PFC/RR-82-16 DOE/ET-51013-46
UC-20B

Solve : A Non-Linear Least Squares Code and its application to the Optimal Placement of Torsatron Vertical Field Coils

John Aspinall

MIT Plasma Fusion Center, Cambridge Ma. USA

May 25 1982

A computational method was developed which alleviates the need for lengthy parametric scans as part of a design process. The method makes use of a least squares algorithm to find the optimal value of a parameter vector. Optimal is defined in terms of a utility function prescribed by the user. The placement of the vertical field coils of a torsatron is such a non-linear problem.

Contents

Introduction

This report describes Solve, a non-linear least-squares code for calculating the optimal positions and currents for the vertical field coils of a torsatron.

This report is intended as both a user's manual of sorts, and as a reference for the techniques used in the code. It must, therefore, blend the sublime and the ridiculous from time to time. Chapter 1 is an introduction to the problem and the method of solution. Chapter 2 descends into the details of running the code; it describes how to execute the code in its present version on the MFE "A" machine — a CDC 7600. Chapter 3 describes how to interpret the output from the code, while chapter 4 gives specific examples of its use. Chapter 5 discusses the theory behind the code operation in detail. Readers interested in the nitty-gritty of using the code should read chapters 1 through 4, putting chapter 5 aside for browsing. Readers interested in the theory of operation, should read chapter 1, skim chapters 2, 3 and 4, and read chapter 5.

I am grateful for many discussions with Dr. P.A. Politzer, and Professor L.M. Lidsky, during this work. Dr. R. Potok also gave me valuable help in the development of this technique. During some of this time, I was supported by a National Science and Engineering Research Council of Canada scholarship. This work was also supported by DOE contract DE-AC02-78ET-51013.

The Problem

A torsatron is a toroidal plasma confinement device closely related to the stellarator. The helical windings of a torsatron all carry current in the same toroidal direction; those of a stellarator are arranged in pairs, carrying equal currents in both directions around the torus. In the stellarator, this means that toroidal field coils (similar to those of a tokamak) are necessary since the helical windings provide no net toroidal field. In the torsatron there is a net toroidal field generated by the helices, however the net current in the toroidal direction also generates a vertical field. In most cases it is necessary to cancel this vertical field in the plasma region to ensure closed magnetic surfaces for plasma confinement. Usually the vertical field of the helical windings is cancelled using pairs of circular coils, placed symmetrically above and below the helices. These will be referred to as "vertical field coils", keeping in mind that they contribute to the net radial field as well as the net vertical field of the torsatron.

There are a number of criteria which may be easily applied to evaluate the "goodness" of a given arrangement of the vertical field coils. From a plasma engineering point of view, a simple requirement is that there exist closed flux surfaces in the plasma region. (This is, of course, only a necessary and not a sufficient condition for confinement.) From a mechanical engineering point of view, it is desirable to reduce the forces on all the windings as much as possible. It is also useful to ensure that the physical arrangement of the vertical field coils allows good access to the helical windings for maintenance, diagnostics, neutral beam ports and so on. These criteria each lead individually to differing solutions to the problem of placing the vertical field coils. For example, accurate cancellation of the vertical and radial fields in the center of the plasma region will best be accomplished by a large number of coils placed close to the plasma. This contradicts sharply the requirements for easy access and maintenance. We have also found (using a complete fields and forces code) that when the vertical field coil is placed close to the helix, the distribution of forces around the helix is peaked at the point of closest approach of the helix to the other coil.

There are other conditions that one may wish the magnetic geometry to satisfy. For a torsatron confinement experiment it may be desirable that no magnetic flux link the plasma so that fluctuations in the power supply do not produce tokamak-like transformer action. For a torsatron reactor design it is probably more desirable to reduce the dipole moment of the entire system as much as possible to reduce stray fields.

A related problem that can be addressed using the same methods is that of finding the optimal arrangement of coils for a tokamak air core transformer. Here the desired situation is just the opposite of that mentioned in the previous paragraph. The transformer should provide a sufficient flux swing through the minor axis of the torus, without contributing significantly to the field in the plasma region.

A Simplified Model of the Torsatron

The purpose of Solve is to find vertical field coils to null the poloidal field contributions of the helix. We are not interested in the toroidal field of the helix at this stage, presumably it has been determined earlier in the design process. By ignoring the toroidal field of the helix, and by considering the poloidal fields from a "smoothed" version of the helix, we can obtain a much simpler model of the torsatron with which to work.

Since the circular vertical field coils have toroidal symmetry, this suggests the first step of the simplification. Let us ignore the changing angular orientation of the helix and consider a "phase-averaged" helix instead. This is a continuous distribution of currents on the surface of the torus with the local pitch angle the same as the discrete helix at all poloidal angles.

This distribution will be the equivalent of taking an ensemble of discrete helices, and averaging the current distribution over all possible starting toroidal angles in a field period. Another way to look at this is to consider it as the limiting case of taking a given winding law, and then letting N and l approach infinity, while keeping their ratio constant. The details of this smoothing process are given in Chapter 5.

Once we have this continuous surface current distribution as a function of poloidal angle, we can ignore the poloidal component of the current as explained above. We can approximate the toroidal component of the current with a set of discrete loops around the major axis. Now we can see the major simplification that has become possible. Both the known currents in the helix, and the "unknown" (i.e. to be found by the optimizer) vertical field coils are now modelled as loops of current around the major axis. This is very advantageous in terms of computational efficiency.

"Wishes", or How To Get What You Want

There is a trade-off between a set of conditions that completely models the entire physical situation, but which is cumbersome computationally, and a simplified set that uses only quantities that are computed quickly, but may ignore certain effects. Because the iterative algorithm used may call the physical model hundreds of times, we have found it more useful to pose the optimization problem in very simple form, and then go to more detailed analysis after this procedure has suggested a solution.

The conditions on the magnetic geometry mentioned in the previous section are therefore formulated in terms of various types of "wishes", as shown below.

Point Field Wishes.

- Make the radial component of the magnetic field at a given point approach a desired value.
- Make the vertical component of the magnetic field at a given point approach a desired value.
- Make the total magnetic field at a given point approach a desired value.
- Make the magnetic flux linked through a loop of given radius and height approach a desired value.

Global Field Wishes.

- Make the dipole moment of the entire system approach a desired value.
- Make the stored energy of the entire system approach a desired value.

Geometric Constraints.

- Ensure that a given vertical field coil has at least a minimum clearance from a point in rz space.
- Ensure that a given vertical field coil has at most a maximum clearance from a point in rz space.
- Ensure that a given vertical field coil has at least a minimum clearance from a line in rz space.
- Ensure that a given vertical field coil has at most a maximum clearance from a line in rz space.

Solve, then, addresses a two dimensional problem which is an approximation to the full three dimensional problem mentioned at the start of this chapter. The simplified problem is posed in terms of circular loop conductors of circular cross section around the major axis of the torsatron. Each loop can, therefore, be specified by four numbers — the major radius, the height, the minor diameter, and the current. An array of these four quantities for all the loops in the simplified problem makes up the set of parameters that specify the model.

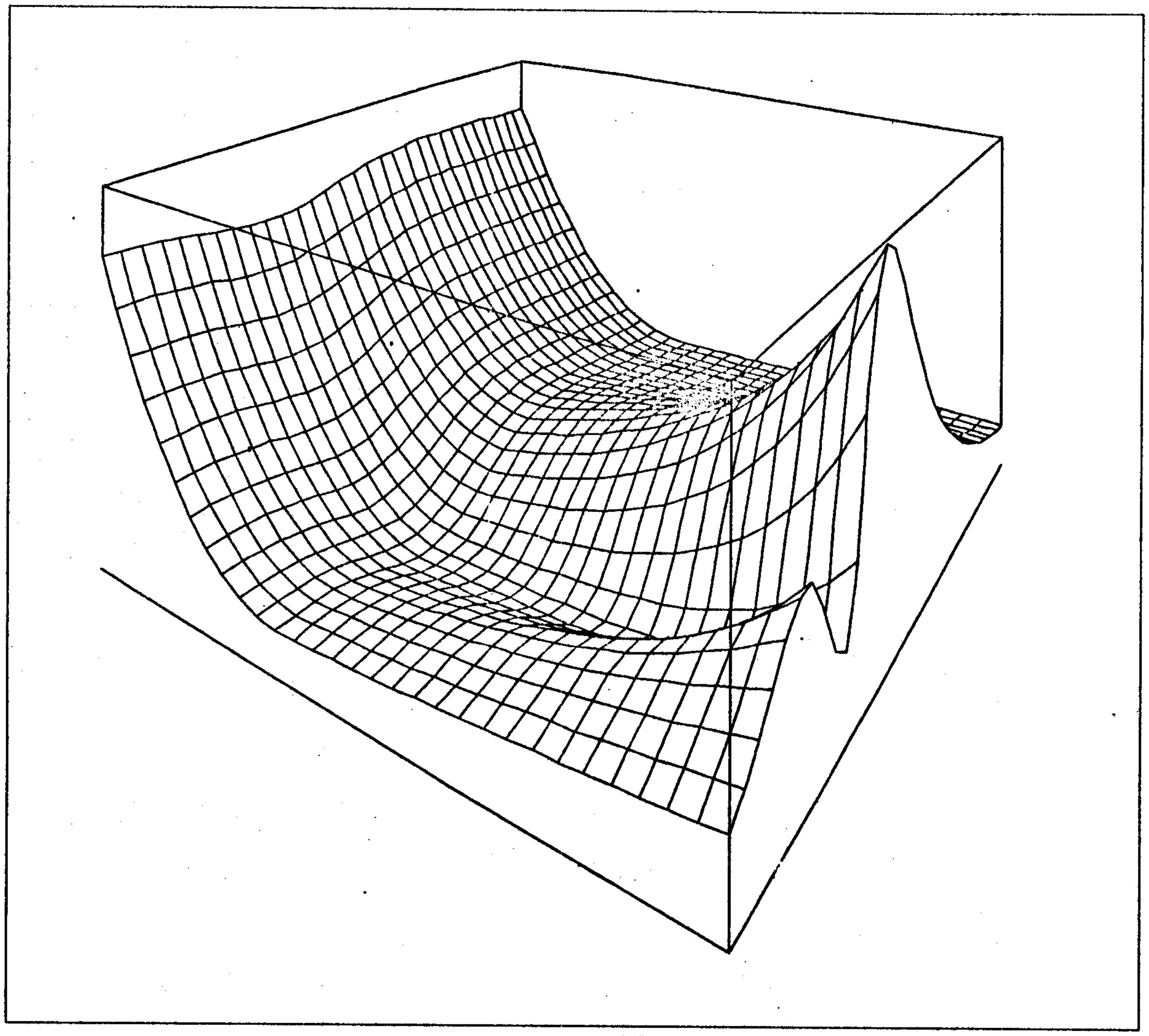

Figure 1.1 — An example of a Two Parameter Utility Function.

The plot above shows an example of how a simple set of wishes can yield a utility function with multiple minima.

A Word of Caution about the Results

All non-linear optimization algorithms suffer from one common problem. There is no way to tell whether a given local minimum that has been found is the global minimum for the domain of the problem. Algorithms that use a systematic search of the problem space may miss a solution that "falls between the cracks". Algorithms which successively estimate the position of the minimum from the gradient and higher derivatives of the utility function, spend most of their time "walking downhill", and are therefore sensitive to their starting position in parameter space.

The algorithm used in Solve (which is described in detail later) is of the second kind above. When solving all but the simplest problems with this code, it will be likely that the utility function will have more than one local minimum. There are some common symptoms of this effect, that will be described below.

Figure 1.1 shows the utility function produced from a simple set of field-nulling wishes plus a geometric constraint "repelling" a single (symmetric) pair of vertical field coils from a circular boundary. The parameter controlling the radius of the coils increases to the right and away from the viewer. The parameter controlling the height of the coils increases to the left and away from the viewer. The plot was made with the coil currents close to, but not exactly at, their ideal value. This function shows how even a simple problem may yield different solutions depending upon the iteration starting point.

If no constraints are placed on the height of a coil, the solution will sometimes place the coil at a ridiculously large distance from the torsatron helix. There is often another minimum that can be found closer to the helix, in this case. The "cookbook" cure for this is to start another run with the coils closer, carrying more current, to see if another solution can be found. (The reason for more current is to increase the change in field for a given change in position — thereby increasing the appropriate component in the gradient, which will cause the minimization algorithm to start with a smaller step.) The problem displayed in Figure 1.1 might exhibit this problem.

Sometimes, two runs of the code with wishes that differ only slightly, will finish with two completely different solutions. This may be an indication that the starting position is near a point in parameter space where the solution "trajectories" diverge — i.e. a saddle point. Again, try different starting positions to see if both minima can be reached with the same set of wishes and constraints.

