Homework 1

Lendel Deguia

Problem 1.1.1

• Let $v_1 = 22/7$ and $v_2 = 355/113$; let $\alpha_1 = |\pi - v_1|$ and let $\alpha_2 = |\pi - v_2|$

Then, $\alpha_1 \approx 1.26 \times 10^{-4}$ and $\alpha_2 \approx 2.67 \times 10^{-7}$.

Thus, $\alpha_2 < \alpha_1$ and we see that $v_2 = 355/113$ is a better approximation for π in terms of **absolute error**.

Now, let $\epsilon_1 = \alpha_1/\pi$ and $\epsilon_2 = \alpha_2/\pi$

Then, $\epsilon_1 \approx 4.02 \times 10^{-4}$ and $\epsilon_2 \approx 8.49 \times 10^{-8}$.

Thus, $\epsilon_2 < \epsilon_1$ and we see that $v_2 = 355/113$ is a better approximation for π in terms of **relative error**.

• Note that if $|\pi - \frac{n}{m}| = 10^{-10}$ for some $n, m \in \mathbb{Q}$, then of course, $|\pi - \frac{n}{m}| < 10^{-9}$. Let $y = 69$ and let $x = y(\pi - 10^{-10})$.

Using Matlab for a precision of sixteen digits, we find that $x = 216.7698930907957$.

Now, let $n = (10^{13})x$ and $m = (10^{13})y$; i.e. $n = 2167698930907957$ and $m = 69 \times 10^{13}$.

Let $q = n/m$. Note that, $n \in \mathbb{Z}$ and $m \in \mathbb{Z}$, therefore, $q \in \mathbb{Q}$. Moreover, using a prime number checker, we find that $n = 2167698930907957$ is prime, so q is in simplest terms.

Using a high precision calculator (https://keisan.casio.com/calculator), we find that

$$\left|\pi - \frac{2167698930907957}{69 \times 10^{13}}\right| \approx 1.000005 \times 10^{-10} < 10^{-9}$$

Therefore, $\boxed{q = \frac{2167698930907957}{69 \times 10^{13}}}$ is a valid simple rational fraction

Problem 1.1.5

part b

$\sum_{j=1}^{n} \sum_{i=1}^{n} a_{ij}$

```
sum ← 0
for j = 1 to n
    for i = 1 to n
        sum ← sum + a_ij
    end for
end for
```

part c

$\sum_{i=1}^{n} \left(\sum_{j=1}^{i} a_{ij} + \sum_{j=1}^{i-1} a_{ji} \right)$

```
sum ← 0
for i = 1 to n
    for j = 1 to i
        sum ← sum + a_ij
    end for
    for j = 1 to i − 1
        sum ← sum + a_ji
    end for
end for
```

part e

$\sum_{k=2}^{2n} \sum_{i+j=k}^{n} a_{ij}$

```
sum ← 0
for k = 1 to 2n
    for i = 1 to n
        for j = 1 to n
            if i + j = k then
                sum ← sum + a_ij
            end if
        end for
    end for
end for
```

Problem 1.1.9

We can use nested multiplication so that $y = 5e^{3x} + 7e^{2x} + 9e^x + 11$ becomes:

$$y = 11 + e^x(9 + e^x(7 + 5e^x)).$$

Thus, we can use a nested multiplication algorithm for our pseudo code:

integer i, n; **real** p, x
real array $(a_i)_{0:3} \leftarrow (11, 9, 7, 5)$
$p \leftarrow a_3$
for i = 3 to 0
    $p \leftarrow a_i + \exp(x)p$
end for

Problem 1.1.12

part a

$$\boxed{v = a_0 + \left(\sum_{i=1}^{n} a_i\right)x}$$

part c

$$\boxed{v = \sum_{i=0}^{n} a_i x^{n-i}}$$

part d

$$\boxed{v = \sum_{i=0}^{n} a_i x^i \ ; \ z = \prod_{i=0}^{n} x}$$

Computer Excercise 1.1.1

The following program will evaluate the finite difference quotient of $\sin(x)$ for different values of h as an approximation for the first derivative of sine; the program will loop through different values of h multiplying the previous value of h by 0.25 on each iteration. The program will then evaluate the error of the approximation by using sine's known derivative, cosine, and will seek out and store a smallest error value and its corresponding index.

