# Ch 5. Polynomial Interpolation and Clustered Grids

Solving homework using Python

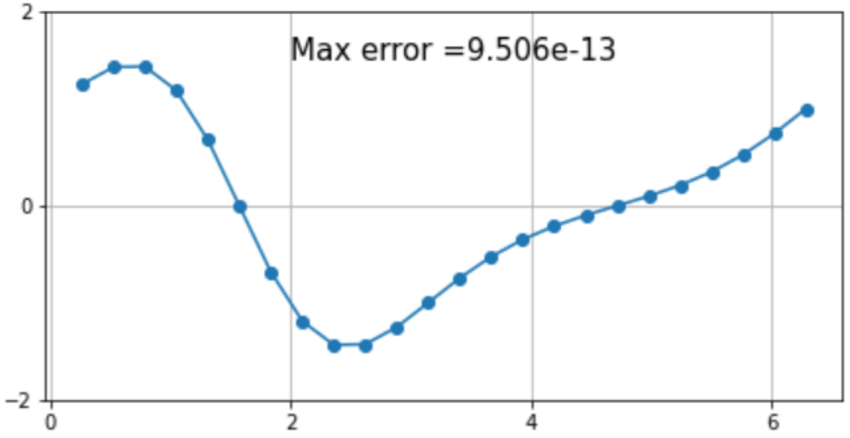
2020.10.21(전산천문학)

