Numerical Method
Purdue University
Spring 2021

Homework 1

Jerry Wang

1 Problem 1

1. The discrete L_∞ approximate of f is $c = \frac{y+1}{2}$. This allows the L_∞ norm to be minized. If the value of c changes, the difference between c and 1 or c and y will be larger.

2. In order to find the L_2 approximation of function f using constant c, we need to minimize the square of L_2 norm of the difference between the function f and c. The squared of L_2 norm of the difference is:

$$\begin{aligned} E^2 = ||f - c||_2^2 &= \sum_{i=1}^{N} |f(t_i) - c|^2 \\ &= (1-c)^2(N-1) + (y-c)^2 \end{aligned}$$

To calculate the value of c while keeping the L_2 norm at minimum, take the derivative of the L_2 with respect to c and set the result expression to zero.

$$\frac{d(E^2)}{dc} = -2(1-c)(N-1) - 2(y-c) = 0$$

$$-2(1-c)(N-1) - 2(y-c) = 0$$

$$(1-c)(N-1) + (y-c) = 0$$

$$(N - Nc - 1 + c) + (y - c) = 0$$

$$-Nc = 1 - N - y$$

$$c = \frac{y-1}{N} + 1$$

3. As $N \to \infty$, the value of c that uses L_∞ approximation does not change, since it is not depending on the value of N. In the case of using L_2 approximation, the value of c will approach 1 as $N \to \infty$ since the fraction will approach zero.

2 Problem 2

1. The normal equation for the least squares problem is

$$\sum_{j=1}^{n}(\pi_i,\pi_j)c_j = (\pi_i, f)$$

   In the problem, the equation $e^x \approx 1 + cx$ can be reordered as $e^x - 1 \approx cx$. From the reordered equation, $f(x) = e^x - 1$ and $\pi_1 = x$. The normal equation can then be solved as:

$$\begin{aligned}
(\pi_1(x),\pi_1(x))c &= (\pi_1, f)\\
\left(\int_0^1 x^2dx\right)c &= \int_0^1 x(e^x-1)dx\\
\frac{1}{3}c &= x(e^x-x)|_0^1 - \int_0^1 (e^x-x)dx\\
\frac{1}{3}c &= (e-1) - \left(e^x - \tfrac{x^2}{2}\Big|_1^0\right)\\
\frac{1}{3}c &= \frac{1}{2}\\
c &= \frac{3}{2}
\end{aligned}$$

2. The error curve for $e_1(x) = e^x - (1+x)$ and $e_2(x) = e^x - (1+cx)$, where $c = \frac{3}{2}$ is shown below.

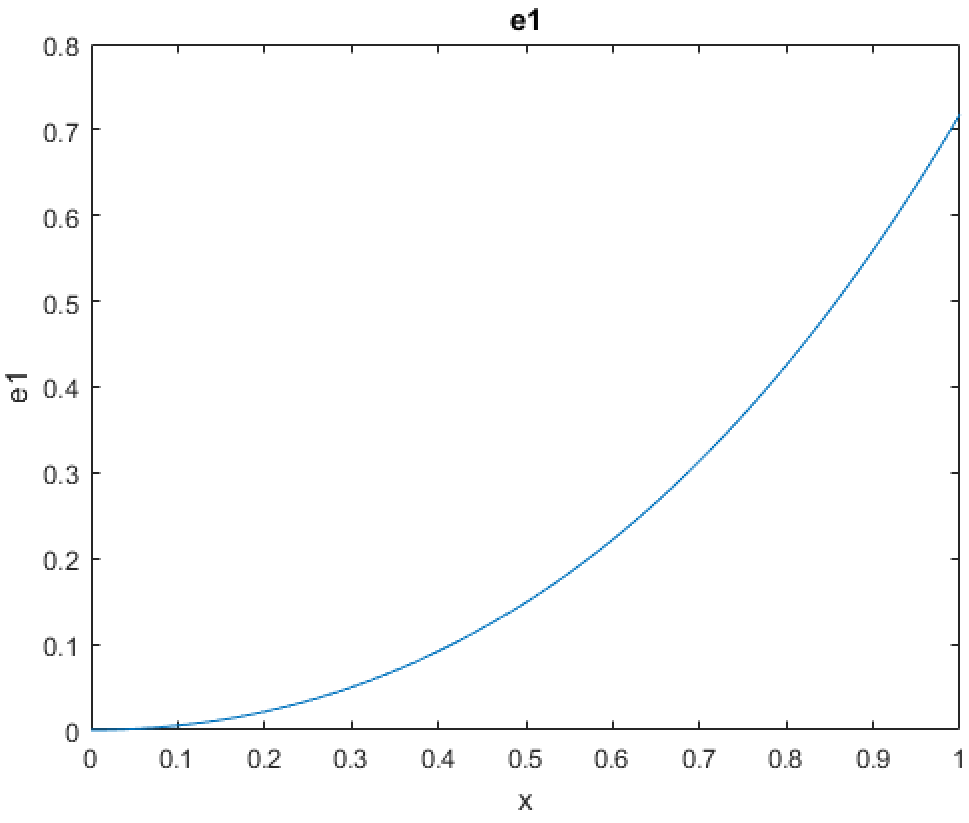

Figure 1: Error curve of e_1 (max error: 0.7183)

