Stress Model Calibration

Project report

submitted to

Mentor Graphics

Aram Jivanyan   Nare Karapetyan

Gurgen Khachatryan   Perouz Taslakian

November 3, 2015

Contents

Chapter 1

Introduction

Calibration is an optimization process that searches through all possible values of given parameters in order to obtain the ones that evaluates the given objective function to a target value. The quality of the calibration is based on the number of required function evaluations for identifying the optimal or near-optimal solution, and the accuracy of the solution.

The aim of this project is to calibrate a specified set of parameters for the stress modelling tool of integrated circuits. The tool simulates mechanical stress distribution across a circuit, extracts values of the stress components, and transforms them into variation of electric current I. The set of parameters P must be chosen so that they minimize the difference of calculated electric currents I and the desired (measured) value $I_{meas} : |I - I_{meas}|$. To compute electric current displacement, parameter u must be computed from the linear algebraic equation, where the coefficients of u are from the subset of parameters P. The problem at hand is a non-linear minimization problem, which is also derivative-free as the derivative of the objective function is not always availabe.

The two main requirements of the optimization algorithm for our problem are efficiency and optimality. Our algorithm must converge to a solution after a small number of iterations and hence, *reasonable* runtime; and the global optimum solution it finds must be at most 0.05 times the theoretical minimum (hence, providing a precision of 5%). The previously applied optimization algorithms was an implementation of the *Nelder Mead* algorithm that was part of the *tcl* library. This algorithm is fast, but most of the time it returns the local minimum of the target optimization function, and thus has a high error margin. One promising implementation of *Nelder*

Mead, which was applied for optimizing the described calibration model, is the algorithm used in *Mathematica*s *FindArgMin* function and gave better results in general. However, as *Mathematica* is not open source, the use of this software becomes inapplicable for solving the current problem.

In this project we define the problem as a derivative-free global optimization problem. The remainder of this report is organized as follows. An overview of recent work in the field is presented Chapeter 2. The problem formulation is described in Chapter 3. Chapter 4 gives an overview of the optimization algorithm applied, and finally the results are summarized in Chapter 5.

Chapter 2

Related Work

For the last decade, with a growing application domain, optimization problems have received particular interest and attention. Extensive research has been done in the field, and numerous software packages have become available for solving optimization problems of various flavors. The approaches are diverse and sometimes domain-specific, but optimization problems can mainly be categorized based on the output optimality – *local*, which converge to a local optimum, and *global*, which converge to a global optimum. Approaches to optimization algorithms are diverse also depending on the mathematical characteristics of the objective function. Some algorithms require that the objective function have a derivative; however, such a requirement cannot be imposed for complex functions for which derivatives are difficult to find (or non-existent). Optimization techniques that operate without the need for derivatives are called *derivative-free optimization algorithms.*

One state of the art derivative-free local optimization technique is the Nelder-Mead simplex [13] method. This method takes initial values for parameters as vertices of a simplex. Consequently, a set of transformations are applied to the simplex that is aimed to decrease the function values at vertices. A number of implementations are available in libraries for Matlab, Mathematica, tcl, SciPy [3] (python library), etc.

Another set of heuristic derivative-free algorithms is the set of *genetic algorithms.* These algorithms use techniques inspired from natural evolution. The general idea of genetic algroithms is to select some population of solutions and iteratively change it, ensuring improved fitness [10]. Fitness is the value of the objective function and the population is the values of the calibration parameters. Genetic algorithms operate by applying *selection*, *crossover*, and *mutation* operations to generate the next population. In each generation selection, the operator makes fitness-based decisions for selecting a population. The next child population is produced from a previously selected pair of populations (crossovers) or by random modification of the current population (mutation). Many genetic algorithms tend to converge toward a local minimum/maximum or even to an arbitrary point. Another significant drawback of this method is the convergence rate [10]. Even though genetic algorithms are not efficient for finding global optima, in combination with other local optimization algorithms they can find near optimal and even optimal solutions [9] with reasonable function evaluations. Duran and Alliot [8] (experimentally) show that, in some cases, a combination of the Nelder-Mead simplex algorithm [13] and genetic algorithms works better than using each algorithm independently to find a global optimum. Genetic algorithms help in identifying the region of a potential global optimum, while Nelder-Mead finds the this optimum with a significantly less function evaluations.

