GLOBAL MINIMIZATION USING COUPLED LOCAL MINIMIZERS AND POTENTIAL APPLICATIONS IN GEOPHYSICAL DATA

A DISSERTATION

submitted towards the partial fulfilment of the

requirements for the award of the degree

of

INTEGRATED MASTER OF TECHNOLOGY

in

GEOPHYSICAL TECHNOLOGY

by

MRIDUL RAZDAN

Enrollment Number: 14410011

DEPARTMENT OF EARTH SCIENCES

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

ROORKEE - 247667 (INDIA)

MAY 2019

CANDIDATE'S DECLARATION

I hereby declare that the work which is presented in this dissertation report titled **"Global Minimization using Coupled Local Minimizers and Potential applications in Geophysical Data"** in partial fulfilment of the requirements for the award of the degree of **"Integrated Master of Technology"** in **Geophysical Technology**, submitted to the Department of Earth Sciences, Indian Institute of Technology Roorkee is an authentic record of my own work carried out during the period from July 2018 to May 2019 under the supervision of **Dr P.K. Gupta**, Emeritus Fellow and **Dr. M. Israil**, Professor, Department of Earth Sciences, Indian Institute of Technology Roorkee.

The matter contained in this thesis has not been submitted by me for award of any other degree at any institution.

Date: **MRIDUL RAZDAN (14410011)**

Place: IIT Roorkee 5th Year, IMT Geophysical Technology

Department of Earth Science, IIT Roorkee

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

DR. P.K. GUPTA	**DR. M. ISRAIL**
Supervisor	Supervisor
Emeritus Fellow	Professor
Department of Earth Sciences,	Department of Earth Sciences,
IIT Roorkee	IIT Roorkee

ACKNOWLEDGEMENT

This dissertation would not have been possible without the guidance and support of Dr. P. K. Gupta, Emeritus fellow & Dr M. Israil, Professor, Department of Earth Sciences, IIT Roorkee. I thank them both for mentoring me and guiding me through all the obstacles. Dr. P.K. Gupta has been a constant source of inspiration throughout my time here as his student. His thought-provoking ideas and constant constructive criticism have helped me to refine my work and develop the skills what I have today.

I would like to thank all the people in Department of Earth Sciences, IIT Roorkee for their administrative support.

I would also like to sincerely extend my thanks to my batchmates and friends for their guidance and encouragement throughout my time here. I'd like to especially thank Omkar, Mandeep, Naveen, Saurav, Rutuj, Gautham, Keshvam, Nikhil for their constant support and Rimple for his technical help and expertise.

Finally, I acknowledge the love and support of my family, who have always stood by my side.

TABLE OF CONTENTS

ABSTRACT

Mathematical optimization of a function is usually necessary in every field of science. Optimization is performed to find the ideal solution to a well-defined quantitative problem in a variety of disciplines. Fundamentally, an optimization problem involves maximizing or minimizing a cost/benefit function by systematically selecting input values from within a permitted set and calculating the function's value. In Geophysics, we employ optimization schemes to solve inverse problems, which are the backbone of any geophysical workflow, to calculate causal parameters from observational data. A lot of optimization methods, linear and stochastic/probabilistic, are popularly used today but each have their own set of problems. This dissertation addresses the latter and focuses on a relatively uncommon but efficient method applicable to global optimization of functions that may possess multiple local optima (minima/maxima), by using global approach, co-operative coupling and quick convergence. The method is then tested to ascertain the quality of its solution. Potential Geophysical applications are also discussed.

LIST OF FIGURES

CHAPTER 1: OPTIMIZATION THEORY

1.1 Introduction

Optimization, in general terms, refers to obtaining the best possible solution to a posed problem. Mathematical Optimization, also referred to as Mathematical Programming, is a collection of mathematical principles and methods that are used to solve problems across several disciplines such as pure sciences, engineering, economics and business. Optimization is performed to find the ideal solution (or as close to an ideal solution as possible) to a well-defined quantitative problem. Fundamentally, an optimization problem involves maximizing or minimizing a cost/benefit function by systematically selecting inputs from within a given domain (the set of all feasible solutions) till the computed value of the function reaches the desired point.

Optimization is a powerful tool in helping us make informed decisions, generate strategies and analyzing physical systems. Optimization problems are expressed in terms of ***variables or unknowns*** (sometimes referred to as 'degrees of freedom'), the ***domain***, ***the function to be optimized*** (referred to as the 'objective'), and occasionally ***constraints*** as and when necessary.

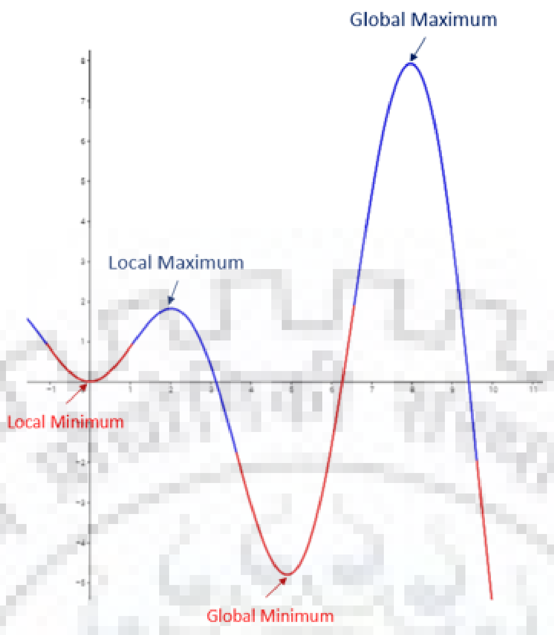


Figure 1: A graph of a function showing its extrema

In mathematical terms, say we have a function $f: X \to \mathbb{R}$ from some set X to the real numbers, then for an optimal solution, we seek an element $x_0 \epsilon\ X$ such that $f(x_0) \leq f(x)$ for all $x\ \epsilon\ X$ in case of minimization (or vice versa for maximization)

Here, $f(x)$ is our objective, x is our variable or unknown. x_0 is our optimal solution. This is an unconstrained optimization problem since we haven't imposed constraints or limits on our variables.

1.2 Types of Optimization Problems

Optimization problems can be classified based on a variety of factors. These primarily include (but are not limited to):

- Type of constraints
  - Constrained optimization problems
  - Unconstrained optimization problems
- Nature of the equations involved
  - Linear programming problems
  - Non-linear programming problems
- Deterministic nature of variables
  - Deterministic programming problems
  - Probabilistic or stochastic programming problems
- Permissible value of design variables
  - Integer programming problems
  - Real-valued programming problems
- Number of objective functions
  - Single-objective programming problem
  - Multi-objective programming problem

Identifying what type of problem we have is imperative since it allows us to construct an appropriate model. From here on, we shall work on the assumption that the optimization to be performed is a minimization problem.

1.3 Optimization workflow

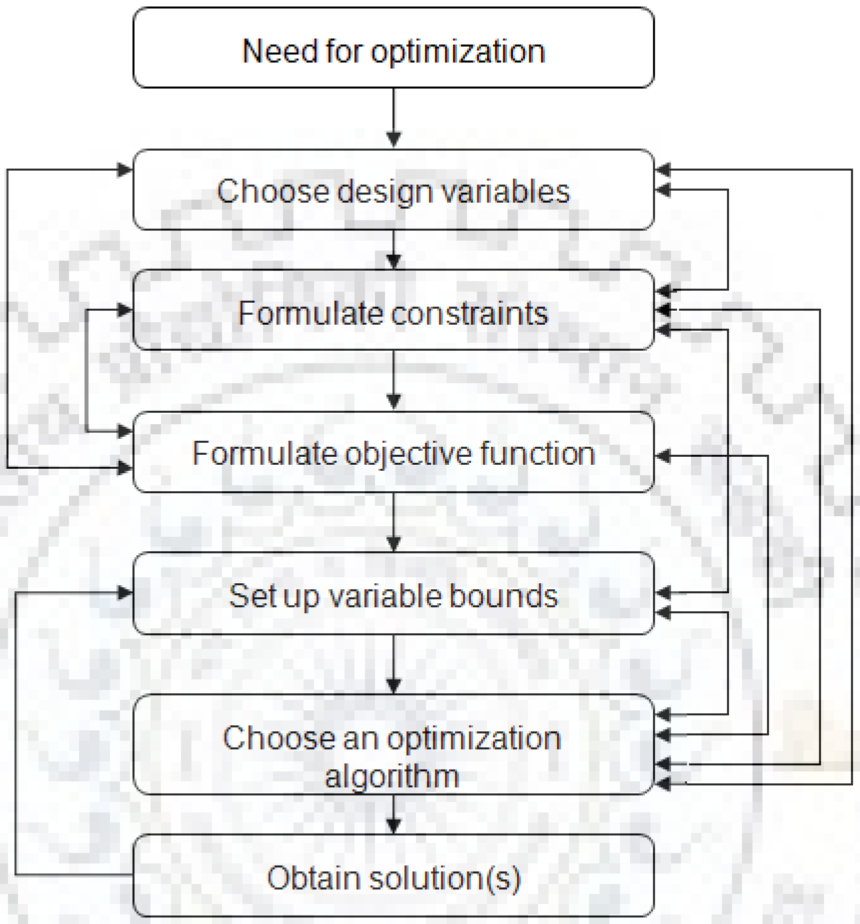


Figure 2: A Flowchart for the optimal design procedure

Constructing a model

The first and quite possibly the most vital step in optimization is to construct an appropriate model, that accurately reflects the characteristics of our problem. A better/more suitable model that fits the given problem will provide a better solution after the optimization process is complete.

Modeling is the process of the identification of the *objective*, the *variables* and the *constraints* of the given problem and then expressing them in mathematical terms.

An **objective** is a quantitative measure of the performance of the system that we want to optimize. This may include maximization of profits in business or minimization of error/misfit in sciences.

The **variables** are the components of the system that we wish to find the optimal values for in order to obtain the desired response from our system. In other words, optimization helps us find the best variables for which the value of the objective function is maximum/minimum. For example, the parameters in a subsurface model.

The **constraints** are the functions that describe the relationships and impose limits on the variables. They help us narrow down the domain of feasible solutions.

Choosing an optimization algorithm

Several computational algorithms have been formulated to solve optimization problems. They may range from those that terminate in a finite number of steps, or those that converge to a solution iteratively, or heuristics and metaheuristics that may provide approximately optimal solutions but are more robust.

For the sake of concision and brevity, we shall only discuss the optimization techniques which are relevant to the topic of this dissertation. Optimization techniques are discussed in the next chapter.

Obtaining the solution

After we have chosen the most appropriate algorithm to suit our problem, the next step is to solve it. Algorithms that terminate in a finite number of steps return the solution when their optimality condition is satisfied. Some meta-heuristics on the other hand can go on until we manually decide when to terminate them (for example after a certain number of iterations) or impose additional constraints (for example the improvement in the solution being below a certain threshold) to re-define an optimality condition.

CHAPTER 2: OPTIMIZATION TECHNIQUES – PART 1

2.1 Fundamental Strategies to approach convergence

Unconstrained minimization is the most basic form of minimization problems and they arise frequently in many direct applications. They can also arise as a result of reformulating constrained minimization problems by replacing the constraints with some penalty parameters added to our objective function which have the effect of making discouraging the optimization procedure to violate constraints.

Every unconstrained minimization algorithm requires the user to provide a starting or initiation point x_0. It can be an arbitrarily chosen point or it can be an educated guess. The latter is more likely when the user possesses good information about the nature of the objective and the dataset. The starting point may also be chosen by an algorithm, either randomly or by a systematic approach.

