Numerical Optimization

Contents

1 Principle

The combination of Monte Carlo simulations and optimization is realized in a way that the optimization routine is the main program and calls the simulation in a subroutine called `ExtFunctions()` (see Fig. 1)[1].

This procedure first writes a file *Pcomm.dat* containing one or more parameter sets. The tool `gener_pipe.exe`[2] reads these sets and combines them with the standard instrument (saved as *std_instr.cmd*), thus generating the VITESS pipe commands (in a shell or batch script) for all simulations to perform.

The simulations are then started and the files (of all simulations) that are needed for the figure of merit are copied. The next step is to start a program to calculate the figures-of-merit (for all simulations) and write them into a file *Fcomm.dat*. Now this/these figure(s)-of-merit are read and returned to the optimization program. This will suggest (a) new parameter set(s) and call `ExtFunctions` again. This procedure continues until the optimization has come to an end.

The optimization is widely controlled by the 3 ASCII input files ***fom.ini***, ***opt_param.ini*** and ***sim_param.ini*** plus an ASCII control file for the optimization algorithm used (e.g. ***swarm.ini***). If there is a one-to-one relation between optimization and simulation parameters and the figure of merit is one of the many options of the delivered `fom` program, no code or script is necessary for the optimization.

The optimization routines can also be used for fitting (see section 6).

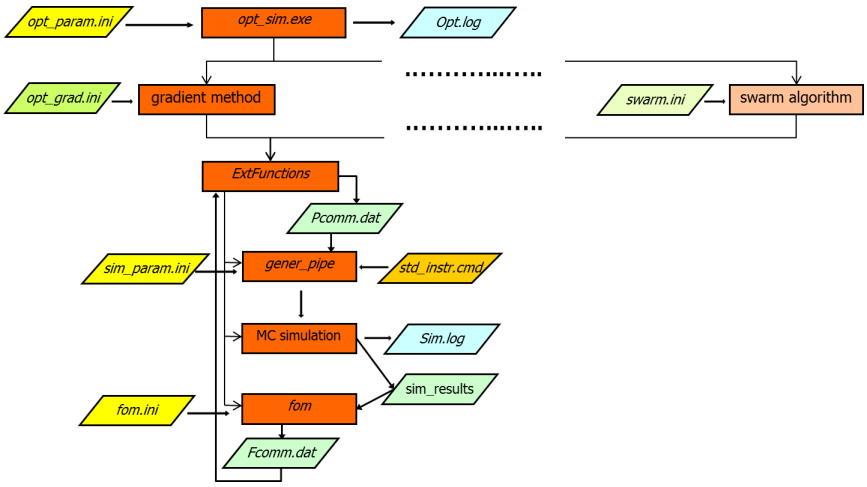


Figure 1: Scheme of optimization

[1] In the code, `ExtFunctions` has been renamed in VITESS version 3.2 with introduction of the cluster parallelization to `OptFctPc` and `OptFctGrid`.

[2] Windows: `gener_pipe.exe`, Linux (64 bit): `gener_pipe_Linux_x86_64`, Macintosh (64 bit): `gener_pipe_Darwin_x86_64`

2 Directories

The directory "...VitessX.Y/MODULES" where the executables are will be called the *module directory*. The directory "...VitessX.Y/OPTIMIZATON" contains all necessary input files for the optimization.

You can run the optimization here or choose a different folder. This will be called *optimization directory*.

The simulations generally run in a different directory. This can be the directory where the instrument has been developed or another one. This will be called *simulation directory*.

3 7 Steps to the Optimization

3.1 Step 1: Necessary Files

The executable `opt_sim` has be copied from the module directory to the optimization directory. If an optimization directory different from ...VitessX.Y/OPTIMIZATON is chosen, all *.ini* files have to be copied from there to optimization directory. If you choose a simulation directory that is not the original directory, you have to copy **all input files** there.

3.2 Step 2: Standard Instrument

The first step is to develop an instrument that can be used as a starting point for the optimization. Make sure that the simulation runs properly and that an adequate number of trajectories is chosen; at least 100000 trajectories should contribute to the figure of merit, i.e. have to be summed up in the file used to calculate the figure of merit.

When it is finished it needs to be saved by clicking "Check" and saving the shown output **without any change** by using the "Save" button (belonging to the output window) as *std_instr.cmd* to the **optimization directory (!!!)**. Note that this command file cannot be used for an optimization on another platform (e.g. saved under Windows and used under Unix), and that it has to be **left in this format!!!** (Of course editing this file, e.g. changing the number of trajectories, is possible.)

Now the different *.ini*-files have to be filled. Note that anything behind a #-symbol is regarded as a comment and ignored by the program.

