BLG202E

HOMEWORK 1

Seyfülmülük Kutluk

150180073

Question 1)

1. The function $f1\ (x0\ ,\ h)$ = sin($x0$ + h) − sin ($x0$) can be transformed into another form, $f2\ (x0\ ,\ h)$ using the trigonometric formula given below:

sin(ϕ) − sin(ψ) = 2 cos (ϕ + ψ 2) sin (ϕ − ψ 2)

Thus, $f1$and $f2$ have the same values, in exact arithmetic, for any given argument values $x0$ and h.
a. Derive $f2\ (x0\ ,\ h)$.

b. Suggest a formula that avoids cancellation errors for computing the approximation ($f(x0 + h)$ − $f(x0$))/h to the derivative of $f(x)$ = sin(x) at x = $x0$. Write a Python program that implements your formula and computes an approximation of f'(1.2), for h = 1e-20, 1e-19, …, 1.

Solution 1)

$f1\ (x0\ ,\ h)$ = sin($x0$ + h) − sin ($x0$)

we are going to implement the formula of sin(ϕ) − sin(ψ) = 2 cos ((ϕ + ψ)/ 2) sin ((ϕ − ψ)/ 2) to f1

ϕ=x0+h and ψ=x0

sin($x0$ + h) − sin ($x0$) = sin(ϕ) − sin (ψ) =2 cos ((ϕ + ψ)/ 2) sin ((ϕ − ψ)/ 2)

sin($x0$ + h) − sin ($x0$) = 2 cos ((2x0+h)/ 2) sin ((h)/ 2)

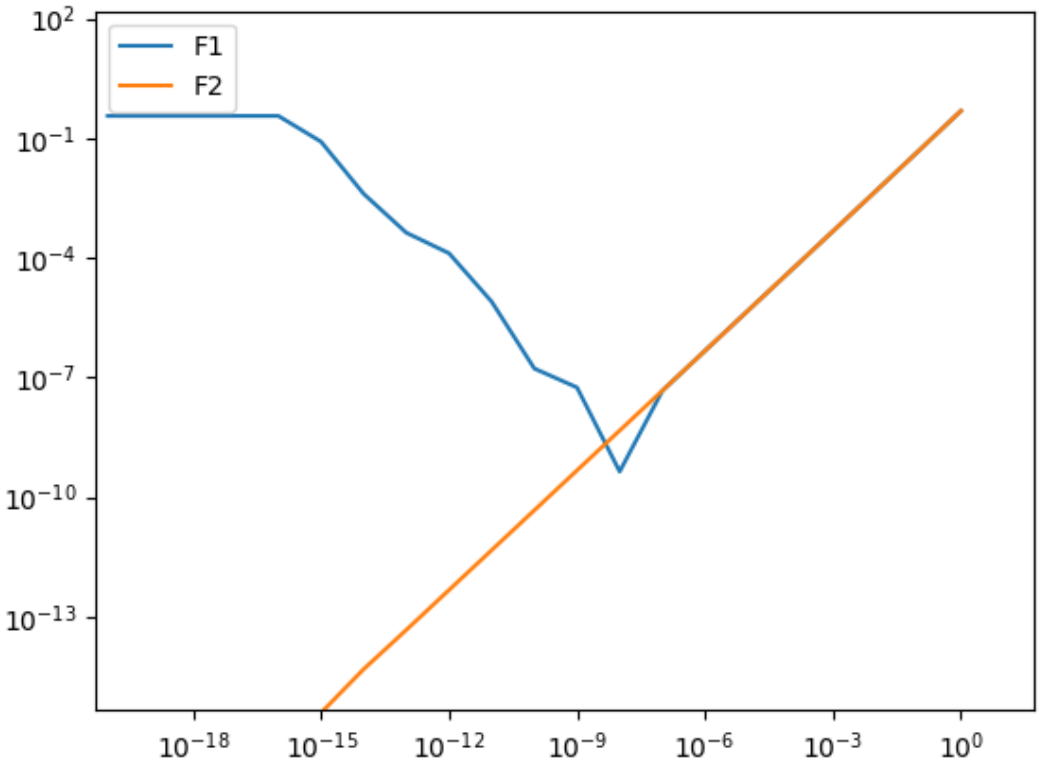
2 cos ((2x0+h)/ 2) sin ((h)/ 2) is the f2 and we find it from the formula above

