MACM 203 Assignment 1

Table of Contents

Part A.

We wish to compute a row vector of n = 1,000,000 entries, where entry i of the vector equals $\frac{1}{n}\sin\left(\frac{i\pi}{n}\right)$. We apply the following approaches:

A1: The vector **u** is created using a for-loop without pre-initializing the array.

```
n=1000000
```

```
n = 1000000
```

```
tic
u = 0
```

```
u = 0
```

```
for i=1:n
    u(i)=(1/n)*sin((i*pi)/n);
end
toc
```

```
Elapsed time is 0.083368 seconds.
```

A2: The vector **v** is created using a for-loop and it is first pre-initialized using Matlab's zeros command.

```
tic
v = zeros(1,n)
```

```
v = 1×1000000
     0     0     0     0     0     0     0     0     0     0     0     0     0⋯
```

```
for i=1:n
    v(i)=(1/n)*sin((i*pi)/n);
end
toc
```

```
Elapsed time is 0.025530 seconds.
```

A3: The vector **w** is created without using a loop

```
tic
w = 1:1:n;
w = pi * w;
w = w/n;
w = sin(w);
w = (1/n)*w;
```

```
toc
```

```
Elapsed time is 0.007818 seconds.
```

The most efficient method to create the vector is without using a loop. The least efficient method is using a for-loop without pre-initializing the array. Yes, there is a signficant different in the compute time. It is about 10x faster without the loop than it is to use a for-loop without pre-initializing the array.

Part B.

Sum the entries of **w** in two ways:

B1: using a for-loop

```
tic
sum_forloop = 0
```

```
sum_forloop = 0
```

```
for i=1:max(size(w))
    sum_forloop = sum_forloop + w(i);
end
toc
```

```
Elapsed time is 0.011073 seconds.
```

```
sum_forloop
```

```
sum_forloop = 0.6366
```

B2: using Matlab's sum command

```
tic
sum_sum = sum(w)
```

```
sum_sum = 0.6366
```

```
toc
```

```
Elapsed time is 0.001405 seconds.
```

The second method, using Matlab's sum command is most efficient.

Part C.

Your sum in Part B approximates $\frac{2}{\pi}$! Why is this?

The sum in Part B is can be written as $\sum_{i=1}^{n} \frac{1}{n} \sin\left(\frac{i\pi}{n}\right)$, where n = 1,000,000. We notice that when i = 1, $\sin\left(\frac{i\pi}{n}\right)$ is close to 0 and when i = 1,000,000, $\sin\left(\frac{i\pi}{n}\right)$ is 1, which are our bounds. We will show it is related to $\int_0^{\pi} \sin(x)\mathrm{dx} = 2$. If we rewrite this definite integral as the limit of a related Reimann

Sum, where $\Delta x = \frac{b-a}{n} = \frac{\pi - 0}{n} = \frac{\pi}{n}$, $x_i = a + \Delta x_i = 0 + \Delta x_i = \frac{\pi}{n} * i$, and $f(x_i) = \sin(x_i) = \sin\left(\frac{\pi}{n} * i\right)$, we get that $\lim_{n\to\infty} \sum_{i=1}^{n} \Delta x f(x_i) = \lim_{n\to\infty} \sum_{i=1}^{n} \frac{\pi}{n} \sin\left(\frac{i\pi}{n}\right)$. If we let n = 1,000,000, then we get $\sum_{i=1}^{n} \frac{\pi}{n} \sin\left(\frac{i\pi}{n}\right)$, which is only different from our sum by a factor of π. So, $\sum_{i=1}^{n} \frac{1}{n} \sin\left(\frac{i\pi}{n}\right)$ is an approximation of $\int_0^{\pi} \frac{1}{\pi} \sin(x) dx = \frac{2}{\pi}$.

Define the *absolute error* to be the absolute value of the difference between your sum and $\frac{2}{\pi}$.

```
abs_error = abs(sum_sum - 2/pi)
```

```
abs_error = 5.2358e-13
```

Consider the method that is obtained by combining calculations A3 and B2. We wish to understand how varying *n* affects the absolute error.

