TANA09 Datatekniska beräkningar

Laboration 1. Error analysis and equations

Name:_____

LiU-Id:_____

Email:_____

Name:_____

Liu-Id:_____

Email:_____

Approved:_____ Sign:_____

Corrections:_____

1 Introduction

If we want to use a computer to perform mathematical calculations we have to approximate the set of real numbers a finite floating point number system. This means that the basic arithmetic operations can only be performed at a fixed level of precision. In this exercise we will investigare how these errors influence the results of numerical calculations performed using a computer.

Most often the errors introduced by the floating point system are small compared to other errors that appear in scientific computations. This is not always the case for instance if cancellation of significant digits occur.

One of the points of the exercise is to demonstrate that mathematically equivalent formulations of a problem may lead to different results when implemented on a computer. An analysis of the errors in numerical calculations is important since it allows the researcher to pick the specific formulation of a problem that leads to the smallest error in the result.

Note that many of the questions are of theoretical nature and can be answered after the scheduled time for the exercise.

2 Representation of Real Numbers

A *floating point system* is characterized by (β, t, L, U), where β is the base, t is number of digits in the fractional part, L and U are the smallest and largest allowed exponents.

In this exercise we will investigate some basic properties of the IEEE Double Precision floating point system commonly used by modern computers.

Exercise 2.1 The *unit roundoff* μ for a floating point number system represents an upper limit for the relative error when storing a real number x in the number system. Give the expression for the unit roundoff μ in terms of the parameters (β, t, L, U).

$\mu =$_____

Exercise 2.2 The base is $\beta = 2$. Experiment with different values of n in the expression $1 + 2^{-n}$. For instance you can write

```
>> n=50 , 1+2^-n
```

What is μ and t?

$\mu =$_____ $t =$_____

Hint Matlab displays the result as 1.0000 unless the result is exactly 1.

Exercise 2.3 Give an upper limit for the *absolute* error when storing the number $y = \sqrt{2}$ on the computer. How many correct decimal digits does the approximation of $\sqrt{2}$, which is stored in the computer memory, have?

Absolute error:_____ Correct decimal digits:_____

Hint The definition of the unit roundoff gives you an upper limit for the *relative error*.

Exercise 2.4 Approximately how many significant digits do we have when storing real numbers in Matlabs floating point system?

Answer:_____

It is possible to study in detail how a specific number is stored in computer memory by writing

```
>>format hex, 3
```

The 64 bits that are used to represent the double precision number is then displayed on screen in hexadecimal form. Use the following table to translate the hexadecimal representation into binary digits.

HEX	0	1	2	3	4	5	6	7
BIN	0000	0001	0010	0011	0100	0101	0110	0111
HEX	8	9	a	b	c	d	e	f
BIN	1000	1001	1010	1011	1100	1101	1110	1111

Exercise 2.5 How is the number 3 stored in memory? Also check how the number -3 is stored. Which bit is used to represent the sign of the number? Which bits are used to represent the exponent and the fractional part of the number?

Answer:_____

Hint: The number is $(3)_{10} = (11)_2 = (1.1)_2 \cdot 2^1$. Reset to regular display format by writing `format short`.

Exercise 2.6 What is the largest possible number x that can be stored in the IEEE Double Precision floating point system? Write an expression for x as a binary number and also evaluate it to get the corresponding value of x in the decimal number system.

Answer:_____

3 Numerical Computation of Limits

In this section we wills study the limit of a function. It is known that

$$\lim_{x\to 0} f_1(x) = 1, \qquad \text{where} \qquad f_1(x) = \frac{\mathrm{e}^x - 1}{x}.$$

Investigate how accurate it is possible to calculate the limit in Matlab by plotting the error (in log-scale) for a wide range of x-values.

