Numerical Method
Purdue University
Spring 2021

Homework 4

Jerry Wang

1 Problem 1

1. The Gauss-Hermite quadrature has the following formula:

$$\int_{-\infty}^{\infty} f(t)e^{-t^2}dt = w_1 f(t_1) + w_2 f(t_2)$$

The weight function for Gauss-Hermite quadrature is $w(t) = e^{-t^2}$ which is an even function. For a 2-point Gauss-Hermite Quadrature:

$$\pi_2(t) = \pi(t; w) = t^2 + bt + a$$

Since $w(t)$ is an even function, the function of $\pi(t)$ has to be even function in order to satisfy the orthogonal requirements. This means the value of b has to be zero.

$$\pi_2(t) = t^2 + a$$

The value of a can be solved using the orthogonality between $w(t)$ and $\pi_2(t)$.

$$\int_{-\infty}^{\infty} w(t)\pi_2(t)dt = 0$$

Since both functions are even functions, the integral can be evaluated as:

$$2\int_{0}^{\infty} w(t)\pi_2(t)dt = 0$$

$$\int_{0}^{\infty} e^{-t^2}(t^2 - a)dt = 0$$

$$\int_{0}^{\infty} e^{-t^2}t^2dt - a\int_{0}^{\infty} e^{-t^2}dt =$$

$$\frac{\sqrt{\pi}}{4} - a\frac{\sqrt{\pi}}{2} = 0$$

$$a = 1/2 \quad \pi_2(t) = t^2 - 1/2$$

The nodes of the quadrature can then be calculated as:

$$t^2 - 1/2 = 0$$

$$t_1 = \frac{1}{\sqrt{2}} \quad t_2 = -\frac{1}{\sqrt{2}}$$

By using $f(x) = 1$ and $f(x) = t$, the values of w_1 and w_2 can then be calculated.

When $f(x) = 1$:

$$\int_{-\infty}^{\infty} e^{-t^2} dt = \sqrt{\pi} = w_1 + w_2$$

When $f(x) = t$:

$$\int_{-\infty}^{\infty} te^{-t^2} dt = 0 = \frac{1}{\sqrt{2}} w_1 - \frac{1}{\sqrt{2}} w_2$$

The system of equation can then be solved as:

$$w_1 = \frac{\sqrt{\pi}}{2} \quad w_2 = \frac{\sqrt{\pi}}{2}$$

The Gauss-Hermite quadrature is:

$$\int_{-\infty}^{\infty} f(t)e^{-t^2} dt = w_1 f(t_1) + w_2 f(t_2)$$

where:

$$t_1 = \frac{1}{\sqrt{2}} \quad t_2 = -\frac{1}{\sqrt{2}}$$

$$w_1 = \frac{\sqrt{\pi}}{2} \quad w_2 = \frac{\sqrt{\pi}}{2}$$

2. By substitute $\gamma = -0.5$ and $1 + e^x$, the function $f(x)$ which will be evaluated in the integral becomes:

$$f(x) = 2(1 + e^x)^{1/2}$$

To evaluate the integral

$$\int_{-\infty}^{\infty} f(x)e^{-(x-\mu)^2/(2\sigma^2)} dx$$

a change of variable is needed

$$t = \frac{x - \mu}{\sqrt{2}\sigma}$$

$$x = \sqrt{2}\sigma t + \mu$$

$$dx = \sqrt{2}\sigma dt$$

The integral then becomes

$$\int_{-\infty}^{\infty} \sqrt{2}\sigma f(\sqrt{2}\sigma t + \mu)e^{-t^2} dt$$

which can be evaluated using Gauss-Hermite Quadrature.