On each iteration, desired values will be stored in an array which will then be casted as a table for display.

```
%Initialize values

n=30;
x=0.5;
h = 1;
emin = 1;

%Initialize output arrays which will then be used to display a table
%This portion is for display purposes

i_out = (1:30)';
h_out = zeros(30,1);
y_out = zeros(30,1);
error_out = zeros(30,1);

%approximation program loop for sin(x)

for i = 1:n
    h = 0.25*h;
    y = (sin(x+h) - sin(x))/h;
    error = abs( cos(x) - y );

    %update the output arrays
    h_out(i) = h;
    y_out(i) = y;
    error_out(i) = error;

    %seek out minimum error and record its index
    if error < emin
        emin = error;
        imin = i;
    end
end



%cast output arrays as a table

T = table(i_out, h_out, y_out, error_out);
disp(T)
```

i_out	h_out	y_out	error_out
1	0.25	0.808852885676524	0.0687296762138483
2	0.0625	0.862034158909074	0.0155484029812989
3	0.015625	0.873801417582083	0.0037811443082898
4	0.00390625	0.876643953269792	0.000938608620581149
5	0.0009765625	0.877348327919719	0.000234233970653364
6	0.000244140625	0.87752402954743	5.85323429422857e-05
7	6.103515625e-05	0.877567930439	1.46314513731483e-05
8	1.52587890625e-05	0.877578904128313	3.65776205946133e-06
9	3.814697265625e-06	0.877581647451734	9.14438638588422e-07
10	9.5367431640625e-07	0.877582333283499	2.28606873875492e-07
11	2.38418579101562e-07	0.877582504646853	5.7243520146244e-08
12	5.96046447753906e-08	0.87758254725486	1.46355123575859e-08
13	1.49011611938477e-08	0.877582557499409	4.3909640368156e-09
14	3.72529029846191e-09	0.877582564949989	3.05961656010822e-09
15	9.31322574615479e-10	0.877582550048828	1.18415446337394e-08
16	2.3283064365387e-10	0.877582550048828	1.18415446337394e-08
17	5.82076609134674e-11	0.877582550048828	1.18415446337394e-08
18	1.45519152283669e-11	0.877582550048828	1.18415446337394e-08
19	3.63797880709171e-12	0.877578735351562	3.82653881025874e-06
20	9.09494701772928e-13	0.8775634765625	1.90853278727587e-05
21	2.27373675443232e-13	0.87744140625	0.000141155640372759
22	5.6843418860808e-14	0.8779296875	0.000347125609627241
23	1.4210854715202e-14	0.87890625	0.00132368810962724
24	3.5527136788005e-15	0.875	0.00258256189037276
25	8.88178419700125e-16	0.875	0.00258256189037276
26	2.22044604925031e-16	0.75	0.127582561890373
27	5.55111512312578e-17	0	0.877582561890373
28	1.38777878078145e-17	0	0.877582561890373
29	3.46944695195361e-18	0	0.877582561890373
30	8.67361737988404e-19	0	0.877582561890373

```
%display the minimum error and its index

fprintf('\n The minimum error occurs at i = %d has a value of %d \n', imin, emin)
```

```
 The minimum error occurs at i = 14 has a value of 3.059617e-09
```

From the above table, we see that more iterations does not necessarily correspond to less error. According to the table, the error decreases from i=1 until the "sweet spot" which occurs at i =14 from which it starts to increase for indicies larger than 14. This routine demonstrates an inherent limitation of computers' capabilities to deal with infinities and infinitesimals; theoretically, the error should approach zero as h approaches zero.

Computer Excercise 1.1.2 (mostly the same as 1.1.1)

The following program will evaluate the finite difference quotient of $f(x) = 1/(1+x^2)$ for different values of h as an approximation for the first derivative of f(x); the program will loop through different values of h multiplying the previous value of h by 0.25 on each iteration. The program will then evaluate the error of the approximation by using f(x)'s known derivative, $f'(x) = -2x/(1+x^2)^2$, and will seek out and store a smallest error value and its corresponding index.

side note: $f(x)$ *is a Lorentzian function*

On each iteration, desired values will be stored in an array which will then be casted as a table for display.

```
%Initialize values
n=30;
x=0.5;
h = 1;
emin = 1;

%Initialize output arrays which will then be used to display a table
%This portion is for display purposes

i_out = (1:30)';
h_out = zeros(30,1);
y_out = zeros(30,1);
error_out = zeros(30,1);

%approximation program loop for f(x)

%function and derivative are written at the bottom of the script

for i = 1:n
    h = 0.25*h;
    y = (f(x + h) - f(x))/h;
    error = abs( deriv(x) - y );

    %update the output arrays
    h_out(i) = h;
    y_out(i) = y;
    error_out(i) = error;

    %seek out minimum error and record its index
    if error < emin
        emin = error;
        imin = i;
    end
end

%cast output arrays as a table

T = table(i_out, h_out, y_out, error_out);
disp(T)
```

