**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[1] [2] [6]**

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: -

(2.1)

: Jacobian Matrix  
: Weights   
: Identity Matrix  
: Damping factor (positive scalar)   
: Computed change in parameters  
: Measured data  
: Curve-fit function data, function of input ‘x’ (data points) and parameters ‘p’  
: Parameters of function y

Jacobian matrix is the matrix which contains the values of partial derivatives of functions w.r.t the parameters at all data points. It can be visualized as:

We basically solve the above equation for .

**2.2 Gauss-Newton Method[1] [6]**

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: -

(2.2)

: 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[1] [6]**

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: -

(2.3)

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

**2.4 Non-linear Finite Element Method[4]**

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:

(2.4)

which then reduces to linear system of equations

(2.5)

solved for .

Here,

(2.6)

: Residual  
: Internal force  
: External force  
: Material parameters (in this case, E: Young’s modulus, : Yield stress, : Viscosity)  
: Displacement  
: Change in displacement  
t: time  
: Tangent stiffness matrix

**2.5 Elastic-Viscoplastic Material[3] [4]**

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:

(2.7)

(2.8)

(2.9)

: Young’s Modulus  
: Initial Yield Stress  
: Total strain   
: Plastic strain  
: Plastic strain rate  
: Stress   
: Viscosity  
: Viscosity exponent;   
: McAuley Brackets, where

Chart

Description automatically generated

**MPa**

**Fig.1** Stress-Strain Response of 1-D Tensile loading of an elastic-viscoplastic material (Image from one of the test cases of this project)

**Chapter 3**

**Numerical Methods & Implementation Details**

**3.1 Implementation of Finite Element Method[4]**

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(Length-1, Length-2) & cross-sectional areas(Area-1, Area-2) 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 where is the length of element. (3.1)

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**[4]**

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 (3.2 & 3.3)

Global Stiffness Matrix (3.4)  
Here, : cross sectional area of element, & : Assignment 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

(3.5)

, : Plastic strain & stress at current time step  
, : Plastic strain & stress of next step

And for the tangent stiffness () in plastic zone,

(3.6)

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(Area-1, Area-2) 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

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

(3.7)

(3.8)

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.

3.1.3 Files Generated by both FEM Programs

* All\_strain.csv, All\_stress.csv: Contains strain & stress values for all elements at all loading steps
* displacement\_2\_bar.csv, displacement\_tapered.csv: Displacement of centre node at all loading steps for 2 Bar & tapered bar geometry (in mm)
* InputFileFEA2Bar.txt: Input log, contains input parameters for computation
* LoadingSteps.csv: The loading step values (in Newton)
* ReactionForces.csv: The reaction forces at two ends at all loading points
* Residual.csv: Residual at end of Newton-Raphson iteration at all loading steps
* strain.csv, stress.csv: Strain and stress at the focus element for plotting
* B-Matrix.txt: B-matrices for all elements
* Assignment-Matrix.txt: Assignment matrices of all elements
* Areas-Lengths-After-Meshing.txt: The list of cross-sectional areas and lengths of all the elements.
* Stress-Strain and Displacement-Time step plot, which can be saved once displayed

**3.2 Implementation of Levenberg-Marquardt Algorithm**

Main Program

LMA Routine

Plotting Routine

**Fig.7** Levenberg-Marquardt (LMA) overview

Main program provides the initial guess input data points and measured data to the LMA routine.

First the input and measured data is generated using known parameters by the *DataGenerator.py*, which produces files ‘xdata.csv’ & ‘y\_measured.csv’. These are read by the main program and used as an input for LMA routine. To keep record of the function, *ActualParameterData.txt* is also produced by the python program.

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

YES

**Fig.8** 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.

Files created by the program:

* final\_fit.csv: Contains the fitted data points computed by the identifies parameters
* ErrorNorm.csv: Error at the end of each iteration
* Hessians.csv: The Hessian matrix in each iteration
* DampingFactorPropagation.csv: Traces the change in damping factor in each iteration
* OutputLog.txt: The output log, i.e., obtained parameters and initial guesses.
* Measured and fitted plot, which can be saved once displayed

**3.3 Implementation of Gauss-Newton Algorithm**

Main Program

GN Routine

Plotting Routine

**Fig.9** Gauss Newton Overview

Main program provides the initial guess, input data points and measured data to the Gauss-Newton routine.

Data is generated like the LMA routine and fed to the main program. The files created by the program are:

* final\_fit.csv: Contains the fitted data points computed by the identified parameters
* ErrorNorm.csv: Error at the end of each iteration
* Hessians.csv: The Hessian matrix in each iteration
* OutputLog.txt: The output log, i.e., obtained parameters and initial guesses.
* Measured and fitted plot, which can be saved once displayed

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.10** 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.11** 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.12** OptimizationofGradient Descent

Data is generated like the LMA & GN routine and fed to the main program. The files created by the program are:

* final\_fit.csv: Contains the fitted data points computed by the identifies parameters
* ErrorNorm.csv: Error at the end of each iteration
* OutputLog.txt: The output log, i.e., obtained parameters and initial guesses.
* Measured and fitted plot, which can be saved once displayed

**3.5 Implementation of Parameter Identification Program**

The final coupled parameter identification program consists of the following C++ files:

1. *Source.cpp*: The main program which is responsible for writing input/output logs, reading user input data from CSV files, and calling the algorithms and plotting routine.
2. *fea\_2\_ele\_main.cpp*, *fea\_tapered\_main.cpp*: The main program for 2 bar FEA function/program and tapered bar FEA function/program respectively.
3. *element\_2\_ele.cpp, element\_tapered.cpp*: The element routines of 2 bar and tapered bar geometries respectively.
4. *material.cpp*: The material routine (common for both FEA functions since only geometries change, the material type remains elastic-viscoplastic)
5. *NLR\_functions\_2\_ele.cpp, NLR\_functions\_tapered.cpp*: Files containing the functions required for parameter identification of 2 bar & tapered bar FEA program. Two different files are used for same functionality because the arguments for FEA Programs are different (the reason being different geometrical parameters).
6. *plotme.cpp*: This file contains the commands/code required for plotting the required fitted displacement vs. time curves.
7. Header files required to call functions from other ‘.cpp’ files, including the *gnuplot-iostream.h* (Author: Dan Stahlke).