3.3 Step 3: Optimization Parameters

In the file *opt_param.ini* you have to give

1. the name of the optimization

2. a code word to define the optimization algorithm
   existing at the moment : "opt_grad", "opt_grad_mc", "metropolis" , "swarm" (see section 4 "Algorithms")

3. option for local ("sim_opt_pc") or cluster ("sim_opt_grid") usage. See section 5 for optimization on a cluster.

4. name of figure of merit binary

5. cluster options; set "none" for local usage

6. all input variables in the form

   - index (starting with 1)
   - initial value
   - minimal value
   - maximal value
   - difference to the actual value for the calculation of the gradient or the next tested value

Notes:

◇ Not all algorithms will need all information about the parameters or can treat them, but for the sake of simplicity this needs to be given for all of them.

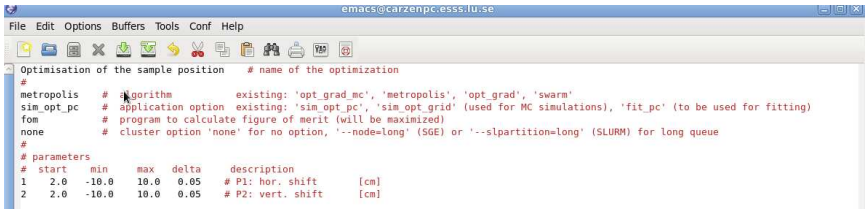
◇ Input lines must not be removed and their order has to be kept.

◇ For gradient methods, the difference "delta" should be only a fraction of the expected value to calculate the differential quotient (for the actual value). For other methods this limits the step width to the next position and should therefore be in the order of magnitude of the value or a bit smaller.

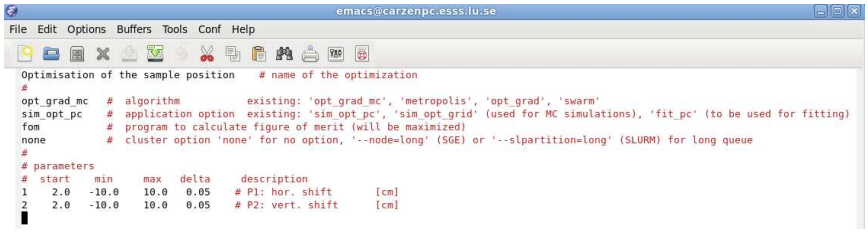
3.4 Step 4: Control of the Simulation

The first two lines in *sim_param.ini* define the directory where the executables can be found (and thus the Vitess version) - line 1 - and the simulation directory (cf. section 2 "Directories") - line 2 (see example files in figure 3). STD means that the path to this directory is taken from the standard instrument file *std_instr.cmd*. The third line defines the file(s) necessary to calculate the figure of merit (see example files in figure 4, cf. Step 5).

The next line of this file defines the function calculating the simulation parameters S_k from the optimization parameters P_j. STD defines a direct assignment: $S_1 = P_1$, $S_2 = P_2$, etc. In the future, simple mathematical

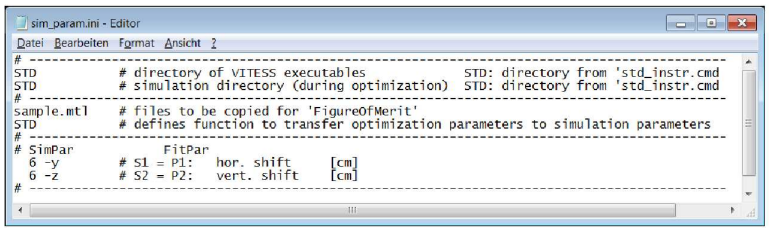


(a) example using `metropolis` algorithm

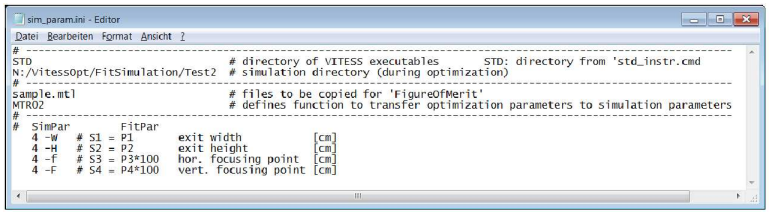


(b) example using `opt_grad_mc` algorithm

Figure 2: Examples of file *opt_param.ini* setting optimization algorithm and parameters to be optimized



(a) example using default settings



(b) example using the function `DetSimParamMtr` and a different simulation directory

Figure 3: Examples of file *sim_param.ini* defining the variable simulation parameters and the function transferring the optimization parameters to simulation parameters

operations between these parameters can be defined in the way indicated in the file. For the time being, only this direct assignment is possible, everything else has to be realized by extending the program `gener_pipe` - see below.