Beginning at x_0, the algorithm iterates the value of x as $\{x_k\}_{k=0}^{\infty}$ which terminates upon reaching the optimization criteria. To move from one iterate to the next, the algorithm uses the information about the function at that point and sometimes also that of earlier iterates.

There are 2 fundamental strategies for moving from the current point x_k to the next x_{k+1}. The *Line search* and *Trust region* approach.

LINE SEARCH

In line search, the algorithm calculates a search direction and subsequently computes how much distance to move along it. The iteration is formulated as:

$$x_{k+1} = x_k + \alpha_k p_k$$

Here, α_k is a scalar and is known as the 'step length' and p_k is the search direction. An effective line search requires good choices of both α_k and p_k.

Step Direction

The condition to be satisfied is: $p_k^T \nabla f_k < 0$

to guarantee the reduction along the search direction (now called the descent direction).

p_k can have the form: $p_k = -B_k^{-1} \nabla f_k$

Where B_k is nonsingular and symmetric. It can be the identity matrix I in the steepest descent method. On the other hand, in Newton's method, it is the exact Hessian $\nabla^2 f(x_k)$ or the approximate to the Hessian (updated at every step) in Quasi-Newton methods.

Combining the above, we get: $p_k^T \nabla f_k = -f_k^T B_k^{-1} \nabla f_k < 0$

Step Length

Step length needs to be chosen in such a way that the reduction of f is substantial but not too computationally expensive. The ideal choice would be the global minimizer of the function defined by $\phi(\alpha) = f(x_{k+}\alpha_k), \alpha > 0$ but it's very expensive to identify. Therefore, for practicality, we perform an infline search to achieve substantial reduction at minimal cost.

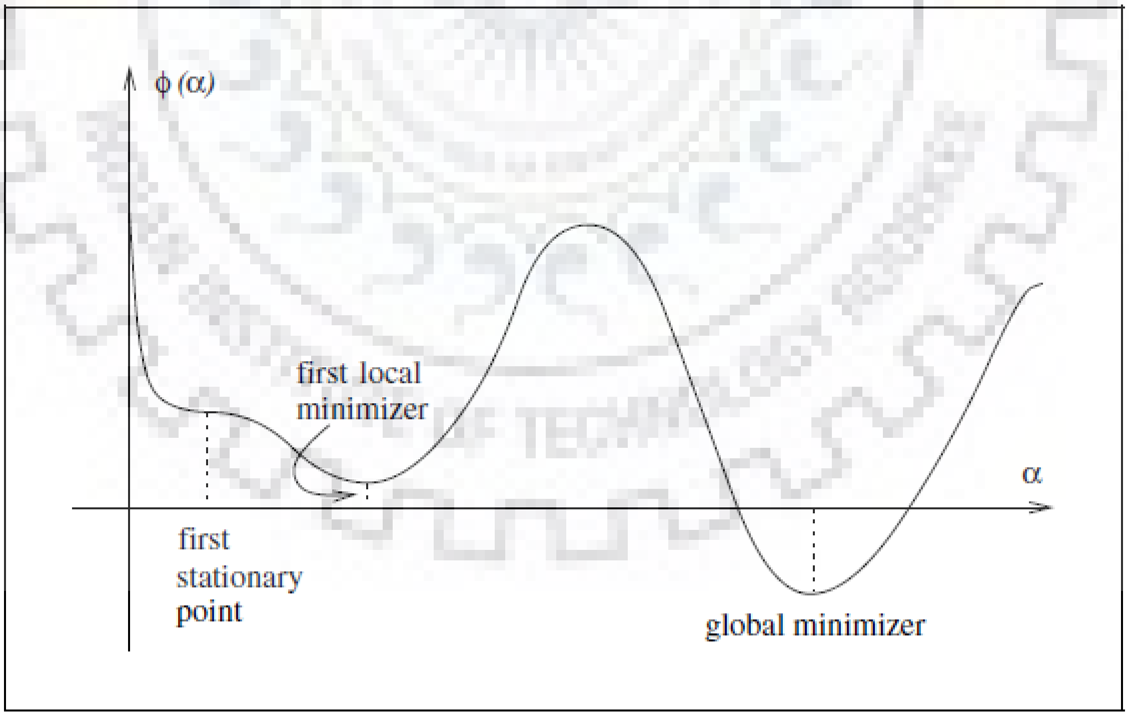


Figure 3: The global minimizer of the aforementioned univariate function is the ideal step length

TRUST REGION

Trust-region methods define a region around the current iterate within which they trust the model to be an appropriate representation of the objective function, and then choose the step to be the approximate minimizer of the model in this region. Effectively, they choose the direction and length of the step simultaneously. If the current step size isn't appropriate, these methods reduce the size of the trust region and find a new minimizer. The direction usually changes when the step size does.

Trust Region

The trust-region is defined as a spherical area of radius Δ_k in which the trust-region subproblem lies.

Trust-region subproblem

If we are using the quadratic model to approximate the original objective function, then our optimization problem is essentially reduced to solving a sequence of trust-region subproblems

$$min\ m_k(p) = f_k + {g_k}^T p + \frac{1}{2} p^T B_k p$$

$$s.t.\ ||p|| <= \Delta_k$$

Where Δ_k is the trust region radius, g_k is the gradient at current point and B_k is the hessian (or a hessian approximation). It is easy to find the solution to the trust-region subproblem if B_k is positive definite.

Actual reduction and predicted reduction

The most critical issue underlying the trust-region method is to update the size of the trust-region at every iteration. If the current iteration makes a satisfactory reduction, we may exploits our model more in the next iteration by setting a larger Δ_k. If we only achieved a limited improvement after the current iteration, the radius of the trust-region then should not have any increase, or in the worst cases, we may decrease the size of the trust-region by adjusting the radius to a smaller value to check the model's validity.

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$$

Whether to take a more ambitious step or a more conservative one is depend on the ratio between the actual reduction gained by true reduction in the original objective function and the predicted reduction expected in the model function. Empirical threshold values of the ratio ρ_k will guide us in determining the size of the trust-region.

Pseudocode

```
Set the starting point at $x_0$, set the iteration number $k = 1$
for $k = 1, 2...$
Get the improving step by solving trust-region sub-problem ()
Evaluate $\rho_k$ from equation()
if $\rho_k < \eta_2$
$\Delta_{k+1} = t_1\Delta_k$
else
if $\rho_k > \eta_3$ and $p_k = ||\Delta_k||$ (full step and model is a good approximation)
$\Delta_{k+1} = min(t_2\Delta_k, \Delta_M)$
else
$\Delta_{k+1} = \Delta_k$
if $\rho_k > \eta_1$
$x_{k+1} = x_k + p_k$
else
$x_{k+1} = x_k$ (the model is not a good approximation and need to solve another trust-region subproblem within a smaller trust-region)
end >
```

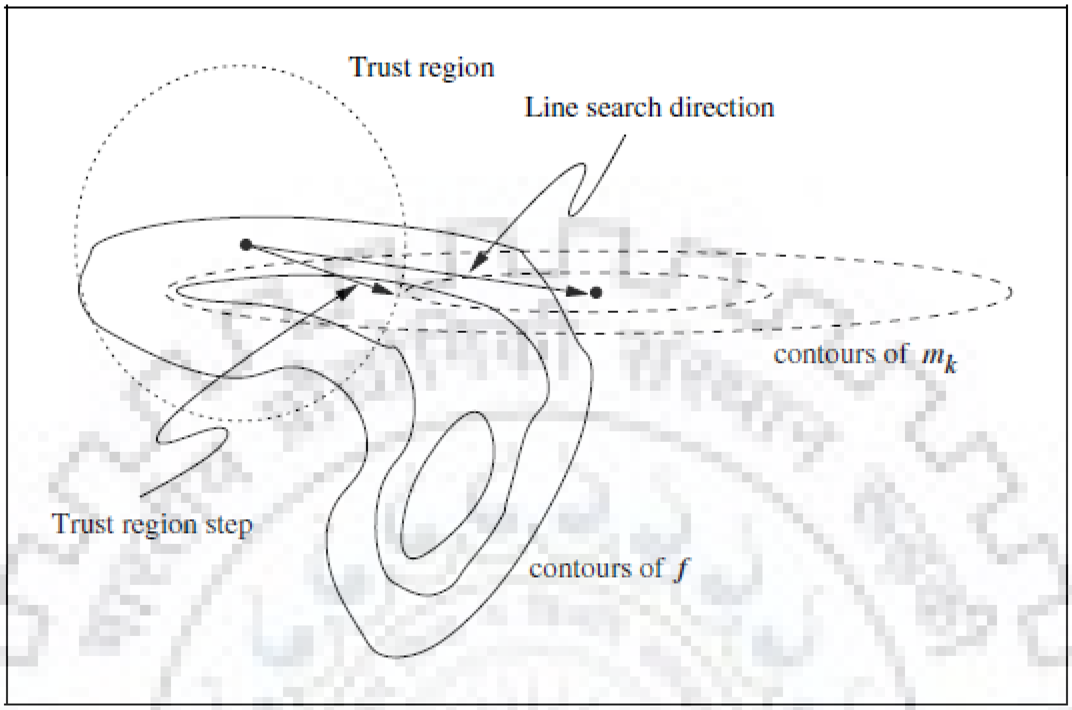


Figure 4: Line Search vs Trust Region Approach

2.2 Conjugate Gradient Method

The conjugate gradient method is a mathematical technique which is useful for both linear and non-linear systems optimization. Conjugate Gradient algorithm, which is generally used as an iterative technique, can also be used as a direct method, producing a numerical solution. This method is used in very large systems where solving with a direct method is not practical.

Non-Linear CGM

Say our initial function is $f(x)$

A residual is calculated. The residual is always the negative of the gradient $r_i = -f'(x)$ in the NL case. using the Gram-Schmidt conjugation of the residuals, we compute the search direction. We subsequently use line search which is much more difficult in the NL case. We search for a value of α_i is found in order to minimize $f(x_i + \alpha_i * d_i)$ This is achieved by ensuring the gradient and the search direction are orthogonal.

To find the value of β there are multiple methods. Two of the better equations are the Fletcher-Reeves (which is used in linear GC) and the Polak-Ribiere method. The former converges only if initial guess is sufficiently close to the desired minimum, while the latter can sometimes cycle infinitely but often converges more quickly.

Fletcher-Reeves: $$\beta^{FR} = \frac{\Delta x_n^\top \Delta x_n}{\Delta x_{n-1}^\top \Delta x_{n-1}}$$

Polak-Ribiere: $$\beta^{PR} = \frac{\Delta x_n^\top (\Delta x_n - \Delta x_{n-1})}{\Delta x_{n-1}^\top \Delta x_{n-1}}$$

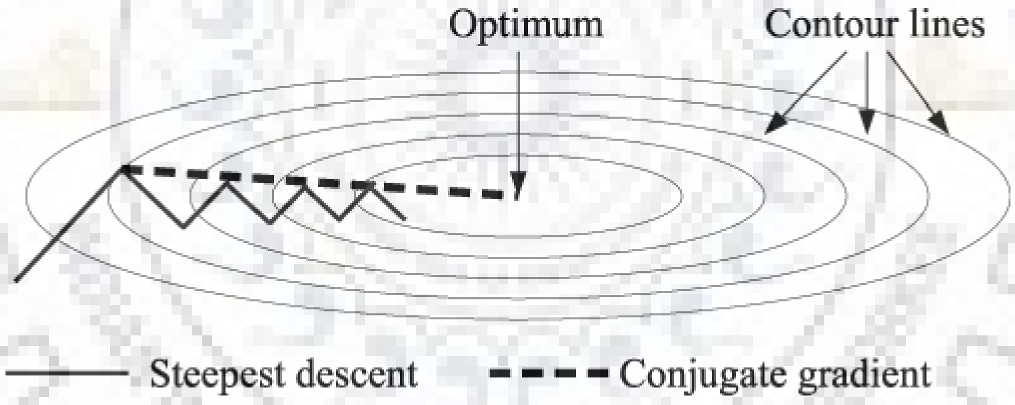


Figure 5: A comparison of CGM and Steepest Descent

2.3 Quasi Newton Methods

Quasi-Newton Methods are a class of optimization methods that are used in Non-Linear Programming when full Newton’s Methods become either exhibit slow convergence or difficulty in use. More specifically, these methods are used to find the global minimum of a function f(x) that is twice-differentiable. There are distinct advantages to using Quasi-Newton Methods over the full Newton's

Method for expansive and complex non-linear problems. Though, these methods are not exact, and may have some limitations depending on the nature of Quasi-Newton Method used and the target problem. In spite of this, Quasi-Newton Methods are generally worth using with the exception of very simple problems. In this section we are going to specifically discuss about the **Broyden–Fletcher–Goldfarb–Shanno** (**BFGS**) algorithm since it has direct application in this work.

