Homework 1

ME 2060: Numerical Methods

Jacob Cunningham[1]

Monday 20th January, 2025

Abstract

Homework discussion, calculations, and answers are provided herein.

Keywords Numerical Methods, Mechanical Engineering

1 Effects of roundoff and truncation errors on numerical accuracy

The one-sided finite difference scheme to approximate the first derivative of a function f is defined as:

$$f_h(x) = \frac{f(x+h) - f(x)}{h} \approx f'(x), \tag{1}$$

where h is the step size if no roundoff error exists, then the accuracy of the scheme is dermined soley by **truncation error**

$$t_e(h) = |f'(x) - f_h(x)| = O(h), \quad f'(x) = f_h(x) + O(h), \tag{2}$$

where the big O - notation means there exists a constant K such that $O < K \cdot h$ for all h. In the presence of roundoff errors, x cannot be represented exactly; instead, it is represented by the rounded value $\tilde{x}$ with the associated roundoff error $r = |\tilde{x} - x|$.

1.1 Part A

Show that the total error of the finite-difference approximation consists of both truncation error $t_e(h)$ due to the finite-difference scheme and the roundoff error r:

$$\epsilon(h) := |f'(x) - f_h(\tilde{x})| = O(h) + \frac{r}{h} \tag{3}$$

Hint: Start from $f'(x) = f_h(x) + O(h)$, and consider that the computation of the finite difference $f_h(x)$ approximation already involves roundoff errors, e.g., $f(x) = \tilde{f}(x) + r$, $f(x+h) = \tilde{f}(x) + h + r$.

When accounting for rounding error, the computed version of (1) is:

$$\tilde{f}_h(x) = \frac{\tilde{f}(x+h) - \tilde{f}(x)}{h}, \tag{4}$$

where $\tilde{f}(x+h)$ and $\tilde{f}(x)$ are the computed values of $f(x+h)$ and $f(x)$, respectively. Since the computed values are equal to the exact values plus rounding error, (4) can be reduced:

---

[1]Correspondence to: jjc132@pitt.edu

$$\tilde{f}_h(x) = \frac{(f(x+h) + r_1) - (f(x) + r_2)}{h}$$
$$= \frac{f(x+h) - f(x) + (r_1 - r_2)}{h}$$
$$= \frac{f(x+h) - f(x)}{h} + \frac{r_1 - r_2}{h}$$
$$= f_h(x) + \frac{r_1 - r_2}{h}$$

This can be rewritten as:

$$\tilde{f}_h(x) = f_h(x) + \frac{r}{h}, \tag{5}$$

where $r = r_1 - r_2$.

The important step here is to realize that the rounding errors at $\tilde{f}(x+h)$ and $\tilde{f}(x)$ are not necessarily the same value. Substituting (5) into (2) gives:

$$f'(x) = (\tilde{f}_h(x) - \frac{r}{h}) + O(h)$$
$$= \tilde{f}_h(x) - \frac{r}{h} + O(h)$$

Since rounding error would add error contribution the result can be rewritten as:

$$f'(x) = \tilde{f}_h(x) + \frac{r}{h} + O(h) \tag{6}$$

The difference between the exact derivative and the computed derivative using this scheme is thus:

$$f'(x) - \tilde{f}_h(x) = O(h) + \frac{r}{h}$$

Since both error contributions are always positive:

$$|f'(x) - \tilde{f}_h(x)| = O(h) + \frac{r}{h}, \tag{7}$$

which is equivalent to (3).

> **Answer 1A**
> The deriviation above shows how (1) can be written as (4) when accounting for rounding error. (4) was reduced to (5) and substituted into (2) to give (6) which was rearranged to (7) and is equivalent to (3).

1.2 Part B

From Part A, what can you say about the numerical accuracy of the finite difference scheme as the step size h is continually decreased?

> **Answer 1B**
> The roundoff error term r/h will dominate and roundoff errors will be amplified. Conversely if h is increased the truncation error term will dominate. This implies that there is probably an optimal step size to balance both error terms.

1.3   Part C

The second order central difference approximation is defined as

$$f_h^c(x) = \frac{f(x+h) - f(x-h)}{2h} \approx f'(x) \tag{8}$$

and has a truncation error of $t_e(h) = |f'(x) - f_h^c(x)| = O(h^2)$.

