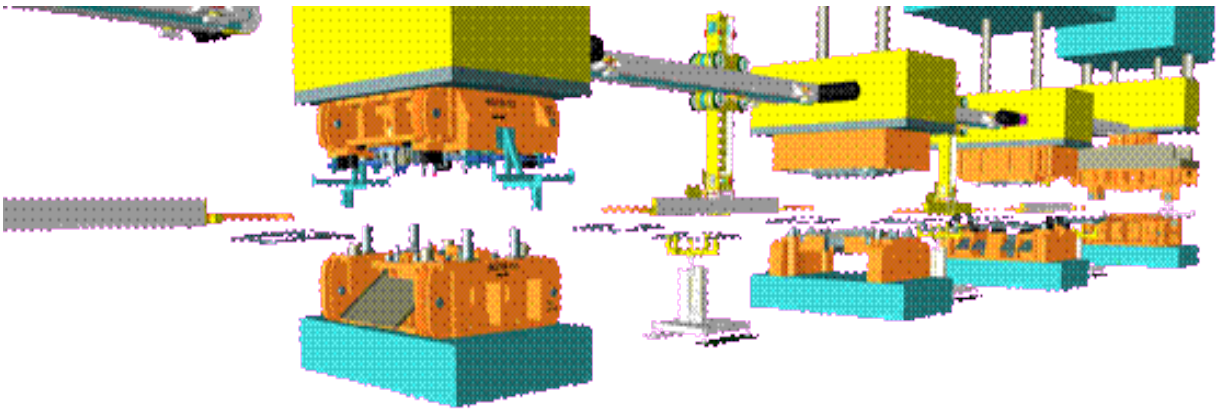
ENSIL-Mechatronic speciality
2nd year

report of technical studies

Implementation of a Response Surface Model based global optimization method for Virtual manufacturing
Part1



TORRES Stephane

Supervised by Fredrik Danielsson, Bo Svensson and David Lindstöm.

ACKNOWLEDGEMENTS
I would like to thank my supervisors at University West.
I would like to thank Fredrik Danielsson for his support and for these advice during the entire project.
I would like to thank Bo Svensson for his support, his help and his availability.
I also would like to thank David Lindstrom for his help and for his availability.
At last, I would like to thank Mattias Ottosson and Fredrik Sikström .

Table of contents

Table of pictures

For many problems from science and engineering it is impractical to perform experiments on the physical world directly. Instead, complex, physics-based simulation codes are used to run experiments on computer hardware. It 's exactly what the research group at Engineer Science of University West in Trollhättan have done in creating what they call the Virtual Manufacturing System.

The Virtual Manufacturing research group at Engineer Science of University West in Trollhättan has developed a general virtual manufacturing concept for industrial Control System. The aim of this concept is to create a virtual manufacturing model controlled by the same control code than the real machine. Currently this Virtual Manufacturing (VM) is applied in a sheet metal press line. So the Virtual Manufacturing model is today the model of a real manufacturing which is the sheet metal press line.

Maximizing throughput, i.e increasing the total number of produced components in a pre-existing production line, is a continuous and never-ending process in industry.

Most sheet press line tuning methods today are on-line and highly empirical, depending on individual operators. But the on-line method disturbs the production and can cause unwanted stops. Particularly if a collision occurs in the press line, long production stops with high economical costs arise. Consequently, most operators prefer a simplified short tuning, resulting in acceptable performances, than a long, detailed and risky tuning, possibly resulting in better performance. Thus, there is a lack of an appropriate off-line parameter tuning method for sheet metal press lines, which could prevent production disturbance and costly collision.

The Virtual Manufacturing is connected to a process optimizer which allow to embed optimization method and so to test optimization method directly.

One of the important problem is that computer experiments often require a substantial investment of computation time (one simulation may take many minutes, hours, days or even weeks). This is especially evident for routine task such as optimization.

Firstly, I am going to introduce the Virtual Manufacturing concept developed in the University West, the simulation based optimization problem and the process optimizer which is connected to this Virtual Manufacturing

Secondly, I will introduce a current optimization method called the Response Surface Model (RSM).

Then, I will make a rapid description of the algorithm invented by David Lindstrom using this RSM.

Next, I will introduce the different ways to interface MATLAB code and C/C++ code. Particularly how call MATLAB code from C/C++ code.

At last, I will introduce the algorithm implementation in the process optimizer and the first results.

I.Virtual Manufacturing System and optimization problem.

The research group at Engineer Science of University West in Trollhättan have used the Virtual Manufacturing concept in order to apply it at the sheet metal press line at Volvo car Manufacturing in Goteborg, sweden. With this five press station and its nine robots, this production line require complex control functions to work. It ensues ninety adjustable parameters. Thus there is a need for a parameters study before starting optimisation of the production line. In a first time only one press stations is optimised and thanks to other assumptions it results a selection of 14 parameters which have mains effects on production goals. They have chosen to optimize 10 parameters in a same time to limit the dimension of the search space to ten. Past ten, it becomes almost impracticable for optimization algorithms and so much greedy on computing (one evaluation with the simulation to product one plate takes ten minutes to be done).

1)The Virtual Manufacturing concept.

The intention is to receive tuned parameters which have been found with an off line parameter tuning method, which directly can be transferred to the real manufacturing (the real production line) without any post processing or transformations. All process parameters constitute control function parameter, e.g. control gate positions, path zones, velocities, accelerations etc… in the real production line(real manufacturing). This implies that the complex control functions from the real manufacturing must be imported to the virtual manufacturing model. Thus Industrial Controller System (**ICS**) with real control code is necessary in the virtual manufacturing model, as the control code is the implementation of the control functions. Owing to the time dependent characteristics, a time synchronisation within the entire simulation is necessary to reach the same production performance as in the real manufacturing. To summarize, a time synchronized virtual manufacturing model including ICS with real control code is necessary. A convenient time synchronized virtual manufacturing concept already exists. It consists mainly of the following parts [ref 2]:

1. the SDSP (Synchronized Distributed Simulation Protocol) architecture.
2. Physical resource representation.
3. ICS including HMI **(**Human Machine Interface**)**.
4. Process optimizer.

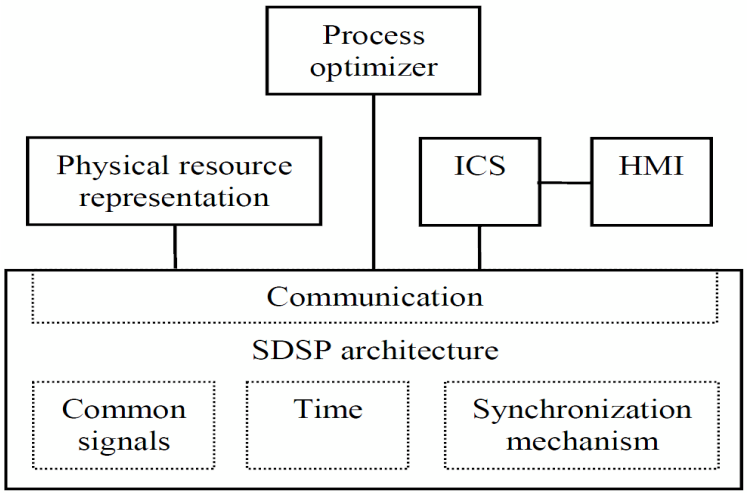


Fig1 : The time synchronised virtual manufacturing concept

The time and synchronization mechanism within the SDSP architecture achieves a common synchronized time which guarantees a deterministic behaviour of the entire simulation. The SDSP architecture is also responsible for the communication within the entire simulation and handles all common signals, e.g. servo values, process I/Os.
Physical resources are modelled in a commercial CAPE tool and form the physical resource representation. Examples of resources are robots, press stations etc. The CAPE tool RobCad utilizes a discretized time with equal time steps, which makes it suitable in the virtual manufacturing.
The representation of control functions in CAPE tools are usually both rather general and abstract and are described using simplified models of the main function behaviour of the real ICS. Therefore the ICS and HMI should remain unchanged from the real manufacturing, while embedding them into the simulation. Since the control code is the implementation of the control function, all complex control functions are included in the simulation and the tuned process parameter are possible to transfer directly to the real manufacturing.
All this together yields a high number of programs of different types, e.g. RobCad, models written in C/C++, ICS control code etc.., all executed on several CPUs and time synchronised in the SDSP architecture .In the test case study more than 20 programs and 20 CPUs are included, and the ICS control code exceeds 100000 lines code. The virtual manufacturing concept is indeed competent to simulate complex automated manufacturing systems including both material processing and material handling as a sheet metal press line.
The VM concept, based on the SDSP architecture, has been validated through several test cases. The most comprehensive test case carried out is a VM scenario of a sheet metal press line.

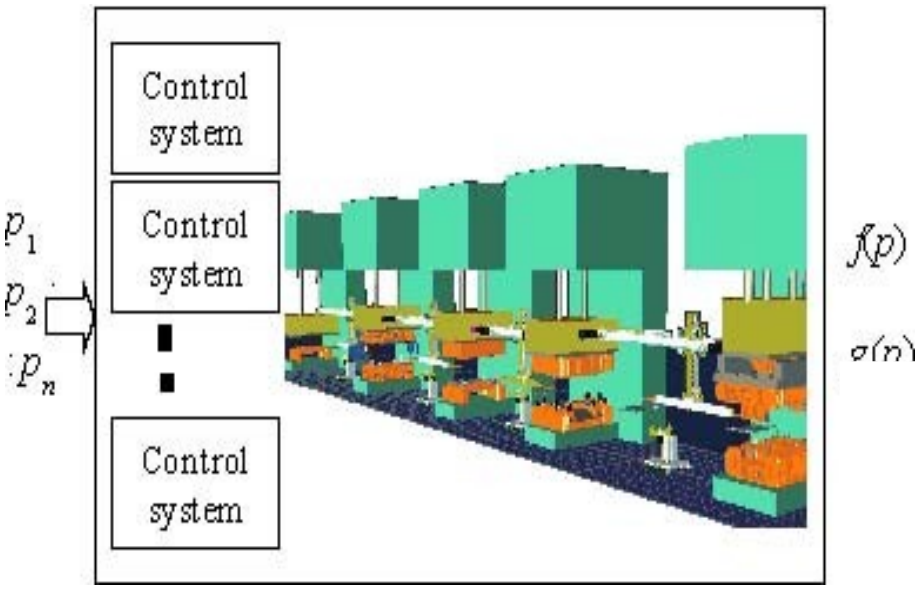


Fig2 : Robcad Production line

Above is a picture of the CAPE tool RobCad representing the simulating production line with the five press stations of a production line.
p=[p1,…,pn] are the real parameters of the production line.
f(p) is the response from the simulation that we want to optimize.

2)Simulation based optimization.

Real-world manufacturing problems often contain nonlinearities, combinatorial relationships and uncertainties that are too complex to be modeled analytically. In these scenarios, simulation-based optimization is a powerful tool to determine optimal system settings. Simulation based optimization is the process of finding the best values of some parameters for a system, where the performance of the system is evaluated based on the output from a simulation model of the system. Finding the optimal parameter values is an iterative process. An optimization procedure generates a set of parameter values and feeds them to a simulation that estimates the performance of the system. Based on the evaluation feedback from the simulation, the optimization procedure generates a new set of parameter values and the generation-evaluation process continues until a user-defined stopping criterion is satisfied.

3)The process optimizer.

As you can see on the figure number 1, a process optimizer which includes production performance calculation and optimization algorithms is connected to the virtual manufacturing model [ref 1]. The virtual manufacturing model is executed as a time discrete simulation with a fixed time step. The purpose of the simulation is to evaluate actual parameter settings. It can be represented like below.