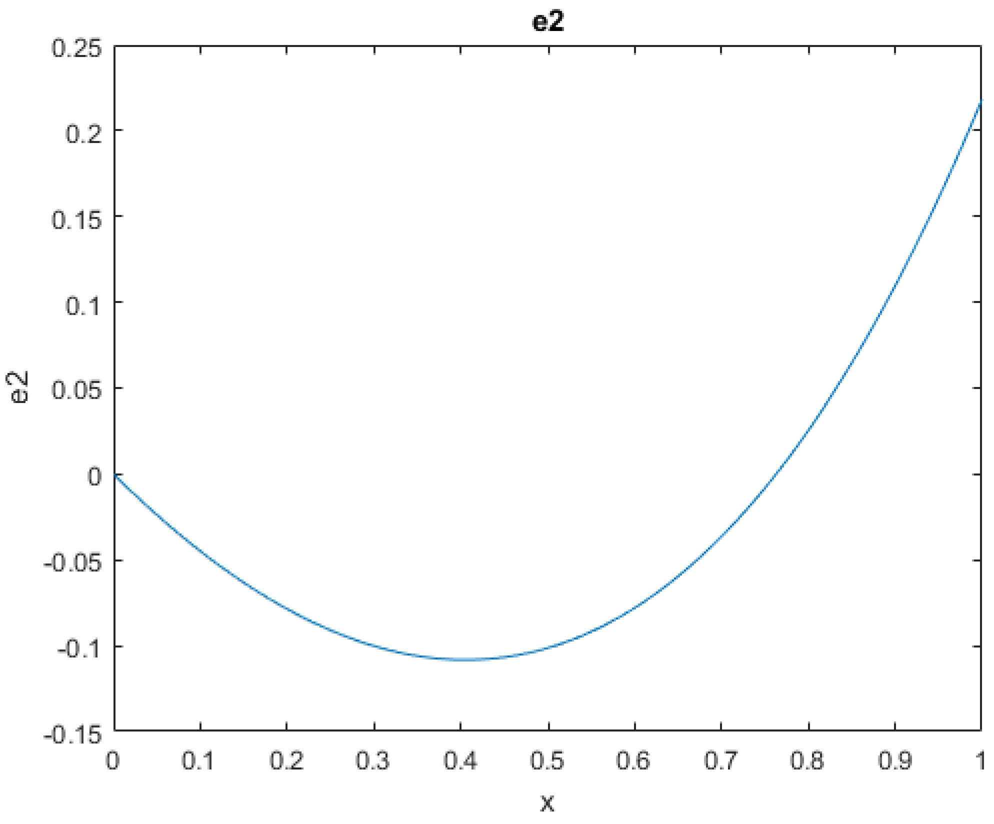

Figure 2: Error curve of e_2 (max error: 0.2183)

3. The equation $e^x \approx 1 + cx + dx^2$ can be rearranged as $e^2 - 1 \approx 1 + c_1x + c_2x^2$. From the equation, $f(x) = e^x - 1$, $\pi_1 = x$, and $\pi_2 = x^2$. The normal equations are:

$$(\pi_1, \pi_1)c_1 + (\pi_1, \pi_2)c_2 = (\pi_1, f)$$
$$(\pi_2, \pi_1)c_1 + (\pi_2, \pi_2)c_2 = (\pi_2, f)$$

The individual items can be solved as:

$$\begin{aligned}
(\pi_1,\pi_1) &= \int_0^1 x^2 dx = \frac{1}{3}\\
(\pi_1,\pi_2) = (\pi_2,\pi_1) &= \int_0^1 x^3 dx = \frac{1}{4}\\
(\pi_2,\pi_2) &= \int_0^1 x^4 dx = \frac{1}{5}\\
(\pi_1,f) &= \frac{1}{2} \text{ (from previous problem)}\\
(\pi_2,f) &= \int_0^1 x^2(e^x-1)dx\\
&= \int_0^1 x^2e^x dx - \int_0^1 x^2 dx\\
&= e - \int_0^1 2xe^x dx - \frac{1}{3}\\
&= e - 2 - \frac{1}{3}\\
&= e - \frac{7}{3}
\end{aligned}$$

Plug in all the coefficients of the normal equation. The system of linear equations are:

$$\begin{aligned}
\frac{1}{3}c_1 + \frac{1}{4}c_2 &= \frac{1}{2}\\
\frac{1}{4}c_1 + \frac{1}{5}c_2 &= e - \frac{7}{3}
\end{aligned}$$

The solution for c_1 and c_2 are:

$$\begin{aligned}
c_1 &= 0.903090292457279\\
c_2 &= 0.795879610056964
\end{aligned}$$

The error curve for $e_1(x) = e^x - (1+x)$ and $e_2(x) = e^x - (1 + c_1x + c_2x^2)$ are:

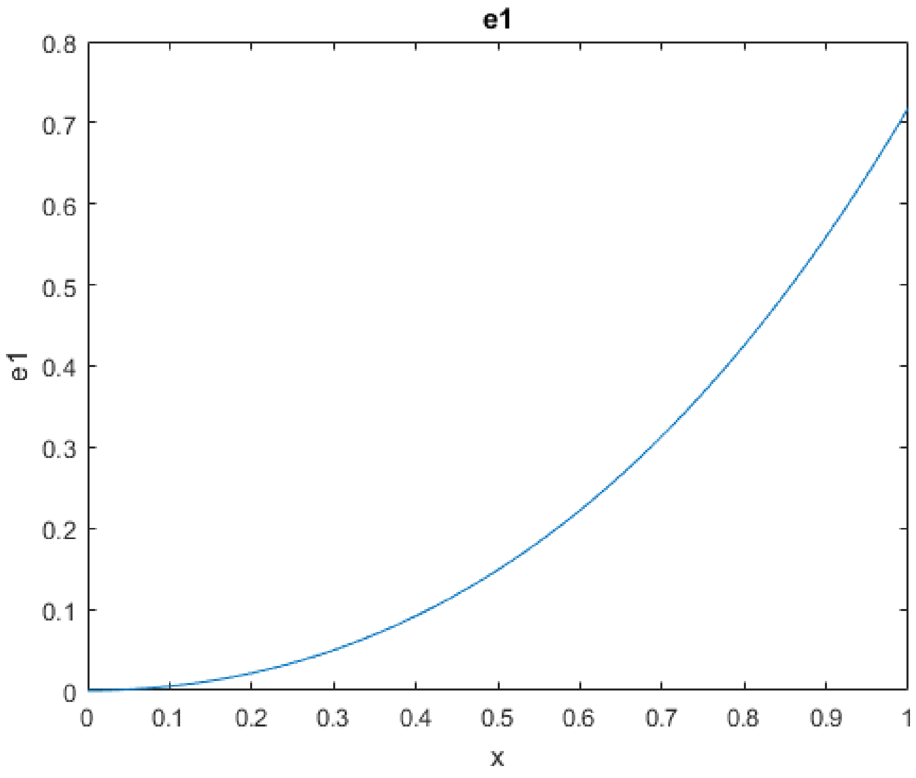

Figure 3: Error curve of e_1 (max error: 0.7183)

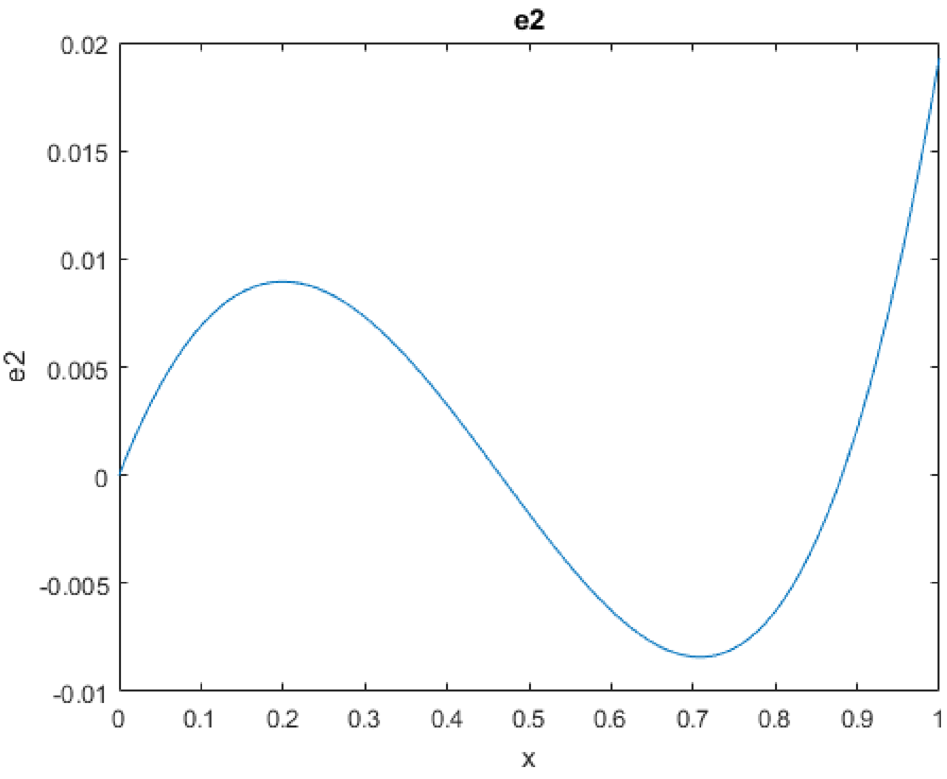

Figure 4: Error curve of e_2 (max error: 0.0193)

3 Problem 3

1. The derivation for the normal equation:

$$(\pi_i, \pi_j) = \int_0^\infty e^{-it}e^{-jt}dt = \int_0^\infty e^{-(i+j)t}dt = \frac{1}{i+j}$$

$$(\pi_i, f) = \int_0^1 e^{-it}dt = \frac{1}{i}(1-e^{-i})$$

The normal equation is:

$$\sum_{j=1}^{n} \frac{1}{i+j}c_j = \frac{1}{i}(1-e^{-i}), i = 1, 2, 3, ..., n$$

The normal equation in matrix form is $Ac = b$, where:

$$A = \begin{bmatrix} 1/2 & 1/3 & 1/4 & ... & 1/(n+1) \\ 1/3 & 1/4 & 1/4 & ... & 1/(n+2) \\ ... & ... & ... & ... & ... \\ 1/(n+1) & 1/(n+2) & 1/(n+3) & ... & 1/(n+n) \end{bmatrix} b = \begin{bmatrix} 1-e^{-1} \\ \frac{1}{2}(1-e^{-2}) \\ \frac{1}{3}(1-e^{-3}) \\ \frac{1}{4}(1-e^{-4}) \\ ... \\ \frac{1}{n}(1-e^{-n}) \end{bmatrix}$$

The matrix A is a $n+1$ order of Hilbert matrix that does not include the first column and row.