Within the notebook `Week2_FD_students.ipynb` available on Canvas Module 1, add a function to evaluate the finite difference approximation f_h^c for the derivative of $f(x) = \sin(x)$ with step size h at a fixed $x = x_0 = 1$. For the same array h of step sizes as in the notebook, calculate the array consisting of errors between the finite difference approximation f_h^c and the exact dervative f'. Plot the approximation error for the central differences scheme as a function of step size h. Display your plot in log format in addition to the previous plot for forward differences.

Let $f(x) = \sin x$ and $x = x_0 = 1$.

Numerical Error $\varepsilon(h)$ vs. Step Size h
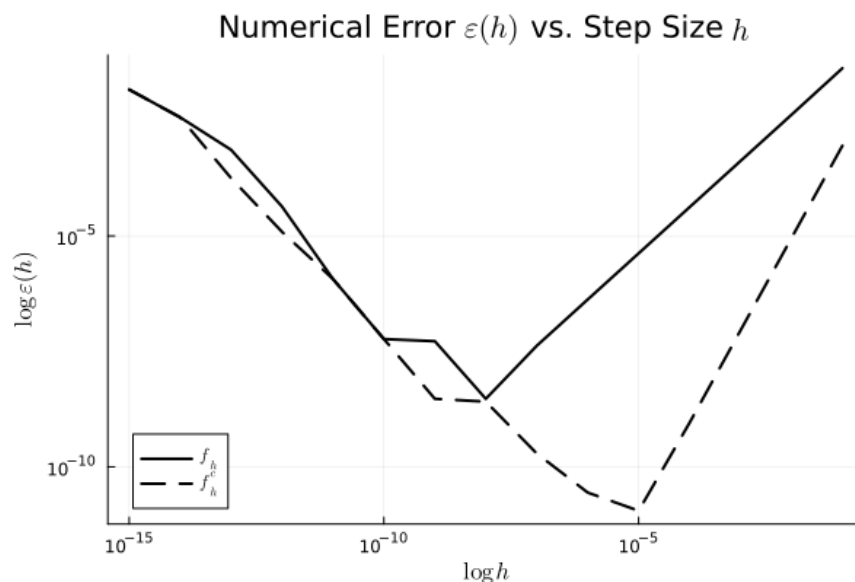
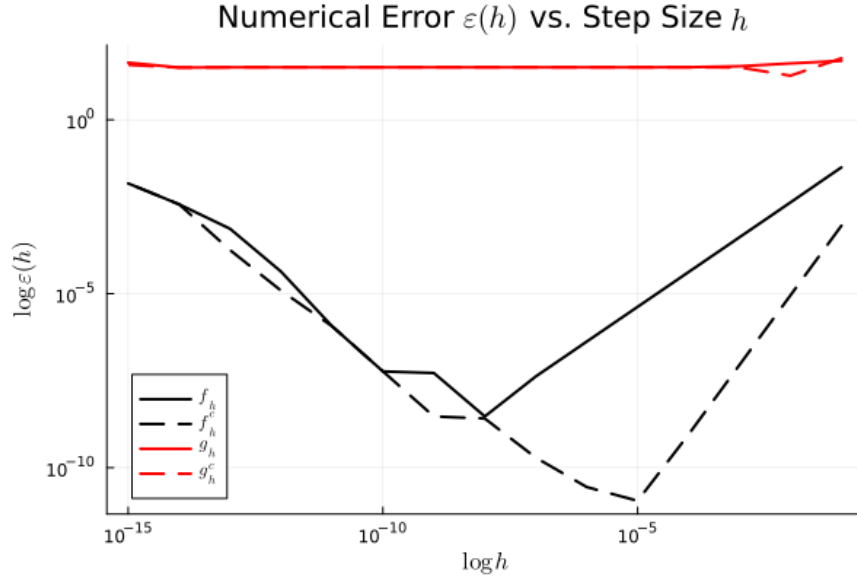The code to produce this plot is provided in Program 1.

> **Answer 1C**
> The figure above shows that the central difference scheme is superior to the forward difference scheme.

