A SOLVER FOR THE GENERAL PARTICLE TRACER PACKAGE

S.B. van der Geer, M.J. de Loos, Pulsar Physics,
De Bongerd 23, 3762 XA Soest, The Netherlands

Abstract

The General Particle Tracer (GPT) code has been extended with a multi-dimensional optimizer and solver to automate the final stages of a design process. The new solver can be used for all GPT simulations, including 3D space-charge and particle-wave interaction. The internal algorithms and two examples are presented in this paper.

1 INTRODUCTION

GPT has become a well-established platform for the design of accelerators and beam lines worldwide [1,2]. Starting from the first release, GPT has been capable of scanning any parameter and plotting the results. However, because an actual design typically involves many parameters and simultaneous constraints, it can be quite time consuming to find a solution using the scan utility. To further assist in the design process, a new solver has been created.

GDFsolve is a multi-dimensional root-finder and optimizer that can be used as a "driver-program" for all GPT simulations. It tries to solve or optimize any number of constraints on beam parameters by varying variables used in the GPT inputfile. The beam parameters are calculated by standard GPT data-analysis, allowing both built-in and custom analysis routines. The described algorithms allow a non-equal number of variables and constraints as well as external boundary conditions for all variables.

2 THE ROOT-FINDER

GDFsolve used as root finder tries to find a simultaneous solution to the following set of non-linear equations:

$$\begin{aligned} f_1(x_1,x_2,\ldots) - ft_t &= 0 \\ f_2(x_1,x_2,\ldots) - ft_2 &= 0 \\ &\vdots \end{aligned} \qquad [1]$$

Or in vector notation:

$$\mathbf{f}(\mathbf{x}) - \mathbf{ft} = \mathbf{0} \qquad [2]$$

where **f** is a function of variables **x** and **ft** is the target value. When the dimensions of **f** and **x** are equal, a relatively simple solution can be obtained by a first order estimate for a new trial $\mathbf{x}_{n+1}$ based on previous guess $\mathbf{x}_n$:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{M}^{-1}(\mathbf{ft} - \mathbf{f}(\mathbf{x}_n)) \text{ where } M_{ij} = \frac{d\, f_i}{d\, x_j} \qquad [3]$$

Iterating the above procedure is known as multidimensional Newton-Raphson [3] and works very well in the vicinity of a root. In any number of dimensions the derivative of the function is extrapolated to produce the next trial as shown in Figure 1 for 1D.

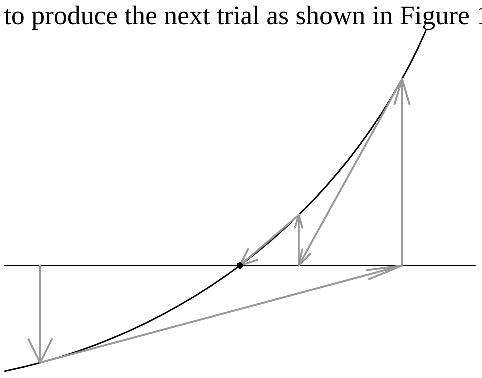

Figure 1: Typical 1D Newton-Raphson iterations.

There are a number of problems with this approach:

- Typical variables *x* can have very different scaling.
- The Jacobian matrix **M** can not be evaluated directly and must be estimated numerically by a finite difference approach $M_{ij} \approx \Delta f_i / \Delta x_j$.
- In a large number of situations basic Newton-Raphson sends the solution to outer space or does not converge at all.
- Not all matrices **M** can be inverted. This is obvious when the number of variables and the number of constraints are not equal. The same problem arises when one variable does not have an effect on **f**, or when one constraint is not affected by changing **x**.
- Some variables are bounded by external constraints such as existing hardware, budget restrictions or the location other beam line components.
- The number of function evaluations required to obtain **M** is equal to the number of free parameters.

Solutions to these problems are presented in the following subsections.

2.1 Scaling and initial stepsizes

To avoid large truncation errors when the matrix *d***f**/*d***x** is inverted, it is made dimensionless by dividing both **f** and **x** by a typical set of scale-factors: **df** and **dx** respectively. To simplify the equations we directly subtract the target value **ft** from **f** such that the new function **F** must be zero at the solution, leading to the following set of equations:

$$\mathbf{F}(\mathbf{x}) = (\mathbf{f}(\mathbf{x}) - \mathbf{ft}) / \mathbf{df}$$
$$\mathbf{X} = \mathbf{x}/\mathbf{dx} \quad [4]$$
$$\mathbf{M} = \Delta\mathbf{F}/\Delta\mathbf{X} = \mathbf{F}(\mathbf{x}_N + \mathbf{dx}) - \mathbf{F}(\mathbf{x}_N)$$
$$\mathbf{x}_{N+1} = \mathbf{x}_N - (\mathbf{M})^{-1}\mathbf{F}(\mathbf{x}_0)\,\mathbf{dx}$$

When the variables are properly scaled, the finite difference step to obtain **M** can be chosen as Δ**X**=1.