All of these subjects — the simplified model, the formulation of wishes, and the algorithmic problems, are dealt with in greater detail in chapter 5. The details of specifying a model and a set of wishes in the input to the code are explained in the next chapter, and interpretation of the output is explained in chapter 3.

Input to Solve

Solve expects two files to be present in the user's active file index. These files will be referred to as the "input file" and the "log file". All the input controlling the operation of the program is read from the input file. It is a fatal error if this file is not present.

If errors in the input are detected by Solve, the code will write a file called `errors` to the active file space. This file will contain an appropriate message.

The Log File

In addition to the input file, Solve looks for a file named `solog`. This is "the log file"; its purpose is to store one number – a unique identification of the particular run of the code. If this file is not present, the code will create it and set the identification number – usually called "the run number" to 1. If the file is present in the active file index the code will read the run number, increment it by one, and write the new run number to the log file. The new run number will appear on the output to identify this particular run of the code. In addition, this number will be used to give a unique name to the default output file (see the section on the execute line).

The log file is simply an ASCII character file, so users may create their own if they want to start a series of runs with an identification number of their own choosing. The only requirement is that the file contains a decimal integer in the first 8 columns of the first row. The lowest four digits of this integer will be used. The user is free to write anything else in the rest of the file — it will be ignored. The present version of the code writes the version number of the code later on the same line using the format specification: `format (i8,8x,3a8)`.

Solve Execute Line

`solve [i=`*`input file`*`] [, o=`*`output file`*`] / t v`

Solve is executed with the above execute line. Square brackets denote optional arguments. If the input file is not specified, Solve will look for a file called `solin`. The default name for the output file is `soutNNNN` where `NNNN` is the low four digits of the log number. Examples of typical execute lines are given in here.

Example 1.

The log file `solog` does not exist. The file `myprob` contains the desired input to the code.

```
solve i=myprob
```

The log file `solog` will be created. The run number will be 1. The output file will be `sout0001`.

Example 2.

The log file `solog` is in the active file index, and it contains the log number 222 from a previous run. The file `solin` contains the desired input to the code.

```
solve
```

The log file `solog` will be changed. The run number will be 223. The output file will be `sout0223`.

Example 3.

The log file `solog` is in the active file index, and it contains the log number 222 from a previous run. The file `solin` contains the desired input to the code.

```
solve o=solvout
```

The log file `solog` will be changed. The run number will be 223. The output file will be `solvout`.

Other than the run number, all input to the code comes from the input file. This input comes in five distinct parts. The following five sections describe, in the order that they must appear in the file, those parts.

Comments

For purposes of documentation, the code allows an unlimited number of comments to be placed at the beginning of the input file. These comments will be reproduced verbatim on the first page of the output file.

Comments are read line by line until a line starting with the string "`*end-comment`" is read. The asterisk *must* be in column 1.

Specification of the Model

The purpose of this part is to describe the idealized torus, and all the other coils used in the model of the torsatron. The code deals with a set of circular coils around the z axis — so each coil is specified by four numbers — major radius, height above the $z = 0$ plane, minor radius, and the current in the loop.

The model coils fall into 3 groups. They are: (1) the variable coils — the coils that will be "moved around" by the optimization process; (2) the fixed coils which approximate the current in the helix; and (3) any other fixed coils (other than (2) above) which are not allowed to change in the optimization process, but are not part of the helix. All this data is supplied by one **Namelist** read. (For a general description of the **Namelist** feature of **LRLtran** see the **Fortlib** documentation [4].)

For the variable coils, specify `nvc`, the number of variable coils. This is the only datum referring to the variable coils that has to be specified in this part of the input — the starting positions of the coils will be specified in part 3 — the initial parameters of the optimization.

In specifying the shape of the torsatron helix, we have adopted the convention that the winding law (i.e. toroidal angle ϕ as a function of poloidal angle θ) is given by a general harmonic series of the form

$$\phi = \frac{l}{N}(\theta + \sum_{m=1} \alpha_m \sin m\theta) \tag{2.1}$$

and the torus on which the helix is wound may have a non-circular cross-section described in a similar manner

$$r = r_0(1 + \sum_{m=1} \beta_m \cos m\theta) \tag{2.2}$$

where r is the minor radius and r_0 is a "nominal" minor radius. The upper limits on the sums are implementation dependent — the current version allows a maximum of eight terms in both series.

To specify the helix requires: `nhc` the number of circular loops that will be used to approximate the helical current; `iwn` and `iwl` the N and l numbers of the torsatron; `alpha` and `orda`, the array α and its size which describe the winding law (see above for winding law used); `rmajor` and `rminor` - the major and nominal minor radii of the helix; `beta` and `ordb` the array β and its size which describe the cross-section shape of the torus; `tdip` - the minor radius (i.e. half the thickness) of the loops used to model the helix; and `cur`, the *total* toroidal current around the torus.

Specifying any non-helix constant coils requires: `ncc` the number of constant coils; and an array `ccoil` which contains (major) radius, height, thickness (minor radius), and current for each constant coil.

For commonly used cases, the user does not have to specify all these variables explicitly - there are default values and a special flag to specify a number of these variables. The default values are assigned to the variables before the **Namelist** read. If a variable does not appear in the input file, the value of the variable remains the default value. These defaults are listed below.

`nhc` 32 This is a reasonable number of coils to model the helix.

`ncc` 0 Often there are no constant coils other than the helix.

`iwn` 0 `iwn` and `iwl`, the N and l numbers of the torsatron may be specified for documentation purposes, but the values are not used by the algorithm. These defaults were deliberately chosen to be non-physical, the values will not appear anywhere in the output if they are zero, however if non-zero they will appear in the model specification.

`iwl` 0 See `iwn` above.

`orda` 0 By setting `orda` to the default value of zero, the harmonic series part of the winding law is not used and phi is linearly related to theta.

`ordb` 0 By setting `ordb` to the default value of zero, the harmonic series part of the minor radius is not used and the minor radius is a constant.

The special case flag `wflag` (a character string) has been included in the code to allow easy specification of certain other torsatron geometries. The default is the null string.

If the input sets `wflag` to the string "`conpitch`", this specifics a constant pitch angle winding law on a circular cross section torus. This also sets `orda`, `alpha`, `ordb`, and `beta` to appropriate values. The constant pitch angle winding law can be generalized with a parameter s as follows.

$$\tan\frac{\theta}{2} = s\sqrt{\frac{R+a}{R-a}}\tan\frac{N}{l}\frac{\phi}{2}$$

If $s = 1$ this is the usual constant pitch angle winding law; larger values of s make the helix steeper (more poloidal component to the current) on the outside and shallower on the inside, while smaller s does the opposite. The input variable `sconp` is s in the formula above.

Parameters of the Optimization

The model of the torsatron is a set of current-carrying loops around the z axis. Some of the loops are allowed to vary in position and current. For two reasons (discussed in detail, later) the optimization routine does not vary the parameters of the variable loops explicitly. The first reason is that the problem, posed in terms of all the positions and currents of the variable loops, may have too many degrees of freedom. This leads into the technique of applying equality constraints to the model. The second reason for not wishing to vary the physical quantities of the model directly, is that the model is written in terms of different physical units (amps, meters) which have widely varying magnitudes. The optimization routine is better behaved when its parameter space is more isotropic. This leads into the topic of rescaling the parameter space.

The mapping used is a linear mapping from the vector of dimensionless parameters that the optimizer uses, to the vector of distances and currents of the model. The mapping can be written as a matrix multiplication; Figure 2.2 shows this in two different notations. In the input file, the starting value of the dimensionless parameter vector is specified by the variable `xstart`, and the matrix by the variable `xnorm`. The first index of `xnorm` iterates over the dimensionless parameters, the second over the different coil quantities (ie 1 – R , 2 – Z , 3 – t , 4 – I), and the third over the different coils in the model.

The first point to notice in this mapping is that the dimensionless parameter vector (`xstart` in the input) has been augmented with a constant (1.0) in the $m + 1$th position. This allows the user to write a general linear equality constraint relating one or a number of model parameters to a fixed constant in a linear manner. For example, by setting two elements in a column of `xnorm` to be non-zero, we can constrain two model parameters to be related to the same dimensionless

$$\begin{pmatrix} N^1_{R_1} & N^2_{R_1} & \dots N^m_{R_1} & N^{m+1}_{R_1} \\ N^1_{Z_1} & N^2_{Z_1} & \dots N^m_{Z_1} & N^{m+1}_{Z_1} \\ N^1_{T_1} & N^2_{T_1} & \dots N^m_{T_1} & N^{m+1}_{T_1} \\ N^1_{I_1} & N^2_{I_1} & \dots N^m_{I_1} & N^{m+1}_{I_1} \\ N^1_{R_2} & N^2_{R_2} & \dots N^m_{R_2} & N^{m+1}_{R_2} \\ N^1_{Z_2} & N^2_{Z_2} & \dots N^m_{Z_2} & N^{m+1}_{Z_2} \\ N^1_{T_2} & N^2_{T_2} & \dots N^m_{T_2} & N^{m+1}_{T_2} \\ \vdots & \vdots & \vdots & \vdots \\ N^1_{Z_q} & N^2_{Z_q} & \dots N^m_{Z_q} & N^{m+1}_{Z_q} \\ N^1_{T_q} & N^2_{T_q} & \dots N^m_{T_q} & N^{m+1}_{T_q} \\ N^1_{I_q} & N^2_{I_q} & \dots N^m_{I_q} & N^{m+1}_{I_q} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ 1 \end{pmatrix} = \begin{pmatrix} R_1 \\ Z_1 \\ T_1 \\ I_1 \\ R_2 \\ Z_2 \\ T_2 \\ \vdots \\ Z_q \\ T_q \\ I_q \end{pmatrix}$$

```
      do 1 ic=1,nvc
        do 1 iq=1,4
          ddip(iq,ic) = xnorm(np+1,iq,ic)
          do 1 ip=1,np
1           ddip(iq,ic) = ddip(iq,ic) + xnorm(ip,iq,ic)*xstart(ip)
```

Figure 2.2 — The Mapping of Dimensionless Parameters Onto the Model, in Algebraic and Fortran Notation.

parameter. We could, for instance, constrain two coils to carry the same current, while allowing their positions to be independent. More examples of equality constraints appear in Chapter 4.

It is also easy to see that if an element of `xstart` is multiplied by a constant, and the corresponding column of `xnorm` is divided by the same constant, then the end result upon the model will be nil. However, when a dimensionless parameter is varied by the optimizer there will be a difference in the response of the model per unit change in the dimensionless parameter.

This leads to the idea of "rescaling" the problem such that the starting values of the dimensionless parameters are a convenient set of constants of the same magnitude, regardless of the quantity (i.e. distance or current) that the parameter controls. This is specified in the input by the variable `rescale`, (default value `yes`), and will be done for all parameters except those with zero starting value. If the user sets `rescale` to anything else, rescaling will not be done. This may be useful in the last stages of a problem when the user knows the range of parameters that is interesting.

For reading `xnorm` and `xstart`, the implicit do-loop feature of the Namelist read is very useful.

The Wishes and the Utility Function

The types of wishes that specify the utility function have already been mentioned in Chapter 1. These wishes are in physical units, so just as we had to map from dimensionless parameters onto physical quantities in the input, we have to map from physical quantities onto dimensionless residuals in the output from the model.

All wishes must specify the desired (ideal) value for the quantity considered. This is the input variable `value`. Like the input to the model, this is specified in MKS units. The contribution of the discrepancy between this desired value and the real value to the (dimensionless) residual vector is given by `weight`. Its units are the inverse of the corresponding value. The type of wish is specified by `type`.

`"type = brad"` The value specifies the desired value of the radial component of the magnetic field at a point.

`"type = bver"` The value specifies the desired value of the vertical component of the magnetic field at a point.

`"type = btot"` The value specifies the desired value of the magnitude of the magnetic field at a point.

`"type = flux"` The value specifies the desired value of the linked flux through a loop around the z axis.

In the four wishes above, the user specifies the r and z positions of the point or loop with the variable `posit` (a vector with two elements).

`"type = dipl"` The value specifies the desired value of the dipole moment of the entire model.