1.4   Part D

Now repeat the same above steps for the function $g(x) = \sin(100x)$ and its derivative $g'(x)$. Add your plots for the forward difference and central difference numerical errors to the same plot. Make sure to use a different linestyle, and specify your legend entries. Looking at your plot, what can you say about the accuracy of each method to approximate the derivative for $\sin(x)$ and $\sin(100x)$ at $x_0 = 1$? Can you explain why?

Let $g(x) = \sin(100x)$ and $x = x_0 = 1$.

Numerical Error $\varepsilon(h)$ vs. Step Size h

The code to produce this plot is provided in Program 2.

Answer 1D

The central differences scheme is more accurate in both cases and this is due to the fact that the central difference scheme converges faster opposed to the forward differences scheme. So as the step size is decreased, the central differences scheme truncation error decreases quadratically while the forward differences scheme decreases linearly. Both schemes are affected by roundoff error as the step size is decreased.

For low frequency functions, e.g., $f(x)$, both methods performed well. There is a U-shaped curve defining the total error which makes sense since at lower step sizes round-off errors would dominate and then at higher step sizes truncation error would dominate.

For high frequency functions, e.g., $g(x)$, both methods struggle because the high frequency amplifies the truncation error.

---

2   Errors in Scientific Computing

The sine function is given by the infinite series

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \tag{9}$$

2.1   Part A

Calculate the forward and backward errors if we approximate the sine function by using only the first term in the series for $x = 0.1, \ 0.5, \ 1.0$.

Let $f(x) = \sin x$. The numerical approximation of $f(x)$ using the first term is defined as:

$$\tilde{f}(x) = x$$

The forward error is found from:

$$\epsilon_f = \tilde{f}(x) - f(x) \tag{10}$$

The backward error is found from:

$$\epsilon_b = \tilde{x} - x \ni \tilde{f}(x) = f(\tilde{x}) \tag{11}$$

For this first order approximation of $\sin x$, $\tilde{x}$ is found as follows:

$$\tilde{f}(x) = f(\tilde{x})$$
$$x = \sin(\tilde{x})$$
$$\arcsin(x) = \tilde{x}$$

3×6 DataFrame

Row	x	Approx x	f(x)	Approx f(x)	Forward Error	Backward Error
	Float64	Float64	Float64	Float64	Float64	Float64
1	0.1	0.100167	0.099833	0.1	0.000167	0.000167
2	0.5	0.523599	0.479426	0.5	0.020574	0.023599
3	1.0	1.5708	0.841471	1.0	0.158529	0.570796

The code to produce the table above is provided in Program 3.

> **Answer 2A**
> For this first order approximation of $\sin x$:
>
> - The forward error and backward error at x = 0.1 is 0.000167 and 0.000167, respectively.
> - The forward error and backward error at x = 0.5 is 0.020574 and 0.023599, respectively.
> - The forward error and backward error at x = 1.0 is 0.158529 and 0.570796, respectively.

2.2 Part B

Calculate the forward and backward errors if we approximate the sine function by using only the first two terms in the series for $x = 0.1,\ 0.5,\ 1.0$.

The numerical approximation of $f(x) = \sin x$ using the first two terms is defined as:

$$\tilde{f}(x) = x - \frac{x^3}{3!}$$

The forward error can be found from (10), and the backward error can be found from (11).

For this second order approximation of $\sin x$, $\tilde{x}$ is found as follows:

$$\tilde{f}(x) = f(\tilde{x})$$
$$x - \frac{x^3}{3!} = \sin(\tilde{x})$$
$$\arcsin(x - \frac{x^3}{3!}) = \tilde{x}$$

3×6 DataFrame

Row	x	Approx x	f(x)	Approx f(x)	Forward Error	Backward Error
	Float64	Float64	Float64	Float64	Float64	Float64
1	0.1	0.1	0.099833	0.099833	-0.0	-0.0
2	0.5	0.499705	0.479426	0.479167	-0.000259	-0.000295
3	1.0	0.985111	0.841471	0.833333	-0.008138	-0.014889

The code to produce the table above is provided in Program 4.