The following code compute the Gauss-Hermite quadrature for the above integral.

```
using LinearAlgebra
using Plots
using DelimitedFiles

function getGaussQuadNodesWeights(alpha, beta, n)
    J = zeros((n,n))
    for i in 1:n
        if i == 1
            J[i, [1,2]] = [alpha[i], sqrt(beta[i+1])]
        elseif i == n
            J[i, [n-1, n]] = [sqrt(beta[n]), alpha[n]]
        else
            J[i, i-1:i+1] = [sqrt(beta[i]), alpha[i], sqrt(beta[i+1])]
        end
    end
    # println(J)
    F = eigen(J)
    t = F.values
    v = F.vectors
    w = beta[1] .* (v[1, :] .^ 2)
    return t, w

end

function gauss_hermite(f, n)
    alpha = zeros((n,))
    beta = convert.(Float64, collect(1:n))
    beta = (beta .- 1) ./ 2.0
    beta[1] = sqrt(pi)
    t, w = getGaussQuadNodesWeights(alpha, beta, n)
    # println(t)
    # println(w)


    # calculate the quadrature
    return sum(w .* f(t) )
end

n_end = 30
results = zeros((n_end-1, ))
mu = 0.15
sigma = 0.25
f(x) = sqrt(2) .* sigma .* 2 .*
        (1 .+ exp.(sqrt(2) .* sigma .* x .+ mu)) .^ 0.5
exact = gauss_hermite(f, 1000)
n = 2:n_end
for (i, n) in enumerate(n)
    # get Gaussian-Hermite Quadrature Nodes and Weights
```

```
        println(n)
        results[i] = gauss_hermite(f, n)
        println(results[i])
        println()
    end
    @show results
    @show error = abs.(results .- exact)
    display(plot(collect(2:n_end), results,
                 title="Gaussian-Hermite Quadrature Estimation",
                 xlabel="n",
                 ylabel="Estimation"))
    png("GH_result.png")
    display(plot(collect(n), error,
                 seriestype= :scatter,
                 yscale=:log10,
                 ylim=[10^-15, 10^0],
                 xlabel="n",
                 ylabel="Error",
                 title="Error of Quadrature",
                 legend=false))
    png("GH_error.png")

    writedlm("error.csv", error)
```

The error of the Gauss-Hermite Quadrature with respect to number of nodes is shown below.

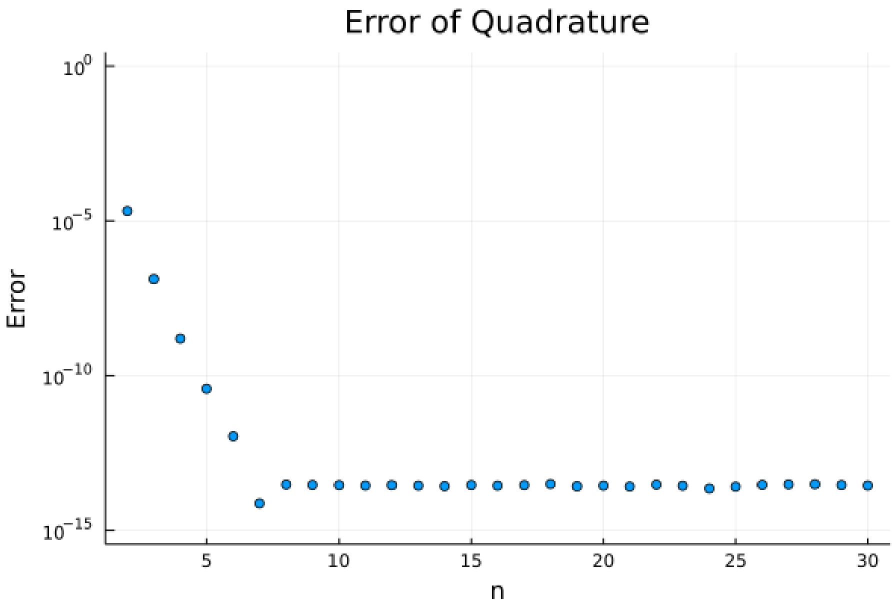

Figure 1: The error of Gauss-Hermite Quadrature with respect to number of nodes

From the plot above, using 7 nodes Gauss-Hermite Quadrature is enough to calculate the expected utility. When $n = 7$, the result of the quadrature reaches machine percision for *Float64*

