HACETTEPE UNIVERSITY
ENGINEERING FACULTY
ELECTRICAL AND ELECTRONICS ENGINEERING PROGRAM

2023-2024
SPRING SEMESTER

ELE708
NUMERICAL METHODS IN ELECTRICAL ENGINEERING

HW1

N23239410 – Ali Bölücü

1) Exercises

1.a) 1.6

1.6. The sin function is given by the infinite series

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

a-) What are the forward and backward error if $\sin(x) = x$ for $x = 0.1$, 0.5 and 1?

Forward Error

$\rightarrow \hat{y} = \hat{f}(x) = x$

$\rightarrow \Delta y = \hat{y} - y = \hat{f}(x) - f(x) = \underline{x - \sin(x)}$

for $x = 0,1$

$= x - \sin(x)$

$= 0,1 - \sin(0,1)$

$= 0,1 - 0,0998$

$= \underline{0,0002}$

for $x = 0,5$

$= 0,5 - \sin(0,5)$

$= 0,5 - 0,4794$

$= \underline{0,00206}$

for $x = 1$

$= 1 - \sin(1)$

$= 1 - 0,8415$

$= \underline{0,01585}$

Backward Error

$\rightarrow \Delta x = \hat{x} - x \longrightarrow \hat{x} = \arcsin(\hat{f}(x)) = \arcsin(\hat{y})$

$\Delta x = \sin^{-1}(\hat{y}) - x$

for $x = 0,1$

$= \sin^{-1}(0,1) - 0,1$

$= 0,100167 - 0,1$

$= \underline{0,000167}$

for $x = 0,5$

$= \sin^{-1}(0,5) - 0,5$

$= 0,523599 - 0,5$

$= \underline{0,023599}$

for $x = 1$

$= \sin^{-1}(1) - 1$

$= 1,5708 - 1$

$= \underline{0,5708}$

---

b-) what are the forward and backward error if $\sin(x) = x - \frac{x^3}{6}$ for $0,1$, $0,5$ and 1?

forward error

$\Rightarrow \hat{y} = \hat{f}(x) = x - \frac{x^3}{6}$ $\quad \Rightarrow \Delta y = x - \frac{x^3}{6} - \sin(x)$

for $x = 0,1$ $= 0,1 - \frac{(0,1)^3}{6} - \sin(0,1)$

$= 0,1 - 0,000166 - 0,099833$

$= 0,000001 = 1 \times 10^{-6}$

for $x = 0,5$ $= 0,5 - \frac{(0,5)^3}{6} - \sin(0,5)$

$= 0,5 - 0,020833 - 0,479426$

$= 0,000259$

for $x = 1$ $= 1 - 0,166667 - 0,841470$

$= 0,008137$

backward error

$\Rightarrow \Delta x = \hat{x} - x \quad \Rightarrow \sin^{-1}\left(\left(x - \frac{x^3}{6}\right)\right) - x$

for $x = 0,1$ $= \sin^{-1}(0,1 - 0,000166) - 0,1$

$= 0,10000058 - 0,1$

$= 0,00000058$

for $x = 0,5$ $= \sin^{-1}(0,5 - 0,020833) - 0,5$

$= 0,499705 - 0,5$

$= (-0,000295)$

for $x = 1$ $= \sin^{-1}(1 - 0,166667) - 1$

$= 0,98511 - 1$

$= (-0,014889)$

1.b) 1.9

1.9

a-) Using 4 digit decimal arithmetic and the formula given $A = 4\pi r^2$, compute the surface area of Earth, with $r = 6370$ km

$A = 4 \cdot \pi \cdot r^2$

→ $r \approx 6370$, emprical measurements error

→ is infinite, truncated error

→ computer arithmetic (rounding) error

→ The earth is not perfectly round, modelling error

$A \approx 4 \cdot 3{,}141 \cdot (6370)^2 = 509808171{,}6 \ km^2$

$= 0{,}5098081716 \cdot 10^9 \ km^2$

b-) Using the same formula, increase radius by 1 km ?

