size+chunk_size) ...
            );
    end
end


% ==========================================================================
% Organises and writes data on the previously generated datasets of the
% mockfile
% ==========================================================================
data_time_curr_mock = zeros(time_size_curr_mock);
if n_zeroes_curr_mock == 0
    for i = 1:n_timecycles_curr_mock
        data_time_curr_mock(:,i) = ...
            times((((i-1)*chunk_size)+1):(((i-1)*chunk_size)+chunk_size));
    end
else
    for i = 1:(n_timecycles_curr_mock - 1)
        data_time_curr_mock(:,i) = ...
            times((((i-1)*chunk_size)+1):(((i-1)*chunk_size)+chunk_size));
    end
    data_time_curr_mock(:,end) = ...
        [times(((n_timecycles_curr_mock-1)*chunk_size+1):end), ...
        zeros(1, n_zeroes_curr_mock)];
end

h5write(name_curr_mock,name_time_dtst,data_time_curr_mock);
%--------------------------------------------------------------------------
h5write(name_curr_mock,name_mass_dtst,data_mass_curr_mock);
%--------------------------------------------------------------------------
write_start = [ ...
    (ix_mass_choice - ixs_neighbourhood), ...
    1, ...
    1, ...
    1 ...
    ];
write_count = [length(masses_mock) 1 6 n_timecycles_curr_mock];

h5write( ...
    name_curr_mock, ...
    name_tofdata_dtst, ...
    data_tofdata_curr_mock, ...
    write_start, ...
    write_count ...
    );
%--------------------------------------------------------------------------
log_curr_mock = string([ ...
    'Acquisition aborted after ' ...
    num2str(n_timecycles_curr_mock) ...
    ' complete writes. ' ...
    num2str(n_zeroes_curr_mock) ...
    ' additional bufs in the incomplete last write.' ...
    ]);

h5write(name_curr_mock,name_log_dtst,log_curr_mock)
end
```

genaddh5calmock.m

```
function name_curr_mock = ...
    genaddh5calmock(user_mass,timelength_mock,signal_intensity)
% ==========================================================================
% INPUTs
% 'user_mass' = Mass value at which simulate a random intensity profile
% 'timelength_mock' = Timespan covered by the mockfile to be generated
% 'signal_intensity' = The intensity value of the constant output that will
%                       be mocked
%
% OUTPUTs
% 'name_curr_mock' = Name of the mock .h5 file in output
%
%
% Function that generates a mock .h5 file to emulate the output of the
% PTR-MS. The output file is generated exactly as the original one, except
% for the data logging regarding the number of completed acquisition cycles
% and the number of zeroes contained in the last time cycle. In addition to
% file generation, the function adds a constant output profile at the mass
% value chosen by the user. This random profile is given by the combination
% of two distributions: (i) a distribution over time based on a 'constant
% distribution' which determines the intensity profile over time for the
% exact mass inputted by the user; (ii) a normal distribution across the
% masses to generate fictional peaks around the chosen mass (usually there
% is a neighbourhood of masses to be considered when detecting a specific
% one)
% ==========================================================================


% ==========================================================================
% Initialisation
% ==========================================================================
% delete 'PTRMSmockcalsequence*.h5'
% close all
% clear all
format long
% clc


% ==========================================================================
% Error handling & optional arguments evaluation
% ==========================================================================
assert(isnumeric(user_mass) & ...
    2 == ndims(user_mass) & ...
    216 > user_mass & ...
    0 < user_mass & ...
    1 == size(user_mass,1) & ...
    1 == size(user_mass,2), ...
    ['First input <user_mass> must be a positive scalar ' ...
    'lower than 216m/q.'])
assert(isnumeric(timelength_mock) & ...
    2 == ndims(timelength_mock) & ...
    5 < timelength_mock & ...
    1 == size(timelength_mock,1) & ...
    1 == size(timelength_mock,2), ...
    ['Second input <timelength_mock> must be a positive ' ...
    'scalar higher than 5s.'])


% ========================================================================
% Constants
% ========================================================================
avg_mass_step = 0.0014; % Mass resolution of the PTRMS
avg_time_step = 0.3; % Time in between ions sampling from the PTRMS
spectrum_extension = 216; % Span of detection range of PTRMS
chunk_size = 6; % Storage file chunk size for time points
cycles_pos = 27; % Position index for the beginning of the 'number of
                 % cycles' entry in the logfile of each .h5 file
shift_to_zeroes_pos = 17; % Shift from end of the 'cycles' entry to the
                          % beginning of the 'zeroes' entry
neighbourhood_mock = 0.14; % Neighbourhood of masses whose ion count still
                           % contributes to the evaluation of the central
                           % mass value the user wants to examine. This is
                           % not for scoping, but rather for the definition
                           % of the structure of the mockfile.
distr_fun = 'constant';


% ========================================================================
% User specified data
% ========================================================================
beg_mass_rng = 1; % starting point of the examined mass range (starts from
                  % 1 because it's the mass of the single proton, the
                  % smallest detectable ion
% user_mass = 154.25; % Target mass to be analysed
mass_of_choice = user_mass + 1; % We're working with the shifted mass
                                % (proton is attached, so +1 in mass)
% timelength_mock = 30; % Length of time covered by the mockfile in seconds


% ========================================================================
% Flags
% ========================================================================
no_prev_files_flag = 0; % Marks if there is already a previous mock spectra
                        % file (0 = there are previous files; 1 = there is
                        % no previous file)


% ========================================================================
% Check for previous mock-files / modifies the relative flag accordingly /
% assess the file-number associated with the last mock-file available (the
% one with the highest number in the name - not the most recent one)
% ========================================================================
mocks = dir('PTRMSmockcalsequence*.h5');
n_mocks = length(mocks);

if n_mocks == 0
    n_last_mock = 0;
    no_prev_files_flag = 1;
elseif n_mocks ==1
    n_last_mock = str2double(mocks.name(end-5:end-3));
else
    ns_mocks = zeros(1,n_mocks);
    for i = 1:n_mocks
        ns_mocks(i) = str2double(mocks(i).name(end-5:end-3));
    end
    n_last_mock = max(ns_mocks);
end


% ========================================================================
% Defines name for the current mockfile to be outputted (the final file
% name has to be identified by three digits '$$$' --> max n_mockfiles is
% therefore '999')
% ========================================================================
n_curr_mock = n_last_mock + 1;

if n_curr_mock >= 100
    id_curr_mock = num2str(n_curr_mock);
elseif n_curr_mock < 100 && n_curr_mock >= 10
    id_curr_mock = ['0' num2str(n_curr_mock)];
else
    id_curr_mock = ['00' num2str(n_curr_mock)];
end

name_curr_mock = ['PTRMSmockcalsequence' id_curr_mock '.h5'];


% ========================================================================
% Defines the exact number of timepoints for the current mockfile being
% produced (I want each mockfile to cover approximately 30s and being ~0.3s
% the average timestep in the MS data acquisition system it means that the
% number of timesteps is 100. As I want to be flexible and allow already
% for good adaptability towards the files the platform will be working with
% I randomly assign a number of timepoints between 96 and 102. This because
% the chunk size of the data storage system is 6. Every cycle consists of 6
% entries and if the data buffer is emptied at a point in which all 6
% points of the current cycles aren't yet acquired, the system fills the
% remaining entries with zeroes, so to have a valid array structure to
% insert inside of the .h5 file. Similarly, if the number of timepoints
% randomly generated here differs from 96 or 102, there'll be a number of
% zeroes in the final data chunk.
% ========================================================================
rng('shuffle');
low_threshold = ...
    floor((timelength_mock/avg_time_step)/chunk_size)*chunk_size;
high_threshold = low_threshold + chunk_size;
n_timepoints_curr_mock = randi([low_threshold, high_threshold]);
n_timecycles_curr_mock = ceil(n_timepoints_curr_mock/6);
remainder = rem(n_timepoints_curr_mock,chunk_size);
if remainder > 0
    n_zeroes_curr_mock = chunk_size - remainder; % From 1 to 5
else
    n_zeroes_curr_mock = 0;
end

% ========================================================================
% Creates all the .h5 structures for the current mockfile
% ========================================================================
name_time_dtst = '/TimingData/BufTimes'; % Where all the times of the
                                         % single timesteps are stored
time_size_curr_mock = [6 n_timecycles_curr_mock];
h5create(name_curr_mock,name_time_dtst,time_size_curr_mock);
%-------------------------------------------------------------------------
name_mass_dtst = '/FullSpectra/MassAxis'; % Where the vector containing all
                                          % of the measured masses is
                                          % stored
masses_length = round(spectrum_extension/avg_mass_step);
mass_size_curr_mock = [masses_length 1];
h5create(name_curr_mock,name_mass_dtst,mass_size_curr_mock);
%-------------------------------------------------------------------------
name_tofdata_dtst = '/FullSpectra/TofData'; % Where ions/s data are stored
                                            % for every time point and for
                                            % every recorded mass
tofdata_size_curr_mock = ...
    [masses_length, ...
    1, ...
    chunk_size, ...
    n_timecycles_curr_mock];
h5create(name_curr_mock,name_tofdata_dtst,tofdata_size_curr_mock);
%--------------------------------------------------------------------------
name_log_dtst = '/AcquisitionLog/Log'; % Where the info about the number of
                                       % of completed cycles and zeroes in
                                       % the last cycle is stored
% log_size_curr_mock = [1 (91 + length(num2str(n_timecycles_curr_mock)))];
log_size_curr_mock = [1 1];
h5create(name_curr_mock,name_log_dtst,log_size_curr_mock, ...
    'Datatype', 'string');


% ========================================================================
% Evaluates the starting timepoint based on the presence/absence of
% previous mockfiles
% ========================================================================
if no_prev_files_flag == 1
    start_time_curr_mock = 0;
else
    % Identifies the file to open to extract the info related to time
    if n_curr_mock >= 2 && n_curr_mock <= 10
        id_last_mock = ['00' num2str(n_last_mock)];
    elseif n_curr_mock > 10 && n_curr_mock <= 100
        id_last_mock = ['0' num2str(n_last_mock)];
    else
        id_last_mock = num2str(n_last_mock);
    end

    % Opens time dataset and file log
    name_last_mock = ['PTRMSmockcalsequence' id_last_mock '.h5'];
    data_time_last_mock = h5read(name_last_mock, name_time_dtst);
    log_last_mock = char(h5read(name_last_mock, name_log_dtst));

    % Extracts no. of cycles & zeroes from log
    spot = '';
    cycles = '';
    while strcmp(spot, ' ') == 0
        spot = log_last_mock(cycles_pos);
        cycles = [cycles spot];
        cycles_pos = cycles_pos + 1;
    end

    n_timecycles_last_mock = str2double(cycles(1:(end-1)));
    n_zeroes_last_mock = ...
        str2double(log_last_mock( ...
        cycles_pos + shift_to_zeroes_pos ...
        ));

    % Extracts last recorded timepoint and adds timestep to get the
    % starting point of the current mockfile
    if n_zeroes_last_mock > 0
        start_time_curr_mock = ...
            data_time_last_mock((end - n_zeroes_last_mock), end) + ...
            avg_time_step;
    else
        start_time_curr_mock = ...
            data_time_last_mock(end,end) + ...
            avg_time_step;
    end

end

% ========================================================================
% Switch between the functions for the creation of the mock intensity
% profile. Switch is based on the variable 'distr_fun'
% ========================================================================
switch distr_fun
%-------------------------------------------------------------------------
    case 'stable'
% ========================================================================
% Define a random distribution parameters over time of the ions count.
% This distributions defines the profile over time of the central
% mass to be analysed. It employs the probability density function 'pdf'
% based on the distribution given by 'makedist' (which in turn employs a
% 4-parameters stable distribution). Distribution parameters are randomly
% picked amongst pre-defined ranges that are found to be suitable for the
% desired profile to output.
% ========================================================================
lim_alpha = [1 2];
lim_beta = [-1 1];
lim_gamma = [10 timelength_mock];

% Defines the delta par based on the time range sampled. Delta is related
% to the central position of the distribution, therefore it needs to
% account for the actually measured time chunk
if no_prev_files_flag == 1
    lim_delta = [0 30];
else
    lim_delta = [(start_time_curr_mock) ...
        (start_time_curr_mock + n_timepoints_curr_mock*avg_time_step)];
end

% Randomly defines the 4 parameters within the defined ranges
lims = [lim_alpha; lim_beta; lim_gamma; lim_delta];
for i = 1:4
    if i == 1
        alpha = (lims(i,2) - lims(i,1))*rand + lims(i,1);
    elseif i == 2
        beta = (lims(i,2) - lims(i,1))*rand + lims(i,1);
    elseif i == 3
        gamma = (lims(i,2) - lims(i,1))*rand + lims(i,1);
    elseif i == 4
        delta = (lims(i,2) - lims(i,1))*rand + lims(i,1);
    end
end

% Creates central distribution
scaling_factor = 100000; % Takes output of distribution function to values
                         % similar to those given by actual ions count on
                         % real measurement of PTRMS

times = linspace(start_time_curr_mock, ...
    (start_time_curr_mock + avg_time_step*(n_timepoints_curr_mock-1)), ...
    n_timepoints_curr_mock);
distribution = ...
    makedist('Stable','alpha',alpha,'beta',beta,'gam',gamma,'delta',delta);
central_ions_distr = pdf(distribution,times);
central_ions_distr = central_ions_distr*scaling_factor;