Answer 2B
For this second order approximation of $\sin x$:

- The forward error and backward error at x = 0.1 is -0.0 and -0.0, respectively.
- The forward error and backward error at x = 0.5 is -0.000259 and -0.000295, respectively.
- The forward error and backward error at x = 1.0 is -0.008138 and -0.014889, respectively.

---

3 Condition Number & Stability

This is Exercise 1 from Section 1.4, page 26 of Driscoll and Braun [2018]. Exercises are also available at the end of each section of a chapter in the online textbook. Refer to your textbook for cross-referenced equations and tables.

Consider the formulas

$$f(x) = \frac{1 - \cos x}{\sin x}, \quad g(x) = \frac{2\sin^2 x/2}{\sin x},$$

which are mathematically equivalent, but they suggest evaluation alrogirthms that can behave quite differently in floating point arithmetic.

3.1 Part A

Using (1.2.6), find the relative condition number of f. Note that because f & g are mathematically equivalent, their condition numbers are the same. Show that the condition number approaches to 1 as $x \to 0$, which means it is possible to compute the function accurately near zero.

The condition number is defined as:

$$\kappa_f(x) = \left| \frac{x f'(x)}{f(x)} \right| \tag{12}$$

The function $f(x)$ is given as:

$$f(x) = \frac{1 - \cos x}{\sin x}$$

The derivative $f'(x)$ is:

$$f'(x) = \frac{1 - \cos x}{\sin^2 x}$$

The function $g(x)$ is given as:

$$g(x) = \frac{2\sin^2 x/2}{\sin x}$$

The derivative $g'(x)$ is:

$$g'(x) = \frac{1 - \cos x}{\sin^2 x}$$

The condition number of $f(x)$ (after simplification) is:

$$\kappa_f(x) = \left| \frac{x}{\sin x} \right| \tag{13}$$

The condition number of $g(x)$ (after simplification) is:

$$\kappa_g(x) = \left| \frac{x}{\sin x} \right| \tag{14}$$

In order to show the condition number approaches 1 as $x \to 0$ the following is computed:

$$\lim_{x \to 0} \left| \frac{x}{\sin x} \right| \tag{15}$$

As x gets increasingly small $\sin x \approx x$ which leads to:

$$\begin{aligned}
\lim_{x \to 0} \left| \frac{x}{\sin x} \right| &= \lim_{x \to 0} \left| \frac{x}{x} \right| \\
&= \lim_{x \to 0} |1| \\
&= 1
\end{aligned}$$

Answer 3A
(12) was used to calculate the condition number for $f(x)$ and $g(x)$ which are equivalent as show in (13) and (14), respectively. Finally, it was shown that (15) approaches 1 as $x \to 0$.

3.2 Part B

Compute $f(10^{-6})$ using a sequence of four elementary operations. Using Table 1.1 on page 13, make a table like the one shown in Demo 1.4.1 in the book that shows the result of each elementary result and the numerical value of the condition number of that step.

The table referenced in Driscoll and Braun [2018] is reproduced in Table 1.

When two functions f and g are combined in a chain as $h(x) = f(g(x))$ the composite condition number is given by:

$$\kappa_h(x) = \kappa_f(g(x)) \cdot \kappa_g(x) \tag{16}$$

The function f consists of four elementary steps:

- $\cos x$
- $x + c$

Table 1: Relative $\kappa_f(x)$ of elementary functions

Function $f(x)$	Condtion Number $\kappa_f(x)$
$x + c$	$\frac{|x|}{|x+c|}$
cx	1
x^p	$|p|$
e^x	$|x|$
$\sin x$	$|x \cot x|$
$\cos x$	$|x \tan x|$
$\log x$	$\frac{1}{|\log x|}$

- $\sin x$

- cx

4×3 DataFrame

Row	Function	Value	Condition Number
	LaTeXStr...	Float64	Float64
1	$\\cos\{10\hat{}\{-6\}\}$	1.0	1.0e-12
2	$1 - \\cos\{10\hat{}\{-6\}\}$	5.00044e-13	1.99982e6
3	$\\sin\{10\hat{}\{-6\}\}$	1.0e-6	1.0
4	$f(x)$	5.00044e-7	1.99982e6

The code to produce the table above is listed in Program 5.