Another hybrid optimization is proposed by Custodio and Madeira [7], and is called *Global and Local Optimization using Direct search (GLODS).* The algorithm operates on objective functions for which the existence of a global minimum is ensured in the region of interest. The global optimum is computed from all the local minimizers, which represents a multi-start approach. As a classical direct search method, GLODS also consist of search and poll steps, but instead of searching around an initial point, it generates a set of points, and for each of them the direct minimizers are applied. Two direct minimizers are merged only if they are in defined appropriate distance. This algorithm is implemented in Matlab [1], and distributed under the GNU LGPL license. However, the algorithm assumes that the number of local optimums are known [7], thus if it is applied to a function with an unknown number of local optima, efficiency will be reduced. On the other hand, if a small number of optima are specified, the algorithm might fail in finding a global minimum.

Sometimes it is difficult and expensive to evaluate objective functions. In such situation, it is possible to approximate the objective by another function – a surrogate one. This method is called function approximation or *surrogate method*. In the work of Mugunthan et al. [12], the function approximation (FA) method was compared with the heuristic and derivative-based nonlinear optimization method for automatic calibration of biokinetic parameters of a groundwater bioremediation model. Two variations of genetic algorithms, binary-coded and real-coded, are used as heuristic algorithms. As a derivative-based optimization, they use the sequential quadratic method

FMINCON implementation from Matlab [1]. Two versions of function approximation methods was tested by Mugunthan et al. [12].

The general idea of function approximation algorithms is to create a surrogate function for the objective function, such that it reasonably mimics the behaviour of the objective function, while at same time is computationally much cheaper to optimize. Both algorithms presented in this work use radial basis function to approximate the objective function [12]. The research shows that FA methods are more reliable for computationally expensive model simulations [12].

In 2013, a complete review on Derivative-free optimization algorithms and software packages was presented by Rios and Sahinidis [16], with an efficiency and optimality comparison. They tested a set of global and local optimizations over $100,000$ problem instances, and the results show that OQNLP and MULTMIN solvers from the TOMLAB library and the NEWUOA solver outperform all other implementations in terms of finding a near-optimal solution. TOMLAB [11] is a commercial environment providing an optimization platform for solving problems in Matlab. OQNLP and MULTIMIN represent a multi-start approach, that uses a non linear programming solver in the search step. NEWUOA is an unconstrained optimization based on a function approximation approach introduced by Powell [15], and it is implemented in Fortran. The LINCOA [14] software package developed by Powell, for solving linearly constrained optimization problems, is also available in Fortran. Similar to NEWUOA algorithm, LINCOA is also an iterative trust region method [14].

For the purposes of this work, some algorithms having a C++ implementation were also considered; these include DIRECT from Dacota library, TSENG and others [16]. The current project was completed using a modified version of the NEWUOA algorithm for performing optimization with bounded parameters.

Chapter 3

Problem Formulation

The developed model simulates mechanical stress distribution across an integrated circuit, extracts the values of the stress components in the regions of transistor channels, and transforms these stresses into variation of electric current for each transistor. The calibration is performed for two types of transistors. The model contains a number of parameters specific for each type of transistor and some common as well. The aim of this project is to properly calibrate these parameters.

This chapter presents a detailed description of the formulation of the *stress model calibration* problem. In Section (3.1), the representation of the input is described. We then formulate a mathematical model for the stress calibration tool in Section (3.2); finally, the problem requirements are presented.

3.1 Input Data

The main idea of the model underlying the required calibration engine is the following. Each transistor is characterized by a surrounding that can be represented as a sequence of two materials silicon (Si) and silicon dioxide (SiO_2). For each transistor channel, the surrounding is extracted for two orthogonal directions; see schematics in Figure (3.1)(a). Two *cut-lines* (in the x and y direction) are generated for each transistor. The cut-lines on each direction represent material segments. The details of the cut-line are represented as a data array, as described in Figure (3.1) for the x cut-line.