2. The plot of condition number for $1 <= n <= 4$ is shown below:

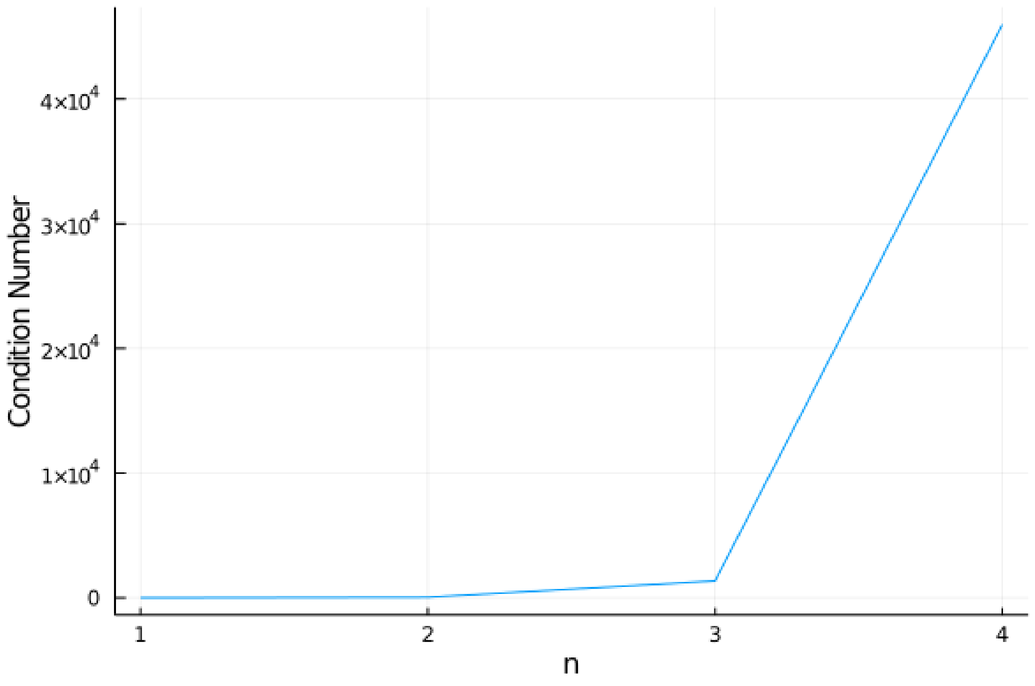

Figure 5: Condition number plot for $n = 1, 2, 3, 4$

4 Problem 4

1. Each items in the matrix A is derived as:

$$a_{ij} = (B_j^n, B_i^n) = \int B_j^n B_i^n \omega(t) dt$$
$$= \frac{\binom{n}{j}\binom{n}{i}}{\binom{2n}{i+j}} \int B_{i+j}^{2n} \omega(t) dt$$
$$= \frac{\binom{n}{j}\binom{n}{i}}{\binom{2n}{i+j}} \int_0^1 B_{i+j}^{2n} dt$$

Since the integral of the Bernstein basis is:

$$\int_a^b B_i^n(t) dt = \frac{(b-a)}{n+1}$$

The result of the inner product is

$$\frac{\binom{n}{j}\binom{n}{i}}{\binom{2n}{i+j}} \int_0^1 B_{i+j}^{2n} dt = \frac{n!}{j!(n-j)!} \frac{n!}{i!(n-i)!} \frac{(i+j)!(2n-i-j)!}{(2n)!} \frac{1}{2n+1}$$
$$= \frac{n!}{j!(n-j)!} \frac{n!}{i!(n-i)!} \frac{(i+j)!(2n-i-j)!}{(2n+1)!}$$

2. One of the properties of Bernstein basis is that

$$\sum_{j=0}^{n} B_j^n(t) = 1$$

Since $f(t)$ is a linear combination of Bernstein basis, all the coefficients for the Bernstein basis is needs to be 1 in order to have $f(t) = 1$. The expected result of c is 1 for all coefficients. The infinity norm of the error vector and condition number of A is shown in the table below for $n = 5 : 5 : 25$

n	$\|\|Error\|\|_\infty$	Cond A
5	1.40998×10^{-14}	462
10	3.37618×10^{-11}	352716
15	5.67370×10^{-6}	3.0054×10^{8}
20	0.01885	2.6913×10^{11}
25	2949.99086	2.4794×10^{14}

The code for generating the result is shown below:

```
using LinearAlgebra

n_vec = 5:5:25
println("n      error      cond")
for n in n_vec
  A = zeros(Float64, n+1, n+1)
  b = zeros(Float64, 1, n+1)
  for i in 0:n
    for j in 0:n
      A[i+1,j+1] = ((factorial(big(n))/(factorial(big(i))
                        *factorial(big(n-i))))
                    *(factorial(big(n))/(factorial(big(j))
                        *factorial(big(n-j))))
                    *((factorial(big(i+j))*factorial(big(2*n-i-j)))
                        /factorial(big(2*n+1))))
    end
    b[i+1] = 1 / (n+1)
  end
  c = b * inv(A)
  condition = cond(A)
  error = norm(c-ones(1, n+1), Inf)
  println(n, "    ", error, "    ", condition)
end
```

