Due date: Monday, April 21, at the beginning of the lab.

1 Objectives of this Assignment

Our plans for this assignment are

- To implement the deflation process for nonlinear equations;
- To move on to the n-dimensional form of the Newton-Raphson and secant algorithms;
- To apply these techniques to a simple practical problem.

For this assignment, I impose again specifications for the Java/C++ implementation. When grading your labs, the TA will execute a program that we have written.

2 Improving Last Week’s Algorithms

2.1 Bracketing

Three of the algorithms you implemented last week have a common problem: The iterated solution does not stay inside the bracket you specified initially, and instead venture wherever the shape of the function’s curve leads them\(^1\). What we would like to do now is modify these algorithms so that once we have specified an initial bracket that contains a solution, the iteration stays inside the bracket.

This is really easy to do: If the \(x_{k+1}\) that you just computed is outside of the bracket, replace that value for \(x_{k+1}\) by the one given by the bisection algorithm and update the bracket.

What to do, Part I: Modify your fixed point, Newton-Raphson, and secant procedure so that

- They all take as parameters two bracketing values, \(a\) and \(b\), a value for the tolerance, and a max number of iteration steps.
- Your iteration stays inside the bracket.

\(^1\)No extra points for guessing which are the three methods I am talking of here.
2.2 Deflation

Certainly it will not come as a major surprise to you that there is a deflation process for NLEs as well. Basically, the idea behind deflation for NLEs is as follows.

Let \( f \) be a function defined over the interval \([a, b]\) and that admits multiple zeros over that interval, \( x_1^*, x_2^*, \ldots, x_n^* \). We can in theory find all these zeros by applying the following algorithm:

```
Initialize \( f_1 = f \)

for \( i \) from 1 to \( n \) and while not finished do
    find \( x_i^* \), a solution of \( f_i(x) = 0 \) on \([a, b]\)
    define \( f_{i+1} : x \mapsto \frac{f_i(x)}{x - x_i^*} \)
end do
```

To see how this works, let us consider the following function:

\[
f : \mathbb{R} \rightarrow \mathbb{R} \\
x \mapsto x^2 e^{-x} - \frac{1}{2}.
\]

On the interval \([-1, 3]\), the graph of this function looks as shown in Figure 1.

![Figure 1: Graph of function \( f \) over the interval \([-1, 3]\).](image)

The zeros of \( f \) over \([-1, 3]\) are \(-0.539835, 1.48796,\) and \(2.61787\). Let’s say that the first zero we find is \( x_1^* = 1.48796 \). We would now define the function

\[
f_2 : \mathbb{R} \rightarrow \mathbb{R} \\
x \mapsto \frac{x^2 e^{-x} - \frac{1}{2}}{x - x_1^*} = \frac{x^2 e^{-x} - \frac{1}{2}}{x - 1.48796}.
\]
Figure 2 shows on the left the graph of $f_2$ over $[-1, 3]$ and on the right that graph superimposed on that of $f_1 = f$. We can observe that $x_1^*$ is not a zero of $f_2$, but that the other two zeros of $f$ are still there.

![Graph 2](image)

Figure 2: Left: Graph of function $f_2$ over the interval $[-1, 3]$; Right: Graphs of $f_1$ and $f_2$ over that interval.

Of course, we will not stop here. We run our trusted nonlinear equation solver once more, and this time we find $x_2^* = 2.61787$. We can now define

$$f_3 : \mathbb{R} \rightarrow \mathbb{R}$$

$$x \mapsto \frac{x^2 e^{-x} - \frac{1}{2}}{(x - x_1^*)(x - x_2^*)} = \frac{x^2 e^{-x} - \frac{1}{2}}{(x - 1.48796)(x - 2.61787)}.$$

Figure 3 shows on the left the graph of $f_3$ over $[-1, 3]$ and on the right that graph superimposed on that of $f_1 = f$ and $f_2$. This time, only one of the original three zeros is left.