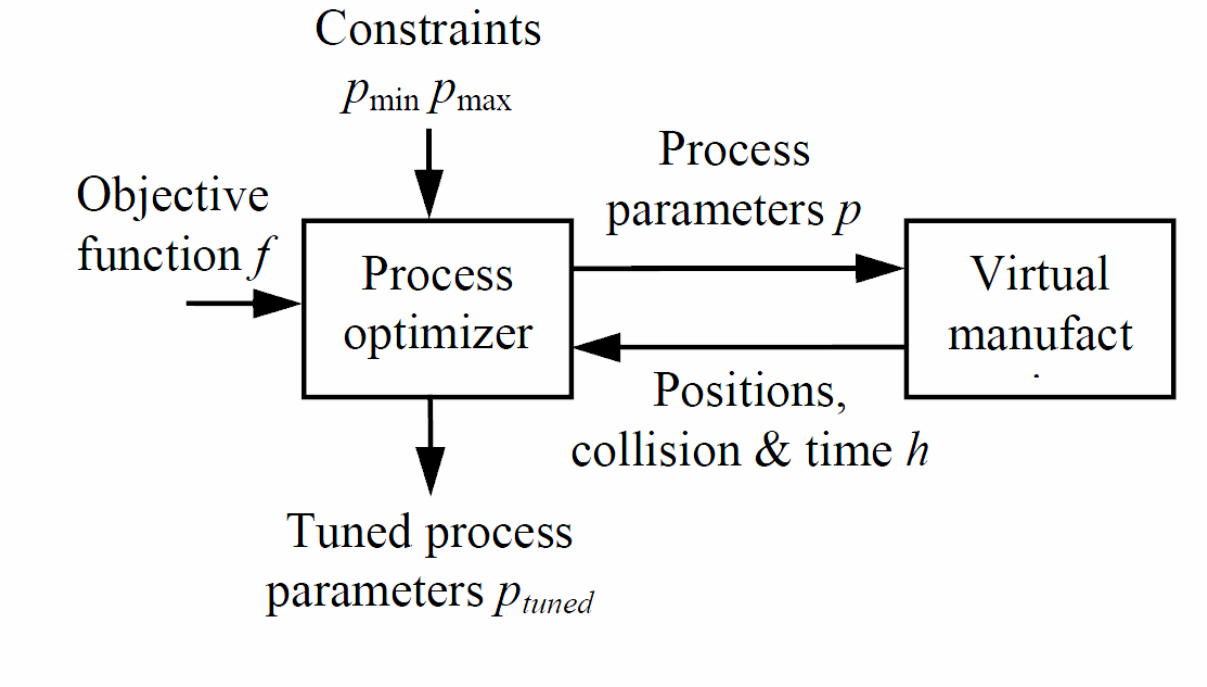


Fig3 : Connection between process optimizer and virtual manufacturing

You can see on the figure above the parameter vector p with all the process parameters to tune, e.g. control gate positions, path zones, velocities, etc. is formed as, p=[p1,…,pn] where n is the number of parameters to be tuned (in our case n = 14). The optimization constraints are constituted of the tuning parameter limitations according to , $p_{jmin} \leq p_j \leq p_{jmax}$, where p_{jmax} and p_{jmin} are upper and lower physical parameter limits, e.g. on position, speeds, accelerations, etc. for all process parameters p_j to be tuned for j=1 to n.

We use the virtual manufacturing model as a black box which takes as inputs a parameters combination and returns after one evaluation of these parameters some information about the sheet metal press line behaviour. The process optimizer is connected to it in order to make the calculations of production performance and optimization algorithms.
One evaluation in the black box includes both warm-up strokes and a measuring stroke, i.e. a production cycle producing one component. This corresponds to an evaluation of the parameter vector p with the simulated process function h, $h(p)=[h_{pos}, h_{collision}, t_{stroke}]$, where h_{pos} is the robot positions at each time step; $h_{collision}$ is a Boolean value indicating collision free motions in the production; and t_{stroke} is the stroke time. From these received results the process optimizer then calculates all desired production performances like production rate, speeds, accelerations, etc…

The process optimizer with these information and with the weight values (which are chosen by the user of process optimizer) calculates what we call the objective function :

$$f_{obj}(p)=c1*f_{pr}(p)-c2*f_{maw}(p)-c3*f_{mae}(p)$$

where :

f_{pr} → production rate (number of pressed sheet metal components per minute)

f_{pr} → the robot mean acceleration with a component in the gripper

f_{pr} → the robot mean acceleration with an empty gripper

c1, c2 and c3 → the weight values (inputs of the process optimizer)

The production rate is calculated as $f_{pr}=\frac{60}{t_{stroke}}$ where t_{stroke} is the time in seconds, to press one component. t_{stroke} is received from the executed virtual manufacturing model. The robot gripper speed, acceleration and jerk are calculated according to ,

$$f_{speed}=\frac{h_{pos}(t+t_{step})-h_{pos}(t)}{t_{step}}$$

$$f_{acc}=\frac{f_{speed}(t+t_{step})-f_{speed}(t)}{t_{step}}$$

$f_{jerk}=\frac{f_{acc}(t+t_{step})-f_{acc}(t)}{t_{step}}$ where t_{step} is the discrete time simulation step of 5ms and h_{pos} is received from the executed virtual manufacturing model.

When a sheet metal component is in the gripper, the robot mean acceleration is :
$f_{maw}=\frac{1}{n_w}\sum_{t\epsilon W}|f_{acc}(t)|$ where W is the set of times t when a sheet metal component is in the gripper and n_w is the number of elements in W where $f_{acc}\neq 0$.
When the gripper is empty, the robot mean acceleration is :
$f_{mae}=\frac{1}{n_E}\sum_{t\epsilon E}|f_{acc}(t)|$ where E is the set of times t when the gripperis empty and n_E is the number of elements in E where $f_{acc}\neq 0$.

The most obvious optimization criterion is high production rate in combination with collision free motions $h_{collision}$. However, increased production rate may imply increased robot speeds, accelerations and jerks in magnitude and they are limited by the real production process constraints. High accelerations and jerks, even if less than those permitted, may cause the sheet metal to slide out of position in the gripper or even fall out, thereby causing lengthy down times in the press line. Thus, production process constraints on speeds, accelerations and jerks are not sufficient; the robot motions also have to be smooth. High accelerations and jerks also cause wear of the production equipment and thus decrease its lifetime, which in turn inevitably leads to halts in production. That is why we want to minimize the robot mean acceleration in order to have the robot motions smoothed.

In reality we use the VM model and a part of the process optmizer as a black box (as you can see below). Indeed, we optimize the objective function which is calculated with the information about production performance and these last are calculated with the information returned by the VM.

Fig4 : Blackbox

The algorithms (methods to find the optimum of the objective function in tuning the process parameter p) are included in the process optimizer.
When the process optimizer has finished to work, it has normally found the tuned process parameter p_{tuned} . It 's this p_{tuned} that we want to send directly to the real manufacturing without any post-processing or transformations.

To reach an industrially useful method it is necessary to keep the total optimization time t_{opt} down to practical useful times, i.e. days or weeks. Since this is determined according to, $t_{opt} = n_{eval} t_{onesim}$, both the number of evaluations n_{eval} and the time of one evaluation t_{onesim} has to be examined. With the continuous progress in calculation power in computer, in model and in simulation, t_{onesim} will be improve with the time. At the moment t_{onesim} is still long, approximately 10 minutes. Consequently it is still vital to keep n_{eval} low. The number of evaluations depends on both the optimization method and the number of process parameters to tune.

II Method optimization based on RSM (Response Surface Model)

1.what is RSM

Response Surface Methodology/Model has its origin in the work of Box and Wilson (1951). Their collaboration initiated at a chemical company when solving the problem of determining optimal operating conditions for chemical processes. RSM is used in many practical applications in which the goal is to identify the levels of p design factors or variables, $\theta=(\theta_1, \theta_2, \ldots, \theta_p)$, that optimizes a response, $f(\theta)$, over an experimental region.
The main idea of RSM is to use a set of designed experiments to obtain an optimal response. RSM can be used for the optimization of physical experiments but also for optimizing functions getting by computer simulation.

RSM or Surrogate model or Metamodels, are compact scalable analytic models which can approximate the multivariate input/output behaviour of complex systems, based on a limited set of computational expensive simulations for example. We use this model to get some information about the function we would like to optimize.
Nevertheless we have to be aware that the model created with RSM is only an approximation of the real function and not the true objective function.

The response Surface Methodology can be divided in two parts:
-Design Of Experiments (DOE) : need to choose the points to evaluate with the experiment to get as much information from the multidimensional search space as possible. Indeed, to construct a model we need some points with their values.
-Model Building : construction of the surface.

Below some examples of DOE :
-Screening
-Full/fractional factorial design
-Central composite design
-Latin hypercube design:
-Optimal experimental design

Below some method to build a model :
-Regression
-Moving least-squares
-Kriging
-Radial Basis Functions

In practise on the factory floor, a common method for finding the optimum is via “one-shot” response surface model. In one-shot RSM, the system responses are fit with a regression model using a classic experimental design, and the optimal solution is determined from this model. This method can be considered inefficient in that it attempts to accurately predict the response curve over the entire domain of feasibility, while we are more interested prediction in the neighbourhood of the optimum. In addition, the regression models are often relatively simple and may not adequately fit certain complex systems over the entire feasible region.

Sequential RSM procedures have been applied for Simulation optimization. Instead of fitting the entire feasible region, small sub-regions are explored in succession, which leads to potential improvement, until the slope is "approximately" zero. Usually, in the initial phase, the subregions are fitted with first order regression models via fractional factorial designs. A move is taken towards the steepest descent direction for example, with step size determined by a line search. In the final phase, a quadratic model with a central composite design for example is fit and the optimum determined.
This sequential strategy for RSM is of course, better than the one-shot strategy but the number of experiments response needed can be very big. When we want to use the RSM for a optimization based simulation problem, reduce the computationally expensive computer simulation is a real challenge. The goal of the sequential RSM is to find the best model as possible with the least time as possible in computer simulation.

2.Examples of using a RSM for optimization problem.

1.Optimization of Kojic Acid Production Rate Using the Box-Wilson Method

Kojic acid is an organic acid produced by fungi and bacteria during fermentation of various carbon-containing substrates. This organic acid is used in the food industry as an inhibitor of polyphenol oxidase, an enzyme responsible for the blackening of agricultural products. Optimization of kojic acid production rate was carried out by two chosen independent process variables using a factorial experimental design. Initial D-glucose concentration(g/l) and initial polypepton concentration(g/l) were designated as independent variables.
The fractional design has been chosen as design of experiment [ref 11]. The fact is that they use physical experiments and so they have to replicate each experiment four times to make an average for each value. With the results of their experiments they construct with regression method a response surface model of kojic acid production rate. With the response surface value they can see where is the optimal point given by the model and in experimenting this point they well find the optimal point given by the model.

2. Metamodel-Assisted simulation-based optimization of a real world manufacturing

The problem is about finding optimal buffer levels in a complex production line in order to minimize product lead times and maximize system throughput.
The idea to solve this problem was to use an Evolutionary algorithm (EA) using a metamodel [ref 6]. Typically, an EA requires thousands of simulation evaluations and one single evaluation may take a couple of minutes to hours of computing time. The purpose of metamodels is to approximate the relationship between the simulation output variables and the input variables by computationally efficient mathematical models. By adopting metamodels in simulation, the computational burden of the optimization process can be greatly reduced, indeed it reduces the limitations of time consuming simulations.

3.Simulation-based optimization of dispatching rules for semiconductor wafer fabrication system scheduling by the response surface methodology

A wafer is a thin slice of semiconductor material.
Dispatching rules is an alternative to scheduling all jobs on all machines in semiconductor manufacturing systems.
A simulation-based evaluation and optimization method has been used in semiconductor wafer fabrication [ref 7]. With this method, dispatching rules were selected by the performance evaluation, and parameters of Dynamic Bottlenecks Dispatching were optimized by RSM and the simulation in using the central composite design as DOE.
Scheduling of a semiconductor wafer fabrication system (SWFS) is complicated due to its re-entrant product flow, high uncertainties in operations, and rapidly changing products and technologies; thus dispatching rules have been widely used for real-time scheduling because they can provide a very quick and pretty good solution. However, deciding how to select appropriate rules is very difficult and seldom tackled. An approach into the evaluation and optimization of dispatching rules by integrating the simulation and response surface methodology (RSM) has been considered. In order to implement this approach, a dynamic bottleneck dispatching(DBD) policy has been designed, in which bottlenecks are detected in a timely way and adaptive dispatching decisions are made according to the real-time conditions. In addition, two case studies have been carried out to demonstrate the approach. One case compares DBD to regular rules, such as 'critical ratio' (CR) + 'first in first out' (FIFO), 'earliest due date' (EDD), 'shortest remaining processing time' (SRPT), 'shortest processing time' (SPT), 'shortest processing time until next bottleneck' (SPNB) and Justice, a bottleneck dispatching method. Simulation results showed that the DBD policy is superior to the other six methods. In another case study, the parameters of DBD are optimized by RSM and desirability function, and the result proved that the optimized DBD method can get better performance.

4. Simulation-based optimization with surrogate models : Application to supply chain management

Simulation is widely used in the decision-making processes associated with supply chain management. An extension of the simulation-based optimization framework which has been proposed for analysing supply chains has been studied [ref 8]. The extension consists of the iterative construction of a surrogate model based on systematically accumulated simulation results to capture the causal relation between the key decision variables and supply chain performance. The decision variables can then be optimized using the surrogate model in place of individual simulation runs to economize on the overall computational effort.