Answer 3B

For $f(x)$ where $x = 10^{-6}$ the elementary functions have the following values and condition numbers, respectively:

- $\cos x$: 0.9999999999995, 1.0000000000003332e-12

- $1 - \cos x$: 5.000444502911705e-13, 1.9998222146405398e6

- $\sin x$: 9.999999999998333e-7, 0.9999999999996667

- $f(x)$: 5.000444502912538e-7, 1.9998222146398732e6

3.3 Part C

Repeat Part B for $g(10^{-6})$, which has six elementary steps.

The function g consists of six elementary steps:

- cx

- $\sin x$

- x^p

- cx

- $\sin x$

- cx

6×3 DataFrame

Row	Function	Value	Condition Number
	LaTeXStr...	Float64	Float64
1	$x/2$	5.0e-7	1.0
2	$\\sin\{x/2\}$	5.0e-7	1.0
3	$\\sin\{2\}\{x/2\}$	2.5e-13	2.0
4	$2 \\sin\{2\}\{x/2\}$	5.0e-13	2.0
5	$\\sin\{x\}$	1.0e-6	1.0
6	$g(x)$	5.0e-7	2.0

The code to produce the table above is listed in Program 6.

Answer 3C

For $g(x)$ where $x = 10^{-6}$ the elementary functions have the following values and condition numbers, respectively:

- $x/2$: 5.0e-7, 1.0
- $\sin x/2$: 4.999999999999791e-7, 0.9999999999999166
- $\sin^2 x/2$: 2.4999999999997914e-13, 1.9999999999998332
- $2\sin^2 x/2$: 4.999999999999583e-13, 1.9999999999998332
- $\sin x$: 9.999999999998333e-7, 0.9999999999996667
- $g(x)$: 5.000000000000416e-7, 1.9999999999991667

3.4 Part D

Based on your answers to Part B & C, is $f(10^{-6})$ or $g(10^{-6})$ more accurate (provide your answer as Markdown text in your Jupyter notebook)?

Answer 3D

The function $g(x)$ is more accurate than $f(x)$ for $x = 10^{-6}$ since $f(x)$ produces large errors when computing $1 - \cos x$.

---

4 Floating Point Arithmetic

Create an example calculation to demonstrate that floating point addition is not associative. Repeat the exercise for floating point multiplication.

Consider the following:

$$a = 1.0 \qquad b = 1 \times 10^{-16} \qquad c = 1 \times 10^{16}$$

```
(a + b) + c = 0.0
```

```
a + (b + c) = 1.0
```

The code to produce the result abover is provided in Program 7.

Consider the following:

$$a = 123456789.0 \qquad b = 1.00000001 \qquad c = 0.00000001$$

```
(a * b) * c = 1.2345679023456788
a * (b * c) = 1.234567902345679
```

The code to produce the result above is listed in Program 8.

> **Answer 4**
> The results from Program 7 and Program 8 show that floating point addition and multiplication is not associative.

Original article

This article is available online at the following URL: https://jacob-cunningham-ds.github.io/me2060/hw1-code-cunningham

References

T. A. Driscoll and R. J. Braun. *Fundamentals of Numerical Computation*. 2018.

```
# Packages used
import Plots as plt
import LaTeXStrings as ltx

# One -sided finite difference approximation
function fh(x::Real, h::Vector{<:Real}, f::Function)
    return (f.(x .+ h) . - f.(x)) ./ h
end

# Second order central difference approximation
function fch(x::Real, h::Vector{<:Real}, f::Function)
    return (f.(x .+ h) . - f.(x . - h)) ./ (2 * h)
end

# Total error (note rounding error implicitly included)
function eps(f_exact::Real, f_num::Vector{<:Real})
        return abs.(f_exact . - f_num)
end

# Fixed x and same h array
x = 1.0
n = 1:15
h = 10.0 .^ -n

# Numerical derivatives
fh_sin = fh(x, h, sin)
fch_sin = fch(x, h, sin)

# Exact derivative
f_deriv_sin = cos(x)

# Total errors
eps_fh_sin = eps(f_deriv_sin, fh_sin)
eps_fhc_sin = eps(f_deriv_sin, fch_sin)

# Plots
plt.plot(h, eps_fh_sin,
    yaxis=:log10, xaxis=:log10,
    label="Forward Difference",
    lc=:black, lw=2, ls=:solid)

plt.plot!(h, eps_fhc_sin,
    label="Central Difference",
    lc=:black, lw=2, ls=:dash)

plt.plot!(legend=:bottomleft)
plt.title!(ltx.L"Numerical Error $\epsilon (h)$ vs. Step Size $h$")
plt.xlabel!(ltx.L"\log h")
plt.ylabel!(ltx.L"\log\epsilon (h)")
```