%-------------------------------------------------------------------------
    case 'sinusoidal'
% ========================================================================
% Define a random distribution parameters over time of the ions count.
% This distributions defines the profile over time of the central
% mass to be analysed. It employs a 'sinusoidal' function, slowed down to
% guarantee a non-excessive oscillation. Distribution parameters are
% randomly picked amongst pre-defined ranges that are found to be suitable
% for the desired profile to output.
% ========================================================================
lim_slow_factor = [2.5 4];
```

```
lim_scaling_factor = [500 5000]; % Takes output of distribution function to
                                 % values similar to those given by actual
                                 % ions count on real measurement of PTRMS

slow_factor = (lim_slow_factor(2) - lim_slow_factor(1))*rand + ...
    lim_slow_factor(1);
scaling_factor = (lim_scaling_factor(2) - lim_scaling_factor(1))*rand + ...
    lim_scaling_factor(1);

times = linspace(start_time_curr_mock, ...
    (start_time_curr_mock + avg_time_step*(n_timepoints_curr_mock-1)), ...
    n_timepoints_curr_mock);

central_ions_distr = scaling_factor*sin(times/slow_factor);

if no_prev_files_flag == 1
    central_ions_distr = central_ions_distr + lim_scaling_factor(2);
end


%---------------------------------------------------------------------------
    case 'constant'
% ==========================================================================
% Define a random distribution parameters over time of the ions count.
% This distributions defines the profile over time of the central
% mass to be analysed. It employs a 'sinusoidal' function, slowed down to
% guarantee a non-excessive oscillation. Distribution parameters are
% randomly picked amongst pre-defined ranges that are found to be suitable
% for the desired profile to output.
% ==========================================================================
times = linspace(start_time_curr_mock, ...
    (start_time_curr_mock + avg_time_step*(n_timepoints_curr_mock-1)), ...
n_timepoints_curr_mock);

output = signal_intensity;

central_ions_distr = linspace( ...
    output, ...
    output, ...
    n_timepoints_curr_mock);
%---------------------------------------------------------------------------
end


% ==========================================================================
% Generates the orthogonal distribution related to the masses. Target mass
% isn't the only one that registers ions related to the target coumpound,
% also a small neighbourhood records ions which are fundamental to the
% accurate measurement of the concentration of the target compound.
% Therefore, it is desirable to simulate the same profile in the mock file.
% To do this, a normal distribution is generated. Combining the ions
% distribution over time with the distribution over the masses, will
% generate the surface of peaks across both masses and timepoints.
% ==========================================================================
% Generates the whole list of masses that the PTRMS will analyse
data_mass_curr_mock = ...
    [beg_mass_rng:...
    avg_mass_step:...
    (beg_mass_rng + (masses_length - 1)*avg_mass_step)];
data_mass_curr_mock = data_mass_curr_mock';

% Extracts the mass range in which the mock profile will be generated
% according to the user defined mass value whose peak must be simulated
[~,ix_mass_choice] = min(abs(data_mass_curr_mock - mass_of_choice));
ixs_neighbourhood = round(neighbourhood_mock/avg_mass_step);
masses_mock = data_mass_curr_mock(...
    ix_mass_choice - ixs_neighbourhood:...
    ix_mass_choice + ixs_neighbourhood...
    );


% Generates the distribution
norm_distr = normpdf( ...
    masses_mock, ... % Created across the prev identified range of masses
    mass_of_choice, ... % Centre is on the target mass
    neighbourhood_mock/3); % Gives quite a narrow distr, okay for purpose
norm_distr = norm_distr/max(norm_distr); % Normalise to get 'height' values
                                         % of the distr to be 0:1


% ==========================================================================
% Generates the mixed time-mass distribution of the ion peaks. Then it adds
% the zeroes to maintain proper chunk size, and lastly it structures the
% data in the same way they are stored in the original .h5 files.
% ==========================================================================
cross_ions_distr = zeros(length(masses_mock),n_timepoints_curr_mock);
for i = 1:length(masses_mock)
    cross_ions_distr(i,:) = norm_distr(i)*central_ions_distr;
end

% Adds zeroes
if n_zeroes_curr_mock == 0
    cross_ions_distr_w_nulls = cross_ions_distr;
else
    cross_ions_distr_w_nulls = [ ...
        cross_ions_distr, ...
        zeros(length(masses_mock), n_zeroes_curr_mock) ...
        ];
end

% Rearrange data to be written to .h5 file
data_tofdata_curr_mock = ...
    zeros(length(masses_mock), 1, 6, n_timecycles_curr_mock);
for i = 1:length(masses_mock)
    for j = 1:n_timecycles_curr_mock
        data_tofdata_curr_mock(i,1,:,j) = ...
            cross_ions_distr_w_nulls( ...
            i, ...
            ((j-1)*chunk_size+1):((j-1)*chunk_size+chunk_size) ...
            );
    end
end


% ==========================================================================
% Organises and writes data on the previously generated datasets of the
% mockfile
% ==========================================================================
data_time_curr_mock = zeros(time_size_curr_mock);
if n_zeroes_curr_mock == 0
    for i = 1:n_timecycles_curr_mock
        data_time_curr_mock(:,i) = ...
            times((((i-1)*chunk_size)+1):(((i-1)*chunk_size)+chunk_size));
    end
else
    for i = 1:(n_timecycles_curr_mock - 1)
        data_time_curr_mock(:,i) = ...
            times((((i-1)*chunk_size)+1):(((i-1)*chunk_size)+chunk_size));
    end
    data_time_curr_mock(:,end) = ...
        [times(((n_timecycles_curr_mock-1)*chunk_size+1):end), ...
        zeros(1, n_zeroes_curr_mock)];
```

```
end

h5write(name_curr_mock,name_time_dtst,data_time_curr_mock);
%--------------------------------------------------------------------------
h5write(name_curr_mock,name_mass_dtst,data_mass_curr_mock);
%--------------------------------------------------------------------------
write_start = [ ...
    (ix_mass_choice - ixs_neighbourhood), ...
    1, ...
    1, ...
    1 ...
    ];
write_count = [length(masses_mock) 1 6 n_timecycles_curr_mock];

h5write( ...
    name_curr_mock, ...
    name_tofdata_dtst, ...
    data_tofdata_curr_mock, ...
    write_start, ...
    write_count ...
    );
%--------------------------------------------------------------------------
log_curr_mock = string([ ...
    'Acquisition aborted after ' ...
    num2str(n_timecycles_curr_mock) ...
    ' complete writes. ' ...
    num2str(n_zeroes_curr_mock) ...
    ' additional bufs in the incomplete last write.' ...
    ]);

h5write(name_curr_mock,name_log_dtst,log_curr_mock)

end
```

BioControl 1.0:

<u>Main interface</u>

```
classdef platform_test_test < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        BioControl                          matlab.ui.Figure
        FileMenu                            matlab.ui.container.Menu
        OpenfolderMenu                      matlab.ui.container.Menu
        GridLayout                          matlab.ui.container.GridLayout
        LeftPanel                           matlab.ui.container.Panel
        StatusPanel                         matlab.ui.container.Panel
        DataacquisitionLampLabel            matlab.ui.control.Label
        DataacquisitionLamp                 matlab.ui.control.Lamp
        ErrorlogTextAreaLabel               matlab.ui.control.Label
        ErrorlogTextArea                    matlab.ui.control.TextArea
        ChooseoperationalmodalityButtonGroup  matlab.ui.container.ButtonGroup
        SimulationButton                    matlab.ui.control.RadioButton
        SampletestingButton                 matlab.ui.control.RadioButton
        ChoosePTRMSdatafilesfolderButton  matlab.ui.control.Button
        FolderSelectedLamp                  matlab.ui.control.Lamp
        MonitoringtargetPanel               matlab.ui.container.Panel
        TargetmassEditFieldLabel            matlab.ui.control.Label
        TargetmassEditField                 matlab.ui.control.NumericEditField
        mqLabel_4                       matlab.ui.control.Label
        NeighbourhoodEditFieldLabel     matlab.ui.control.Label
        NeighbourhoodEditField          matlab.ui.control.NumericEditField
        mqLabel_5                       matlab.ui.control.Label
        STARTButton                     matlab.ui.control.Button
        STOPButton                      matlab.ui.control.Button
        ElapsedtimeEditFieldLabel       matlab.ui.control.Label
        ElapsedtimeEditField            matlab.ui.control.EditField
        RightPanel                      matlab.ui.container.Panel
        TabGroup                        matlab.ui.container.TabGroup
        CalibrationTab                  matlab.ui.container.Tab
        CalibrationPanel                matlab.ui.container.Panel
        StartnewcalibrationButton       matlab.ui.control.Button
        CalibrationacquiredLampLabel    matlab.ui.control.Label
        CalibrationacquiredLamp         matlab.ui.control.Lamp
        ChoosecalibrationfileButton     matlab.ui.control.Button
        UIAxesCal                       matlab.ui.control.UIAxes
        CalibrationparametersPanel      matlab.ui.container.Panel
        LOG10INTENSITYLabel             matlab.ui.control.Label
        SlopeEditField                  matlab.ui.control.NumericEditField
        xLOG10CONCLabel                 matlab.ui.control.Label
        InterceptEditField              matlab.ui.control.NumericEditField
        SimulationTab                   matlab.ui.container.Tab
        Panel2_3                        matlab.ui.container.Panel
        MockfiletimespanEditFieldLabel  matlab.ui.control.Label
        MockfiletimespanEditField       matlab.ui.control.NumericEditField
        sLabel                          matlab.ui.control.Label
        SimulatedmassEditFieldLabel     matlab.ui.control.Label
        SimulatedmassEditField          matlab.ui.control.NumericEditField
        mqLabel                         matlab.ui.control.Label
        Panel2_5                        matlab.ui.container.Panel
        MockfilegenerationLampLabel     matlab.ui.control.Label
        MockfilegenerationLamp          matlab.ui.control.Lamp
        UIAxesSim                       matlab.ui.control.UIAxes
        MonitoringTab                   matlab.ui.container.Tab
        UIAxesMon1                      matlab.ui.control.UIAxes
        UIAxesMon2                      matlab.ui.control.UIAxes
        HistoryTab                      matlab.ui.container.Tab
        WholeHistory                    matlab.ui.control.UIAxes
        SlidingHistory                  matlab.ui.control.UIAxes
        TimeSlider                      matlab.ui.control.Slider
        ImportoptionsPanel              matlab.ui.container.Panel
        ChoosefolderfileButton          matlab.ui.control.Button
        ImportexperimentButton          matlab.ui.control.Button
        HistoryDataLamp                 matlab.ui.control.Lamp
        ImportGCMScontrolsButton        matlab.ui.control.Button
        ChoosetoimportDropDownLabel     matlab.ui.control.Label
        ChoosetoimportDropDown          matlab.ui.control.DropDown
        GraphoptionsPanel               matlab.ui.container.Panel
        PlottedtimeintervalminSpinnerLabel  matlab.ui.control.Label
        PlottedtimeintervalminSpinner   matlab.ui.control.Spinner
    end

    % Properties that correspond to apps with auto-reflow
    properties (Access = private)
        onePanelWidth = 576;
    end


    properties (Access = private)
        TimerMock % Timer object to time the generation of the mockfiles
        TimerAnalysisMock % Timer object to time the acquisition of data from
                          % the mockfiles
        TimerAnalysis % Timer object to time the acquisition of data from
                      % the PTRMS files
        TimerStopWatch % Timer object to account for elapsed time
        TicTimer % Object to indicate the startpoint of the stopwatch
        LastMock % Name of the last mockfile that has been generated
        ErrorNumber % Indicator of the consecutive error to be logged
        CalSlope % Slope of the current calibration
        CalIntercept % Intercept of the current calibration
        DataFolder % Folder containing the data outputted by the PTRMS
        LastAnalysedFile % Name of the last file that has been loaded
        NoAnalysedFiles % Number of files analysed so far
        StartTimerVec % Vector indicating the starting time of the
                      % current acquisition
        HistoryData % Folder containing all of teh files acquired during
                    % an experiment (can be a pointer to both a folder or a
                    % file)
        EndOfSlider % Last time value of the current monitoring history
                    % that has been imported
        TimeToPlotSlide % Array of time values from the imported monitoring
                        % history
        ConcToPlotSlide % Array of conc values from the imported monitoring
                        % history
    end

    methods (Access = private)


%***************************************************************************
%***************************************************************************
% FUNCTION 1
%***************************************************************************
%***************************************************************************
        function ClearGraphsNoCalFcn(app,~,~)
            % Stores info on current A/R and position
            aspect_ratio1 = app.UIAxesSim.PlotBoxAspectRatio;
            position1 = app.UIAxesSim.Position;
            % Wipes UIAxesSim
            cla(app.UIAxesSim,'reset');
            % Creates new axes
            app.UIAxesSim = uiaxes(app.SimulationTab);
            title(app.UIAxesSim, '')
            xlabel(app.UIAxesSim, 'Time [s]')
            ylabel(app.UIAxesSim, 'Intensity [ions]')
            app.UIAxesSim.PlotBoxAspectRatio = aspect_ratio1;
            app.UIAxesSim.Position = position1;