5.Simulation-Based Optimization of a Complex Mail Transportation Network

The Swedish Postal Services receives and distributes over 22 million pieces of mail every day. Mail transportation takes place overnight by airplanes, trains, trucks, and cars. The transportation network comprises about 12 freight centrals and a huge number of possible routes. A discrete-event simulation model of the transportation network has been developed for testing and analysis of different transportation plans [ref 9]. To support generation and optimization of transportation plans, there is now ongoing work in developing efficient metaheuristic optimization techniques to be coupled with the simulation model. The vast transportation network in combination with a large number of possible transportation configurations make the optimization problem very challenging. Complex real-world problems like this tend to be computationally expensive as a large number of simulation evaluations are needed before an acceptable solution can be found. To adress this problem, a computationally cheap surrogate model is used to offload the optimization process.
In summary, the problem is to find optimal routes in a complex mail distribution network taking into consideration transportations' time, cost, and pollution. A property of the transportation network adding further complexity to the problem is that transports can be merged and split at certain nodes. Transports therefore do not necessarily start and end at freight centrals. A discrete-event simulation model of the transportation network has been developed and is used in the optimisation to evaluate solutions. However, a huge number of solutions need to be evaluated and for increased efficiency a computationally cheap surrogate model has also been developed. The surrogate model is used to guide the search and to cull unpromising solutions.

6. Surrogate-based process optimization for reducing warpage in injection molding

Injection molding is the most widely used process for making plastic products. In this process, the melted polymer is forced into a cold mold cavity with a desired shape and then cooled down under a high packing pressure. Warpage is a serious defect in injection molded parts, especially for the thin-wall plastic products, so how to reduce warpage is becoming the key to improve the part quality. Warpage can be reduced by changing the geometry of parts, modifying the structure of molds, or adjusting the process parameters. The mold design and part design are usually determined in the initial stage of product development, which cannot be easily changed. So systematical optimization of process parameters should be more feasible and reasonable.
In order to reduce the computational cost in warpage optimization, some researchers introduced some surrogate models, such as polynomial regression, artificial neural network and support vector regression [ref 10]. These surrogate models were used to construct a mathematical approximation for replacing the expensive simulation analyses in warpage optimization.
Gao and Wang (2008) proposed a sequential optimization method based on Kriging surrogate model. The sequential optimization can improve the surrogate function using a current optimal solution until the convergence criteria are satisfied.
An adaptive optimization method based on Kriging surrogate model was proposed to minimize the warpage of injection molded parts. Kriging surrogate model combining design of experiment (DOE) methods is used to build an approximate function relationship between

warpage and the process parameters, replacing the expensive simulation analysis in the optimization iterations.
The adaptive process is implemented by an infilling sampling criterion named expected improvement . This criterion can balance local and global search and tend to find the global optimal design, even though the DOE size is small. As an example, a cellular phone cover was investigated, where mold temperature, melt temperature, injection time, packing time and packing pressure were selected to be the design variables. The results show that the proposed adaptive optimization method can effectively decrease the warpage of injection molded parts. The warpage was reduced by 38% comparing with minimal warpage value in samples. This indicates that the optimization method is efficient for reducing warpage of injection molded parts.

With all these examples we can see different strategy using the RSM, with physical experiments or computing experiments, and for our simulation based optimization problem, it 's the sequential strategy which could be interested.
The question is always the same: “Which parameter combinations to simulate in order to find the optimum?”
In the next part, I describe an optimization algorithm based on RSM.

III Algorithm based on RSM

Algorithm invented by David Lindström and Kenneth Eriksson [ref 4].

Method to optimize a function whose we just know some points.
In our case we can have until 10 parameters to optimize, so the search space can be in 10 dimensions which is a large search space. This search space depends on the parameter boundaries and more the distance between the lower and upper boundaries is large more the search space will be large.

Firstly, we have to choose how many points to evaluate to initialize the method to look in all the search space. This number is important because the next step consists of constructing the statistically most probable interpolating function $\hat{f}$ which will represent the objective function model. Each point is a parameter combination.(point x1 depends of param1,param2, …,param n if we are in n dimension. Indeed param1,param2,…,param n are the coordinates of the point xi in the n dimension space with n the number of points). The DOE just need to have the boundaries of the n parameters and a number of initialization points required (for example 25 initialization points). With these boundaries, the number of initialization points and in using the Latin hypercube design (LHD, see annex) we get coordinates for the initialization points in thinking that use LHD allow us to get as much information from the multidimensional search space as possible. After receiving the value of the objective function for these initialization points from the black box we can construct the first interpolating function but this interpolating function is just one possible metamodel that could be used to approximate the objective objective function fobj(x) in the continued search for its global minimum. This interpolating function is aiming to be the statistically most probable interpolating function.

The second goal of the method is to choose the next point to evaluate in order to come close to the optimum of the objective function in using all the points already calculated.
To do that, we create a normalization of the predicted function values $\tilde{f}(\tilde{x})$ with the interpolating function, and with this normalization of the predicted function values we construct an auxiliary function Φ depending of

$$\tilde{x} = \frac{point\ x - Low\ value\ of\ the\ parameter}{High\ value\ of\ the\ parameter - Low\ value\ of\ the\ parameter}$$

$$\Phi(\tilde{x}) = w(\tilde{x}) * d(\tilde{x})^{(n+1)}$$

The auxiliary function is composed by one weight function $w(\tilde{x})$ and by one distance function $d(\tilde{x})$. The weight function is large in regions where the metamodel is forecasting an improved objective function value, and small elsewhere.
The distance function is a measure of the local uncertainty in the metamodel.
The next point chosen to be evaluated x26 in our example is the point which maximizes this auxiliary function.
We evaluate this point with the black box and so we get fobj(point x26).
With this new point, a new interpolating function and so a new normalization of the prediction function values and so a new auxiliary function are built.
And we keep maximizing the auxiliary function and running the process with all the new points until reaching the maximum iteration. At the end of the process (if max iteration is enough large) the algorithm normally find the global minimum of the objective function.

At each new iteration that is to say at each new evaluation, we change the size of σ , which is a parameter of the weight function $w(\tilde{x})$, to obtain a robust method with a good balance between global and local search. The basic idea behind the design of the auxiliary function is the fact that by maximizing $w(\tilde{x})$ separately we will obtain local search, and by maximizing $d(\tilde{x})$ separately we will get global search. Indeed, evaluation points are chosen in such way that local search and global exploration is balanced. Instead of putting the next point exactly where it is most likely to find the global optimum, the method prepares for steps to come by minimizing the total uncertainty of the interpolated function within the most interesting areas of the search space.

To be able to use this algorithm in the Process Optimizer described above, the algorithm was divided in two MATLAB functions :
_initialization method (initdoe.m)
_newpoint method (newpoint.m)

Initdoe use the boundaries of the n dimension space and the number of points chosen by us(to initialize) to calculate the coordinates of points in order to create the first statistically most probable interpolating function.
We create a model with the coordinates of the points calculated by the initdoe method and their values given by the black box in using the dacefit function. The dacefit function is used to find a DACE model to a given set of design data and given regression and correlation models. Then we sample the area limited by the boundaries of the parameters in cutting it in small area with the gridsamp function. The gridsamp function allow to get design sites in a rectangular grid. Then we evaluate with the predictor function all the point in the design sites given by gridsamp. The predictor function use the DACE model to predict the function at one or more untried sites. Then the algorithm look for which point could be interesting to evaluate in the next step in maximazing the auxiliary function as said above. This is done by newpoint. We use this newpoint function in a sequential way.

Below the two MATLAB functions description :

-initdoe(A , B , C , D) : function with four inputs and which returns the resulting initial array of design sites getting with the LHD.
A : Dimension of the parameter space (pdim)
B : Number of initialization point (np)
C : Row vector with lower parameter boundaries(lbbox)
D : Row vector with upper parameter boundaries (ubbox)

-newpoint(E , F , G , H , J) : function with five inputs and which return the next point to evaluate in order to come close to the optimum.
E : Vector with previously evaluated design sites (X)
F : Vector with the response at E (Y)
G : Row vector with lower parameter boundaries (lbbox)
H : Row vector with upper parameter boundaries (ubbox)
J : The iteration number of the evaluated point (after the initial array of design sites)(i)

IV Matlab Interface with C/C++

As seen above, the algorithm has been created with the MATLAB software and the process optimizer is in C++. So in order to use this algorithm in our simulation based optimization problem, I have had to look for how interfacing MATLAB code and C/C++ code.
I say C/C++ code because the method to interface C and C++ code with MATAB are quite similar and C++ code can call C code.

goal : deploy an algorithm created with MATLAB in the process optimizer.

I have : -Algorithm.m
-files .c/.cpp & .h (inside the process optimizer)

The goal of my project is to find a method to be able to use an algorithm already created with MATLAB in the process optimizer. This means that I have to use the MATLAB code from the process optimizer. When someone want to interface MATLAB code and other programs he has to choose between two questions :
_ Do I need to call MATLAB code from the other programs?
or
_Do I need to call the other programs from MATLAB code?
In my case, I want to call MATLAB code from an other program. So I looked for how can I call MATLAB code from other program like C/C++ program.

1.Solutions to interface matlab code with other programs.

There are many methods of calling MATLAB functionality from other programs. And an important thing to know before deploying MATLAB code in other programs is :
Have we MATLAB installed on the computer where we want to run the other programs or not?

a) If Yes, the following methods are possible :

-MATLAB Engine

If we have access to the source code of the other program, or if the other program has a method of calling a shared library (.DLL in windows, or .so in UNIX) we can call MATLAB code from C or Fortran through the MATLAB Engine. The MATLAB Engine contains routines for controlling the computation engine.
For example :
examples of C engine routines (functions that can be used in a C program):
engOpen : Start up MATLAB engine
engClose : Shut down MATLAB engineering
enGetVariable : Get a MATLAB array from the engine
engPutVariable : Send a MATLAB array to the engine

engEvalString : Execute a MATLAB command

The C engine routines and also the mx functions allow to handle data between C and MATLAB.

-MATLAB Batch Mode Processing

We can start MATLAB in such a way that it executes a script or function immediately after startup. In our M-code we could then have it write the results to a file which we could then load into the other program.
Functions to consider using for data output include: fprintf, fwrite, csvwrite, dlmwrite and xlswrite.

b) If No, the following methods are possible :

-Compile M code to a shared library

If we have access to the source code of the other program, or if the other program has a method of calling a shared library (.DLL in windows, or .so in UNIX) we can compile the M-code into a shared library using MATLAB Compiler that we can then call from the other program.

-Compile M code to a stand alone executable.

We can compile the M-code into a stand alone executable using MATLAB Compiler which we could then invoke from inside the other program or manually outside of the other program. In our M-code we could then have it write the results to a file which we could then import into the other program.
Functions to consider using for data output include: fprintf, fwrite, csvwrite, dlmwrite and xlswrite.

Of course, an other solution will be to rewrite the algorithm in C++ but the algorithm use, as we have seen above, some functions (dacefit, gridsamp, predictor) wich are in some other m files than the algorithm and it will be a tricky problem if we want to rewrite these files in C++ because it used some special MATLAB functions and huge library.

In our case, the computer where we want to deploy the application with the process optimizer has not MATLAB installed. So I thought to use :
-Compile M code to a shared library
-Compile M code to a stand alone executable.

The solution chosen is : Mixing M-Files and C/C++ with a compile M code to a shared library created with MATLAB COMPILER.

I chose this solution because all the previous algorithm have been implemented in the process optimizer in C/C++ and in creating a shared library , we can then call the algorithm function directly from a C/C++ file and use the results of the algorithm function directly in the C/C++ file.
If I decide to use a stand alone executable, as said above, I have to launch the executables (as I have two method I need two executables) with the C/C++ file (I cannot do it manually because I need a lot of executable launching), write the results in a file, read this file with the C/C++ file and repeat this operation a lot of times. It will work but it will take too much time. And also the time to debug the program will be certainly bigger than with the shared library.