2 Problem 2

1. The code for calculate the multivariate quadrature is shown below:

```
using LinearAlgebra

function getGaussQuadNodesWeights(alpha, beta, n)
    J = zeros((n,n))
    for i in 1:n
        if i == 1
            J[i, [1,2]] = [alpha[i], sqrt(beta[i+1])]
        elseif i == n
            J[i, [n-1, n]] = [sqrt(beta[n]), alpha[n]]
        else
            J[i, i-1:i+1] = [sqrt(beta[i]), alpha[i], sqrt(beta[i+1])]
        end
    end
    # println(J)
    F = eigen(J)
    t = F.values
    v = F.vectors
    w = beta[1] .* (v[1, :] .^ 2)
    return t, w
end

function multivar_quad(f, Nx, Ny)
    # generate the nodes and weights in x and y direction
    beta = convert.(Float64, collect(1:Nx))
    beta = 1 ./ (4 .- (1 ./ (beta .- 1) .^ 2))
    beta[1] = 2.
    tx, wx = getGaussQuadNodesWeights(zeros((Nx,)), beta, Nx)

    beta = convert.(Float64, collect(1:Ny))
    beta = 1 ./ (4 .- (1 ./ (beta .- 1) .^ 2))
    beta[1] = 2.
    ty, wy = getGaussQuadNodesWeights(zeros((Ny,)), beta, Ny)

    # calculate gauss-legendre quadrature of the function
    result = 0.
    for i in 1:Nx
        for j in 1:Ny
            result += f(tx[i], ty[j]) * wx[i] * wy[j]
        end
    end
    return result
end
```

2. The estimation of the integral for the following functions are:

$$f(x,y) = x^2 + y^2 \qquad \text{Estimation} = 2.6666$$
$$f(x,y) = x^2y^2 \qquad \text{Estimation} = 0.2666$$
$$f(x,y) = e^{x^2+y^2} \qquad \text{Estimation} = 8.5574$$
$$f(x,y) = (1-x^2) + 100 * (y-x^2)^2 \qquad \text{Estimation} = 216$$

3. The investigation of finding the relation between the number of nodes and the order of polynomial that can be integrated exactly starts with the first two functions in the previous question.

$$f(x,y) = x^2 + y^2 \quad f(x,y) = x^2y^2$$

By trying out various N_x and N_y value for those two function with various power. The results are shown in the table below

N_x	N_y	Estimation
2	2	1.7777
3	2	2.1333
2	3	1.7777
3	3	2.1333

Table 1: $f(x,y) = x^4 + y^2$, Exact Value = 2.1333

N_x	N_y	Estimation
2	2	0.1481
3	2	0.2666
2	3	0.1481
3	3	0.2666

Table 2: $f(x,y) = x^4y^2$, Exact Value = 0.2666

N_x	N_y	Estimation
2	2	0.0493
3	2	0.0888
2	3	0.0888
3	3	0.16

Table 3: $f(x,y) = x^4y^4$, Exact Value = 0.2666

From the evidence shown above, it suggest that 2D Gauss-Legendre quadrature will exactly integrate two-dimensional functions $f(x,y)$ when the largest degree of x is less than $2N_x - 1$ and the largest degree of y is less than $2N_y - 1$.

3 Problem 3

1. The exact value of the interals are:

   (a) $\int_{-1}^{1} x^{20} = \frac{2}{21}$

   (b) $\int_{-1}^{1} e^{-x^2} \approx 1.49365$

   (c) $\int_{-1}^{1} e^{-1/(x^2)} \approx 0.178148$

   (d) $\int_{-1}^{1} e^{x} = e - \frac{1}{e}$

   (e) $\int_{-1}^{1} 1/(1+16x^2) = \frac{1}{2}\tan^{-1}(4)$

   (f) $\int_{-1}^{1} |x|^3 = 0.5$

2. The eigenvalue of the Jacobi matrix for Gauss-Legendre quadrature is

$$\begin{array}{ccc} -0.9641030614981564, & -0.8715507067630738, & -0.7299809463525079, \\ -0.5491369709621342, & -0.3403381248683587, & -0.11521287286174864, \\ 0.1152128728617492, & 0.340338124868359, & 0.5491369709621349, \\ 0.7299809463525082, & 0.8715507067630734, & 0.9641030614981563 \end{array}$$

   The following code is used to calculate the eigenvalue of the Gauss-Legendre quadrature.

```
using LinearAlgebra

function gauss_legendre_nodes(n)
    beta = 0.5 .* sqrt.(1 .- (2 .* collect(1:n)) .^ (-2))
    T = diagm(1 => beta, -1 => beta)
    F = eigen(T)
    x = F.values
    w = 2 .* F.vectors[1,:] .^ 2
    return x, w
end
```