The i-th segment of the cut-line is characterized by the coordinate of the left node $x = X_i$. The segment length is L_i, which will be equal to $X_{i+1} - X_i$ and L_{ch} is the length of transistor channel (3.1)(b). An *initial strain* ε^{init} is assigned to each node. In addition, for each segment the corresponding material is defined.

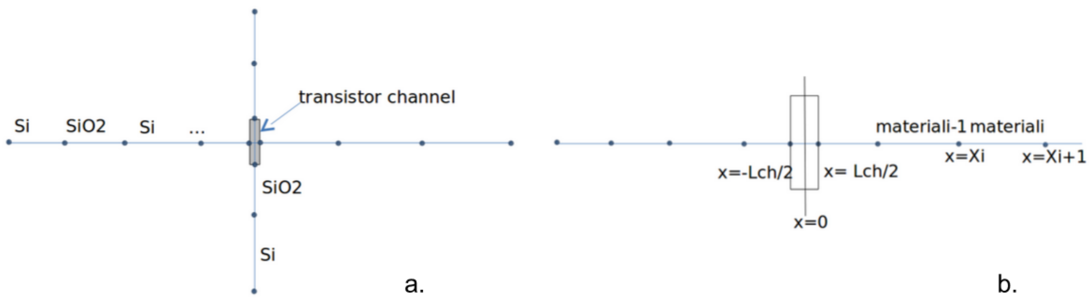

Figure 3.1: The materials surrounding a transistor are represented by cut-lines in the x and y directions.

The cut-line for the y direction is constructed similarly. Thus, the input formulation for each transistor is as follows:

$$\begin{gathered}\{name\ type\ L_{ch}W_{ch}\{dir_x w_1\{X_1 material_i \ldots X_i material_i \ldots X_k end\}\{\varepsilon^{init}_{x_1} \ldots \varepsilon^{init}_{x_i} \ldots \varepsilon^{init}_{x_k}\}\{X_0 Y_0\}dir\}\\ dir_x w_2\{X_1 material_i \ldots X_i material_i \ldots X_k end\}\{\varepsilon^{init}_{x_1} \ldots \varepsilon^{init}_{x_i} \ldots \varepsilon^{init}_{x_k}\}\{X_0 Y_0\}dir\} \ldots\\ dir_y w_1\{Y_1 material_i \ldots Y_i material_i \ldots Y_n\ end\}\{\varepsilon^{init}_{y_1} \ldots \varepsilon^{init}_{y_i} \ldots \varepsilon^{init}_{y_n}\}\{X_0 Y_0\}dir\}\\ dir_y w_2\{Y_1 material_i \ldots Y_i material_i \ldots Y_n\ end\}\ \varepsilon^{init}_{y_1} \ldots \varepsilon^{init}_{y_i} \ldots \varepsilon^{init}_{y_n}\}\{X_0 Y_0\}dir\} \ldots \varepsilon^{init}_{zch} dir\}\end{gathered} \tag{3.1}$$

Each variable is represented by some convention and has the following physical meaning.

- $\{X_0 Y_0\}$ are the coordinates of the cut-line center point.
- *dir* relates to the transistor orientation/direction. By convention, 40 stands for dir_x and 43 stands for dir_y.
- *name* is a list containing two numbers x, y.
- *type* denotes a transistor type: "0" means type $= N$, "1" means type $= P$.

- In the sub-list $\{X_i material_i \dots\}$, that represents surrounding materials, the following mapping is used: "1" maps to SiO_2, "6" maps to SI. The end of cut-line is identified by *end*, which mapped by "66".
- L_{ch} and W_{ch} are the channel length and width; dir_x and dir_y denote the cut-lines orientation. The segment numbers can be different in the x and y directions; also they can be different for different transistors. The initial strain x and y components $\varepsilon_{x_i}^{init}$ and $\varepsilon_{y_i}^{init}$ are defined for each node of the cut-lines; the z component ε_{zch}^{init} is defined only for the channel center.
- Both x and y directions can be described by several cut-lines. Each cut-line is characterized by a weigh factor $w_i \leq 1$.

