1) Consider the function

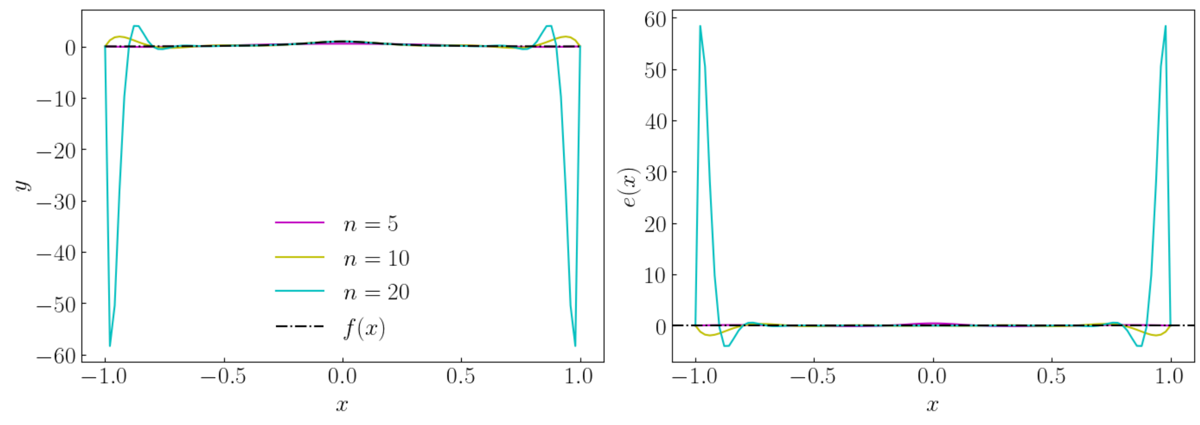
$$f(x) = \frac{1}{1+25x^2}.$$

on the interval $[-1, 1]$.

a) For $n = 5$, 10, 20 plot the error $e(x) = f(x) - p_n(x)$ where the polynomial $p_n(x)$ is computed by interpolating at the $n+1$ equally-spaced nodes over $[-1, 1]$.

The code for interpolation is recycled from the previous homework. Again, we perform Lagrange interpolation since it is the most straightforward to program, and there is no clear advantage to using divided differences. A script for the interpolation and generation of data points is shown below. The program also addresses parts (b) and (c) of this question.

```
#!/usr/bin/env python3

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['text.latex.preamble'] = r'\usepackage{amsmath}'
rcParams['text.usetex']         = True
rcParams['font.family']         = 'sans-serif'
rcParams['font.sans-serif']     = ['Helvetica']

def Lagrange_poly(t,x,i):
    res = 1.0
    for j in range(len(x)):
        if j == i:
            continue
        else:
            res *= (t-x[j])/(x[i]-x[j])
    return res

def Lagrange_interp(t,x,y):
    res = 0.0
    for i in range(len(x)):
        res += y[i]*Lagrange_poly(t,x,i)
    return res

def gen_plot(f,T,N,save_path,nodes='uniform'):
    F = f(T)

    ls = ['m-','y-','c-']
    fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(7*2,5))
    for n in N:
        if nodes == 'uniform':
            x_data = np.linspace(-1.0,1.0,n+1)
        elif nodes == 'chebyshev':
            x_data = np.array([np.cos( (2*j+1)*np.pi / 2 / (n+1) ) for j in
    range(n+1)])
        y_data = f(x_data)

        interp = np.array([Lagrange_interp(t,x_data,y_data) for t in T])
        error  = F - interp
        ax[0].plot(T,interp,ls[N.index(n)],label=r'$n = %d$'%n)
        ax[1].plot(T,error,ls[N.index(n)])

    ax[0].plot(T,F,'k-.',label='$f(x)$')
    ax[1].axhline(y=0,color='k',linestyle='-.')
    ax[0].set_xlabel(r'$x$',size=20)
    ax[1].set_xlabel(r'$x$',size=20)
    ax[0].set_ylabel(r'$y$',size=20)
    ax[1].set_ylabel(r'$e(x)$',size=20)
    ax[0].tick_params(axis='both',which='major',labelsize=20,direction='in'
    )
    ax[1].tick_params(axis='both',which='major',labelsize=20,direction='in'
    )
    ax[0].legend(fontsize=20,frameon=False)
    plt.tight_layout()
    plt.savefig(save_path,bbox_inches='tight')

if __name__ == '__main__':

    f = lambda x: 1.0/(1.0 + 25.0*x**2.0)
    T = np.linspace(-1.0,1.0,100)
    N  = [5,10,20]

    gen_plot(f,T,N,'fig1.png',nodes='uniform')
    gen_plot(f,T,N,'fig2.png',nodes='chebyshev')

    uniform_nodes   = []
    chebyshev_nodes = []
    for n in N:
            uniform_nodes.append(np.linspace(-1.0,1.0,n+1))
            chebyshev_nodes.append(np.array([np.cos( (2*j+1)*np.pi / 2 / (n
    +1) ) for j in range(n+1)]))

    fix,ax = plt.subplots(nrows=1,ncols=1,figsize=(7,5))
    for i in range(len(N)):
        y = -1*np.ones(N[i] + 1) + i
        y1 = y - 0.1
        y2 = y + 0.1

        ax.plot(uniform_nodes[i],y1,'k.')
        ax.plot(chebyshev_nodes[i],y2,'r.')
    ax.plot([],[],'r.',label='chebyshev')
    ax.plot([],[],'k.',label='uniform')
    for i in range(len(N)):
        ax.text(1.1,-1+i,r'$n=%d$'%N[i],fontsize=20)
    ax.set_xlim(right=1.5)
    ax.set_ylim(top=2.0)
    ax.set_xlabel(r'$x$',size=20)
    ax.tick_params(axis='both',which='major',labelsize=20,direction='in')
    ax.legend(loc='upper center',fontsize=15)
    plt.savefig('fig3.png',bbox_inches='tight')
```