The data is first generated using the finite element programs. The displacement files from the FEA directories are copied to the parameter identification directory, and the user input file is modified for the same parameters, which were the input to FEA Programs. The files created by this program are:

* LMA\_fit\_2ele.csv, LMA\_fit\_t.csv: Contains the fitted data points computed by the identifies parameters by Levenberg-Marquardt algorithm for both geometries
* GN\_fit\_2ele.csv, GN\_fit\_t.csv: Contains the fitted data points computed by the identifies parameters by Gauss Newton algorithm for both geometries
* GD\_fit\_2ele.csv, GD\_fit\_t.csv: Contains the fitted data points computed by the identifies parameters by Gradient Descent algorithm for both geometries
* InputFile\_NLR\_2Bar.txt, InputFile\_NLR\_TAPERED.txt: Input logs
* OutputLog.txt: The output log, i.e., obtained parameters and final % error between identified and actual Parameters.
* ComputationTime%%.txt: Computation times for various functions and operations
* Measured and fitted plot of displacement vs. time steps for all geometries and methods, which can be saved once displayed

The programming aspects which are featured in the above files are given in Section 3.6. The program flow is as follows:

Main Program

Read User Input File(Geometry & Experimental Parameters), Read Experimental Data(Displacement), Evaluate Computation Time

Levenberg Marquardt Algorithm for 2 Bar Experiment(calls FEA code in each iteration)

Gauss Newton Algorithm for 2 Bar Experiment(calls FEA code in each iteration)

Gradient Descent Method for 2 Bar Experiment(calls FEA code in each iteration)

Levenberg Marquardt Algorithm for Tapered Bar Experiment(calls FEA code in each iteration)

Gauss Newton Algorithm for Tapered Bar Experiment(calls FEA code in each iteration)

Gradient Descent Method for Tapered Bar Experiment(calls FEA code in each iteration)

Compute Displacement using Obtained Material Parameters (Young’s Modulus, Yield Stress, Viscosity) for both Geometries

Write Input/Output Log, Files for corresponding curve fit data

Call Plotting Routine

END

**Fig.13** Parameter Identification Program Flow

**3.6 Implementation of Libraries/Programming aspects[7]**

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.

3.6.1 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.

3.6.2 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. This library is utilized in the FEA and parameter identification program to read the user input data from ‘.csv’ 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.

3.6.3 fstream Library to read/write files

C++ provides this library for reading and writing from/to files. The stream class ‘ofstream’ is used to write on files. Firstly, the header #include<fstream> is to be added to the program. An instance of the class has to be created and the argument(naming the file and its format) should be specified. In the program, values which are to be written are appended to the instance name using the ‘<<’ operator. After writing the file, it should be closed using ‘.close()’ operator. More information for the usage of ‘fstream’ class can be gained from its documentation : [<fstream> typedefs | Microsoft Docs](https://docs.microsoft.com/en-us/cpp/standard-library/fstream-typedefs?view=msvc-170#fstream)

3.6.4 chrono Library to evaluate Computation time

The chrono library is one of the two types of time manipulation supports in C++. It is a flexible connection of types that track time with varying degrees of precision. To use this library, the header #include<chrono> is required. Also, ‘using namespace std::chrono’ can be added to reduce the effort of specifying the class. The two aspects of ‘chrono’ used in the program are ‘high\_resolution\_clock’ & ‘duration\_cast’. The usage of these is as follows:

At the starting point of time evaluation, we define a variable with ‘auto’ data type, which captures the time at that point, for eg.: auto start = high\_resolution\_clock::now();.

At the end point of time evaluation, we define another variable, for eg.: auto stop = high\_resolution\_clock::now();.

The time difference between two points then will give us the computation time elapsed, for eg.: auto duration\_total = duration\_cast<minutes>(stop - start); .

The unit of time can be specified as required. The computation time for various functions and operations is implemented in the parameter identification program. During the testing, the bottleneck in terms of computation time is identified in the testing section.

3.6.5 gnuplot-iostream Interface

The challenge of plotting in C++ is overcome using a header-only library by Dan Stahlke, namely ‘gnuplot-iostream.h’. The steps to use the plotting media using this library are as follows:

1. The language of the program is to be set to version C++17. Gnuplot-iostream interface works only with C++17 for all configurations.
2. Install the GNUPLOT software from its website: http://gnuplot.info/download.html
3. We need ‘Mircosoft vcpkg’ to manage the C++ library. For setting up vcpkg, clone the git repository from the terminal: https://github.com/Microsoft/vcpkg.git
4. After cloning is finished successfully, browse to the directory and type the command: .\bootstrap-vcpkg.bat
5. Install vcpkg boost for both configurations: vcpkg install boost:x64-windows boost:x86-windows
6. Browse to the directory where vcpkg.exe is present, use following commands to run executable: .\vcpkg.exe install boost:x64-windows boost:x86-windows
7. Integrate the executable: .\vcpkg.exe integrate install
8. From the git repository of Dan Stahlke, download the header file gnuplot-iostream.h: https://github.com/dstahlke/gnuplot-iostream
9. Copy the header file to the Working Directory and include it in the program/plotting routine: #include"gnuplot-iostream.h"

What this header file does is, it helps to connect the software GNUPLOT with the output stream of C++17, and hence allows to control it from C++ code. The documentation for this can be found on https://github.com/dstahlke/gnuplot-iostream/wiki , as well as more information on http://www.stahlke.org/dan/gnuplot-iostream.

We make an instance from the ‘gnuplotio space’ of ‘Gnuplot’ class, and as an argument give the path of ‘gnuplot.exe’ executable with double backslashes, for eg.: gp("\"C:\\Program Files\\gnuplot\\bin\\gnuplot.exe\"");

The instance ‘gp’ now takes in the commands in the form of strings, which are recognized by gnuplot syntax. The commands such as gp.send(getmeasured2ele); takes in the argument of vector data type, and plots. In the end of plotting routine, cin.get(); takes the final input. After calling the plotting function, the plot is displayed in a ‘gnuplot’ display window.

It is not possible to install vcpkg on Linux machines. In such cases, the plotting routine in the code can be disabled, and the similar commands from plotting routine can be used to plot using GNUPLOT.

The main reason to use gnuplot-iostream interface here is to perform computations and plotting both using the same script.

**Chapter 4**

**Testing, Verification & Result Discussion**

The testing of the code is done in parts, and the results are verified by comparing with the expected/analytical results.

Firstly, each function/routine used in the parameter identification program is tested individually: FEA codes for 2 bar & tapered Bar, Levenberg-Marquardt, Gauss-Newton & Gradient Descent method.

Once the results from these individual programs/routines are verified, the testing of parameter identification code can be proceeded. Thus, if individual routines work correctly, the coupled code is also expected to give correct results.

