Below is a list of problems for our lab session

Wednesday, November 19th (kl. 13.00-16.00), Room MA436-446.

**Remark 1** This version was updated after the Lab with a correction to Remark 2 which contained a error in the MATLAB code. Moreover, based on the extensive discussions I had with different groups of students, I added a lot of explanation to Problem 3.
**Problem 1** The class website contain an implementation of Newton’s method called *newton.m*. The goal of this problem is to solve the non-linear equation

\[ f(x) = 0. \]  

where \( f(x) = x - \tan(x) \). We begin by verifying that there are in fact solutions to this equation.

1. Plot the function \( \tan \) in MATLAB and explain why equation (1) has exactly one solution \( \xi_j \) in each of the intervals \( I_j \) given by

\[ I_j = \left( -\frac{\pi}{2}, \frac{\pi}{2} \right) + j\pi, \quad j \in \mathbb{Z} \]

If your explanation does not include the words “continuity” and “asymptote”, then you are skipping some key steps and you would loose points during the final exam.

2. Compute \( \xi_j \) using Newton’s method for \( j = 1, 2, 3 \). You will need a good guess for \( \xi_1 \), but you can probably use

\[ \xi_{j+1} \approx \xi_j + \pi \]

to initialize the search for \( \xi_j \) for \( j \geq 2 \). You will find it easy to show, that

\[ \tan'(x) = \frac{d}{dx} \left( \frac{\sin(x)}{\cos(x)} \right) = 1 + \tan^2(x) = \frac{1}{\cos^2(x)}. \]  

(2)

It is always nice to obtaining approximations, but they are useless on their own without a reliable error estimate!

3. In general, let \( x \) be a very good approximation of the root \( r = x + \Delta x \). Show that the relative error satisfies

\[ \frac{\Delta x}{x} \approx -\frac{f(x)}{x f'(x)} \]  

(3)

4. Compute error estimates for each of the approximations that you obtained in question 2.

**Remark 2** There is a subtle trap embedded in the above problem. Exactly how good is the approximation given by (3)? There is a way out, but it involves more work on our part. See the next problem for an explanation on how to do this!
Problem 2 The bisection function has been overhauled based on questions received during Lab session 1 and 2. The new function is called bisection2 and it will eventually replace bisection. The bisection algorithm is used to solve nonlinear equations of the form given by equation (1) where $f : I \to \mathbb{R}$ and $I \subseteq \mathbb{R}$ is an interval.

1. Read though the documentation of bisection2 and take note of the changes.

2. Run the minimal working example of bisection2 which solves the equation (1) where

$$f(x) = x^2 - 2$$

and verify manually that the arrays $a$ and $b$ satisfy

$$a_i < r < b_i, \text{ and } b_{i+1} - a_{i+1} = \frac{b_i - a_i}{2}$$

where $r = \sqrt{2}$ is the only positive root.

3. In general, let $c_i = \frac{a_i + b_i}{2}$ be the $i$th estimate for the root $r \neq 0$. Show that the relative error satisfies

$$\left| \frac{r - c_i}{r} \right| \leq \frac{|b_i - a_i|/2}{\min\{|a_i|, |b_i|\}}$$

In particular, explain why $\min\{|a_i|, |b_i|\} \leq |r|$.

You will now adjust the input to the minimal working example of bisection2 to force the different values of flag to occur.

4. Pick values of bad values $a_1$ and $b_1$ so that flag = -1 and it=1 is returned.

5. Pick values of values of $\delta$, $\epsilon$, and $\maxit$, so that flag=0 is returned with it=50.

6. Pick values of $\delta$, $\epsilon$ which are so large that flag=1 is returned with it=20.

Remark 3 During Lab session 2 you applied the bisection algorithm to computing a firing solution for an artillery gun. There was a problem, when we picked a new target and forgot to redefine the residual function. A much cleaner solution is given here. Define a gun using the minimal working example of range_rkx. Let $d$ denote distance to the target and let the range function and the residual function be defined as follows:

```matlab
>> r=@(theta)range_rkx(v0,theta,method,dt,maxstep);
>> f=@(theta,d)r(theta)-d
```

Then a firing solution can be computed as follows:

```matlab
>> d=12345;
>> [c, flag, it, a, b]=bisection2(@(theta)f(theta,d),0,6.2622e-1,50,0,30);
```

Simply verify what the procedure works, when you change the value of $d$ at will. It is worth noticing how this construction feeds the bisection function a real function of a single real variable. The angle 0.626622 rad is not picked at random, but is very close to the elevation which gives the longest range for the gun defined by the minimal working example of range_rkx.