2.2 Backtracking

One of the problems with the proposed scheme is that it can send a solution to infinity when the first estimate of **x** is not sufficiently near a root in **F**. A related problem can occur when the sequence simply does not converge. Proper convergence is monitored by testing a reduction in the length of **F** after every step:

$$|\mathbf{F}(\mathbf{x}_{n+1})| \geq (1-\alpha)\,|\mathbf{F}(\mathbf{x}_n)| \quad [5]$$

When [5] is not satisfied with a typical α of a few percent this indicates that the used step is too large. The stepsize is reduced by a factor of 2, for three iterations if necessary, in a final attempt to find a smaller **F**.

2.3 Singular Value Decomposition

Instead of inverting the Jacobian **M**, Singular Value Decomposition is used to write the matrix as the product of three matrices:

$$\mathbf{M} = \mathbf{U}\cdot\operatorname{diag}(w1, w2, \ldots)\cdot\mathbf{V}^T \quad [6]$$

where the columns of **V** define an orthonormal set of directions of the variables $\mathbf{X}_i$ and the columns of **U** define the corresponding change in the constraints $\mathbf{F}_i$. The diagonal matrix containing w_i, the singular values, defines the scaling between these two. Once the Singular Value Decomposition is calculated, inverting **M** is simple:

$$\mathbf{M}^{-1} = \mathbf{V}\cdot\operatorname{diag}(1/w_1, 1/w_2, \ldots)\cdot\mathbf{U}^T \quad [7]$$

A relatively small w_j indicates a change in **F** that requires a very large change in **X**. Typically, such large steps lead you away from the solution rather than put you on top of it. Therefore the pragmatic approach is simple: Because you can hardly change **F** in a direction of small w_i, typically between 10^{-2} and 10^{-6}, **X** is kept constant in the corresponding direction. An additional advantage of the SVD algorithm is that the direction information can be used in a clear diagnostic message to the user.

When **M** is not square, in the case of over- and under-constrained systems, the matrix **U** is not square but the algorithm still functions properly.

2.4 External boundary conditions

In many design scenario's, the variables x_j can not be chosen freely. They are restricted by boundary conditions such as the location of other beam line components. When a new trial value $\mathbf{x}_{n+1}$ lies outside the hypercube of boundary conditions, an attempt is made to move it inside by changing $\mathbf{x}_{n+1}$ in the nullspace of **M,** see Figure 2:

$$\mathbf{x} \leftarrow \mathbf{x} + \mathbf{V}'\lambda \quad [8]$$

where only the columns of **V** corresponding to small w_i are used in **V'**. The additional unknown vector λ is solved from: $\mathbf{xb} = \mathbf{x} + \mathbf{V}''\lambda$ where **V''** contains only the rows of **V'** corresponding to the component of *x* that must be moved into the boundary hypercube, given by a corresponding *xb*.

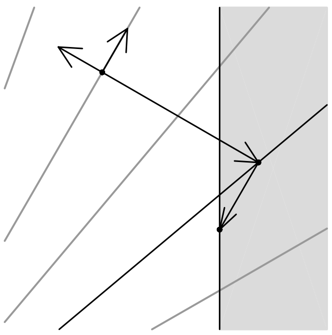

Figure 2: A boundary condition is solved by moving the new estimate in the nullspace of **M**.

2.5 Broydens method

A new estimate of **M** can be calculated from the function information from the previously successful step using the following Broyden [3] update equation:

$$\mathbf{M}_{N+1} \approx \mathbf{M}_N + \frac{(\delta\mathbf{F}_i - \mathbf{M}_i\cdot\delta\mathbf{X}_i)\otimes\delta\mathbf{X}_i}{\delta\mathbf{X}_i\cdot\delta\mathbf{X}_i} \quad [9]$$

where δ**F** is the difference in **F** in a step with size δ**X**.

When the method fails to produce a good representation of the actual Jacobian the backtracking algorithm will not find a solution and GDFsolve reinitializes **M** by calculating a full Jacobian again.

3 THE OPTIMIZER

GDFsolve as optimizer tries to find the minimum of any function *g*(**x**) by varying all components of **x**. Our implementation is very close to the Powell implementation [3] with the following modifications: Function evaluations with identical parameters are not repeated, one-dimensional optimization properly generalizes to a single line-minimization and relative termination detection is changed to an absolute value.

Qualitatively, the algorithm is as follows. The first steps find the minimum in the direction of the first component of **x**. Starting from there, the second component of **x** is varied until a minimum is found. This process is repeated as many times as there are dimensions in **x**. To improve the efficiency of the algorithm, the average direction resulting from these iterations is also used as minimization direction, replacing the direction of the largest function decrease. The complete process is iterated until a stable solution is found. The line-minimization routine takes larger and larger steps downhill until a minimum is bracketed. The actual minimum is found by either parabolic interpolation or golden section search.