**4.1 FEA Code for 2 Bar Geometry**

4.1.1 Tests to Validate Element Routine Functions

The element routine deals with the meshing and elemental entities such as B-matrix & assignment matrix. Thus, for different geometrical dimensions and number of elements, the B-matrix for all elements, assignment matrix for all elements and the list of areas & lengths of meshed geometry were produced. The test cases can be found in the directory: *Test\_Element\_Routine\_Functions > Test-%-B-Mat\_A-Mat\_Areas\_Lengths* . Each test case contains the following files for verification, produced by the program during computation:

* InputFileFEA2Bar.txt: The input parameters used to produce the test results.
* B-Matrix.txt: B-matrices for all elements
* Assignment-Matrix.txt: Assignment matrices of all elements
* Areas-Lengths-After-Meshing.txt: The list of cross-sectional areas and lengths of all the elements.
* README.txt: Contains details regarding respective test.

Conclusion: The values were verified with hand calculation using the formulae mentioned in Section 3.1. For e.g.:

*Area of Bar 1, Bar 2 (mm^2) = 6, 12 ; Length of Bar 1, Bar 2 (mm) = 15, 40 ; No. of Elements = 5 ; Gauss Points = 1*

*Thus, the cross-sectional area after meshing will be: 6, 6, 12, 12, 12. And, the lengths of elements will be : 7.5, 7.5, 13.3333, 13.3333, 13.3333 (Since Bar 1 will be divided in 2 elements, and Bar 2 in 3).*

*The B-matrices for the elements will be according to mentioned in Section 3.1.1.1, as follows:*

*B - Matrix for Element [0] of length = 7.5 mm is : -0.133333 0.133333*

*B - Matrix for Element [1] of length = 7.5 mm is : -0.133333 0.133333*

*B - Matrix for Element [2] of length = 13.3333 mm is : -0.075 0.075*

*B - Matrix for Element [3] of length = 13.3333 mm is : -0.075 0.075*

*B - Matrix for Element [4] of length = 13.3333 mm is : -0.075 0.075*

*Also, the assignment matrices for all the elements will be:*

*Assignment Matrix for Element [0] in system of 5 Elements is :*

*1 0 0 0 0 0*

*0 1 0 0 0 0*

*Assignment Matrix for Element [1] in system of 5 Elements is :*

*0 1 0 0 0 0*

*0 0 1 0 0 0*

*Assignment Matrix for Element [2] in system of 5 Elements is :*

*0 0 1 0 0 0*

*0 0 0 1 0 0*

*Assignment Matrix for Element [3] in system of 5 Elements is :*

*0 0 0 1 0 0*

*0 0 0 0 1 0*

*Assignment Matrix for Element [4] in system of 5 Elements is :*

*0 0 0 0 1 0*

*0 0 0 0 0 1*

4.1.2 Tests to Validate the Global Stiffness Matrix/Tangent Stiffness

The global stiffness matrix Kt is assembled using the local stiffness matrix of each element according to the formulae in Section 3.1.1.4. The test cases are present in the directory: *Test\_Assembly\_Kt*. For the test, different parameters were used, and the resulting shape/ /coefficients of stiffness matrices compared with calculations. The matrices were displayed by adding a breakpoint in the code and running it in the debugger mode, which displayed the values in terminal. The snapshots taken contain the displayed local and global stiffness matrices in front, and the parameters used to produce them in the background.

* Test-1-2Ele-Kt.jpg: The image displays the element stiffness matrices for 2 elements, and the assembled global stiffness matrix.
* Test-2-3Ele-Kt.jpg: The image displays the element stiffness matrices for 3 elements, and the assembled global stiffness matrix.
* Test-3-4Ele-Kt.jpg: The image displays the element stiffness matrices for 4 elements, and the assembled global stiffness matrix.
* Test-4-10Ele-Kt.jpg: The image displays the element stiffness matrices for 10 elements, and the assembled global stiffness matrix.
* Test-5-11Ele-Kt.jpg: The image displays the element stiffness matrices for 11 elements, and the assembled global stiffness matrix.
* Test-1-Ct-ValueCheck.jpg: For the plastic state, the stiffness is no more valid. Thus, tangent stiffness is computed and used to evaluate the coefficients of stiffness matrix. The value is verified using the formulae for tangent stiffness in Section 3.1.1.4.
* Test-2-Ct-ValueCheck.jpg: For the plastic state, the tangent stiffness is computed, and the value is verified using the formulae for Tangent Stiffness in Section 3.1.1.4.
* Test-4-Elastic-Plastic-State-Ct&E.jpg: This test results show the global stiffness Matrix when bar 1 is in plastic state and bar 2 is in elastic state. Theoretically, the local stiffness for bar 1 will be tangent stiffness Ct. Thus, the correctness of global tangent stiffness is verified in such material state by performing this test.
* README.txt: Information about tests.

Conclusion: The stiffness matrix computation/assembly is giving the correct results for various parameters and conditions. For e.g.:

A screenshot of a computer

Description automatically generated with medium confidence

**Fig.14** Elastic-Plastic Stiffness Matrix Computation (Test-4-Elastic-Plastic-State-Ct&E.jpg)

4.1.3 Tests for Verification using Analytical Solution

For the considered geometry and boundary conditions, an analytical solution can be developed. The resultant FBD of the geometry can be deduced as:

P2

P1

F

**Fig.15** Reaction Forces acting on the body P1 & P2

(4.1)

(4.2)

(4.3)

The displacement of centre node computed from left and right bar data,

(4.4)

This results into & (4.5)

To solve the problem analytically, these equations have been written in a Python Program, *AnalyticalSolution2BarFEA.py*.

In the analytical solution program, the force iterates through a series of steps, like that of the FEA program, and the reaction forces, Stresses in the two Bars, and displacement is written in the *AnalyticalSolution.txt.* The values are written to the file, only until one of the bars is in plastic zone. The features which can be verified by comparing the results of FEA program and the analytical solution are:

* Force at which the stress reaches the yield limit.
* Stress & strain values for the points below yield limit
* Displacement at the point of yield limit
* Reaction forces at the fixed boundary conditions.

The test results are in the directory:

*Test\_Comparision\_Analytical\_Solution > Test-%-AnalyticalSol.*

Each test directory contains the following files:

* InputFileFEA2Bar.txt: Contains input parameters used to produce the test
* All\_stress.csv, All\_strain.csv: Stress & strains of both bars at all loading steps(in MPa)
* displacement\_2\_bar.csv: displacement of the centre node at all loading steps(in mm)
* LoadingSteps.csv: The loading step values (in Newton)
* ReactionForces.csv: The reaction forces at two ends at all loading points
* AnalyticalSolution.txt: Results of the analytical solution
* Mat0\_Test.pdf: The plot for stress vs. strain & displacement vs. time from the FEA program.
* README.txt: Information regarding the test.

Conclusion: The programs were run in variable parameters and the values from ‘.csv’ files generated by the FEA program & the ‘.txt’ file from analytical solution, on comparison, lead to similar results. Hence, the FEA Program was verified.