b) Describe what you observe numerically in (a) and Google **Runge's phenomenon** to study the reason.

It is seen that as the degree of the polynomial becomes larger, the interpolation becomes less stable at the end points. That is, the polynomial develops a stronger oscillating behavior in order to match at the increasing number of nodes. At the end points this leads to larger values in the function evaluation since the maximum error, which is proportional to $f^{(n+1)}(\xi)$, becomes unbounded.

c) Repeat Part (a) but interpolate at the $n+1$ Chebyshev nodes over $[-1,1]$. Describe what you observe numerically.

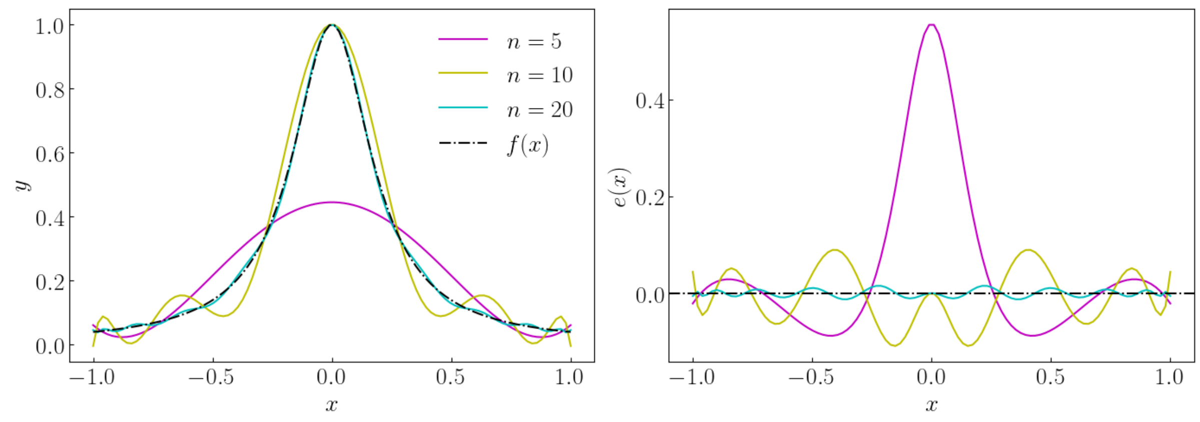
The Lagrange interpolation is repeated using Chebyshev nodes on the given interval:

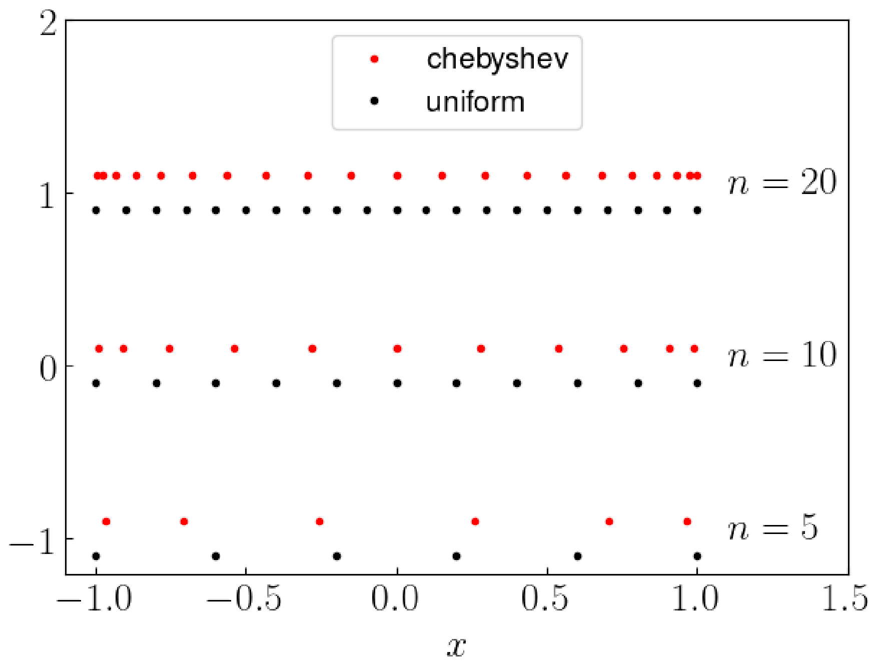
$$x_j = \frac{a+b}{2} + \frac{b-a}{2}\cos\left[\frac{(2j+1)\,\pi}{2\,(n+1)}\right].$$

for $j = 0,\ 1, \ldots,\ n$.

It is clear that the interpolation is more stable across the entire interval. This is because the points become closer together at the end points, meaning that the function cannot vary as much at the boundaries. That is, the error term does not become unbounded since the derivatives are not as large.

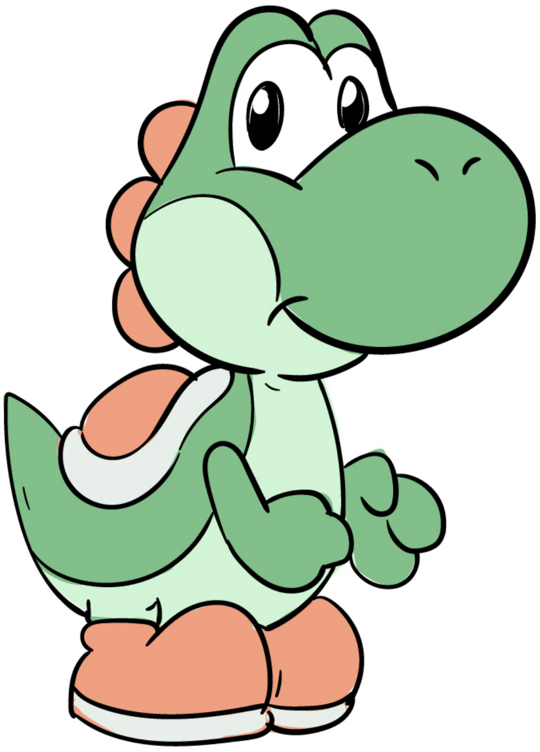
A plot of the distribution of x data points is shown below the interpoation to illustrate the difference between the Chebyshev distribution against the uniform distribution of points.