3. The code for the Clenshaw-Curtis quadrature is shown below:

```
using FFTW

function clenshaw_curtis(f, n)
    #generate chebyshev points
    x = cos.(pi .* collect(0:n) ./ n)
    #calculate function value at each chebyshev points
    fx = f(x) ./ (2 .* n)
    #take the fourier transform of the calculated function values
    g = real(fft(vcat(fx[1:n+1], reverse(fx[2:n]))))
    #get the chebyshev coefficients from the fft result
    a = vcat(vcat(g[1], g[2:n] .+ reverse(g[n+2:2*n])), g[n+1])
    #calcualte the weight of each chebyshev coefficient
    w = 0 .* a
    w[range(1, stop=n, step=2)] = 2 ./
```

```
                        (1 .- collect(range(0, stop=n, step=2)) .^ 2)
    return sum(w .* a)
end
```

4. The code for Gauss-Legendre quadrature is shown below:

```
using LinearAlgebra

function gauss_legendre_nodes(n)
    beta = 0.5 .* sqrt.(1 .- (2 .* collect(1:n)) .^ (-2))
    T = diagm(1 => beta, -1 => beta)
    F = eigen(T)
    x = F.values
    w = 2 .* F.vectors[1,:] .^ 2
    return x, w
end

function gauss_legendre_quad(f, n)
    x, w = gauss_legendre_nodes(n)
    return sum(f(x) .* w)
end
```

5. The variance of the using the Monte-Carlo method is shown below:

function	$n = 100$	$n = 1000$
$f(x) = x^{20}$	0.0002172	2.39236e-5
$f(x) = e^{-x^2}$	0.0003920	3.80709e-5
$f(x) = e^{-1/(x^2)}$	0.0001333	1.41831e-5
$f(x) = e^x$	0.0042723	0.0004208
$f(x) = 1/(1 + 16x^2)$	0.0008507	8.31995e-5
$f(x) = \|x\|^3$	0.0008031	8.77430e-5