Below some pictures to see why I choose to use the shared library.
-Compile M code to a shared library

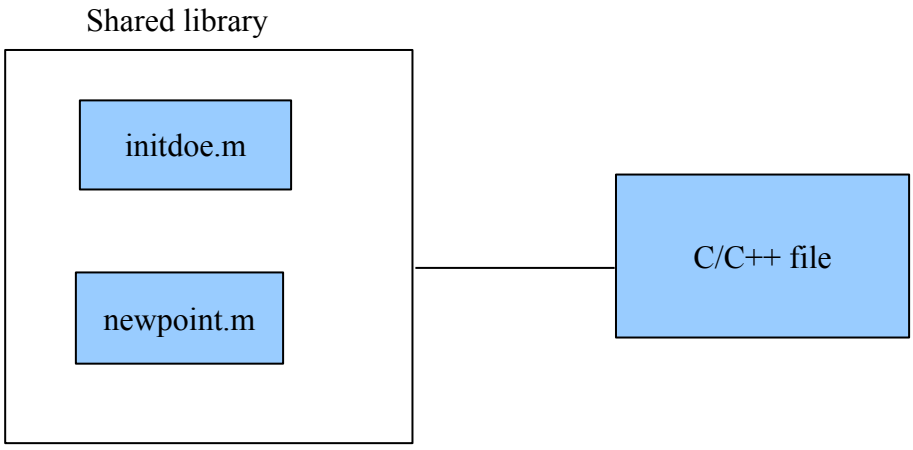


Fig5 : schema of using a compile M code to a shared library with C/C++ file

The functions can be used directly in the C/C++ file.

-Compile M code to a stand alone executable.

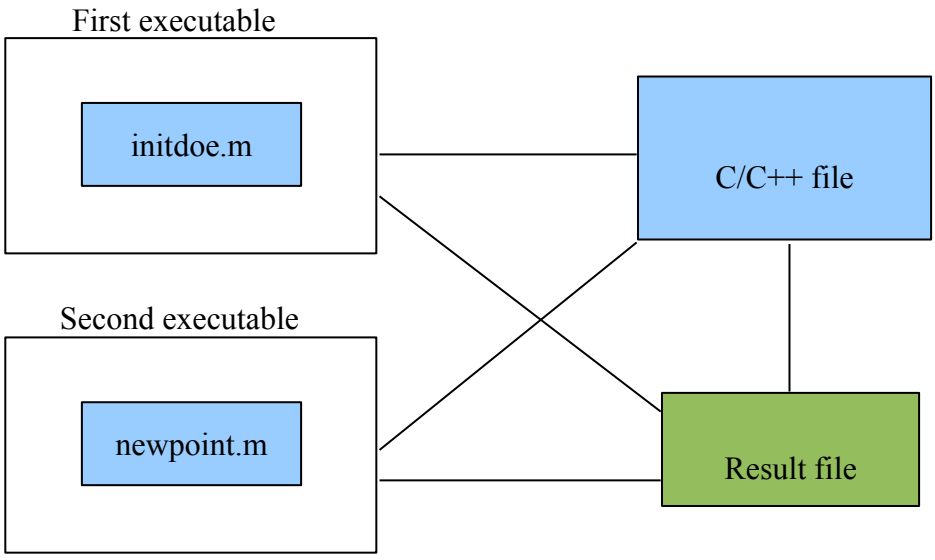


Fig6 : schema of using a compile M code to a stand alone executable with C/C++ file

The functions cannot be used directly in the C/C++ file and an intermediate file is necessary.

As shown above, the algorithm is divided in two functions in order to deploy it. Each function is a m file.
The shared library with the two files .m will be included in the C/C++ file in order to deploy the matlab algorithm in an environment without matlab.
That is why I said above mixing M files and C/C++.

2.How create a C shared library?

To create this shared library, I use MATLAB Compiler. MATLAB Compiler allow to share a MATLAB application as an executable or a shared library.
Procedure to convert M-Files into a C shared library using MATLAB Compiler that can be called from C/C++ program.

a. In MATLAB, we have to select a compiler with the mbuild command :
mbuild -setup

mbuild compile and link source files into stand alone application or shared library.

b. Compile M-Files into a C shared library with the MCC command :
mcc -B csharedlib:libname <M-Files>

mcc is the MATLAB command that invokes the MATLAB Compiler.
The -B csharedlib option is a bundle option that expands into -W lib:<libname> -T link:lib. The -W lib:<libname> option tells the MATLAB Compiler to generate a function wrapper for a shared library and call it libname. The -T link:lib option specifies the target output as a shared library.

If we execute this command and for example choose libdone as the name of the library, we will get new files as :

libdone.h
The library wrapper header file. This file has to be included by the applications that call the exported functions of libdone.

libdone.dll
The shared library binary file. On Windows, this file is libdone.dll. On Unix, this file is libdone.so.

libdone.lib
Import library. An import library is used to validate that a certain identifier is legal, and will be present in the program when the .dll is loaded. The linker uses the information from the import library to build the lookup table for using identifiers that are not included in the .dll.

I will have to use these files to embed the algorithm in the process optimizer.

3.MATLAB Compiler Runtime (MCR)

To distribute a shared library or a stand alone application for using with external application, I need to install MCRinstaller.exe on the computer where I want to deploy the application if it has not MATLAB installed. MCR allows to create a MATLAB environment on a computer without MATLAB. It 's a Self-extracting MATLAB Compiler Runtime library utility. This MCR is necessary to use MATLAB Compiler products on target computer without MATLAB installed.

4.Using a Shared Library

To use a MATLAB Compiler generated shared library in my program, I must perform the following steps :

1.Include the generated header file for the library in my program. Each MATLAB Compiler generated shared library has an associated header file named libname.h as we have shown above.

2.Initialize the MATLAB libraries by calling the mclInitializeApplication API function. I must call this function and it must be called before calling any other MATLAB API functions. mclInitializeApplication must be called before calling any functions in a MATLAB Compiler generated shared library. mclInitializeApplication sets up the global MCR state and enables the construction of MCR instances. mclInitializeApplication returns a Boolean status code. A return value of true indicates successful initialization, and false indicates failure.

3.For the MATLAB Compiler generated shared library that I include in my program, I have to call the library's initialization function. This function create the MCR instance required by the library. This function returns a Boolean status code. A return value of true indicates successful initialization, and false indicates failure.

4.When all the previous steps are done, I can call the exported functions of the library as needed. I need to use the MATLAB API functions to process input and output arguments for these functions.

5.When my program no longer needs the library, I call the library's termination function. This function frees the resources associated with its MCR instance. The library termination function will be named <libname>Terminate(). Once a library has been terminated, that library's exported functions should not be called again in the application.

6.When my program no longer needs to call any MATLAB Compiler generated libraries, I call the mclTerminateApplication API function. This function frees application-level resources used by the MCR. Once we call this function, no further calls can be made to MATLAB Compiler generated libraries in the program.

5 . MATLAB function in a shared library created with MATLAB Compiler.

For each m file (in our case we have several m files), MATLAB Compiler generates two functions, the mlx functions and the mlf functions. Each of these generated functions performs the same action (Call a m file). The two functions have different names and present different interface.

a. mlx interface function.

The function that begins with the prefix mlx takes four arguments. The first argument, nlhs, is the number of output arguments, and the second argument, plhs, is a pointer to an array that

the function will fill with the requested number of return values. The third and fourth parameters are the number of inputs and an array containing the input variables.

void mlxFunction(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[])

The variables nrhs and nlhs are the number of variables that MATLAB requested. They are analogous to nargin and nargout in MATLAB.

b. mlf interface function.

The function that begins with the prefix mlf expects its input and output arguments to be passed in as individual variables rather than packed into arrays. If the function is capable of producing one or more outputs, the first argument will be the number of outputs requested by the caller.

void mlfFunction(int nargout (2 in this case), mxArray ** out1, mxArray **out2, mxArray *in1, mxArray *in2)
This function need two inputs arguments in1 and in2 and return two output arguments out1 and out2.

So in our case, as I have two functions in my library I will have at least four functions in my library :
-mlxInitdoe(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[])
-mlfInitdoe(int nargout, mxArray ** out, mxArray *pdim, mxArray *np, mxArray *lbbox, mxArray *ubbox)
-mlxNewpoint(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[])
-mlfNewpoint(int nargout, mxArray **out, mxArray *X, mxArray *Y, mxArray *lbbox, mxArray *ubbox, mxArray *i)

6.API Functions (Application Programming Interface)

To be able to use the two methods created with MATLAB Compiler in the library (the mlx and mlf functions) with C code I have to use special functions to handle data between C/C++ code and M code which are MxArray Manipulation functions (mx functions) but also called API functions.
mx functions are used to access data inside of mxArrays. They are also used to do memory management and to create and destroy mxArrays. Some useful are:

Array creation	mxCreateDoubleMatrix, mxCreateCellArray, mxCreateCharArray
Array access	MxGetPr, mxGetPi, mxGetPM, mxGetData, mxGetCell
Array modification	mxSetPr, mxSetPi, mxSetData, mxSetField
Memory management	mxMalloc, mxCalloc, mxFree, mxDestroyArray, mempcy

7 . Difference of memory storing between MATLAB and C/C++ code.

In computing, row-major order and column-major order describe methods for storing multidimensional arrays in linear memory. We have to be careful when we use an array which method is used to store the array. As I will use array handling between C/C++ code and M code, I have to be careful of the way to store the elements in the array when I want to get the results of the functions in the library.

Column-major order is a method of flattening arrays onto linear memory, but the columns are listed in sequence. The programming language MATLAB use column-major ordering.
Example :

A= 1 2 3
4 5 6
7 8 9

if stored in linear memory with column-major order would look like the following:
1 4 7 2 5 8 3 6 9

In row-major storage, a multidimensional array in linear memory is accessed such that rows are stored one after the other. It is the approach used by the C/C++ programming language.
Example :

A= 1 2 3
4 5 6
7 8 9

if stored in linear memory with row-major order would look like the following:
1 2 3 4 5 6 7 8 9

8.The mxArray

The mxArray is a special structure that contains MATLAB data. It is the C representation of a MATLAB array. In my program, I use the mxArray as inputs and ouptuts of the API functions. All types of MATLAB arrays (scalars, vectors, matrices, strings, cell arrays, etc.) are mxArrays.

The MATLAB language works with only a single object type, the mxArray. All MATLAB variables, including scalars, vectors, matrices, strings, cell arrays, and structures are stored as mxArrays. The mxArray declaration corresponds to the internal data structure that MATLAB uses to represent arrays. The MATLAB array is the C language definition of a MATLAB variable. The mxArray structure contains, among other things:

-The MATLAB variable's name
-Its dimensions
-Its type
-Whether the variable is real or complex

9.example.

For example we have a MATLAB function which takes two inputs and return one output. The function just make an addition.

```
Function [a]=somme[a1,a2]
a=a1+a2
```

We create a library with this function libsomme.

Then to use the functions in this library from a C program, we can do like below :

```
mclInitializeApplication();
libsommeInitialize();

mxArray in1, in2;
mxArray out;
double data1, data2;

data1=1;
data2=2;

in1=mxCreateDoubleMatrix(1,1,mxReal);
in2=mxCreateDoubleMatrix(1,1,mxReal);

mempcy( mxGetPr(in1), data1, 1 * sizeof(double));
mempcy( mxGetPr(in2), data2, 1 * sizeof(double));

mlfSomme( 1, &out, in1, in2);

printf("result of the addition is : %lf ", mxGetPr(out));

libsommeTerminate;
mclTerminateApplication;
```

We will see as result of the program : result of the addition is : 3.00000

V Implementation in C/C++

The goal is to embed an algorithm in the process optimizer, Pressopt, in order to optimize the sheet metal press line. Some simulation based optimization methods have already been implemented in Pressopt but none of them use RSM. In implementing this algorithm based on RSM, we will see if the algorithm is ready or not for this kind of optimization problem. And we will see if it 's a good idea to use this kind of algorithm for this kind of optimization problem.

1.Pressopt

Below you can see the process optimizer which is used to off line optimize the performance of a manufacturing : Pressopt
Pressopt has been written to optimize the objective function which is calculated in Pressopt with the informations returned by the VM. It has been written in C++ with C++builder 6.