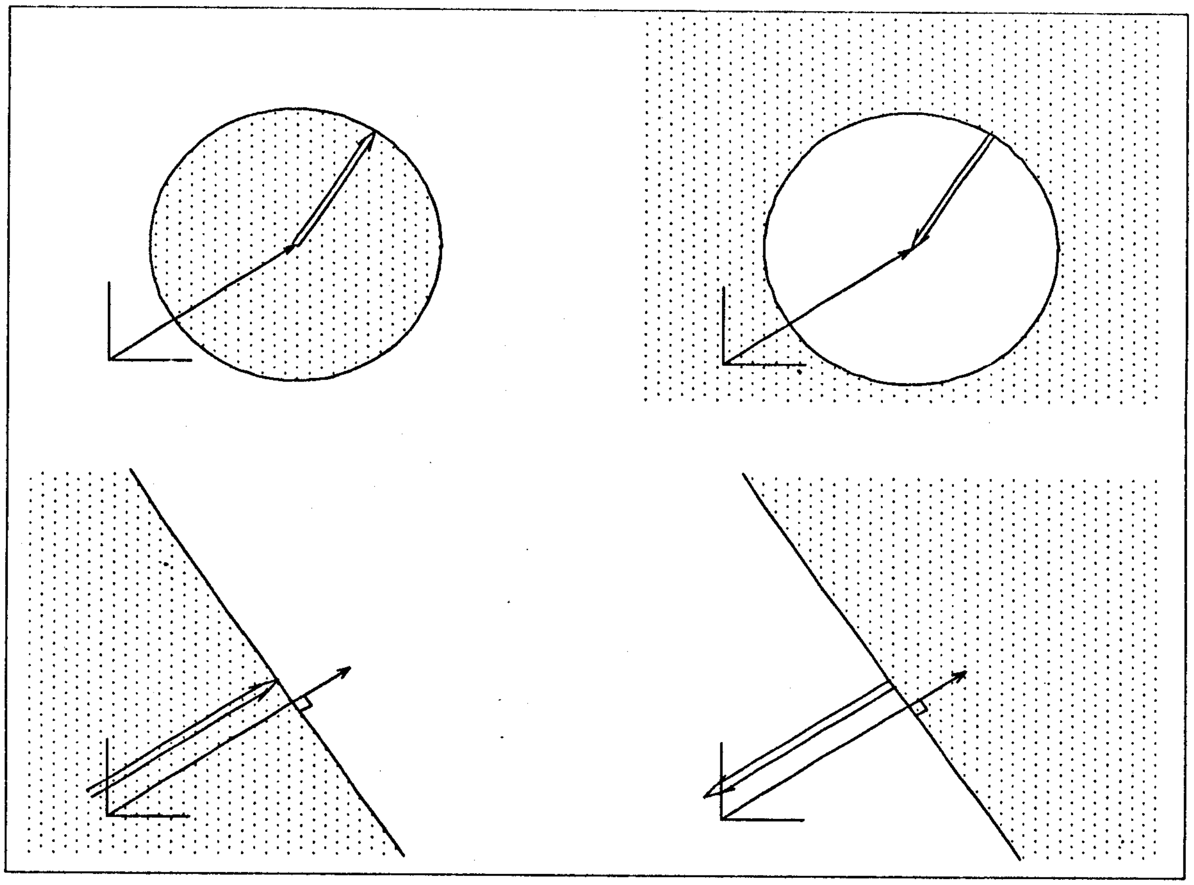

Figure 2.3 — The Two Types of Geometric Inequality Constraints.

Above : radial constraints ; below : linear constraints. Positive weight on the left. The shaded region denotes where the residual becomes non-zero. The thin arrow shows `posit`; the thick arrow, `value`.

`"type = enrg"` The value specifies the desired value of the total stored energy of the entire model.

No position information is needed for these two global wishes.

`"type = conr"` The value specifies the radius of a circular boundary around a point in *rz* space. The point is specified by `posit`, the radius by `value`. When the weight is positive, the residual is zero outside the circular boundary, and increases linearly inside the boundary by `weight` per meter. Negative weight has the opposite effect.

`"type = conl"` `posit` specifies the normal to a straight line boundary, which is a distance `value` from the origin. Positive weight "forces" the coil in the direction of the `posit` vector — that is the residual increases on the side of line opposite to the normal. As with the radial constraint, negative weight has the opposite effect.

These last two types of wishes are the only ones where negative weights are allowed. These inequality constraints are shown graphically in Figure 2.3.

Control of the Code

This last section of the input file is where the user specifies the details of the optimization operation. Most are concerned with the convergence of the code to a solution. There are three different criteria for convergence of the algorithm. The first convergence criterion is satisfied if, on two successive iterations, the dimensionless parameter estimates agree to `nsig` figures. (This has a default value of 5.) The second convergence criterion is satisfied if, on two successive iterations, the sum of the squares of the residuals have a relative difference less than or equal to `eps`. The third convergence criterion is satisfied if the Euclidean norm of the gradient vector is less than or equal to `delta`. Both `eps` and `delta` may be set to zero. The iteration terminates if any one condition is satisfied. Convergence of the algorithm is discussed in Chapter 5.

The sophisticated user may want more control over the minimization algorithm than is provided by the above variables. The input variables `iopt` and `parm` provide this. If `iopt` is not specified, it has the default value 1, and the minimizer uses default values of `parm` in a strict-descent algorithm. If `iopt = 2` the user must supply his own values of `parm`. If `iopt = 0` then another algorithm, called Brown's algorithm without strict-descent, is used. The use of these options is discussed in Chapter 5, the default action is sufficient for most cases.

The variable `maxfn` provides a way to terminate the minimization iteration if the model function is being called a large number of times. `maxfn` specifies the maximum number of function calls to the model function — the code will terminate if this number is reached. It has a default value of 1000.

All variables in this section are read with one **Namelist** read.

Glossary of Input Variables

All variables that may appear in the input file are listed here with a brief description. The word in square brackets at the end of each entry indicates in which section of the input the variable will appear. This is also the section above where more detailed documentation relating to that variable can be found.

`alpha`	Array of coefficients for the winding law harmonic series.**[model]**
`beta`	Array of coefficients for radius harmonic series.**[model]**
`ccoil`	Array of radius, height, thickness, current for constant coils.**[model]**
`coil`	Indicates to which variable coil a constraint wish applies.**[wishes]**
`cur`	Total toroidal current in the torsatron.**[model]**
`delta`	Convergence limit for the norm of the gradient.**[control]**
`eps`	Convergence limit for the sum of squares of residuals.**[control]**
`iopt`	Algorithm option for the least-squares minimizer.**[control]**
`iwl`	l : Number of poloidal periods in the torsatron field.**[model]**
`iwn`	N : Number of toroidal periods in the torsatron field.**[model]**
`maxfn`	Maximum number of function evaluations by the least-squares routine.**[control]**
`np`	Number of independent parameters in the optimization.**[parameters]**
`ncc`	Number of constant coils (other than those in the helix).**[model]**
`nhc`	Number of coils modelling the helix.**[model]**
`nsig`	Desired number of significant figures in the final parameter vector.**[control]**
`nvc`	Number of variable coils.**[model]**
`orda`	Number of terms in phi harmonic series - see `alpha`.**[model]**
`ordb`	Number of terms in radius harmonic series - see `beta`.**[model]**
`parm`	Parameters for control of least-squares algorithm.**[control]**

`posit`	r and z coordinates associated with field or clearance wish.**[wishes]**
`rescale`	Flag to indicate whether or not rescaling `xnorm` and `x` should take place at initialization time.**[parameters]**
`rmajor`	Major radius of torsatron.**[model]**
`rminor`	Nominal minor radius of torsatron.**[model]**
`sconp`	"Tuning" parameter for constant pitch angle winding law.**[model]**
`tdip`	Radius of conductors used to model helix.**[model]**
`type`	String indicating type of wish (eg dipole moment).**[wishes]**
`value`	Desired (optimal) value for field quantity wishes, desired minimum distance for clearance wishes.**[wishes]**
`weight`	Relative contribution of wish discrepancy to residual.**[wishes]**
`wflag`	Flag to indicate special model (eg constant pitch angle).**[model]**
`xnorm`	Matrix which describes the mapping of the parameter vector onto the variable part of the model.**[parameters]**
`xstart`	Starting value of the parameter vector (see rescale).**[parameters]**

Example of a Complete Input File

A complete input file is included here for purposes of illustration. The output that corresponds to this input is used for examples in the next chapter.

```
  THIS SHOULD PRODUCE RESULTS SIMILAR TO THE RESULTS
  FROM THE T-1 DATA IN FILE SOLINT1

*END-COMMENTS

* SPECIFICATION OF THE MODEL

        NVC = 2
        NHC = 16
        IWN = 18
        IWL = 4
        ORDA = 0
        RMAJOR = 29.18
        RMINOR = 4.00
        ORDB = 0
        TDIP = 0.20
        CUR = 3.667E+07
        $

* PARAMETERS CONTROLLING THE MODEL

        NP = 3
        RESCALE = "YES"
        (XSTART(IP),IP=1,3) =

                   36.0,    9.0,   -3.5E+7

        (((XNORM(IP,IQ,IC),IP=1,4),IQ=1,4),IC=1,2) =

                    1       0       0           0
                    0       1       0           0
                    0       0       0           0.15
                    0       0       1           0

                    1       0       0           0
                    0      -1       0           0
                    0       0       0           0.15
                    0       0       1           0
          $

* WISHES

        POSIT = 28.18, 1.00 VALUE = 0 WEIGHT = 1.E+4
           TYPE = "BRAD"    $
        POSIT = 28.68, 1.00 VALUE = 0 WEIGHT = 1.E+4
           TYPE = "BRAD"    $
        POSIT = 29.18, 1.00 VALUE = 0 WEIGHT = 1.E+4
           TYPE = "BRAD"    $
        POSIT = 29.68, 1.00 VALUE = 0 WEIGHT = 1.E+4
           TYPE = "BRAD"    $
        POSIT = 30.18, 1.00 VALUE = 0 WEIGHT = 1.E+4
           TYPE = "BRAD"    $

        POSIT = 28.18, 0.00 VALUE = 0 WEIGHT = 1.E+4
           TYPE = "BVER"    $
        POSIT = 28.68, 0.00 VALUE = 0 WEIGHT = 1.E+4
           TYPE = "BVER"    $
        POSIT = 29.18, 0.00 VALUE = 0 WEIGHT = 1.E+4
           TYPE = "BVER"    $
        POSIT = 29.68, 0.00 VALUE = 0 WEIGHT = 1.E+4
           TYPE = "BVER"    $
        POSIT = 30.18, 0.00 VALUE = 0 WEIGHT = 1.E+4
           TYPE = "BVER"    $

        POSIT = 29.18, 0.0 VALUE =    0 WEIGHT = 1.E-8
           TYPE = "DIPL"    $

        POSIT = 29.18, 0.0 VALUE =  0.6 WEIGHT = 1.E+4
           TYPE = "CONR"   COIL = 1    $

        POSIT = 0.0, 1.0 VALUE =  10.0 WEIGHT = -1.E+4
           TYPE = "CONL"   COIL = 1    $

           TYPE = "END"    $ (DUMMY - TERMINATOR)

* CONTROL

        MAXFN = 1000
        NSIG = 5
        EPS = 1.0E-08
        DELTA = 1.0E-07
        IOPT = 1
        $
```

Output from Solve

Like the input, the code output is easily divided into parts. However, where the input divided along functional lines related to the operation of the code, the output divides according to complexity. The first part of the output summarizes the problem input. The second part simply gives you "The Answer" and an indication of whether or not it's a good one. The third part tells you a bit more about the result — how good it is, in terms of the criteria for optimization that were set forth in the input. Finally the fourth part gives details about the neighbourhood of the solution, revealing information about how sensitive the result is to small changes in the input conditions.

Each page of the output contains the run number in the upper left corner. If the page is one of a number that all deal with the same type of data, then a page number will appear in the upper right corner. This does not refer to the page position in the entire file, but only to the position in a number of pages dealing with the same data. For example, the eigenvalues are printed column-wise across the page. If there are more than five parameters in the solution, there will be more than one page of eigenvalues, and the first page of eigenvalues will contain `p(1)`, the second `p(2)`, and so on.

In the sections that follow, page breaks are indicated by horizontal lines across the full width of the page.