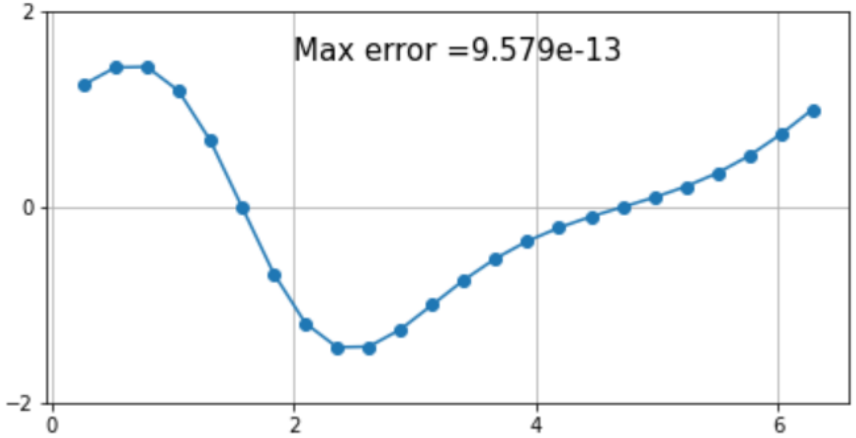
202050135 홍지민

# Program. Previous Homework results

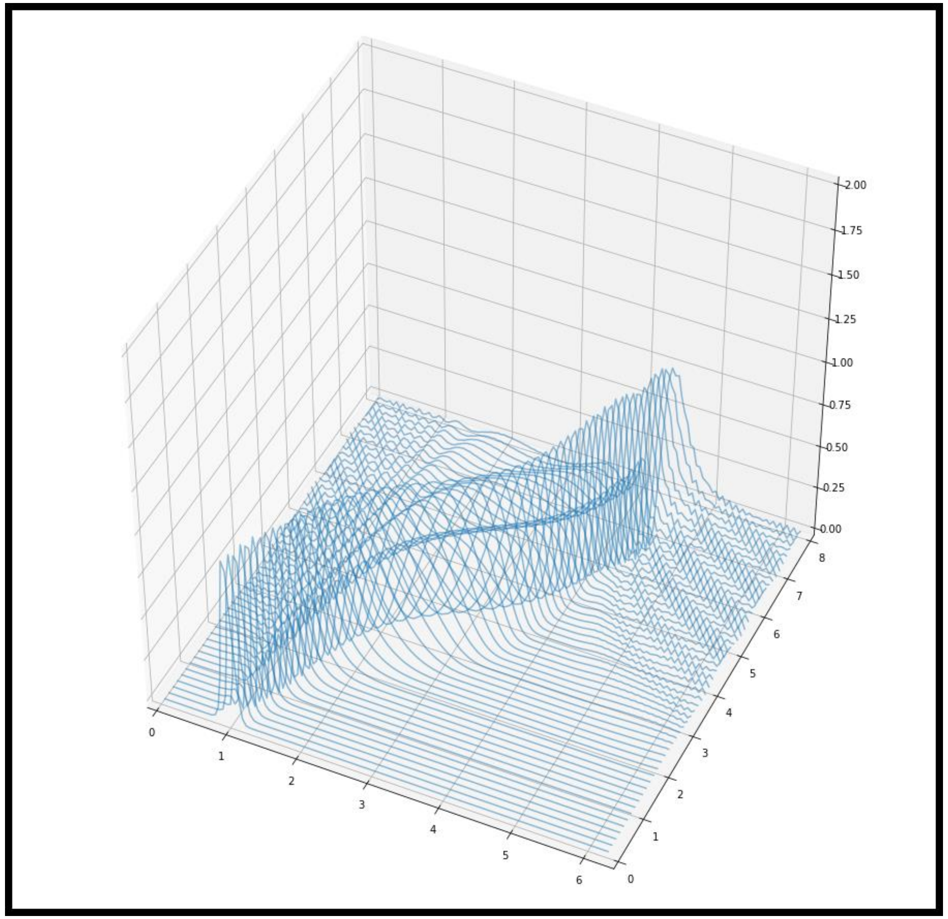
$$\begin{pmatrix} w_1 \\ \\ \vdots \\ \\ w_N \end{pmatrix} = h^{-1} \begin{pmatrix} 0 & \frac{1}{2} & & & -\frac{1}{2} \\ -\frac{1}{2} & 0 & \ddots & & \\ & & \ddots & & \\ & & \ddots & 0 & \frac{1}{2} \\ \frac{1}{2} & & & -\frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} u_1 \\ \\ \vdots \\ \\ u_N \end{pmatrix}. \qquad (1.2)$$

Calculate Max Error Value




Modified of the matrix and Derived correct plot



# Program1. Matlab Code

1. Follow Program 8* (in Chapter 4) to solve the problem of a quantum harmonic oscillator

$$-u_{xx} + x^2 u = \lambda u, \quad x \in \mathbb{R}$$

Produce a result similar to Output 8.

(Program 8 is our first eigenvalue problem. We will see more of these later in the course.)

## Program 8

```
% p8.m - eigenvalues of harmonic oscillator -u"+x^2 u on R

  format long
  L = 8;                              % domain is [-L L], periodic
  for N = 6:6:36
    h = 2*pi/N; x = h*(1:N); x = L*(x-pi)/pi;
    column = [-pi^2/(3*h^2)-1/6 ...
       -.5*(-1).^(1:N-1)./sin(h*(1:N-1)/2).^2];
    D2 = (pi/L)^2*toeplitz(column);   % 2nd-order differentiation
    eigenvalues = sort(eig(-D2 + diag(x.^2)));
    N, eigenvalues(1:4)
  end
```

## Output 8

(with added shading of unconverged digits)

```
N =  6                      N = 12

   0.46147291699547            0.97813728129859
   7.49413462105052            3.17160532064718
   7.72091605300656            4.45593529116679
  28.83248377834015            8.92452905811993

N = 18                      N = 24

  0.99997000149932            0.99999999762904
  3.00064406679582            3.00000009841085
  4.99259532440770            4.99999796527330
  7.03957189798150            7.00002499815654

N = 30                      N = 36

  0.99999999999993            0.99999999999996
  3.00000000000075            3.00000000000003
  4.99999999997560            4.99999999999997
  7.00000000050861            6.99999999999999
```

# Program1. Python Code

```
from scipy.linalg import toeplitz
import pandas as pd

L = 8

df = pd.DataFrame()

for N in np.arange(6,36 +6,6):
  h = 2*np.pi/N
  x = h*np.arange(1,N+1,1) ; x = L*(x-np.pi)/np.pi
  column = [-np.pi**2/(3*h**2)-1/6]
  column = np.append(column,-0.5 * (-1)**np.arange(1,N,1)/np.sin(1/2*h*np.arange(1,N,1))**2)
  D2 = (np.pi/L)**2*toeplitz(column)
  eigenvalues = np.sort(np.linalg.eigvals(np.diag(x**2)-D2))
  df['N = '+str(N)] = eigenvalues[0:6]

df
```