            % Stores info on current A/R and position
            aspect_ratio2 = app.UIAxesMon1.PlotBoxAspectRatio;
            position2 = app.UIAxesMon1.Position;
            % Wipes UIAxesMon1
            cla(app.UIAxesMon1,'reset');
            % Creates new axes
            app.UIAxesMon1 = uiaxes(app.MonitoringTab);
            title(app.UIAxesMon1, 'Raw headspace ion count')
            xlabel(app.UIAxesMon1, 'Time [min]')
            ylabel(app.UIAxesMon1, 'Intensity [ions]')
            app.UIAxesMon1.PlotBoxAspectRatio = aspect_ratio2;
            app.UIAxesMon1.Position = position2;


            % Stores info on current A/R and position
            aspect_ratio3 = app.UIAxesMon2.PlotBoxAspectRatio;
            position3 = app.UIAxesMon2.Position;
            % Wipes UIAxesMon2
            cla(app.UIAxesMon2,'reset');
            % Creates new axes
            app.UIAxesMon2 = uiaxes(app.MonitoringTab);
            title(app.UIAxesMon2,['2 min rolling average liquid ' ...
                'concentration'])
            xlabel(app.UIAxesMon2, 'Time [min]')
            ylabel(app.UIAxesMon2, 'Concentration [mg/L]')
            app.UIAxesMon2.PlotBoxAspectRatio = aspect_ratio3;
            app.UIAxesMon2.Position = position3;
        end



%***************************************************************************
%***************************************************************************
% FUNCTION 2
%***************************************************************************
%***************************************************************************
        function GenerateMockFcn(app,~,~)
            user_mass= app.SimulatedmassEditField.Value;
            timelength_mock= app.MockfiletimespanEditField.Value;
            app.LastMock.Value = ...
                genaddh5mock(user_mass,timelength_mock);
        end



%***************************************************************************
%***************************************************************************
% FUNCTION 3
%***************************************************************************
%***************************************************************************
        function StartFcn(app,~,~)
            ClearGraphsNoCalFcn(app)

            if app.SimulationButton.Value == true
                start(app.TimerAnalysisMock)
            else
                start(app.TimerAnalysis)

            end
        end



%***************************************************************************
%***************************************************************************
% FUNCTION 4
%***************************************************************************
%***************************************************************************
        function ReadMockFcn(app,~,~)
            app.DataacquisitionLamp.Color = [0 1 0];

            filename = app.LastMock.Value;
            central_mass = app.TargetmassEditField.Value + 1;
            neighbourhood = app.NeighbourhoodEditField.Value;
            [cumpeakprof,~,times] = geth5mrcumpeaks( ...
                filename, ...
                central_mass, neighbourhood);

            % ==============================================================
            % Saves time and central ion profile to plot over time
            % ==============================================================
            cumpeakprof = cumpeakprof';
            times = times';

            mocks = dir('PTRMSmocksequence*.h5');
            n_mocks = length(mocks);

            if n_mocks <= 1
```

```
                x = times;
                y = cumpeakprof;
                RawInt = y;
                Time = x;
            else
                load('whole_simulation.mat')
                x = [x; times];
                y = [y; cumpeakprof];
                RawInt = y;
                Time = x;
            end

            save('whole_simulation.mat','x','y','Time','RawInt')


            % ============================================================
            % Plots ions peaks distribution across elapsed time since the
            % first mockfile that has been generated
            % ============================================================
            plot(app.UIAxesSim,times,cumpeakprof)
            plot(app.UIAxesMon1,x,y)

            rolling_rng = 360; % Timespan for rolling range graph (~=2mins)
            y = (y - app.CalIntercept.Value)./app.CalSlope.Value;

            len = length(x);
            if len > rolling_rng
                X = x(rolling_rng:end);
                Y = zeros((len - (rolling_rng - 1)),1);
                for i = rolling_rng:len
                    Y(i - (rolling_rng - 1)) = ...
                        sum(y((i - (rolling_rng - 1)):i))/rolling_rng;
                end

                plot(app.UIAxesMon2,X,Y)
            end

            % ============================================================
            % Gets rid of past, useless mockfiles (leaves 5 buffer files)
            % ============================================================
            if n_mocks > 5
                delete(mocks(1).name)
            end

        end



%************************************************************************
%************************************************************************
% FUNCTION 5
%************************************************************************
%************************************************************************
        function ReadFcn(app,~,~)
            app.DataacquisitionLamp.Color = [0 1 0];

            cd(app.DataFolder.Value)
            filelist = dir('*.h5');

            numeric_dates = zeros(length(filelist),1);
            for j = 1:length(filelist)
                numeric_dates(j) = datenum(filelist(j).date);
            end
            [~, index]= max(numeric_dates);
            filename = filelist(index).name;
```

```
check = strcmp(filename(1), '2');
if ~check
    return
end


if strcmp(filename, app.LastAnalysedFile.Value)
    return
end


app.NoAnalysedFiles.Value = app.NoAnalysedFiles.Value + 1;

central_mass = app.TargetmassEditField.Value + 1;
neighbourhood = app.NeighbourhoodEditField.Value;
[cumpeakprof,~,times] = geth5mrcumpeaks( ...
    filename, ...
    central_mass, neighbourhood);

% ==============================================================
% Saves time and central ion profile to plot over time
% ==============================================================
cumpeakprof = cumpeakprof';
times = times';

FormatInput = 'yyyymmdd_HHMMSS';
[~,DateStringCurr,~] = fileparts(filename);

if isempty(app.LastAnalysedFile.Value)
    app.StartTimerVec.Value = ...
        datevec(DateStringCurr,FormatInput);
    x = times;
    y = cumpeakprof;
    RawInt = y;
    Time = x;
else
    load('whole_simulation.mat')
    endtime_current_file = datevec(DateStringCurr,FormatInput);

    [~,DateStringPast,~] = ...
        fileparts(app.LastAnalysedFile.Value);
    endtime_past_file = datevec(DateStringPast,FormatInput);


    t_elapsed_from_beg = ...
        etime(endtime_past_file, app.StartTimerVec.Value) ...
        + times(end);
    t_between_files = ...
        etime(endtime_current_file, endtime_past_file);
    addition = ...
        t_elapsed_from_beg + t_between_files - times(end);

    times = times + addition;

    x = [x; times];
    y = [y; cumpeakprof];
    RawInt = y;
    Time = x;

end

save('whole_simulation.mat','x','y','Time','RawInt')


% ==============================================================
            % Plots ions peaks distribution across elapsed time since the
            % first mockfile that has been generated
            % ==============================================================
            if length(x) < 1000
                plot(app.UIAxesMon1, x./60, y)
                xlim(app.UIAxesMon1, [0 5])
                xlim(app.UIAxesMon2, [0 5])
            else
                plot(app.UIAxesMon1, x((end-999):end)./60,y((end-999):end))
                xlim(app.UIAxesMon1, [(x(end)./60 - 5) x(end)./60])
                xlim(app.UIAxesMon2, [(x(end)./60 - 5) x(end)./60])
            end

            rolling_rng = 400; % Timespan for rolling range graph (~=2mins)
            y = 0.858*0.001*(10.^...
                ((log10(y) - app.CalIntercept.Value)./app.CalSlope.Value));

            len = length(x);
            if len > rolling_rng && len <= 1500
                X = x(rolling_rng:end)./60;
                Y = [];
                for i = rolling_rng:len
                    Y = [Y; ...
                        sum(y((i - (rolling_rng-1)):i))/rolling_rng];
                end

                plot(app.UIAxesMon2,X,Y)
            elseif len > 1500
                X = x((end-999):end)./60;
                Y = [];
                for i = (len-999):len
                    Y = [Y ...
                        sum(y((i - (rolling_rng-1)):i))/rolling_rng];
                end

                plot(app.UIAxesMon2,X,Y)
            end

            app.LastAnalysedFile.Value = filename;
        end


%**************************************************************************
%**************************************************************************
% FUNCTION 6
%**************************************************************************
%**************************************************************************
        function StopWatchStartFcn(app,~,~)
            TocTimer = 0;
            TocTimer = seconds(TocTimer);
            TocTimer.Format = 'hh:mm:ss';
            text = char(TocTimer);
            app.ElapsedtimeEditField.Value = text;
        end



%**************************************************************************
%**************************************************************************
% FUNCTION 7
%**************************************************************************
%**************************************************************************
        function StopWatchFcn(app,~,~)
            TocTimer = toc;
            TocTimer = seconds(TocTimer);
            TocTimer.Format = 'hh:mm:ss';
            text = char(TocTimer);
            app.ElapsedtimeEditField.Value = text;
        end


    end

    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            format long

            app.ErrorNumber.Value = 0;
            app.LastMock.Value = '';
            app.DataFolder.Value = '';
            app.LastAnalysedFile.Value = '';
            app.NoAnalysedFiles.Value = 0;
            app.StartTimerVec.Value = [];
            app.HistoryData.Value = '';
            app.EndOfSlider.Value = 0;
            app.TimeToPlotSlide.Value = [];
            app.ConcToPlotSlide.Value = [];

            app.MockfilegenerationLamp.Color = [.8 .8 .8];
            app.DataacquisitionLamp.Color = [.8 .8 .8];
            app.CalibrationacquiredLamp.Color = [.8 .8 .8];
            app.FolderSelectedLamp.Color = [.8 .8 .8];

            app.MockfiletimespanEditField.Value = 30;
            app.SimulatedmassEditField.Value = 154.25;
            app.TargetmassEditField.Value = 154.25;
            app.NeighbourhoodEditField.Value = 0.2;

            app.ElapsedtimeEditField.Value = '00:00:00';



            delete 'PTRMSmocksequence*.h5' 'whole_simulation*'

            period = 30;    %Period for timer (in seconds)

            app.TimerMock = timer(...
                'ExecutionMode', 'fixedRate', ...
                'Period', period, ...
                'BusyMode', 'queue');
            app.TimerMock.TimerFcn = @(~,~) app.GenerateMockFcn;
            app.TimerMock.StartFcn = @(~,~) app.StartFcn;

            app.TimerAnalysisMock = timer(...
                'StartDelay', 2, ...
                'ExecutionMode', 'fixedRate', ...
                'Period', period, ...
                'BusyMode', 'queue');
            app.TimerAnalysisMock.TimerFcn = @(~,~) app.ReadMockFcn;

            app.TimerAnalysis = timer(...
                'ExecutionMode', 'fixedRate', ...
                'Period', 5, ...
                'BusyMode', 'queue');
            app.TimerAnalysis.TimerFcn = @(~,~) app.ReadFcn;

            app.TimerStopWatch = timer(...
                'ExecutionMode', 'fixedRate', ...
                'Period', 0.25, ...
                'BusyMode', 'queue');
            app.TimerStopWatch.TimerFcn = @(~,~) app.StopWatchFcn;
            app.TimerStopWatch.StartFcn = @(~,~) app.StopWatchStartFcn;
        end

        % Button pushed function: STARTButton
        function STARTButtonPushed(app, event)
            % Checks that all the necessary inputs are non-null
            % Checks on calibration to be acquired
            check = app.CalibrationacquiredLamp.Color == [0 1 0];
            if ~check
                app.ErrorNumber.Value = app.ErrorNumber.Value + 1;
                heading = ['ERROR NO.' num2str(app.ErrorNumber.Value)];
                text = ['Need to import calibration first.'];
                app.ErrorlogTextArea.Value = ...
                    [heading ...
                    newline text ...
                    newline ' ' ...
                    newline ' '];
                return
            end


            % Checks on target mass
            check = app.TargetmassEditField.Value <= 1;
            if check
                app.ErrorNumber.Value = app.ErrorNumber.Value + 1;
                heading = ['ERROR NO.' num2str(app.ErrorNumber.Value)];
                text = ['Target mass needs to be higher than 1.'];
                app.ErrorlogTextArea.Value = ...
                    [heading ...
                    newline text ...
                    newline ' ' ...
                    newline ' '];
                return
            end

            %Checks on neighbourhood
            check = app.NeighbourhoodEditField.Value <= 0;
            if check
                app.ErrorNumber.Value = app.ErrorNumber.Value + 1;
                heading = ['ERROR NO.' num2str(app.ErrorNumber.Value)];
                text = ['Neighbourhood needs to be a positive number.'];
                app.ErrorlogTextArea.Value = ...
                    [heading ...
                    newline text ...
                    newline ' ' ...
                    newline ' '];
                return
            end

            %Checks on simulation timespan
            simulation = app.SimulationButton.Value == true;
            check = app.MockfiletimespanEditField.Value <= 5;
            if simulation && check
                app.ErrorNumber.Value = app.ErrorNumber.Value + 1;
                heading = ['ERROR NO.' num2str(app.ErrorNumber.Value)];
                text = ['The timespan covered by each mockfile ' ...
                    'can''t be lower than 5s.'];
                app.ErrorlogTextArea.Value = ...
                    [heading ...
                    newline text ...
                    newline ' ' ...
                    newline ' '];
                return
            end
            %Checks on simulated mass
            simulation = app.SimulationButton.Value == true;
            check = app.SimulatedmassEditField.Value <= 1;
            if simulation && check
                app.ErrorNumber.Value = app.ErrorNumber.Value + 1;
                heading = ['ERROR NO.' num2str(app.ErrorNumber.Value)];
                text = ['Simulated mass needs to be higher than 1.'];
                app.ErrorlogTextArea.Value = ...
                    [heading ...
                    newline text ...
                    newline ' ' ...
                    newline ' '];
                return
            end