---

1. During Lab session 3 a typo was discovered here as I had accidentally copied $f=@(theta,d)r(theta,d)$ into the text.
Problem 3 Floating point arithmetic imposes limitations on how accurately we can define and solve nonlinear equations. Let
\[ f(x) = (x - 1)^3 \quad \text{and} \quad g(x) = ((x - 3)x + 3)x - 1. \] (7)
In exact arithmetic, it is clear that
\[ f(x) = g(x). \] (8)
A human can read off the solution of equation (1) simply from looking at the expression for \( f \), whereas a machine typically evaluates \( f \) using the expression for \( g \). Numerically, these two functions behave quite differently.

1. Solve equation (1) as accurately as possible using the bisection algorithm by feeding \( f \) to \texttt{bisection2}.
   If the initial interval is denote \([a, b]\), then pick \( a \neq 0, b \neq 2 \), to avoid trivial output and choose \( \delta = \epsilon = 0 \) and \texttt{maxit} = 50 to give us a lot of numbers to look at. Compute both \( f(c) \) and \( g(c) \) and notice the dramatic difference in magnitude between these two values.

2. Solve equation (1) as accurately as possible using the bisection algorithm by feeding \( g \) to \texttt{bisection2}.
   The function will probably terminate after a much smaller number of iterations than before. Examine both the \texttt{flag} and the arrays \( a \) and \( b \). Do the computed intervals even contain the the true root \( r = 1 \)? (Probably not!)

Remark 4 It might be that you choose input which behaved differently than I expected! This is good and I want to hear about it, so that I can incorporate it into next years course. This is my input which triggers the behavior that I wanted you to see

\[
\begin{align*}
[fc, fflag, fit, fa, fb] &= \texttt{bisection2}(f, 0.8, 1.3, 50, 0, 0); \\
[gc, gflag, git, ga, gb] &= \texttt{bisection2}(g, 0.8, 1.3, 50, 0, 0);
\end{align*}
\]
In the first case, you always get intervals which always contain the root. In the second case this desirable property is lost!

Getting the correct sign is critical for the correct application of the bisection algorithm. The expressions for \( f \) and \( g \) both appear to have a problem, when \( x \) is close to 1, right?

3. Explain, why catastrophic cancellation is a problem for \( g \) and we can not trust even the sign of \( g \) when \( x \) is close to 1.

Hint: You will want to examine
\[
\begin{align*}
a &= @(x)((x-3).*x+3).*x; \\
g &= @(x)a(x)-1; \\
x &= 1+\text{lin}snpace(-1,1,1025)*2.^{-22}; \\
y &= g(x); \\
z &= a(x);
\end{align*}
\]
in some detail!

Remark 5 The important thing to keep in mind is that if \( x \) is a real number, then the floating point representation \( \text{fl}(x) \) of \( x \) satisfies
\[ \text{fl}(x) = x(1 + \delta), \quad \delta = \delta(x), \quad |\delta| \leq u \] (9)
where \( u \) is the unit round off. Moreover, if \( x \) and \( y \) are floating point numbers and you ask the computer to evaluate \( x - y \), then you do not get the exact result back, but the machine returns
\[ \text{fl}(x - y) = (x - y)(1 + \delta), \] (10)
where we do not know the exact value of \( \delta \), but we can be certain that \( |\delta| \leq u \).

Remark 6 Let \( \hat{x} \) be one of the midpoints computed during the run of the bisection algorithm. The computer wants to determine the sign of \( g(\hat{x}) \) in order to choose the correct interval. It must first evaluate \( a(\hat{x}) \) and
then do the final subtraction, i.e. $g(\hat{x}) = a(\hat{x}) - 1$. Unfortunately, because of rounding error the poor bastard is unlikely to get the exact value $a(\hat{x})$, at best it produces the floating point representation of $a(\hat{x})$, i.e:

$$\text{fl}(a(\hat{x})) = a(\hat{x})(1 + \delta), \quad |\delta| \leq u,$$

(11)

where $u$ is the unit round off error and even this is not likely to occur. As a result the computer does NOT do the exact subtraction, we want it to do. At best, the computer returns

$$\text{fl}\left(\text{fl}(a(\hat{x})) - 1\right) = \left(a(\hat{x})(1 + \delta) - 1\right)(1 + \nu), \quad |\nu| \leq u$$