In the next lines, all simulation parameters affected by the optimization parameters are listed (see files in figure 3). They are characterized by the module number and the symbol (like "-H") defining the parameter (within the list of existing parameters of this module). The symbol is shown if you click on the text belonging to this item.

In order to calculate simulation parameters from optimization parameters, a function similar to `DetSimParamStd()` has to be written in *gener_pipe.c*[3], which performs this calculation. The easiest and safest way is the following: copy, rename and change `DetSimParamStd()`. Then add 3 lines of code in the if...else if...else command identifying this function by code that will then be given in *sim_param.ini* instead of STD.

Example: Adding

```
else if (strcmp(sParFct, "MTR02")==0)
{ DetSimParamMtr(P, m, nSimParNo, nFitParNo, 2);
}
```

assigns that the function `DetSimParamMtr()` with additional parameter *iFirstMPar=2* is called. It transfers the optimization parameters to simulation parameters, if the code "MTR02" is given in the fourth line of *sim_param.ini*. (This has in fact been realized for the optimization of the elliptical guide exit of the EXED instrument, cf. file in figure 3(b). MTR denotes the change from meter to cm and 02 indicates for which simulation parameters this is applied.)

3.5 Step 5: The Figure of Merit

The next step is to assess the figure of merit. The function `ExtFunctions()` in *calc_sim_fom.c* will call the program `fom`; this writes the file `Fcomm.dat`. It is an ASCII file containing just one number for each simulation to be executed. In case of N simulations, these N numbers are written in N rows. The files defined in `sim_param.ini` - in this example *sample.mtl* - serve as a data basis for these calculations.

In the future, using user defined programs will be made easier. In the present version, it has to be called `fom` (plus extension) and be written to the folder MODULES replacing the existing program, or the name in `ExtFunctions()` has to be changed and *opt_sim* re-compiled.

`fom` is delivered with the package. It is controlled via *fom.ini*. The general formulae are

[3]The file *gener_pipe.c* is in the SRC directory. Don't forget to recompile after adding your function. Copy the new executable to the MODULES directory and add the appropriate extension (.exe or _Linux_x86_64 etc).

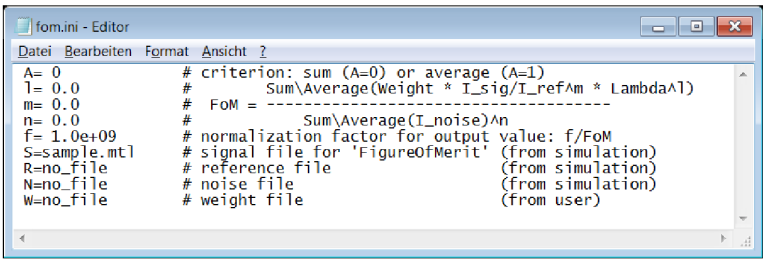


Figure 4: File *fom.ini* to define the figure of merit

$$FoM = \frac{\sum\limits_i \frac{w_i I_{sign,i} \lambda_i^l}{I_{ref,i}^m}}{\sum\limits_i I_{noise,i}^n}$$

or

$$FoM = \frac{\left\langle \frac{w_i I_{sign,i} \lambda_i^l}{I_{ref,i}^m} \right\rangle}{\left\langle I_{noise,i}^n \right\rangle}$$

where the parameter A defines if sums (A=0) or averages (A=1) are used. The numbers for the powers l,m,n have to be given; they can be float values. The files containing *signal*, *noise*, *reference* and *weight* have to be given. If they are output files of the simulations, they have to be listed in *sim.ini* as well. If a weighting with wavelength shall be used, the intensity files have to be intensities as a function of wavelength.

As the optimization routine searches for the minimum, the value f/FoM is written to *Pcomm.dat*. The factor f is also read from *fom.ini* and allows setting output values to the order of magnitude 1, which is an advantage (in terms of numerical precision) for some optimization algorithms.

In the example shown in figure 4, the figure of merit is reduced to

$$FoM = \sum I_{sign,i}$$

thus optimizing the integrated flux (in a slit).

Example: If the transmission averaged over a wavelength range shall be optimized, the parameters have to be set as: A=1; m=1; l=n=0 and the reference file (=input of the section, whose transmission is regarded) has to be given in addition to the signal file.