Summary of the Problem

This part is self-explanatory. All the data specifying the problem from the input file, plus the effects of any defaults, are shown here. The parameter vector `xstart` and mapping matrix `xnorm` appear in rescaled form if that was desired. The wishes are sorted by type.

```
                         SOLVE   VERSION  4.2
                        *************************


                      RUN NUMBER     1 AT 06:39:54

                      A (A ) MACHINE ON 06/03/81


  THIS SHOULD PRODUCE RESULTS SIMILAR TO THE RESULTS
  FROM THE T-1 STUFF IN FILE SOLINT1
```

```
(    1)  SETUP OF TORSATRON MODEL


                NUMBER OF VARIABLE COILS :    2
                NUMBER OF CONSTANT COILS :    0
                NUMBER OF  HELIX   COILS :   16



         HELIX WINDING DATA

                N : 18
                L :  4
                LINEAR DEPENDANCE OF THETA ON PHI


                MAJOR RADIUS :  29.1800
                MINOR RADIUS :   4.0000
                CONSTANT MINOR RADIUS
```

```
        SIZE OF COILS USED TO MODEL HELIX :   0.2000

        TOTAL TOROIDAL CURRENT :  3.6670E+07

 -------------------------------------------------------------------------

 (    1)  XSTART AND XNORM - PARAMETER TO MODEL COUPLING              P( 1, 1)

   PARAM         ( 1)          ( 2)          ( 3)        CONSTANT

              1.0000E+02    1.0000E+02    1.0000E+02       1.0
  COIL

 (   1) R     3.6000E-01    0.            0.            0.
        Z     0.            9.0000E-02    0.            0.
        T     0.            0.            0.            1.5000E-01
        I     0.            0.           -3.5000E+05    0.

 (   2) R     3.6000E-01    0.            0.            0.
        Z     0.           -9.0000E-02    0.            0.
        T     0.            0.            0.            1.5000E-01
        I     0.            0.           -3.5000E+05    0.

 -------------------------------------------------------------------------

 (    1)  WISHES                                                        P( 1)

          TYPE                R         Z          VALUE          WEIGHT      COIL

    1     RADIAL FIELD     28.180     1.000      0.            1.0000E+04
    2     RADIAL FIELD     28.680     1.000      0.            1.0000E+04
    3     RADIAL FIELD     29.180     1.000      0.            1.0000E+04
    4     RADIAL FIELD     29.680     1.000      0.            1.0000E+04
    5     RADIAL FIELD     30.180     1.000      0.            1.0000E+04
  6   VERTICAL FIELD      28.180     0.         0.            1.0000E+04
  7   VERTICAL FIELD      28.680     0.         0.            1.0000E+04
  8   VERTICAL FIELD      29.180     0.         0.            1.0000E+04
  9   VERTICAL FIELD      29.680     0.         0.            1.0000E+04
 10   VERTICAL FIELD      30.180     0.         0.            1.0000E+04

 11    DIPOLE MOMENT                            0.            1.0000E-08
 12   RADIAL CONSTR.      29.180     0.         6.0000E-01    1.0000E+04        1
 13   PLANAR CONSTR.       0.        1.000      1.0000E+01   -1.0000E+04        1
```

```
(   1)  CONTROL OF SOLVE

             MAXIMUM NUMBER OF FUNCTION EVALUATIONS :  1000
                      NUMBER OF SIGNIFICANT FIGURES :     5
          MINIMUM RELATIVE CHANGE IN SUM OF SQUARES :  1.000E-08
                           MINIMUM NORM OF GRADIENT :  1.000E-07


                  ALGORITHM OPTION IOPT = 1
                  STRICT DESCENT WITH DEFAULT PARAMETERS
```

Basic Results

This part of the output contains the final values of the dimensionless parameters and the corresponding configuration of the variable coils. It also contains indications of whether the least-squares routine converged or not, and if so, what convergence criteria were satisfied.

```
(   1)  BASIC RESULTS
```

Output from Solve

```
INFER =    1

     FIRST CONVERGENCE CRITERION SATISFIED
     PARAMETER ESTIMATES AGREE TO NSIG DIGITS
     ESTIMATED NUMBER OF SIGNIFICANT FIGURES :    5


FINAL PARAMETER VALUES


          ( 1)        1.02416E+02
          ( 2)        9.88423E+01
          ( 3)        3.35716E+01


            ESTIMATED SIGNIFICANT DIGITS :      5

          NUMBER OF FUNCTION EVALUATIONS :     39
                    NUMBER OF ITERATIONS :     12

                        NORM OF GRADIENT :  2.2109E-02
               FINAL MARQUARDT PARAMETER :  3.9502E-08


FINAL VARIABLE COIL CONFIGURATION

             R              Z              T              I

( 1)     36.8698         8.8958         0.1500     -1.1750E+07
( 2)     36.8698        -8.8958         0.1500     -1.1750E+07
```

At the top of the page is the variable `infer`. This will be non-zero if the least–squares routine has satisfied one of the convergence criteria. The explanation of the value is directly below. If the routine has not converged the explanation of the failure will appear there instead. (For an example of a failing iteration see Chapter 4.)

Next on the page are the final values of the dimensionless parameters. The mapping from these onto the coil configuration appears later on this page, but these dimensionless values are sometimes useful as well for sensitivity analysis.

The estimated number of significant digits in the final values for the parameters is shown, regardless of which convergence criterion is satisfied. This is only valid if the algorithm converged in some way. If the magnitudes of the parameters differ greatly, this will overestimate slightly the significant digits in the smaller parameters.

The number of function evaluations is the total number of times the model routine was called, while the number of iterations is the number of steps in parameter space that the least-squares routine made. More than one function evaluation is needed for each step as the routine uses finite difference techniques to evaluate gradients in parameter space.

The norm of the gradient is an indication of how flat the utility function is in the region of the final parameter values. The final Marquardt parameter is an indication of how the algorithmic step process converged on the answer. These are discussed in Chapter 5.

At the end of the page the final coil configuration is displayed. This is just the mapping of the dimensionless parameters shown earlier, using `xnorm` as supplied in the input.

Residuals — The Quality of the Solution

This section of the output is concerned with the fulfillment of the user's wishes. To be more precise, it lists the discrepancy between the wished-for value and the result as found from the model. At the top of the page is the total sum of the squars of all the dimensionless residuals. This is the number that the code has been minimizing. Below that there is a line for each wish, showing the dimensionless residual and the discrepancy in physical units. Since it is difficult to list all the information about a wish on one line, the general type of the wish is listed along with the number corresponding to the full listing of the wishes in the input section.

```
(    1)  RESIDUALS                                                      P( 1)
```

Output from Solve

```
RESIDUAL SUM OF SQUARES :  9.08021E+03


          WISH                   NORMALIZED                PHYSICAL


(  1)     RADIAL FIELD         -5.77003E+01          -5.77003E-03   TESLA
(  2)     RADIAL FIELD         -3.44177E+01          -3.44177E-03   TESLA
(  3)     RADIAL FIELD         -7.92696E+00          -7.92696E-04   TESLA
(  4)     RADIAL FIELD          2.20298E+01           2.20298E-03   TESLA
(  5)     RADIAL FIELD          5.56426E+01           5.56426E-03   TESLA

(  6)   VERTICAL FIELD          1.80981E+01           1.80981E-03   TESLA
(  7)   VERTICAL FIELD         -3.89764E+00          -3.89764E-04   TESLA
(  8)   VERTICAL FIELD         -1.33602E+01          -1.33602E-03   TESLA
(  9)   VERTICAL FIELD         -8.63534E+00          -8.63534E-04   TESLA
( 10)   VERTICAL FIELD          1.20333E+01           1.20333E-03   TESLA

( 11)    DIPOLE MOMENT         -1.34711E+01          -1.34711E+09   AMP-M↑2
( 12)   RADIAL CONSTR.          0.                    0.             METER
( 13)   PLANAR CONSTR.          0.                    0.             METER
```

You should note that the physical residuals are still just that — residuals. They show the discrepancy between your wish and the value from the model in its final form. Refer to the list of wishes in the input summary to find out what you wished for.

In the next section on sensitivity analysis I will show how to use the information about the residuals to greater advantage.

Jacobians and Hessians — Sensitivity Analysis

In this section there are three interesting quantities. If we represent the parameter vector by x_i, the residuals by R_j, and the sum of the squares of the residuals by Φ, then these quantities are the Jacobian matrix :

$$J = \begin{pmatrix} \frac{\partial R_1}{\partial x_1} & \frac{\partial R_1}{\partial x_2} & \cdots \\ \frac{\partial R_2}{\partial x_1} & \frac{\partial R_2}{\partial x_2} & \cdots \\ \frac{\partial R_3}{\partial x_1} & \frac{\partial R_3}{\partial x_2} & \cdots \\ \vdots & \vdots & \end{pmatrix}$$

; the gradient vector:

$$\nabla\Phi = \begin{pmatrix} \frac{\partial \Phi}{\partial x_1} \\ \frac{\partial \Phi}{\partial x_2} \\ \vdots \end{pmatrix}$$

; and the Hessian matrix :

$$H = \begin{pmatrix} \frac{\partial^2 \Phi}{\partial {x_1}^2} & \frac{\partial^2 \Phi}{\partial x_2\, \partial x_1} & \frac{\partial^2 \Phi}{\partial x_3\, \partial x_1} & \cdots \\ \frac{\partial^2 \Phi}{\partial x_1\, \partial x_2} & \frac{\partial^2 \Phi}{\partial {x_2}^2} & \frac{\partial^2 \Phi}{\partial x_3\, \partial x_2} & \cdots \\ \frac{\partial^2 \Phi}{\partial x_1\, \partial x_3} & \frac{\partial^2 \Phi}{\partial x_2\, \partial x_3} & \frac{\partial^2 \Phi}{\partial {x_3}^2} & \cdots \\ \vdots & \vdots & \vdots & \end{pmatrix}$$

The Jacobian and gradient appear as follows in the output.

```
(    1)  JACOBIAN (DIMENSIONLESS)                                          P( 1, 1)

   PARAM        ( 1)          ( 2)          ( 3)

 WISH

(  1)       -1.4575E+01    3.2671E+00    1.1125E+00

(  2)       -1.6857E+01    3.1830E+00    1.7398E+00

(  3)       -1.9336E+01    3.0105E+00    2.4652E+00

(  4)       -2.1976E+01    2.7281E+00    3.2962E+00

(  5)       -2.4720E+01    2.3128E+00    4.2384E+00

(  6)       -8.0889E-01    3.5601E+01   -1.3557E+02

(  7)       -8.7364E+00    3.7190E+01   -1.3482E+02
( 8)     -1.7847E+01    3.8707E+01   -1.3374E+02

( 9)     -2.8223E+01    4.0105E+01   -1.3226E+02

( 10)    -3.9923E+01    4.1320E+01   -1.3034E+02

( 11)    -1.9599E+01    0.           -2.9894E+01

( 12)     0.            0.            0.

( 13)     0.            0.            0.
```

The Jacobian shows the effects of changing a parameter on individual residuals. It is here that you can find the details of "what is being traded–off against what?" Consider the Jacobian matrix shown above.

Zero components show where a parameter has no effect on a wish. Since the constraints are not violated in the final solution, the Jacobian has two rows of zeros corresponding to those wishes. Also in wish 11, parameter 2 has no effect on the dipole moment. This is as expected — parameter 2 controls the height of the coil.

The remaining entries in the second column are all positive. This shows that all the residuals corresponding to those wishes are increasing as the parameter increases. However, looking back at the residuals in the previous section we can see that some are negative and some positive. This is a common situation. The field quantities that the first ten wishes specify are changing in approximately the same manner with the parameter. However not all achieve their desired values at exactly the same parameter value. So the result minimizes the sum of the square residuals, leaving some residuals positive and some negative.

Column 3 shows another effect. Some elements have much larger magnitudes than others. It shows that the current in the vertical field coil has much more effect on the vertical field than on the radial field component. This is a function of the position of the coil of course. Keep in mind that these magnitudes are all in terms of the dimensionless parameters and residuals. If you want the effect in terms of real–model parameter on real–wish result, you will have to multiply by the weight of the wish and similarly extract the effect of the parameter mapping matrix.

While the Jacobian shows the effect of parameter changes on individual residuals, the gradient and Hessian show the effects on the sum-of-squares function that is being minimized. The Hessian is not printed in its raw form. Instead it is decomposed into its eigenvectors, and the normalized vectors and corresponding eigenvalues are shown instead. (Since the Hessian is real symmetric, the eigenvalues are real.)

```
(    1)  GRADIENT (DIMENSIONLESS)


                      ( 1)        2.46294E-02
                      ( 2)       -3.30213E-03
                      ( 3)       -1.05249E-02
```

```
(    1)  EIGENVALUES AND EIGENVECTORS (DIMENSIONLESS)                          P( 1)

   EIGEN-       ( 1)           ( 2)           ( 3)
   VALUE     1.1183E+02     3.2279E+03     9.9160E+04

  PARAM

(   1)      -1.2285E-01    -9.8211E-01     1.4270E-01

(   2)      -9.5869E-01     8.0271E-02    -2.7288E-01

(   3)      -2.5654E-01     1.7033E-01     9.5141E-01
```

The gradient should be small at the final solution, because the code is supposed to find a minimum. The gradient is displayed mainly to show "how flat is Φ around the point that has been found?" It provides a way to compare the different convergence criteria also.