4.1.4 Testing for various Boundary Conditions

The FEM code should be tested for different sets boundary conditions to check its expandability. 5 Cases of boundary conditions are considered for this purpose.

Area-1

Area-2

**Force**

Case-1

Area-1

Area-2

**Force**

Case-2

Area-1=Area-2

**Force**

Case-3

Case-4

Area-1=Area-2

Case-5

Area-1=Area-2

**Fig.16** Set of Boundary Conditions for Testing FEM Code

To change the boundary conditions and force application points in the code, the main program has commented set of lines which are to be replaced. The ‘.block()’ function from the Eigen library is used here to reduce the matrix according to the required boundary conditions. For more information regarding the function, refer documentation or the code.

Conclusion: In all the boundary conditions, the FEM code gives results which confer to the literature (Viscoplastic behaviour of the homogenous stress state in tensile and compressive loading). Also, the residual values being under the criterion of || R || < 0.005 || F\_internal || show us that the Newton-Raphson converges for all the cases. The stress-strain & displacement-time steps plots resulting from the FEM computations are also present in the directory *Test\_VaryingBoundaryConditions.* The plots below, represent the stress-strain behaviour of bar-1 in all 5 cases. The plots look essentially the same for tension & compression cases (as expected) respectively, because only the point of application of force changes. The values for the stress in bar-2 for Case-1 & Case-2 can be distinguished in a way, that because of no fixed point/restriction applied to it, it has non-zero value to stress in Case-2 and ‘0’ stress in Case-1. This can be checked from the ‘All\_stress.csv’ in the directory *Test\_VaryingBoundaryConditions > Case-1 & Case-2,* in which the 2nd column shows the stress values for bar-2*.*

Chart, line chart

Description automatically generated Chart, line chart

Description automatically generated Chart, line chart

Description automatically generated

MPa

MPa

MPa

Chart, line chart, histogram

Description automatically generated Chart, line chart

Description automatically generated

MPa

MPa

**Fig.17** Stress-Strain Plots from all the Cases

The stress-strain, displacement values of all elements, input log and residuals can be found in the’.csv’ and ‘.txt’ files present in the Case-% directory.

4.1.5 Tests for Verification of Material behaviour with varying Strain Rate

Viscoplastic material is a rate dependent material. Here, it follows the power law. Thus, the increase in number of time steps for loading, will affect the stress strain behaviour of the material due to its viscous behaviour. Increasing the strain rate will increase the value of stress, i.e., decreasing the step size, or increasing the number of time steps will have this effect. This can be proven as follows:

The tangent stiffness equation here is: . Here, we can see an explicit relation of with time step size in the denominator. Thus, with lower time step size, effect of viscosity will reduce, and the tangent stiffness will increase. This will increase stress in the material.

Diagram

Description automatically generated

**Fig.18** Stress-Strain Plot for varying strain-rate**[5][B.Kiefer,Plasticity]**

The Case-3 boundary conditions shown in the section 4.1.4 are applied for performing the test. 5 tests were performed for the same material parameters & geometry, which turns out equivalent to a homogenous stress state tensile test. The number of time steps for the test are: 1000, 2500, 5000, 10000, 20000 steps. Thus, 5 directories in the *Test\_Strain\_Rate\_Case3 > %\_Steps* contain all the resulting files from the tests, which can be reproduced using the information in input log. README.txt has information regarding tests. The results are as follows in the order of increasing strain rate/number of time steps:

Chart

Description automatically generated Chart

Description automatically generated Chart

Description automatically generated

MPa

MPa

MPa

1. (b) (c)

Chart

Description automatically generated Chart

Description automatically generated

MPa

MPa

(d) (e)

**Fig.19** (a) 1000 (b) 2500 (c) 5000 (d) 10000 (e) 20000 Steps

|  |  |
| --- | --- |
| **Maximum Stress (MPa)** | **Number of Steps** |
| 523.235 | 1000 |
| 556.975 | 2500 |
| 571.326 | 5000 |
| 579.125 | 10000 |
| 583.236 | 20000 |

**Table.1** Stress Values for different strain rates

Conclusion: It can be concluded that the material routine confers to the theoretical aspects of the visco-plastic material which is verified by the above results.

4.1.6 Testing for various Parameter Sets

The final test intends to run the FEM code for the 2 bar geometry with various material and geometry parameters, in which the code should work and converge. 4 sets of parameters are tested here for which the FEM code is tested. The files produced by the program including the input log can be found in the directory *Test\_Mat\_Properties > Mat%.*

Conclusion: The FEM code converges for all the geometrical and material parameters.

An example is shown below:

Chart, line chart, histogram

Description automatically generated

mm

MPa

**Fig.20** Stress (MPa)-Strain, Displacement(mm)-Time step Curve for Mat2

4.1.7 Abaqus Result Comparison

As the last step of verifying the results, the material sets from Section 4.1.6, Mat0 & Mat1 were also implemented using the finite element software ABAQUS for the 2 bar Geometry. This resulted in the similar stress-strain behaviour which we see from the self-written FEM code. The directory *Abaqus\_Mat0\_Mat1\_Figure\_INP\_CAE* contains the following test files:

* Job-Mat0.inp, Job-Mat1.inp: Input files for the two tests
* Job-Mat0.odb, Job-Mat1.odb: Output data base for the two tests
* Mat0\_Stress.jpg, Mat1\_Stress.jpg: Stress states
* Mat0\_Graph.jgp, Mat1\_Graph.jpg: Stress-strain plots for two tests

Chart

Description automatically generated with medium confidenceChart, line chart, histogram

Description automatically generated

**Fig.21** Result from FEA Program

A picture containing line chart

Description automatically generatedA picture containing shape

Description automatically generated

Graphical user interface

Description automatically generated Graphical user interface, application

Description automatically generated

**Fig.22** Result from ABAQUS Simulation

Comparison for values can be done from the’.odb’ files in above mentioned directory, and the’.csv’ files from *Test\_Mat\_Properties > Mat0 & Mat1.*

Less emphasis has been put on the ABAQUS analysis, whereas more emphasis has been on the verification using other testing strategies.

Hence, carefully analysing the results of the following tests, the FEA code for the 2 bar geometry can be verified.

**4.2 FEA Code for Tapered Bar**

The FEA code for tapered bar has essentially only the meshing, i.e., the element routine different from the 2 bar geometry. Thus, its only important to check the overall behaviour of the code, and the element routine.