Because not all parameters x_i can always be chosen freely all variables are bounded by a minimum and maximum value.

4 APPLICATIONS

Both the root-finder and optimizer have been used extensively for various design problems. A simple example and a difficult optimization problem are described below.

4.1 *Simple example*

This example simulates the focusing of a 50 MeV parallel beam using two quadrupole lenses as shown in Figure 3. GDFsolve as optimizer has been used to find the two required quadrupole strengths by minimizing the beam-size in both the horizontal and vertical plane.

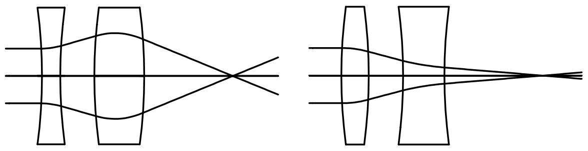
Figure 3: Horizontal and vertical plane of a parallel 50 MeV beam focused using two quadrupole lenses.

Alternatively a zero Courant-Snyder alpha parameter in both transverse planes, no divergence, can be used as constraint for the root-finder. The final results are identical, but as shown in Table 1 the required number of iterations is very different: The root-finder is far more efficient compared to the optimizer because only order N iteration are required compared to N^2 for the optimizer.

Table 1: Required number of GPT runs to find the quadrupole strengths.

	Root finder	Root with Broyden	Optimizer
2 Variables	7	5	49
3 Variables	9	6	57

When the second quadrupole strength is defined as the sum of two variables, there is an additional degree of freedom resulting in a singular-value in the SVD algorithm. The root-finder with Broyden still finds the correct solution with a minimum of additional GPT runs.

4.2 *FOM-Rijnhuizen FEM*

The Fusion-FEM is the prototype of a high power, electrostatic mm-wave source, tunable in the range 130-260 GHz [4]. The device is driven by a 2 MeV, 12 A dc electron beam and is designed to generate 1 MW microwave power. The part of the system described here consists of an 80 keV thermionic gun, a 2 MV dc electrostatic accelerator, and a step tapered undulator.

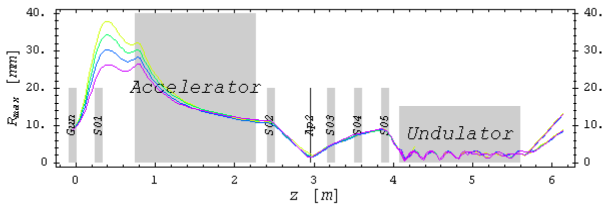

Figure 4: GPT Simulation of the maximum beam radius in the first part of the Rijnhuizen FEM for 6 A (red) to 12 A (yellow) with the settings of Table 2.

During initial experiments a much lower current than 12 A is typically used, resulting in different settings for the first five focusing solenoids. GDFsolve has been used [5] to find these settings automatically, taking all non-linear space-charge forces and particle-wave interaction in the undulators into account. First the beam-entrance criteria for best beam transport in the undulators are obtained by minimizing beam radius and divergence in the undulators. The thus obtained beam conditions and a number of intermediate sizes are subsequently used as constraints for the root-finder to solve for the currents through the first five solenoids, see Table 2. The resulting maximum beam radius is shown in Figure 4.

Table 2: Settings for the first five solenoid of the Rijnhuizen-FEM as function of beam current.

	6A	8A	10A	12A	
Is01	8.40	8.73	8.91	9.02	A
Is02	13.69	13.57	13.26	13.11	A
Is03	14.55	14.41	14.86	16.74	A
Is04	8.02	8.09	8.68	9.57	A
Is05	14.06	14.43	15.83	15.96	A

5 CONCLUSION

GDFsolve is a valuable tool to automate GPT simulations in the final stages of a design. It efficiently solves non-linear multi-dimensional optimization and root-finding problems, while properly taking care of over- and under-constrained systems and boundaries.

The current status of the GPT project can be found on the web at *www.pulsar.nl/gpt*

6 REFERENCES

[1] S.B. van der Geer, M.J. de Loos, Proc. 1998 Eur. Part. Acc. Conf., Stockholm, (1998) pp. 1245.

[2] GPT User Manual, Pulsar Physics, Flamingostraat 24, 3582 SX Utrecht, The Netherland www.pulsar.nl

[3] W.H. Press, et al., Numerical Recipes in C, Cambridge Univ. Press, 2nd edition, (1992).

[4] W.H. Urbanus, et al, Nucl. Instr. and Meth. A375, (1996) p. 401.

[5] C.A.J. van der Geer, Optimum Transport in FEM-III, (2000).