Derivative for f1. using this formula f ' (x0) ≈ (f(x0 + h) − f(x0))/h is

(sin($x0$ + h) − sin ($x0$))/h

Derivative of f2 using this formula if ' (x0) ≈ (f(x0 + h) − f(x0))/h is

(2 cos ((2x0+h)/ 2) sin ((h)/ 2))/h



We can clearly observe cancellation errors for computing the approximation $(f(x0 + h) - f(x0))/h$ to the derivative of $f(x)$ = sin(x) at x = x0 in f1. This occurs because of the subtracting two quantities that are close to each other.

To remove the cancellation errors, we can change the f1 to f2 with the given formula above. As can be seen in the figure there is no cancellation errors in f2.

Question 2) 2. Consider the linear system $\begin{bmatrix} a & b \\ b & a \end{bmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ with $a, b > 0$.

1) If $a \approx b$, what is the numerical difficulty in solving this linear system?
2) Suggest a numerically stable formula for computing $z = x + y$ given a and b
3) Determine whether the following statement is true or false, and explain why: "When $a \approx b$, the problem of solving the linear system is ill-conditioned but the problem of computing $x + y$ is not ill-conditioned."

Solution 2)

1) The determinant of the matrix $\begin{pmatrix} a & b \\ b & a \end{pmatrix}$ is a^2-b^2 and since the $a \approx b$ the determinant is close to zero.
   Since the solution of Ax=B is x=(A^-1)*B
   and A^(-1)=1/|A| so the result of the $\begin{pmatrix} x \\ y \end{pmatrix}$ is (1/|A|)/B

Since a small perturbation in the data produce a large difference in the result the problem is ill-conditioned.

2) $\begin{matrix} a & b \\ b & a \end{matrix}$]$\binom{x}{y}$ = $\binom{1}{0}$

ax+by=1 ----------→ ax+by-1=0

bx+ay=0 ----------→ bx+ay=0

if we add two equations we will have ax + bx + ay + by = 1

we will combine ax, bx in x parentheses and ay, by in y parentheses.---------→ x(a+b) + y(a+b)=1

we will combine x(a+b), y(a+b) in a+b parentheses. ----------→ (a+b)(x+y)=1

(x+y)=1/(a+b)

z=(x+y)

z=1/(a+b) ----------→ Numerically stable formula

3) "When $a \approx b$, the problem of solving the linear system is ill-conditioned but the problem of computing $x + y$ is not ill-conditioned.".

As explained in part 1 the determinant is close to zero. And this produces an ill-conditioned problem.

However, in part 2 we observe that x+y=1/(a+b) it depends on the a + b and it is not ill-conditioned.

The statement is true.

Question 3) With exact rounding, we know that each elementary operation has a relative error which is bounded in terms of the rounding unit η, e.g., for two floating point numbers x and y,

$fl\,(x + y) = (x + y)\,(1 + \epsilon)$, $|\epsilon| \leq \eta$. But is this true also for elementary functions such as sin, ln, and exponentiation?

Consider exponentiation, which is performed according to the formula below.

x ^y = e^ $yln(x)$ $(assuming\ x > 0)$

Estimate the relative error in calculating x^y in floating point, assuming $fl(lnz) = (lnz)(1 + \epsilon)$, $|\epsilon| \leq \eta$, and that everything else is exact. Show that the sort of bound we have for elementary operations and for ln does not hold for exponentiation when $x\ y$ is very large.

Solution 3) fl(fl(x) ± fl(y)) = (fl(x) ± fl(y))(1 + ϵ1) $|\epsilon| \leq \eta$.