Since the first derivatives of Φ with respect to the parameters are approximately zero, it is the second derivatives which give the lowest order description of the way that Φ changes significantly around the solution point. (In a two dimensional parameter space you can imagine Φ plotted as the height of a surface, like Figure 1.1, and then the Hessian tells you what shape is the hole that you're in.)

The eigenvectors are normalized, and shown ordered by increasing eigenvalue. The larger the eigenvalue, the greater the change of Φ for a given step in the direction of the eigenvalue in parameter space. In the example above, for instance, a change in the height of the coil (parameter 2), with small changes in the radius and current, is the least significant change in terms of the utility function. It is in this direction that errors will be least critical. Changes in parameter 3, on the other hand, are an order of magnitude more critical then changes in the height.

Keep in mind that these are changes in the dimensionless parameters — multiply by `xnorm` to find the consequences in terms of real-world parameters, but do this cautiously. Remember that the problem has been solved with the number of dimensionless parameters as the number of degrees of freedom, and looking at "eigenvectors" in real-model parameter space obscures the fact that the solution algorithm has not, in general, been able to move through all of that model space. This warning is particularly important when `xnorm` has a more complicated structure than the essentially diagonal one in the above version.

Sample Input and Output

The last two chapters were a very basic primer on using Solve. While talking in grandiose terms about parameter space and so forth, a simple result was obtained for a simple problem. This chapter is meant to fill the gap between promise and practice by demonstrating some of the "tricks of the trade", and showing a couple of interesting problems that can be posed to Solve. (One of them can be solved too!)

Equality Constraints Using the Dimensionless Parameters

This section deals mainly with using the linear mapping specified by `xnorm` to select the portion of the model space to be investigated. For reasons of brevity (and boredom), complete input files are not shown. Instead, only the relevant portions are shown here; they should be placed in the context of a complete input file as described in Chapter 2.

In the excerpts from the input files in the examples that follow, I have followed the convention in the implict do loops that `ip` is the index that iterates over different dimensionless parameters; `iq` is the index that iterates over different model quantities (ie $1 - R$, $2 - Z$, $3 - t$, $4 - I$), and `ic` iterates over different coils of the model. Also remember that the mapping matrix `xnorm` has one more column than the number of parameters (`ip` runs to one more than `np`).This is the column that maps the constant 1.0 onto the model vector.

Setting up a standard pair of coils above and below the $Z = 0$ plane. This is the simplest common situation. We want to optimize the arrangement of a single pair of coils, giving the variable coils all the usual degrees of freedom. We know that from symmetry arguments the coils should have the same radius and current, and heights of equal magnitude but opposite sign. We set the minor diameter of the coils to a constant 15cm.

```
   NP = 3

   (XSTART(IP),IP=1,3) =      2.0     1.5     1.5e+06

   (((XNORM(IP,IQ,IC),IP=1,4),IQ=1,4),IC=1,2) =

                                      1.      0.       0.       0.
                                      0.      1.       0.       0.
                                      0.      0.       0.       0.15
                                      0.      0.       1.       0.

                                      1.      0.       0.       0.
                                      0.     -1.       0.       0.
                                      0.      0.       0.       0.15
                                      0.      0.       1.       0.
```

The first parameter controls the radii; the second, the heights; and the third, the current in the loops. In the case where you have two or more pairs of loops to optimize, it makes sense to take advantage of the default value of 0 for all the elements of xnorm and avoid explicitly specifying regions of zeros.

```
* PARAMETERS CONTROLLING THE MODEL

   NP = 7

* set all the variable-coil minor radii

   (XNORM(8,3,IC),IC=1,4) = 4( 0.22 )

* a standard pair of variable coils controlled by parameters 1,2,3

   (XSTART(IP),IP=1,3) =  10.0,   8.0,   2.5E+7
   (((XNORM(IP,IQ,IC),IP=1,3),IQ=1,4),IC=1,2) =

                                1.        0.        0.
                                0.        1.        0.
                                0.        0.        0.
                                0.        0.        1.

                                1.        0.        0.
                                0.       -1.        0.
                                0.        0.        0.
                                0.        0.        1.

* another standard pair of variable coils controlled by parameters 4,5,6

   (XSTART(IP),IP=4,6) =  6.0,   8.0,   -1.5E+7

   (((XNORM(IP,IQ,IC),IP=4,6),IQ=1,4),IC=3,4) =

                                1.        0.        0.
                                0.        1.        0.
                                0.        0.        0.
                                0.        0.        1.

                                1.        0.        0.
                                0.       -1.        0.
                                0.        0.        0.
                                0.        0.        1.
```

Here I have specified the minor diameters of all the variable coils first. Then I specified the pieces of `xstart` and `xnorm` pertaining to the first pair, and then the pieces pertaining to the second pair. Nearly half of `xnorm` did not have to be mentioned at all since it was all zero. Also note the use of comments in the input. This is a useful feature of the **Namelist** input.

Setting up coils at fixed positions. A plausible situation is one where the machine designer has room for a loop in a restricted location, and he wants to optimize the current in that loop along with the positions and currents in loops farther out. This would probably be done in conjunction with a constraint keeping the other coils out of the restricted region, however the parameter section of the input might contain something like this.

```
NP = 7

XSTART(7) = 2.0E+6

(((XNORM(IP,IQ,IC),IP=7,8),IQ=1,4),IC=5,6) =

                 0.    1.2
                 0.    0.7
                 0.    0.22
                 1.    0.

                 0.    1.2
                 0.   -0.7
                 0.    0.22
                 1.    0.
```

Here I have specified a pair of coils with fixed radius, height and thickness, but with their current controlled by parameter 7. Remember that if you want a coil with *all* its quantities fixed, there is no need to take up `xnorm` space, and slow the code down — it should be specified in the model section of the input as a fixed coil.

Modelling a coil with more than one filamentary loop. This is one of the more complicated things that you can do with the mapping matrix. Suppose you want to check the limitations of modelling a coil with a single filament, and instead you want to model the coil with four filaments which are "tied together" — ie maintain a constant relationship in space with respect to each other.

```
NP = 3

(XSTART(IP),IP=1,3) =  10.0     6.5     5.0E+6

(((XNORM(IP,IQ,IC),IP=1,4),IQ=1,4),IC=1,8)  =

                              1.       0.       0.    -0.04
```

```
0.        1.        0.      -0.04
0.        0.        0.       0.05
0.        0.        0.20     0.

1.        0.        0.       0.04
0.        1.        0.      -0.04
0.        0.        0.       0.05
0.        0.        0.30     0.

1.        0.        0.       0.04
0.        1.        0.       0.04
0.        0.        0.       0.05
0.        0.        0.30     0.

1.        0.        0.      -0.04
0.        1.        0.       0.04
0.        0.        0.       0.05
0.        0.        0.20     0.

1.        0.        0.      -0.04
0.       -1.        0.      -0.04
0.        0.        0.       0.05
0.        0.        0.20     0.

1.        0.        0.       0.04
0.       -1.        0.      -0.04
0.        0.        0.       0.05
0.        0.        0.30     0.

1.        0.        0.       0.04
0.       -1.        0.       0.04
0.        0.        0.       0.05
0.        0.        0.30     0.

1.        0.        0.      -0.04
0.       -1.        0.       0.04
0.        0.        0.       0.05
0.        0.        0.20     0.
```

Here we have eight filaments, in two groups of four. The usual method has been used to make symmetric pairs above and below the $Z = 0$ plane. In addition small constants are alternately added and subtracted from the radius and height parameters letting us keep the four filaments in a square configuration with sides of 8cm. Finally note that the current has been divided up so that the outer filaments (greater radius) carry 60% of the current and the inner filaments only carry 40%.

As you can see, the mapping matrix allows quite sophisticated control of the model in parameter space.

Geometric Inequality Constraints Using the Wishes

As described above, the mapping from dimensionless parameters to model parameters allows one sort of control over the problem domain, essentially by allowing the least–squares routine to only access a subset of the entire model space. The inequality constraints which are part of the wishes are not true constraints in the same sense as above. Instead of making a region of model space inaccessible, they make the region increasingly undesirable by increasing the residual linearly from the boundary (which means the utility function will increase quadratically). This approach is called the penalty function technique, for obvious reasons.

Various excerpts from input files are shown in the same manner as the previous section to illustrate some common applications of the inequality constraints.

Constraining a coil to be outside the helix volume. This is the most common use of the geometric constraints. In fact, nearly every problem posed to Solve will have constraints of this form. Suppose you have a problem with four coils in the usual pairs, coils 1 and 2 forming one pair, coils 3 and 4 the other. The machine has a major radius of 10 meters and a coil minor radius of 4 meters. You decide to start the residual increasing a meter out from the helix coils themselves, so that a small violation of the constraint is not serious.

```
POSIT = 10.00, 0.00 VALUE = 5.0  WEIGHT = 1.E+4
   TYPE = "CONR"   COIL = 1    $

POSIT = 10.00, 0.00 VALUE = 5.0  WEIGHT = 1.E+4
   TYPE = "CONR"   COIL = 3    $
```

Note that, because of the usual symmetry in the pairs of coils, I don't have to explicitly put a constraint on coils 2 or 4.

Constraining a coil inside a region. Now suppose that Solve has produced a solution with a coil in a possible, but somewhat undesirable place. You want to "push" the coil into a more desirable place, but see how much it costs in the final residuals. Lets suppose you want the coil no more than 10 meters away from the center of the machine (major radius 15 meters), but at least 3 meters away. Also you want to be strict about the 3 meter inside criterion because of clearance from other coils, while you can be more relaxed about the 10 meter maximum. Finally, you want to keep the coil at a greater radius than the major radius of the machine, and this is more important than the 10 meter maximum, but not as important as the 3 meter minimum.

```
POSIT = 15.00, 0.00 VALUE = 3.0  WEIGHT = 1.E+4
   TYPE = "CONR"   COIL = 1     $

POSIT = 15.00, 0.00 VALUE = 10.0  WEIGHT = -1.E+3
   TYPE = "CONR"   COIL = 1     $

POSIT = 1.00, 0.00  VALUE = 15.0  WEIGHT =  3.E+3
   TYPE = "CONL"   COIL = 1     $
```

Remember that `posit` specifies the center for a radial constraint, the normal direction for a linear constraint, and `value` specifies the radius for a radial constraint, the distance from the origin for a linear one.

Keeping two coils separated in different regions. This example shows why it is useful to have geometric constraints apply to individual coils, rather than to all coils. Suppose you want to keep one coil above and to the left of the line $2Z = R$, while another is to be kept below and to the right of the same line.

```
POSIT = -1.00, 2.00  VALUE = 0.0  WEIGHT =  2.E+3
   TYPE = "CONL"   COIL = 1     $
```

```
        POSIT = -1.00, 2.00  VALUE = 0.0  WEIGHT =   -2.E+3
           TYPE = "CONL"   COIL = 3    $
```

Coil 1 will be "encouraged" to stay above the line, coil 3 below. Note that the only difference between the wishes is the sign of the weights. This could have been done by changing the sign of all the normal components (**posit**) instead.

A Large Residual Solution

This section demonstrates an example of a well posed problem for Solve. The code converges well on a minimum, however the minimum has large residuals left. The problem is adapted from the design of TOREX-4, a proposed [3] experimental torsatron. It was desirable to have well formed flux surfaces in the plasma region, and also to prevent power supply fluctuations from inducing currents in the plasma. The majority of wishes are therefore concerned with nulling both the field and the linked flux in the plasma region. In addition there is a geometric constraint to keep the nulling pair out of the helix and a dipole moment wish to reduce the far field magnitude.