$A = 4 \cdot 3{,}141 \cdot (6371)^2 = 509968249{,}5 \ km^2$

$= 0{,}509968249 5 \cdot 10^9 km^2$

Difference $= 509968249{,}5 - 509808171{,}6 = 160077{,}9$

c-) Since $\frac{dA}{dr} = 8\pi r$, change in the surface area is app. by $8\pi rh$, $(h = \Delta r)$. Compute the difference
How does the value obtained using this approximate formula compare with that obtained from part b

$\frac{dA}{dr} = 8\pi \cdot r \Rightarrow \Delta A = 8 \cdot 3{,}141 \cdot 6370 \cdot 1 = 160065{,}36 \ km^2$

Compare ⇒ diff $= 160077{,}9 - 160065{,}36 = +12{,}54$

$\frac{+12{,}54}{160065{,}36} \cdot 100 = 7{,}83 \cdot 10^{-3} \%$

d-) Determine which of previous two computation ~~correct~~ more nearly correct by repeating both computation ~~repeating~~ using 6-digit decimal aritmethic

a-) $A = 4 \cdot 3{,}14159 \cdot (6370)^2 = 509903933{,}1 \ km^2$

b-) $A = 4 \cdot 3{,}14159 \cdot (6371)^2 = 510064041{,}1 \ km^2$

diff $= 160107{,}9768 \ km^2$

$\frac{dA}{dr} \Rightarrow \Delta A = 8 \cdot 3{,}14159 \cdot 6370 \cdot 1 = 160095{,}4264 \ km^2$

Compare ⇒ diff $= 160107{,}9768 - 160095{,}4264$

$= 12{,}5504$

e-) Explain your results

(a) For part a, we observed that there are many kind of computation error.
Thus our answer is not accurate.

(b) For part b, we observed that little error in computation cause bigger change on output.

(c) For part c, we used better approach to minimize the error.

(d) For part d, we observed using more digit can give more precise and accurate answer. But whe comparing previous answer, the answer in part (a) was more accurate then part(b) due to computational errors.

1.c) 1.12

1.12

a-) Which of the two mathematically equivalent expression can be evaluated more accurately in floating-point arithmetic?

$$(x^2 - y^2) \quad \text{and} \quad ((x-y) \cdot (x+y))$$

ans = $(x-y) \cdot (x+y)$

why = when calculating square first, it will have more digit, so rounding error will be higher but in the $(x-y) \cdot (x+y)$, the substraction will produce relatively small number before the multiplication.

b-) For what values of x and y, relative to each other, is there a substantial difference in the accuracy of the two expressions?

To see this, we will say $y = x + e$

$$\Rightarrow x^2 - y^2 \Rightarrow x^2 - (x+e)^2 = x^2 - (x^2 + 2xe + e^2) = -2xe - e^2$$

$$\Rightarrow (x-y) \cdot (x+y) \Rightarrow (x - (x+e)) \cdot (x+x+e) \Rightarrow (-e) \cdot (2x+e) = -2xe - e^2$$

So when $x \approx y$, the square of difference(e^2) will have an impact on floating point but since the difference is in the mantissa, there is not substantial difference.

2) Computer Problems

2.a) 1.1 Write a program to compute the absolute and relative errors in Stirling's approximation

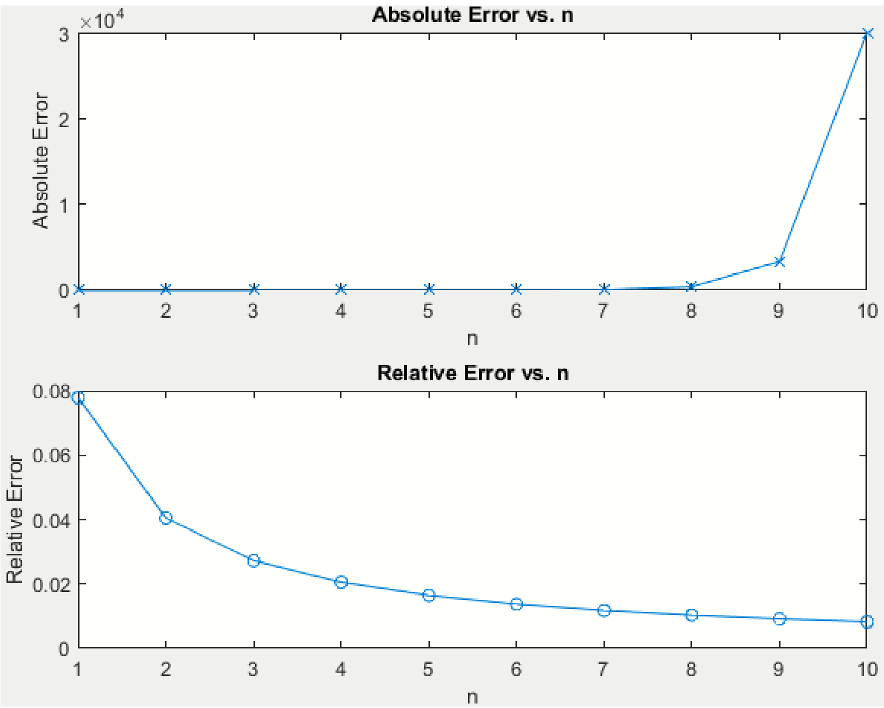
$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

for n = 1, . . . , 10. Does the absolute error grow or shrink as n increases? Does the relative error grow or shrink as n increases?