3.6 Step 6: Control of the optimization routine

The optimization process is controlled by the file *opt_name.ini*, i.e. each algorithm has a control file of its own. Useful default parameters are de-

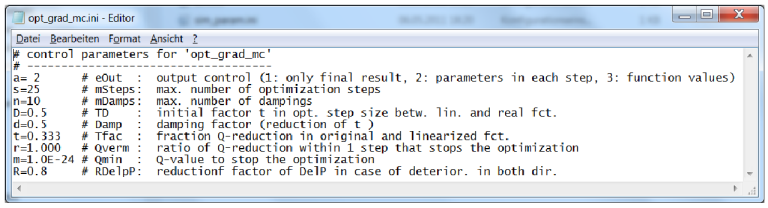


Figure 5: File *opt_grad_mc.ini* controlling a gradient method adapted to Monte Carlo simulations

livered so that often no changes are necessary for the two gradient and the `metropolis` algorithms. When using the `swarm` algorithm, adjusting these parameters to the optimization problem at hand is advisable. The algorithms and their respective steering parameters aqre described in detail in section 4.

3.7 Step 7: Run

Now the optimization can be run by starting the program `opt_sim` in the optimization directory.

3.8 Output

A log file `Opt.log` of the optimization is written to the optimization directory. It contains the initial and final parameter values and (depending on the algorithm) all or some of the parameters sets in between as well as the figures of merit for these parameter sets. Additionally, log files *Sim0.log*, *Sim1.log*... for the different simulations of the last optimization step can be found in the simulation directory. Finally, the files contributing to the figure of merit are copied. So those files obtained in the final optimization step still exist.

4 Algorithms

In VITESS version 3.1, three algorithms are available, others follow in upcoming versions: a usual and an adapted gradient method as well as a metropolis algorithm. In version 3.2, a swarm algorithm has been added.

The usual gradient method chosen by `opt_grad` was developed for fitting and works there very well in practically all cases. It has been changed to allow for the statistical effects in combination with Monte Carlo simulations. This adapted gradient method is used when choosing `opt_grad_mc`. Both gradient methods work best if the parameters are all in a similar order of magnitude and especially not larger than 10^6. The gradient is determined

numerically varying one parameter in both direction and calculating the differential quotient. Care should be taken in defining the parameter variation: it should not be too large to get a good estimation of the gradient for this parameter value, but especially not too small to avoid zero difference and a statistically induced false estimation. `opt_grad_mc` allows setting minimal and maximal value for a parameter, but the optimization runs better if this option is not used. Generally, gradient methods work well with Monte Carlo simulations, if a sufficient number of events - ideally up to 10^6 - contribute to the figure of merit.

The Metropolis algorithm is used if `metropolis` is chosen. The risk of finding a local instead the global optimum is reduced by using the metropolis algorithm; but it needs by far more simulations, at least 1000 steps. On the other hand, it allows getting an estimate of the width of the local optimum by calculating centre and variance over the path in the local optimum.

A particle swarm optimization is used if `swarm` is chosen in *opt_param.ini*. In this case, more than one simulation is run for different sets of trial parameters in each optimization step (the number is set in *swarm.ini*, see section `swarm` algorithm), so setting a good starting point is not important. No gradient is used, so the optimization problem does not have to be differentiable but there is also no guarantee that an optimal solution is found.

All optimization algorithms implemented in Vitess 3.2 are described in more detail in the following.

4.1 Gradient methods: `opt_grad` and `opt_grad_mc`

The gradient methods use the partial derivatives to determine the direction towards the expected optimum: The Matrix $\mathbf{N}$ of partial derivatives at position $\vec{P0} = (P0_1, ..., P0_n)$ is calculated as

$$N_{i,j} = \frac{\partial f}{\partial P_i}\frac{\partial f}{\partial P_j} \text{ , with } f = FoM^{-1} \text{ .} \tag{1}$$

The figure of merit FoM is delivered by the executable `fom.exe` by evaluating output data of the MC simulation.

The Matrix $\mathbf{N}$ is calculated by numerical variation of the parameter set $\vec{P}$, i.e. the function f is calculated for $(p_1 + \delta p_1, p_2, p_3, ...p_n)$, $(p_1 - \delta p_1, p_2, ...p_n)$, $(p_1, p_2 + \delta p_2, ...p_n)$ .... $(p_1, p_2, p_n x - \delta p_n)$. Values for δp_i are taken from *opt_param.ini*.

For a linear function, the vector to the optimum can be directly calculated as

$$\Delta\vec{P} = \mathbf{N}^{-1}\vec{r} \tag{2}$$

with

$$\vec{r} = -\frac{1}{2}\nabla Q \text{ ,}$$

where Q is the so-called quality factor defined as $Q := 1/FoM^2$.