If we take the exact rounding of lnx for example exact rounding of the f(x)=ln(x) is f(x)=ln(x)(1+E)

If we implement this to our example we can compare them and find whether

$F1(x\ ^{\wedge}y) = e^{\wedge}\ yln(x)$

$F1(x\ ^{\wedge}y) = e^{yln(x)(1+E)}$ ----------→ exact rounding

$F1(x\ ^{\wedge}y) = e^{yln(x)} * e^{yln(x)E}$ ----------→ since $e^{\wedge}(yln(x)$ is equal to x^y we can put it instead

$F1(x\ ^{\wedge}y) = x^y * e^{yln(x)E}$ ----------→ at this point we can compare the two of them and find the relative error

Relative error is |u-v|/u

$|x^y - x^y * e^{yln(x)E}|/x^y$ ----------→ we will take them in x^y parenthesis.

$|x^y (1- e^{yln(x)E})|/x^y$----------→

$(1- e^{yln(x)E})$ ----------→

$(1- e^{yln(x)E}) = 1\text{-}1+\epsilon ylnx+(1/2)*(\epsilon ylnx)^2+(1/6)(\epsilon ylnx)3+O(\epsilon^3)$ ----------→Taylor expansion

$(1- e^{yln(x)E}) = \epsilon ylnx+O(\epsilon^2)$ ----------→

$(1- e^{yln(x)E}) = \epsilon ylnx$ ----------→ For small numbers this is true however for large numbers this is not true

Lets say $x=e^{10000}$, $y=1000$, $E=10^{-10}$

$|x^y - x^y * e^{yln(x)E}|/x^y$

$F1(x\ ^{\wedge}y) = e^{\wedge}\ yln(x)$

$F1(x\ ^{\wedge}y) = e^{\wedge}\ 1000ln(e^{10000}) = e^{1000000}$ ----------→ This is the real result

$x^y * e^{yln(x)E} = e^{10000000*10^{\wedge}(-10)} = e^{10^{\wedge}(-3)}$ ----------→ Measured result

The relative error = $|e^{10000000}-e^{10^{\wedge}(-3)}|/e^{10000000}$ İt can be observed that the error is too much for big numbers.

Question 4) You are required by a computer manufacturer to write a library function for a given floating point system to find the cube root y 1/3 of any given positive number y. Any such relevant floating-point number can be represented as $y = a * 2\ ^{\wedge}e$, where a normalized fraction ($0.5 \leq a < 1$) and e is an integer exponent. This library function must be very efficient, and it should always work. For efficiency purposes it makes sense to store some useful constants ahead of computation time, e.g., the constants $2\ ^{\wedge}1/3$, 2/3 and a/3, and should these prove useful.

a. Show how $y\ ^{\wedge}1/3$ can be obtained, once $a^{\wedge}$ 1/3 has been calculated for the corresponding fraction, in at most five additional flops.

b. Derive the corresponding Newton iteration. What is the flop count per iteration?

c. How would you choose an initial approximation? Roughly how many iterations are needed? (The machine rounding unit is 2 −52.)

Solution 4)

Let's say given floating point is y when we take the cube root of y and $a^{(1/3)}$ is already calculated.

$y^{1/3}=(a^{1/3})*2^{e/3}$

$(a^{1/3}) =(2^{e/3})/(y^{1/3})$ this means $(2^{e/3})/(y^{1/3})$ is already calculated

since we can calculate $(2^{e/3})$ we can calculate $(y^{1/3})$

b)

$x_{n+1}= x_n -(f(x_n)/f'(x_n))$ the Newtons iterations rule

$y_{n+1}= y_n -(y_n\text{^}(1/3)/y_n\text{^}(-2/3)) = y_n -3\ y_n = -2\ y_n$ -----→ İnstead of y I will write $y = a * 2\ \text{^}e$

$y_{n+1}= -2\ y_n$

$a_{n+1} * 2\ \text{^}e_{n+1}= -\ a_{n+1} * 2\ \text{^}(e_{n+1}+1)$ ----------→ n flops needed

c) I will choose first iteration in the middle of the array.

Later, I will observe in which side the real number stay.

Let's say it stayed at the right I will iterate the middle part of the right part.

I will continue until the number is matched or it is too close.