5 Problem 5

1. The plots for the Chebyshev points of the first kind is shown below.

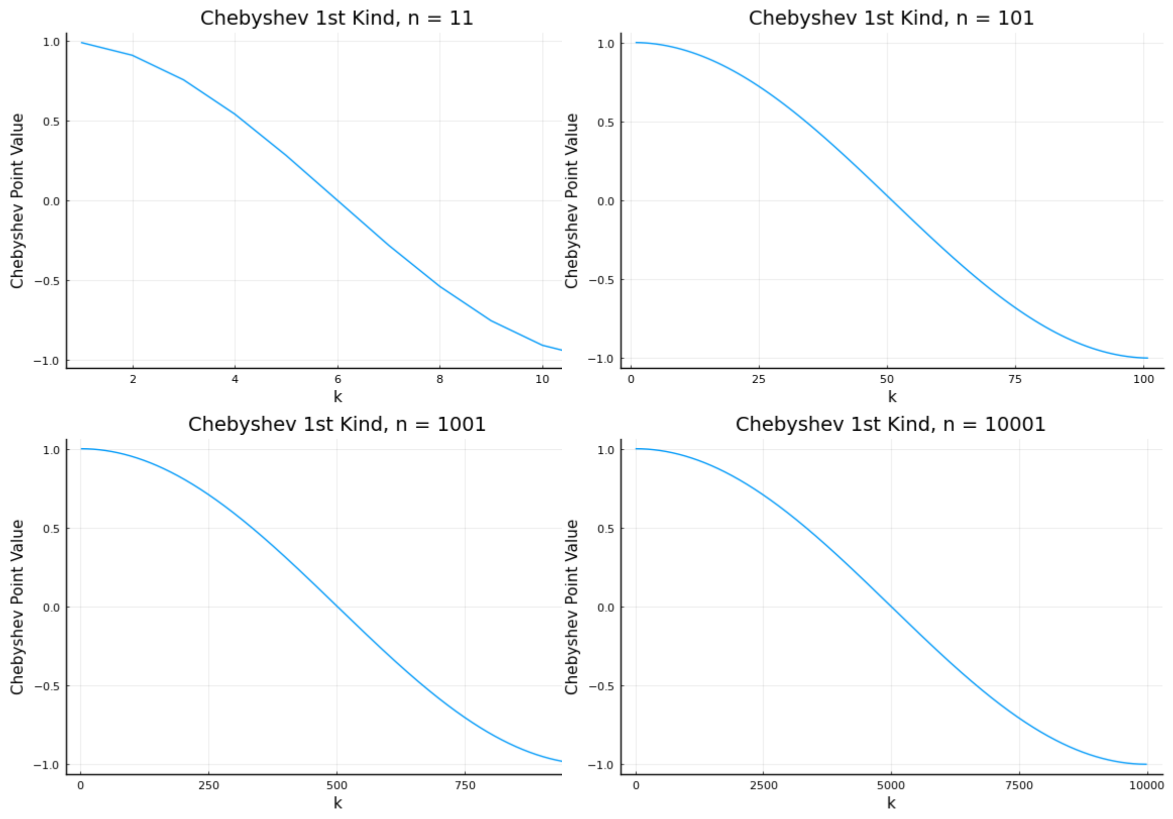

Figure 6: Plots of Chebyshev points of the first kind when $n = [11, 101, 1001, 10001]$

The plots for the Chebyshev points of the second kind is shown below.

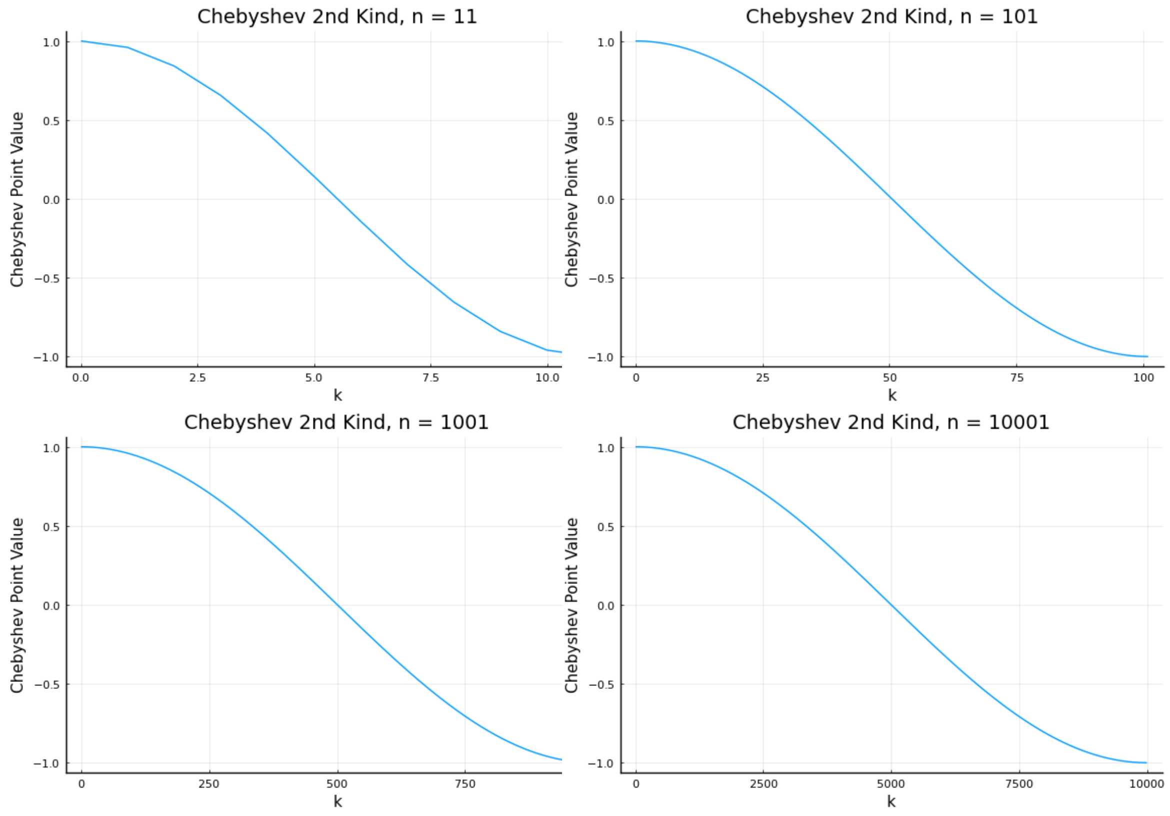

Figure 7: Plots of Chebyshev points of the second kind when $n = [11, 101, 1001, 10001]$