The following Matlab code will create a vector x, that contains values between 1 and 10^{-16}, and compute the a vector `f1` that contains the corresponding function values. The error $|f_1(x) - 1|$ is also displayed

```
>> x=10.^-(0:0.1:16);
>> f1=(exp(x)-1)./x;
>> loglog(x,abs(f1-1)), xlabel('x'), ylabel('|f_1(x)-1|')
```

Note the dot in front of the division on the second line. What is its purpose?

Exercise 3.1 Which value x gives the smallest error? What happens for very small values of x?

Answer:_____

Exercise 3.2 Investigate the accuracy when using the mathematically equivalent formula,

$$f_2(x) = \frac{\mathrm{e}^x - 1}{\ln(\mathrm{e}^x)}.$$

Compute a vector `f2` containing the function values obtained using the new formula using the Matlab command

```
>> f2=(exp(x)-1)./log(exp(x));
```

What happens now?

Answer:_____

Obviously the two equivalent mathematical experessions behave differently. In the following few exercises we will calculate the dominating terms of the errors $|f_1(x) - 1|$ and $|f_2(x) - 1|$. There are two contributing sources of error. The truncation error and the computational error.

Exercise 3.3 Use Taylor series expansion to estimate the *truncation error*,

$$R_T = |f_1(x) - 1| \leq \_\_\_\_\_$$

The truncation error is the same for both expressions $f_1(x)$ and $f_2(x)$. Why is that?

Answer:_____

Exercise 3.4 Assume that all the numerical operations (`-`,`/`,`exp`, etc) are performed with a *relative error* less than, or equal to, the unit roundoff μ. Perform an analysis of the computational errors for both expressions $f_1(x)$ and $f_2(x)$. Find an estimates

$$R_X \leq |\text{fl}[f_1(x)] - f_1(x)|.$$

Answer:

Exercise 3.5 Illustrate the results by plotting $|f_1(x) - 1|$ and $R_T + R_X$, for x-values between 1 and 10^{-16}, in a log-log scale. Do the same for the expression $f_2(x)$. How does the error analysis compare with the actual errors?

Answer:

Print and hand-in the graph together with your report.

4 Solution of Non-Linear Equations

In this section we will solve equations using a standard solver from Matlab. We will also use Newtons method to implement division.

4.1 The `fzero` solver

In this exercise we will investigare the convergence speed for the built-in Matlab function `fzero`. Again we will attempt to solve the equation

$$f(x) = \sqrt{1+x}e^{x/2} - 2\sin(2x)(x+x^2).$$

We are intressted in the root $x^* \approx 1.5$.

Exercise 4.1 Plot the function $f(x)$ on the interval $[0, 3]$. Also use `fzero` to find the root x^*. Type the command you use and the result.

Answer:_____

Exercise 4.2 In order to see more clearly what `fzero` does we write a Matlab function

```
function[y]=fun(x)
 y=sqrt(1+x).*exp(x/2)-2*sin(2*x).*(x+x.^2);
 fprintf('%18.15f %6.2e\n',x,y);
end
```

This allows us to see exactly which x-values are used during the iterations. We also change the stopping criteria and solve the equation again by typing

```
>> options = optimset('TolFun',1e-15);
>> x = fzero( @fun , 1.5 , options );
```

How many function evaluations are needed to find the root?

Answer:_____

Exercise 4.3 Copy the x values used during the iterations into a vector `x`. If we assume that the final x-value is exact we can obtain a vector `e` containing the error in each step by typing `e=abs(x-x(end))`. Plot the errors using `semilogy`. Also write down the errors during the last 5 iterations. Does this look like linear or quadratic convergence?

Answer:_____

Hint Since the function just prints the x-value and $f(x)$ on each row you can type `x=[` in the Matlab terminal. Then `paste` the numbers and end with `];`. To get the first column type `x=x(:,1)`. To see the errors more clearly use `format short e`.