Absolute error grows and relative error shrinks.

Command Window

```
>> CP1_1
For n = 1
Approximate value: 0.92214
True value: 1
Absolute Error: 0.077863
Relative Error: 0.077863
-----------------------------------
For n = 2
Approximate value: 1.919
True value: 2
Absolute Error: 0.080996
Relative Error: 0.040498
-----------------------------------
For n = 3
Approximate value: 5.8362
True value: 6
Absolute Error: 0.16379
Relative Error: 0.027298
-----------------------------------
For n = 4
Approximate value: 23.5062
True value: 24
Absolute Error: 0.49382
Relative Error: 0.020576
-----------------------------------
For n = 5
Approximate value: 118.0192
True value: 120
Absolute Error: 1.9808
Relative Error: 0.016507
-----------------------------------
For n = 6
Approximate value: 710.0782
True value: 720
Absolute Error: 9.9218
Relative Error: 0.01378
-----------------------------------
For n = 7
Approximate value: 4980.3958
True value: 5040
Absolute Error: 59.6042
Relative Error: 0.011826
-----------------------------------
For n = 8
Approximate value: 39902.3955
True value: 40320
Absolute Error: 417.6045
Relative Error: 0.010357
-----------------------------------
For n = 9
Approximate value: 359536.8728
True value: 362880
Absolute Error: 3343.1272
Relative Error: 0.0092128
-----------------------------------
For n = 10
Approximate value: 3598695.6187
True value: 3628800
Absolute Error: 30104.3813
Relative Error: 0.008296
-----------------------------------
```



2.b) 1.4 Write a program to compute the mathematical constant e, the base of natural logarithms, from the definition

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

Specifically, compute (1 + 1/n)^n for n = 10^k, k = 1, 2, . . . , 20. Determine the error in your successive approximations by comparing them with the value of exp(1). Does the error always decrease as n increases? Explain your results.

The relative error decreases untill the 15th iteration. After the 15th iteration the values of (1/n)^n gets very small, causing floating point arithmetic to round it to zero. Consequently $\left(1+\frac{1}{n}\right)^n$ act as $(1+0)^n$.

```
>> CP1_4
For k = 1
For n = 10
Approximate value: 2.5937
True value: 2.7183
Absolute Error: 0.12454
Relative Error: 0.045815
--------------------------------------------
For k = 2
For n = 100
Approximate value: 2.7048
True value: 2.7183
Absolute Error: 0.013468
Relative Error: 0.0049546
--------------------------------------------
For k = 3
For n = 1000
Approximate value: 2.7169
True value: 2.7183
Absolute Error: 0.0013579
Relative Error: 0.00049954
--------------------------------------------
For k = 4
For n = 10000
Approximate value: 2.7181
True value: 2.7183
Absolute Error: 0.0001359
Relative Error: 4.9995e-05
--------------------------------------------
For k = 5
For n = 100000
Approximate value: 2.7183
True value: 2.7183
Absolute Error: 1.3591e-05
Relative Error: 4.9999e-06
--------------------------------------------
For k = 6
For n = 1000000
Approximate value: 2.7183
True value: 2.7183
Absolute Error: 1.3594e-06
Relative Error: 5.0008e-07
--------------------------------------------
For k = 7
For n = 10000000
Approximate value: 2.7183
True value: 2.7183
Absolute Error: 1.3433e-07
Relative Error: 4.9416e-08
--------------------------------------------
For k = 8
For n = 100000000
Approximate value: 2.7183
True value: 2.7183
Absolute Error: 3.0112e-08
Relative Error: 1.1077e-08
--------------------------------------------
For k = 9
For n = 1000000000
Approximate value: 2.7183
True value: 2.7183
Absolute Error: 2.2355e-07
Relative Error: 8.224e-08
--------------------------------------------
For k = 10
For n = 10000000000
Approximate value: 2.7183
True value: 2.7183
Absolute Error: 2.2478e-07
Relative Error: 8.269e-08
--------------------------------------------
For k = 11
For n = 100000000000
Approximate value: 2.7183
True value: 2.7183
Absolute Error: 2.249e-07
Relative Error: 8.2735e-08
--------------------------------------------
For k = 12
For n = 1000000000000
Approximate value: 2.7185
True value: 2.7183
Absolute Error: 0.00024167
Relative Error: 8.8905e-05
--------------------------------------------
For k = 13
For n = 10000000000000
Approximate value: 2.7161
True value: 2.7183
Absolute Error: 0.0021718
Relative Error: 0.00079896
--------------------------------------------
For k = 14
For n = 100000000000000
Approximate value: 2.7161
True value: 2.7183
Absolute Error: 0.0021718
Relative Error: 0.00079896
--------------------------------------------
For k = 15
For n = 1000000000000000
Approximate value: 3.035
True value: 2.7183
Absolute Error: 0.31675
Relative Error: 0.11653
--------------------------------------------
For k = 16
For n = 1e+16
Approximate value: 1
True value: 2.7183
Absolute Error: 1.7183
Relative Error: 0.63212
--------------------------------------------
For k = 17
For n = 1e+17
Approximate value: 1
True value: 2.7183
Absolute Error: 1.7183
Relative Error: 0.63212
--------------------------------------------
For k = 18
For n = 1e+18
Approximate value: 1
True value: 2.7183
Absolute Error: 1.7183
Relative Error: 0.63212
--------------------------------------------
For k = 19
For n = 1e+19
Approximate value: 1
True value: 2.7183
Absolute Error: 1.7183
Relative Error: 0.63212
--------------------------------------------
For k = 20
For n = 1e+20
Approximate value: 1
True value: 2.7183
Absolute Error: 1.7183
Relative Error: 0.63212
--------------------------------------------
>> é
```