BFGS ALGORITHM

The BFGS algorithm is an iterative algorithm of the line search family used to solve unconstrained non-linear optimization challenges.

BFGS (similar to other newton-like methods) uses quadratic Taylor approximation of the objective function in a d-vicinity of x:

$f(x + d) \approx q(d) = f(x) + d^{T}g(x) + \frac{1}{2}\, d^{T}H(x)\, d$,

where $g(x)$ is the gradient vector and $H(x)$ is the Hessian matrix.

The necessary condition for a local minimum of $q(d)$ with respect to d results in the linear system:

$g(x)+ H(x)\ d = 0$

which, in turn, gives the **Newton direction** d for line search:

$d = - H(x)^{-1}g(x))$

The exact Newton direction (which is subject to define in Newton-type methods) is reliable when

- The Hessian matrix exists and positive definite;
- The difference between the true objective function and its quadratic approximation is not large.

In Quasi-Newton methods, the idea is to use matrices that approximate the Hessian matrix and/or its inverse rather than exact Hessian matrix computation (as in Newton-type methods).The matrices are normally named $B \approx H$ and $D \approx H^{-1}$.

On each iteration, the matrices are adjusted and can be produced in many different ways, from very simple techniques to highly advanced schemes.

The **BFGS** method uses the *BFGS updating formula* which converges to $H(x^*)$:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$$

where

- $s_k = x_{k+1} - x_k$,
- $y_k = g_{k+1} - g_k$.

As a starting point, B_0 can be set to any symmetric positive definite matrix, for example and very often, the identity matrix.

The **BFGS** method exposes superlinear convergence; resource-intensivity is estimated as $O(n^2)$ per iteration for n-component argument vector.

CHAPTER 3: OPTIMIZATION TECHNIQUES – PART 2

3.1 Heuristics and Metaheuristics

In this chapter, we shall be discussing another method of optimization besides finitely terminating algorithms and convergent iterative methods. These methods are known as Heuristics and Metaheuristics

Heuristics are problem dependent techniques i.e. they are adapted to the problem at hand. They do not have the guarantee of an exact or optimal solution but give a 'good' solution in a 'reasonable' amount of time. They work similar to empirical search methods but may get too greedy and trapped in a local minima

Metaheuristics on the other hand, are a high level, relatively problem independent implementations that provide a set of guidelines or strategies to develop heuristic methods suitable for the problem. Since there are no satisfactory exact general solutions for all global optimization problems, we therefore employ metaheuristics to formulate a solution to such problems without having to deeply adapt to each. We shall be discussing some important metaheuristics that are used to solve global optimization problems.

Metaheuristics are very diverse in that they can be single-solution based or population based, may be memetic or hybridized with other optimization approaches. A lot of them are inspired by nature or real-life processes. In particular, we shall be discussing two of the most influential metaheuristics in use today.

3.2 Simulated Annealing

Simulated Annealing is a probabilistic search technique to approximate the global minimum for a general function (meaning that it isn't problem specific and therefore not optimized to solve a particular class of problems). In a large search space, the SA technique works as a metaheuristic to find an approximate optimum in a reasonable amount of time. Therefore, SA is preferable when finding an exact solution is not a priority but rather, minimizing time and computational constraints are.

Inspiration

The algorithm is inspired from the real-life metallurgical process of the eponymous 'Annealing'. Annealing in metal-working is a thermodynamic process that involves heating a metal above its recrystallization point and then controlled cooling to *remove internal stresses, increase crystal size and decrease solid defects*. These characteristics of the metal being worked are dependent on its **Thermodynamic Free Energy** while the parameter that controls the process is **Temperature**.

Interpretation of the analogy

These factors are integrated into the SA algorithm in the following way:

- On cooling quickly, the metal becomes hard but brittle and glassy. This is analogous to a local minimum.
- On slow and controlled cooling, the metal crystals grow in size and internal defects reduced. This optimizes the internal structure of the metal to remove internal stresses. This is analogous to a global minimum.

As the metal cools, its structure becomes more and more fixed allowing it to retain its newly obtained properties. The process of slow cooling discussed above is interpreted as a gradual reduction in the probability of acceptance of worse solutions as the algorithm explores the search space. The idea of accepting worse solutions is one of the fundamental properties of metaheuristics since it allows them to more thoroughly explore the search space and have a greater chance of finding the global optimum. This ensures that they do not get stuck at a local minimum.

Parameters and their implementation

To simulate the annealing process, we implement the notion of cooling via a temperature variable **T**. The fitness or worseness of solution is characterized by the internal energy of the system which represented by **E**. It is our cost function or misfit function that has to be minimized. The stable state of any system is the one with the least internal energy so this analogy makes sense.

Now, we use a Maxwell-Boltzmann distribution for statistical simulation of probability. This distribution is also followed by gas molecules.

$$p = e^{-\Delta E/kT}$$

In the distribution model, ***p*** the probability that a system with a lower energy state **E** will exist. In our SA algorithm, we use it to denote the acceptance probability of worse solutions.

k is the Boltzmann constant

So, we can see that when T is high, E has a marginal effect (since *p* is close to 1). Thus the probability of acceptance of worse solutions is higher.

At low T, E has a significant effect, thus the lowered threshold for misfit and only good solutions have a chance to be accepted.

Algorithm

1. Generate a random solution. Doesn't need to be the best guess at optimal solution.
2. Calculate using its cost using a defined cost function. The cost function needs to be relatively simple since it gets called at every iteration.
3. Generate a random neighbouring solution.
4. Calculate the cost of new solution.
5. Compare the cost of both solutions and evaluate the acceptance probability *p*.

6. Compare p to a random number (generate using RNG) to decide whether to update the model with the new solution or not.
7. Repeat until optimality criteria are fulfilled.

Pseudocode

```
• Let $s = s_0$
• For $k = 0$ through $k_{\max}$ (user defined):
    ○ $T \leftarrow$ temperature( $k_{\max}/(k+1)$ )
    ○ Pick a random neighbour, $s_{\text{new}} \leftarrow$ neighbour($s$)
    ○ If $P(E(s), E(s_{\text{new}}), T) \geq$ random(0, 1):
        ▪ $s \leftarrow s_{\text{new}}$
• Output: the final state $s$
```

Important considerations

- The initial value of T chosen can be important to control the acceptance rate at the start
- Temperature reduction scheduling is important. The algorithm performs much better when T is reduced after many iterations
- The cost function must be efficient since it gets called at every iteration.

3.3 Genetic Algorithm

The genetic algorithm is a search-based optimization technique inspired by the process of natural selection (survival of the fittest). We use GA to find optimal or near-optimal solutions that would otherwise take a long time to solve using exact algorithms.

Herein we define a fitness function as our survival criteria (in this case it may be the misfit between observed and computed data values, which we need to minimize)

Our model is represented as a string of genes that represent a solution. This string of genes is called the chromosome. It contains model parameters and is a single encoding of the part of the solution space, i.e. one of the possible solutions to the problem.

A **gene** is one element position of a chromosome

An **allele** is the value a gene takes for a particular chromosome

All of the optimization is done over a population, which is a subset of all possible (encoded) solutions to the given problem

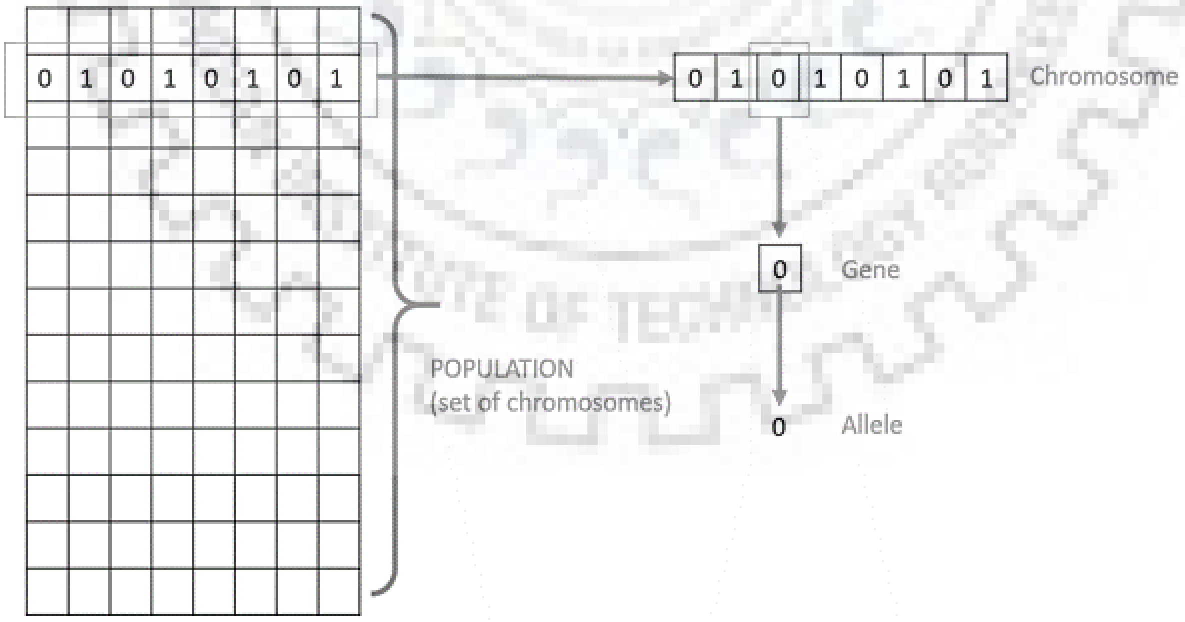


Figure 6: A representation of the metaphors commonly used in GA

Process

The basic premise of GA is to mimic biologic evolution. We start with an initial population which may be generated at random. The fitness of each initial model is calculated. Out of these some models are selected to act as parent models for the next generation of solutions. We may implement elitism to choose models with highest fitness to directly enter the next generation. A few models with low fitness are also selected to promote diversity. The chosen models subsequently undergo reproduction wherein we apply crossover and mutation operators on the parents to generate new offsprings. These replace their parent population and form the new selection pool of solutions. This process repeats until the stop criterion is reached.