Library for creating tables

$$S_N''(x_j) = \begin{cases} -\dfrac{\pi^2}{3h^2} - \dfrac{1}{6} & j \equiv 0 \pmod{N}, \\ -\dfrac{(-1)^j}{2\sin^2(jh/2)} & j \not\equiv 0 \pmod{N}. \end{cases} \qquad (3.11)$$

Thus second-order spectral differentiation can be written in the matrix form

$$D_N^{(2)} v = \begin{pmatrix} \ddots & \vdots & \\ \ddots & -\frac{1}{2}\csc^2(\frac{2h}{2}) & \\ \ddots & \frac{1}{2}\csc^2(\frac{h}{2}) & \\ & -\frac{\pi^2}{3h^2} - \frac{1}{6} & \\ & \frac{1}{2}\csc^2(\frac{h}{2}) & \ddots \\ & -\frac{1}{2}\csc^2(\frac{2h}{2}) & \ddots \\ & \vdots & \ddots \end{pmatrix} v. \qquad (3.12)$$

$$\frac{d^2}{dx^2} = \left(\frac{\pi}{L}\right)^2 \frac{d^2}{dt^2}$$

Approximation $(-D_N^{(2)} + S)v = \lambda v,$

$$-u'' + x^2 u = \lambda u,$$

$x's\ space : [-L, L], t's\ space : [0,2\pi]$

# In Python

| | N = 6 | N = 12 | N = 18 | N = 24 | N = 30 | N = 36 |
|---|---|---|---|---|---|---|
| 0 | 0.461473 | 0.978137 | 0.999970 | 1.000000 | 1.0 | 1.0 |
| 1 | 7.494135 | 3.171605 | 3.000644 | 3.000000 | 3.0 | 3.0 |
| 2 | 7.720916 | 4.455935 | 4.992595 | 4.999998 | 5.0 | 5.0 |
| 3 | 28.832484 | 8.924529 | 7.039572 | 7.000025 | 7.0 | 7.0 |
| 4 | 29.037940 | 9.288546 | 8.814572 | 8.999765 | 9.0 | 9.0 |
| 5 | 64.494202 | 17.836071 | 11.462089 | 11.001484 | 11.0 | 11.0 |

As the value of N increases, it can be seen that it converges at 1,3,5,7,9……

In other words, the error is thought to be decreasing.