4.2.1 Tests to Validate Element Routine Functions

The element routine deals with the meshing and elemental entities such as B-matrix & assignment matrix. Thus, for different geometrical dimensions and number of elements, the B-matrix for all elements, assignment matrix for all elements and the list of areas & lengths of meshed geometry were produced. The test cases can be found in the directory: *Test\_Element\_Routine\_Functions > Test-%-B-Mat\_A-Mat\_Areas\_Lengths* . Each test case contains the following files for verification, produced by the program during computation:

* InputFileFEATaperedBar.txt: The input parameters used to produce the test results.
* B-Matrix.txt: B-matrices for all elements
* Assignment-Matrix.txt: Assignment matrices of all elements
* Areas-Lengths-After-Meshing.txt: The list of cross-sectional areas and lengths of all the elements.
* README.txt: Contains details regarding respective test.

Conclusion: The values were verified with hand calculation using the formulae mentioned in Section 3.1 & 3.2. For e.g.:

*Area of Left & Right End of Bar (mm^2) = 12, 4 ; Length of Bar (mm) = 180 ; No. of Elements = 10 ; Gauss Points = 1*

*Thus, the cross-sectional area after meshing will be: 12, 11.0071, 10.057, 9.14985, 8.28554, 7.4641, 6.68554, 5.94985, 5.25703, 4. And, the lengths of elements will be : 18,18,18,18,18,18,18,18,18,18*

The assignment matrix function and the B-matrix function remains same as of the FEA 2 bar code.

4.2.2 Testing for various Boundary Conditions

The FEM code should be tested for different sets boundary conditions to check its expandability. 5 cases of boundary conditions are considered for this purpose.

**Force**

Case-1

**Force**

Case-2

**Force**

Case-3

Case-4

Case-5

**Fig.23** Set of Boundary Conditions for Testing FEM Code

To change the boundary conditions and force application points in the code, the main program has commented set of lines which are to be replaced. The ‘.block()’ function from the Eigen library**[7]** is used here to reduce the matrix according to the required boundary conditions. For more information regarding the function, refer documentation or the code.

Conclusion: In all the boundary conditions, the FEM code gives results, where the residual values being under the criterion of || R || < 0.005 || F\_internal ||**[4]** show us that the Newton-Raphson converges for all the cases. The stress-strain & displacement-time steps plots resulting from the FEM computations are also present in the directory *Test\_VaryingBoundaryConditions.*

A picture containing timeline

Description automatically generated Timeline

Description automatically generated with medium confidence

MPa

MPa

A picture containing box and whisker chart

Description automatically generated Chart

Description automatically generated with medium confidence

MPa

MPa

A picture containing diagram

Description automatically generated

MPa

**Fig.24** Stress-Strain Plots from all the Cases

The stress-strain, displacement values of all elements, input log and residuals can be found in the’.csv’ and ‘.txt’ files present in the Case-% directory.

4.2.3 Testing for various Parameter Sets

The final test intends to run the FEM code for the tapered bar geometry with various material and geometry parameters, in which the code should work and converge. 3 sets of parameters are tested here for which the FEM code is tested. The files produced by the program including the input log can be found in the directory *Test\_Mat\_Properties > Mat%.*

Conclusion: The FEM code converges for all the geometrical and material parameters.

An example of output is shown below:

A picture containing timeline

Description automatically generated

mm

MPa

**Fig.25** Stress (MPa)-Strain, Displacement(mm)-Time step Curve for Mat3

**4.3 Levenberg-Marquardt Algorithm**

The Levenberg-Marquardt algorithm is tested for complex polynomials of different degrees. After generating the data, it is used as an input for the LMA, which gives back the parameters.

Expected results are the same parameters which were used to generate the data. 5 Tests were done to do this. For each case(equation), the *function\_y()* in *function.cpp* of the Levenberg Marquardt program needs to be modified. The OutputLog.txt in the test case directories in *Equations\_Error\_Test* shows the initial guess and obtained parameters. The test cases contain following files:

* ActualParameterData.txt: The actual function & parameters used to generate the data points
* Test%.pdf: The plot showing measured and fitted data points
* OutputLog.txt: Initial guess & obtained parameters
* Other files generated by program, mentioned in section 3.2

The test cases are as follows:

4.3.1 Test-1

(4.6)

|  |  |  |  |
| --- | --- | --- | --- |
| **Actual Para.** | **Initial Guess** | **Computed Para.** | **Error (%)** |
| 6.89 | -3.4 | 6.89 | 0 |
| -2.17 | 0 | -2.17 | 0 |
| 1.97 | 1 | 1.97 | 0 |

**Table.2** Parameter comparison LMA Test-1

The curve fit plot is in Test1.pdf.

Conclusion: The parameters obtained were exactly equal even if the initial guess was far away. The curve fit plot is in Test%.pdf.

4.3.2 Test-2

(4.7)

|  |  |  |  |
| --- | --- | --- | --- |
| **Actual Para.** | **Initial Guess** | **Computed Para.** | **Error (%)** |
| 0.745 | 0.4 | 0.79 | 6.040268 |
| 4 | 2 | 2.533 | 36.675 |
| 2 | 0.8 | 0.88 | 56 |

**Table.3** Parameter comparison LMA Test-2

The curve fit plot is as follows:

Chart, line chart, histogram

Description automatically generated

x

y(x)

**Fig.26** Curve fit for Test2 LMA

Conclusion: The error in this fit is because of the wavy nature of the function. The possible reason for that can also be that the average local minimum is gained but not the global minimum.

4.3.3 Test-3

The same equation for Test-2 is used but reducing the wavy nature of the equation, i.e., setting the coefficient of *cos()* to 1e-8.

|  |  |  |  |
| --- | --- | --- | --- |
| **Actual Para.** | **Initial Guess** | **Computed Para.** | **Error (%)** |
| 0.745 | 0.4 | 0.745 | 0 |
| 1e-8 | 8e-5 | 9.245e-9 | 7.55 |
| 2 | 0.9 | 0.900017 | 55 |

**Table.4** Parameter comparison LMA Test-3

The curve fit is as follows:

Chart

Description automatically generated

y(x)

x

**Fig.27** Curve fit for LMA Test-3

Conclusion: The fit is comparatively better but not exact. This is possibly due to the convergence at average local minimum of the function.

4.3.4 Test-4

(4.8)

|  |  |  |  |
| --- | --- | --- | --- |
| **Actual Para.** | **Initial Guess** | **Computed Para.** | **Error (%)** |
| 0.745 | 0.4 | 0.745 | 0 |
| 8.24 | 10 | 8.24 | 0 |
| -0.08 | -0.2 | -0.08 | 0 |
| 9.4 | 10 | 9.87557 | 5.059255 |
| 1 | 1.5 | 1 | 0 |
| -4 | -2.8 | -4.04935 | 1.23375 |

**Table.5** Parameter comparison LMA Test-4

The parameters here are evaluated using Damping factor = 10, just like all other tests.