The new parameter set $\vec{P}1 = \vec{P}0 + \Delta\vec{P}$ is tested for the (usually nonlinear) function under consideration. It is accepted, if the improvement in the quality factor Q

$$\Delta Q = Q(\vec{P}1) - Q(\vec{P}0)$$

is at least a fraction '$Tfac$' (control parameter 't') of the value ΔQ_{lin} calculated for the linear function:

$$|\Delta Q| < T_{fac} \cdot |\Delta Q_{lin}| \ .$$

If this is not the case, the direction of change from $\vec{P}0$ is kept, but the step width is reduced by a factor '$Damp$' (control parameter 'd'), i.e.

$$\vec{P} = \vec{P}0 + Damp \cdot \Delta\vec{P}$$

and the comparison between Q of linear and real function is performed again. This is repeated in a loop until the criterion is fulfilled. This procedure avoids jumping back and forth over the optimum [1]. Values of T_{fac} between 0.3 and 0.5 are recommended, lower values allow this jumping, higher values restrict the step size. Finding a new value for $\vec{P}$ is regarded as one step. The algorithm searches for a minimum of Q.

The optimization stops if

- Q is below '$Qmin$' (control parameter 'm')
- the improvement in one step is less than given by '$Qverm$' (control parameter 'r')
- the maximal number of dampings '$mDamps$' has been executed (control parameter 'n')
- the maximal number of optimization steps '$mStep$' has been executed (control parameter 's')

The parameter 'a' controls the output:

- a=1 means that only the parameter set of the final result is written.
- with a=2 (default) the actual parameter set is written in every step.
- with a=3 even more information is written to the output file, which might help to find bugs if the optimization does not work.

As can be seen from equation (1) and (2), the algorithm includes a matrix inversion, which fails if

- the dependence on one parameter is zero

- values of $\partial f/\partial P_j$ are many orders of magnitude from 1

Generally, the optimization works best, if all values of P_j and $\partial f/\partial P_j$ are close to the order of magnitude 1.

The usual gradient method selected by 'opt_grad' does not consider the parameter range given in *opt_param.ini*. In contrast, the algorithm selected by 'opt_grad_mc' does, but optimization runs better if it is not used.

The former one is meant for fitting (see section 6), the latter one for combination with Monte Carlo (MC) simulations. The statistical nature of these simulations creates a rough surface, which complicates optimization: A parameter set that is closer to the real optimum may yield a worse figure of merit, because accidentally less 'good events' are chosen. The effect is diminished by chosing a large number of events - ideally 1 million events should contribute to the figure of merit - but as simulations often take much time, there is a limit to increassing the number of events. Therefore, '`opt_grad_mc` contains some features to adapt the method to the conditions found in MC simulations:

- As the step width is often overestimated, it can be reduced by a factor '$TD0$' already before the first test (control parameter 'D')
- if $(p_1, p_i - \delta p_i, ...p_n)$ and $(p_1, p_i + \delta p_i, ...p_n)$ do not give the same information about the gradient, this parameter remains unchanged
- if the variation of parameter i leads to a worse result in both directions, the variation δp_i is reduced by a factor '$RDelP$' (control parameter 'R')

4.2 Metropolis algorithm

The `metropolis` algorithm is a random walk through parameter space to find the optimal parameter set [2]. The step $\Delta\vec{P}$ from the actual position $\vec{P}0$ to the next possible position

$$\vec{P} = \vec{P}0 + \Delta\vec{P}$$

is determined by a Monte Carlo choice, where one parameter is changed. The range in which $\vec{P}$ is varied is defined by the values '*delta*' (for each parameter) taken from *opt_param.ini*:

$$-\delta_j < \Delta P_j < \delta_j \ .$$

$\vec{P}$ is accepted as the new position if it is an improvement in the quality factor Q, i.e. if

$$Q(\vec{P}) < Q(\vec{P}0)$$

but there is also a chance that it is accepted in case of worsening. The probability is

$$\begin{aligned} p &= e^{\left(-\frac{\chi^2-\chi_0^2}{2}\right)} \\ &= e^{\left(-\frac{Q^2-Q_0^2}{\sigma^2}\right)} \qquad \text{with } Q := Q(\vec{P}),\ Q_0 := Q(\vec{P}0) \end{aligned}$$

This gives the chance to traverse a local minimum and and find the absolute minimum. Q is defined as $Q := 1/FoM^2$, and FoM is delivered by the executable `fom.exe` by evaluating output data of the MC simulation. σ has to be chosen properly to get a realistic probability to get out of a local minimum.

The parameter 'n' gives the number of new positions tested in 1 step. The description above described the default case $n = 1$. If a number larger than 1 is chosen, the best of all new possible positions is chosen as the possible position. This option was added to take advantage of parallel computing, but it alters the optimizations process.

The parameter range can be restricted (control parameter $c = 1$). The range of each parameter has to be defined in *opt_param.ini*. For $c = 0$, these values are ignored.

Each attempt to find a new parameter set $\vec{P}$ is one step. The optimization stops after '$mSteps$' steps (control parameter 's') or if Q is below '$Qmin$' (control parameter 'm').

The actual and the best parameter are saved. The best parameter set $\vec{P_{best}}$ is updated, if the error square sum of the new parameter set is less than '$Qverm$' (control parameter 'r') of the previously best.