![Graph 3](image)

Figure 3: Left: Graph of function $f_3$ over the interval $[-1, 3]$; Right: Graphs of $f_1$, $f_2$, and $f_3$ over that interval.

One last time, we seek a zero of a nonlinear equation, this time, for $f_3(x) = 0$. We find, naturally, the third zero of the original equation, $x_3^* = -0.539835$. If we were to try applying the deflation
process once more to the function, we would define

\[ f_4 : \mathbb{R} \rightarrow \mathbb{R} \]

\[ x \quad \text{by} \quad x^2 e^{-x} - \frac{1}{2} = \frac{x^2 e^{-x} - \frac{1}{2}}{(x - x_1^*)(x - x_2^*)(x - x_3^*)} = \frac{x^2 e^{-x} - \frac{1}{2}}{(x - 1.48796)(x - 2.61787)(x + 0.539835)} \]

The graph of this function is shown on the left side of Figure 4, while the right side shows \( f \) and all its deflations. We can verify that all the zeros of the function over the interval \([-1, 3]\) have been found.

Figure 4: Left: Graph of function \( f_4 \) over the interval \([-1, 3]\); Right: Graphs of \( f_1, f_2, f_3, \) and \( f_4 \) over that interval.

**What to do, Part II:** Implement the deflation algorithm and test it with the following function over the interval \([0, 10]\):

\[ g : \mathbb{R} \rightarrow \mathbb{R} \]

\[ x \quad \text{by} \quad 4 \left( (x + 1) \sqrt{x} - x \sqrt{x + 4} \cos \left( \sqrt{x} \ln x - \sin x \right) \right) - 1. \]

**What to do, Part III:** You should have enough experience in numerical computations by now to see that the deflation algorithm is not without flaws. List and discuss the problems you see with this algorithm, and if possible illustrate them with an example.

### 3 Systems of Nonlinear Equations (SNLEs)

#### 3.1 Formulation of an SNLE

As we saw in class, an SNLE is simply a set of \( n \) nonlinear equations in terms of \( n \) unknowns. We are trying to find \( (x_1^*, x_2^*, \ldots x_n^*) \), such that

\[
\begin{align*}
  f_1(x_1^*, x_2^*, \ldots x_n^*) &= 0, \\
  f_2(x_1^*, x_2^*, \ldots x_n^*) &= 0, \\
  &\vdots \\
  f_n(x_1^*, x_2^*, \ldots x_n^*) &= 0.
\end{align*}
\]
We switch to a more compact vector notation by defining

\[ X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad \text{and} \quad F(X) = \begin{pmatrix} f_1(X) \\ f_2(X) \\ \vdots \\ f_n(X) \end{pmatrix} = \begin{pmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) \end{pmatrix}. \]

The Jacobian matrix of \( F \) at \( X \) is

\[
JF(X) = \begin{pmatrix}
\frac{\partial f_1}{\partial x_1}(X) & \frac{\partial f_1}{\partial x_2}(X) & \cdots & \frac{\partial f_1}{\partial x_n}(X) \\
\frac{\partial f_2}{\partial x_1}(X) & \frac{\partial f_2}{\partial x_2}(X) & \cdots & \frac{\partial f_2}{\partial x_n}(X) \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial f_n}{\partial x_1}(X) & \frac{\partial f_n}{\partial x_2}(X) & \cdots & \frac{\partial f_n}{\partial x_n}(X)
\end{pmatrix}.
\]

If \( X^* = (x_1^*, x_2^*, \ldots, x_n^*) \) is the solution we are seeking and \( X_k \) is our estimate of \( X^* \) at step \( k \) of our iteration, with \( X_k = X^* + s_k \), then

\[
F(X^*) = F(X_k - s_k) = F(X_k) - JF(X_k) s_k.
\]

Since \( F(X^*) = 0 \) (the \( n \)-dimensional null vector), the corrective term \( s_k \) is the solution of the SLE

\[ JF(X_k) s = F(X_k). \]