The entire input file, apart from the comments, is shown here.

```
* SPECIFICATION OF THE MODEL

        NVC = 2
        NHC = 16
        IWN = 18
        IWL = 4
        RMAJOR = 2.06
        RMINOR = 0.36
        WFLAG = "CONPITCH"
        TDIP = 0.20
        CUR = 6.88E+06
        $

* PARAMETERS CONTROLLING THE MODEL
```

```
        NP = 3
        RESCALE = "YES"
        (XSTART(I),I=1,3) =

                   2.0,   1.0,  -3.5E+6

        (((XNORM(K,J,I),K=1,4),J=1,4),I=1,2) =

                   1       0       0          0
                   0       1       0          0
                   0       0       0          0.15
                   0       0       1          0

                   1       0       0          0
                   0      -1       0          0
                   0       0       0          0.15
                   0       0       1          0
          $

* WISHES

        POSIT = 1.96, 0.05 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BRAD"    $
        POSIT = 2.01, 0.05 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BRAD"    $
        POSIT = 2.06, 0.05 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BRAD"    $
        POSIT = 2.11, 0.05 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BRAD"    $
        POSIT = 2.16, 0.05 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BRAD"    $

        POSIT = 1.96, 0.00 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BVER"    $
        POSIT = 2.01, 0.00 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BVER"    $
        POSIT = 2.06, 0.00 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BVER"    $
        POSIT = 2.11, 0.00 VALUE = 0. WEIGHT = 4.E+4
```

```
   TYPE = "BVER"    $
POSIT = 2.16, 0.00 VALUE = 0. WEIGHT = 4.E+4
   TYPE = "BVER"    $

POSIT = 1.96, 0.00 VALUE = 0. WEIGHT = 4.E+4
   TYPE = "FLUX"    $
POSIT = 2.01, 0.00 VALUE = 0. WEIGHT = 4.E+4
   TYPE = "FLUX"    $
POSIT = 2.06, 0.00 VALUE = 0. WEIGHT = 4.E+4
   TYPE = "FLUX"    $
POSIT = 2.11, 0.00 VALUE = 0. WEIGHT = 4.E+4
   TYPE = "FLUX"    $
POSIT = 2.16, 0.00 VALUE = 0. WEIGHT = 4.E+4
   TYPE = "FLUX"    $

 VALUE =    0 WEIGHT = 1.
   TYPE = "DIPL"    $

POSIT = 2.06, 0.0 VALUE =  0.6 WEIGHT = 1.E+5
   TYPE = "CONR"   COIL = 1    $

   TYPE = "END"    $ (DUMMY - TERMINATOR)

* CONTROL

        MAXFN = 1000
        NSIG = 5
        EPS = 1.0E-08
        DELTA = 1.0E-07
        IOPT = 1
        $
```

The results of the run are shown here. The input summary and sensitivity analysis have been deleted for brevity.

```
(   7)  BASIC RESULTS

        INFER =   3

             FIRST CONVERGENCE CRITERION SATISFIED
             PARAMETER ESTIMATES AGREE TO NSIG DIGITS
             ESTIMATED NUMBER OF SIGNIFICANT FIGURES :   5

             SECOND CONVERGENCE CRITERION SATISFIED
             RELATIVE CHANGE IN RESIDUAL LESS THAN EPS


        FINAL PARAMETER VALUES


                  ( 1)       1.03049E+02
                  ( 2)       3.56461E+01
                  ( 3)       9.66818E+01


                     ESTIMATED SIGNIFICANT DIGITS :     5

                  NUMBER OF FUNCTION EVALUATIONS :    50
                            NUMBER OF ITERATIONS :    22

                                NORM OF GRADIENT :  1.9797E+04
                       FINAL MARQUARDT PARAMETER :  7.3458E-09


        FINAL VARIABLE COIL CONFIGURATION

               R               Z               T               I

  ( 1)       2.0610          0.3565          0.1500       -3.3839E+06
  ( 2)       2.0610         -0.3565          0.1500       -3.3839E+06
1(   7)  RESIDUALS                                                         P( 1)


   RESIDUAL SUM OF SQUARES :  7.49917E+09
```

```
         WISH                     NORMALIZED                  PHYSICAL


( 1)      RADIAL FIELD            1.73091E+04          4.32727E-01   TESLA
( 2)      RADIAL FIELD            2.05661E+04          5.14153E-01   TESLA
( 3)      RADIAL FIELD            2.16637E+04          5.41594E-01   TESLA
( 4)      RADIAL FIELD            2.01779E+04          5.04449E-01   TESLA
( 5)      RADIAL FIELD            1.66606E+04          4.16515E-01   TESLA

( 6)    VERTICAL FIELD           -4.62023E+04         -1.15506E+00   TESLA
( 7)    VERTICAL FIELD           -2.73868E+04         -6.84669E-01   TESLA
( 8)    VERTICAL FIELD           -6.46137E+03         -1.61534E-01   TESLA
( 9)    VERTICAL FIELD            1.42656E+04          3.56639E-01   TESLA
( 10)   VERTICAL FIELD            3.25503E+04          8.13756E-01   TESLA

( 11) NET LINKED FLUX             9.44211E+03          2.36053E-01   WEBER
( 12) NET LINKED FLUX            -6.33632E+03         -1.58408E-01   WEBER
( 13) NET LINKED FLUX            -8.95266E+03         -2.23816E-01   WEBER
( 14) NET LINKED FLUX             1.89905E+03          4.74763E-02   WEBER
( 15) NET LINKED FLUX             2.50378E+04          6.25945E-01   WEBER

( 16)    DIPOLE MOMENT           -2.10743E+02         -2.10743E+02   AMP-M↑2
( 17)   RADIAL CONSTR.            2.43538E+04          2.43538E-01   METER
```

Despite the radial constraint, the code has placed the coil very close to the helix winding. Even with the nulling coil this close, the solution is not a good one. There are residual fields of more than 1 Tesla in the plasma region. The weight of the constraint wish could be increased to force the nulling coils outward, but this would certainly yield even worse field conditions. There may be a better solution to be found elsewhere in parameter space, but it turns out that this is not so. The most important conclusion from this run is that one pair of coils is not sufficient to satisfy all the wishes.

A Small Residual Solution

Continuing with the example from the previous section, a two pair solution was tried. New parameters and another constraint wish have been added for the second pair of coils, but otherwise the input looks much the same as the previous example.

```
* SPECIFICATION OF THE MODEL

        NVC = 4
        NHC = 16
        IWN = 18
        IWL = 4
        RMAJOR = 2.06
        RMINOR = 0.36
        WFLAG = "CONPITCH"
        TDIP = 0.20
        CUR = 6.88E+06
        $

* PARAMETERS CONTROLLING THE MODEL

        NP = 6
        RESCALE = "YES"

        (XNORM(7,3,IC),IC=1,4) = 4(0.15)

        (XSTART(I),I=1,3) =

                   1.5,    1.0,   -1.0E+6

        (((XNORM(IP,J,IC),IP=1,3),J=1,4),IC=1,2) =

                    1.        0.        0.
                    0.        1.        0.
                    0.        0.        0.
                    0.        0.        1.
```

```
                  1.       0.       0.
                  0.      -1.       0.
                  0.       0.       0.
                  0.       0.       1.

        (XSTART(I),I=4,6) =

                  2.5,   1.0,  -1.0E+6 .

        (((XNORM(IP,J,IC),IP=4,6),J=1,4),IC=3,4) =

                  1.       0.       0.
                  0.       1.       0.
                  0.       0.       0.
                  0.       0.       1.

                  1.       0.       0.
                  0.      -1.       0.
                  0.       0.       0.
                  0.       0.       1.
          $

    * WISHES

        POSIT = 1.96, 0.05 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BRAD"    $
        POSIT = 2.01, 0.05 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BRAD"    $
        POSIT = 2.06, 0.05 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BRAD"    $
        POSIT = 2.11, 0.05 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BRAD"    $
        POSIT = 2.16, 0.05 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BRAD"    $

        POSIT = 1.96, 0.00 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BVER"    $
        POSIT = 2.01, 0.00 VALUE = 0. WEIGHT = 4.E+4
           TYPE = "BVER"    $
        POSIT = 2.06, 0.00 VALUE = 0. WEIGHT = 4.E+4
```

```
     TYPE = "BVER"    $
  POSIT = 2.11, 0.00 VALUE = 0. WEIGHT = 4.E+4
     TYPE = "BVER"    $
  POSIT = 2.16, 0.00 VALUE = 0. WEIGHT = 4.E+4
     TYPE = "BVER"    $

  POSIT = 1.96, 0.00 VALUE = 0. WEIGHT = 4.E+4
     TYPE = "FLUX"    $
  POSIT = 2.01, 0.00 VALUE = 0. WEIGHT = 4.E+4
     TYPE = "FLUX"    $
  POSIT = 2.06, 0.00 VALUE = 0. WEIGHT = 4.E+4
     TYPE = "FLUX"    $
  POSIT = 2.11, 0.00 VALUE = 0. WEIGHT = 4.E+4
     TYPE = "FLUX"    $
  POSIT = 2.16, 0.00 VALUE = 0. WEIGHT = 4.E+4
     TYPE = "FLUX"    $

   VALUE =    0. WEIGHT = 1.E-4
     TYPE = "DIPL"    $

  POSIT = 2.06, 0.0 VALUE =   0.6 WEIGHT = 1.E+6
     TYPE = "CONR"   COIL = 1    $

  POSIT = 2.06, 0.0 VALUE =   0.6 WEIGHT = 1.E+6
     TYPE = "CONR"   COIL = 3    $

     TYPE = "END"    $ (DUMMY - TERMINATOR)

* CONTROL

  MAXFN = 1000
  NSIG = 5
  EPS = 1.0E-08
  DELTA = 1.0E-07
  IOPT = 1
  $
```

The results are shown here.

```
(    9)  BASIC RESULTS

         INFER =    0 : CONVERGENCE FAILED

          IER  = 131 : THE MARQUARDT PARAMETER EXCEEDED
                       THE LIMIT PARM(3)


         FINAL PARAMETER VALUES


                    ( 1)       1.09313E+02
                    ( 2)      -4.28165E+01
                    ( 3)       3.09721E+02
                    ( 4)       9.91635E+01
                    ( 5)      -4.29255E+01

                    ( 6)       1.81887E+02


                      ESTIMATED SIGNIFICANT DIGITS :     9

                    NUMBER OF FUNCTION EVALUATIONS :   521
                              NUMBER OF ITERATIONS :   152

                                  NORM OF GRADIENT :  2.0314E+02
                         FINAL MARQUARDT PARAMETER :  2.2583E+02


         FINAL VARIABLE COIL CONFIGURATION

                R               Z               T               I

( 1)         1.6397         -0.4282          0.1500        -3.0972E+06
( 2)         1.6397          0.4282          0.1500        -3.0972E+06
( 3)         2.4791         -0.4293          0.1500        -1.8189E+06
( 4)         2.4791          0.4293          0.1500        -1.8189E+06
(    9)  RESIDUALS                                                    P( 1)


  RESIDUAL SUM OF SQUARES :   3.55599E+07


        WISH                     NORMALIZED                  PHYSICAL


(  1)     RADIAL FIELD            1.61397E+03           4.03493E-02   TESLA
(  2)     RADIAL FIELD            7.10159E+02           1.77540E-02   TESLA
(  3)     RADIAL FIELD            2.80916E+02           7.02289E-03   TESLA
(  4)     RADIAL FIELD            2.94909E+02           7.37272E-03   TESLA
(  5)     RADIAL FIELD            7.19029E+02           1.79757E-02   TESLA

(  6)   VERTICAL FIELD           -2.27861E+02          -5.69653E-03   TESLA
(  7)   VERTICAL FIELD            9.70279E+02           2.42570E-02   TESLA
(  8)   VERTICAL FIELD            1.50215E+03           3.75538E-02   TESLA
(  9)   VERTICAL FIELD            1.82558E+03           4.56395E-02   TESLA
( 10)   VERTICAL FIELD            2.36652E+03           5.91630E-02   TESLA

( 11) NET LINKED FLUX             2.14057E+03           5.35143E-02   WEBER
( 12) NET LINKED FLUX             7.74095E+02           1.93524E-02   WEBER
( 13) NET LINKED FLUX            -2.81851E+02          -7.04628E-03   WEBER
( 14) NET LINKED FLUX            -1.13515E+03          -2.83787E-02   WEBER
( 15) NET LINKED FLUX            -1.62122E+03          -4.05304E-02   WEBER

( 16)    DIPOLE MOMENT           -3.22476E+03          -3.22476E+07   AMP-M↑2
( 17)   RADIAL CONSTR.            1.64494E+01           1.64494E-05   METER
( 18)   RADIAL CONSTR.            8.79970E+01           8.79970E-05   METER
```

When the Marquardt parameter exceeds the limit, it is usually an indication that the iteration has broken down in a large uniform region of parameter space. The solution is not too bad in terms of the residuals, so this indicates that another try with slightly different starting point might converge. In fact, if we start the same problem with the following starting point, a solution with satisfactory convergence is obtained.

```
    (XSTART(I),I=1,3) =

              1.4,   0..7,  -6.0E+6

    (XSTART(I),I=4,6) =

              2.5,   0.5,  -1.5E+6
```

The final coil configuration is as follows.

```
     FINAL VARIABLE COIL CONFIGURATION

         R            Z            T            I

( 1)    1.4592       0.6649       0.1500     -6.3040E+06
( 2)    1.4592      -0.6649       0.1500     -6.3040E+06
( 3)    2.5020       0.4055       0.1500     -1.5493E+06
( 4)    2.5020      -0.4055       0.1500     -1.5493E+06
```

Theory of Operation

Solve and the Philosophy of Design

In some sense, most design is optimization. It is, however, a long way from this global view to defining some sort of value or utility function for any design problem that the designer can then minimize. (Or maximize — traditionally, optimizers have always minimized, but the difference is only one of viewpoint.)