2.c) 1.9 (a) Write a program to compute the exponential function ex using the infinite series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

(b) Summing in the natural order, what stopping criterion should you use?

(c) Test your program for x = ±1, ±5, ±10, ±15, ±20, and compare your results with the built-in function exp(x).

(d) Can you use the series in this form to obtain accurate results for x < 0? (Hint: e−x = 1/ex.)
(e) Can you rearrange the series or regroup the terms in any way to obtain more accurate results for x < 0?

(a)

```
function CP1_4(exponent)

    x = exponent;
    n = 1;
    Approximate_value = 1;
    True_value = exp(x);

    while abs(x^n) < 1e300 && factorial(n) < 1e300

        Approximate_value = (x^n/factorial(n)) + Approximate_value;
        n = n+1;
    end

        Absolute_error = abs(Approximate_value - True_value);
        Relative_error = Absolute_error / True_value;

        disp(['Approximate value: ', num2str(Approximate_value)]);
        disp(['True value: ', num2str(True_value)]);
        disp(['Absolute Error: ', num2str(Absolute_error)]);
        disp(['Relative Error: ', num2str(Relative_error)]);
        disp('--------------------------------------------');
end
```

(b) The value of x^n or factorial becomes infinite after the 1e300, so it used as stopping criterion.

(c)

```
>> CP1_9(1)
Approximate value: 2.7183
True value: 2.7183
Absolute Error: 4.4409e-16
Relative Error: 1.6337e-16
--------------------------------------------
>> CP1_9(5)
Approximate value: 148.4132
True value: 148.4132
Absolute Error: 2.8422e-14
Relative Error: 1.915e-16
--------------------------------------------
>> CP1_9(10)
Approximate value: 22026.4658
True value: 22026.4658
Absolute Error: 7.276e-12
Relative Error: 3.3033e-16
--------------------------------------------
>> CP1_9(15)
Approximate value: 3269017.3725
True value: 3269017.3725
Absolute Error: 0
Relative Error: 0
--------------------------------------------
>> CP1_9(20)
Approximate value: 485165195.4098
True value: 485165195.4098
Absolute Error: 1.1921e-07
Relative Error: 2.4571e-16
--------------------------------------------
```

```
>> CP1_9(-1)
Approximate value: 0.36788
True value: 0.36788
Absolute Error: 1.1102e-16
Relative Error: 3.0179e-16
--------------------------------------------
>> CP1_9(-5)
Approximate value: 0.0067379
True value: 0.0067379
Absolute Error: 1.4398e-15
Relative Error: 2.1369e-13
--------------------------------------------
>> CP1_9(-10)
Approximate value: 4.54e-05
True value: 4.54e-05
Absolute Error: 3.2843e-13
Relative Error: 7.2342e-09
--------------------------------------------
>> CP1_9(-15)
Approximate value: 3.0591e-07
True value: 3.059e-07
Absolute Error: 3.1672e-12
Relative Error: 1.0354e-05
--------------------------------------------
>> CP1_9(-20)
Approximate value: 4.1736e-09
True value: 2.0612e-09
Absolute Error: 2.1125e-09
Relative Error: 1.0249
--------------------------------------------
```