            %Checks on selected folder for PTRMS datafiles
            if app.SimulationButton.Value == false
                check = isempty(app.DataFolder.Value);
                if check
                    app.ErrorNumber.Value = app.ErrorNumber.Value + 1;
                    heading = ['ERROR NO.' num2str(app.ErrorNumber.Value)];
                    text = ['Need to select the folder containing ' ...
                        'PTRMS datafiles when operating in ''Sample ' ...
                        'Testing'' mode.'];
                    app.ErrorlogTextArea.Value = ...
                        [heading ...
                        newline text ...
                        newline ' ' ...
                        newline ' '];
                    return
                end
            end


            close all
            delete 'PTRMSmocksequence*.h5' 'whole_simulation*'

            app.LastAnalysedFile.Value = '';
            app.StartTimerVec.Value = [];
            app.NoAnalysedFiles.Value = 0;

            if app.SimulationButton.Value == true
                start(app.TimerMock)
            else
                StartFcn(app)
            end

            StopwatchStatus = get(app.TimerStopWatch,'Running');
            if strcmp(StopwatchStatus, 'on')
                stop(app.TimerStopWatch)
            end
            start(app.TimerStopWatch)

            app.MockfilegenerationLamp.Color = [0 1 0];

            tic
        end

        % Button pushed function: STOPButton
        function STOPButtonPushed(app, event)
            load('whole_simulation.mat', 'x', 'y')

            check = isempty(x) | isempty(y);
            if ~check
                time = datetime('now','format','yyyy.MM.dd-HH.mm.ss');
                temp = datestr(time,'yyyy.mm.dd-HH.MM.ss');
                name = ['monitoring-' temp '.mat'];
                filter = {'*.mat'};
                [file,path] = uiputfile(filter,'BioControl 1.0',name);

                check = file == 0 || path == 0;
                if check
                    msg = 'Sure about not saving your acquisition?';
                    title = 'BioControl 1.0';
                    selection = uiconfirm(app.BioControl,msg,title,...
                       'Options',{'Don''t save','Cancel'},...
                       'DefaultOption',2,'CancelOption',2, ...
                       'Icon','warning');
                    switch selection
                        case 'Don''t save'
                            stop(app.TimerMock)
                            stop(app.TimerAnalysis)
                            stop(app.TimerStopWatch)

                            app.MockfilegenerationLamp.Color = [.8 .8 .8];
                            app.DataacquisitionLamp.Color = [.8 .8 .8];
                        case 'Cancel'
                            return
                    end
                else
                    Time = x;
                    RawInt = y;
                    directory = fullfile(path, file);
                    save(directory,"Time","RawInt")

                    stop(app.TimerMock)
                    stop(app.TimerAnalysis)
                    stop(app.TimerStopWatch)

                    app.MockfilegenerationLamp.Color = [.8 .8 .8];
                    app.DataacquisitionLamp.Color = [.8 .8 .8];
                end
            else
                stop(app.TimerMock)
                stop(app.TimerAnalysis)
                stop(app.TimerStopWatch)

                app.MockfilegenerationLamp.Color = [.8 .8 .8];
                app.DataacquisitionLamp.Color = [.8 .8 .8];
            end


        end

        % Close request function: BioControl
        function BioControlCloseRequest(app, event)
            stop(app.TimerMock)
            stop(app.TimerAnalysis)
            stop(app.TimerStopWatch)

            close all
            delete 'PTRMSmocksequence*.h5' 'whole_simulation*'

            delete(app)
        end

        % Button pushed function: StartnewcalibrationButton
        function StartnewcalibrationButtonPushed(app, event)
            cal
        end
```

```
        % Button pushed function: ChoosecalibrationfileButton
        function ChoosecalibrationfileButtonPushed(app, event)
            filter = '.xlsx';
            title = 'BioControl 1.0';
            [file, path] = uigetfile(filter, title);

            check = ischar(file) & ischar(path);
            if ~check
                return
            end

            filename = fullfile(path, file);

            fit_parameters = readcell(filename,'Range','A6:B6');
            fit_parameters = cell2mat(fit_parameters);
            exp_points = readcell(filename,'Range','A8:B20');
            for i = 1:length(exp_points)
                check = ismissing(exp_points{i});
                if check
                    exp_points = exp_points(1:(i-1),:);
                end
            end
            exp_points = cell2mat(exp_points);

            X = [exp_points(1,1) exp_points(end,1)];
            X = log10(X);
            Y = X.*fit_parameters(1) + fit_parameters(2);
            plot(app.UIAxesCal,10.^X,10.^Y,'LineWidth', 2,'Color','b');

            hold(app.UIAxesCal, 'on')

            scatter(app.UIAxesCal,exp_points(:,1),exp_points(:,2), ...
                'Marker', 'o', ...
                'MarkerFaceColor', 'r', ...
                'SizeData', 50);

            set(app.UIAxesCal,'YScale','log')
            set(app.UIAxesCal,'XScale','log')
            grid(app.UIAxesCal, 'on')


            app.CalSlope.Value = fit_parameters(1);
            app.CalIntercept.Value = fit_parameters(2);

            app.InterceptEditField.Value = app.CalIntercept.Value;
            app.SlopeEditField.Value = app.CalSlope.Value;

            app.CalibrationacquiredLamp.Color = [0 1 0];
        end

        % Button pushed function: ChoosePTRMSdatafilesfolderButton
        function ChoosePTRMSdatafilesfolderButtonPushed(app, event)
            title = 'BioControl 1.0';
            selpath = uigetdir(title);

            if selpath ~= 0
                app.FolderSelectedLamp.Color = [0 1 0];
                app.DataFolder.Value = selpath;
            end
        end

        % Button pushed function: ChoosefolderfileButton
        function ChoosefolderfileButtonPushed(app, event)
            switch app.ChoosetoimportDropDown.Value
                case 'Raw data'
                    title = 'BioControl 1.0';
                    selpath = uigetdir(title);

                    if selpath ~= 0
                        app.HistoryDataLamp.Color = [0 1 0];
                        app.HistoryData.Value = selpath;
                    end
                case 'Pre-processed data'
                    filter = '.mat';
                    title = 'BioControl 1.0';
                    [file, path] = uigetfile(filter, title);

                    check = ischar(file) & ischar(path);
                    if ~check
                        return
                    end

                    app.HistoryData.Value = fullfile(path, file);
                    app.HistoryDataLamp.Color = [0 1 0];
            end
        end

        % Value changed function: ChoosetoimportDropDown
        function ChoosetoimportDropDownValueChanged(app, event)
            value = app.ChoosetoimportDropDown.Value;
            app.HistoryDataLamp.Color = [.8 .8 .8];
        end

        % Button pushed function: ImportexperimentButton
        function ImportexperimentButtonPushed(app, event)
            % Checks that all the necessary inputs are non-null
            % Checks on folder/file pointer to be present
            check = app.HistoryDataLamp.Color == [0 1 0];
            if ~check
                app.ErrorNumber.Value = app.ErrorNumber.Value + 1;
                heading = ['ERROR NO.' num2str(app.ErrorNumber.Value)];
                text = ['Need to import folder/file first.'];
                app.ErrorlogTextArea.Value = ...
                    [heading ...
                    newline text ...
                    newline ' ' ...
                    newline ' '];
                return
            end

            % Checks on calibration to be acquired
            check = app.CalibrationacquiredLamp.Color == [0 1 0];
            if ~check
                app.ErrorNumber.Value = app.ErrorNumber.Value + 1;
                heading = ['ERROR NO.' num2str(app.ErrorNumber.Value)];
                text = ['Need to import calibration first.'];
                app.ErrorlogTextArea.Value = ...
                    [heading ...
                    newline text ...
                    newline ' ' ...
                    newline ' '];
                return
            end


            % Checks on target mass
            check = app.TargetmassEditField.Value <= 1;
            if check
                app.ErrorNumber.Value = app.ErrorNumber.Value + 1;
                heading = ['ERROR NO.' num2str(app.ErrorNumber.Value)];
                text = ['Target mass needs to be higher than 1.'];
                app.ErrorlogTextArea.Value = ...
```

```
                 [heading ...
                 newline text ...
                 newline ' ' ...
                 newline ' '];
             return
         end

         %Checks on neighbourhood
         check = app.NeighbourhoodEditField.Value <= 0;
         if check
             app.ErrorNumber.Value = app.ErrorNumber.Value + 1;
             heading = ['ERROR NO.' num2str(app.ErrorNumber.Value)];
             text = ['Neighbourhood needs to be a positive number.'];
             app.ErrorlogTextArea.Value = ...
                 [heading ...
                 newline text ...
                 newline ' ' ...
                 newline ' '];
             return
         end

         switch app.ChoosetoimportDropDown.Value
             case 'Raw data'
                 cd(app.HistoryData.Value)
                 filelist = dir('*.h5');

                 Dialog = uiprogressdlg(app.BioControl, ...
                     'Title', 'BioControl 1.0', ...
                     'Message', 'Loading files...');

                 files_count = length(filelist);

                 for i = 1:files_count
                     Dialog.Value = i/files_count;
                     curr_filename = filelist(i).name;

                     central_mass = app.TargetmassEditField.Value + 1;
                     neighbourhood = app.NeighbourhoodEditField.Value;
                     [cumpeakprof,~,times] = geth5mrcumpeaks( ...
                         curr_filename, ...
                         central_mass, neighbourhood);


                     cumpeakprof = cumpeakprof';
                     times = times';

                     FormatInput = 'yyyymmdd_HHMMSS';
                     [~,DateStringCurr,~] = fileparts(curr_filename);

                     if i == 1
                         app.StartTimerVec.Value = ...
                             datevec(DateStringCurr,FormatInput);
                         x = times;
                         y = cumpeakprof;
                     else
                         endtime_current_file = ...
                             datevec(DateStringCurr,FormatInput);

                         [~,DateStringPast,~] = ...
                             fileparts(past_filename);
                         endtime_past_file = ...
                             datevec(DateStringPast,FormatInput);


                         t_elapsed_from_beg = ...
                             etime(...
                             endtime_past_file, ...
                             app.StartTimerVec.Value) ...
                             + times(end);
                         t_between_files = ...
                             etime(...
                             endtime_current_file, endtime_past_file);
                         addition = ...
                             t_elapsed_from_beg + t_between_files ...
                             - times(end);

                         times = times + addition;

                         x = [x; times];
                         y = [y; cumpeakprof];
                     end

                     past_filename = curr_filename;
                 end

                 Time = x;
                 RawInt = y;
                 save('monitoring_history.mat',"Time","RawInt")

                 app.TimeSlider.Enable = 'on';

                 app.EndOfSlider.Value = x(end)/60; % in minutes

                 X = x(400:end);
                 app.TimeToPlotSlide.Value = X;

                 y = 0.858*0.001*...
                     (10.^...
                     ((log10(y)-app.CalIntercept.Value)./...
                     app.CalSlope.Value));

                 Y = [];
                 for i = 400:length(x)
                 Y = [Y; ...
                     sum(y((i - (399)):i))/400];
                 end
                 app.ConcToPlotSlide.Value = Y;

                 plot(app.WholeHistory, ...
                     X(1:300:end)./3600, Y(1:300:end))
                 xlim(app.WholeHistory, [x(1)/3600 x(end)/3600])

                 plot(app.SlidingHistory, ...
                     X(1:(app.PlottedtimeintervalminSpinner.Value*200))./60,...
                     Y(1:(app.PlottedtimeintervalminSpinner.Value*200)))
                 xlim(app.SlidingHistory, ...
                     [0 app.PlottedtimeintervalminSpinner.Value])

                 close(Dialog)

             case 'Pre-processed data'
                 load(app.HistoryData.Value)
                 x = Time;
                 y = RawInt;

                 app.TimeSlider.Enable = 'on';

                 app.EndOfSlider.Value = x(end)/60; % in minutes

                 X = x(400:end);
                 app.TimeToPlotSlide.Value = X;
```

```
                y = 0.858*0.001*...
                    (10.^...
                    ((log10(y)-app.CalIntercept.Value)./...
                    app.CalSlope.Value));

                Y = [];
                for i = 400:length(x)
                Y = [Y; ...
                    sum(y((i - (399)):i))/400];
                end
                app.ConcToPlotSlide.Value = Y;

                plot(app.WholeHistory, ...
                    X(1:300:end)./3600, Y(1:300:end))
                xlim(app.WholeHistory, [x(1)/3600 x(end)/3600])

                plot(app.SlidingHistory, ...
                    X(1:(app.PlottedtimeintervalminSpinner.Value*200))./60,...
                    Y(1:(app.PlottedtimeintervalminSpinner.Value*200)))
                xlim(app.SlidingHistory, ...
                    [0 app.PlottedtimeintervalminSpinner.Value])
        end
    end