Figure 1: f(x) central differences scheme forward difference scheme approximation error.

```
# New function to evaluate
g(x::Real) = sin(100*x)

# Numerical derivatives
fh_g = fh(x, h, g)
fch_g = fch(x, h, g)

# Exact derivative
g_deriv = 100 * cos(x)

# Total errors
_fh_g = (g_deriv, fh_g)
_fch_g = (g_deriv, fch_g)

# Plots
plt.plot!(h, _fh_g,
    label=ltx.L"g_{h}",
    lc=:red, lw=2, ls=:solid)

plt.plot!(h, _fch_g,
    label=ltx.L"g^{c}_{h}",
    lc=:red, lw=2, ls=:dash)
```

Figure 2: f(x) and g(x) central differences scheme forward difference scheme approximation error.

```
# Packages used
import DataFrames as dfs

# Generic forward error function
function _f(f_num::Function, f_exact::Function, x::Real)
    return f_num(x) - f_exact(x)
end

# Generic backward error function
function _b(x_num::Real, x_exact::Real)
    return x_num - x_exact
end

# Approx x solver for first order approximation of sin(x) = x
function x_num_1(x::Real)
    return asin(x)
end

# Given exact x values
x_exact = [0.1, 0.5, 1.0]

# Given first order approximation
f_num(x::Real) = x

# Initialize lists
x_list = []
f_x_list = []
f_x_approx_list = []
x_approx_list = []
err_f_list = []
err_b_list = []

# Loop over the given exact x values to find forward and backward error
for x in x_exact
    # Update lists
    push!(x_list, x)
    push!(x_approx_list, x_num_1(x))
    push!(f_x_list, sin(x))
    push!(f_x_approx_list, f_num(x))
    push!(err_f_list ,_f(f_num, sin, x))
    push!(err_b_list, _b(x_num_1(x), x))
end

# Make a dataframe for presentation purposes
data = Dict(
    :"x" => round.(Vector{Float64}(x_list), digits=2),
    :"f(x)" => round.(Vector{Float64}(f_x_list), digits=6),
    :"Approx f(x)" => round.(Vector{Float64}(f_x_approx_list), digits=6),
    :"Approx x" => round.(Vector{Float64}(x_approx_list), digits=6),
    :"Forward Error" => round.(Vector{Float64}(err_f_list), digits=6),
    :"Backward Error" => round.(Vector{Float64}(err_b_list), digits=6)
    )

df_part_a = dfs.DataFrame(data)
df_part_a = dfs.select(df_part_a, [:"x", :"Approx x", :"f(x)",
        :"Approx f(x)", :"Forward Error", :"Backward Error"])
dfs.display(df_part_a)
```

Figure 3: Forward and backward error for $f(x) = \sin x$, $\tilde{f}(x) = x$, and $x = 0.1,\ 0.5,\ 1.0$.

```
# Approx x solver for first order approximation of sin(x) = x
function x_num_2(x::Real)
    return asin(x - (x^3 / factorial(3)))
end

# Given exact x values
x_exact = [0.1, 0.5, 1.0]

# Given second order approximation
f_num(x::Real) = x - (x^3 / factorial(3))

# Initialize lists
x_list = []
f_x_list = []
f_x_approx_list = []
x_approx_list = []
err_f_list = []
err_b_list = []

# Loop over the given exact x values to find forward and backward error
for x in x_exact
    # Update lists
    push!(x_list, x)
    push!(x_approx_list, x_num_2(x))
    push!(f_x_list, sin(x))
    push!(f_x_approx_list, f_num(x))
    push!(err_f_list ,_f(f_num, sin, x))
    push!(err_b_list, _b(x_num_2(x), x))
end

# Make a dataframe for presentation purposes
data = Dict(
    :"x" => round.(Vector{Float64}(x_list), digits=2),
    :"f(x)" => round.(Vector{Float64}(f_x_list), digits=6),
    :"Approx f(x)" => round.(Vector{Float64}(f_x_approx_list), digits=6),
    :"Approx x" => round.(Vector{Float64}(x_approx_list), digits=6),
    :"Forward Error" => round.(Vector{Float64}(err_f_list), digits=6),
    :"Backward Error" => round.(Vector{Float64}(err_b_list), digits=6)
    )

df_part_b = dfs.DataFrame(data)
df_part_b = dfs.select(df_part_b, [:"x", :"Approx x", :"f(x)",
        :"Approx f(x)", :"Forward Error", :"Backward Error"])
dfs.display(df_part_b)
```

