**PARAMETER IDENTIFICATION USING A FINITE ELEMENT MODEL & INVERSE ALGORITHMS**

**Personal Programming Project**

by

Omkar Junnarkar

Matriculation Nr.: 66157

Under Guidance of:

Dr.-Ing. Arun Prakash

Dr.-Ing. Stefan Prüger

&

Supervision of:

Dr.-Ing. Martin Abendroth

Institut für Mechanik und Fluiddynamik

TU Bergakademie Freiberg

April 2022

**Table of Contents**

**Abstract**

In this Programming Project, an implementation of three Parameter Identification Algorithms has been done using two Finite Element Models as their base. The Levenberg-Marquardt Algorithm (also known as Damped Least-Squares) is one of the robust non-linear least squares methods which qualitatively interpolates between Gauss Newton Method and Gradient Descent Method. It can be seen as an advanced implementation of Gauss Newton Method with a trust-region approach. In the regression of well-behaved functions or the implementation where starting parameters are close to the actual parameters, Gauss Newton Method works faster. Gradient Descent Method (also known as steepest descent), which is a first order iterative optimization algorithm, can be fast for continuous functions with lower order of non-linearity, but, can show very slow or sometimes no convergence for complex functions. A Non-Linear Finite Element Model in this project serves as a source for Data Generation (displacement of nodes), and as a function for each iteration. An elastic-viscoplastic material routine has been implemented in the FEM code. Two 1-D Truss geometries have been used for parameter identification. The program has been written in C++ to make it computationally efficient. Library such as Eigen have been used to perform Linear Algebraic computations and operations. To overcome plotting challenges, an interface connecting CPP and GNUPLOT have been used via input-output-stream Thus, the behaviour of these three methods for two problems can be observed by such an implementation.

**Key Words**: Parameter Identification, Non-Linear Regression, Trust-Region Approach, Non-Linear Finite Element Method, Visco-plastic Material

**Chapter 1**

**Introduction**

Since the advent of Computational Methods, they are routinely used for scientific computing and various problems related to Engineering. By creating a virtual physical environment in the numerical space, such methods are utilized to reflect physical behaviours. Parameter Identification Algorithms are algorithms which deal with such numerical space and compute the Material Parameters from the experimental data. During the process, simulation of the experiments (Finite Element Analysis in this case) is performed for numerous iterations. Thus, to check the application & compare behaviour of such parameter identification methods coupled with finite element simulations, is the main goal of this project. The displacement data of the single node obtained by a virtual experiment (FEM) is used as an input to the Parameter Identification Algorithms - Levenberg-Marquardt, Gauss-Newton, & Gradient Descent. As a result, the three parameters of the elastic-viscoplastic material - Young’s Modulus, Yield Stress, and Viscosity are determined.

**Chapter 2**

**Theory**

The theoretical/scientific background for the project includes the Levenberg-Marquardt, Gauss-Newton, & Gradient Descent Methods, the non-linear Finite Element Method and elastic-viscoplastic material model.

**2.1 Levenberg-Marquardt Method**

The Levenberg-Marquardt method solves the nonlinear least square problems. It is analogous to a trust region-based method, where the step size towards the global minimum is determined by the damping factor Lambda. It combines the two minimization routines: Gauss-Newton & Gradient Descent. When the parameters are far from their optimal value, it acts more like gradient-descent method; and acts more like Gauss-Newton Method when parameters are close to their optimal value. The main equation, which is solved in each iteration is: -

: Jacobian Matrix  
: Weights   
: Identity Matrix  
: Damping Factor (positive scalar)   
: Computed change in parameters  
: Measured data  
: Curve-fit function data

Jacobian Matrix is the matrix which contains the values of partial derivatives of functions w.r.t the parameters at all data points. We basically solve the above equation for .

**2.2 Gauss-Newton Method**

The Gauss-Newton method basically minimizes the sum-of-squares kind of objective function. For moderately sizes problems, this method converges faster than the gradient descent method. The main equation, which is solved in during every iteration is: -

: Computed change in parameters

We basically solve the above equation for . If the first bracket term is not positive definite, the problem is supposed to be ill-posed. As a solution, we can use some perturbation or weight addition.

**2.3 Gradient-Descent Method**

The steepest descent method which works on the principle of ‘downhill’ minimization, i.e., the step direction is opposite to where the gradient of the objective function points. The method is well suitable for simple objective functions. The main equation, which is solved in each iteration is: -

: Computed change in parameters  
: length of step (positive scalar)

**2.4 Non-linear Finite Element Method**

The finite element method solves numerically the differential equations, in the field of interest of our case, structural analysis. It considers various sources of nonlinearity such as material, geometry, and boundary conditions such as contact. Here, material nonlinearity is taken into consideration. In such a method, we solve a non-linear system of equations using for example, the Newton-Raphson method. The method shows quadratic convergence with the equation:

which then reduces to linear system of equations

solved for .