i_out	h_out	y_out	error_out
1	0.25	-0.64	1.11022302462516e-16
2	0.0625	-0.645697329376855	0.00569732937685485
3	0.015625	-0.64185149469624	0.00185149469624035
4	0.00390625	-0.64049064823493	0.000490648234930391
5	0.0009765625	-0.640124414425145	0.000124414425144992
6	0.000244140625	-0.640031213384646	3.1213384645512e-05
7	6.103515625e-05	-0.64000781021241	7.81021240980895e-06
8	1.52587890625e-05	-0.640001952982857	1.95298285687873e-06
9	3.814697265625e-06	-0.640000488288933	4.88288933397918e-07
10	9.5367431640625e-07	-0.640000122133642	1.22133642421751e-07
11	2.38418579101562e-07	-0.64000003086403	3.08640301094343e-08
12	5.96046447753906e-08	-0.640000008046627	8.04662703135506e-09
13	1.49011611938477e-08	-0.640000008046627	8.04662703135506e-09
14	3.72529029846191e-09	-0.640000015497208	1.54972076282789e-08
15	9.31322574615479e-10	-0.640000104904175	1.04904174791365e-07
16	2.3283064365387e-10	-0.640000343322754	3.43322753892927e-07
17	5.82076609134674e-11	-0.64000129699707	1.29699707029918e-06
18	1.45519152283669e-11	-0.639999389648438	6.10351562513323e-07
19	3.63797880709171e-12	-0.6400146484375	1.46484374999867e-05
20	9.09494701772928e-13	-0.6400146484375	1.46484374999867e-05
21	2.27373675443232e-13	-0.64013671875	0.000136718749999987
22	5.6843418860808e-14	-0.640625	0.000624999999999987
23	1.4210854715202e-14	-0.640625	0.000624999999999987
24	3.5527136788005e-15	-0.65625	0.01625
25	8.88178419700125e-16	-0.75	0.11
26	2.22044604925031e-16	-1	0.36
27	5.55111512312578e-17	0	0.64
28	1.38777878078145e-17	0	0.64
29	3.46944695195361e-18	0	0.64
30	8.67361737988404e-19	0	0.64

```
%display the minimum error and its index

fprintf('\n The minimum error occurs at i = %d has a value of %d \n', imin, emin)
```

```
 The minimum error occurs at i = 1 has a value of 1.110223e-16
```

Remarkable! Our lowest error occurs exactly on the first iteration! Note that:

```
fprintf('The derivative of f(x) at x = 0.5 is: %5.2f', deriv(0.5))
```

```
The derivative of f(x) at x = 0.5 is: -0.64
```

The first derivative of $f(x)$ at $x = 0.5$ is exactly equal to $\frac{f(x+h) - f(x)}{h}$ where $h = 0.25$ (the nonzero error arises instead of zero due to the limitations of floating point arithmetic). At first glance this appears to be related to the mean value theorem, but so called mean value point in this case would be $c = 0.5$ which is also an endpoint, so perhaps this is coincidential.

This function and its derivative are a bit chunky so I'm going to write them separately to make things neater

```
function y=f(x)
    y = 1/(1 + x^2);
end

function y=deriv(x)
    y = -(2*x)/((1 + x^2)^2);
end
```

Computer Excercise 1.1.7

The following program takes as input a 1D array of real numbers and evaluates the arithmetic mean, variance, and standard deviation of the array. I used the 'sum' function to sum over the array elements instead of using a loop.

```
%generate a real 1D array of 100 random floating point values
rng('default') %set random seed to keep output consistent
a=rand(1,100);

%execute the function
prob7func(a)
```

```
The arithmetic mean of the array is: 0.527994
The variance of the array is: 0.088219
and the standard deviation of the array is: 0.297017
```

Here, we have an array $(a_i)_{i:n} = (a_1, \dots, a_n)$ where $n = 100$ such that $\forall i \in \{1, \dots, n\}, a_i \in \mathbb{R}$. Let m be the mean, v be the variance and σ be the standard deviation. Our output was evaluated in the following way:

mean: $m = \frac{1}{n}\sum_{i=1}^{n} a_i$

variance: $v = \frac{1}{n-1}\sum_{i=1}^{n}(a_i - m)^2$

standard deviation: $\sigma = \sqrt{v}$

The function for displaying the mean, variance, and standard deviation is written separately here:

```
function prob7func(a)
    n = length(a);
    %evaluate mean
    m = sum(a)/n ;

    %evaluate variance
    v = sum((a - m).^2)/(n-1);

    %evaluate standard deviation
    sigma = sqrt(v);

    %display values
    fprintf(['\n The arithmetic mean of the array is: %f \n The variance of the array is: ' ...
        '%f \n and the standard deviation of the array is: %f \n'], m, v, sigma)
end
```