The input data consists of two files. One file contains the cut-lines for a set of transistors as described above. The second file contains the measured values for stress-induced current variation for each transistor ΔI, and a parameter $diff_I$:

$$\{name \Delta I^{meas} diff_I\} \tag{3.2}$$

3.2 Stress Model Calibration

The following section describes the mathematical model of stress calibration tool in terms of the parameters defined in Section (3.1). The problem requirements are presented at the end.

3.2.1 Calculation of Electric Current Variation

In order to get the required value of ΔI, we perform the following steps.

(a) **Calculation of displacements in cut-line nodes.** ($u = u_x$ for $x-$direction, and $u = u_y$ for $y-$direction).

Each segment in the cut-line (3.1) is characterized by $material_i$. The property of material is *Youngs modulus* E_i (i.e. $E_{Si} \vee E_{SiO_2}$); the segment length is $L_i = X_{i+1} - X_i$. A displacement u_i is prescribed to each node of the cut-lines; these displacements are obtained from the following set of equations.

$$E_i^{''} u_{i-1} - (E_i^{'} + E_{i+1}^{'}) u_i + E_{i+1}^{''} u_{i+1} = (E_i - E_{i+1}) \varepsilon_{x_i}^{init} \tag{3.3}$$

where

$$E_i^{'} = \frac{E_i + E_{Si}}{2H_i}\frac{1}{\tanh\frac{L_i}{H_i}}, E_i^{"} = \frac{E_i + E_{Si}}{2H_i}\frac{1}{\sinh\frac{L_i}{H_i}}, H_i = h\sqrt{2\frac{\alpha E_i + \alpha^2 E_{Si}}{E_{Si}}}. \tag{3.4}$$

The parameters $E_{Si}, E_{SiO_2}, h, \alpha$ are calibration parameters. To solve equation (3.3), the following condition is imposed: $u_i = 0; u_{end} = 0$. The displacements at the channel edges should be obtained as follows.

$$\begin{aligned} &\text{for } x \text{ direction } u_{x_0} = u_x(x = -L_{ch}/2), u_{x_1} = u_x(x = L_{ch}/2) \\ &\text{for } y \text{ direction } u_{y_0} = u_y(y = -W_{ch}/2), u_{y_1} = u_y(y = W_{ch}/2) \end{aligned} \tag{3.5}$$

(b) **Calculation of strain components in transistor channel.**

Using the calculated displacements at the channel edges (3.5), as well as values of initial strain at the same nodes $\varepsilon_{x_0}^{init}, \varepsilon_{x_1}^{init}, \varepsilon_{y_0}^{init}, \varepsilon_{y_1}^{init}$, and ε_{zch}^{init} from the cut-line (3.1), the following strain values are calculated:

$$\varepsilon_x = \frac{u_{x_1} - u_{x_1}}{L_{ch}} + \frac{\varepsilon_{x_0}^{init} + \varepsilon_{x_1}^{init}}{2}, \varepsilon_y = \frac{u_{y_1} - u_{y_0}}{W_{ch}} + \frac{\varepsilon_{y_0}^{init} + \varepsilon_{y_1}^{init}}{2}, \varepsilon_z = \varepsilon_{zch}^{init} \tag{3.6}$$

The procedures (3.3) and (3.6) are applied for calculation of strain components for each cut-line (i.e. a number of values $\varepsilon_i^x, \varepsilon_i^y$ will be obtained). Finally, average values of x and y strains are obtained as follows.

$$\varepsilon_x = \sum w_j \varepsilon_x^i, \qquad \varepsilon_y = \sum w_j \varepsilon_y^i \tag{3.7}$$