**2.4 Elastic-Viscoplastic Material**

An integral part of structural analysis and mechanics of materials is the Material Modelling. The material model for an elastic-viscoplastic material is rate dependent. The constitutive equation for the elastic-viscoplastic model is based on two principles: Hooke’s law for linear elasticity is always valid; total strain has an additive decomposition of elastic & plastic strain. The considered constitutive equations & conditions are as follows:

: Young’s Modulus  
: Initial Yield Stress  
: Viscosity  
: Viscosity exponent;   
: McAuley Brackets

Chart

Description automatically generated

**Fig.1** Stress-Strain Response of 1-D Tensile loading of an elastic-viscoplastic material

**Chapter 3**

**Numerical Methods & Implementation Details**

**3.1 Implementation of Finite Element Method**

Two geometries are considered for the finite element simulation.

3.1.1 Finite Element Code for 2 Bar Geometry

**Fig.2** Geometry & Loading for 2 Bar FEA

Area-1

Area-2

**Force**

Length-1

Length-2

**F**

**t**

Two segments of bar, with different lengths & cross-sectional areas are loaded with linearly increasing force ‘F’, having an elastic-viscoplastic material mentioned in Section 2.4.

3.1.1.1 B-Matrix

For 1-D case, the B-Matrix can be derived as , where is the linear shape function array. This reduces to

3.1.1.2 Assignment Matrix

For a bi-nodal linear truss element, the assignment matrix is of the shape [2 x No. of Elements]. It is used to ensure continuity of nodal/elemental properties which are adjacent.

3.1.1.3 Seeding/Meshing

In the considered geometry, the centre node has force application, and the area of all elements before the node, and after the node are equal **respectively**. Thus, the seeding is implemented following the same idea.

3.1.1.4 Material Point Computations

Material Routine with the constitutive equations in Section 2.4 can be implemented to get updated stress/strain & stiffness values for all the elements. One gauss point is considered in the problem, thus the weight for single gauss point is 1. Summation over the nodes gives us equivalent gauss weight 2.

Internal Force  **,** where = (length of element)/2 &  **=** weight

Stiffness Matrix

Global Stiffness Matrix

Now, for computing properties for each element we divide our time steps into further smaller time steps and use discretized form of constitutive equations. Using Euler implicit formulation for plastic strain, we get

And for the tangent stiffness in plastic zone,

The material routine is implemented as follows:

**Fig.3** Material Routine: Time Discretization

Element Loop, ,  
h=; FOR k=100 times, do:

Euler Explicit Update (once, for each k)

Euler Implicit Update (until error<1e-5)

,

Hence, our Material Routine finally delivers Global Tangent Stiffness, Global Internal Force, Plastic Strains, Total Strains, & Stresses.

3.1.1.5 Main Program Flow

Main Program

Read / Initialize Quantities from User Input File

Element Routine

Geometry Data

Meshed Geometry

Writing Input Files, Looping through Loading steps

Load Step

Newton Raphson Iteration

Entering Newton Raphson   
Iteration Loop with initial u=0

Material Routine

Material Parameters

Tangent Stiffness, Internal Force, Plastic Strain, Total Strain, Stress

Apply Boundary Conditions (Matrix Reduction), Residual Computation,  
 Computation

Calls Element Routine: For B-Matrix,  
Assignment Matrix

Update with

Max. Iterations Reached ?

Max. Force Reached ?

YES

YES

Plotting Routine (gnuplot-iostream Interface)

Write Output Files

END

NO

NO

Force Update

Next Iteration

**Fig.4** FEA Program Flow

3.1.2 Finite Element Code for Tapered Bar Geometry

**Fig.5** Geometry & Loading for Tapered Bar FEA

Area-1

Area-2

**Force**

Length

**F**

**t**

A tapered bar, with defined cross-sectional areas at both ends is loaded with linearly increasing force ‘F’, having an elastic-viscoplastic material mentioned in Section 2.4. The whole finite element program essentially remains the same as that for the section 3.1.1, but the Element Routine which deals with geometry and Meshing changes. The Tapered Bar is approximated to be made up of truss elements of uniformly decreasing cross-sections as shows below:

**Fig.6** Approximated Geometry with ‘n’ elements for Tapered Bar

Area-1

Area-2

**Force**

Length

**R2**

**R1**

L

Φ

Now, again approximating the cross section to be a ‘circle’, the radii change of the circle can be computed according to the taper angle.

Thus, for nth element, we can approximate the radius to be & corresponding area to be . Length is equal for all the elements, and the force is applied on the node at centre of geometry.

**3.2 Implementation of Levenberg-Marquardt Algorithm**

Main Program

LMA Routine

Plotting Routine

Main Program provides the Initial Guess, Input and Measured Data to the LMA routine.

Levenberg-Marquardt Algorithm

Compute error between measured data and estimated data, Define damping factor

Compute Jacobian, Hessian

Compute change in parameters, Temporary Update Parameters, Evaluate Error

Error Decreased ?

Decrease Damping Factor, Accept Parameter Update, Error Criterion fulfilled ?