The process can be depicted as:

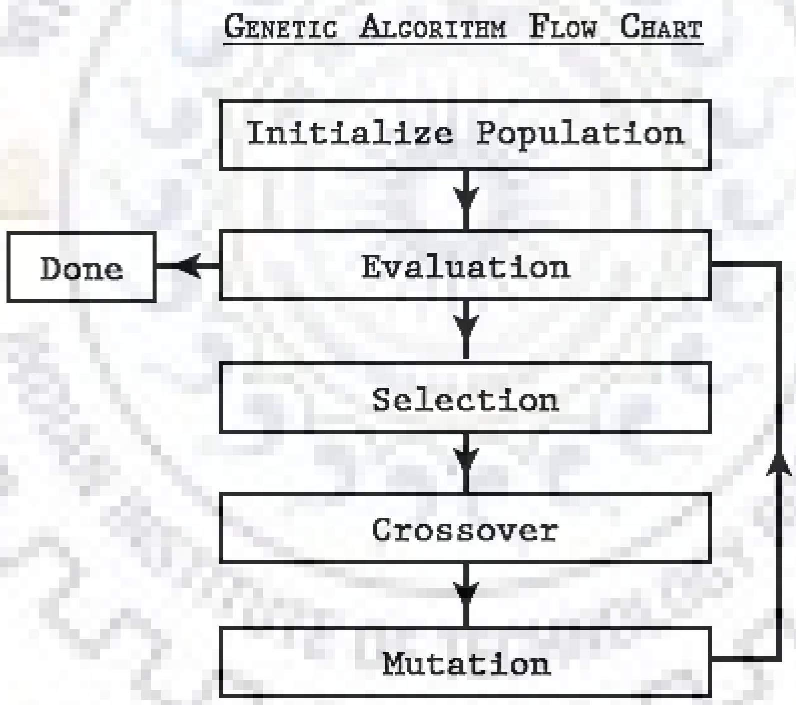


Figure 7: Flowchart of the GA process

STEPS

Pre-initialization

Before initializing our model population, we need to choose a proper representation for our solutions. The parameters need to be discretized first so that we can represent our model as a chromosome. For Example:

MODEL 1 $\longrightarrow$ $({}^1P_1, {}^1P_2, {}^1P_3, \ldots, {}^1P_N)$

MODEL 2 $\longrightarrow$ $({}^2P_1, {}^2P_2, {}^2P_3, \ldots, {}^2P_N)$

After that we need to map the chromosome to our chosen representation. Binary representation is one of the simplest and most widely used.

MODEL $\longrightarrow$ [2, 5, 4, 7] $\longrightarrow$ [0010|0101|0100|0111]

Initialization

After choosing a suitable representation, we can initialize our population. It can either be a random initialization or we may use a known heuristic (normally only used to seed the population with a few good solutions). Because it is the diversity of the models that lead us to the optimal solution.

Fitness Function and Elitism

The generated models are characterized by their 'fitness' which can be a function of the system we're trying to optimize. For example, it can be the misfit between observed and computed data.

Using the fitness function, a process called elitism is implemented. It basically involves the selection of the fittest candidates (the elite) from a generation to be directly copied into the next generation without changes. The fitness function assigns weightage to the individual models.

An important consideration is that the choice should be probabilistic rather than deterministic. (to avoid getting trapped into the local minima) because otherwise the algorithm with sacrifice long term fitness in favour of short-term success.

Reproduction

Reproduction involves:

A. Parent Selection

B. Crossover

Parent selection is done to add selection pressure towards fitter solutions and yet prevent some extremely fit solutions from taking over the entire population because diversity is essential for reaching the optimum.

It may be done by:

a) Fitness proportionate selection (via Roulette wheel method)

b) K-way tournament selection

Crossover is analogous to biological crossover. More than one parents are selected to produce one or more offsprings with similar characteristics (genetic material) of their parents. It can be applied in different ways

- One-point crossover
- Multipoint crossover
- Uniform crossover

Mutation

Mutations are small tweaks in the existing chromosomes to get new properties. It is used to introduce and maintain diversity in the population so that the algorithm doesn't converge at a local minimum. It is applied with a low probability since if we introduce mutations at high probabilities, GA gets reduced to random search.

CHAPTER 4: COUPLED LOCAL MINIMIZERS (CLMs)

4.1 Introduction

While the previously discussed optimization schemes work well, they have their own set of drawbacks. The convergent iterative methods may be fast but they often get stuck in local minima. On the other hand, metaheuristics like SA and GA navigate the solution space well but fail to take into account any of the information provided by the nature of the function (unlike the gradient based methods). A method known as optimization via coupled local minimizers (**CLM**) is a co-operative search mechanism which incorporates the advantages of both the fast convergence gradient based methods with the global approach, parallelism and information exchange of popular metaheuristics. This combination results in an efficient global optimization algorithm.

4.2 The Method

In the CLM method a group of search points is initially set up, ideally spread over the search space. The search process is guided by the derivative information at each of these locations. But instead of performing distinct individual searches (which happens in case of multi-start local optimization), these local optimizers are coupled throughout the search process via imposed constraints that compel all these to converge to a single point.

The co-operativeness of this method is enforced by minimizing the average value of the objective function, which is the value of the function averaged over all the search points. This causes all the points in the group to search for the for the minimum value of this average objective function using derivative information about the function.

The coupling is enforced by subjecting the search points to pairwise synchronization constraints that force them to end at the same final location.

Augmented Lagrangian Method

We use the Augmented Lagrangian method (sometimes called the Method of Multipliers) which is a method to solve constrained optimization problems.

We construct an augmented Lagrangian function L $_A$ which is defined by the average objective function of the points along with the pairwise sync constraints between individual minimizers.

The function is defined as:

$$\mathscr{L}_A(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \sum_i \lambda_i h_i(\boldsymbol{x}) + \frac{\gamma}{2} \sum_i h_i^2(\boldsymbol{x}),$$

Here $\mathscr{L}_A$ is the augmented Lagrangian function, ***f(x)*** is the objective function $\boldsymbol{h}_i(\boldsymbol{x})$ are the equality constraints where $x \in \mathbb{R}^n$.

λ$_i$ are the Lagrange parameter estimates and **γ** is the penalty parameter.

$\sum_i \boldsymbol{\lambda}_i \boldsymbol{h}_i(\boldsymbol{x})$ expresses the hard constraints.

$\frac{\gamma}{2} \sum_i \boldsymbol{h}_i^2(\boldsymbol{x})$ expresses the soft constraints.

For every iteration (say 'k'), the augmented Lagrangian $\mathscr{L}_A$ *(x, λ$_k$)* is minimized w.r.t. '*x*' to compute x_k^*, which is the optimum x for that iteration. Next, we update the values of $\lambda_k = (\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_k)$ using x_k^* to start the next iteration. *λ's* are updated using the formula:

$$(\lambda_i)_{k+1} = (\lambda_i)_k + \gamma h_i(x_k^*)$$

This continues until the the ideal λ^* are found.

CLM Method

Now, say we have a population of q local minimizers. For the objective function ***f(x)***, the average cost is defined as:

$$\langle f \rangle = \frac{1}{q} \sum_{i=1}^{q} f(\boldsymbol{x}^{(i)}).$$

Next, pairwise sync constraints are enforced on the design vectors that represent our minimizers $x^{(i)}$. This results in a constrained minimization problem.

$$\min_{\boldsymbol{x}^{(i)} \in \mathbb{R}^n} \langle f \rangle \text{ such that } \boldsymbol{x}^{(i)} - \boldsymbol{x}^{(i+1)} = 0$$

Subject to the boundary conditions

$$\boldsymbol{x}^{(q+1)} = \boldsymbol{x}^{(1)}$$

Thereafter we define the Augmented Lagrangian function:

$$\mathscr{L}_{\mathrm{A}}(\mathbf{x}, \boldsymbol{\Lambda}) = \frac{\eta}{q} \sum_{i=1}^{q} f(\boldsymbol{x}^{(i)}) + \sum_{i=1}^{q} \boldsymbol{\lambda}^{(i)\,\mathrm{T}} [\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(i+1)}] + \frac{\gamma}{2} \sum_{i=1}^{q} \|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(i+1)}\|^2$$

where:

$$\mathbf{x} = [\boldsymbol{x}^{(1)}; \ldots ; \boldsymbol{x}^{(q)}] \quad \boldsymbol{\Lambda} = [\boldsymbol{\lambda}^{(1)}; \ldots ; \boldsymbol{\lambda}^{(q)}]$$

$x^{(i)}, \boldsymbol{\lambda}^{(i)} \in \mathbb{R}^n$

||.|| is the Euclidean norm of a vector.

η is the weighting factor for the average objective function.

Now, the main conditions we want to enforce on the design vectors of local minimizers is to look for the minimum of the average cost of all search points and to end up at the same final point. When the initial states are located in the neighbourhood of different minima, there will be a decision regarding which one to choose. For an appropriately chosen pair of **η** and **γ**, the optimal solution (usually the global minimum) is obtained. The number of search points needed (q) usually depends on the shape of the function; more specifically the number of local minima per of volume.

4.3 Implementation

In this dissertation, we use the Broyden–Fletcher–Goldfarb–Shanno algorithm, which belongs to the class of Quasi-Newton methods. We have discussed the BFGS method in section 3 of chapter 2.

For an objective function f_x, from an initial guess $\boldsymbol{x_0}$, and an appropriate Hessian Matrix $\boldsymbol{B_0}$, we iterate the following procedure to have $\boldsymbol{x_k}$ converge to the solution.

1. Solve $B_k p_k = -\nabla f(x_k)$ to obtain the direction $\boldsymbol{p_k}$
2. Find the step size $\boldsymbol{\alpha_k}$ (in the direction found above) using a line search optimization of $\boldsymbol{f}(\boldsymbol{x_k} + \boldsymbol{\alpha p_k})$
3. Set $\mathrm{s_k} = \mathrm{\alpha_k p_k}$ and update $x_{k+1} = s_k + x_k$
4. Compute $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
5. $B_{k+1} = B_k - \frac{B_k s_k s_k{}^T B_k{}^T}{s_k{}^T B_k\, s_k} + \frac{y_k y_k{}^T}{y_k{}^T s_k}$

In this case our function would be the final Augmented Lagrangian with all the constraints incorporated.

Due to the usage of BFGS, this method converges very fast using derivative based information.

Tuning Parameters

η and γ are the tuning parameters for this method and are problem specific. A priori assumptions for these is difficult. η emphasizes the minimization of the average objective function while γ acts as a penalty parameter. Increasing γ, we emphasize soft constraints and convergence rate is improved but too high a value can cause CLM to converge to a local minimum. On the other hand, a low γ value makes CLM explore the search space more thoroughly but decreases the speed at which the algorithm converges. η emphasizes the minimization of the average objective function.

4.4 Application

The CLM method has been used to minimize a function to check its effectiveness. A 2-D Test function from (Teughels, et al., 2003) has been used to illustrate the difference in the performance of both CLM implementations.