Apart from searching for the best parameter setting, it is also possible to obtain average values of the random around a minimum. When the Q-value falls below '$Qlimit$' (defined by control parameter 'l'), the parameter averages are calculated as a weighted average using the weight

$$p_i = Q_{limit} - Q_i$$

until Q exceeds '$Qlimit$' again.

The parameter 'a' controls the output:

- a=1 means that only the parameter set of the final result is written.
- With a=2 (default) the actual parameter set is written every '$nStpOut$' steps (given by control parameter 'o') additionally the best parameter found so far is written every '$nStpMin$' steps (given by control parameter 'd')
- Using a=3 even more information is written to the output file, which might help to find bugs if the optimization does not work.

4.3 Swarm algorithm

The particle swarm optimization (PSO) method is an attemp to model swarm intelligence: a population of individuals searches the parameter space such that the movement of each individual is influenced by both the personal best value found so far (*local* best, *cognitive* component) as well as by the best value found by the entire swarm (*global* best, *social* component). This movement is described by a "velocity" v specifying the change of a parameter p between two optimization steps i and $i+1$:

$$v_{i+1} = \omega_0 v_i + \omega_1 r_1 (p_{localBest} - p_i) + \omega_2 r_2 (p_{globalBest} - p_i) \quad (3)$$

$$p_{i+1} = p_i + v_{i+1} \quad (4)$$

where ω_0 is known as inertia weight, $\omega_{1,2}$ as acceleration coefficients and $r_{1,2}$ are random numbers in the range [0,1]. Typical values are $\omega_0 = 0.9$ and $\omega_1 = \omega_2 = 2$.

The implemetation used in VITESS follows the *time-varying acceleration coefficients* method (TVAC) described in [3]. The weights $w_{0,1,2}$ of the three terms in (3) are linearly varied during the optimization process to ensure sufficient exploration of the parameters space in the beginning of the

optimization as well as a sufficiently fast convergence towards the end:

$$\omega_{0,i} = \left(\omega_0^{max} - \omega_0^{min}\right) \frac{N_{it}^{max} - i}{N_{it}^{max}} + \omega_0^{min} \quad (5)$$

$$\omega_{1,i} = \left(\omega_1^{min} - \omega_1^{max}\right) \frac{i}{N_{it}^{max}} + \omega_1^{max} \quad (6)$$

$$\omega_{2,i} = \left(\omega_2^{max} - \omega_2^{min}\right) \frac{i}{N_{it}^{max}} + \omega_2^{min} \ , \quad (7)$$

where i is the current iteration and N_{it}^{max} the maximal number of optimization steps (input variable *nSteps*, see below). The default values used in VITESS for minimal and maximal coefficients are those which have been reported as optimal in the literature ([3] and references therein):
$\omega_0^{min,max}$=(0.4,0.9), $\omega_1^{min,max}$=(0.5,2.5), $\omega_2^{min,max}$=(0.5,2.5).
This time dependence can of course be switched off by simply setting $w_{0,1,2}^{min} = w_{0,1,2}^{max}$.

The starting values of the parameters and velocities are set by random numbers within the parameter range and step size defined by the user (in *opt_param.ini*):

$$p_0 = random[p_{min}, p_{max}]$$
$$v_0 = random[-\Delta p, \Delta p]$$

The starting parameters of one swarm individual are set to the user-defined starting values.

During optimization, the maximum possible verlocity is set to $|v_{i,max}| = 0.5|p_{max} - p_{min}|$. If the calculated velocity or parameter position exceeds its limit, it is set to $\pm v_{max}$ or $p_{min,max}$, respectively.

The optimization ends if either the maximum number of iterations is reached, or the figure of merit is not being improved for a certain number of iterations.

The PSO method is thus steered by the following parameters (in *swarm.ini*):

nBees The number of swarm individuals. Note that in each optimization step, *nBees* simulations are started.

nSteps The maximal number of optimization steps (N_{it}^{max} in eqn. (5)-(7)). Note that a large *nSteps* leads to a slow change of the weights $\omega_{0,1,2}$. If *nSteps* is chosen too small (using the default min./max. weights), the swarm will quickly cluster around the global best value and might get trapped in a local optimum.

nFinish The maximum number of optimization steps without further improvement.

w0_min Minimal value of the inertia weight.

w0_max Maximal value of the inertia weight.

w1_min Minimal value of the local weight

w1_max Maximal value of the local weight

w2_min Minimal value of the global weight

w2_max Maximal value of the global weight

eOut Output level:
(1) standard output,
(2) each new global best parameter set written to log-file, additional file *Swarm_GlobalBestValues.dat* written containing a list of global best values in each optimization step (iteration, fitness, p1...pN)
(3) DEBUG mode: position and velocity of every individual at every optimization step written to log file, additional file *Swarm_BeeMovement_Parameter1.dat* containing values of first optimization parameter for each swarm individual (columns) in each optimization step (rows)