(d) As it can be seen by comparing previous result. Absolute error decreased. Better results were achieved for the negative number by taking 1/ans at the end.

```
function CP1_4(exponent)

    if exponent < 0
        x = abs(exponent);
        True_value = 1/exp(x);

    else
        x = exponent;
        True_value = exp(x);

    end

    n = 1;
    Approximate_value = 1;


    while abs(x^n) < 1e300 && factorial(n) < 1e300
        Approximate_value = (x^n/factorial(n)) + Approximate_value;
        n = n+1;
    end

    Approximate_value = 1/Approximate_value;


        Absolute_error = abs(Approximate_value - True_value);
        Relative_error = Absolute_error / True_value;

        disp(['Approximate value: ', num2str(Approximate_value)]);
        disp(['True value: ', num2str(True_value)]);
        disp(['Absolute Error: ', num2str(Absolute_error)]);
        disp(['Relative Error: ', num2str(Relative_error)]);
        disp('-------------------------------------------');
end
```

```
--------------------------------------------
>> CP1_9v2(-1)
Approximate value: 0.36788
True value: 0.36788
Absolute Error: 5.5511e-17
Relative Error: 1.5089e-16
--------------------------------------------
>> CP1_9v2(-5)
Approximate value: 0.0067379
True value: 0.0067379
Absolute Error: 1.7347e-18
Relative Error: 2.5746e-16
--------------------------------------------
>> CP1_9v2(-10)
Approximate value: 4.54e-05
True value: 4.54e-05
Absolute Error: 1.3553e-20
Relative Error: 2.9851e-16
--------------------------------------------
>> CP1_9v2(-15)
Approximate value: 3.059e-07
True value: 3.059e-07
Absolute Error: 0
Relative Error: 0
--------------------------------------------
>> CP1_9v2(-20)
Approximate value: 2.0612e-09
True value: 2.0612e-09
Absolute Error: 4.1359e-25
Relative Error: 2.0066e-16
--------------------------------------------
```

2.d) Codes for 1.1 and 1.4

2.d.i) 1.1

Editor - D:\Ders\M.Sc\M.Sc\ELE708 - Numerical Methods in Electrical Engineering\HW1\CP1_1.m

CP1_1.m

```matlab
function CP1_1()

    range = 10;
    Absolute_Errors = zeros(1,range);
    Relative_Errors = zeros(1,range);

    for n = 1:range

        True_value = factorial(n);
        Stirling_approximation = sqrt(2*pi*n) * (n/exp(1))^n;

        Absolute_error = abs(Stirling_approximation - True_value);
        Relative_error = Absolute_error / True_value;

        Absolute_Errors(n) = Absolute_error;
        Relative_Errors(n) = Relative_error;

        disp(['For n = ', num2str(n)]);
        disp(['Approximate value: ', num2str(Stirling_approximation)]);
        disp(['True value: ', num2str(True_value)]);
        disp(['Absolute Error: ', num2str(Absolute_error)]);
        disp(['Relative Error: ', num2str(Relative_error)]);
        disp('--------------------------------------------');
    end

    figure;
    subplot(2, 1, 1);
    plot(1:range, Absolute_Errors,'-x');
    title('Absolute Error vs. n');
    xlabel('n');
    ylabel('Absolute Error');

    subplot(2, 1, 2);
    plot(1:range, Relative_Errors, '-o');
    title('Relative Error vs. n');
    xlabel('n');
    ylabel('Relative Error');


end


```

2.d.ii) 1.4

Editor - D:\Ders\M.Sc\M.Sc\ELE708 - Numerical Methods in Electrical Engineering\HW1\CP1_4.m

CP1_4.m

```
% Write a program to compute the mathematical constant e, the base of natural logarithms, from the definition
% e = lim n→∞ (1 + 1/n)^n.
% Specifically, compute (1 + 1/n)n for n = 10k, k = 1, 2,..., 20.
% Determine the error in your successive approximations by comparing them with the value of exp(1).
% Does the error always decrease as n increases?

function CP1_4()

    range = 20;

    for k = 1:range

        n = 10^k;

        True_value = exp(1);
        Approximate_value = (1+1/n)^n;

        Absolute_error = abs(Approximate_value - True_value);
        Relative_error = Absolute_error / True_value;

        disp(['For k = ', num2str(k)]);
        disp(['For n = ', num2str(n)]);
        disp(['Approximate value: ', num2str(Approximate_value)]);
        disp(['True value: ', num2str(True_value)]);
        disp(['Absolute Error: ', num2str(Absolute_error)]);
        disp(['Relative Error: ', num2str(Relative_error)]);
        disp('---------------------------------------------');
    end
end
```