    % Value changing function: TimeSlider
    function TimeSliderValueChanging(app, event)
        changingValue = event.Value;

        tot_range = ...
            app.EndOfSlider.Value ...
            - app.PlottedtimeintervalminSpinner.Value;
        plot_range = [(changingValue*tot_range), ...
            (changingValue*tot_range ...
            + app.PlottedtimeintervalminSpinner.Value)];

        indexes = ...
            find((app.TimeToPlotSlide.Value./60) > plot_range(1) & ...
            (app.TimeToPlotSlide.Value./60) < plot_range(2));

        plot(app.SlidingHistory, ...
            app.TimeToPlotSlide.Value(indexes)./60, ...
            app.ConcToPlotSlide.Value(indexes))
        xlim(app.SlidingHistory, plot_range)

        ylim(app.SlidingHistory, ...
            [min(app.ConcToPlotSlide.Value(indexes)), ...
            max(app.ConcToPlotSlide.Value(indexes))])

    end

    % Button pushed function: ImportGCMScontrolsButton
    function ImportGCMScontrolsButtonPushed(app, event)
        condition = true;
        while condition
            prompt = {'Enter time value [min]:', ...
                'Enter concentration value [mg/L];'};
            dlgtitle = 'BioControl 1.0';
            dims = [1 50];
            answer = inputdlg(prompt, dlgtitle, dims);

            check = isempty(answer);
            if check
                return
            end

            x = str2double(answer{1});
```

```
                  y = str2double(answer{2});

                  check = isempty(answer{1}) | isempty(answer{2}) | ...
                      isnan(x) | isnan(y);
                  if check
                      uialert(app.BioControl, ...
                          'Entered inputs are not valid!', ...
                          'BioControl 1.0', 'Icon', 'error');
                      condition = false;
                  else
                      hold(app.WholeHistory, 'on')
                      scatter(app.WholeHistory,x./60,y, ...
                          'Marker', 'o', ...
                          'MarkerFaceColor', 'r', ...
                          'SizeData', 50);
                      hold(app.WholeHistory, 'off')
                  end
              end
          end

          % Changes arrangement of the app based on UIFigure width
          function updateAppLayout(app, event)
              currentFigureWidth = app.BioControl.Position(3);
              if(currentFigureWidth <= app.onePanelWidth)
                  % Change to a 2x1 grid
                  app.GridLayout.RowHeight = {655, 655};
                  app.GridLayout.ColumnWidth = {'1x'};
                  app.RightPanel.Layout.Row = 2;
                  app.RightPanel.Layout.Column = 1;
              else
                  % Change to a 1x2 grid
                  app.GridLayout.RowHeight = {'1x'};
                  app.GridLayout.ColumnWidth = {236, '1x'};
                  app.RightPanel.Layout.Row = 1;
                  app.RightPanel.Layout.Column = 2;
              end
          end
      end

      % Component initialization
      methods (Access = private)

          % Create UIFigure and components
          function createComponents(app)

              % Create BioControl and hide until all components are created
              app.BioControl = uifigure('Visible', 'off');
              app.BioControl.AutoResizeChildren = 'off';
              app.BioControl.Position = [100 100 1076 655];
              app.BioControl.Name = 'BioControl 1.0';
              app.BioControl.CloseRequestFcn = createCallbackFcn(app,
@BioControlCloseRequest, true);
              app.BioControl.SizeChangedFcn = createCallbackFcn(app,
@updateAppLayout, true);

              % Create FileMenu
              app.FileMenu = uimenu(app.BioControl);
              app.FileMenu.Text = 'File';

              % Create OpenfolderMenu
              app.OpenfolderMenu = uimenu(app.FileMenu);
              app.OpenfolderMenu.Text = 'Open folder';

              % Create GridLayout
              app.GridLayout = uigridlayout(app.BioControl);
              app.GridLayout.ColumnWidth = {236, '1x'};
```

```
            app.GridLayout.RowHeight = {'1x'};
            app.GridLayout.ColumnSpacing = 0;
            app.GridLayout.RowSpacing = 0;
            app.GridLayout.Padding = [0 0 0 0];
            app.GridLayout.Scrollable = 'on';

            % Create LeftPanel
            app.LeftPanel = uipanel(app.GridLayout);
            app.LeftPanel.Layout.Row = 1;
            app.LeftPanel.Layout.Column = 1;

            % Create StatusPanel
            app.StatusPanel = uipanel(app.LeftPanel);
            app.StatusPanel.Title = 'Status';
            app.StatusPanel.Position = [25 15 189 294];

            % Create DataacquisitionLampLabel
            app.DataacquisitionLampLabel = uilabel(app.StatusPanel);
            app.DataacquisitionLampLabel.HorizontalAlignment = 'right';
            app.DataacquisitionLampLabel.Position = [12 237 92 22];
            app.DataacquisitionLampLabel.Text = 'Data acquisition';

            % Create DataacquisitionLamp
            app.DataacquisitionLamp = uilamp(app.StatusPanel);
            app.DataacquisitionLamp.Position = [144 238 20 20];
            app.DataacquisitionLamp.Color = [0.8 0.8 0.8];

            % Create ErrorlogTextAreaLabel
            app.ErrorlogTextAreaLabel = uilabel(app.StatusPanel);
            app.ErrorlogTextAreaLabel.HorizontalAlignment = 'right';
            app.ErrorlogTextAreaLabel.Position = [14 168 55 22];
            app.ErrorlogTextAreaLabel.Text = 'Error log:';

            % Create ErrorlogTextArea
            app.ErrorlogTextArea = uitextarea(app.StatusPanel);
            app.ErrorlogTextArea.Position = [14 10 162 156];

            % Create ChooseoperationalmodalityButtonGroup
            app.ChooseoperationalmodalityButtonGroup = 
uibuttongroup(app.LeftPanel);
            app.ChooseoperationalmodalityButtonGroup.Title = 'Choose operational 
modality';
            app.ChooseoperationalmodalityButtonGroup.Position = [24 496 189 143];

            % Create SimulationButton
            app.SimulationButton = 
uiradiobutton(app.ChooseoperationalmodalityButtonGroup);
            app.SimulationButton.Text = 'Simulation';
            app.SimulationButton.Position = [12 90 78 22];
            app.SimulationButton.Value = true;

            % Create SampletestingButton
            app.SampletestingButton = 
uiradiobutton(app.ChooseoperationalmodalityButtonGroup);
            app.SampletestingButton.Text = 'Sample testing';
            app.SampletestingButton.Position = [12 60 102 22];

            % Create ChoosePTRMSdatafilesfolderButton
            app.ChoosePTRMSdatafilesfolderButton = 
uibutton(app.ChooseoperationalmodalityButtonGroup, 'push');
            app.ChoosePTRMSdatafilesfolderButton.ButtonPushedFcn = 
createCallbackFcn(app, @ChoosePTRMSdatafilesfolderButtonPushed, true);
            app.ChoosePTRMSdatafilesfolderButton.Position = [12 15 102 36];
            app.ChoosePTRMSdatafilesfolderButton.Text = {'Choose PTRMS'; 'datafiles 
folder'};
```

```
            % Create FolderSelectedLamp
            app.FolderSelectedLamp = 
uilamp(app.ChooseoperationalmodalityButtonGroup);
            app.FolderSelectedLamp.Position = [145 23 20 20];
            app.FolderSelectedLamp.Color = [0.8 0.8 0.8];

            % Create MonitoringtargetPanel
            app.MonitoringtargetPanel = uipanel(app.LeftPanel);
            app.MonitoringtargetPanel.Title = 'Monitoring target';
            app.MonitoringtargetPanel.Position = [25 387 189 100];

            % Create TargetmassEditFieldLabel
            app.TargetmassEditFieldLabel = uilabel(app.MonitoringtargetPanel);
            app.TargetmassEditFieldLabel.HorizontalAlignment = 'right';
            app.TargetmassEditFieldLabel.Position = [6 49 70 22];
            app.TargetmassEditFieldLabel.Text = 'Target mass';

            % Create TargetmassEditField
            app.TargetmassEditField = uieditfield(app.MonitoringtargetPanel, 
'numeric');
            app.TargetmassEditField.Position = [102 49 46 22];

            % Create mqLabel_4
            app.mqLabel_4 = uilabel(app.MonitoringtargetPanel);
            app.mqLabel_4.Position = [152 49 27 22];
            app.mqLabel_4.Text = 'm/q';

            % Create NeighbourhoodEditFieldLabel
            app.NeighbourhoodEditFieldLabel = uilabel(app.MonitoringtargetPanel);
            app.NeighbourhoodEditFieldLabel.HorizontalAlignment = 'right';
            app.NeighbourhoodEditFieldLabel.Position = [6 16 89 22];
            app.NeighbourhoodEditFieldLabel.Text = 'Neighbourhood';

            % Create NeighbourhoodEditField
            app.NeighbourhoodEditField = uieditfield(app.MonitoringtargetPanel, 
'numeric');
            app.NeighbourhoodEditField.Position = [102 16 46 22];

            % Create mqLabel_5
            app.mqLabel_5 = uilabel(app.MonitoringtargetPanel);
            app.mqLabel_5.Position = [152 16 27 22];
            app.mqLabel_5.Text = 'm/q';

            % Create STARTButton
            app.STARTButton = uibutton(app.LeftPanel, 'push');
            app.STARTButton.ButtonPushedFcn = createCallbackFcn(app, 
@STARTButtonPushed, true);
            app.STARTButton.FontWeight = 'bold';
            app.STARTButton.FontColor = [0.4667 0.6745 0.1882];
            app.STARTButton.Position = [25 336 88 28];
            app.STARTButton.Text = 'START';

            % Create STOPButton
            app.STOPButton = uibutton(app.LeftPanel, 'push');
            app.STOPButton.ButtonPushedFcn = createCallbackFcn(app, 
@STOPButtonPushed, true);
            app.STOPButton.FontWeight = 'bold';
            app.STOPButton.FontColor = [1 0 0];
            app.STOPButton.Position = [128 336 86 28];
            app.STOPButton.Text = 'STOP';

            % Create ElapsedtimeEditFieldLabel
            app.ElapsedtimeEditFieldLabel = uilabel(app.LeftPanel);
            app.ElapsedtimeEditFieldLabel.HorizontalAlignment = 'right';
            app.ElapsedtimeEditFieldLabel.Position = [39 219 75 22];
            app.ElapsedtimeEditFieldLabel.Text = 'Elapsed time';
            % Create ElapsedtimeEditField
            app.ElapsedtimeEditField = uieditfield(app.LeftPanel, 'text');
            app.ElapsedtimeEditField.Position = [141 219 60 22];

            % Create RightPanel
            app.RightPanel = uipanel(app.GridLayout);
            app.RightPanel.Layout.Row = 1;
            app.RightPanel.Layout.Column = 2;

            % Create TabGroup
            app.TabGroup = uitabgroup(app.RightPanel);
            app.TabGroup.Position = [7 6 828 643];

            % Create CalibrationTab
            app.CalibrationTab = uitab(app.TabGroup);
            app.CalibrationTab.Title = 'Calibration';

            % Create CalibrationPanel
            app.CalibrationPanel = uipanel(app.CalibrationTab);
            app.CalibrationPanel.Tooltip = {'Provide a preliminary calibration of
the target compouns(s). Either call the calibration tool or select the appropriate
calibration file'};
            app.CalibrationPanel.Title = 'Calibration';
            app.CalibrationPanel.Position = [576 295 222 196];

            % Create StartnewcalibrationButton
            app.StartnewcalibrationButton = uibutton(app.CalibrationPanel, 'push');
            app.StartnewcalibrationButton.ButtonPushedFcn = createCallbackFcn(app,
@StartnewcalibrationButtonPushed, true);
            app.StartnewcalibrationButton.Position = [52 118 118 42];
            app.StartnewcalibrationButton.Text = {'Start'; 'new calibration'};

            % Create CalibrationacquiredLampLabel
            app.CalibrationacquiredLampLabel = uilabel(app.CalibrationPanel);
            app.CalibrationacquiredLampLabel.Position = [21 13 64 27];
            app.CalibrationacquiredLampLabel.Text = {'Calibration'; 'acquired'};

            % Create CalibrationacquiredLamp
            app.CalibrationacquiredLamp = uilamp(app.CalibrationPanel);
            app.CalibrationacquiredLamp.Position = [179 15 23 23];
            app.CalibrationacquiredLamp.Color = [0.8 0.8 0.8];

            % Create ChoosecalibrationfileButton
            app.ChoosecalibrationfileButton = uibutton(app.CalibrationPanel,
'push');
            app.ChoosecalibrationfileButton.ButtonPushedFcn =
createCallbackFcn(app, @ChoosecalibrationfileButtonPushed, true);
            app.ChoosecalibrationfileButton.Position = [52 62 118 39];
            app.ChoosecalibrationfileButton.Text = {'Choose'; 'calibration file'};

            % Create UIAxesCal
            app.UIAxesCal = uiaxes(app.CalibrationTab);
            title(app.UIAxesCal, 'Calibration graph')
            xlabel(app.UIAxesCal, 'Target compound concentration [ppb vol]')
            ylabel(app.UIAxesCal, 'Intensity [ions/s]')
            app.UIAxesCal.PlotBoxAspectRatio = [1 1.04312114989733 1];
            app.UIAxesCal.Position = [15 16 536 564];

            % Create CalibrationparametersPanel
            app.CalibrationparametersPanel = uipanel(app.CalibrationTab);
            app.CalibrationparametersPanel.Title = 'Calibration parameters';
            app.CalibrationparametersPanel.Position = [567 177 240 100];