# In Matlab

N = 6

0.46147291699547
7.49413462105052
7.72091605300656
28.83248377834015

N = 12

0.97813728129859
3.17160532064718
4.45593529116679
8.92452905811993

N = 18

0.99997000149932
3.00064406679582
4.99259532440770
7.03957189798150

N = 24

0.99999999762904
3.00000009841085
4.99999796527330
7.00002499815654

N = 30

0.99999999999993
3.00000000000075
4.99999999997560
7.00000000050861

N = 36

0.99999999999996
3.00000000000003
4.99999999999997
6.99999999999999

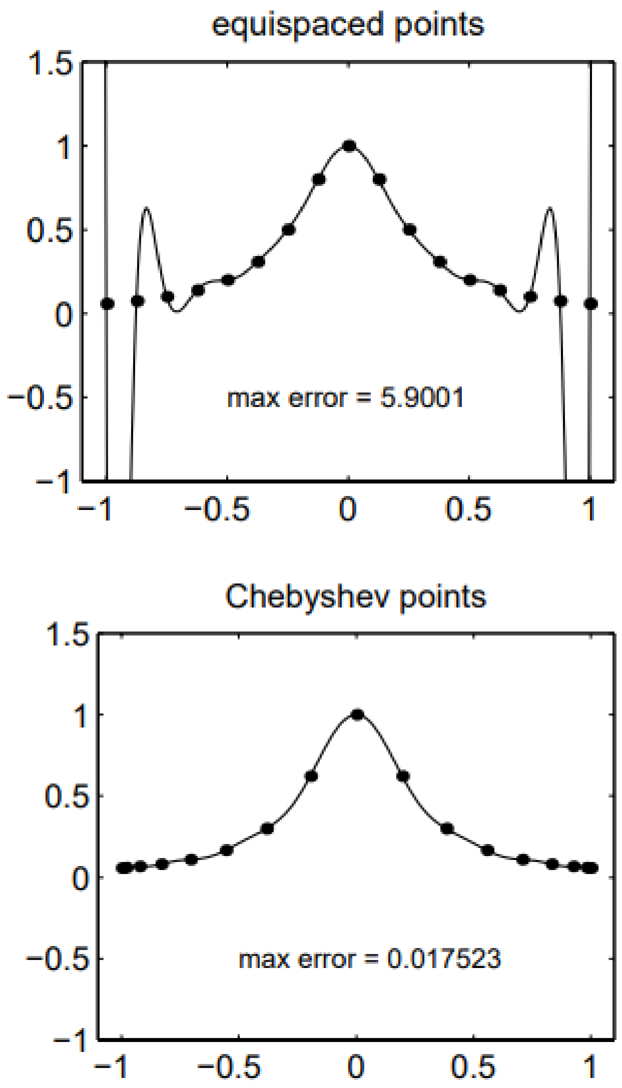
# Program2. Matlab Code

2. Implement Program 9 and produce a plot similar to Output 9.

## Program 9

```
% p9.m - polynomial interpolation in equispaced and Chebyshev pts

  N = 16;
  xx = -1.01:.005:1.01; clf
  for i = 1:2
    if i==1, s = 'equispaced points'; x = -1 + 2*(0:N)/N; end
    if i==2, s = 'Chebyshev points';  x = cos(pi*(0:N)/N); end
    subplot(2,2,i)
    u = 1./(1+16*x.^2);
    uu = 1./(1+16*xx.^2);
    p = polyfit(x,u,N);                 % interpolation
    pp = polyval(p,xx);                 % evaluation of interpolant
    plot(x,u,'.','markersize',13)
    line(xx,pp,'linewidth',.8)
    axis([-1.1 1.1 -1 1.5]), title(s)
    error = norm(uu-pp,inf);
    text(-.5,-.5,['max error = ' num2str(error)])
  end
```

## Output 9



# Program2. Python Code

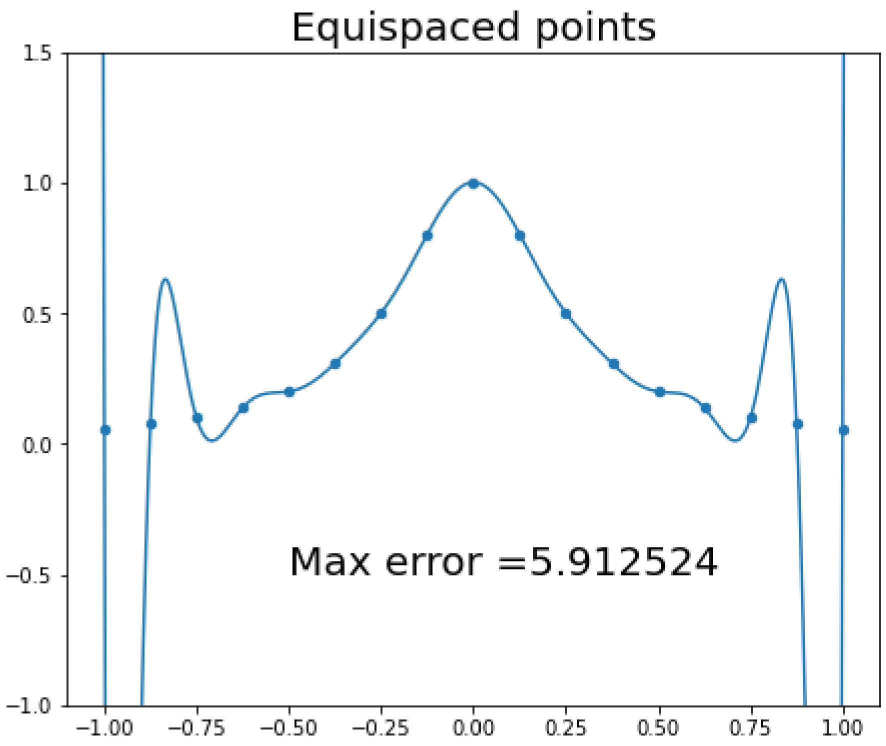
```
N = 16
xx = np.linspace(-1.01,1.01,1000)

fig, ax = plt.subplots(1,2,sharex=True,sharey=True,figsize=(16,6))

for i in [1,2]:
  if i == 1:
    s = 'Equispaced points'
    x = -1 + 2 * np.arange(0,N+1,1)/N
  else:
    s = 'Chebyshev points'
    x = np.cos(np.pi*np.arange(0,N+1,1)/N)
  u = 1 / (1 + 16 * x**2)
  uu = 1 / (1 + 16 * xx**2)
  p = polyfit(x,u,N)
  pp = polyval(p,xx)
  ax[i-1].scatter(x,u,marker='o',s=20)
  ax[i-1].plot(xx,pp)
  ax[i-1].set_title(s,fontsize=20)
  error = max(uu-pp)
  ax[i-1].text(-0.5,-0.5,'Max error ='+str(round(error,6)),fontsize=20)

plt.xlim(-1-0.1,1+0.1)
plt.ylim(-1,1.5)
```