Exercise 4.4 If the order of convergence is p we have a relation $e_k \approx Ce_{k-1}^p$. Since the errors are known obtain a relation

$$\frac{e_k}{e_{k-1}} = (\frac{e_{k-1}}{e_{k-2}})^p.$$

Taking logarithms we find an expression for p. Make use of the vector `e` containing the errors and try to find a value for p. If you type

```
>> e,k=15,e(k),p=(log(e(k)/e(k-1))/log(e(k-1)/e(k-2)))
```

you can easily see which values are used to estimate p and its value. What is your experimentally determined value of p? What iteration index k was used?

Answer:_____

Hint Typically for equation solvers the approximation x_k has to be sufficiently close to x^* for the rapid convergence to occur. This means that normally no significant improvements take place during the first iterations. Also once the approximation x_k is close enough the convergence is so fast that the unit round-off is reached within just two or three steps. This means that in practice it is difficult to estimate the order of convergence from experimental data.

4.2 Implementation of Division

When designing a computer chip you have to decide what operations should be supported by hardware and which ones that should be implemented as software. For instance the square root operation is sometimes included as a basic instruction in the hardware. On the other hand if you want to keep the instruction set small and the chip as simple as possible one might look into possible instructions that might as well be left out and implemented through software.

A basic arithmetic operation sometimes left out from hardware is floating point division. In this exercise we will look at how to do an efficient implementation of the division. The problem is really simple but great care has to be taken to reduce the work needed, and also to make sure the result is accurate.

Exercise 4.5 First compute $z = 1/y$. This can be formulated as a non-linear equaton,

$$f(z) = y - \frac{1}{z} = 0.$$

Apply Newtons method to the equation $f(z) = 0$ and derive the corresponding iteration formula. Take care not to use any divisions in your expression.

Answer:

Remark One might try to solve for $z = x/y$ directly but then its not possible to avoid divisions in the Newton formulas.

Exercise 4.6 On the computer the floating point numbers are of the form

$$y = \pm(1.f)_2 2^e.$$

Since $(2^e)^{-1} = 2^{-e}$ we only need to compute $1/y$ for $1 \leq y < 2$. This means that the root of the equation $f(z) = 0$ satisfies $\frac{1}{2} < z \leq 1$. Suppose we use the starting approximation $z_0 = \frac{3}{4}$. What is the maximum initial error $e_0 = |z - z_0|$?

Answer:

The Newton iteration can be written $z_{k+1} = \phi(z_k)$, where $\phi(z)$ is the iteration function. What is $\phi(z)$ here?

Answer:

Exercise 4.7 The error analysis for the Newton iteration leads to a formula,

$$e_{k+1} = \frac{\phi''(\eta)}{2} e_k^2,$$

where η belongs to the interval (z_k, z). Use the available information to find the maximum value of $|\phi''(\eta)|$. Also find the maximum number of iterations required until $|e_k| \leq \mu$, where μ is the unit roundoff. Present the calculations!

Answer:

Exercise 4.8 Use the above results and implement a function

```
>>  z = Division( x , y );
```

The function should solve the problem in two steps. First compute $1/y$ and then multiply $z = x \cdot (1/y)$. You can assume that $1 \leq y < 2$.

Use your function to compute $z = 1.32/y$, for `y=1:0.01:2`. Plot the relative error in the result. In order to compute the errors you can assume that the Matlab division x/y is exact. Is the result good? How many multiplications are needed to compute one division?

Answer:

Remark It is possible that the Newton method converges faster for certain values of y. However by implementing the worst case scenario we avoid conditional statements in the loop. This has the advantage of ensuring that a division is equally fast regardless of the arguments x and y.

It is possible to improve the speed of the algorithm by picking a better starting value x_0. For instance divide the interval $[1, 2)$ in 8 subintervals of equal length $1 + [k/8, (k+1)/8)$, $k = 0, 1, 2, \ldots, 7$. Then pick a starting approximation $z_{0,k}$ depending on which interval y belongs to. This was the initial error is at most $e_0 \leq 1/16$ and only $N = 4$ iterations is needed. Thus we can trade memory (i.e. a table of starting approximations) for computational speed (i.e. number of multiplications). Though this is not convinient to write in Matlab.