I don't suggest that all design problems should be posed as minimization problems, nor do I even claim to have the final word on the particular problem described in this report. The technique used here is most useful in conjunction with some educated physical insight, and is presented here as a part of the design process. To illustrate this, examples of its use were shown, with comments showing the physical reasoning that went into the process between computations.

I found that the torsatron vertical field coil problem was too complex to solve analytically in its entirety. On the other hand, it was simple enough to specify the criteria for an acceptable solution in semi-quantitative terms — such as reducing certain quantities as much as possible. This lead to formulating the problem as one of reducing the error in a set of quantities as much as possible, and implementing this by using a least-squares technique. This approach should be useful in any situation where parametric studies would otherwise be performed.

Overview of the Program

The overall structure of the program can be described in terms of four levels. The top level handles input and output, including various utilities for interpreting the solution to the user. The second level is the least squares routine. Next, is the level which maps the dimensionless quantities of the least-squares routine onto the physical quantities of the model. At this level the reverse transformation — from the discrepancy in physical units to the dimensionless residual — is also performed. The lowest level contains the physical model of the torsatron.

Figure 5.1 shows a schematic diagram of the structure of the code. This is not an exact flowchart of the program, but rather an overview, showing the important flows of control and data. Data flows pertaining to input and output have been omitted for clarity — it should be fairly obvious, for instance, that the routine that reads the wishes has to initialize the residual mapping data. The four levels can be seen, approximately, running from left to right.

The least squares algorithm used is a modified Levenberg-Marquardt algorithm [1] as implemented in the July 1977 release of the IMSL mathematical subroutines package [2]. This routine uses finite differences to evaluate the local Jacobian matrix of derivatives. This algorithm is discussed in a later section.

The routine that interfaces the model to the least–squares routine does three things. Every time it is called by the least–squares routine it maps the dimensionless parameters of the least–squares routine onto the parameters of the model. It then changes the model to match these parameters. Secondly it calls the routines to find fields, fluxes and the like and directly accesses the model data to calculate geometric clearances. Finally it takes the wish data and converts that to dimensionless residuals for passing back to the least–squares routine.

Why the Model was Simplified

There is a trade-off between a set of conditions that completely models the entire physical situation, but which is cumbersome computationally, and a simplified set that uses only quantities that are computed quickly, but may ignore certain effects. Because the iterative algorithm used may call the physical model hundreds of times, we have found it more useful to pose the optimization problem in very simple form, and then go to more detailed analysis after this procedure has suggested a solution.

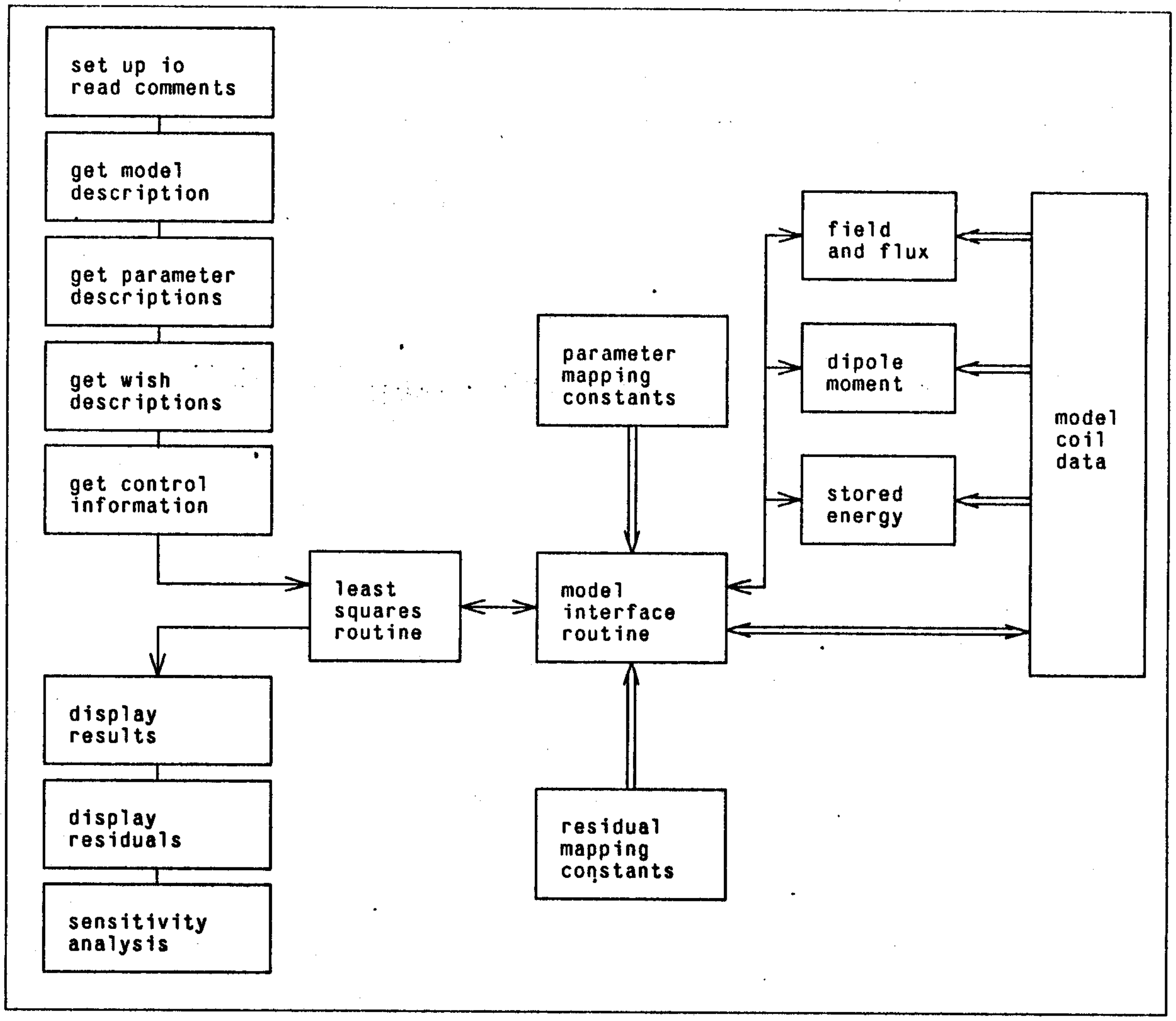

Figure 5.1 — Structure of Solve.

The general organization of Solve is shown here. Flow of control is indicated by thin arrows, while important data paths are shown with thick arrows. Some data paths (eg initialization of the model) have been omitted for clarity.

By separating the model loops into variable and constant coils we also increase efficiency. The field contribution from the constant coils to the wishes is calculated only once, likewise the stored energy terms for pairs of constant coils are calculated to start with and never altered again.

By limiting the geometric wishes to simple clearances from a point and a line we have a set of constraints that can delimit a variety of regions, without becoming too cumbersome, or causing problems when they overlap.

It would be useful to be able to optimize (i.e. wish for) quantities such as the peak force on the helix winding, or the maximum field in the conductor, but the increase in the model complexity would be so great that the cost of a large number of trial runs would become prohibitive. Some thoughts on possible extensions are mentioned in the last section, however.

How the Model was Simplified

As mentioned in Chapter 1, it was useful to be able to express the distribution of currents in the helices of the torsatron as an equivalent surface current, having components in the surface of the torus. (Remember also, that we allow the torus to be non-circular in cross section.)

Let us assume that we have specified the winding law of the helix and the cross-section shape of the torus by relations such as

$$\phi = \frac{l}{N} T^*(\theta)$$

and

$$r = r_0 \rho^*(\theta).$$

(Note that equations 2.1 and 2.2 are of this form.) Then the vector tangent to the helix (not normalized) will have toroidal components (r, θ, ϕ) of

$$\left(\frac{\partial \rho^*}{\partial \theta}\, ,\, \rho^*\, ,\, \frac{l}{N}\frac{R}{r_0}(1 + \frac{r_0}{R}\rho^* \cos\theta)\frac{\partial T^*}{\partial \theta} \right). \tag{5.1}$$

We define a surface current density K which is everywhere parallel to the helix tangent, so

$$K_r = \frac{1}{\rho^*}\frac{\partial \rho^*}{\partial \theta}. \tag{5.2}$$

and we normalize it such that

$$\begin{aligned} \int_0^{2\pi} K_\phi r_0 \, d\theta &= \sum_{toroidal} I \\ \int_0^{2\pi} \sqrt{K_\theta^2 + K_r^2}(R + r_0 \cos\theta)\, d\phi &= \sum_{poloidal} I \end{aligned} \tag{5.3}$$

Now in the limit of a smooth distribution (i.e. as $l \to \infty$, and $N \to \infty$, but the ratio N/l remains constant), the surface current is no longer a function of ϕ. Realizing that the current is divergenceless, we get the following dependencies:

$$K_r(\theta) = \frac{1}{2\pi} \sum_{poloidal} I \frac{1}{(R + r_0\rho^* \cos\theta)} \frac{\frac{\partial\rho^*}{\partial\theta}}{\sqrt{\rho^{*2} + \frac{\partial\rho^*}{\partial\theta}^2}} \tag{5.4a}$$

$$K_\theta(\theta) = \frac{1}{2\pi} \sum_{poloidal} I \frac{1}{(R + r_0\rho^* \cos\theta)} \frac{\rho^*}{\sqrt{\rho^{*2} + \frac{\partial\rho^*}{\partial\theta}^2}} \tag{5.4b}$$

$$K_\phi(\theta) = \frac{1}{2\pi} \sum_{toroidal} I \frac{1}{r_0} \frac{\frac{\partial T^*}{\partial\theta}}{\sqrt{\rho^{*2} + \frac{\partial\rho^*}{\partial\theta}^2}} \tag{5.4c}$$

This sheet current can be further simplified to a number of filamentary circular loops. This is done by replacing the torus surface with a number of loops, equally spaced in poloidal angle, on the surface. The total current in all the loops is equal to the total toroidal current, as used in the formula for K_ϕ, and the individual currents are in the same ratios as the point values of $K_\phi(\theta)$ at their respective places on the surface.

Thus the model for all the conductors in the torsatron, both helices and vertical field coils, is reduced to a sum of the well known functions of elliptic integrals that describe the field

$$B_r = \frac{\mu_0 I}{4\pi} \sqrt{\frac{rm}{a}} \frac{z}{2r^2} \left[2K(m) - \frac{2-m}{1-m} E(m) \right] \tag{5.5a}$$

$$B_z = \frac{\mu_0 I}{4\pi} \sqrt{\frac{rm}{a}} \frac{1}{r} \left[K(m) - (1 - m\frac{r+a}{2r(1-m)})E(m) \right] \tag{5.5b}$$

and the vector potential

$$A_\phi = \frac{\mu_0 I}{4\pi} \sqrt{\frac{a}{rm}} 2[(2-m)K(m) - 2E(m)] \tag{5.5c}$$

from a circular filamentary conductor, given here in cylindrical coordinates. K and E are complete elliptic integrals of the first and second kind, and $m = 4ar/((a+r)^2 + z^2)$.

Introduction to the Operation of Least-Squares Optimizers.

The general optimization problem is defined simply. Find the vector x such that the magnitude of some function $\Phi(x)$ is reduced as much as possible subject to inequality constraints $G(x) \leq 0$ and equality constraints $H(x) = 0$.

When the utility function Φ and the constraints G and H are linear in the components of x then the optimization problem is well understood. The problem is called a linear programming problem and a large arsenal of methods is available for its solution. The well known Simplex algorithm is the basis of most of these.

If the utility function is not linear, we are not so well off, but if the function is convex over the domain specified by the constraints, and the domain is convex, then there are still some useful properties of a solution. Most importantly, if the above properties hold, then a local solution will be a global solution. That is if we have some x^*, such that $\Phi(x^*)\Phi(x)$ for all x in the *neighborhood* of x^*, then $\Phi(x^*)\Phi(x)$ for *all* x in the domain of the problem.

In cases where we cannot guarantee such convexity properties *there is no way to tell whether a given local minimum is in fact a global minimum.* This single fact is the root cause of most of the difficulties associated with this method. In terms of the specific torsatron problem, this is the reason that (as mentioned in Chapter 1) the final solution may depend on the starting point of the optimizer.

With these caveats in mind let us take a look at algorithms for minimizing Φ, concentrating on common features, rather than small differences. In all cases we are basicly interested in producing a sequence of approximations $x_0, x_1, x_2, \ldots$ which approaches x^*.