### 3.2 The \( n \)-dimensional Newton-Raphson algorithm

Very little has changed from the 1-D version:

Initialize \( X_0 \)

while we have not converged do

solve the SLE \( JF(X_k) s_k = F(X_k) \) for \( s_k \)

\[ X_{k+1} = X_k + s_k \]

\[ k++ \]

end do

**What to do, Part IV** Implement the Newton-Raphson algorithm and test it on a couple of examples of SNLEs. For one of your examples, you could for example select the equations of two intersecting circles. For the other example, take something a bit more complicated. Note that since the notion of bracket does not apply in \( n \)-D, your procedure will take an initial estimate \( X_0 \) as parameter (besides, of course, the function, the max number of iterations, and the tolerance parameters, as in the 1-D case).
3.3 The $n$-dimensional secant algorithm

What can we do when the derivatives of $F$ are not available? Just as we computed an approximate value of the derivative of our function $f$ at $x_k$ in the 1-dimensional case, we need to compute an approximate value of $JF$ at $X_k$. How can this be done?

One possibility would be to make a small displacement along each dimension of the $X$ space, and use the variation of $F$ along these axes as estimate of our Jacobian matrix. If we stick to the naive notion that the function is always something that is defined by a simple equation, but this is not always the case. The “function” might be a physical process for which we collect a measurement, or it could be a computational module that has to run for several minutes/hours to return the value taken at a given $x$. In this case, the strategy “let’s move along all axes” is simply not feasible.

A better idea would to compute at each step a corrective term for the current estimate of the Jacobian matrix. So, if the estimate of $JF(X_k)$ at step $k$ was $B_k$, we want to compute a corrective term $\Delta B_k$ so that the estimate for $JF(X_{k+1})$ at step $k+1$ is $B_{k+1} = B_k + \Delta B_k$. Now, how can we compute this corrective term $\Delta B_k$? Let us first look at a first version of the algorithm:

**Incomplete secant algorithm**

Initialize $X_0$

Initialize $B_0$

while we have not converged do

solve the SLE $JF(X_k)s_k = F(X_k)$ for $s_k$

$X_{k+1} = X_k + s_k$

somehow compute $B_{k+1} = B_k + \Delta B_k$

$k++$

end do

Once we have computed our new solution estimate, $X_{k+1}$, we are going to compute the value taken by $F$ at that point, $F(X_{k+1})$, to see how good our estimate was. The difference between the value taken by $F$ at $X_{k+1}$ and at $X_k$ can be written as

$$F(X_{k+1}) - F(X_k) \approx JF(X_{k+1}) (X_{k+1} - X_k).$$

Of course, $X_{k+1} - X_k$ is the term $s_k$ that we just computed. Replacing $JF(X_{k+1})$ by $B_{k+1}$, that is, $B_k + \Delta B_k$, we get

$$(B_k + \Delta B_k) s_k = F(X_{k+1}) - F(X_k)$$

$$\implies \quad \Delta B_k s_k = F(X_{k+1}) - F(X_k) - B_k s_k.$$ 

All the terms on the right side are known, so what we have to do is compute a $\Delta B_k$ that satisfies the equation. As you can imagine, this is not a problem with a unique solution. In fact, there are an infinity of matrices $\Delta B_k$ that would satisfy this equation? So which one do we select?
The classical answer is to select the smallest possible solution, in the sense of the $\| \|_2$ norm. This solution is simply

$$\Delta \mathbf{B}_k = \frac{1}{s_k^T s_k} (\mathbf{F}(\mathbf{X}_{k+1}) - \mathbf{F}(\mathbf{X}_k) - \mathbf{B}_k s_k) s_k^T.$$ 

So the final version of the secant algorithm is:

**Secant algorithm**

- Initialize $\mathbf{X}_0$
- Initialize $\mathbf{B}_0$

while we have not converged do

- solve the SLE $\mathbf{J}\mathbf{F}(\mathbf{X}_k) s_k = \mathbf{F}(\mathbf{X}_k)$ for $s_k$
- $\mathbf{X}_{k+1} = \mathbf{X}_k + s_k$
- $\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{1}{s_k^T s_k} (\mathbf{F}(\mathbf{X}_{k+1}) - \mathbf{F}(\mathbf{X}_k) - \mathbf{B}_k s_k) s_k^T$

end do

**What to do, Part V** Implement the secant algorithm and test it on the examples of SNLEs you used to test the Newton-Raphson algorithm.

4 A Practical Application: Image distortion

The pictures we take with cameras all suffer to some extent from distortion. One cause of distortion is the lens used on the camera. Even for a very good lens we will observe some degree of “pincushion” distortion. This distortion is show in Figure 5, on the left for a single point, and on the right for a square.

![Figure 5: Left: Model of the pincushion distortion; Right: Effect of the distortion on a square.](image)

The picture on the left shows the center of the image, $O$, and “undistorted” point $M_u$ (that is, where the point would appear if there was no distortion), and the “distorted” point $M_d$ (where the point
really appears in the image, due to distortion. For obvious reasons, this type of distortion is often called radial distortion. The relationship between $M_d$ and $M_u$ is fairly simple (to write):

$$\overrightarrow{OM_u} = \left(1 + \kappa_1 \|\overrightarrow{OM_d}\|_2^2\right) \overrightarrow{OM_d},$$

where $\kappa_1$ is a constant, called the first-order radial distortion coefficient. The presence of the term “first-order” indicate that we are dealing with a simplified model.

What to do, Part VI Write a Maple procedure that takes as parameters the value of the distortion coefficient and the coordinates of an “undistorted” point and returns the coordinates of the corresponding “distorted” point. Obviously, you should use here one of your procedures for solving nonlinear equations.

Then, to visualize the results, select a set of points (forming a grid, for example) and show the “before” and after” images for this set of points.

5 C++/Java Implementation

5.1 Slight modification of the NonlinearSolver class

These modifications aim at integrating the improvements we just brought to three of the algorithms. Now all your zero-seeking methods will require a bracket, and will restrict the solution to that bracket. Here are the slightly modified interfaces:

The nonlinear solver class This class is named NonlinearSolver and it is not abstract. If you program in C++, then you should provide a header file named NonlinearSolver.h. This class offers the following public methods/functions:

- `NonlinearSolver(FunctionClass myFunc)`
  this constructor takes as parameter the FunctionClass object on which to operate.

- `double solveBisection(double a, double b, double tol)`
  returns the result of a search by bisection on the interval $[a, b]$, with tolerance $tol$. If no solution exists, your method should throw an exception.

- `double solveFixedPoint(double a, double b, int maxNbIter, double tol)`
  returns the result of a search by application of the fixed point algorithm on the interval $[a, b]$, with $maxNbIter$ as maximum number of iterations, and tolerance $tol$. If no solution exists, your method should throw an exception.

- `double solveNewton(double a, double b, int maxNbIter, double tol)`
  returns the result of a search by application of the Newton-Raphson algorithm on the interval $[a, b]$, with $maxNbIter$ as maximum number of iterations, and tolerance $tol$. If no solution exists, your method should throw an exception.
double solveSecant(double a, double b, int maxNbIter, double tol)
returns the result of a search by application of the secant algorithm on the interval [a, b],
with maxNbIter as maximum number of iterations, and tolerance tol. If no solution
exists, your method should throw an exception.

5.2 \textit{n}-dimensional functions
This is a straightforward adaptation of the 1D class.

- The parent class is an abstract class named \textit{NDFunctionClass}. If you program in C++,
you should provide a header file named \textit{NDFunctionClass.h}.