(c) **Calculation of stress components in transistor channel**. The calculated strain components (3.6) are used for calculation of stresses:

$$\begin{aligned} \sigma_x &= \frac{E_{Si}}{(1 + v_{Si})(1 - 2v_{Si})}[(1 - v_{Si})\varepsilon_x + v_{Si}(\varepsilon_y + \varepsilon_z)], \\ \sigma_y &= \frac{E_{Si}}{(1 + v_{Si})(1 - 2v_{Si})}[(1 - v_{Si})\varepsilon_y + v_{Si}(\varepsilon_x + \varepsilon_z)], \\ \sigma_x &= \frac{E_{Si}}{(1 + v_{Si})(1 - 2v_{Si})}[(1 - v_{Si})\varepsilon_z + v_{Si}(\varepsilon_x + \varepsilon_y)] \end{aligned} \tag{3.8}$$

Parameter v_{Si} is a calibration parameter.

(d) **Calculation of electric current variation.** Current variation is a linear combination of the stresses (3.7); the involved parameters are different for N and P type transistors.

$$\Delta I_T = diff_I(R_T - (\pi_x^T \sigma_x + \pi_y^T \sigma_y + \pi_z^T \sigma_z)) \tag{3.9}$$

where $T \in N, P$. The parameters $R_T, \pi_x^T, \pi_y^T, \pi_z^T$ are calibration parameters. The parameter $diff_I$ is defined for each transistor in the input file (3.2).

3.2.2 Model calibration

The calibration parameters for the presented model that should be optimized are the followings.
$E_{SI}, E_{SiO_2}, v_{Si}, h, \alpha; R_N, R_p, \pi_x^N, \pi_y^N, \pi_z^N, \pi_x^P, \pi_y^P, \pi_z^P$.

This optimization is performed by minimization of difference of calculated (3.8) and (3.2) measured values of current variation, i.e. through search of minimum root mean square deviation $\sqrt{\sum(\Delta I - \Delta I^{meas})^2}$. The optimization algorithm must converge to the global optimum with 5% precision and with small number of iterations, such that the running time is reasonable. The required calibration engine should be able to find values of parameters within pre-defined ranges $p_i^{min} < p_i < p_i^{max}$ (constrained optimization). The ranges are presented in the following table 3.1.

E_{SI}	$[160, 180]$
E_{SiO_2}	$[60, 80]$
v_{Si}	$[0.2, 0.3]$
h	$[100, 300]$
α	$[0.1, 5]$
R_N	$[-0.1, 0.1]$
R_p	$[-0.1, 0.1]$
π_x^N	$[-1, -0.1]$
π_y^N	$[-0.6, -0.01]$
π_z^N	$[0.1, 1]$
π_x^P	$[0.1, 1]$
π_y^P	$[-1, -0.1]$
π_z^P	$[-0.05, 0]$

Table 3.1: Ranges for calibration parameters

Chapter 4

Proposed Solution

For this project the we define an optimization problem as a derivative-free optimization. As literature review 2 revealed one of the latest and near-optimal solution finding algorithm is NEWUOA [15], with an open source C++ implementation [2]. But the algorithm is works for unconstrained parameters, in contrast with the problem defined in the following report. To apply above mentioned alorithm to parameter calibration problem, we modified it in a way to handle constrained parameters as well. The following chapter presents an overview of the derivative free-optimization algorithm NEWUOA [15] and the modification method of representing it as a constrained optimization.

4.1 Algorithm Overview

NEWUOA is an approximation optimization algorithm based on trust region method, that finds the minimum value of a function when the latter can be calculated in any point. The basic steps of any trust region method is the followings [6]:

1. Build an initial trust region with the x_0 center.
2. Building an approximation $Q(x)$ model of objective function $F(x)$ around x_0 point, e.g. in the trust region.
3. The next x_k trial point in the region is examined, such that it decreases the $Q(x)$.

4. Reduction of objective function in x_k is compared with the model reduction, and if ratio is sufficiently positive x_k becomes new trust region center. It is possible also to update trust region radius.
5. Otherwise, if ratio is not sufficiently positive, trust region radius is decreased.
6. The next iteration continues from step 2 but around new trust region center instead of x_0.
7. The Process stops when there are no improvements of model.