            % Create LOG10INTENSITYLabel
            app.LOG10INTENSITYLabel = uilabel(app.CalibrationparametersPanel);
            app.LOG10INTENSITYLabel.HorizontalAlignment = 'right';
            app.LOG10INTENSITYLabel.Position = [2 45 124 22];
            app.LOG10INTENSITYLabel.Text = 'LOG10 (INTENSITY) =';

            % Create SlopeEditField
            app.SlopeEditField = uieditfield(app.CalibrationparametersPanel,
'numeric');
            app.SlopeEditField.HorizontalAlignment = 'center';
            app.SlopeEditField.Position = [6 21 59 22];

            % Create xLOG10CONCLabel
            app.xLOG10CONCLabel = uilabel(app.CalibrationparametersPanel);
            app.xLOG10CONCLabel.HorizontalAlignment = 'right';
            app.xLOG10CONCLabel.Position = [63 22 112 22];
            app.xLOG10CONCLabel.Text = 'x LOG10 (CONC) + ';

            % Create InterceptEditField
            app.InterceptEditField = uieditfield(app.CalibrationparametersPanel,
'numeric');
            app.InterceptEditField.HorizontalAlignment = 'center';
            app.InterceptEditField.Position = [175 21 60 22];

            % Create SimulationTab
            app.SimulationTab = uitab(app.TabGroup);
            app.SimulationTab.Title = 'Simulation';

            % Create Panel2_3
            app.Panel2_3 = uipanel(app.SimulationTab);
            app.Panel2_3.AutoResizeChildren = 'off';
            app.Panel2_3.Title = 'Simulation parameters';
            app.Panel2_3.Position = [593 341 200 94];

            % Create MockfiletimespanEditFieldLabel
            app.MockfiletimespanEditFieldLabel = uilabel(app.Panel2_3);
            app.MockfiletimespanEditFieldLabel.HorizontalAlignment = 'right';
            app.MockfiletimespanEditFieldLabel.Position = [2 41 103 22];
            app.MockfiletimespanEditFieldLabel.Text = 'Mockfile timespan';

            % Create MockfiletimespanEditField
            app.MockfiletimespanEditField = uieditfield(app.Panel2_3, 'numeric');
            app.MockfiletimespanEditField.Position = [117 41 46 22];

            % Create sLabel
            app.sLabel = uilabel(app.Panel2_3);
            app.sLabel.Position = [167 41 25 22];
            app.sLabel.Text = 's';

            % Create SimulatedmassEditFieldLabel
            app.SimulatedmassEditFieldLabel = uilabel(app.Panel2_3);
            app.SimulatedmassEditFieldLabel.HorizontalAlignment = 'right';
            app.SimulatedmassEditFieldLabel.Position = [1 12 91 22];
            app.SimulatedmassEditFieldLabel.Text = 'Simulated mass';

            % Create SimulatedmassEditField
            app.SimulatedmassEditField = uieditfield(app.Panel2_3, 'numeric');
            app.SimulatedmassEditField.Position = [118 12 46 22];

            % Create mqLabel
            app.mqLabel = uilabel(app.Panel2_3);
            app.mqLabel.Position = [168 12 27 22];
            app.mqLabel.Text = 'm/q';

            % Create Panel2_5
            app.Panel2_5 = uipanel(app.SimulationTab);
            app.Panel2_5.AutoResizeChildren = 'off';
            app.Panel2_5.Title = 'Simulation state';
            app.Panel2_5.Position = [592 235 200 71];

            % Create MockfilegenerationLampLabel
            app.MockfilegenerationLampLabel = uilabel(app.Panel2_5);
            app.MockfilegenerationLampLabel.HorizontalAlignment = 'right';
            app.MockfilegenerationLampLabel.Position = [2 17 111 22];
            app.MockfilegenerationLampLabel.Text = 'Mockfile generation';

            % Create MockfilegenerationLamp
            app.MockfilegenerationLamp = uilamp(app.Panel2_5);
            app.MockfilegenerationLamp.Position = [169 18 20 20];
            app.MockfilegenerationLamp.Color = [0.8 0.8 0.8];

            % Create UIAxesSim
            app.UIAxesSim = uiaxes(app.SimulationTab);
            title(app.UIAxesSim, 'Last randomly-generated datachunk')
            xlabel(app.UIAxesSim, 'Time [s]')
            ylabel(app.UIAxesSim, 'Intensity [ions]')
            app.UIAxesSim.PlotBoxAspectRatio = [1 1.04857444561774 1];
            app.UIAxesSim.XTick = [0 0.2 0.4 0.6 0.8 1];
            app.UIAxesSim.Position = [41 28 524 552];

            % Create MonitoringTab
            app.MonitoringTab = uitab(app.TabGroup);
            app.MonitoringTab.Title = 'Monitoring';

            % Create UIAxesMon1
            app.UIAxesMon1 = uiaxes(app.MonitoringTab);
            title(app.UIAxesMon1, 'Raw headspace ion count')
            xlabel(app.UIAxesMon1, 'Time [min]')
            ylabel(app.UIAxesMon1, 'Intensity [ions]')
            app.UIAxesMon1.PlotBoxAspectRatio = [3.03846153846154 1 1];
            app.UIAxesMon1.Position = [29 313 760 290];

            % Create UIAxesMon2
            app.UIAxesMon2 = uiaxes(app.MonitoringTab);
            title(app.UIAxesMon2, '2 min rolling average liquid concentration')
            xlabel(app.UIAxesMon2, 'Time [min]')
            ylabel(app.UIAxesMon2, 'Concentration [mg/L]')
            app.UIAxesMon2.PlotBoxAspectRatio = [3.03846153846154 1 1];
            app.UIAxesMon2.Position = [29 19 760 290];

            % Create HistoryTab
            app.HistoryTab = uitab(app.TabGroup);
            app.HistoryTab.Title = 'History';

            % Create WholeHistory
            app.WholeHistory = uiaxes(app.HistoryTab);
            title(app.WholeHistory, 'Entire acquisition')
            xlabel(app.WholeHistory, 'Time [h]')
            ylabel(app.WholeHistory, 'Concentration [mg/L]')
            app.WholeHistory.PlotBoxAspectRatio = [2.69444444444444 1 1];
            app.WholeHistory.Position = [39 330 590 272];

            % Create SlidingHistory
            app.SlidingHistory = uiaxes(app.HistoryTab);
            title(app.SlidingHistory, 'Zoomed graph')
            xlabel(app.SlidingHistory, 'Time [min]')
            ylabel(app.SlidingHistory, 'Concentration [mg/L]')
            app.SlidingHistory.PlotBoxAspectRatio = [2.59821428571429 1 1];
            app.SlidingHistory.Position = [39 42 590 280];

            % Create TimeSlider
            app.TimeSlider = uislider(app.HistoryTab);
            app.TimeSlider.Limits = [0 1];
            app.TimeSlider.MajorTicks = [];
            app.TimeSlider.ValueChangingFcn = createCallbackFcn(app,
@TimeSliderValueChanging, true);
            app.TimeSlider.MinorTicks = [];
            app.TimeSlider.Enable = 'off';
            app.TimeSlider.Position = [126 19 457 3];

            % Create ImportoptionsPanel
            app.ImportoptionsPanel = uipanel(app.HistoryTab);
            app.ImportoptionsPanel.Title = 'Import options';
            app.ImportoptionsPanel.Position = [633 374 183 198];

            % Create ChoosefolderfileButton
            app.ChoosefolderfileButton = uibutton(app.ImportoptionsPanel, 'push');
            app.ChoosefolderfileButton.ButtonPushedFcn = createCallbackFcn(app,
@ChoosefolderfileButtonPushed, true);
            app.ChoosefolderfileButton.Position = [14 81 109 36];
            app.ChoosefolderfileButton.Text = {'Choose'; 'folder/file'};

            % Create ImportexperimentButton
            app.ImportexperimentButton = uibutton(app.ImportoptionsPanel, 'push');
            app.ImportexperimentButton.ButtonPushedFcn = createCallbackFcn(app,
@ImportexperimentButtonPushed, true);
            app.ImportexperimentButton.Position = [14 46 152 23];
            app.ImportexperimentButton.Text = 'Import experiment';

            % Create HistoryDataLamp
            app.HistoryDataLamp = uilamp(app.ImportoptionsPanel);
            app.HistoryDataLamp.Position = [140 89 20 20];
            app.HistoryDataLamp.Color = [0.8 0.8 0.8];

            % Create ImportGCMScontrolsButton
            app.ImportGCMScontrolsButton = uibutton(app.ImportoptionsPanel,
'push');
            app.ImportGCMScontrolsButton.ButtonPushedFcn = createCallbackFcn(app,
@ImportGCMScontrolsButtonPushed, true);
            app.ImportGCMScontrolsButton.Position = [14 13 152 22];
            app.ImportGCMScontrolsButton.Text = 'Import GCMS controls';

            % Create ChoosetoimportDropDownLabel
            app.ChoosetoimportDropDownLabel = uilabel(app.ImportoptionsPanel);
            app.ChoosetoimportDropDownLabel.Position = [14 149 102 23];
            app.ChoosetoimportDropDownLabel.Text = 'Choose to import:';

            % Create ChoosetoimportDropDown
            app.ChoosetoimportDropDown = uidropdown(app.ImportoptionsPanel);
            app.ChoosetoimportDropDown.Items = {'Raw data', 'Pre-processed data'};
            app.ChoosetoimportDropDown.ValueChangedFcn = createCallbackFcn(app,
@ChoosetoimportDropDownValueChanged, true);
            app.ChoosetoimportDropDown.Position = [13 129 153 22];
            app.ChoosetoimportDropDown.Value = 'Raw data';

            % Create GraphoptionsPanel
            app.GraphoptionsPanel = uipanel(app.HistoryTab);
            app.GraphoptionsPanel.Title = 'Graph options';
            app.GraphoptionsPanel.Position = [633 144 183 85];

            % Create PlottedtimeintervalminSpinnerLabel
            app.PlottedtimeintervalminSpinnerLabel =
uilabel(app.GraphoptionsPanel);
            app.PlottedtimeintervalminSpinnerLabel.Position = [10 36 142 23];
            app.PlottedtimeintervalminSpinnerLabel.Text = 'Plotted time interval
[min]';

            % Create PlottedtimeintervalminSpinner
            app.PlottedtimeintervalminSpinner = uispinner(app.GraphoptionsPanel);
            app.PlottedtimeintervalminSpinner.Position = [10 12 140 22];
            app.PlottedtimeintervalminSpinner.Value = 100;

            % Show the figure after all components are created
            app.BioControl.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = platform_test_test

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.BioControl)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.BioControl)
        end
    end
end
```

Calibration interface

```
classdef cal < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        BioControl10CalibrationtabUIFigure  matlab.ui.Figure
        UIAxes                             matlab.ui.control.UIAxes
        PaircalibrationfileButton          matlab.ui.control.Button
        CompoundconcentrationppbLabel      matlab.ui.control.Label
        ComponentConcentrationEditField    matlab.ui.control.NumericEditField
        CleargraphButton                   matlab.ui.control.Button
        SlopeEditFieldLabel                matlab.ui.control.Label
        SlopeEditField                     matlab.ui.control.NumericEditField
        LinearfitButton                    matlab.ui.control.Button
        StorecalibrationButton             matlab.ui.control.Button
        InterceptEditFieldLabel            matlab.ui.control.Label
        InterceptEditField                 matlab.ui.control.NumericEditField
        R2EditFieldLabel                   matlab.ui.control.Label
        R2EditField                        matlab.ui.control.NumericEditField
        GeneratemockcalibrationButton      matlab.ui.control.Button
        CompoundcentralmassDaLabel         matlab.ui.control.Label
        ComponentCentralMassEditField      matlab.ui.control.NumericEditField
        CompoundmassneighbourhoodDaLabel   matlab.ui.control.Label
        ComponentMassNeighbourhoodEditField  matlab.ui.control.NumericEditField
    end


    properties (Access = private)

        TempFolderPath % Path of the temporary folder where mock
                       % calibrations points are stored
        NoPointsOnGraph % Number of points already plotted on the graph
        DataToFit % Array containing concentration&intensity values for
                  % each point added to the calibration
        FitParameters % Structure storing the last fitted parameters
    end

    methods (Access = private)

        function DeleteH5Content(app,~,~)
            filePattern = fullfile( ...
                app.TempFolderPath.Value, 'PTRMSmockcalsequence*.h5');
            files = dir(filePattern);
            for k = 1:length(files)
              baseFileName = files(k).name;
              fullFileName = fullfile( ...
                  app.TempFolderPath.Value, baseFileName);
              delete(fullFileName);
            end
        end

    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            format long
            hold(app.UIAxes, 'on')

            app.TempFolderPath.Value = '';
            app.NoPointsOnGraph.Value = 0;
            app.DataToFit.Value = [];
            app.FitParameters.Value = [];
        end