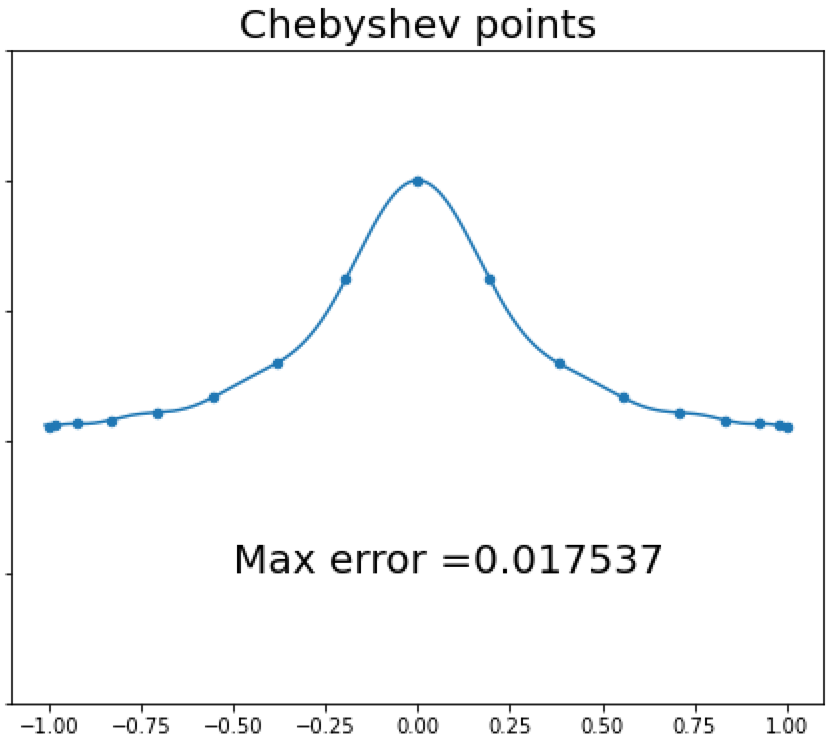
This part selects that the grid's method.
(Equispace or Chebyshev)

Execute polynomial Least Squares Method
(N is the highest order term setting value)