The code that is used to generate plot is shown below.

```
using Pkg
Pkg.add("Plots")
using Plots
using LinearAlgebra

n_vec = [11, 101, 1001, 10001]

# compute the Chebyshev points of first kind
for n in n_vec
    k = 1:n
    x1_k = cos.((2 .* k .- 1) ./ (2 .* n) .* pi)
    plot(
        k,
        x1_k,
        xlabel = "k",
        ylabel = "Chebyshev Point Value",
        title = string("Chebyshev 1st Kind, n = ", n),
        legend = false,
        fmt = png,
    )
    png(string("1st_point_", n, ".png"))
```

```
end

# compute the Chebyshev points of second kind
for n in n_vec
    k = 0:n
    x2_k = cos.(k .* (pi / n))
    plot(
        k,
        x2_k,
        xlabel = "k",
        ylabel = "Chebyshev Point Value",
        title = string("Chebyshev 2nd Kind, n = ", n),
        legend = false,
        fmt = png,
    )
    png(string("2nd_point_", n, ".png"))
end
```

2. The program for approximating functions using Chebyshev points of the second kind is shwon below:

```
# using Pkg
# Pkg.add("Plots")
using Plots
using LinearAlgebra

function interpolate_unif(f, r, n, pts)
    # generate the chebyshev points (2nd kind)
    k = 0:n
    x_k = cos.(k .* (pi / n))
    x_k = (r[2] + r[1]) / 2 .+ (r[2] - r[1]) / 2 .* x_k
    y = zero(pts)
    w = ones(Float64, (n + 1,))
    f_k = f(x_k)
    for i in k .+ 1
        w[i] = prod(x_k[i] .- x_k[1:i-1])
        w[i] *= prod(x_k[i] .- x_k[i+1:n+1])
    end
    for i in 1:size(pts, 1)
        wt = w .* (pts[i] .- x_k)
        y[i] = sum(f_k ./ wt) / sum(1.0 ./ wt)
    end
    return y

end

# f(x) = exp.(-x)
# plot_title = "Chebyshev Approximation of e^x"
# file_name = "chebyshev_approx_ex.png"
# r = [1,5]
f(x) = 1 ./ (x .^ 2 .+ 5)
plot_title = "Chebyshev Approximation of 1/(x^2+5)"
file_name = "chebyshev_approx_poly.png"
r = [-5,5]
pts = range(r[1], stop = r[2], length = 10000)
n_vec = [5, 8, 11, 30, 51]
plot(pts, f(pts), label="function", title=plot_title)
for n in n_vec
    y = interpolate_unif(f, r, n, pts)
    display(plot!(pts,
                  y,
                  label=string("n=", n)))
end
png(string("Homework2/", file_name))
```

3. The result of the above program approximating $f(x) = e^{-x}$ is shown below.

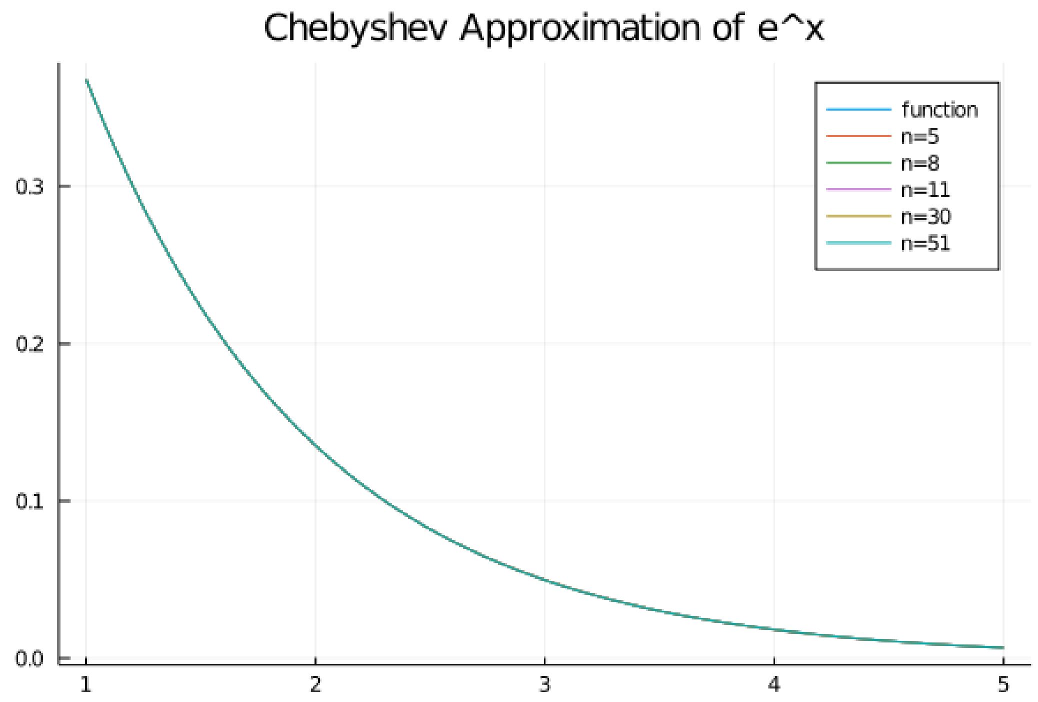

Figure 8: Chebyshev approximation of $f(x) = e^{-x}$

From the result above, for all the n values, the approximation to the function $f(x) = e^{-x}$ is very close to function plot. The approximation for $n = 5$ is sufficient for a lot of the use cases.