Even though there is no clear evidence of the effectiveness of trust region methods over linear search, some trust region methods may lead to global optimum [4]. trust region methods not only minimize the value of the model, but also remain in the optimal solution region and provide a wider search range 4.1.

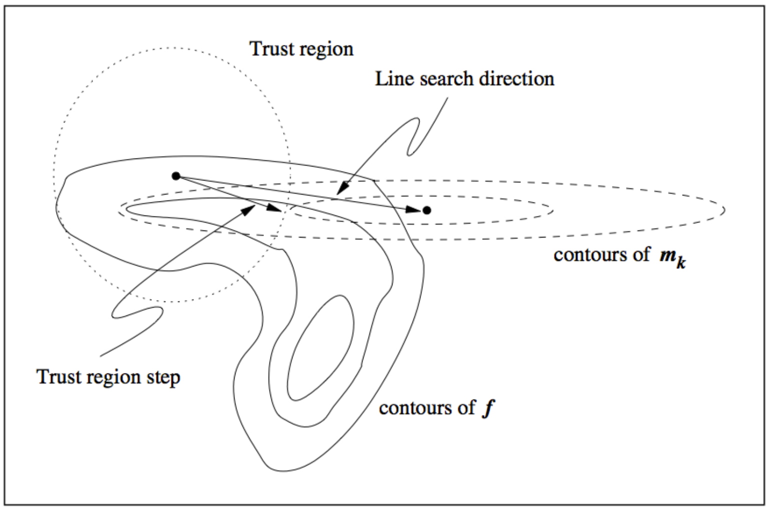

Figure 4.1: Geometric interpretation of trust region steps vs line search, where f is an objective function and m_k is the approximation model [5]

Algorithms based on the trust region method differ in the initialization and approximation function creation step. For regulating the parameters, the NEWUOA algorithm creates quadratic forms for $m = n{+}1$ points, where

n is the dimension of the parameters. The quadratic forms in unconstrained optimisations provide, as the literature suggests [15], a fast convergence rate. For more information on algorithm specification, we refer reader to the paper by Powell [15].

4.2 Bounded NEWUOA

The NEWUOA algorithm represents unconstrained derivative free optimization. But the stress calibration tool has a bound constrains on the parameters. To handle bounded parameters we introduce a modification to the original algorithm. As each parameter in NEWUOA can take values within the set of rational numbers, we project each value in the bounded region of specified parameter. The projection of the parameters into the $[a, b]$ segment is done in the following way 4.2 :

1. Build a rectangle that has $[a, b]$ as one of its sides and height equal to $(a - b)/2$, as $aABb$ rectangle in Figure 4.2.
2. Connect the vertices A and B with the midpoint of the segment $[a, b]$. (see the diagonals AO and BO in Figure'4.2).
3. Any point c on the left of the midpoint O of segment $[a, b]$ is connected to the midpoint C of segment AB.
4. The corresponding bounded value of c is the projection of the intersection point of line cC and a diagonal AO on segment $[a, O]$.
5. The second and third steps are symmetrically performed for the right side elements.

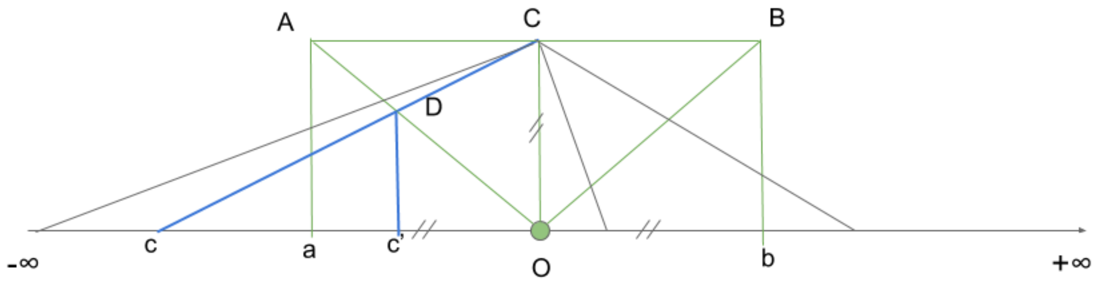

Figure 4.2: Projection of parameter values into the bounded region.