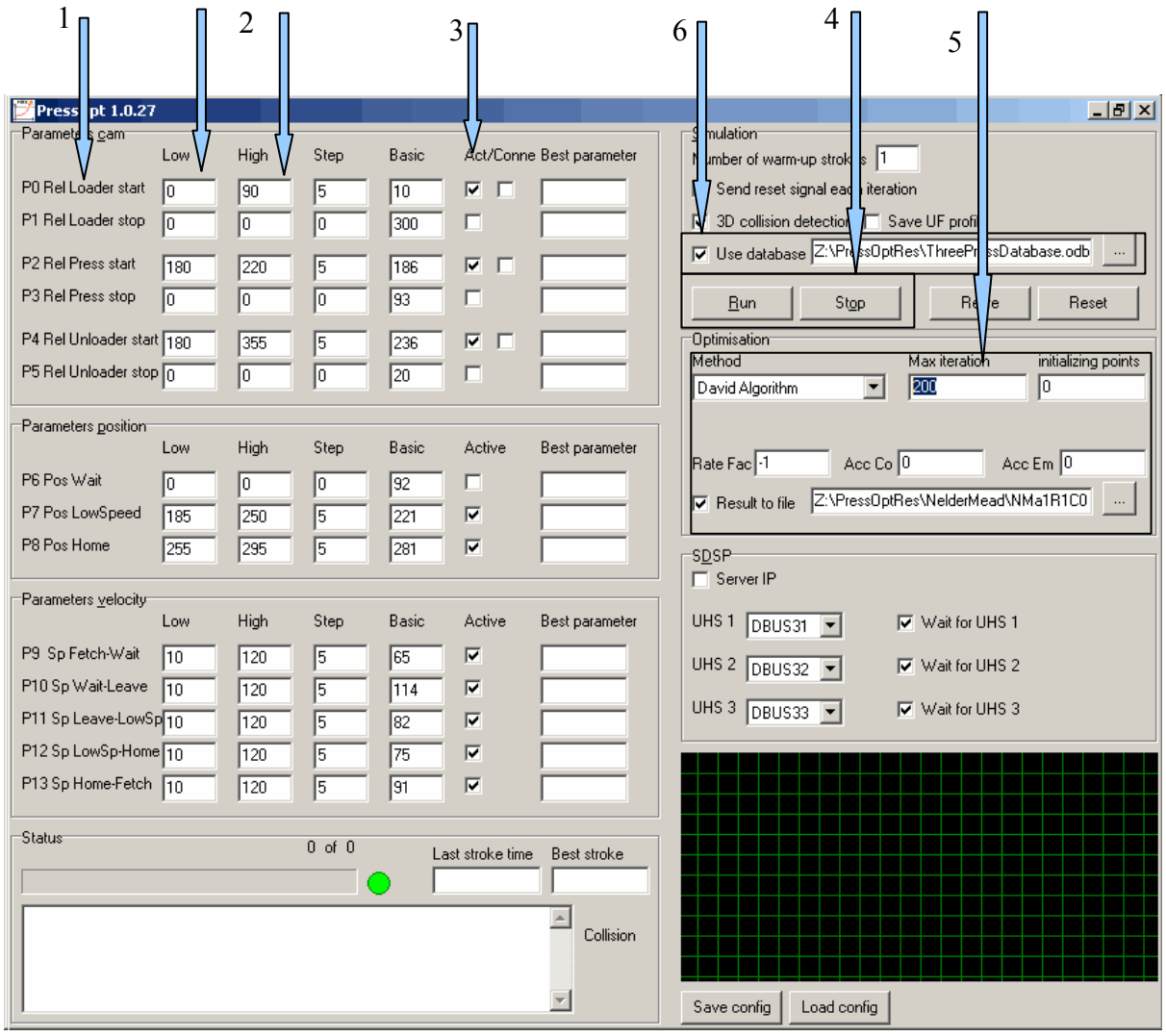


Fig7 : Pressopt panel

This picture shows the control panel of Pressopt.
In this control panel we have :

1. the fourteen parameters (P0,...,P13)
2. lower and upper boundaries for each parameter (Low and High value)
3. activated check box
4. control buttons for the method chosen (Run and Stop)
5. -box to choose the method we want to use to optimize the parameters
   -box to put some parameter for each optimization method
   -weight values for the objective function (c1=Rate Factor, c2=Acceleration Component, c3=Acceleration with empty gripper)
   -Directory file for printing the results of the method
6. Database containing all the points already simulating.

As said before, the goal is to optimize until 10 parameters and the VM need 14 parameters as inputs, so we have to choose only 10 parameters to optimize among these 14. This is done with the activated check box. We can choose a maximum of 10 parameters. Many elements in this panel are useful only for the first three optimization methods already implemented in Pressopt. For example step box, basic value and best parameter box are only use for Nelder Mead method and screening method.
As the algorithm look for the minimum of the objective function, I fill the negative value -1 for the weight value c1 to maximize the throughput of the sheet metal press line. For the first tests, I put 0 as the value of c2 and c3 but next I will put a positive value.

2.Program

The goal is to write a program in C/C++ using the algorithm based on RSM functions already created and include this program in Pressopt in order to use the VM to evaluate the points given by the algorithm based on RSM functions.

2.1algorithm architecture in the program

The algorithm based on RSM described above has a structure like below :

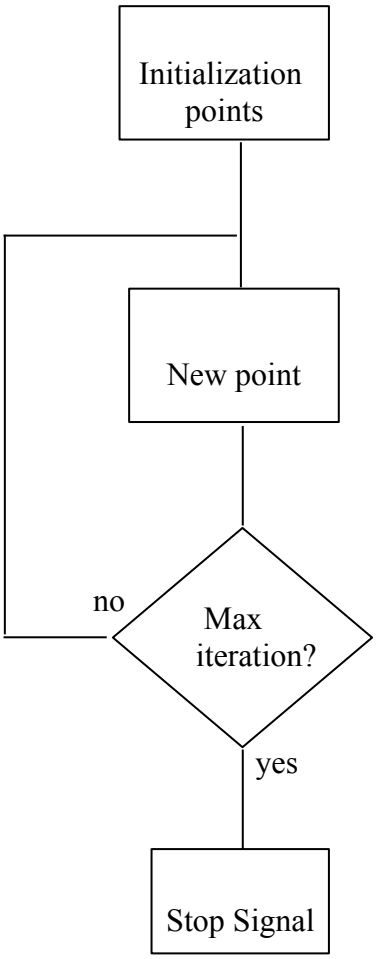


Fig8 : architecture using the MATLAB function in the program

The initialization points are the points returned by the initdoe method. For each point, one evaluation is needed, and with this evaluation we get the results of the objective function for the point evaluated. Then with these points and their values, we go to the second step of the algorithm which is to find a newpoint to evaluate in order to come close to the optimum of the objective function. After each newpoint calculated, we launch an evaluation with the VM and get the results. And we keep calculating and evaluating a new point until we arrive at the number max iteration. Once the max iteration number is getting, we stop the algorithm and so we stop the communication between Pressopt (process optimizer) and the VM.

2.2 procedure to simulate one point with the VM

Below a schema showing what Pressopt does when the program containing the algorithm want to evaluate one point.

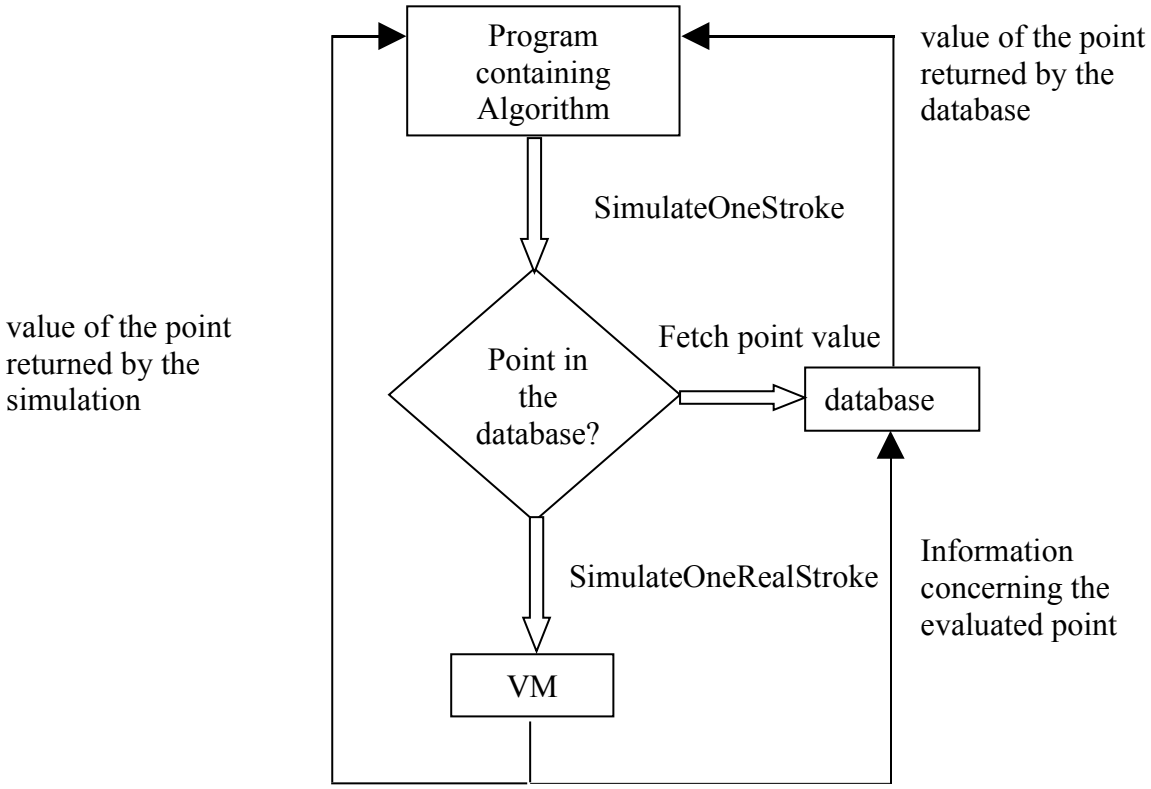


Fig9 : schema to evaluate one point with Pressopt

When the program containing the algorithm want to get the value of one point with the simulation, it uses the function SimulateOneStroke. The VM takes integers as format for the coordinates of the points that it simulates.This function is inside Pressopt, and when I call this function in my program, Pressopt, firstly, check if the point has already been calculated or not. All the points already calculated are in the database. Secondly, if the point has already been calculated, Pressopt use the database to return the value of the evaluation of the point. If not, Pressopt launch the function SimulateOneRealStroke which use the VM to simulate the real production line behaviour as seen in the process optimizer part. This simulation takes around 10 minutes, and at the end of this 10 minutes the results of the point evaluation is returned in the program and in the database.

2.3 procedure to include the library and the files in C/C++ code in Pressopt

For the library created with MATLAB COMPILER I have several files:

-libname.h is necessary to compile.
-libname.lib is necessary to link.
-libname.dll is necessary to the execution.

For my program, I have a file.c and a file.h which contains the functions which are used in .c

As my program use a .DLL, I looked for how using a DLL in C/C++ file.

There are two ways to use a dll in a project (depend of the file with the DLL we have):
-The more simple, we use the .h and the .lib of the dll to link with it. The .h describes the interfaces and the functions, the .lib is a library almost empty which allow to create the link. In this way, the dll is loaded by the loader of the program during the execution, the loader knows which dll it has to load because this information is in the .lib.

-The second way is when we have only the .dll and no .lib (we can have the .h or not). In this case, there is no link edition with the library almost empty (because we haven't this) and the program has to use Loadlibrary() function and then GetProcAddress() to get the adresses of functions to call in the dll.

As MATLAB Compiler provides the .h, the .lib and the .dll, I choose to use the more simple way.

In order to import the library from MATLAB to Borland (C++ builder environment), I have to use the implib function to generate the .lib from the .dll which is compatible with Borland. Indeed, there is a difference between the LIBfile format for Microsoft Visual C++ and C++ builder. Both vendors use different exporting conventions. Microsoft supports the Common Object Format File (COFF), whereas Borland uses the Object Model Format(OMF).

Then to deploy my program using the DLL, I just have have to have the DLL in the same directory than the executable created by the Pressopt project with C++ builder.

3.First results.

As first experiments, I launched my program for two different numbers of initialization points. This number is important because we construct the first model with this initialization points and it ´s obvious that more points we have and better will be the model.
The algorithm works good until 6 dimension [ref 4], but it has never been tested in more dimension than six and for a simulation based optimization problem.
So one important part of my project is to launch the program containing the algorithm to see the algorithm behaviour for this simulation based optimization problem in 10 dimension.

3.1 25 initialization points.

In a first time, I launch my program with 25 initialization points. Below,the values of the 10 parameters for each evaluation and the results of the simulation for 120 newpoints after the 25 first initialization points.