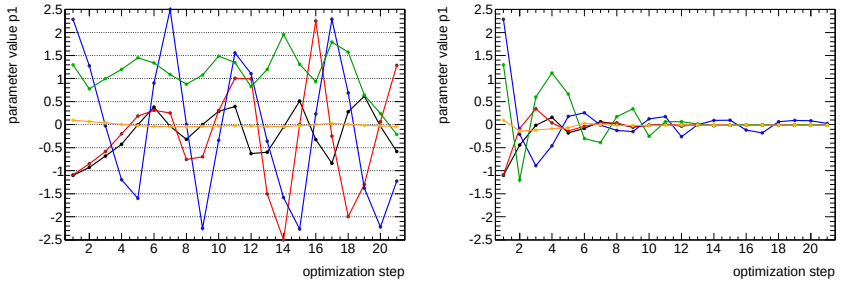
The relative size of the weight parameters determines how quickly the swarm converges towards the global maximum value[4]. The default values are chosen such that exploration of the parameter space is dominant in the beginning of the optimization and convergence to the global best value is enhanced towards the end of the optimization.

The effects of these parameters are illustrated in figures 6 and 7, which show the movement of 5 swarm individuals in one parameter dimension during the optimization. The optimization example used here is the simple case of finding the optimal position of a pinhole in the focal plane of an elliptic guide. Obviously, the solution is the focal point, which is at (y,z)=(0,0). Only the y-position is shown in the plots.

Figure 6 compares the swarm behaviour for weights fixed (i.e. $w_{min} = w_{max}$) at their initial (a) and final (b) default values. In the first case, swarm individuals are seen to oscillate around the individual best values with a large amplitude, covering the whole parameter range. In the latter case in which the global weight is largest, the oscillation is damped and the swarm converges quickly.

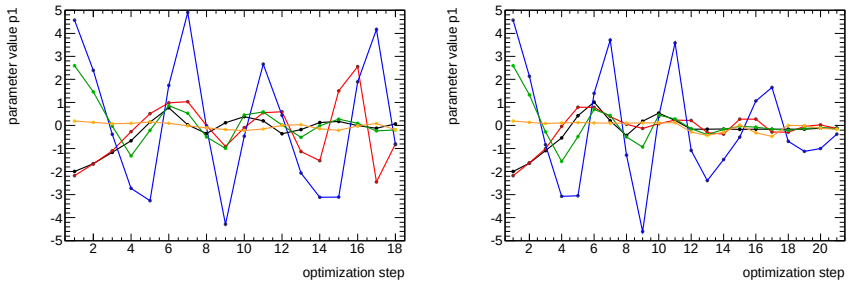
The weights are varied between their respective default minimum and maximum values in figure 7. However, the swarm individuals do not converge in figure 6(a) because the maximum number of optimization steps is chosen too large for the simple problem at hand, hence the weights are still close to their initial values when the finishing condition (no improvement for $nFinish$ steps) is met.

[4]Note that this definition of conversion is not equal to the global best value converging towards the true optimum value.



(a) $w0 = 0.9$, $w1 = 2.5$, $w2 = 0.5$ (b) $w0 = 0.4$, $w1 = 0.5$, $w2 = 2.5$

Figure 6: Movement of 5 swarm individuals in one parameter dimension with weight parameters fixed to initial (a) or final (b) default values. The optimization problem has been slightly modified here to create artificial local maxima at ± 1 cm (by a filter).



(a) *nSteps* 50 (b) *nSteps* 20

Figure 7: Movement of 5 swarm individuals in one parameter dimension with time varying weight parameters and $nFinish = 10$.

5 Optimization on a computer cluster

Since VITESS version 3.2, the optimization can be done on a computer cluster instead of the local computer by setting the "application option" in *opt_param.ini* from 'sim_opt_pc' to 'sim_opt_grid'. Instead of writing and running *Simulations.sh*, the optimization routine then uses `gridrun` to start VITESS simulations on worker nodes of a computer cluster. In case of the `swarm` algorithm, a simulation job for each swarm individual is sent simultaneously to the cluster in every optimization step, i.e. $nBees$ subjobs run in parallel. In case the `metropolis` algorithm is used, the number of simulations running parallel on the cluster equals the number of parameters to be changed simultaneously, i.e. $nTstPar$. In case of the gradient methods, the number of simulations running in parallel is two times the number of parameters. A further splitting of the simulations is not supported! This will lead to errors because the simulation results of these subjobs would have