Computer Excercise 1.1.21(a)

The following code will analyze the extent to how floating point values are rounded by comparing different display formats.

```
% part a

x=0.1;
y=0.01;
z = x-y;
p = 1.0/3.0;
q = 3.0*p;
u = 7.6;
v = 2.9;
w = u - v;

%display using default format
fprintf('x = %f, y = %f, z = %f, p = %f, q = %f, u = %f, v = %f, w = %f \n',x,y,z,p,q,u,v,w)
```

```
x = 0.100000, y = 0.010000, z = 0.090000, p = 0.333333, q = 1.000000, u = 7.600000, v = 2.900000, w = 4.700000
```

These values appear to be reasonable and consistent with the intended values assigned.

```
%display using extremely large format field
fprintf(['large format field:\n x = %30.20f,\n y = %30.20f,\n z = %30.20f,\n p = %30.20f,\n ' ...
    'q = %30.20f,\n u = %30.20f, \n v = %30.20f,\n w = %30.20f \n'],x,y,z,p,q,u,v,w)
```

```
large format field:
 x =         0.10000000000000000555,
 y =         0.01000000000000000021,
 z =         0.09000000000000001055,
 p =         0.33333333333333331483,
 q =         1.00000000000000000000,
 u =         7.59999999999999964473,
 v =         2.89999999999999991118,
 w =         4.69999999999999928946
```

Well, well well.. the true colors are revealed. Unveiling more decimal places indicates that a lot more is going behind the scenes than what the first set of displayed values would suggest. This demonstrates that we should be very carefuly when working floating point arithmetic, ESPECIALLY if we want lots of precision. For example, something whacky is going on with the last five digits of 'w'; it was supposed to be just a simple subtraction of $7.6 - 2.9 = 4.7$. In fact, something crazy is going on with all of the variables except 'q' (but maybe if we displayed more digits, 'q' might also reveal its true colors).

Computer Excercise 1.1.21(b)

The following code uses different functions that map $n \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \subseteq \mathbb{N}$ onto the real numbers and stores them in x, y, and z.

```
%part b
for n = 1:10
    x = (n-1)/2;
    y = (n^2)/3.0;
```

```
    z = 1.0 + 1/n;
    fprintf('n = %d: x = %f, y = %f, z = %f \n\n',n,x,y,z)
end
```

```
n = 1: x = 0.000000, y = 0.333333, z = 2.000000

n = 2: x = 0.500000, y = 1.333333, z = 1.500000

n = 3: x = 1.000000, y = 3.000000, z = 1.333333

n = 4: x = 1.500000, y = 5.333333, z = 1.250000

n = 5: x = 2.000000, y = 8.333333, z = 1.200000

n = 6: x = 2.500000, y = 12.000000, z = 1.166667

n = 7: x = 3.000000, y = 16.333333, z = 1.142857

n = 8: x = 3.500000, y = 21.333333, z = 1.125000

n = 9: x = 4.000000, y = 27.000000, z = 1.111111

n = 10: x = 4.500000, y = 33.333333, z = 1.100000
```

Analysis of part (b)

Let's try to reduce the amount of floating point digits in the function expressions to see if we get the same results:

```
for n = 1:10
    x = (n-1)/2;
    y = (n^2)/3; %changed 3.0 to 3
    z = 1 + 1/n; %changed 1.0 to 1
    fprintf('n = %d: x = %f, y = %f, z = %f \n\n',n,x,y,z)
end
```

```
n = 1: x = 0.000000, y = 0.333333, z = 2.000000

n = 2: x = 0.500000, y = 1.333333, z = 1.500000

n = 3: x = 1.000000, y = 3.000000, z = 1.333333

n = 4: x = 1.500000, y = 5.333333, z = 1.250000

n = 5: x = 2.000000, y = 8.333333, z = 1.200000

n = 6: x = 2.500000, y = 12.000000, z = 1.166667

n = 7: x = 3.000000, y = 16.333333, z = 1.142857

n = 8: x = 3.500000, y = 21.333333, z = 1.125000

n = 9: x = 4.000000, y = 27.000000, z = 1.111111

n = 10: x = 4.500000, y = 33.333333, z = 1.100000
```

The values displayed here are consistent with the ones above, so at least to the extent of the digits displayed, the values we acquired for these expressions are reliable.