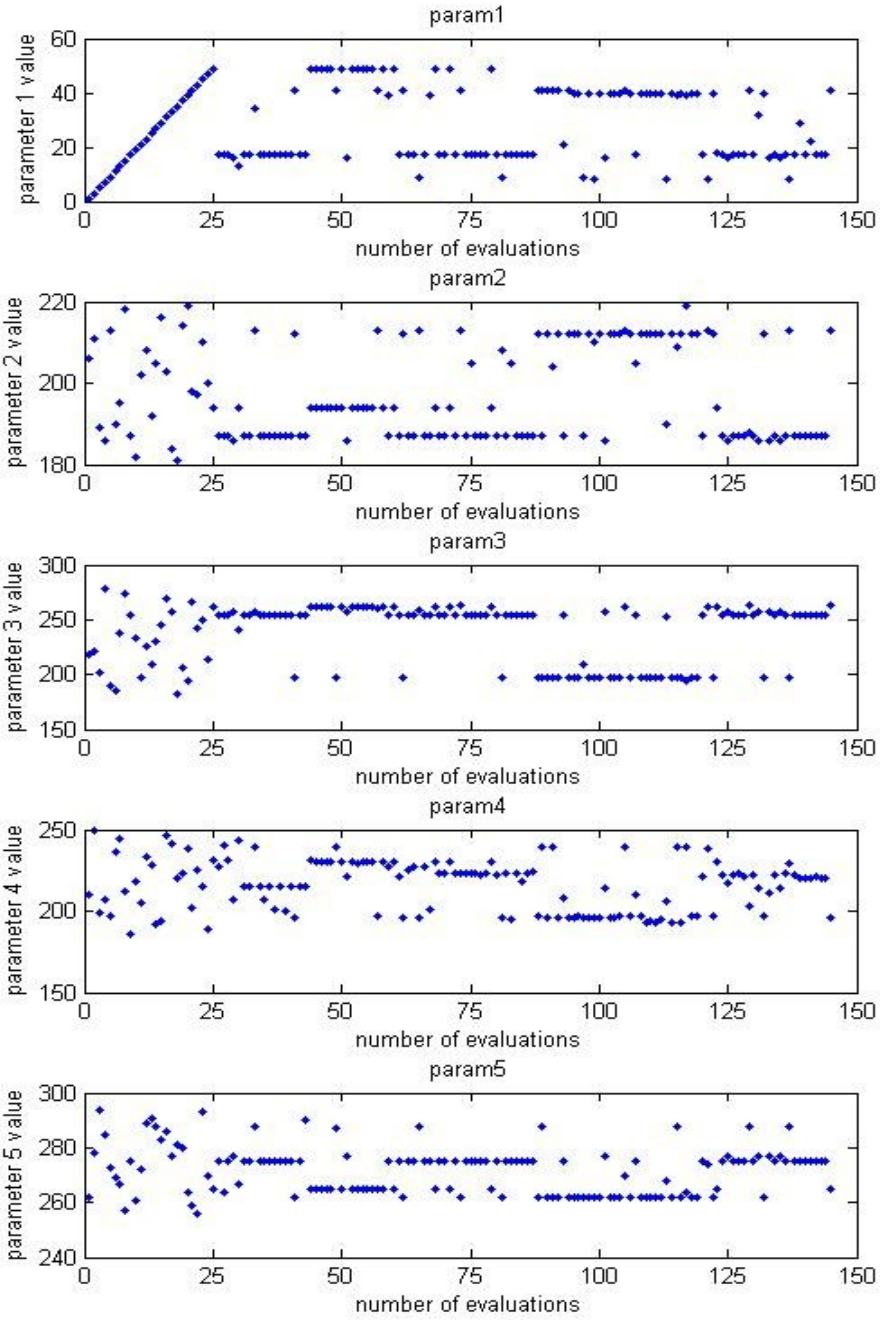


Fig10 : parameter values for the first experiment

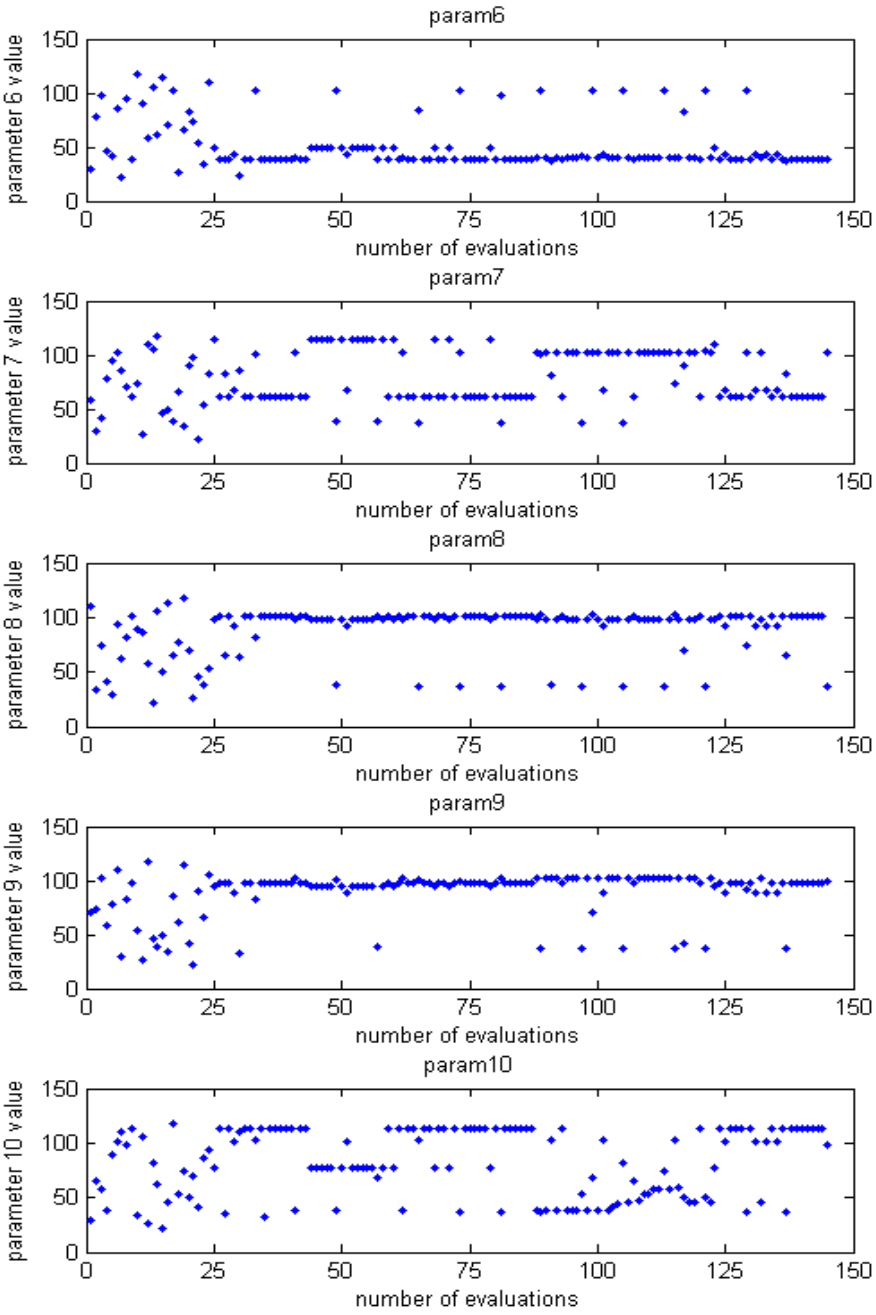


Fig10 : parameter values for the first experiment

The 25 first values for the 10 parameters represents the Design Of Experiments Latin Hyper Cube which is getting in maximizing the minimum distance between the points(see annex). These 25 first parameter values for the parameter 1 are typical of the Latin Hyper Cube design. These initialization values are used to build a model, we need to have as much information from the multidimensional space as possible to look everywhere in the space with these 25 points.

After the initialization points, the algorithm let some parameter value unchanged during several evaluations and even sometimes doesn´t change the value of the parameter or a little. It means that the search for the optimum is more local than global. Indeed, when the algorithm has found an interesting point, it continues to look around this point to see if it finds a better point. But the problem is that we are more interested in a global search than in a local search and make a too much local search takes too much time to not learn a lot of things about the objective function. We can gather all the points in 5 point groups :
group 1 between 25 and 45
group 2 between 45 and 60
group 3 between 60 and 85
group 4 between 85 and 120
group 5 between 120 and 145
We see in some parameter values that their values stay constant during some evaluations. The group 1, group 2 and group 3 have the same point value than the best point found during the Design Of Experiment. The algorithm always comes back to the best point found in the DOE. Firstly, it explores the group 1 to see if around this group of point there is an optimum. Then, it explores the group 2 but as it doesn´t find a better value than with the group 1 it comes back to the group 3 which is the same than the group 1. And we see the same thing with the group 4 and 5, the group being the same than the group 1. The parameter sigma which allow the balance between local and global search is responsible of this behaviour but the fact that we use integer for the simulation and double with the functions in the library make lost time to the algorithm. Indeed, sometimes the algorithm return two or more points which are not the same in double but the same in integer. And so we SimulateOneStroke for the same parameter combination value and we learn any information about the objective function.

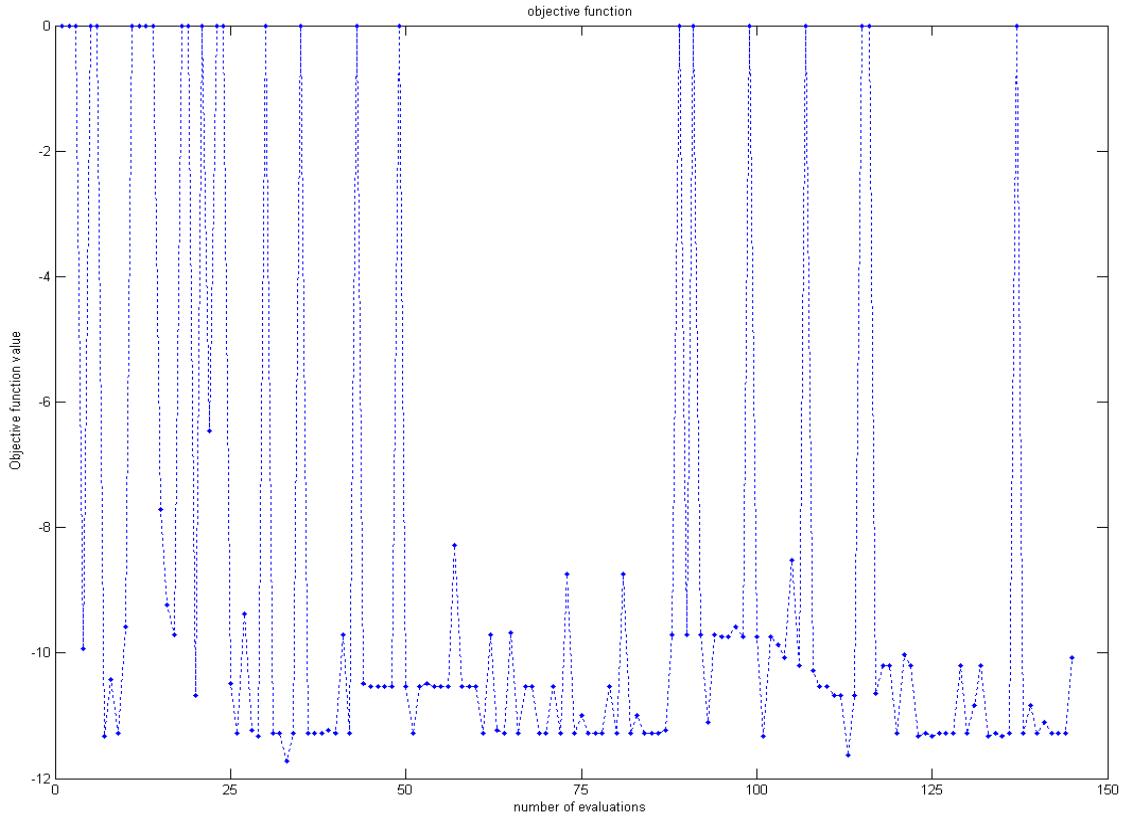


Fig11 : objective function with 25 initializating points

We see on the fig 11 above that the best value found is -11.7187 at the 32nd evaluation. And we see on this picture the same thing than with the parameter value, the search seems more local than global and the algorithm look too much around points and doesn't explore all the search space. The Design Of Experiment has found two points with the values -11.3 at the 6th and 8th evaluations and the algorithm then look too much around this points. It seems explores 5 groups of point in global search exactly the same with some of the parameters value and then look around these group of points in local search.

The group 1, group 3 and the group 5 have the same objective function value than the best point value in the DOE. So the values got with the DOE affect too much the newpoint function, the process to find a newpoint use more a local search than a global search and particularly use too much the best point already found. It should look in all the space to find the optimum.
When the algorithm look around one point (local search), for example in group 1, it changes only one or two parameter values between each newpoint and that ´s why the search is too much local.