Chapter 5

Experimental Results

The algorithm used for solving the problem is a C++ implementation of the NEWUOA solver [2]. 5 We have made slight modifications to the algorithm and have added a module that reads the data in the format given in the previous sections. The algorithm has a parameter *tolerance*, which in our case is set to 10^{-9}, and specifies that any two values that differ by at most 10^{-9} are considered to be equal. The default value for the start points is the centres of the bounding rage for each parameter. The following is a summary of the results obtained.

Data	Number of transistors	Number of iterations	Actual running time	Accuracy (Δ_I)
Dataset 1	40	$1,000$	0.2 seconds	0.6%
Dataset 2	40	$4,000$	1.3 seconds	0.4%
		$50,000$	22 seconds	0.5%
Dataset 3	18	$1,000$	0.12 seconds	
		$4,000$	0.7 seconds	
		$50,000$	11 seconds	

Bibliography

[1] fmincon, matlab documentation. http://www.mathworks.com/help/optim/ug/fmincon.html. Online; accessed 21 June 2015.

[2] NEWUOA method for derivative-free nonlinear programming. C++ template function. adapted from fortran implementation. http://attractivechaos.awardspace.com/, 2008. Accessed: 24 June 2015.

[3] Scipy, collection of mathematical algorithms for python. http://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html, May 5, 2010. Accessed: 21 June 2015.

[4] Bernardetta Addis and Sven Leyffer. A trust-region algorithm for global optimization. *Computational Optimization and Applications*, 35(3):287–304, 2006.

[5] MIHAI ANITESCU. Trust region fundamentals. http://www.mcs.anl.gov/~anitescu/CLASSES/2012/LECTURES/S310-2012-lect5.pdf, 2012. Retrieved from Agronne National Laboratory Mathematics and Computer Science Division lecture notes; Accessed: 05 June 2015.

[6] Andrew R Conn, Nicholas IM Gould, and Ph L Toint. *Trust region methods*, volume 1. Siam, 2000.

[7] A.L. Custdio and J.F.A. Madeira. Glods: Global and local optimization using direct search. *Journal of Global Optimization*, 62(1):1–28, 2015.

[8] Nicolas Durand and Jean marc Alliot. A combined nelder-mead simplex and genetic algorithm. In *GECCO99: Proc. Genetic and Evol. Comp. Conf*, pages 1–7, 1999.

[9] Tarek A El-Mihoub, Adrian A Hopgood, Lars Nolle, and Alan Battersby. Hybrid genetic algorithms: A review. *Engineering Letters*, 13(2):124–137, 2006.

[10] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[11] Anders O. Göran Kenneth Holmström and Marcus M. Edvall. Tomlab user's guide - the tomlab manual. http://tomopt.com/docs/TOMLAB.pdf, May 5, 2010. Accessed: 21 June 2015.

[12] Pradeep Mugunthan, Christine A. Shoemaker, and Rommel G. Regis. Comparison of function approximation, heuristic, and derivative-based methods for automatic calibration of computationally expensive groundwater bioremediation models. *Water Resources Research*, 41(11):n/a–n/a, 2005. W11427.

[13] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[14] M.J.D. Powell. Least frobenius norm updating of quadratic models that satisfy interpolation conditions. *Mathematical Programming*, 100(1):183–215, 2004.

[15] M.J.D. Powell. The newuoa software for unconstrained optimization without derivatives. In G. Di Pillo and M. Roma, editors, *Large-Scale Nonlinear Optimization*, volume 83 of *Nonconvex Optimization and Its Applications*, pages 255–297. Springer US, 2006.

[16] Luis Miguel Rios and Nikolaos V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, 2013.