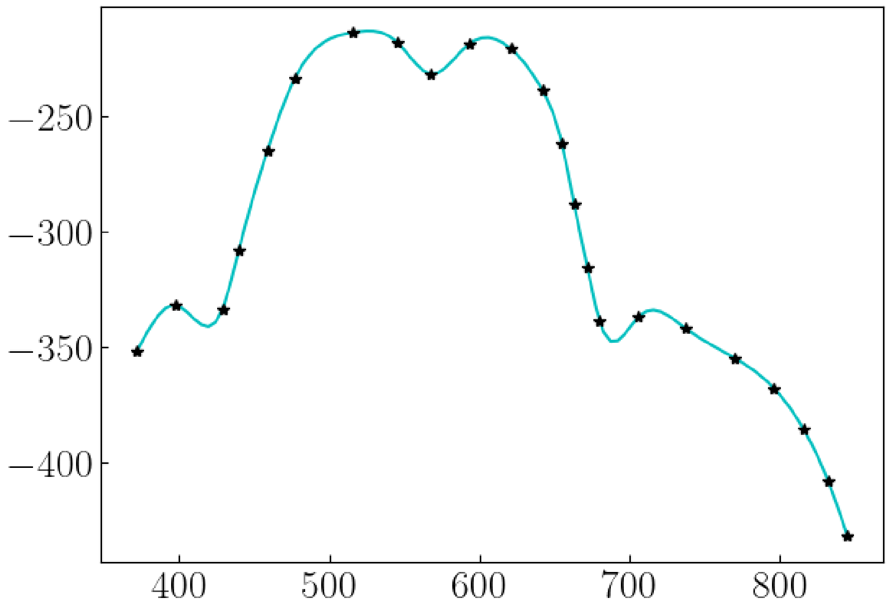




2) Find one picture with some object and reconstruct the shape with cubic spline reconstruction.

For this problem, I used the cubic spline code, translated into python, to reproduce the top outline on the yoshi (insert super mario bros sound effect here!) image below. It is seen that the cubic spine reconstruction is fairly good. That is, it generally matches the shape of the curve, but since spline fits are meant to possess a high degree of smoothness, it does not inherit the cusps (i.e. the places where the derivative is discontinuous) of the original image.





```
#!/usr/bin/env python3

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['text.latex.preamble'] = r'\usepackage{amsmath}'
rcParams['text.usetex']         = True
```

```
rcParams['font.family']         = 'sans-serif'
rcParams['font.sans-serif']     = ['Helvetica']

import csv
from scipy.interpolate import interp1d

def spline(dataX,dataY,t):
    a,b,c,d = spline_coeff(dataX,dataY)
    for i in range(len(dataX)-1):
        if dataX[i] <= t and dataX[i+1] >= t:
            k = i
            break
    xk = dataX[k]
    y = a[k] + b[k]*(t-xk) + c[k]*(t-xk)**2.0 + d[k]*(t-xk)**3.0
    return y

def spline_coeff(dataX,dataY,printA=False):
    n = len(dataX) - 1
    h = np.zeros(n)
    for i in range(n):
        h[i] = dataX[i+1] - dataX[i]
    A = np.zeros([n+1,n+1])
    A[0,0]    = 1.0
    A[-1,-1] = 1.0
    for i in range(1,n):
        A[i,i-1] = h[i-1]
        A[i,i+1] = h[i]
        A[i,i]   = 2.0*(h[i]+h[i-1])
    bb = np.zeros(n+1)
    for i in range(1,n):
        bb[i] = 3.0/h[i]*(dataY[i+1]-dataY[i]) - 3.0/h[i-1]*(dataY[i]-dataY
    [i-1])
    c = np.linalg.solve(A,bb)
    a = np.copy(dataY)
    b = np.zeros(n)
    d = np.zeros(n)
    for i in range(n):
        b[i] = 1.0/h[i]*(a[i+1]-a[i]) - h[i]/3.0*(2.0*c[i]+c[i+1])
        d[i] = (c[i+1]-c[i])/3.0/h[i]
    a = a[:-1]
    c = c[:-1]
    return a,b,c,d

if __name__ == '__main__':

    file = open('prob5.txt')

    csvreader = csv.reader(file)
    header = []
    header = next(csvreader)

    x_data = []
    y_data = []
    for row in csvreader:
        x_data.append(float(row[0]))
        y_data.append(float(row[1]))
    file.close()
    x_data = np.array(x_data)
    y_data = -1.0 * np.array(y_data)

    T = np.linspace(min(x_data),max(x_data),100)
    reconstruct = np.array([spline(x_data,y_data,t) for t in T])
    f = interp1d(x_data,y_data,kind='cubic')

    a,b,c,d = spline_coeff(x_data,y_data,True)
    for i in range(len(a)):
        print('%.5f  %.5f  %.5f  %.5f'%(a[i],b[i],c[i],d[i]))

    fig,ax = plt.subplots(nrows=1,ncols=1,figsize=(7,5))
    ax.plot(T,reconstruct,'c')
    ax.plot(x_data,y_data,'k*')
    ax.tick_params(axis='both',which='major',labelsize=20,direction='in')
    plt.savefig('./fig4.png',bbox_inches='tight')
```