We can see for some points 0 as the value of the objective function. This value indicates that there is a collision in the sheet metal press line and we aren´t interesting of these collision points. The algorithm minimize the objective function so as the bigger value found is -4 I chose to put 0 as the value of a collision point. When the model is built, all the collision point correspond to a peak so we aren´t sure of the algorithm behaviour (which build a surface) with all these peaks. On this experiment, in the DOE we get 14 collision points on the 25 initialization points, so more than half of the initialization point are collision, then the model built is very bad and can maybe explain the bad behaviour of the algorithm.
As the algorithm build a surface, if we have a lot of peak the surface will be very bad.
That ´s why I make a new test with 50 initialization points.
I said above that there is a parameter inside the algorithm, sigma, which allows to change the balance between local and global search. The algorithm, for these first experiments uses a list of sigma values which makes a balance but this balance seems not very efficient in this case. So it will be interesting to change this sigma list to see if there is a difference, and to focus more in global search.

3.2 50 initialization points.

With the problem saw with the first experiment I chose to launch the simulation for 50 initialization points in order to have a better model and 100 newpoints.
Below the parameter values for the 10 parameters at each evaluation and the objective function value.

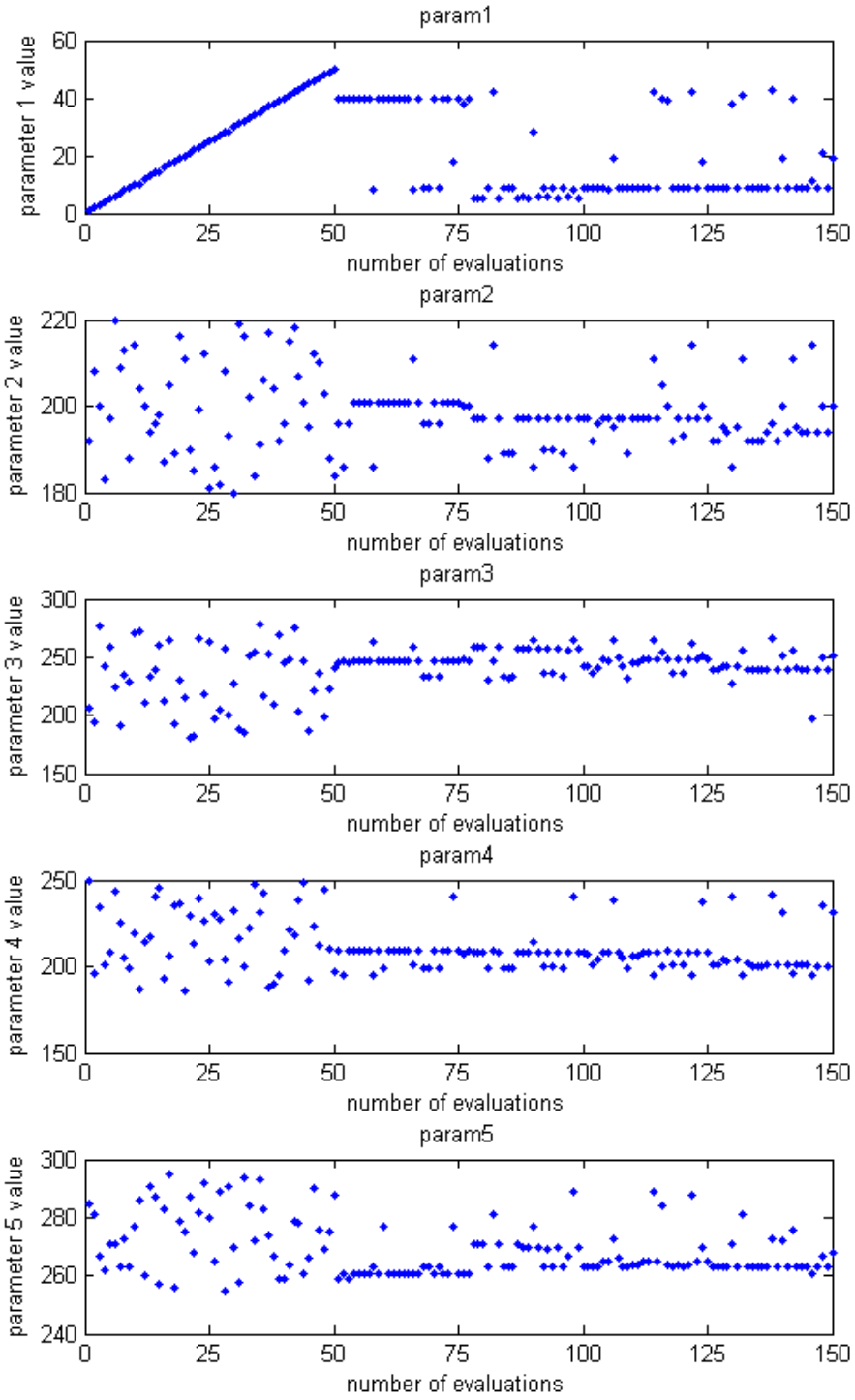


Fig12 : parameter values for the second experiment

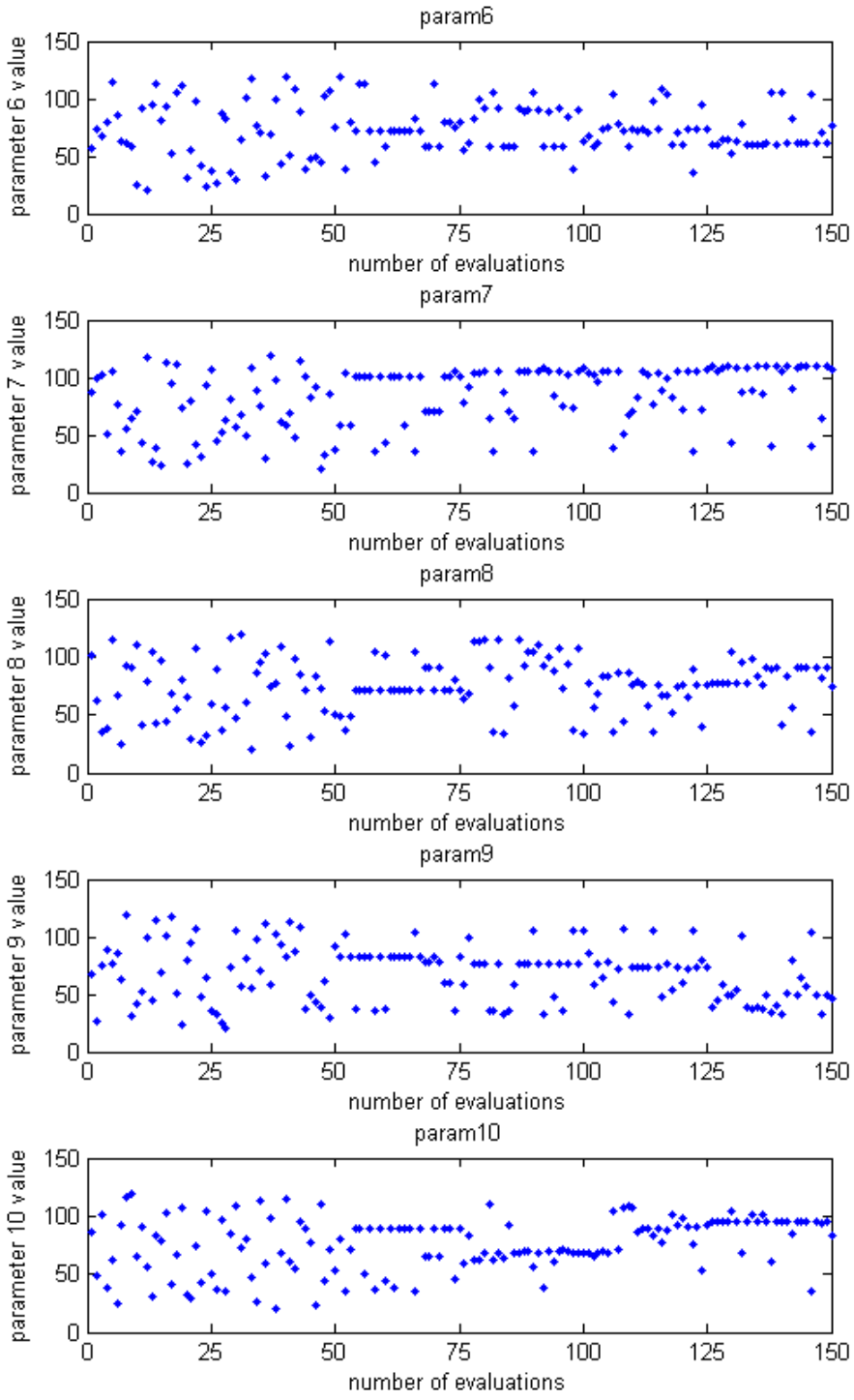


Fig12 : parameter values for the second experiment

The first 50 points values for the 10 parameters represents the Design Of Experiments Latin Hyper Cube .
We see the same behaviour than with 25 initialization points for the first 50 points values of the first parameter which is typical of Latin Hyper Cube design.

The results are different than with 25 initialization points, indeed the parameter values don´t make some step as in the first experiment. But for some of them their values don´t change a lot (param4 param3 for example) and for the other it seems that it explores better all the search space than with a model built with 25 initialisation points. The parameter values change more than with 25 points, so the behaviour of the algorithm seems to improve when the model is construct with more point. But this is what I expected because with the 50 initialization points the model is better than with 25 points initialization points. Most of the parameter values explore an area bigger than in the first experiment. But it is right that there are also some steps or lines for the parameter values, so there is still a local search which is present but less than in the first experiment. We need a local search but just few evaluation for the local search. The goal is to find a compromise between the global and the local search. We have to do a global search to find the global optimum but this optimum global is hard to find and the local search look for more accurately than the global search. So we need to use a local search but in the same time we want to find the global optimum and if the local search use too much time in an area where in fact there isn't the global optimum we will lost too much time. This is the problem.

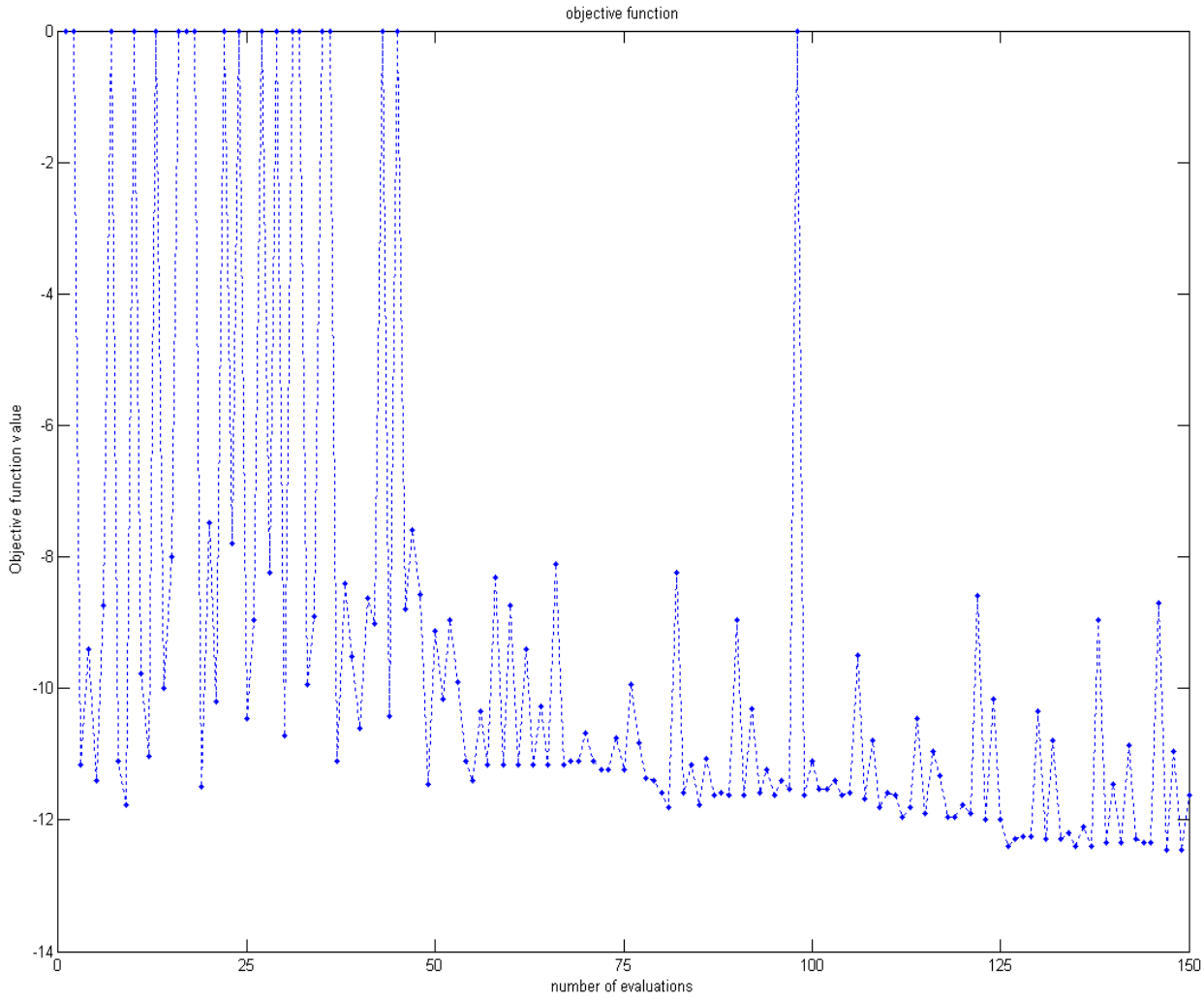


Fig13 : objective function with 50 initialization points

We see on the figure above that the best point found is -12.4481 at the 147th and 149th evaluation. So if we compare just the results of the best point, we have found a better point with the second experiment. We also see that the algorithm behaviour is quite good because it continually improves the results. So more we have points to construct the first model and more the algorithm behaviour seems good. And in the 50 first points, we have 18 collision points which is less than half of the initialization points. As the parameter values changes more than in the first experiment, the search space is better explored but as said above , the search space is very huge.