Table 4: Variance of the result of Monte-Carlo Quadrature over 1000 iterations

6. The convergence of all 6 functions are shown below

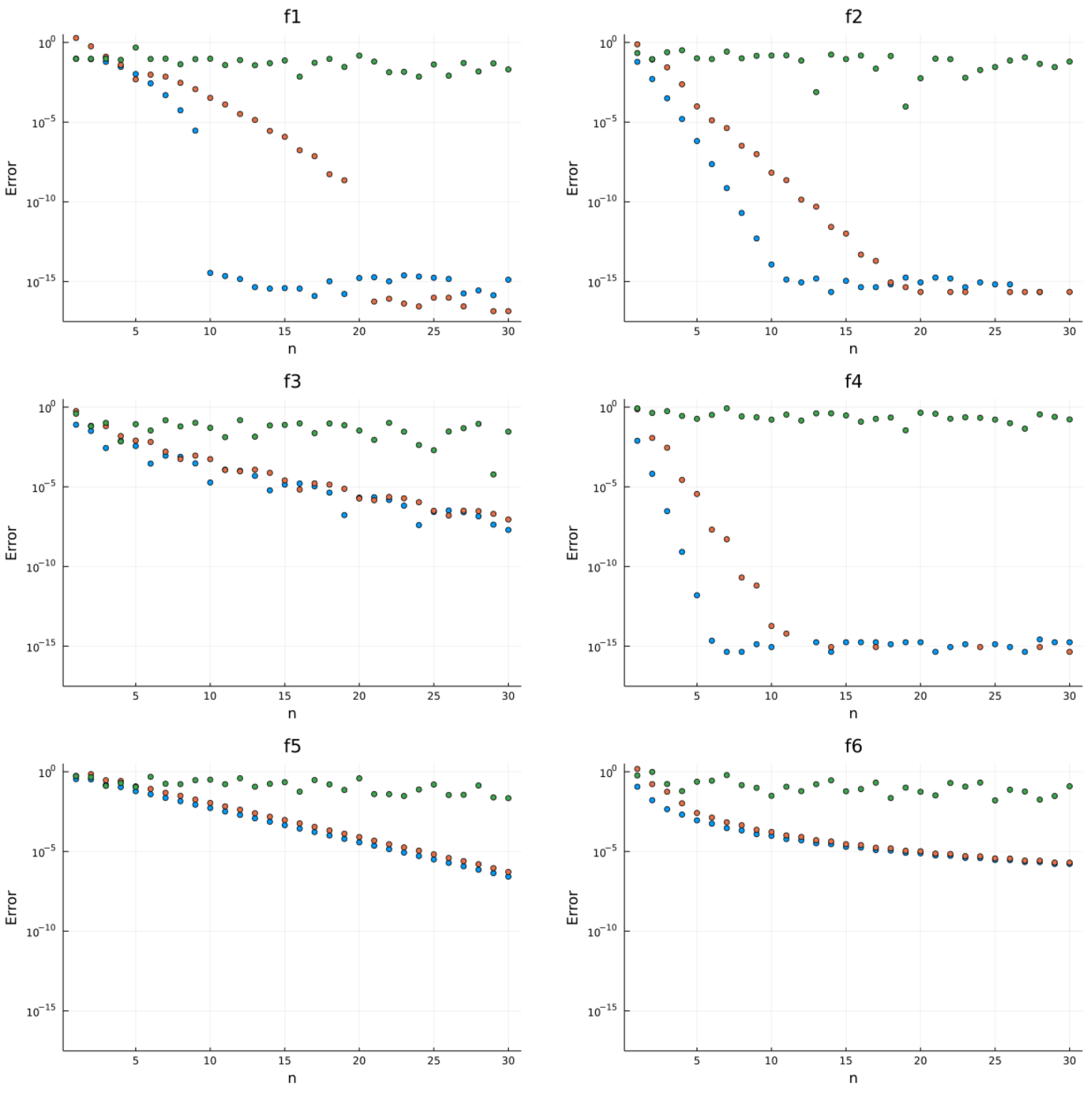

Figure 2: Convergence Plot for all 6 functions. The blue dots are for Gaussian-Legendre Quadrature. The Orange dots are for Clenshaw-Curtis Quadrature. The green dots are for Monte-Carlo Quadrature

7. The time complexity of the Gaussian-Legendre Quadrature is compared with the worst case of selection sort algorithm. Both algorithm has a time complexity of $O(n^2)$. The result of the time complexity is shown below:

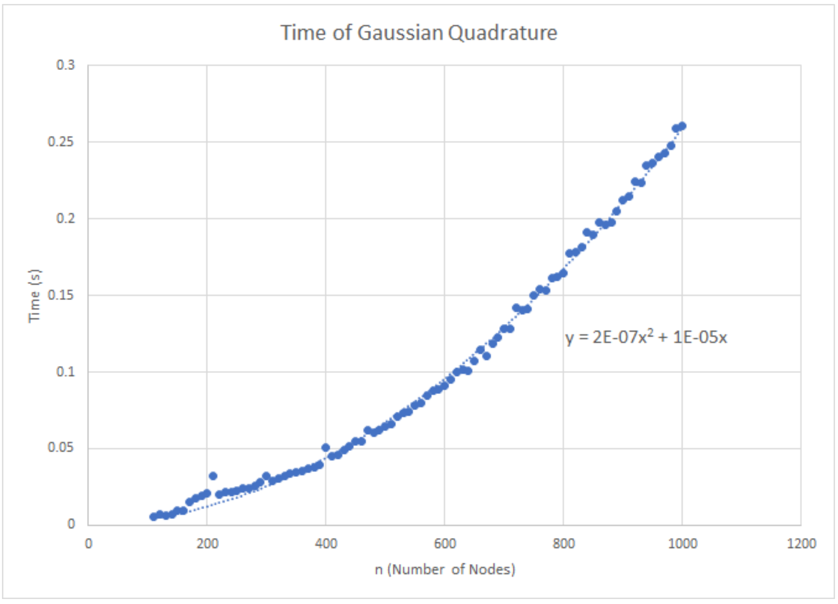

Figure 3: Time complexity of Gaussian Quadrature

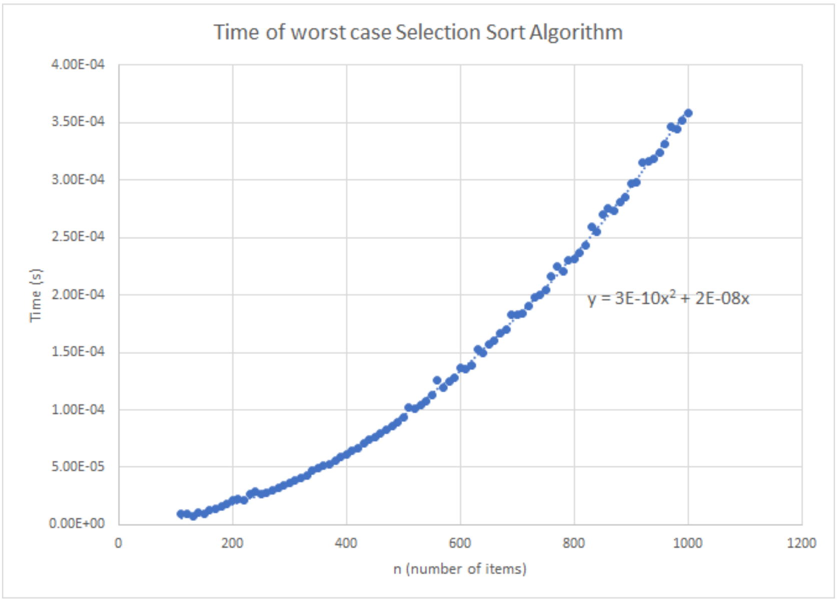

Figure 4: Time complexity of Worst Case Selection Sort

From the fitted trend line of both algorithm, I can conclude that the final scaling constant in term of the runtime of Selection Sort Algorithm is about 650. This means that the Gaussian Quadrature runs 650 times slower than selection sort algorithm with the same amount of input.

4 Problem 4

1. $v_n = n^{10}$

$$\frac{v_{n+1}}{v_n} = \frac{(n+1)^{-10}}{n^{-10}} = \left(\frac{n}{n+1}\right)^{10}$$

$$\lim_{n\longrightarrow\infty}\left(\frac{n}{n+1}\right)^{10} = 1$$

This is a sublinear sequence.

2. $w_n = 10^{-n}$

$$\frac{w_{n+1}}{w_n} = \frac{10^{-(n+1)}}{10^{-n}} = \frac{10^{-n}\cdot 10^{-1}}{10^{-n}} = \frac{1}{10}$$

This is a linear sequence.

3. $x_n = 10^{-n^2}$

$$\frac{x_{n+1}}{x_n} = \frac{10^{-(n+1)^2}}{10^{-n^2}} = \frac{10^{-n^2}\cdot 10^{-2n}\cdot 10^{-1}}{10^{-n^2}} = \frac{1}{10^{2n+1}}$$

$$\lim_{n\longrightarrow\infty}\frac{1}{10^{2n+1}} = 0$$

This is a superlinear sequence.

4. $y_n = n^{10}3^{-n}$

$$\frac{y_{n+1}}{y_n} = \frac{(n+1)^{10}\cdot 3^{-(n+1)}}{n^{10}\cdot 3^{-n}} = \left(\frac{n+1}{n}\right)^{10}\cdot\frac{1}{3}$$

$$\lim_{n\longrightarrow\infty}\left(\frac{n+1}{n}\right)^{10}\cdot\frac{1}{3} = \frac{1}{3}$$

This is a linear sequence

5. $z_n = 10^{-3\cdot 2^n}$

$$\frac{z_{n+1}}{z_n^2} = \frac{10^{-3\cdot 2^{n+1}}}{(10^{-3\cdot 2^n})^2} = \frac{10^{-3\cdot 2^n\cdot 2}}{10^{-3\cdot 2^n\cdot 2}} = 1$$

This is a quadratic sequence.

5 Problem 5

1. To compute the value of $\sqrt{x}$ numerically, the root of the following function is the result of $\sqrt{x}$.

   $$f(x) = x^2 - N$$

   where N is the value that needs to be square root.

   In the following code, the method of false position is used to numerically calculate the value of the square root of an given value. The function only uses addition, subtraction, multiplication, and division to compute the result.

```
function square_root(value::BigFloat)
    f(y) = y .^ 2 - value
    a = value
    b = 0
    x = 0
    n = 0
    while abs(f(x)) > eps(Float64)
        x = a - (a - b) / (f(a) - f(b)) * f(a)
        if f(x) > 0 && f(a) > 0
            a = x
        else
            b = x
        end
        n += 1
    end
    return x
end
```

2. Comparing to function above and the Julia *sqrt* function, there is are difference between result when tolerance of the function above is set to the machine percision of *Float64*. For a *Float64* percision, the result of the function above is only accuracte to 17 significant digits.

Problem 0

I have worked on this homework on my own.