Calculate using derived coefficients

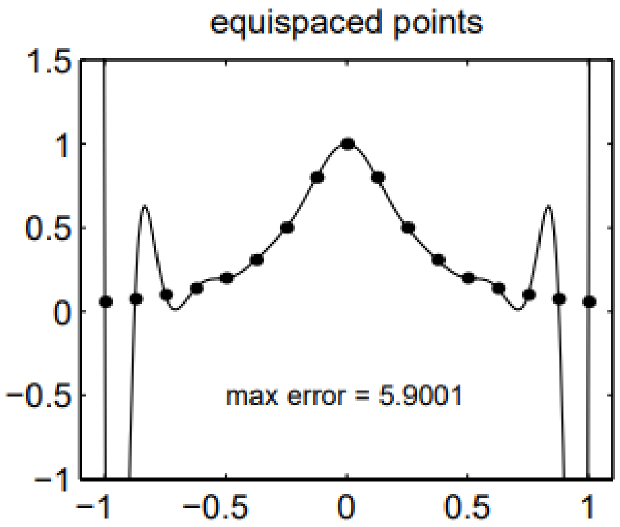
# In Python

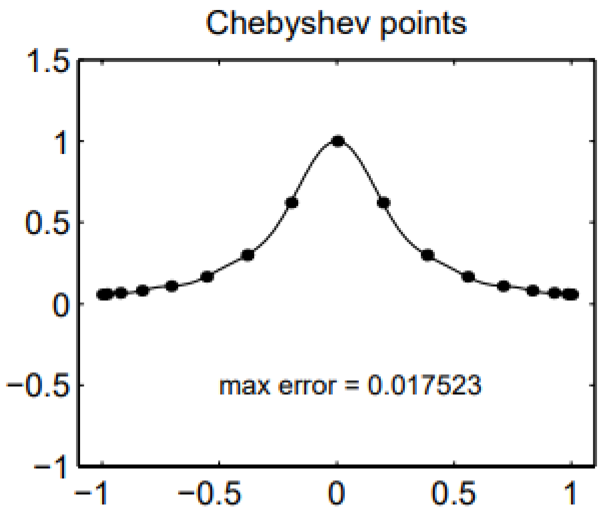



I think it's a better fitting because the error is small in Chebyshev's method.