Conclusion for the two experiments :

I chose to launch only 145 points in the first test and 150 in the second because the calculation time just to get the coordinate of a newpoint increase with the number of evaluation and it was my first test just to see the behaviour of the algorithm. But 150 or 145 points are very small number of points in comparison with the several billion of possibilities in the search space. Actually the best point found with all the methods already implemented is -13.1004 after several thousand of evaluation. We have seen that more we have initialization points and better is the algorithm behaviour so a test with 100 initialization points will be interested to do. But in all the cases, I think the algorithm has to be improved with better parameters algorithm(as the sigma value for example). Indeed, this algorithm has the same parameter value identical as when it has been used in 3 or 4 dimension.

3.3 calculation time

An other problem met with this algorithm is the calculation time to get the coordinate of a newpoint. Indeed, the newpoint function use all the points previously calculated and after a certain number of evaluation, the calculation time become enough important. One evaluation takes 10 minutes and for example after 900 newpoints , we need more than 2 min to get a newpoint with the algorithm. The goal, as said before, is to keep the total optimization time t_{opt} down to practical useful times, i.e. days or weeks. If we need several minutes just to make the calculation and we need 10 minutes more for the simulation, the method will take too much time. But as it´s a new algorithm for this kind of optimization problem, maybe it needs less evaluation than the other simulation based optimization method already implemented like Nelder Mead or Direct. For example, Direct need several thousand of evaluation to find the optimum of the objective function. It ´s impossible to launch the algorithm that I use for several thousands evaluations as show below.

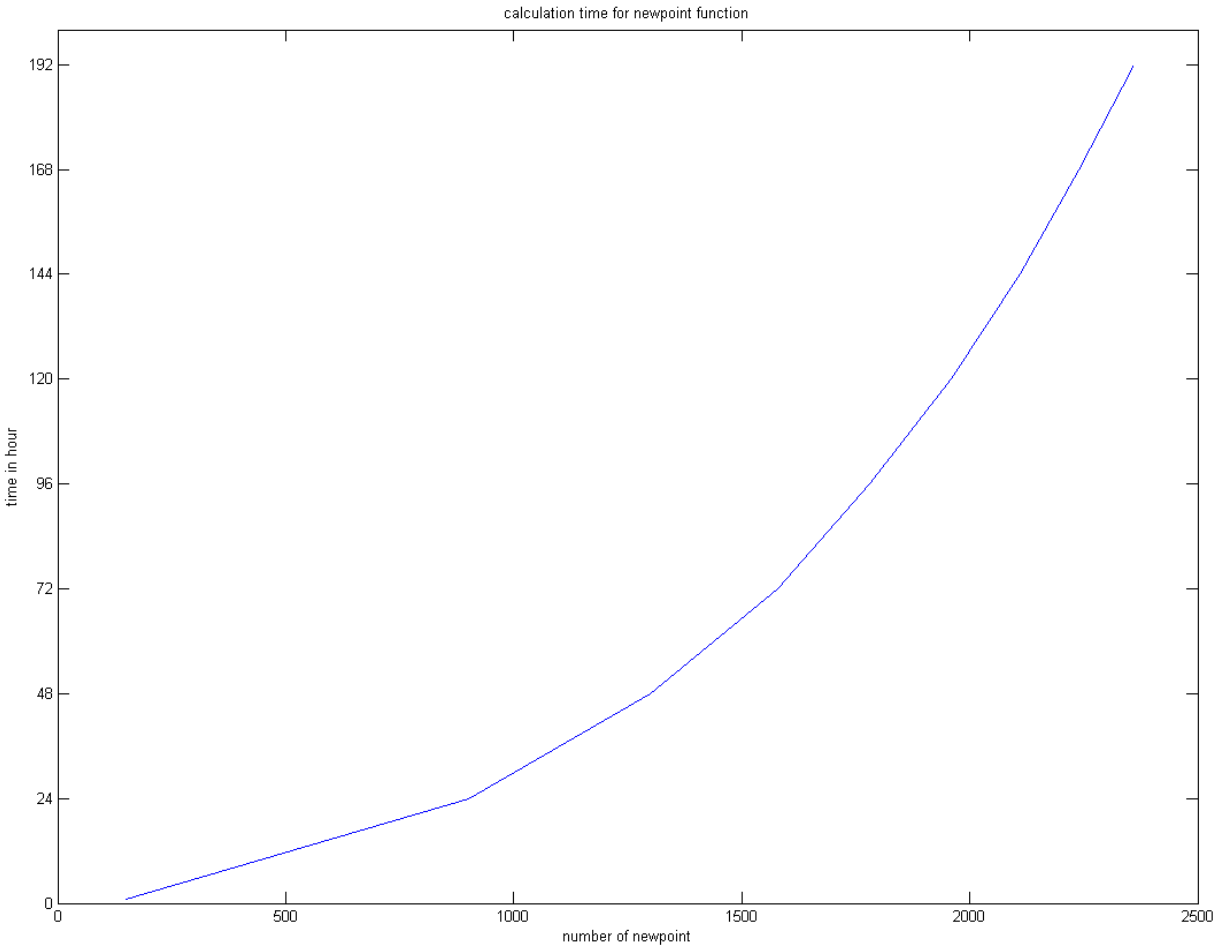


Fig14 : calculation time for the newpoint function

We see on this picture that the calculation time become very important for several thousands of evaluations and the time given is just the calculation time without the simulation .I made this experiment in simulating the value returned by the VM. Seeing this problem, we define that the calculation time must be at maximum 2/10 of simulation time, it means 2 min maximum. We can choose this because we know that in the future the calculation power of the computer will increase. We cannot launch the simulation for more than 900 newpoints.

Conclusion :

The main objective of this project was to create a kind of interface between MATLAB Code and C/C++ Code in order to be able to use an algorithm which has been created with MATLAB in the process optimizer Pressopt. This process optimizer developed by the Virtual Manufacturing research group of the University West works and has already been used for three different optimization methods (Screening, Nelder Mead, Direct).

Firstly, I focused my study on the understanding of simulation based optimization. I saw the difficulty of this kind of optimization problem and why we cannot use “classical” optimization method.
Secondly, I learned how work the algorithm that I have to use in the process optimizer but not in details and I also learned about RSM to solve optimization problem.
Then, I focused my work on how interface MATLAB code and C/C++ code.
And then, I embed my program using the algorithm in Pressopt.

After some experiments, I saw that the algorithm wasn´t still ready for this simulation based optimization problem in 10 dimension. In the algorithm there are some parameters that we can tune to change the search type for the optimum. There is a need to improve the algorithm and so I have to learn it more in details. One idea is to use the database in order to print the objective function value in remaining constant 12 parameters and changed only 2 and make the plot of all these results to see in three dimension the behaviour of the objective function. Knowing this behaviour could help the tune of the algorithm parameters and also would allow to learn more things about this function.
With the time calculation problem, it 's really impossible to launch my program for more than 1000 newpoints and even 500 newpoints will take half of a day just to the calculation time. One solution could be to launch the algorithm for 400 newpoints for example and then with the results launch again the algorithm in searching more in special area (then we will decrease the research space). This will consist in making a double loop with two launches of the algorithm.
As my program works, the next part of my project will be to improve the algorithm and try to do some changes in the algorithm as the method used to calculate the distance between two points or the method to sample the search space (gridsamp function) or also the sigma value or even the interpolating method. And of course, I will change my program in consequence of these changes.

Bibliography :

ref 1. Bengt Lennartson, Bo Svensson and Fredrik Danielsson, Simulation Based Optimization of a Sheet-Metal Press Line, 2009.

ref 2. Henrik Carlsson, Bo Svensson, Fredrik Danielsson, A General Virtual Manufacturing Concept for Programming Verification and Optimisation of Complex Control Functions, 2007.

ref 3. Bo Svensson, Fredrik Danielsson and Bengt Lennartson , Off-Line Optimisation of Complex Automated Production Lines - Applied on a Sheet-Metal Press Line, 2007.

ref 4. Lindström, D. & Eriksson. A Surrogate Model Based Global Optimization Method. Proceedings of the 38th CIE Conference, China., 2008.

ref 5. Project Report Implementation of direct method for Virtual Manufacturing, Marc Emilien Chauvin, 2008.

ref 6. Anna Persson, Marcus Andersson, Henrik Grimm, and Amos Ng, Metamodel-Assisted Simulation-Based Optimization of a Real-World Manufacturing Problem.

ref 7. Huai Zhang**,** Zhibin Jiang and Chengtao Guo, Simulation-based optimization of dispatching rules for semiconductor wafer fabrication system scheduling by the response surface methodology, 2007.

ref 8. XIAOTAO WAN., PEKNY Joseph F. and REKLAITIS Gintaras V., Simulation-based optimization with surrogate models : Application to supply chain management, 2005.

ref 9. Anna Syberfeldt, Henrik Grimm, Amos Ng, Martin Andersson, Ingemar Karlsson, Simulation-Based Optimization of a Complex Mail Transportation Network, 2008.

ref 10. Yuehua Gao and Xicheng Wang, Surrogate-based process optimization for reducing warpage in injection molding, 2007.

ref 11. Kazuhiro Takamizawa, Shinobu Nakashima, Yuuichi Yahashi, Kilunga B. Kubata, Tohru Suzuki, Keiichi Kawa and Hiroyuki Horitsu, Optimization of Kojic Acid Production Rate Using the Box-Wilson Method, 1996.

ref 12. Morris, M. D. and Mitchell, T. J. , Exploratory Designs for Computer Experiments, J. Statist. Plann. Inference 43, 381-402, 1995.

ref 13. Tang, B. , Selecting Latin hypercubes using correlation criteria, 1998.

Annex:

The Latin hypercube design is a popular choice of experimental design when computer simulation is used to study a physical process. An LHD has the property that by projecting an *n*-point design on to any factor, we will get *n* different levels for that factor. This property makes an LHD very suitable for computer experimentation.

Suppose the *n* levels of a factor are denoted by 1*; 2; ... ; n*. Figure 1a shows an LHD with two factors in six points. In general, an *n*-run LHD can be generated using a random permutation of *{*1*; 2; ... ; n }* for each factor. Each permutation leads to a different LHD. For *k* factors, we can thus obtain $(n!)^k$ LHDs. Figure 1b shows one such LHD. Clearly, this is not a good design. It is not good due to the following two reasons. First, the two factors are perfectly correlated. Therefore, we will not be able to distinguish between the effects of the two factors based on this experiment. Second, there is a large area in the experimental region that is not explored. Therefore, if we use such a design to develop a prediction model, then the prediction will be poor in those unexplored areas.

There has been some work in the literature to avoid the above problems and obtain a good" LHD. The idea is to find the best design by optimizing a criterion that describes a desirable property of the design. Iman and Conover (1982), Owen (1994), and Tang (1998) proposed to find designs minimizing correlations among factors. Figure 1c shows the optimal LHD found by the procedure in Tang (1998), which is clearly much better than the one in Figure 1a and 1b. Apart from the correlations we are also interested in spreading the points out across the experimental region. This is the idea behind space filling designs. Morris and Mitchell (1995) proposed to find the best LHD by maximizing the minimum distance between the points. The optimal LHD under this criterion is shown in Figure 1d.