4.3.5 Test-5

To test the significance of the damping factor, the same equation of Test-4 was fitted using lambda=0.0001. This resulted into an acceptable curve fit, but with different set of parameters: 1.37722, 5.77431, 2.17478, 570.638, 380.127, -2393.55.

Chart

Description automatically generated

x

y(x)

**Fig.28** Curve fit for LMA Test-5

Conclusion: The lambda reduced the search region which prevented an initial bigger leap to find the correct set of parameters. Hence, for the same curve fit. Different set of parameters were obtained.

Hence, the working of Levenberg-Marquardt algorithm was verified.

**4.4 Gauss Newton Algorithm**

The Gauss-Newton algorithm is tested for the same polynomials as LMA. After generating the data, it is used as an input for the GN, which gives back the parameters.

Expected results are the same parameters which were used to generate the data. 4 Tests were done to do this. For each case(equation), the *function\_y()* in *function.cpp* of the Gauss Newton program needs to be modified.

For the same equations used in LMA Testing, we obtain same results for the Gauss Newton algorithm. The OutputLog.txt in the test case directories in *Equations\_Error\_Test* shows the initial guess and obtained parameters.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Actual Para.** | **Initial Guess** | **Computed Para.** | **Error (%)** |
| **Test-1** | 6.89 | 1 | 6.89 | 0 |
| -2.17 | 0 | -2.17 | 0 |
| 1.97 | 1.2 | 1.97 | 0 |
| **Test-2** | 0.745 | 0.4 | 1.1455 | 53.75839 |
| 4 | 3 | -5.13614 | - |
| 2 | 1.1 | 1.33925 | 33.0375 |
| **Test-3** | 0.745 | 0.6 | 0.745 | 0 |
| 1.00E-08 | 8e-7 | 5.34E-09 | 46.6 |
| 2 | 1.8 | 1.80211 | 9.8945 |
| **Test-4** | 0.745 | 0.4 | 0.745 | 0 |
| 8.24 | 10 | 8.24 | 0 |
| -0.08 | -0.2 | -0.08 | 0 |
| 9.4 | 10 | 9.87598 | 5.063617 |
| 1 | 1.5 | 1 | 0 |
| -4 | -2.8 | -4.0494 | 1.235 |

**Table 6.** Parameter Comparison for Gauss Newton Tests

Conclusion: The Gauss Newton algorithm gives similar results to the Levenberg-Marquardt algorithm, and in most cases equal number of iterations. Thus, algorithm’s working is tested & verified.

**4.5 Gradient Descent Method**

The Gradient Descent method is tested with the same equations as of for the LMA & GN algorithms. This leads us to the following results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Actual Para.** | **Initial Guess** | **Computed Para.** | **Error (%)** |
| **Test-1** | 6.89 | 1 | 5.32184 | 22.7599 |
| -2.17 | 0 | -2.16387 | 0.28249 |
| 1.97 | 1.2 | 1.83841 | 6.6797 |
| **Test-2** | 0.745 | 1 | 0.568635 | 23.6732 |
| 4 | 3 | 3.69342 | 7.6645 |
| 2 | 1.5 | 2.02106 | 1.053 |
| **Test-3**  Same as Test-2 (with different Data Set Range) | | | 0.672178 | 9.77477 |
| 1.32901 | 66.7748 |
| -5.41042 | - |
| **Test-4** | 0.745 | 0.4 | 0.926286 | 24.33369 |
| 8.24 | 10 | 7.97035 | 3.27245 |
| -0.08 | -0.2 | 0.316725 | - |
| 9.4 | 10 | 9.78945 | 4.143085 |
| 1 | 1.5 | -2.47869 | - |
| -4 | -2.8 | -4.89161 | 22.29025 |
| **Test-5**  Same as Test-4 (with different ‘alpha=3e-4’) | | | -nan(ind) Because the computations are sensitive to ‘alpha’ | - |
| **Test-6**  Same as Test-4 (with different ‘alpha=8e-8) | | | 0.65629 | 11.9074 |
| 8.4886 | 3.01699 |
| -0.430567 | - |
| 9.90604 | 5.383404 |
| 0.993401 | 0.6599 |
| -3.73515 | 6.62125 |

**Table 7.** Parameter Comparison for Gradient Descent Tests

Conclusion: Test 1,2,3 & 4 were done with step size 3e-6. Reducing alpha to 8e-8, makes the convergence very slow and it takes 1000000 iterations, and still doesn’t converge well.

Hence, by testing the following algorithms, it can be concluded that Gradient Descent method is slow, step-size sensitive (which is also data sensitive) and provide a poor fit for complex functions. On the other hand, Levenberg-Marquardt & Gauss-Newton Algorithms provide a better and faster convergence in almost all the cases.

**4.6 Parameter Identification Program (Final Coupled Code)**

Now, since the functionality of all the routines is verified, it is now justified to test the final parameter identification program which has the 3 algorithms coupled with the 2 FEM programs.

4.6.1 Tests for the verification of Parameter Identification for various Material Parameters

The files for the test cases can be found in the directory *Test\_Material\_Properties > Test-%.* The list of files is as follows:

* displacement\_2\_bar.csv, displacement\_tapered\_10e.csv: Data used as input for parameter identification
* LMA\_fit\_2ele.csv, LMA\_fit\_t.csv: Contains the fitted data points computed by the identifies parameters by Levenberg-Marquardt algorithm for both geometries
* GN\_fit\_2ele.csv, GN\_fit\_t.csv: Contains the fitted data points computed by the identifies parameters by Gauss Newton algorithm for both geometries
* GD\_fit\_2ele.csv, GD\_fit\_t.csv: Contains the fitted data points computed by the identifies parameters by Gradient Descent algorithm for both geometries
* InputFile\_NLR\_2Bar.txt, InputFile\_NLR\_TAPERED.txt: Input Logs
* OutputLog.txt: The output log, i.e., obtained parameters and final % error between identified and actual parameters.
* InputFileFEA2Bar.txt, InputFileFEATaperedBar.txt: Input log files which were used to create data, i.e., perform the FEA computation. This will help to reproduce the test again.
* Plots indicating curve fits with measured and fitted curves for results obtained from all algorithms and geometries, as *Mat%%.pdf*.