# In Matlab

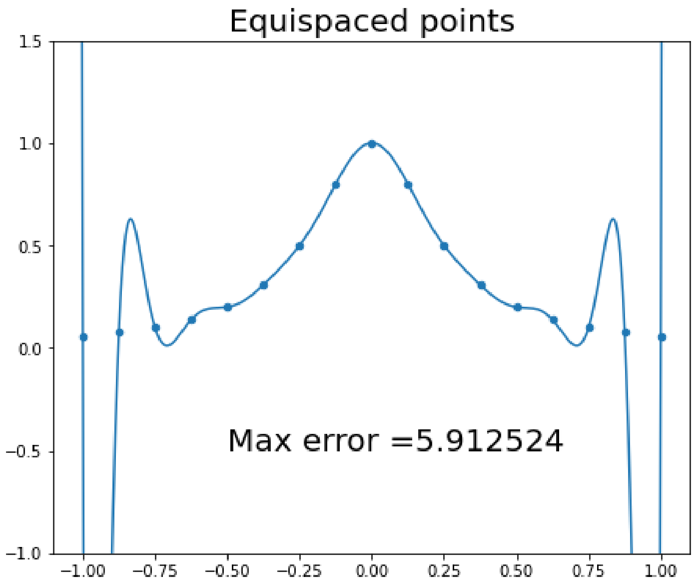


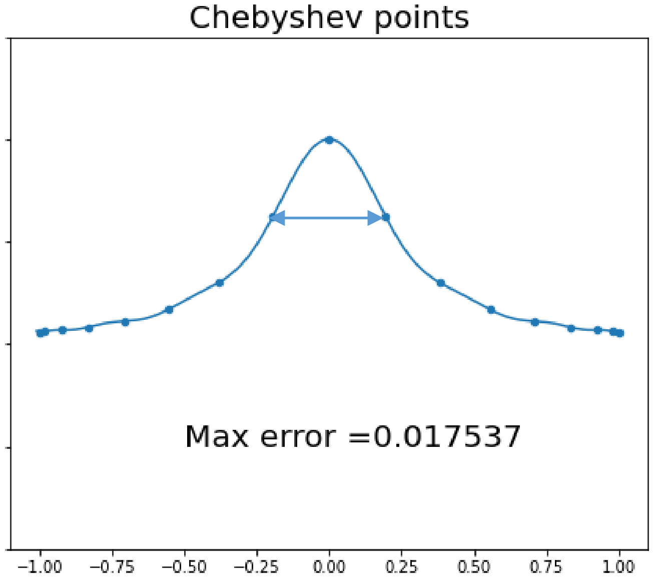



# Program2. Additionally

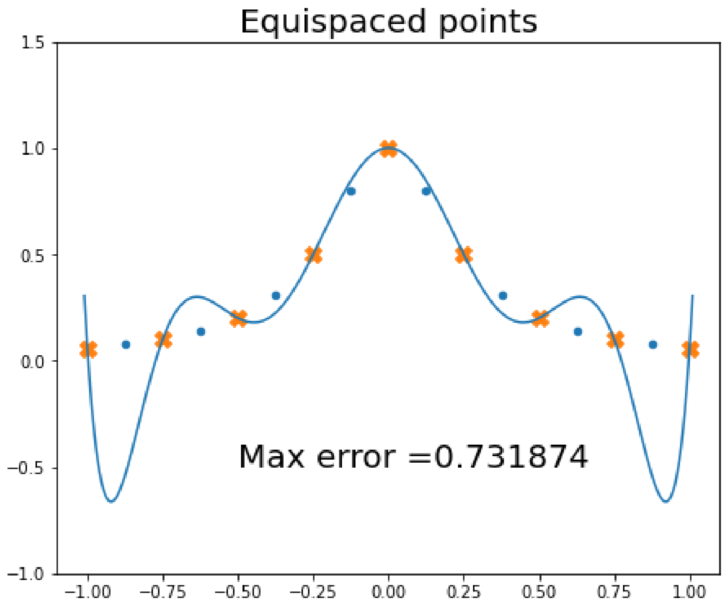
But I found a problem with this method.
There was a tendency to not fit well with data on low N values.