Newton's method in one dimension is the starting point for this discussion. If we seek a root of $f(x) = 0$, where this is a scalar function of a scalar, Newton's method prescribes the iteration step as

$$x_{i+1} - x_i = -\frac{df(x_i)}{dx}^{-1} f(x_i)$$

which extrapolates the tangent to the curve towards zero. It is useful to regard this as having been derived from the first order Taylor series for f around the present value.

$$0 = f(x^*) = f(x) + \frac{df(x)}{dx}(x^* - x)$$

In many dimensional parameter space we are trying to minimize the length (more generally, some norm) of a residual vector R, or equivalently minimize

$$\Phi = \frac{1}{2}\,||R||^2.$$

A Taylor expansion of R about the zero :

$$0 = R(x^*) = R(x) + J(R(x))(x^* - x)$$

may be written in the same manner as the one dimensional case. (J denotes the Jacobian matrix, and x is now a vector.) In the sort of problem that we are dealing with, the Jacobian is not square — the problem is overdetermined because there are more components to the residual than there are parameters — so the Newton step is found as a minimum norm solution :

$$\left(J(R(x_i))^T J(R(x_i))\right)(x_{i+1} - x_i) = -J(R(x_i))^T R(x_i)$$

The problem with Newton's method, in one or more dimensions, is that if the starting point is not sufficiently close to the optimum, the method may overshoot the point and oscillate about it, or fail to converge in other ways. The modified Newton method is an attempt to correct this. Even though the Newton step may be of the wrong size, we know that Φ is decreasing in that direction. So the modified Newton step is given by :

$$x_{i+1} - x_i = -\lambda_i s_i$$

where the s_i are typically the Newton step vectors, and the λ_i are scalar constants chosen to make $\Phi(x_i)$ a decreasing series. One choice for λ_i is

$$\frac{1}{\lambda_i} = \mathrm{cond}\left(\left(J(R(x_i))^T J(R(x_i))\right)^{-1} J(R(x_i))^T\right)$$

which looks complicated, but it isn't really. The argument of cond is simply the solution matrix from the newton step equation above, and cond is the condition number of the matrix. This means that if the matrix is well conditioned, the condition number will be near 1, and the step approaches a full Newton step. If the eigenvalues of the matrix differ widely in magnitude, the condition number becomes large, and a smaller step will be taken.

This algorithm is quite expensive to evaluate, so certain efficiency measures can be taken. The Jacobian can be replaced with a finite difference equivalent. One can go even further and replace the finite difference matrix with another, even simpler, one that requires less work to evaluate. This is the basis of update methods, however this is getting beyond the scope of an introductory discussion. A good reference for further study is Stoer and Bulirsch [6].

Instead of choosing λ_i as above to ensure a decreasing sequence of $\Phi(x_i)$ it is instead possible to choose μ_i and define the Levenberg Marquardt step for a decreasing sequence of $\Phi(x_i)$. This is the default algorithm for Solve.

$$\left(\mu_i I + J(R(x_i))^T J(R(x_i))\right)(x_{i+1} - x_i) = -J(R(x_i))^T R(x_i)$$

The μ_i are small positive constants and I is the identity matrix. Having mentioned the method I will point the interested reader at the literature [2] for a full explanation.

A short word about terminating these algorithms is in order. The most obvious criterion is to stop when the steps in parameter space get small enough. This is the `nsig` criterion. This is reasonable as long as the final parameter magnitudes are similar. (And this, incidentally, is the reason for default rescaling of the starting parameters. Assuming that the starting point is of the same magnitude as the optimum, then the number of significant figures in each of the optimum coordinates will be the same.)

The other two criteria (norm of the gradient, and relative change in the residual) both serve the same purpose. They stop the algorithm from wandering about in a flat region of Φ. The norm of the gradient refers to parameter space, the other to residual space. Both of these, however, are not scale-free, that is they depend on the "shape" of the space around the minimum. A better criterion that has been suggested by Dennis, Gay, and Welsch [7], is to use the fact that at a minimum, the column space of the Jacobian is perpendicular to the residual vector. Dennis et al. suggest using the maximum cosine between the residual and any column as a convergence criterion. This is independent of the scale of the problem, but this condition has not been implemented in the present algorithm.

Applications, Extensions

Solve was first written to optimize the vertical field coil configuration for the proposed experimental device TOREX-4 [3]. The wishes included field and flux nulling in the plasma region. A zero residual solution was found with two pairs of coils.

Similar methods have been applied to other magnetic configurations. The O.R.N.L. mhd equilibrium code has been run in conjunction with a similar least squares code [5]. The method finds a distribution of real conductors which produces the poloidal fields necessary for a given free boundary equilibrium.

Finally, it is tempting to speculate about a much more powerful optimizing code, which allows a very general model beneath it, with a large variety of "diagnostics" to examine the model and make wishes. It would, of course, be interactive with graphical feedback — it could plot the utility function surface (like Figure 1.1) at various points, or draw the current state of the model. Most importantly, it would be extensible, so that a user could write his own diagnostic into the model, and define a new type of wish. This, however, is an undertaking of no small magnitude and would require a fair amount of computational power.

References

1. Brown, K.M. and Dennis, J.E., "Derivative Free Analogues of the Levenberg-Marquardt and Gauss Algorithms for Nonlinear Least Squares Approximations", *Numerische Mathematik* **18**, 289–297, 1972.
2. I.M.S.L. Inc., *International Mathematical Subroutine Library, Edition 6*, I.M.S.L., Houston, Texas. 1977.
3. Politzer, P.A., L.M. Lidsky, D.B. Montgomery, "TOREX-4. A Torsatron Proof of Principle Experiment", MIT report PFC-TR-79-2, 1979.
4. Lundeen, T, et al. **FORTLIB** writeup. Available online at MFECC from **DOCUMENT** routine.
5. Peng. M., private communication.
6. Stoer, J. and R. Bulirsch *Introduction to Numerical Analysis*, Springer-Verlag, New York, NY. 1980.
7. Dennis, J.E., D.M. Gay and R.E. Welsch, "An Adaptive Nonlinear Least-Squares Algorithm", University of Wisconsin, Mathematics Research Center Technical Summary Report 2010, 1979.

Appendix — Obtaining Solve

Solve is available on the Magnetic Fusion Energy (MFE) Network CDC7600. To get the executable code, execute **filem** as follows:

```
filem read 11027 .sol solve
```

Also available from directory `.sol` are four demonstration input files — **demo1, demo2, demo3, demo4**, which will produce the examples in Chapter 4.

EXTERNAL DISTRIBUTION

Institutions

Argonne National Laboratory
Association Euratom-CEA
  Grenoble, France
  Fontenay-aux-Roses, France
Atomics International
Austin Research Associates
Bank of Tokyo
Brookhaven National Laboratory
CNEN-Italy
College of Wiliam and Mary
Columbia University
Cornell University
  Laboratory for Plasma Studies
  Applied & Engineering Physics
Culham Laboratory
Culham Laboratory/Project JET
E G & G Idaho, Inc.
Electric Power Research Institute
Gneral Atomic Company
General Electric Company
Georgia Institute of Technology
Grumman Aerospace Corporation
Hanform Engineering Development Lab.
Hiroshima University
Japan Atomic Energy Research Institute
Kernforshungsanlage/Julich GmbH
Kyoto University
Kyushu University
Lawrence Berkeley Laboratory
Lawrence Livermore Laboratory
Los Alamos Scientific Laboratory
Max Planck Institut für Plasma Physik
McDonnel Douglas Astronautics Co.
Nagoya University
Naval Research Laboratory
New York University/Courant Institute
Nuclear Service Corporation
Oak Ridge National Laboratory
Osaka University
Physics International Group
Princeton University/Plasma Physics
Sandia Research Laboratories
Science Applications, Inc.
  Fusion Energy Development
  Lab for Applied Plasma Studies
  Plasma Research Institute
Stanford University
University of California/Berkeley
  Dept. of Electrical Engineering
  Dept. of Physics
University of California/Irvine
University of California/Los Angeles
  Dept. of Electrical Engineering
  Dept. of Physics
  Tokamak Fusion Laboratory
  School of Eng. & Applied Science
University of Maryland
  Dept. of Electrical Engineering
  Dept. of Physics
  Inst. for Physical Science & Tech.
University of Michigan
University of Rochester
University of Texas
  Dept. of Mechanical Engineering
  Dept. of Physics
University of Tokyo
University of Washington
University of Wisconsin
  Dept. of Nuclear Engineering
  Dept. of Physics
Varian Associates
Westinghouse Electric Corporation
Yale University

EXTERNAL DISTRIBUTION

Individuals

Amheard, N.
Electric Power Research Institute
Balescu, R.C.
University Libre de Bruxelles
Bartosek, V.
Nuclear Res. Inst., Czechoslovakia
Berge, G.
University of Bergen, Norway
Braams, C.M.
FOM/Inst. for Plasma Phys., Netherlands
Brunelli, B.
C.N.E.N.-Centro Frascati, Italy
Brzosko, J.S.
Inst. of Physics, Warsaw University
Cap, F.
Inst. fur Theor. Physik, Innsbruck
Conn, R.W.
Chemical Engineering, UCLA
Consoli, T.
Residence Elysee I, Claud, France
Cuperman, S.
Dept. of Physics, Tel-Aviv University
Engelhardt, W.
Max-Planck Institute für Plasmaphysik
Engelmann, F.
FOM/Inst. for Plasma Phys., Netherlands
Fiedorowicz, H.
Kaliski Inst. of Plasma Physics, Warsaw
Frolov, V.
Div. of Research & Laboratories, Vienna
Fushimi, K.
Science Council of Japan, Tokyo
Gibson, A.
JET/Culham, Abingdon, England
Goedbloed, J.P.
FOM/Inst. for Plasma Phys., Netherlands
Goldenbaum, G.
Lawrence Livermore Laboratories
Hamberger, S.M.
Australian National University
Hellberg, M.A.
University of Natal, South Africa
Hintz, E.A.K.
Kernforschungsanlage/Julich GmbH
Hirose, A.
University of Saskatchewan
Hirsch, R.
EXXON Research & Engineering Co.
Hosking, R.J.
University of Waikato, New Zealand
Ito, H.
Osaka University
Jacquinot, J.G.
CEN/Fontenay-aux-Roses, France
Jensen, V.O.
Riso National Lab, Denmark
Jones, R.
National University of Singapore
Kadomtsev, B.B.
Kurchatov Institute, Moscow
Kostka, P.
Central Res. Inst., Budapest
Kunze, H.-J.
Ruhr-Universitat, F. R. Germany
Lackner, K.
Max-Planck Inst. für Plasmaphysik
Lee, S.
University of Malay
Lenhert, B.P.
Royal Inst. of Technology, Sweden
Malo, J.O.
University of Nairobi, Kenya
Mercier, C.H.B.
C.N.E.N./Fontenay-aux-Roses, France
Nodwell, R.A.
University of British Columbia, Canada
Offenberger, A.A.
University of Alberta, Canada
Ortolani, S.
Centro di Studio/C.N.R., Italy
Palumbo, D.
Rue de la Loi, 200, Bruxelles
Pellat, R.
Centre National, Palaiseau, France
Paquette, G.
Universite de Montreal, Canada
Rabinovich, M.S.
Lebedev Institute, Moscow
Razumova, K.A.
Kurchatov Institute, Moscow
Rogister, A.
Kernforschungsanlage/Julich GmbH
Rosenau, P.
Technion, Haifa, Israel
Rosenblum, M.
Soreq Research Center, Yavne, Israel
Rudakov, L.I.
Kurchatov Institute, Moscow
Ryutov, D.D.
Nuclear Physics Instit., Novosibirsk
Salas, J.S.R.
Inst. Nacional de Investig. Nucleares
Shafranov, V.D.
Kurchatov Institute, Moscow
Smirnov, V.P.
Kurchatov Institute, Moscow
Spalding, J.-J.
Culham Laboratory, Abingdon, England
Tachon, J.
CEN/Fontenay-aux-Roses, France
Tewari, D.D.
Dept. of Physics, IIT, New Dehli
Trocheris, M.
CEN/Fontenay-aux-Roses, France
Vandenplas, P.E.
Ecole Royale Militaire, Bruxelles
Verheest, F.
Rijksuniversiteit, Gent, Belgium
Watson-Munro, C.N.
University of Sydney, Australia
Wesson, J.A.
Culham Laboratory, Abindgon, England
Wilhelm, R.
Inst. für Plasmaphysik, Stuttgart
Wilhelmsson, K.H.B.
Chalmers Univ. of Technology, Sweden
Wobig, H.
Max-Planck Inst. für Plasmaphysik