The function is as follows:

$$f(\mathbf{x}) = \sum_{j=1}^{2} 0.01\left((x_j + 0.5)^4 - 30x_j^2 - 20x_j\right)$$

$$\text{with } -6 \leqslant x_j \leqslant 6.$$

There are 4 minima in this function out of which, one is the global minimum (4.454, 4.454)

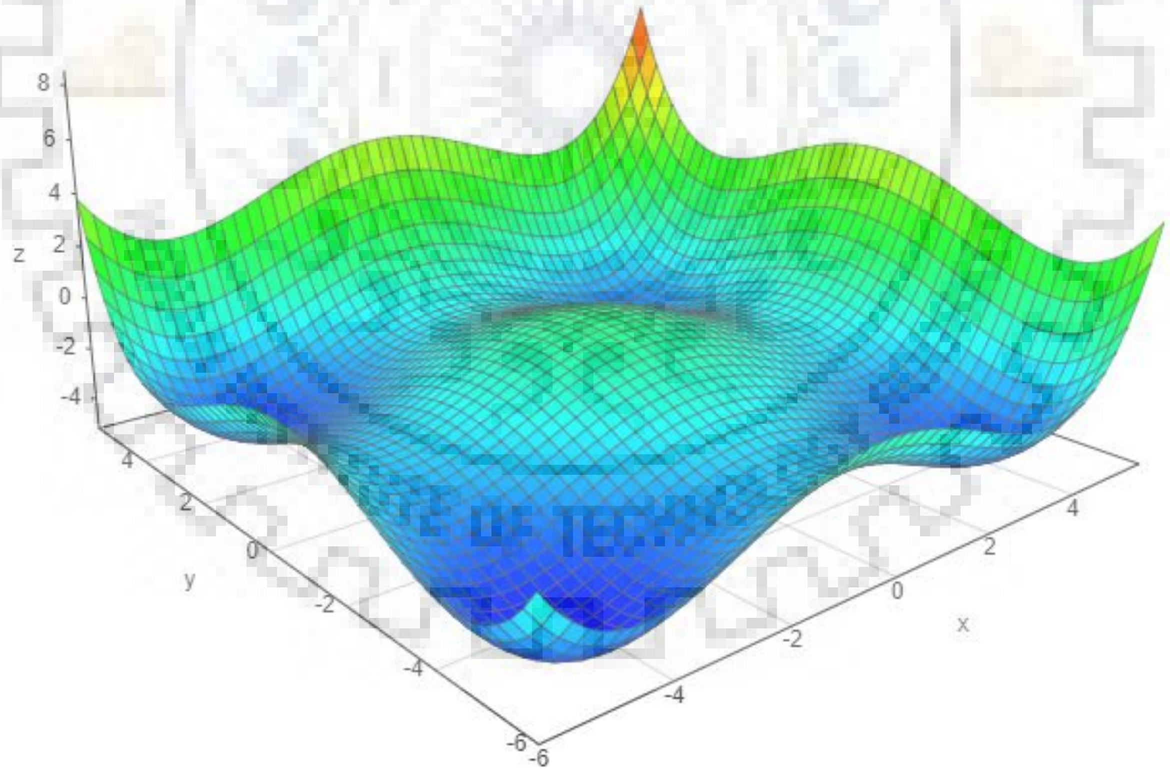


Figure 8: 3-D Surf Plot of the 2-D test function

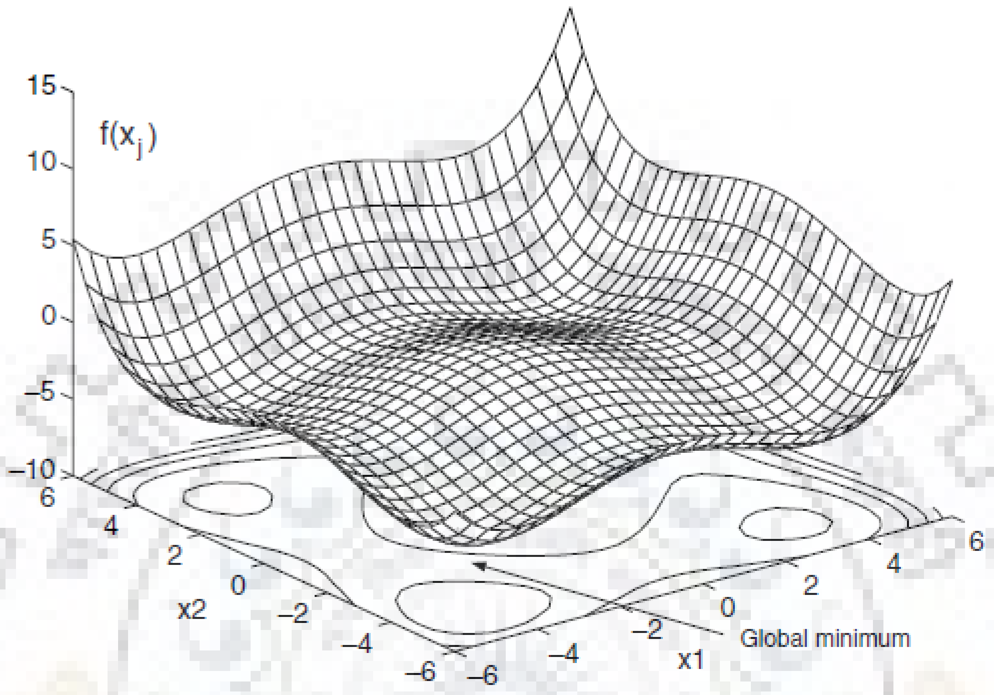


Figure 9: Test function plot as given in Teughels et al (2003) indicating the global minimum.

Two different test runs were carried out with parameters

1. q=8 ; η =3 ; γ=0.02
2. q=8 ; η =3 ; γ=2

This was mainly to test the performance of the algorithm in cases where

a) Solution space is explored thoroughly

b) Fast convergence is prioritized

The initial search points are randomly distributed in the search space. And initial Lagrange parameter estimates are seeded randomly between [-1,1] .

4.5 Performance

The complete population of local minimizers ended up in the global minima regardless of their initial location.

Test Run #1

q=8 ; η =3 ; γ=0.02

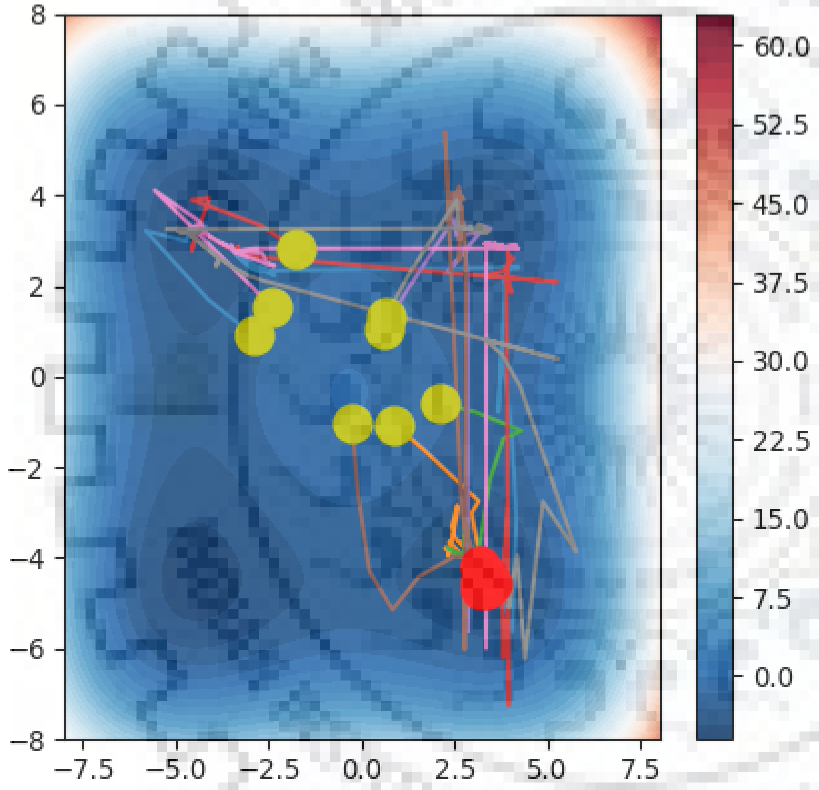


Figure 10: Paths taken by the individual minimizers for Test run #1

The yellow dots represent the initial positions of search points while the red ones are the final positions.

Since the **γ** value is quite low, we can see that the solution space is explored well.

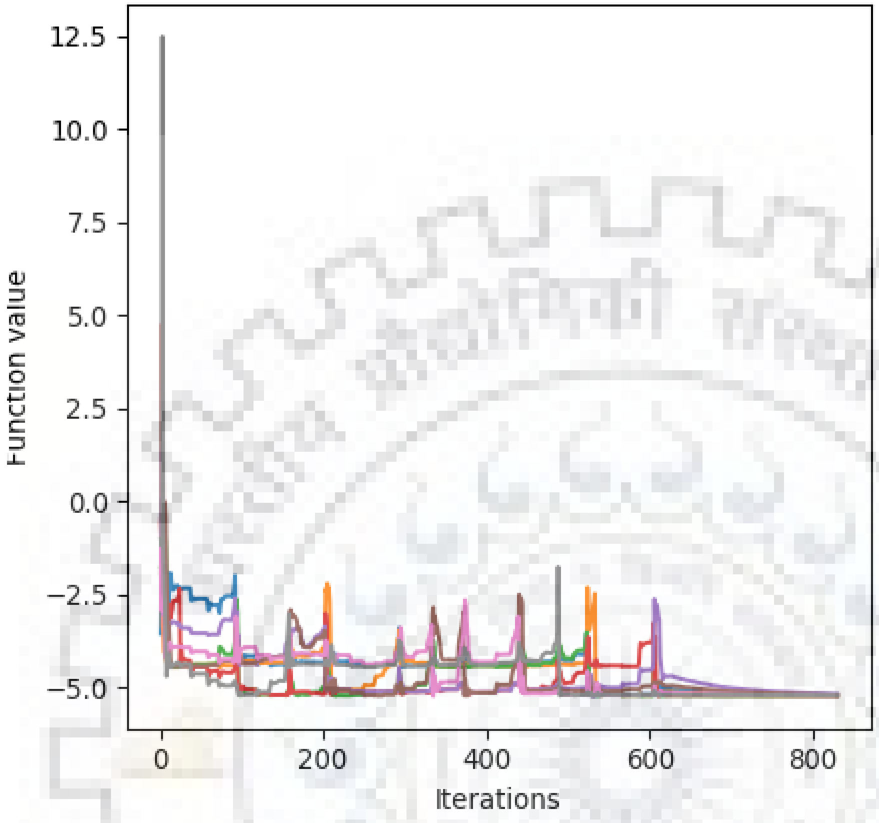


Figure 11:Function Value vs Iterations for Test run #1

We can see that a low **γ** value results in a pretty slow convergence. The number of iterations is very high even though after a point, there is barely any decrease in the function value. This reinforces the idea that the tuning parameters must be adjusted according to the problem at hand.

Test Run #2

q=8 ; η =3 ; γ=2

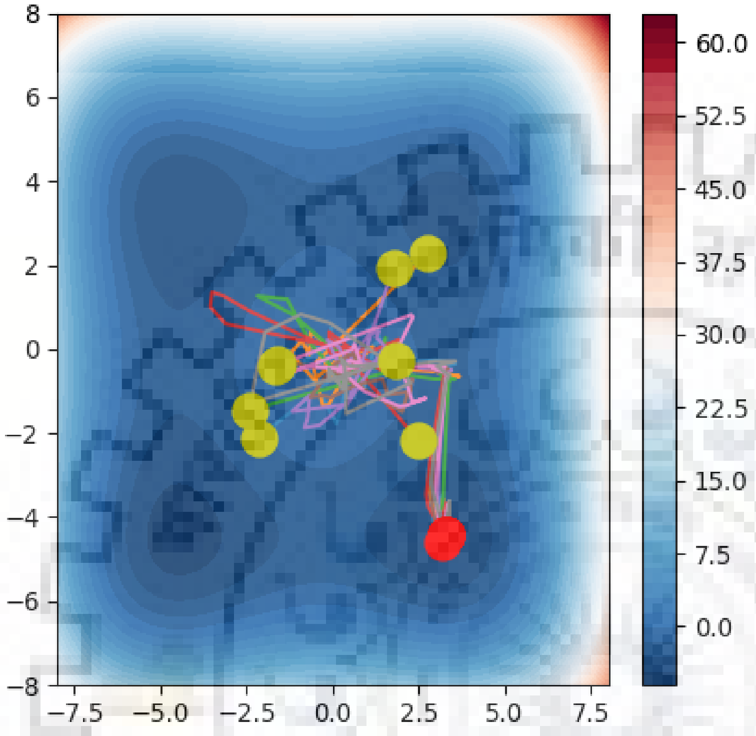


Figure 12: Paths taken by the individual minimizers for Test run #2

The solution space is not explored as well in this test and the algorithm moves quickly towards convergence.