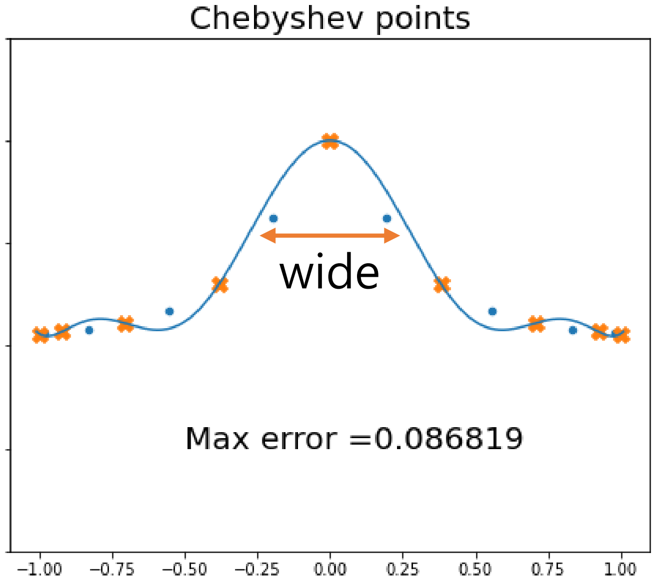
N = 16




N = 8



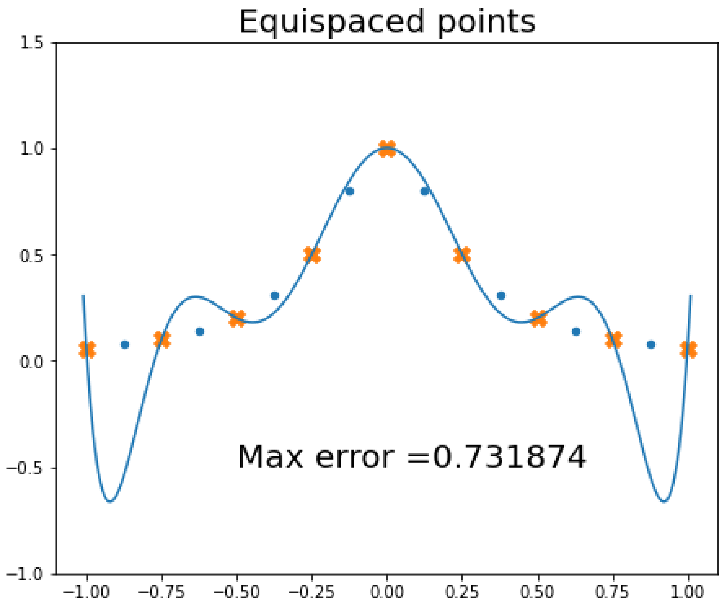

Because only red points are used in the polynomial least square method

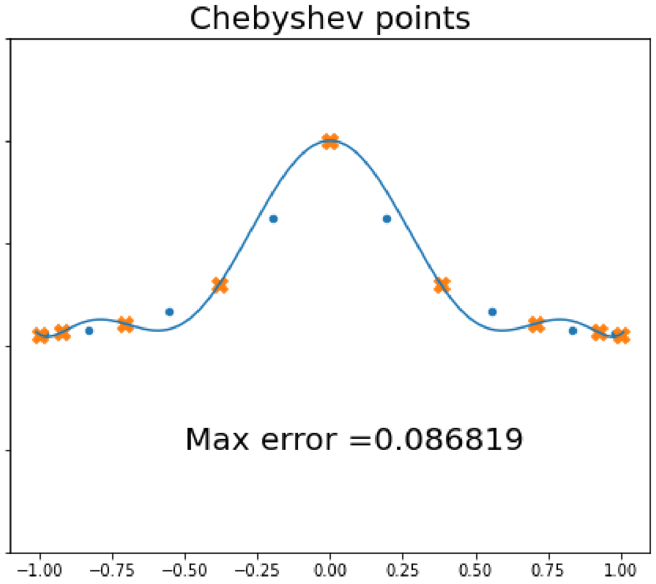
# Program2. Additionally

I solved this with machine learning techniques.
I will skip detailed machine learning techniques.

N = 8

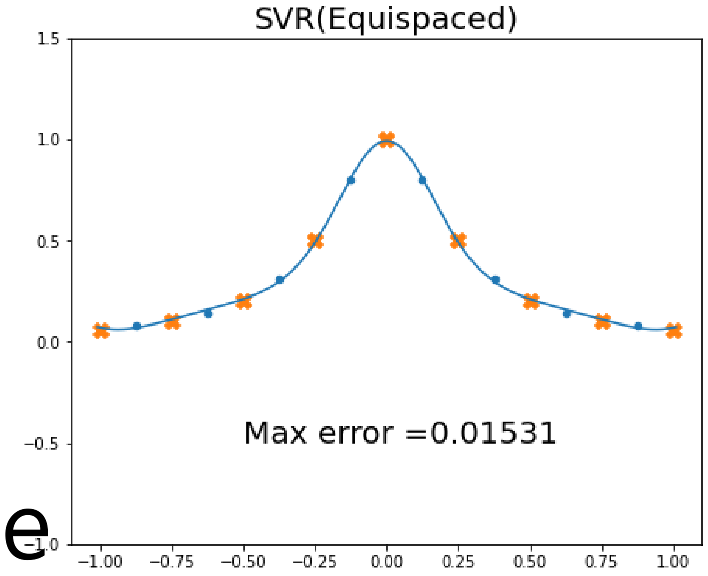
Book's Method

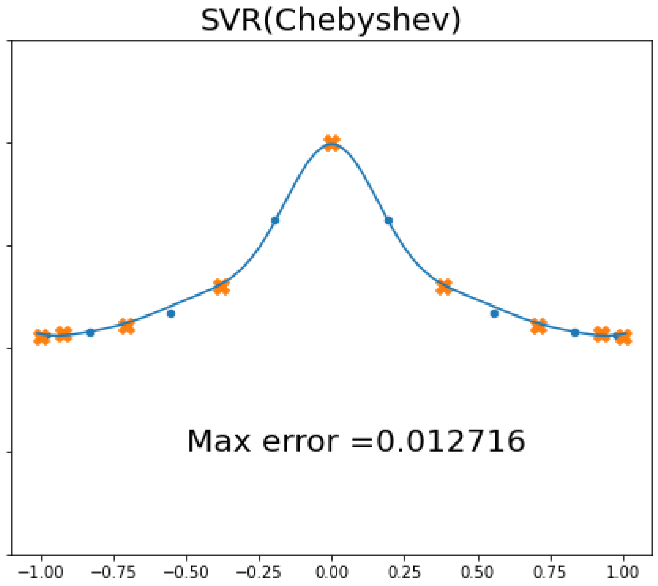



N = 8

Machine Learning




The error could also be made low by adjusting several set values.