Figure 4: Forward and backward error for $f(x) = \sin x$, $\tilde{f}(x) = x + x^3/3!$, and $x = 0.1,\ 0.5,\ 1.0$.

```
# Packages used
import LaTeXStrings as ltx
import DataFrames as dfs

# Given x
x = 1e -6

# Elementary function values
u1 = cos(x)
u2 = 1 - u1
u3 = sin(x)
u4 = u2 / u3

# Elementary function condition numbers
k_u1 = abs(x * tan(x))
k_u2 = abs(x) / abs(1 - cos(x))
k_u3 = abs(x * cot(x))
k_u4 = k_u2 * k_u3

# Make a dataframe for presentation purposes
data = Dict(
    :"Function" => [ltx.L"\cos{10^{ -6}}", ltx.L"1 - \cos{10^{ -6}}",
        ltx.L"\sin{10^{ -6}}", ltx.L"(1 -\cos{x})/\sin{x}"],
    :"Value" => [u1, u2, u3, u4],
    :"Condition Number" => [k_u1, k_u2, k_u3, k_u4]
    )

df_p3b = dfs.DataFrame(data)
df_p3b = dfs.select(df_p3b, [:"Function", :"Value", :"Condition Number"])
dfs.display(df_p3b)
```

Figure 5: Elementary function values and condition numbers for $f(x)$ where $x = 10^{-6}$.

```
# Packages used
import LaTeXStrings as ltx
import DataFrames as dfs

# Given x
x = 1e -6

# Elementary function values
u1 = x / 2
u2 = sin(u1)
u3 = u2^2
u4 = 2 * u3
u5 = sin(x)
u6 = u4 / u5

# Elementary function condition numbers
k_u1 = 1
k_u2 = abs(u1 * cot(u1))
k_u3 = 2 * k_u2
k_u4 = k_u3
k_u5 = abs(x * cot(x))
k_u6 = k_u3 * k_u5

# Make a dataframe for presentation purposes
data = Dict(
    :"Function" => [ltx.L"x/2", ltx.L"\sin{x/2}",
        ltx.L"\sin^{2}{x/2}", ltx.L"2 \sin^{2}{x/2}",
        ltx.L"\sin{x}", ltx.L"f(x)"],
    :"Value" => [u1, u2, u3, u4, u5, u6],
    :"Condition Number" => [k_u1, k_u2, k_u3, k_u4, k_u5, k_u6]
    )

df_p3c = dfs.DataFrame(data)
df_p3c = dfs.select(df_p3c, [:"Function", :"Value", :"Condition Number"])
dfs.display(df_p3c)
```

Figure 6: Elementary function values and condition numbers for $g(x)$ where $x = 10^{-6}$.

```
# Define the numbers
a = 1.0
b = 1e -10
c = 1e10

# Compute (a + b) + c
result1 = (a + b) + c

# Compute a + (b + c)
result2 = a + (b + c)

# Print the results
println("(a + b) + c = ", result1)
println("a + (b + c) = ", result2)
```

Figure 7: Example that floating point addition is non-associative.

```
# Define the numbers
a = 123456789.0
b = 1.00000001
c = 0.00000001

# Compute (a * b) * c
result1 = (a * b) * c

# Compute a * (b * c)
result2 = a * (b * c)

# Print the results
println("(a * b) * c = ", result1)
println("a * (b * c) = ", result2)
```

Figure 8: Example that floating point multiplication is non-associative.