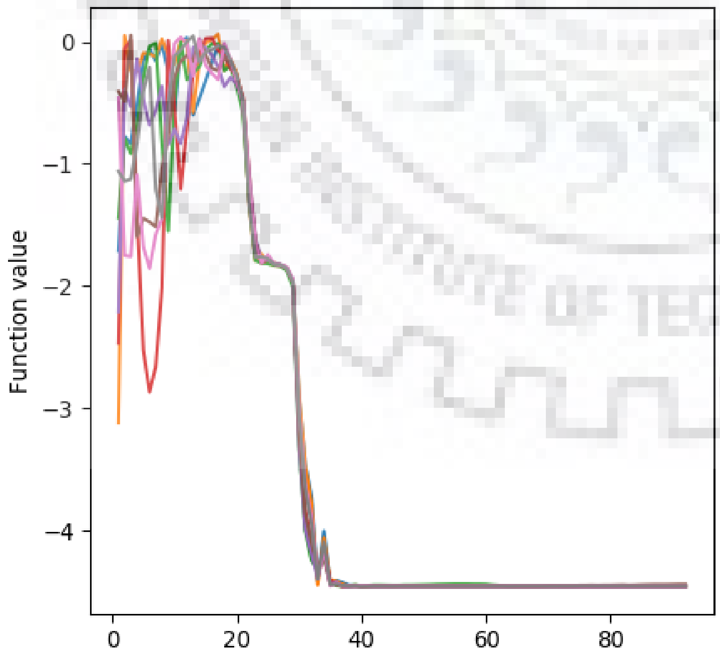


Figure 13: Function Value vs Iterations for Test run #2

We see a greatly reduced number of iterations and quick convergence towards the global optimum without wasting computational resources. In this case, the tuning parameters can be said to have been better adjusted.

4.6 Potential Geophysical Applications

Since CLM proves to be an efficient and robust algorithm, it can be a very useful tool in inverting geophysical data and constructing accurate models.

Degrande et al (2008) have demonstrated this by utilizing CLM based inversion in Spectral Analysis of Surface Waves (**SASW**).

The fast convergence property that CLM gets by using derivative based methods for local optimization are highly beneficial to geosciences since it typically contains huge datasets for example in Seismic and EM data.

On the other hand the parallelism can also highly benefit the speed of obtaining solutions because of strong parallel processing capabilities in today's machines.

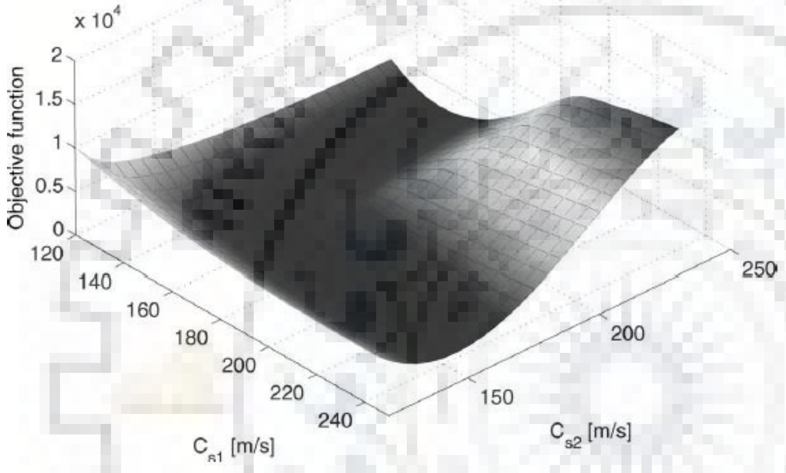


Figure 14: Objective function based on fundamental mode in Degrande et al (2008)

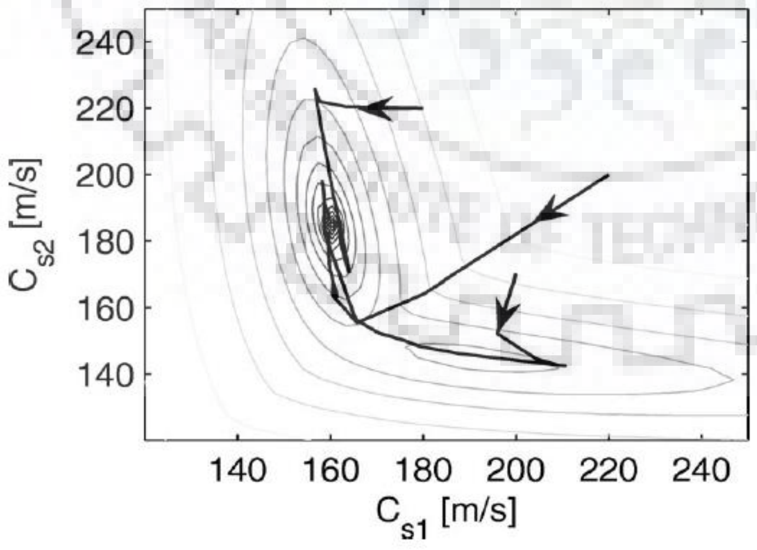


Figure 15: CLM optimization of the function

CONCLUSION

5.1 Summary of Results

A relatively uncommon but efficient method of global optimization is investigated. The CLM algorithm, which combines the benefits from derivative information based fast-convergence methods and the information exchange and parallelism found in Global Optimization metaheuristics, was successfully designed and implemented to minimize a 2-dimensional function. The global convergence to a single point is enforced using Augmented Lagrangian method and boundary constraints. Instead of Trust Region approach and Newton's method for convergence as used in Teughels et al (2003), we used the Line search based Quasi-Newton method known as Broyden–Fletcher–Goldfarb–Shanno (**BFGS**) Algorithm which arguably provides better performance due to faster convergence and no requirement of computing the exact Hessian Matrix.

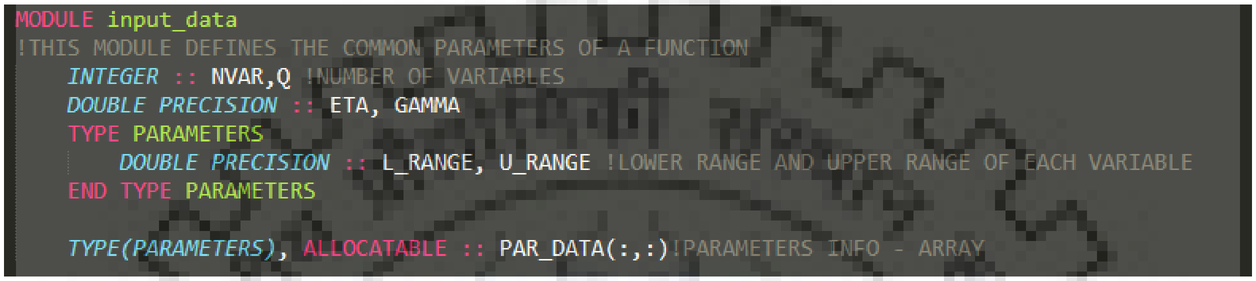
The population of minimizers is shown to converge at the global minima regardless of their initial start positions showing the robustness of the CLM method. The effect of tuning parameters is also demonstrated.

5.2 Discussion

Despite being relatively old, CLM hasn't seen many users in an era dominated by Global Optimization Metaheuristics such as SA and GA. As discussed in section 4.6 there are possible geophysical applications in every major field and the specific advantages CLM offers are very well suited to datasets and inversion models in Earth Sciences. More work needs to be done in finding a general solution to the problem of fine-tuning parameters and utilize the parallelism for tangible computational benefits.

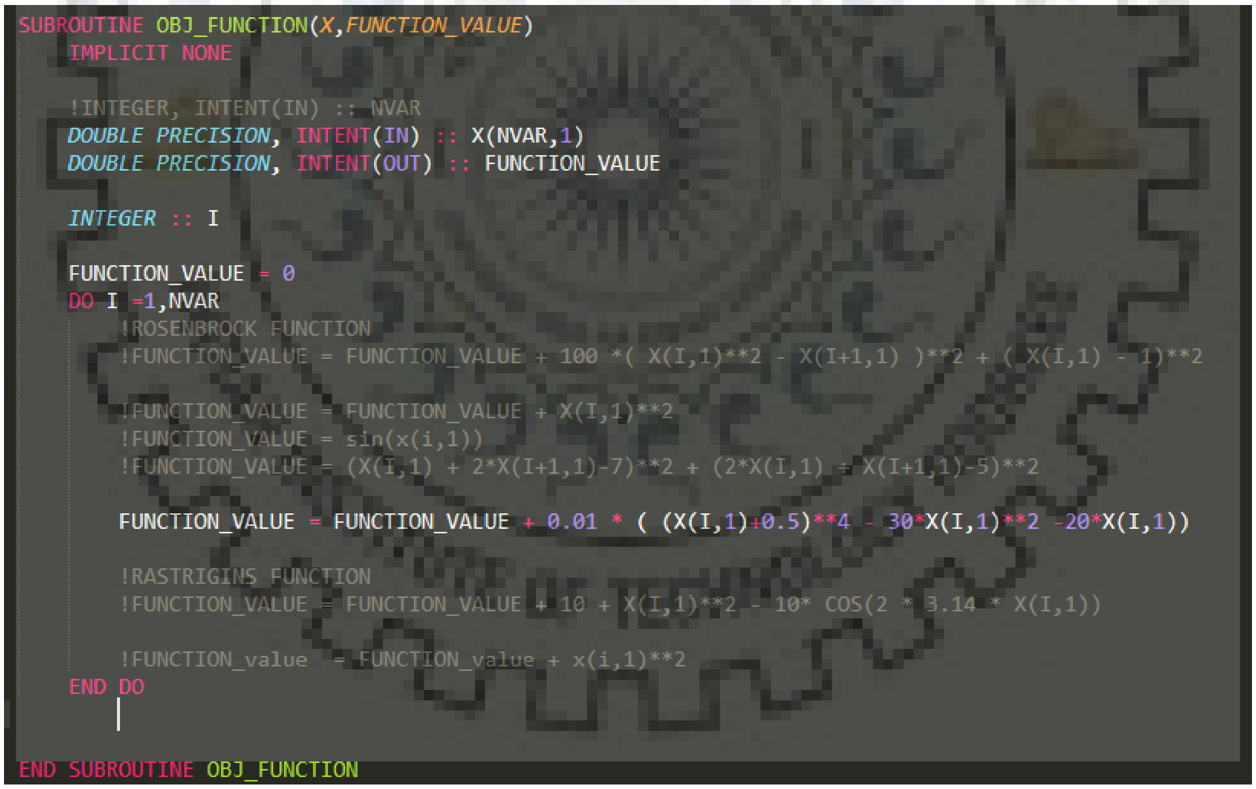
APPENDIX A: CODE SNIPPETS

In this appendix are included snippets from the code as including complete code would be too tedious and consume too much space.