4. The result of the above program is shown below.

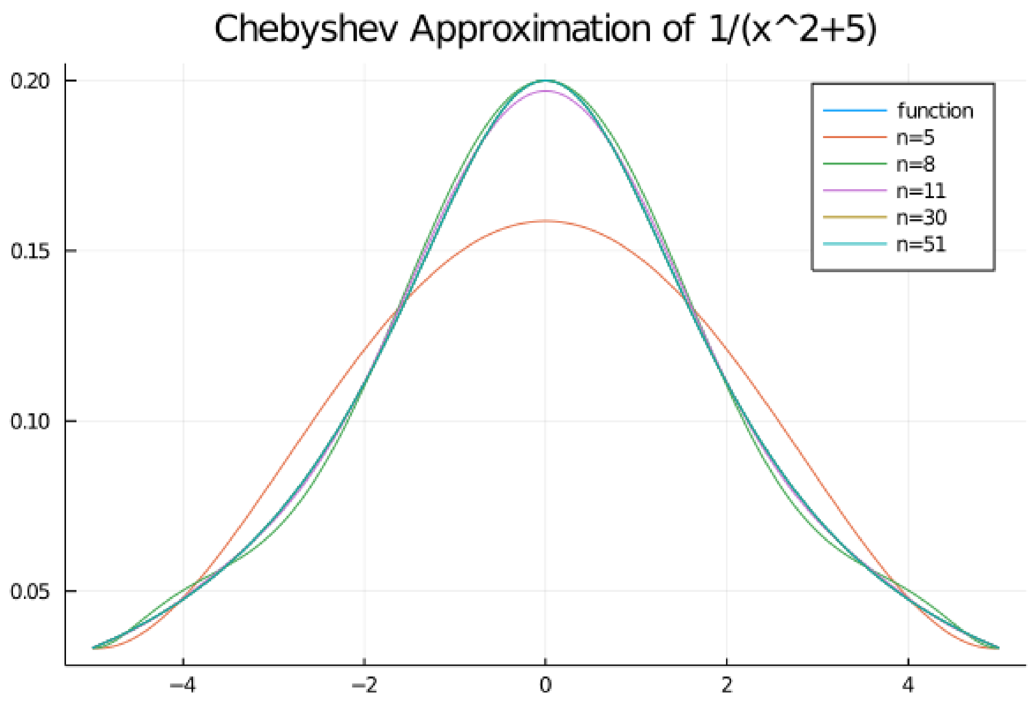

Figure 9: Chebyshev approximation of $f(x) = \frac{1}{x^2+5}$

From the result above, as the value of n increases, the error of the approximation decreases. The maximum error happens at the each end of the range and at when $x = 0$. Comparing between Figure 9 and Figure 8, a smooth function without steep hills and valley generally requires less terms to approximate. The approximation for $f(x) = \frac{1}{x^2+5}$ when $n = 51$ still have visible error between the approximation and function plot.

5. In the course logo, it looks like a Chebyshev approximation to the blue function. The lines with other color are the Chebyshev approximation with polynomials with different degrees. Same as the observation is part 4, the maximum error happens at the end point. As the degree of the polynomial increases, the approximation becomes close to the actual function.

6 Problem 6

1. The L_2 norm of the difference between $f(x,y)$ and constant c is calculated as:

$$\begin{aligned}\|f-c\|_2^2 &= \int_{-1}^{0}\int_{-1}^{0}(c+1)^2dxdy + \int_{0}^{1}\int_{0}^{1}(c-1)^2dxdy \\ &+ \int_{-1}^{0}\int_{0}^{1}c^2dxdy + \int_{0}^{1}\int_{-1}^{0}c^2dxdy \\ &= (c+1)^2+(c-1)^2+c^2+c^2 \\ &= 2+4c^2\end{aligned}$$

To minimize the different, take the derivative of the L2 norm of the difference with respect to c and set it to zero.

$$\frac{d(\|f-c\|_2^2)}{dc} = 8c = 0$$

$$c = 0$$

2. The number of points can be determined by obersvation.

d	Terms	Number of Terms
0	1	1
1	$1, x, y$	1+2
2	$1, x, y, xy, x^2, y^2$	1+2+3
3	$1, x, y, xy, x^2, y^2, x^2y, xy^2, x^3, y^3$	1+2+3+4

From observation, the number of unique points needed can be calculated as:

$$\text{(Num of Points)} = \sum_{i=1}^{d+1} i$$

3. The number of points to uniquely determine the interpolating polynomial of degree at most d if the are k variable can be calculated by counting number of terms in each of degrees from 0 to k. For each degree, the number of unique points for that degree is the number of combination of all variables with the sum of the power equals to the degree. For example, if $k = 3$, the total number of points can be counted as shown below:

d	Terms	Count
0	1	1
1	x, y, z	3
2	$xy, yz, xz, x^2, y^2, z^2$	6
3	$xyz, x^2y, xy^2, y^2z, z^2y, x^2z, xz^2, x^3, y^3, z^3$	10
...	...	...

By adding all the numbers in the *count* column, the result is the total number of points to uniquely determine the interpolation polynomial of degree at most d when there are k variable. In the case of $d = 3 \text{and} k = 3$, the total number of points is 20.

4. The Lagrange form of the interpolating polynomial in the bivariate case can be derived by first evaluate the polynomial with respect to the individual variable.

$$L_i(x) = \prod_{s=0, s\neq i}^{n} \frac{(x - x_s)}{(x_i - x_s)}, i = 0, 1, 2, 3, ..., n$$

$$L_j(y) = \prod_{r=0, r\neq j}^{m} \frac{(y - y_r)}{(y_j - y_r)}, j = 0, 1, 2, 3, ..., m$$

For the bivariate polynomial, we want the similar properties to univariate polynomial which is:

$$\text{univariate} : L_i(x_k) = \begin{cases} 1 & i = k \\ 0 & i \neq k \end{cases}$$

$$\text{bivariate} : L_{ij}(x_s, y_r) = \begin{cases} 1 & s = i, r = j \\ 0 & otherwise \end{cases}$$