For large enough *n*, if you double *n* you will see a predictable change in the error. How does the absolute error change when *n* doubles? Experimentally determine (within a factor of 2) the smallest *n* so that the absolute error is less than 1e-4. Justify your answer.

```
n = 1;
c = 20;
error_curr = 1;
error_prev = 1;
for i = 1:c
    i
    n
    % create the vector
    x = 1:1:n;
    x = pi * x;
    x = x/n;
    x = sin(x);
    x = (1/n)*x;
    % sum the vector
    sum_x = sum(x)
    % calculate the error
    error_curr = abs(sum_x - (2/pi))
    error_change_div1 = abs(error_prev/error_curr)
    error_change_div2 = abs(error_curr/error_prev)
    error_change_sub = abs(error_prev - error_curr)
    % save error for next calculation
    error_prev = error_curr;
    % double n for next iteration
    n = 2 * n;
end
```

```
i = 1
n = 1
```

```
sum_x = 1.2246e-16
error_curr = 0.6366
error_change_div1 = 1.5708
error_change_div2 = 0.6366
error_change_sub = 0.3634
i = 2
n = 2
sum_x = 0.5000
error_curr = 0.1366
error_change_div1 = 4.6598
error_change_div2 = 0.2146
error_change_sub = 0.5000
i = 3
n = 4
sum_x = 0.6036
error_curr = 0.0331
error_change_div1 = 4.1317
error_change_div2 = 0.2420
error_change_sub = 0.1036
i = 4
n = 8
sum_x = 0.6284
error_curr = 0.0082
error_change_div1 = 4.0313
error_change_div2 = 0.2481
error_change_sub = 0.0249
i = 5
n = 16
sum_x = 0.6346
error_curr = 0.0020
error_change_div1 = 4.0077
error_change_div2 = 0.2495
error_change_sub = 0.0062
i = 6
n = 32
sum_x = 0.6361
error_curr = 5.1141e-04
error_change_div1 = 4.0019
error_change_div2 = 0.2499
error_change_sub = 0.0015
i = 7
n = 64
sum_x = 0.6365
error_curr = 1.2784e-04
error_change_div1 = 4.0005
error_change_div2 = 0.2500
error_change_sub = 3.8357e-04
i = 8
n = 128
sum_x = 0.6366
error_curr = 3.1958e-05
error_change_div1 = 4.0001
error_change_div2 = 0.2500
error_change_sub = 9.5879e-05
i = 9
n = 256
sum_x = 0.6366
error_curr = 7.9895e-06
error_change_div1 = 4.0000
error_change_div2 = 0.2500
error_change_sub = 2.3969e-05
i = 10
n = 512
sum_x = 0.6366
```

```
error_curr = 1.9974e-06
error_change_div1 = 4.0000
error_change_div2 = 0.2500
error_change_sub = 5.9921e-06
i = 11
n = 1024
sum_x = 0.6366
error_curr = 4.9934e-07
error_change_div1 = 4.0000
error_change_div2 = 0.2500
error_change_sub = 1.4980e-06
i = 12
n = 2048
sum_x = 0.6366
error_curr = 1.2484e-07
error_change_div1 = 4.0000
error_change_div2 = 0.2500
error_change_sub = 3.7451e-07
i = 13
n = 4096
sum_x = 0.6366
error_curr = 3.1209e-08
error_change_div1 = 4.0000
error_change_div2 = 0.2500
error_change_sub = 9.3627e-08
i = 14
n = 8192
sum_x = 0.6366
error_curr = 7.8022e-09
error_change_div1 = 4.0000
error_change_div2 = 0.2500
error_change_sub = 2.3407e-08
i = 15
n = 16384
sum_x = 0.6366
error_curr = 1.9506e-09
error_change_div1 = 4.0000
error_change_div2 = 0.2500
error_change_sub = 5.8517e-09
i = 16
n = 32768
sum_x = 0.6366
error_curr = 4.8764e-10
error_change_div1 = 4.0000
error_change_div2 = 0.2500
error_change_sub = 1.4629e-09
i = 17
n = 65536
sum_x = 0.6366
error_curr = 1.2191e-10
error_change_div1 = 4.0000
error_change_div2 = 0.2500
error_change_sub = 3.6573e-10
i = 18
n = 131072
sum_x = 0.6366
error_curr = 3.0478e-11
error_change_div1 = 4.0000
error_change_div2 = 0.2500
error_change_sub = 9.1432e-11
i = 19
n = 262144
sum_x = 0.6366
error_curr = 7.6195e-12
```

```
error_change_div1 = 4.0000
error_change_div2 = 0.2500
error_change_sub = 2.2858e-11
i = 20
n = 524288
sum_x = 0.6366
error_curr = 1.9049e-12
error_change_div1 = 3.9999
error_change_div2 = 0.2500
error_change_sub = 5.7145e-12
```

As n doubles, the absolute error is about 25% less than the previous error. We determine exprimentally that the smallest n so that the absolute error is less than 1e-4, is when $n = 128$. From the caluclations, when $n = 64$, the abs(sum_x - (2/pi)) = 1.2784e-04, which is larger than the required 1e-4. So we double n and get $n = 128$, then abs(sum_x - (2/pi)) = 3.1958e-05, which satisfies the absolute error being less than 1e-4.

We also see from the output that when $n = 128$, and each subsequent n, that sum_x = 0.6366, so it seems that $n = 128$ has minimized the error sufficiently.