- \textit{NDFunctionClass}'s derived classes should offer the following methods:
  - public NDFunctionClass(double xmin[], double xmax[])
    a constructor taking as parameters the endpoints of the interval along all axes, for the domain on
    which the function is defined. In C++ you will need to pass along as well the number
    of variables, \( n \) (as an int).
  - public FunctionClass(int n)
    a constructor for a function defined over \( \mathbb{R}^n \).
  - public double func[](double x[])
    returns the value of the function at \( x \).
    If \( x \) is not in the valid range for the function, an exception should be thrown.
  - public double jFunc[][](double x[])
    returns the value of the function’s
    Jacobian matrix at \( x \) when this derivative is defined. If \( x \) is not in the valid range for
    the function, or if the derivative is not defined, an exception should be thrown.
  - public boolean isJacobianDefined()
    returns true if the function’s Jacobian matrix is defined, false otherwise.

5.3 The \textit{n}-dimensional system solver class
This class is named \textit{NDNonlinearSolver} and it is \textit{not} abstract. If you program in C++, then
you should provide a header file named \textit{NDNonlinearSolver.h}. This class offers the following
public methods/functions:

- NDNonlinearSolver(NDFunctionClass myFunc)
  this constructor takes as parameter the \textit{NDFunctionClass} object on which to operate.

- double[] solveNewton(double x0[], int maxNbIter, double tol)
  returns the result of a search by application of the Newton-Raphson algorithm with \( x_0 \) as
  starting point, \( \text{maxNbIter} \) as maximum number of iterations, and tolerance \( tol \). If no
  solution exists, your method should throw an exception.
• double[] solveSecant(double x0[], int maxNbIter, double tol) returns the result of a search by application of the secant algorithm with x0 as starting point, maxNbIter as maximum number of iterations, and tolerance tol. If no solution exists, your method should throw an exception.

6 Evaluation

6.1 What to hand in

A: You should upload to the EnVision server a folder containing the following:

1. A complete, cleaned-up CodeWarrior project folder.
   • Complete means that everything needed to compile and execute the project is there: source files, project file, header files (.h files) and precompiled header source files (.pch files) if you program in C++, test data files.
   • Cleaned-up means that files resulting from the compilation should be removed: .exe files, .sym files, precompiled header binary files (if in C++), and the entire <name of project>_data folder (close the project before you throw away this folder).

2. Maple worksheets if you did any prototyping using Maple (either remove the results from the worksheet or export the worksheet as text first, please).


B: You should hand in hard copies of your report and of your Maple/muPad worksheet.

6.2 Point distribution

The maximum number of points is 100, but extra points could be awarded for excellent aspects of the project or report. The point distribution for this assignment is as follows:

<table>
<thead>
<tr>
<th>Maple/muPad Modelling</th>
<th>25 pts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accomplishes what was demanded</td>
<td></td>
</tr>
<tr>
<td>Comments and analysis</td>
<td>10 pts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C++/Java Code</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accomplishes what was demanded</td>
<td>15 pts</td>
</tr>
<tr>
<td>Good class design</td>
<td>10 pts</td>
</tr>
<tr>
<td>General quality &amp; readability</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Report</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Discussion and analysis of the results</td>
<td>20 pts</td>
</tr>
<tr>
<td>General quality of the writing and presentation</td>
<td>10 pts</td>
</tr>
</tbody>
</table>
6.3 Various point penalties

Hopefully we won’t have to apply many of these:

- Project folder incomplete or not properly cleaned up -5 pts
- Report file missing from the project folder -5 pts
- Maple file missing 0 for that part

Late penalties
- Printed copy of the report, 1 day late -5 pts
- Project folder (uploaded to EnVision server), per day late -5%

If you submit a project late, then it is your responsibility to notify the TA (with CC. to me) that the project is finally available for download on the EnVision server. If you fail to do so, then the “late penalty clock” will keep ticking until the TA gets around to checking your folder on the EnVision server and notices your project. Unless asked explicitly to do so, do not mail your project folder as an attachment.