If the result of $L_i(x)$ and $L_j(y)$ will be multiplied together, it will produce the desired result for bivariate lagrange polynomial where it equals 1 at $s = iandr = j$. The expression for the bivariate Lagrange Polynomial is:

$$L_{ij}(x, y) = \left(\prod_{s=0, s\neq i}^{n} \frac{(x - x_s)}{(x_i - x_s)} \right) \left(\prod_{r=0, r\neq j}^{m} \frac{(y - y_r)}{(y_j - y_r)} \right)$$

The interpolation of the $f(x, y)$ using the bivariate Lagrange polynomial is:

$$p(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{m} f(x_i, x_j) L_{ij}(x, y)$$

5. Using the Lagrange polynomial derived above, the interpolation of the function, $f(x,y) = \frac{1}{x^2+y^2+5}$, using 5 Chebyshev points of the first kind with a region of [-5,5] is shown below:

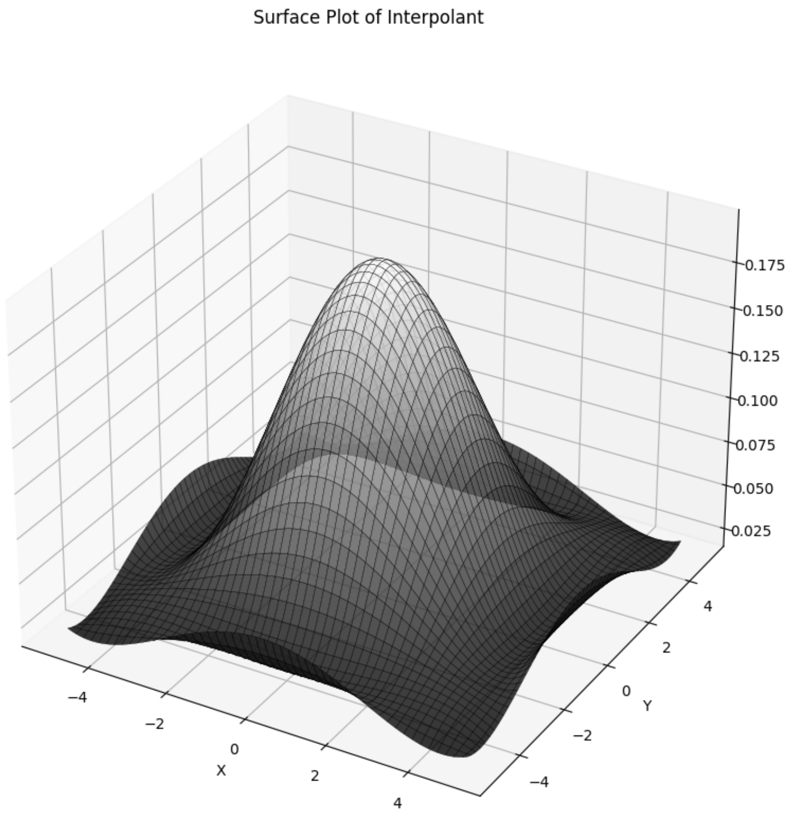

Figure 10: Chebyshev approximation of $f(x,y) = \frac{1}{x^2+y^2+5}$ using 5 Chebyshev points of first kind

The code for generate the figure is shown below:

```
using PyPlot
using LinearAlgebra

f(x,y) = 1.0 ./ (x.^2 .+ y.^2 .+ 5)
r = [-5,5]
n = 100
x_vec = range(r[1], stop=r[2], length=n)
y_vec = range(r[1], stop=r[2], length=n)

x_grid = repeat(x_vec',n,1)
y_grid = repeat(y_vec,1,n)
```

```
z = zeros(n,n)

z_func = f(x_grid, y_grid)

fig = figure("function_surfactplot", figsize=(10,10))
plot_surface(
    x_grid,
    y_grid,
    z_func,
    rstride = 2,
    edgecolors = "k",
    cstride = 2,
    cmap = ColorMap("gray"),
    alpha = 0.8,
    linewidth = 0.25,
)
xlabel("X")
ylabel("Y")
PyPlot.title("Surface Plot of Function")
show()

# generate Chebyshev points
println("Generate Chebyshev Points")
k = 1:5
c_pts = cos.((2 .* k .- 1) ./ (2 .* 5) .* pi)
c_pts = (r[2] + r[1]) / 2 .+ (r[2] - r[1]) / 2 .* c_pts

for ind_x in 1:n
    for ind_y in 1:n
        x = x_vec[ind_x]
        y = y_vec[ind_y]
        for i = 1:5
            for j = 1:5
                Lix = 1
                Liy = 1
                for s = 1:5
                    if (i != s)
                        Lix *= (x-c_pts[s])/(c_pts[i]-c_pts[s])
                    end
                    if (j != s)
                        Liy *= (y-c_pts[s])/(c_pts[j]-c_pts[s])
                    end
                end
                L = Lix * Liy
                z[ind_x, ind_y] += f(c_pts[i], c_pts[j]) * L
            end
        end
    end
```

```
end

fig = figure("interpolation_surfactplot", figsize=(10,10))
plot_surface(
    x_grid,
    y_grid,
    z,
    rstride = 2,
    edgecolors = "k",
    cstride = 2,
    cmap = ColorMap("gray"),
    alpha = 0.8,
    linewidth = 0.25,
)
xlabel("X")
ylabel("Y")
PyPlot.title("Surface Plot of Interpolant")
show()

fig = figure("interpolation_contour", figsize=(10,10))
cp =contour(
    x_grid,
    y_grid,
    z,
    color="black",
    linewidth=2.0,
)
xlabel("X")
ylabel("Y")
PyPlot.title("Contour Plot of Interpolant")
show()
```

7 Problem 0

For this homework, I have only ask our TA, Sai, about the homework. All the source code for this homework has been included in this homework.