The results of the test are as follows:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Geometry** |  | **Actual Parameters** | | | **Identified Parameters** | | | **Avg. Error** |
| **Method** | **Young's Mod.** | **Yield Stress** | **Viscosity** | **Young's Mod.** | **Yield Stress** | **Viscosity** |
|  | MPa | MPa | 1/s | MPa | MPa | 1/s | % |
| **1** | 2 Bar | LMA | 200000 | 400 | 1 | 199935 | 399.916 | 0.999897 | 0.021302 |
| GN | 199935 | 399.916 | 0.999897 | 0.021302 |
| GD | 194004 | 403.025 | 1.26805 | 10.1864 |
| Tapered | LMA | 100000 | 160 | 0.1 | 89957.9 | 159.608 | 0.0991208 | 4.5239 |
| GN | 83004.7 | 159.704 | 0.0992572 | 1.56124 |
| GD | 81000 | 162.001 | 0.105839 | 2.78011 |
| **2** | 2 Bar | LMA | 100000 | 200 | 5 | 99963.7 | 199.951 | 4.99882 | 0.028206 |
| GN | 99963.8 | 199.951 | 4.99882 | 0.028203 |
| GD | 97007.2 | 197.333 | 4.32284 | 5.9565 |
| Tapered | LMA | 100000 | 200 | 5 | 95964.1 | 200.184 | 5.02145 | 1.51895 |
| GN | 99955.5 | 199.944 | 4.99807 | 0.037019 |
| GD | 97000 | 197.001 | 5.39696 | 4.14629 |
| **3** | 2 Bar | LMA | 85000 | 150 | 40 | 89394.1 | 149.806 | 39.633 | 2.07219 |
| GN | 84968.5 | 149.962 | 39.9902 | 0.028902 |
| GD | 87974.7 | 151.156 | 42.5508 | 3.54907 |
| Tapered | LMA | 85000 | 150 | 40 | 158416 | 210.808 | 4.32904 | 72.0294 |
| GN | 35491.4 | 146.926 | -1.31363 | 54.5263 |
| GD | 88044.2 | 155.539 | 39.9779 | 2.44298 |
| **4** | 2 Bar | LMA | 210000 | 400 | 0.2 | 209921 | 399.897 | 0.199949 | 0.02963 |
| GN | 209921 | 399.897 | 0.199949 | 0.029631 |
| GD | 1.05E+216 | 1.10E+213 | 2.04E+213 | - |
| Tapered | LMA | 210000 | 400 | 0.2 | 21158.3 | 403 | 0.18 | 33.5582 |
| GN | -9.26E+11 | 403 | 0.18 | - |
| GD | 208500 | 398.9 | 0.185 | 2.82976 |
| **5** | 2 Bar | LMA | 120000 | 240 | 10 | 125078 | 239.313 | 9.82794 | 2.07966 |
| GN | 119949 | 239.928 | 9.99699 | 0.034301 |
| GD | 128000 | 235.996 | 9.06337 | 5.90042 |
| Tapered | LMA | 120000 | 240 | 10 | 122028 | 239.674 | 9.94095 | 0.805437 |
| GN | 119924 | 239.923 | 9.99643 | 0.043752 |
| GD | 128000 | 236 | 9.00006 | 6.11093 |

**Table 8.** Parameter Identification Test Results

For the Test 4, the results were re-produced using different value of the damping factor (lambda=1000) in LMA & step-size in GD to observe the effect. The results then obtained are as follows:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **4-REPEATED** | 2 Bar | LMA | 210000 | 400 | 0.2 | 206373 | 399.847 | 0.199949 | 0.820233 |
| GN | 209921 | 399.897 | 0.199949 | 0.029631 |
| GD | 207000 | 394 | 0.174303 | 5.25903 |
| Tapered | LMA | 210000 | 400 | 0.2 | 209902 | 399.892 | 0.199931 | 0.0361688 |
| GN | 209902 | 399.892 | 0.199931 | 0.0361736 |
| GD | 207000 | 394 | 0.174655 | 5.20039 |

**Table 9.** Repeated Test-4 Results

Similarly, to improve the LMA results in Test 5, Lambda was increased to 10000, and the program was run again, but no significant change was observed.

Example of the curve fit is:

Graphical user interface, diagram

Description automatically generated

**Fig.29** Curve fit for Parameter Identification Test-4

Graphical user interface, diagram, application

Description automatically generated

**Fig.30** Curve fit for Parameter Identification Improved Test-4

Conclusion: The tapered bar computation is highly nonlinear because of continuous changing stress state for each element. Whereas 2 bar geometry is comparatively straight forward for computation.

* Thus, for most cases, Levenberg-Marquardt algorithm & Gauss-Newton gives better fit compared to Gradient Descent.
* For 2 bar computations, Gauss-Newton is sometime faster than Levenberg-Marquardt algorithm.
* Gradient Descent shows very slow convergence, and in some cases, no convergence at all. It converges only if the initial guess is near to the actual values.
* The algorithms overall give better results for 2 bar experiment. Thus, for parameter identification of materials, Simpler experiments result in easier convergence and less error for parameter identification procedures.

4.6.2 Verification by Noisy Data Computation

The errors during the measurement of data during an experiment play a vital role in causing error during the parameter identification process. Thus, to check the functionality of the parameter identification program for such input data with errors, this test is performed.

The *NoiseAddition* directory has the C++ code for adding noise to the data (by generating random numbers between 0 & 1). By copying the displacement data file to this directory and running *addNoise.cpp*, noise of the specified order will be added to the data, and an output ‘.csv’ file will be created.

The order of the noise to be added can be specified in the code. This noisy data can then be copied to the parameter identification project directory and used as input file, mimicking the measurement errors during an experiment.

The test directory *Test\_NoiseAddition>Noise\_oder\_1e-%* contains all the files produced by the program including plots, and log files for reproduction of the tests.

The average errors found for different order of noises in the data is as follows:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Noise Order | Geometry | Method | Avg. Error |  | Noise Order | Geometry | Method | Avg. Error |
|  |  |
| % |  | % |  |
| 1.00E-05 | 2 Bar | LMA | 0.0341978 |  | 1.00E-03 | 2 Bar | LMA | 0.051905 |  |
| GN | 0.0341978 |  | GN | 0.0519039 |  |
| GD | 5.54734 |  | GD | 5.54734 |  |
| Tapered | LMA | 2.96949 |  | Tapered | LMA | 2.9757 |  |
| GN | 1.54775 |  | GN | 1.23767 |  |
| GD | - |  | GD | - |  |
| 1.00E-04 | 2 Bar | LMA | 0.051905 |  | 1.00E-02 | 2 Bar | LMA | 0.051905 |  |
| GN | 0.0519039 |  | GN | 0.0519039 |  |
| GD | 5.54734 |  | GD | 5.54734 |  |
| Tapered | LMA | 2.96949 |  | Tapered | LMA | 3.02642 |  |
| GN | 1.54775 |  | GN | 2.055 |  |
| GD | - |  | GD | - |  |

**Table 10.** Parameter Identification withNoisy Input (Experimental) Data

Conclusion: With the noisy data, the error slightly increases in LMA & GN Methods. The error is comparatively high in GD results. Whereas unrealistic error is observed in the Gradient Descent methods in tapered bar noisy experiment even with error of very low order. This may be rectified by modifying the step size.