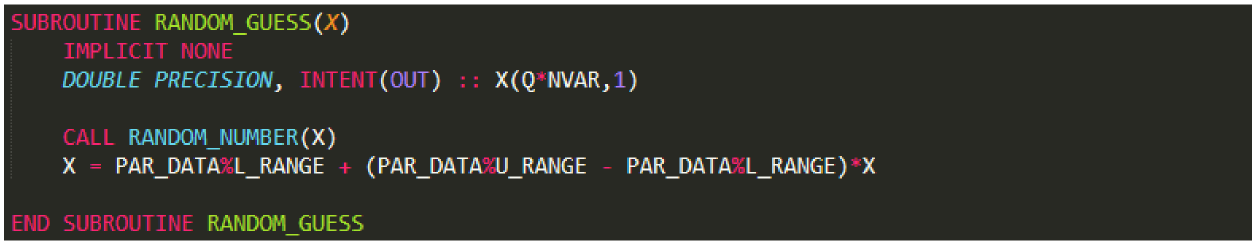
```fortran
MODULE input_data
!THIS MODULE DEFINES THE COMMON PARAMETERS OF A FUNCTION
    INTEGER :: NVAR,Q !NUMBER OF VARIABLES
    DOUBLE PRECISION :: ETA, GAMMA
    TYPE PARAMETERS
        DOUBLE PRECISION :: L_RANGE, U_RANGE !LOWER RANGE AND UPPER RANGE OF EACH VARIABLE
    END TYPE PARAMETERS

    TYPE(PARAMETERS), ALLOCATABLE :: PAR_DATA(:,:)!PARAMETERS INFO - ARRAY
```

Figure 16: Input data initialization:



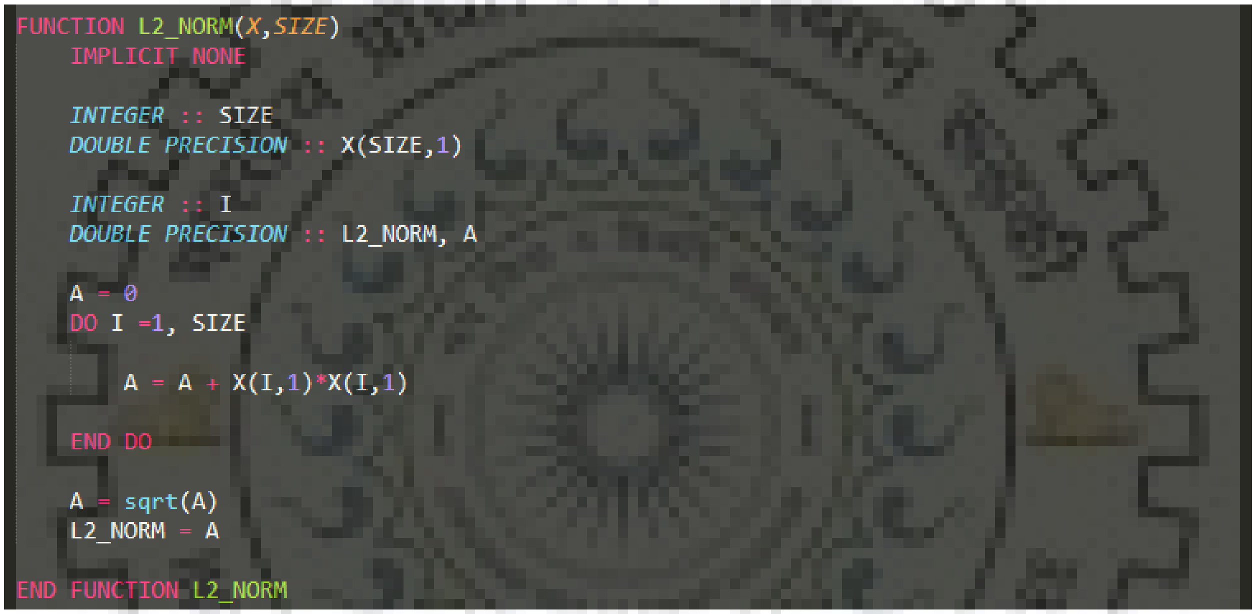
```fortran
SUBROUTINE OBJ_FUNCTION(X,FUNCTION_VALUE)
    IMPLICIT NONE

    !INTEGER, INTENT(IN) :: NVAR
    DOUBLE PRECISION, INTENT(IN) :: X(NVAR,1)
    DOUBLE PRECISION, INTENT(OUT) :: FUNCTION_VALUE

    INTEGER :: I

    FUNCTION_VALUE = 0
    DO I =1,NVAR
        !ROSENBROCK FUNCTION
        !FUNCTION_VALUE = FUNCTION_VALUE + 100 *( X(I,1)**2 - X(I+1,1) )**2 + ( X(I,1) - 1)**2

        !FUNCTION_VALUE = FUNCTION_VALUE + X(I,1)**2
        !FUNCTION_VALUE = sin(x(i,1))
        !FUNCTION_VALUE = (X(I,1) + 2*X(I+1,1)-7)**2 + (2*X(I,1) + X(I+1,1)-5)**2

        FUNCTION_VALUE = FUNCTION_VALUE + 0.01 * ( (X(I,1)+0.5)**4 - 30*X(I,1)**2 -20*X(I,1))

        !RASTRIGINS FUNCTION
        !FUNCTION_VALUE = FUNCTION_VALUE + 10 + X(I,1)**2 - 10* COS(2 * 3.14 * X(I,1))

        !FUNCTION_value  = FUNCTION_value + x(i,1)**2
    END DO


END SUBROUTINE OBJ_FUNCTION
```

Figure 17: Function defined as an object. The greyed out functions were also tested separately



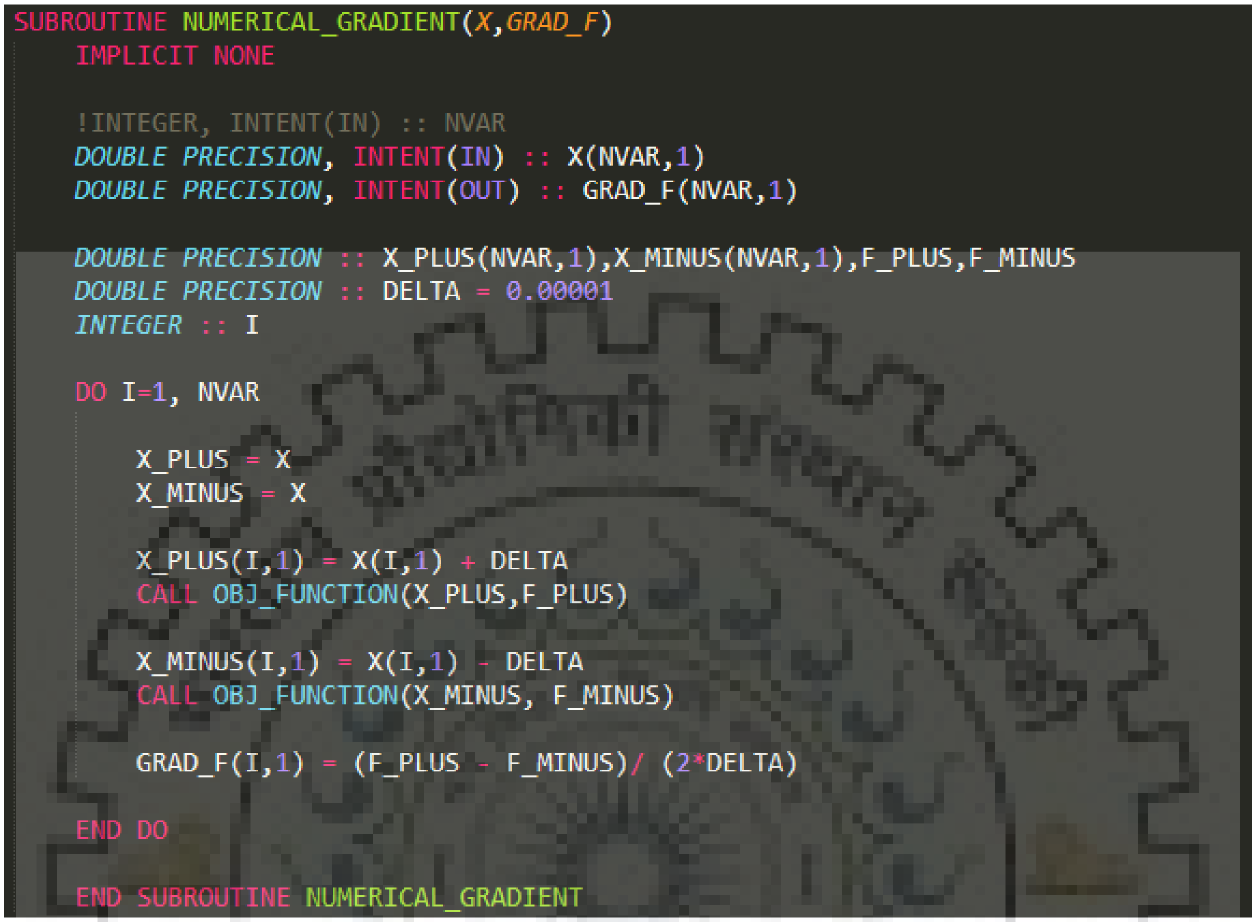
```
SUBROUTINE RANDOM_GUESS(X)
    IMPLICIT NONE
    DOUBLE PRECISION, INTENT(OUT) :: X(Q*NVAR,1)

    CALL RANDOM_NUMBER(X)
    X = PAR_DATA%L_RANGE + (PAR_DATA%U_RANGE - PAR_DATA%L_RANGE)*X

END SUBROUTINE RANDOM_GUESS
```

Figure 18: Random guess function for search point initialization



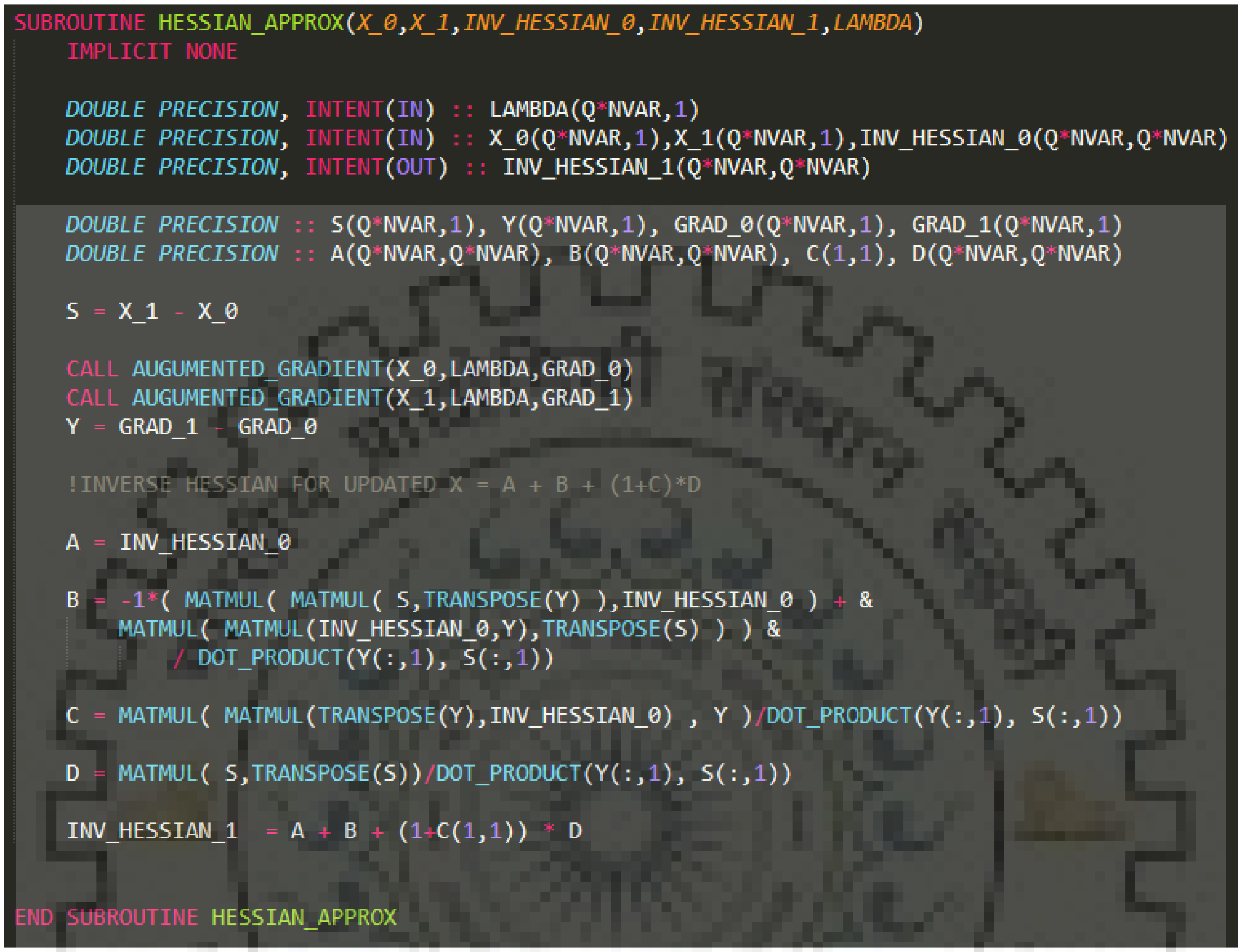
```
FUNCTION L2_NORM(X,SIZE)
    IMPLICIT NONE

    INTEGER :: SIZE
    DOUBLE PRECISION :: X(SIZE,1)

    INTEGER :: I
    DOUBLE PRECISION :: L2_NORM, A

    A = 0
    DO I =1, SIZE

        A = A + X(I,1)*X(I,1)

    END DO

    A = sqrt(A)
    L2_NORM = A

END FUNCTION L2_NORM
```