        % Button pushed function: PaircalibrationfileButton
        function PaircalibrationfileButtonPushed(app, event)
            check = isempty(app.ComponentConcentrationEditField.Value);
            if check
                uialert(app.BioControl10CalibrationtabUIFigure, ...
                ['Insert a concentration value before proceding ' ...
                'to pair up file and concentration.'], ...
                'BioControl 1.0', 'Icon', 'error');

                return
            end

            check = app.ComponentCentralMassEditField.Value == 0;
            if check
                uialert(app.BioControl10CalibrationtabUIFigure, ...
                ['Insert target mass value before proceding ' ...
                'to pair up file and concentration.'], ...
                'BioControl 1.0', 'Icon', 'error');

                return
            end
            check = app.ComponentMassNeighbourhoodEditField.Value == 0;
            if check
                uialert(app.BioControl10CalibrationtabUIFigure, ...
                ['Insert a mass neighbourhood value before proceding ' ...
                'to pair up file and concentration.'], ...
                'BioControl 1.0', 'Icon', 'error');
                
                return
            end
            
            filter = '.h5';
            title = 'BioControl 1.0 - Select one or more calibration files';
            [filename,path] = uigetfile(filter, title,'MultiSelect','on');
            
            multiple_entrance_check = iscell(filename);
            if ~multiple_entrance_check
                single_entrance_check = ischar(filename);
                if ~single_entrance_check
                    return
                end
            end
            
            if multiple_entrance_check
                ext_check = false;
                for i = 1:length(filename)
                    [~,~,extension] = fileparts(filename{i});
                    ext_check = ext_check | ~strcmpi(extension, filter);
                    if ext_check
                        uialert(app.BioControl10CalibrationtabUIFigure, ...
                        ['The selected file doesn''t have the ' ...
                        'expected extension (<.h5>).'], ...
                        'BioControl 1.0', 'Icon', 'error');
                        
                        return
                    end
                end
            else
                [~,~,extension] = fileparts(filename);
                ext_check = ~strcmpi(extension, filter);
                if ext_check
                    uialert(app.BioControl10CalibrationtabUIFigure, ...
                    ['The selected file doesn''t have the ' ...
                    'expected extension (<.h5>).'], ...
                    'BioControl 1.0', 'Icon', 'error');
                    
                    return
                end
            end
            
            
            cd(path);
            if multiple_entrance_check
                pruned_data = [];
                for i = 1:length(filename)
                    [cumpeakprof,~,~] = geth5mrcumpeaks( ...
                        filename{i}, ...
                        (app.ComponentCentralMassEditField.Value + 1), ...
                        app.ComponentMassNeighbourhoodEditField.Value);
                    
                    pruned_data = [pruned_data rmoutliers(cumpeakprof)];
                end
            else
                [cumpeakprof,~,~] = geth5mrcumpeaks( ...
                    filename, ...
                    (app.ComponentCentralMassEditField.Value + 1), ...
                    app.ComponentMassNeighbourhoodEditField.Value);

                pruned_data = rmoutliers(cumpeakprof);
            end

            avg_output = mean(pruned_data);
            error = std(pruned_data);


            errorbar(app.UIAxes, ...
                app.ComponentConcentrationEditField.Value, ...
                avg_output, ...
                error, ...
                'Marker', 'o', ...
                'MarkerFaceColor', 'b');

            set(app.UIAxes,'YScale','log')
            set(app.UIAxes,'XScale','log')
            grid(app.UIAxes, 'on')

            app.NoPointsOnGraph.Value = app.NoPointsOnGraph.Value + 1;

            new_point = ...
                [app.ComponentConcentrationEditField.Value; ...
                avg_output; ...
                error];
            app.DataToFit.Value = [app.DataToFit.Value new_point];
        end

        % Button pushed function: CleargraphButton
        function CleargraphButtonPushed(app, event)
            aspect_ratio = app.UIAxes.PlotBoxAspectRatio;
            position = app.UIAxes.Position;

            cla(app.UIAxes,'reset');

            % Create UIAxes
            app.UIAxes = uiaxes(app.BioControl10CalibrationtabUIFigure);
            title(app.UIAxes, 'Calibration data')
            xlabel(app.UIAxes, 'Component concentration [ppb vol]')
            ylabel(app.UIAxes, 'Average intensity [ions/s] ')
            app.UIAxes.PlotBoxAspectRatio = aspect_ratio;
            app.UIAxes.Position = position;

            hold(app.UIAxes, 'on')

            app.NoPointsOnGraph.Value = 0;
            app.DataToFit.Value = [];
            app.FitParameters.Value = [];
        end

        % Button pushed function: LinearfitButton
        function LinearfitButtonPushed(app, event)
            check = app.NoPointsOnGraph.Value >= 2;

            if ~check
                uialert(app.BioControl10CalibrationtabUIFigure, ...
                ['Can''t fit less than 2 experimental points. ' ...
                'Add more points to the graph.'], ...
                'BioControl 1.0', 'Icon', 'error');

                return
            end


            X = app.DataToFit.Value(1,:);
            Y = app.DataToFit.Value(2,:);
            logx = log10(X);
            logy = log10(Y);


            [app.FitParameters.Value,S] = polyfit(logx,logy,1);
            R2 = 1 - (S.normr^2)/(norm(logy-mean(logy))^2);


            app.R2EditField.Value = R2;


            app.SlopeEditField.Value = app.FitParameters.Value(1);


            app.InterceptEditField.Value = app.FitParameters.Value(2);




            A = [min(logx) max(logx)];


            B = polyval(app.FitParameters.Value,A);


            loglog(app.UIAxes,10.^A,10.^B,'Color','r','LineWidth',1)
        end

        % Button pushed function: StorecalibrationButton
        function StorecalibrationButtonPushed(app, event)
            check = isempty(app.DataToFit.Value);

            if check
                uialert(app.BioControl10CalibrationtabUIFigure, ...
                ['No data points available for storing. ' ...
                'Plot data points before proceeding.'], ...
                'BioControl 1.0', 'Icon', 'error');

                return
            end

            time = datetime('now','format','yyyy.MM.dd-HH.mm.ss');
            temp = datestr(time,'yyyy.mm.dd-HH.MM.ss');
            name = ['calibration-' temp '.xlsx'];


            filter = {'*.xlsx';'*.csv'};
            [file,path] = uiputfile(filter,'BioControl 1.0',name);

            check = file == 0;
            if check
                return
            end

            if strcmpi(app.TempFolderPath.Value, path(1:(end-1)))
                path = path(1:(end-5));
            end

            name = fullfile(path, file);

            prompt = 'Comments:';
            dlgtitle = 'BioControl 1.0';
            dims = [7 70];
            comments = inputdlg(prompt,dlgtitle,dims);
            if ~isempty(char(comments))
                textlog{1} = 'Comments:';
                textlog{end + 1} = char(comments);
                textlog{end + 1} = '';
            else
                textlog{1} = 'Comments:';
                textlog{end + 1} = 'none';
                textlog{end + 1} = '';
            end
            writecell(textlog, name)

            if isempty(app.FitParameters.Value)
                % Stores just data points
                datalog = {'Date','Time','';
                datestr(datetime('now','format','dd/MM/yyyy'),'dd/mm/yyyy'),...
                datestr(datetime('now','format','HH:mm:ss'),'HH:MM:ss'),'';
                    'LINEAR FIT ON LOG10 VALUES','','';
                    'Slope','Intercept','R2';
                    '','','';
                    'Concentration [ppm vol]','Intensity [ions/s]','STD Error []'};
            else
                % Stores data points and calibration data
                datalog = {'Date','Time','';
                datestr(datetime('now','format','dd/MM/yyyy'),'dd/mm/yyyy'),...
                datestr(datetime('now','format','HH:mm:ss'),'HH:MM:ss'),'';
                    'LINEAR FIT ON LOG10 VALUES','','';
                    'Slope','Intercept','R2';

app.FitParameters.Value(1),app.FitParameters.Value(2),app.R2EditField.Value;
                    'Concentration [ppb vol]','Intensity [ions/s]','STD Error []'};
            end

            writecell(datalog, name, 'Range', 'A2:C7')

            co = (app.DataToFit.Value(1,:));
            av = (app.DataToFit.Value(2,:));
            er = (app.DataToFit.Value(3,:));
            data = [co' av' er'];

            textpoints = 'datalog_points = {';
            for i = 1:length(co)
                textpoints = [textpoints 'data(' num2str(i) ',1),data(' num2str(i)
',2),data(' num2str(i) ',3)'];
                if i ~= length(co)
                    textpoints = [textpoints ';'];
                end
            end
            textpoints = [textpoints '};'];

            eval(textpoints);

            writecell(datalog_points, name, 'WriteMode', 'append')
        end

        % Close request function: BioControl10CalibrationtabUIFigure
        function BioControl10CalibrationtabUIFigureCloseRequest(app, event)
            if ~isempty(app.TempFolderPath.Value)
                if 7 == exist(app.TempFolderPath.Value, 'dir')
                    cd(app.TempFolderPath.Value)
                    DeleteH5Content(app)
                    if isempty(ls)
                        cd ..
                        rmdir(app.TempFolderPath.Value)
                    end
                end
            end

            clear parameters
            global parameters
            parameters = app.FitParameters.Value;
            delete(app);
            fclose('all');
        end

        % Button pushed function: GeneratemockcalibrationButton
        function GeneratemockcalibrationButtonPushed(app, event)
            prompt = 'Mass [Da]';
            dlgtitle = 'Enter numerical value of mocked mass';
            dims = [1 100];
            output = inputdlg(prompt,dlgtitle,dims);

            check = isempty(output);
            if check
                return
            end
            %---------------------------
            CleargraphButtonPushed(app, event)
            %---------------------------
            if isempty(app.TempFolderPath.Value)
                mkdir('Temp')
                cd('Temp')
                app.TempFolderPath.Value = cd;
            else
                cd(app.TempFolderPath.Value)
                DeleteFolderContent(app)
            end
            %----------------------------
            user_mass = str2double(output{1});
            timelength_mock = 60;
            signal_intensity = 25;
            conc = 0;

            for i = 1:5
                filename = genaddh5calmock( ...
                    user_mass,timelength_mock,signal_intensity);

                [cumpeakprof,~,~] = geth5mrcumpeaks( ...
                    filename, ...
                    (user_mass + 1), ...
                    0.3);

                avg_output = mean(cumpeakprof);
                error = std(cumpeakprof);

                errorbar(app.UIAxes, ...
                conc, ...
                avg_output, ...
                error, ...
                'Marker', 'o', ...
                'MarkerFaceColor', 'b');

                new_point = [conc; avg_output; error];
                app.DataToFit.Value = [app.DataToFit.Value new_point];

                signal_intensity = signal_intensity + 500;
                conc = conc + 10;
            end
            %----------------------------
            app.NoPointsOnGraph.Value = app.NoPointsOnGraph.Value + 5;

            LinearfitButtonPushed(app, event)

        end
    end
    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create BioControl10CalibrationtabUIFigure and hide until all
components are created
            app.BioControl10CalibrationtabUIFigure = uifigure('Visible', 'off');
            app.BioControl10CalibrationtabUIFigure.AutoResizeChildren = 'off';
            app.BioControl10CalibrationtabUIFigure.Position = [330 130 700 480];
            app.BioControl10CalibrationtabUIFigure.Name = 'BioControl 1.0 -
Calibration tab';
            app.BioControl10CalibrationtabUIFigure.Resize = 'off';
            app.BioControl10CalibrationtabUIFigure.CloseRequestFcn =
createCallbackFcn(app, @BioControl10CalibrationtabUIFigureCloseRequest, true);

            % Create UIAxes
            app.UIAxes = uiaxes(app.BioControl10CalibrationtabUIFigure);
            title(app.UIAxes, 'Calibration data')
            xlabel(app.UIAxes, 'Target compound concentration [ppb vol]')
            ylabel(app.UIAxes, 'Average intensity [ions/s] ')
            app.UIAxes.PlotBoxAspectRatio = [1.24148606811146 1 1];
            app.UIAxes.Position = [13 79 449 378];

            % Create PaircalibrationfileButton
            app.PaircalibrationfileButton =
uibutton(app.BioControl10CalibrationtabUIFigure, 'push');
            app.PaircalibrationfileButton.ButtonPushedFcn = createCallbackFcn(app,
@PaircalibrationfileButtonPushed, true);
            app.PaircalibrationfileButton.Position = [531 244 100 36];
            app.PaircalibrationfileButton.Text = {'Pair'; 'calibration file'};

            % Create CompoundconcentrationppbLabel
            app.CompoundconcentrationppbLabel =
uilabel(app.BioControl10CalibrationtabUIFigure);
            app.CompoundconcentrationppbLabel.Position = [486 402 110 28];
            app.CompoundconcentrationppbLabel.Text = {'Compound'; 'concentration
[ppb]'};

            % Create ComponentConcentrationEditField
            app.ComponentConcentrationEditField =
uieditfield(app.BioControl10CalibrationtabUIFigure, 'numeric');
            app.ComponentConcentrationEditField.Position = [613 402 61 28];

            % Create CleargraphButton
            app.CleargraphButton = uibutton(app.BioControl10CalibrationtabUIFigure,
'push');
            app.CleargraphButton.ButtonPushedFcn = createCallbackFcn(app,
@CleargraphButtonPushed, true);
            app.CleargraphButton.Position = [531 131 100 22];
            app.CleargraphButton.Text = 'Clear graph';