YES

NO

Increase Damping Factor, Reject Parameter Update

Return Parameters

NO

Estimated Data computed by a separate function, which accepts input data points & parameters as arguments

**Fig.7** Levenberg-Marquardt Program Flow

The tricky aspect in the algorithm is the updating scheme and Damping factor propagation. The parameter change is computed by solving the equation in Section 2.1. The initial value for damping factor must not be too large which would make a bigger leap in search direction, and not too small that it converges only to a local minimum without expanding its trust region.

**3.3 Implementation of Gauss-Newton Algorithm**

Main Program

GN Routine

Plotting Routine

Main Program provides the Initial Guess, Input and Measured Data to the Gauss-Newton routine.

The tricky aspect in the implementation of this method is the solving of equation in Section 2.2, to obtain the change in parameters. Due to Hessian, for many non-linear problems, turns out to be NOT positive definite. This makes the problem ill posed. To solve such an issue, either we add a diagonal matrix with weights, which can make its inverse possible, OR use the complete orthogonal decomposition and compute the Pseudo Inverse of Hessian. The former is as good as solving the equation for LMA; the later one can be used to test the Gauss-Newton computation.

Gauss-Newton Algorithm

Compute error between measured data and estimated data

Compute Jacobian, Hessian

Compute change in parameters, Update Parameters, Evaluate Error

Error Criterion fulfilled ?

NO

Return Parameters

YES

Estimated Data computed by a separate function, which accepts input data points & parameters as arguments

**Fig.8** Gauss Newton Program Flow

**3.4 Implementation of Gradient-Descent Method**

Gradient Descent Method

Initialize Step size, compute error between measured data and estimated data

Compute Jacobian

Compute change in parameters, Update Parameters, Evaluate Error

Error Criterion fulfilled ?

NO

Return Parameters

YES

Estimated Data computed by a separate function, which accepts input data points & parameters as arguments

**Fig.9** Gradient Descent Program Flow

The learning rate/step size is data sensitive. If the solution diverges OR Jacobian becomes non-positive definite, change the learning rate or initial guess of parameters. In the Parameter Identification Algorithm, a scheme where the step size is reduced if the error increases, is implemented. This prevents the algorithm to diverge. It can be stated as follows:

Error(n+1) > Error(n) ?

Reduce Step Size, Reject Parameter Update

YES

NO

**Fig.10** OptimizationofGradient Descent

**3.5 Implementation of Libraries/Programming aspects**

The IDE used during programming is Microsoft Visual Studio 2019, and the programming language C++17. For simplified application of Linear Algebraic computations, ‘Eigen3’ library is used. To read and write ‘.csv’ files, RapidCSV library & to write other file formats, the ‘fstream’ library is utilized. C++ does not have in-built plotting libraries. Thus, an interface linking software ‘gnuplot’ & the input output library ‘iostream’ helps overcoming this challenge.

3.5.1 EIGEN Library for Linear Algebra

EIGEN is a C++ template library for linear algebra: Matrices, Vectors, Numerical Solvers and related algorithms. It is a free software licensed under MPL2 (owned by TUXfamily) & is compatible with any compliant compiler.

Steps to use the library:

1. Set the configuration of IDE environment and the version to C++ 17 (not necessary for Eigen3, but for forthcoming libraries).
2. Download Eigen3 from [Eigen](https://eigen.tuxfamily.org/index.php?title=Main_Page) Main Page and unzip the folder.
3. Go to Properties (IDE Project) > C/C++ > Additional Include Directories > Add > Browse to the extracted Eigen folder; apply changes.
4. Add the header file: #include<Eigen/Dense>
5. Use the name space to reduce effort of specifying library/class: using namespace Eigen;
6. Refer to the documentation for data structures and its usage: [Eigen: Main Page](https://eigen.tuxfamily.org/dox/)

In the program, to avoid any possible computational errors for Matrix and Vector data types, only ‘MatrixXd’ data type is used which is expandable, & has ‘double’ data type as its elements.

3.5.2 RapidCSV Library for Data Parsing

RapidCSV is a C++ header-only library for CSV parsing, developed by Kristofer Berggren. It is a public repository and can be used for reading/writing files.

Steps to use the library:

1. Set the configuration of IDE environment and the version to C++ 17 (not necessary for rapidcsv, but for forthcoming libraries).
2. Download the code/clone the repository from: [d99kris/rapidcsv: C++ CSV parser library (github.com)](https://github.com/d99kris/rapidcsv) , unzip the folder.
3. Go to Properties (IDE Project) > C/C++ > Additional Include Directories > Add > Browse to the extracted rapidcsv folder; apply changes.
4. Add the header file: #include "src/rapidcsv.h"
5. Use the name space to reduce effort of specifying library/class: using namespace rapidcsv;
6. Refer to the documentation for data structures and its usage: https://github.com/d99kris/rapidcsv

‘Document’ is a class representing the CSV document; thus, we create an instance of it, giving the string of the file name as an argument. The ‘.csv’ file to be read should be in the same directory as the project. We read the column wise data into a ‘vector of double’ data type of standard library and write it to the Eigen Matrix for further usage.