Figure 19: L2 Norm calculation



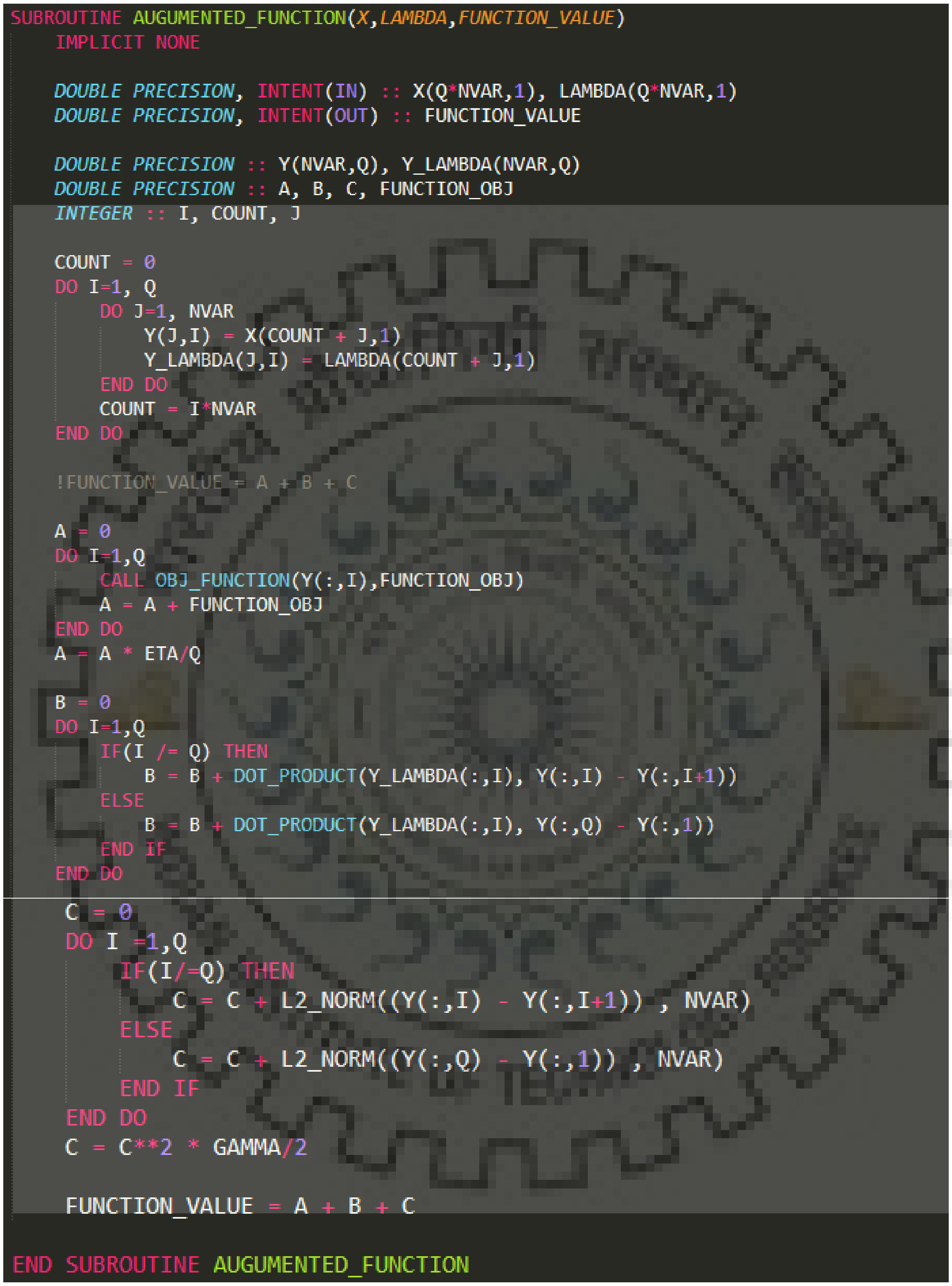
```
SUBROUTINE NUMERICAL_GRADIENT(X,GRAD_F)
    IMPLICIT NONE

    !INTEGER, INTENT(IN) :: NVAR
    DOUBLE PRECISION, INTENT(IN) :: X(NVAR,1)
    DOUBLE PRECISION, INTENT(OUT) :: GRAD_F(NVAR,1)

    DOUBLE PRECISION :: X_PLUS(NVAR,1),X_MINUS(NVAR,1),F_PLUS,F_MINUS
    DOUBLE PRECISION :: DELTA = 0.00001
    INTEGER :: I

    DO I=1, NVAR

        X_PLUS = X
        X_MINUS = X

        X_PLUS(I,1) = X(I,1) + DELTA
        CALL OBJ_FUNCTION(X_PLUS,F_PLUS)

        X_MINUS(I,1) = X(I,1) - DELTA
        CALL OBJ_FUNCTION(X_MINUS, F_MINUS)

        GRAD_F(I,1) = (F_PLUS - F_MINUS)/ (2*DELTA)

    END DO

    END SUBROUTINE NUMERICAL_GRADIENT
```

Figure 20: Gradient Calculation



```
SUBROUTINE HESSIAN_APPROX(X_0,X_1,INV_HESSIAN_0,INV_HESSIAN_1,LAMBDA)
    IMPLICIT NONE

    DOUBLE PRECISION, INTENT(IN) :: LAMBDA(Q*NVAR,1)
    DOUBLE PRECISION, INTENT(IN) :: X_0(Q*NVAR,1),X_1(Q*NVAR,1),INV_HESSIAN_0(Q*NVAR,Q*NVAR)
    DOUBLE PRECISION, INTENT(OUT) :: INV_HESSIAN_1(Q*NVAR,Q*NVAR)

    DOUBLE PRECISION :: S(Q*NVAR,1), Y(Q*NVAR,1), GRAD_0(Q*NVAR,1), GRAD_1(Q*NVAR,1)
    DOUBLE PRECISION :: A(Q*NVAR,Q*NVAR), B(Q*NVAR,Q*NVAR), C(1,1), D(Q*NVAR,Q*NVAR)

    S = X_1 - X_0

    CALL AUGUMENTED_GRADIENT(X_0,LAMBDA,GRAD_0)
    CALL AUGUMENTED_GRADIENT(X_1,LAMBDA,GRAD_1)
    Y = GRAD_1 - GRAD_0

    !INVERSE HESSIAN FOR UPDATED X = A + B + (1+C)*D

    A = INV_HESSIAN_0

    B = -1*( MATMUL( MATMUL( S,TRANSPOSE(Y) ),INV_HESSIAN_0 ) + &
        MATMUL( MATMUL(INV_HESSIAN_0,Y),TRANSPOSE(S) ) ) &
            / DOT_PRODUCT(Y(:,1), S(:,1))

    C = MATMUL( MATMUL(TRANSPOSE(Y),INV_HESSIAN_0) , Y )/DOT_PRODUCT(Y(:,1), S(:,1))

    D = MATMUL( S,TRANSPOSE(S))/DOT_PRODUCT(Y(:,1), S(:,1))

    INV_HESSIAN_1  = A + B + (1+C(1,1)) * D


END SUBROUTINE HESSIAN_APPROX
```

Figure 21: Hessian Approximator



```fortran
SUBROUTINE AUGUMENTED_FUNCTION(X,LAMBDA,FUNCTION_VALUE)
    IMPLICIT NONE

    DOUBLE PRECISION, INTENT(IN) :: X(Q*NVAR,1), LAMBDA(Q*NVAR,1)
    DOUBLE PRECISION, INTENT(OUT) :: FUNCTION_VALUE

    DOUBLE PRECISION :: Y(NVAR,Q), Y_LAMBDA(NVAR,Q)
    DOUBLE PRECISION :: A, B, C, FUNCTION_OBJ
    INTEGER :: I, COUNT, J

    COUNT = 0
    DO I=1, Q
        DO J=1, NVAR
            Y(J,I) = X(COUNT + J,1)
            Y_LAMBDA(J,I) = LAMBDA(COUNT + J,1)
        END DO
        COUNT = I*NVAR
    END DO

    !FUNCTION_VALUE = A + B + C

    A = 0
    DO I=1,Q
        CALL OBJ_FUNCTION(Y(:,I),FUNCTION_OBJ)
        A = A + FUNCTION_OBJ
    END DO
    A = A * ETA/Q

    B = 0
    DO I=1,Q
        IF(I /= Q) THEN
            B = B + DOT_PRODUCT(Y_LAMBDA(:,I), Y(:,I) - Y(:,I+1))
        ELSE
            B = B + DOT_PRODUCT(Y_LAMBDA(:,I), Y(:,Q) - Y(:,1))
        END IF
    END DO
    C = 0
    DO I =1,Q
        IF(I/=Q) THEN
            C = C + L2_NORM((Y(:,I) - Y(:,I+1)) , NVAR)
        ELSE
            C = C + L2_NORM((Y(:,Q) - Y(:,1)) , NVAR)
        END IF
    END DO
    C = C**2 * GAMMA/2

    FUNCTION_VALUE = A + B + C

END SUBROUTINE AUGUMENTED_FUNCTION
```

Figure 22: Augmented Lagrangian function definition

REFERENCES

Degrande Geert [et al.] Application of the Coupled Local Minimizers Method to the Optimization Problem in the Spectral Analysis of Surface Waves Method [Journal]. - [s.l.] : JOURNAL OF GEOTECHNICAL AND GEOENVIRONMENTAL ENGINEERING, 2008.

Kumar D Nagesh Optimization Methods: M1L3. - [s.l.] : IISc Bangalore.

Nocedal Jorge and Wright Stephen J Numerical Optimization [Book]. - [s.l.] : Springer.

Suykens Johan A.K., Vandewalle Joos and Moor Bart De Intelligence and Cooperative Search by Coupled Local Minimizers [Journal]. - [s.l.] : Int. J. Bifurcation and Chaos, 2001.

Teughels Anne, Roeck Guido De and Suykens Johan A. K. Global optimization by coupled local minimizers and its application to FE model updating [Journal] // Computers and Structures. - 2003.

University of British Columbia UBC Calculus Online [Book].

University of Wisconsin [Online] // NEOS Guide .

You Fengqi Northwestern University Process Optimization Open Textbook [Book].