            % Create SlopeEditFieldLabel
            app.SlopeEditFieldLabel =
uilabel(app.BioControl10CalibrationtabUIFigure);
            app.SlopeEditFieldLabel.HorizontalAlignment = 'right';
            app.SlopeEditFieldLabel.Position = [171 29 36 22];
            app.SlopeEditFieldLabel.Text = 'Slope';

            % Create SlopeEditField
            app.SlopeEditField =
uieditfield(app.BioControl10CalibrationtabUIFigure, 'numeric');
            app.SlopeEditField.Editable = 'off';
            app.SlopeEditField.Position = [214 29 74 22];
            % Create LinearfitButton
            app.LinearfitButton = uibutton(app.BioControl10CalibrationtabUIFigure,
'push');
            app.LinearfitButton.ButtonPushedFcn = createCallbackFcn(app,
@LinearfitButtonPushed, true);
            app.LinearfitButton.Position = [44 29 100 22];
            app.LinearfitButton.Text = 'Linear fit';

            % Create StorecalibrationButton
            app.StorecalibrationButton =
uibutton(app.BioControl10CalibrationtabUIFigure, 'push');
            app.StorecalibrationButton.ButtonPushedFcn = createCallbackFcn(app,
@StorecalibrationButtonPushed, true);
            app.StorecalibrationButton.Position = [530 104 102 22];
            app.StorecalibrationButton.Text = 'Store calibration';

            % Create InterceptEditFieldLabel
            app.InterceptEditFieldLabel =
uilabel(app.BioControl10CalibrationtabUIFigure);
            app.InterceptEditFieldLabel.HorizontalAlignment = 'right';
            app.InterceptEditFieldLabel.Position = [321 29 52 22];
            app.InterceptEditFieldLabel.Text = 'Intercept';

            % Create InterceptEditField
            app.InterceptEditField =
uieditfield(app.BioControl10CalibrationtabUIFigure, 'numeric');
            app.InterceptEditField.Editable = 'off';
            app.InterceptEditField.Position = [380 29 74 22];

            % Create R2EditFieldLabel
            app.R2EditFieldLabel = uilabel(app.BioControl10CalibrationtabUIFigure);
            app.R2EditFieldLabel.HorizontalAlignment = 'right';
            app.R2EditFieldLabel.Position = [499 29 25 22];
            app.R2EditFieldLabel.Text = 'R2';

            % Create R2EditField
            app.R2EditField = uieditfield(app.BioControl10CalibrationtabUIFigure,
'numeric');
            app.R2EditField.Editable = 'off';
            app.R2EditField.Position = [536 29 74 22];

            % Create GeneratemockcalibrationButton
            app.GeneratemockcalibrationButton =
uibutton(app.BioControl10CalibrationtabUIFigure, 'push');
            app.GeneratemockcalibrationButton.ButtonPushedFcn =
createCallbackFcn(app, @GeneratemockcalibrationButtonPushed, true);
            app.GeneratemockcalibrationButton.Position = [531 203 100 36];
            app.GeneratemockcalibrationButton.Text = {'Generate mock';
'calibration'};

            % Create CompoundcentralmassDaLabel
            app.CompoundcentralmassDaLabel =
uilabel(app.BioControl10CalibrationtabUIFigure);
            app.CompoundcentralmassDaLabel.Position = [488 363 108 28];
            app.CompoundcentralmassDaLabel.Text = {'Compound central'; 'mass
(Da)'};

            % Create ComponentCentralMassEditField
            app.ComponentCentralMassEditField =
uieditfield(app.BioControl10CalibrationtabUIFigure, 'numeric');
            app.ComponentCentralMassEditField.Position = [613 363 61 28];

            % Create CompoundmassneighbourhoodDaLabel
            app.CompoundmassneighbourhoodDaLabel =
uilabel(app.BioControl10CalibrationtabUIFigure);
            app.CompoundmassneighbourhoodDaLabel.Position = [488 323 112 28];
            app.CompoundmassneighbourhoodDaLabel.Text = {'Compound mass';
'neighbourhood (Da)'};

            % Create ComponentMassNeighbourhoodEditField
            app.ComponentMassNeighbourhoodEditField =
uieditfield(app.BioControl10CalibrationtabUIFigure, 'numeric');
            app.ComponentMassNeighbourhoodEditField.Position = [613 323 61 28];

            % Show the figure after all components are created
            app.BioControl10CalibrationtabUIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = cal

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.BioControl10CalibrationtabUIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.BioControl10CalibrationtabUIFigure)
        end
    end
end
```

Control simulation:

full_parallel_transfer_function_regression.m

```
% ========================================================================
% PROFILE FITTING
% Script for fitting real experimental data of microbial fermentation of
% linalool with a parallel effects model. Parallel first order transfer
% functions are employed to approximate the kinetics of terpene
% production upon induction while considering substrate consumption.
% IPTG induction is the system input from 0 to 100% (here equal to 0 to 1).
% ========================================================================

% ========================================================================
% Initialisation
% ========================================================================
close all
clear all
clc
```

```
% ==========================================================================
% Data import and preparation
% ==========================================================================
% Import data that are saved in a .mat file containing the time values of
% the timepoints ('TimePoints) and the concentration of the target compound
% to be modeled ('Concs'). Both objects are arrays arranged as 1xN, where N
% is the number of the acquired timepoints.
load (['/Users/a30754gl/Desktop/MATLAB works/' ...
    'support scripts/limonene_conc_time_profile.mat'])


% ==========================================================================
% Parameters boundaries
% ==========================================================================
% Defined over observation of the experimental productivity values
LB = [5 500 500]; % 1st is K, 2nd is Tau1, 3rd is Tau2
UB = [5 4500 4500]; % Same order

iter = 100;
regression_performances = zeros(iter,4);
KoptData = zeros(iter,4);
f = figure('visible','off');
for n = 1:iter
% ==========================================================================
% Initialise parameters guesses
% ==========================================================================
Kin = LB(1) + (UB(1) - LB(1))/2;
Tau1in = LB(2) + (UB(2) - LB(2))/2;
Tau2in = LB(3) + (UB(3) - LB(3))/2;


% ==========================================================================
% Regress parameters
% ==========================================================================
% Minimum search to minimise the objective function, given by Sum of
% Squared Errors (SSE)
X0 = [Kin Tau1in Tau2in];
FOBFun = @(pars)SSECalcFun(pars,Productivity,TimePoints);
options = optimset('MaxIter',100000, ...
    'MaxFunEvals',100000);

[regressed_pars, SSEvalue] = fminsearchbnd(FOBFun,X0, ...
    LB, UB, ...
    options);

% ==========================================================================
% Calculate fitted productivity
% ==========================================================================
K = regressed_pars(1);
Tau1 = regressed_pars(2);
Tau2 = regressed_pars(3);
DeltaU = 1;
Time = 0:1:4500;

FittedProductivity = ParallelTFStep(K,Tau1,Tau2,DeltaU,Time);


% ==========================================================================
% Data plotting
% ==========================================================================
% Top graph
if n == 100
    subplot(1,2,1)
    hold on
    plot(Time,FittedProductivity,'b')
    hold on
    scatter(TimePoints,Productivity,'r','x')

    legend('Optimal fit','Experimental points')
else
    subplot(1,2,1)
    hold on
    plot(Time,FittedProductivity,'b','HandleVisibility','off')
end

ax = gca;
ax.YLim = [-1 10];


% ===========================================================================
% Find K based on current regressed parameters
% ===========================================================================
time_index = round(Tau1);
ProdEstimate = FittedProductivity(time_index);
first_effect_contribution = 0.632; % Choosing to pick productivity value at
                                   % 1 time constant (Tau1) means that I am
                                   % at 63.2% of the contribution of the
                                   % first effect by deafult. The
                                   % contribution of the other effect must
                                   % be calculated instead
second_effect_contribution = 1 - exp((-time_index)/(Tau1 + Tau2));

syms X
eqn = ProdEstimate - ...
    (first_effect_contribution*X - second_effect_contribution*X) == 0;
Kestimate = solve(eqn,X);

KoptData(n,:) = ...
    [time_index ProdEstimate second_effect_contribution Kestimate];


% ===========================================================================
% Store performance data
% ===========================================================================
regression_performances(n,:) = [K Tau1 Tau2 SSEvalue];


% ===========================================================================
% Update parameters boundaries
% ===========================================================================
UB(1) = UB(1) + 1; % Only K boundaries need to change, as it is the only
                   % parameter which is not dependant on anything else and
                   % can become unphysical
end


% ===========================================================================
% Find best fit
% ===========================================================================
SSEvalues = regression_performances(:,4);
SSEdiffs = zeros(iter-1);
for i = 2:iter
    SSEdiffs(i-1) = abs(SSEvalues(i-1) - SSEvalues(i));
end

acceptable_indexes = find(SSEdiffs < .01*min(SSEvalues));
best_fit_index = acceptable_indexes(1);


% ===========================================================================
% Calculate best profile
% ========================================================================
Kopt = regression_performances(best_fit_index,1);
Tau1opt = regression_performances(best_fit_index,2);
Tau2opt = regression_performances(best_fit_index,3);
FittedProductivity = ParallelTFStep(K,Tau1,Tau2,DeltaU,Time);


% ========================================================================
% Adjust plots
% ========================================================================
% Bottom graph
subplot(1,2,2)
hold on
plot(Time,FittedProductivity,'b')
scatter(TimePoints,Productivity,'r','x')
xlabel('Time [min]')
ylabel('Linalool productivity [mgL^{-1}h^{-1}]')

legend('Optimal fit','Experimental points')

set(f, 'visible', 'on');

disp(regression_performances(best_fit_index,:))


% ========================================================================
% Object function definition (FOBfun)
% ================ ========================================================
% The objective function used in here is a simple Sum of Squared Errors
% (SSE)
function FOBoutput = SSECalcFun(pars,prod,t)
    K = pars(1);
    Tau1 = pars(2);
    Tau2 = pars(3);
    DeltaU = 1;

    y = ParallelTFStep(K,Tau1,Tau2,DeltaU,t);

    FOBoutput = 0;
    for i = 1:length(y)
        FOBoutput = FOBoutput + (prod(i) - y(i))^2;
    end
end


function response_profile = ...
    ParallelTFStep(k,tau1,tau2,delta_u,time)

    response_profile = delta_u*(...
        k*(exp(-time./(tau1 + tau2)) - exp(-time./tau1)));

end
```

control_tuning_sensitivity_test.m

```
% ========================================================================
% PID TUNING SENSITIVITY TEST
% PID controller is tuned in order to execute feedback control on linalool
% productivity. Biological system transfer function is derived from
% experimental data and used here to build a close loop control system.
% Control efficacy sensitivity is tested by modifying PID parameters up to
% 20% from their optimal value in a random way. 1000 scenarios are randomly
% generated.
% ==========================================================================

% ==========================================================================
% Initialisation
% ==========================================================================
close all
clear all
clc

format long
rng('shuffle');


% ==========================================================================
% Closed loop building
% ==========================================================================
% Definition of bioprocess transfer function from previously obtained
% parameters
numerator = 23;
denominator = [1219.6 1];
sys = tf(numerator,denominator);
sys.TimeUnit = 'minutes';

% PID is tuned on the basis of the bioprocess transfer function
[C_pid,info] = pidtune(sys,'PIDF',0.5);
C_pid.TimeUnit = 'minutes';

% Outer cycle defining 2 cases: step change & disturbance rejection
for c = 1:2
    if c == 1
        CL_pid = feedback(C_pid*sys,1);
        [y1,t1] = step(CL_pid);

        figure(c)
        hold on
    else
        CL_pid_dist = feedback(sys,C_pid);
        [y1,t1] = step(CL_pid_dist);

        figure(c)
        hold on
    end

    % Inner cycle to generate 1000 random combinations of the 4 parameters,
    % varied up to a maximum of 20% from thei optimal value
    pid_tunings = zeros(1001,4);
    pid_tunings(1,:) = [C_pid.Kp C_pid.Ki C_pid.Kd C_pid.Tf];
    ranges = abs(.20*pid_tunings(1,:));
    for i = 1:1000
        for n = 1:4
            eval(['random' num2str(n) ' = rand;']);
            eval([ ...
            'if random' num2str(n) ' >=.5;' ...
                 'sign' num2str(n) ' = 1;' ...
            'else;' ...
                 'sign' num2str(n) ' = -1;' ...
            'end']);
        end

        C_pid.Kp = pid_tunings(1,1) + sign1*random1*ranges(1);
        C_pid.Ki = pid_tunings(1,2) + sign2*random2*ranges(2);
        C_pid.Kd = pid_tunings(1,3) + sign3*random3*ranges(3);
        C_pid.Tf = pid_tunings(1,4) + sign4*random4*ranges(4);

        pid_tunings(n+1,:) = [C_pid.Kp C_pid.Ki C_pid.Kd C_pid.Tf];
        if c == 1
            CL_pid = feedback(C_pid*sys,1);
        else
            CL_pid = feedback(sys,C_pid);
        end

        [y,t] = step(CL_pid);
        if i ~= 1000
            plot(t,y,'r','HandleVisibility','off')
        else
            plot(t,y,'r')
        end

    end

    plot(t1,y1,'b')
    xlabel('Minutes')
    ylabel('\DeltaProductivity/\DeltaProductivity_{max}')
    legend('Random tuning parameters deviation (up to 20% from optimal)', ...
        'Optimal tuning parameters')

end
```