(12)

which can be quite different from the number we want, namely $a(\hat{x}) - 1!$ In fact, we have

$$\text{fl}\left(\text{fl}(a(\hat{x})) - 1\right) = \left(a(\hat{x})(1 + \delta) - 1\right)(1 + \nu)$$

$$= a(\hat{x})(1 + \delta + \nu + \delta\nu) - (1 + \nu) = \left(a(\hat{x}) - 1\right) + a(\hat{x})(\delta + \nu + \delta\nu) - \nu$$

It follows that the relative error is

$$\left|\frac{\left(a(\hat{x}) - 1\right) - \text{fl}\left(\text{fl}(a(\hat{x})) - 1\right)}{a(\hat{x}) - 1}\right| \leq \frac{|a(\hat{x})|(2u + u^2) + u}{|a(\hat{x}) - 1|}$$

(13)

Now, the right hand side of this inequality blows up when $\hat{x}$ is close to 1, simply because $a(1) = 1$. It does not follow in the strict mathematical sense that the left hand side must explode as well, but in practice this is exactly what happens. We get relative errors which are greater than 1 and the two numbers

$$\left(a(\hat{x}) - 1\right), \quad \text{fl}\left(\text{fl}(a(\hat{x})) - 1\right)$$

have different sign! As a general rule, if you do not FORCE the machine to behave, then it WILL misbehave, maybe not every time, but even one wrong result is enough to throw off your computation.

4. Explain, why catastrophic cancellation is not a problem for $f$ and that $f(x)$ is computed with the right sign!

**Hint:** $1$ is floating point number, so if $\hat{x}$ is a floating point number, then $\hat{x} - 1$ is computed with a very small relative error.

**Remark 7** The difference between the two situations is insidious. Here in the second case the computed midpoint $\hat{x}$ is almost certainly not the exact midpoint, but we literally do not care as long as we get to pick the right interval! In this case we always pick the right interval, because the subtraction $\hat{x} - 1$ involves two floating point numbers and is therefore done with a small relative error. The computer returns

$$\text{fl}(\hat{x} - 1) = (\hat{x} - 1)(1 + \nu), \quad |\nu| \leq u$$

(14)

and since $u \ll 1$ we have a relative error which is less than 1 and we get the correct sign!
Problem 4 In this problem you will develop a subroutine which can be used to compute \( \sqrt{\alpha} \) for all \( \alpha > 0 \).

We begin by dramatically simplifying the problem.

1. Any single precision floating point number \( \alpha > 0 \) can be written as
   \[
   \alpha = (1.f_1f_2f_3\ldots f_{23})_2 \times 2^m,
   \]
   where \( m \) is the exponent and \( f_i \in \{0, 1\} \) are the individual bits of the mantissa. Show that in order to compute \( \sqrt{\alpha} \) it suffices to have the ability to compute \( \sqrt{x} \) where \( x \in [1,4] \).

   **Hint:** In general \( \sqrt{ab} = \sqrt{a} \sqrt{b} \) and it is very simple to compute the square root of \( 2^{2k} \), is it not?

2. If \( \alpha \in [1,4] \) then a really good initial guess for \( \sqrt{\alpha} \) is given by
   \[
   x_0(\alpha) = \frac{1}{3} \alpha + \frac{17}{24}
   \]
   Write a MATLAB function \texttt{my.sqrt.m} which uses \texttt{newton.m} to compute the square root of a real number \( z \in [1,4] \). Compare your output to the built-in function \texttt{sqrt} in MATLAB. I expect you to find relative errors which are very small!

3. The initial guess \( x_0 = x_0(\alpha) \) is an extremely good approximation of \( f(x) = \sqrt{x} \) on the interval \( [1,4] \).
   Prove, that
   \[
   -\frac{1}{24} \leq \sqrt{\alpha} - x_0(\alpha) \leq \frac{1}{24}, \quad x \in [1,4]
   \]
   by explicitly computing the range of the error function
   \[
   g(\alpha) = x_0(\alpha) - \sqrt{\alpha},
   \]
   using standard techniques for analyzing functions.