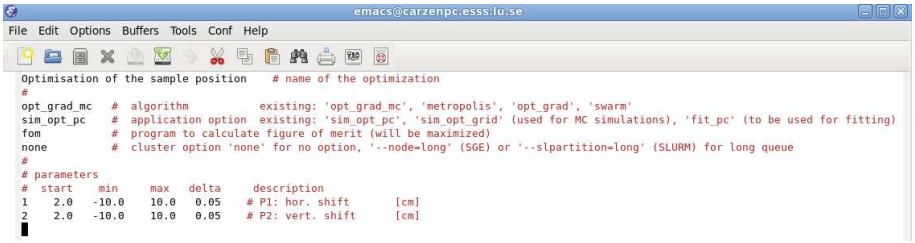
to be merged in a separate step, which is currently not implemented.

The queue or worker nodes of te cluster can be specified as "cluster option" in *opt_param.ini*:

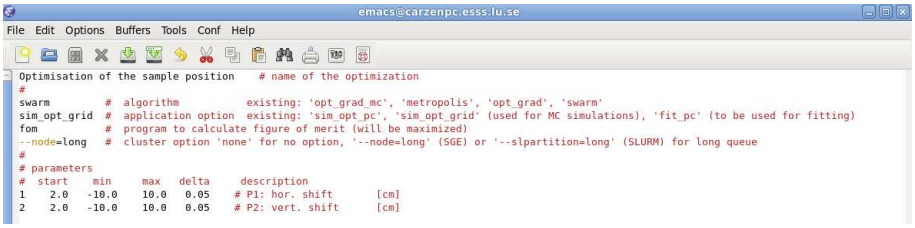
- On a cluster using SGE[5], `--node=long` will send the simulations to the long queue
- On a cluster using SLURM[6], `--slpartition=long` will send the simulations to the long queue

In both cases, `--name=MyJobName` can be used to name the cluster job.

Examples of *opt_param.ini* for a local optimization and one on a cluster using SGE are shown in figure 8.



(a) local option



(b) cluster option, using the *long* queue

Figure 8: In *opt_param.ini*, the parameters for running on the local computer (a) or running on a computer cluster (b) are defined.

note: When running parallel jobs in an optimization on the ESS-DMSC cluster, the path to the simulation directory should be removed from *std_instr.cmd* such that input/output files are given without any path, and `--P[my_path]` is replaced by `--P$P`. The working directories will be created in the directory `~/.vitess` same as when using `gridrun` without optimization, but will be named `o<No>` for optimization instead of `w<No>`. These jobnumbers are counted separately and will not be displayed when calling `gridrun` .

If several optimizations run at the same time, there is a smal chance of running into problems if two optimization routines try to give the same

[5]tested on the HZB cluster `dirac`

[6]tested on the ESS-DMSC cluster

number to a new job in a new iteration. This can be avoided by changing the working directory names with `export GRIDRUNDIR=a` before starting the optimization. The working directories will then be ∼`/.vitess/a<No>` instead of ∼`/.vitess/o<No>`.

6 Fitting

The gradient and `metropolis` optimization routines can also be used for fitting. The quality factor Q then becomes the error square sum:

$$Q = \sum_{i=1}^{N_{pts}} w_i \left(f(\vec{P}; x_i) - y_i \right)^2 \ .$$

y_i is the measuring value at point x_i, w_i is the weighting factor of this data point. x, y and w values are read from file in the main program (parameter *-Dfilename*). The first column is regarded as x-value, the second as corresponding y-value. If there is a third column, it is interpreted as weight. Otherwise the weight is set to 1 for all points.

The function f depends on the parameter set $\vec{P}$ and calculates values for all x_i, which are compared with y_i. It has to written by the user in the form

```
short FitFctPc(double F[IMAX+1], const double X[IMAX+1], const double
P[NMAX+1], const int nPts, const short nP)
```

with

F : function corresponding to parameter set P

X : parameter $X_1 ... X_{pts}$ (e.g. wavelength)

P : parameter set, for which the fct is calculated

nPts: number of points in spectrum (N_{pts})

nP : number of parameters in a parameter set ($P_1 ... P_n$)

return: TRUE/FALSE

In the `metropolis` algorithm, the probability for acceptance becomes now

$$p = e^{-\left(\frac{Q^2 - Q_0^2}{N_{pts}\sigma^2}\right)}$$

with N_{pts} being the number of measuring points and σ the standard deviation of a measuring value.

References

[1] D. Braess. Über Dämpfung bei Minimalisierungsverfahren. *Computing*, 1(3):264–272, 1966. ISSN 0010-485X.

[2] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6): 1087–1092, 1953.

[3] A. Ratnaweera, S. Halgamuge, and H.C. Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *Evolutionary Computation, IEEE Transactions on*, 8(3):240–255, 2004.