4.6.3 Computation Time Evaluation & Bottleneck

One of the most important deciding factors of quality of code is its computational time. The parameter identification program has various functions iterating several times. This results in a average computational time of 20 minutes for the whole computation to finish. Three cases have been provided in the directory *ComputationTime > Case-%* . During the program runtime, the following files for computation times are written:

* ComputationTimeFEA\_2Bar.txt: Evaluated time for one FEA 2 bar simulation
* ComputationTimeFEA\_Tapered.txt: Evaluated time for one FEA tapered bar simulation
* ComputationTimeInverse.txt: Evaluated time for inverse operation of Hessian matrix
* ComputationTimeJacobian2Bar.txt: Evaluated time for Jacobian matrix computation for 2 bar FEA function
* ComputationTimeJacobianTapered.txt: Evaluated time for Jacobian matrix computation for tapered bar FEA function
* ComputationTimePsuedoInverse.txt: Evaluated time for pseudo inverse operation of Hessian in Gauss Newton mehtod
* ComputationTimeOverall.txt: Evaluated overall computation time & for individual parameter identification methods

The average computation times are as follows:

|  |  |
| --- | --- |
| Function/Process | Avg. Time (seconds) |
| FEA 2 Bar | 0.58 |
| FEA Tapered Bar | 2.7 |
| Inverse | 0.0003 |
| Pseudo Inverse | 0.0003 |
| Jacobian 2 Bar | 2.15 |
| Jacobian Tapered Bar | 10.6 |
| Levenberg-Marquardt Algorithm (2 Bar) | 25 |
| Gauss-Newton Algorithm (2 Bar) | 18 |
| Gradient Descent Method (2 Bar) | 120 |
| Levenberg-Marquardt Algorithm (Tapered Bar) | 85 |
| Gauss-Newton Algorithm (Tapered Bar) | 300 |
| Gradient Descent Method (Tapered Bar) | 755 |
| Overall Program | 1080 |

**Table 11.** Computational Time Evaluation

* The tapered bar FEA takes more time than 2 bar, because of multiple elements are considered.
* The inverse & pseudo inverse functions take similar time (Eigen library functions)
* Within the functions, Jacobian matrix is the one which is called multiple times in iterations. Also, the Jacobian matrix function calls the FEA function multiple times as it must find the change in output w.r.t change in parameters. Thus, this makes it computationally expensive, especially for the complex geometries and geometries with higher number of elements.
* Gradient Descent method always shows slow convergence compared to the other methods.

Conclusion: The major ‘Bottleneck’ observed is the Jacobian matrix computation, which adopts the Finite Difference Method. For the straight-forward differentiable functions, it is thus recommended to use the partial derivative equations to compute the Jacobian.

**Chapter 5**

**Further Summarized Conclusions**

* No one method can solve every problem. Behaviour of all three algorithms were observed to be different for various cases.
* In some well-behaved curves/functions, Gauss Newton method gave a faster convergence compared to Levenberg-Marquardt Algorithm, even with an ill-posed problem (which was solved by using pseudo inverse) .
* Gradient Descent method showed poor convergence almost in all the cases. Thus, it can be suggested to use a non-linear regression method instead of steepest-descent method for iterative base functions.
* During parameter identification of a new material, for which no initial parameters are known, it is advised to use a simpler geometry if it is to be coupled with a finite element simulation. This makes it easier for the parameter identification algorithm to reduce errors in computations. Like it was observed above, the computational effort/error was higher for the tapered bar experiment parameter identification compared to the 2 bar geometry.
* The non-linear regression methods work well even if the data has measurement errors of low orders.

**Chapter 6**

**Manual**

Required environment & libraries to run the programs are discussed in the Section 3.6.

To run the parameter identification program, it is first required to generate the data using FEA programs.

* In the *FEA\_2\_ELE\_PROBLEM* directory, open the UserInputFile2BarFEA.csv. Modify the file according to the required inputs. Run the *main.cpp* from IDE or terminal. The file of interest amongst the files generated is displacement\_2\_bar.csv.
* In the *FEA\_TAPERED\_PROBLEM* directory, open the UserInputFileTaperedBarFEA.csv. Modify the file according to the required inputs. Run the *main.cpp* from IDE or terminal. The file of interest amongst the files generated is displacement\_tapered\_10e.csv.

Copy these two files to the *PersonalProgrammingProject66157* directory, which has the parameter identification program.

Modify the user defined input files UserInputFile\_PARA\_2BarFEA.csv & UserInputFile\_PARA\_TaperedBarFEA.csv. The experimental parameters in these files should be the same which were used to generate the data. Run the *Source.cpp* from IDE or terminal. The program will then read the user defined input files, perform computation, generate files, and display the plot (stress vs. strain & displacement vs. time steps) in the end. To save the plot, from the GNUPLOT Window, click on ‘Export as PDF’ and save it to desired location.

To run the tests for the individual algorithms, open the *Levenberg\_Marquardt\_Polynomial*, *Gauss\_Newton\_Algorithm* & *Gradient\_Descent\_Algorithm* directories, use python to generate data from *DataGenerator.py* by modifying the equation in the program. After generating, modify the input data in *main.cpp* in the mentioned directories, and run the program.

To change the parameters which are not mentioned in user input files, for e.g., the number of time steps, these can be changed from the *main.cpp* in FEA programs, and *fea\_2\_ele\_main.cpp/fea\_tapered\_main.cpp* in the parameter identification program.

NOTE: The files to be read by C++ program must be in the same working directory as the program. If input from a different file is to be given, its path must be specified in the code wherever required.

**References**

[1] *The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems*, Henri P. Gavin, Duke University, 2020

[2] *A Memo on How to use LMA for Refining Camera Calibration Parameters*, Pradit Mittrapiyanuruk, Robot Vision Laboratory, USA

[3] *Modelling of elastic-viscoplastic behaviour and its finite element implementation*, Ted Diehl, New York, 1988

[4] *Lecture Notes-Nonlinear Finite Element Method*, Dr. Geralf Hütter, TUBAF, 2021

[5] *Lecture Notes-Plasticity,* Prof. Björn Kiefer, TUBAF, 2022

[6] *Lecture Notes-Parameter Identification in Nonlinear Solid Mechanics*, Dr. Martin Abendroth, TUBAF, 2022

[7] *Documentation*: C++, Eigen3, rapidcsv, gnuplot-iostream, gnuplot, Microsoft vcpkg