CS450 HW5

For all the code below I imported the following packages

```
import numpy as np
import numpy.linalg as la
import seaborn
import matplotlib.pyplot as plt
import scipy.interpolate as inter
%matplotlib inline
```

Problem 1 : (15 points)

Given the three data points (-1, 1), (0, 0), (1, 1), determine the interpolating polynomial of degree two:
(a) Using the monomial basis
(b) Using the Lagrange basis
(c) Using the Newton basis
Show that the three representations give the same polynomial.

(a):

$$\begin{bmatrix} 1 & -1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$y = x^2$$

(b):
Three Lagrange basis

$$\begin{aligned} p_1 &= \frac{x(x+1)}{(-1)(-1-1)} \\ p_2 &= \frac{(x-1)(x+1)}{(0+1)(0-1)} \\ p_3 &= \frac{x(x-1)}{(1)(1+1)} \end{aligned}$$

$$y = \frac{1}{2}x(x-1) + 0 + \frac{1}{2}(x+1)x = x^2$$

(c):

$$\begin{aligned} \pi_0 &= a \\ \pi_1 &= b(x+1) + a \\ \pi_2 &= c(x+1)(x) + b(x+1) + a \end{aligned}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 2 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

Three Newton's basis

$$y = 1 - (x+1) + (x+1)x$$

Problem 2: (10 points)

(a) Determine the polynomial interpolant to the following data using the monomial basis.

t	1	2	3	4
y	11	29	65	125

Using monomial, we would create solve 4 equations

$$a_3x^3 + a_2x^2 + a_1x + a_0 = y$$

```
def monomial(points):
    x = np.fromiter(map(lambda x: x[0], points), dtype=np.float)
    y = np.fromiter(map(lambda x: x[1], points), dtype=np.float)
    n = len(points)
    A = np.zeros((n,n))
    for i in range(n):
        A[:,i] = x**i
    coe = la.solve(A, y)
    return np.poly1d(coe[::-1])

p = monomial([(1,11),(2,29),(3,65),(4,125)])
print(p)
```

The result is

```
   3     2
1 x + 3 x + 2 x + 5
```

$f(x) = x^3 + 3x^2 + 2x + 5$

(b) Determine the Lagrange polynomial interpolant to the same data and show that the resulting polynomial is equivalent to that obtained in part a.

Using the different basics, in the form

$$\sum_i y_i \prod_j^{i \neq j} \frac{x - x_j}{x_i - x_j}$$

Thus, the following code,

```
def lagrange(points):
    x = np.fromiter(map(lambda x: x[0], points), dtype=np.float)
    y = np.fromiter(map(lambda x: x[1], points), dtype=np.float)
    n = len(points)
    f = np.poly1d([])
    for i in range(n):
        fi = np.poly1d([y[i]])
        for j in range(n):
            if i!=j:
                fi *= np.poly1d([1, -x[j]])/(x[i]-x[j])
        f += fi
    return f
p = lagrange([(1,11),(2,29),(3,65),(4,125)])
print(p)
```

The result:

```
   3     2
1 x + 3 x + 2 x + 5
```

$f(x) = x^3 + 3x^2 + 2x + 5$

Problem 3: (15 points)

Use the error bound from Section 7.3.5 to estimate the maximum error in interpolating the function sin(t) by a polynomial of degree four using five equally spaced points on the interval [0,π/2].Check your bound by comparing a few sample values of the function and the interpolant. How many points would be required to achieve a maximum error of 10^{-10}?

The bound of the error was given by the following fomula:

$$f(t) - p_{n-1}(t) \leq \frac{f^{(n)}(\theta)}{n!}(t - t_1)(t - t_2)\ldots(t - t_n)$$

And in this case, there are 5 points and they are $0, \pi/8, \pi/4, 3\pi/8, \pi/2$,

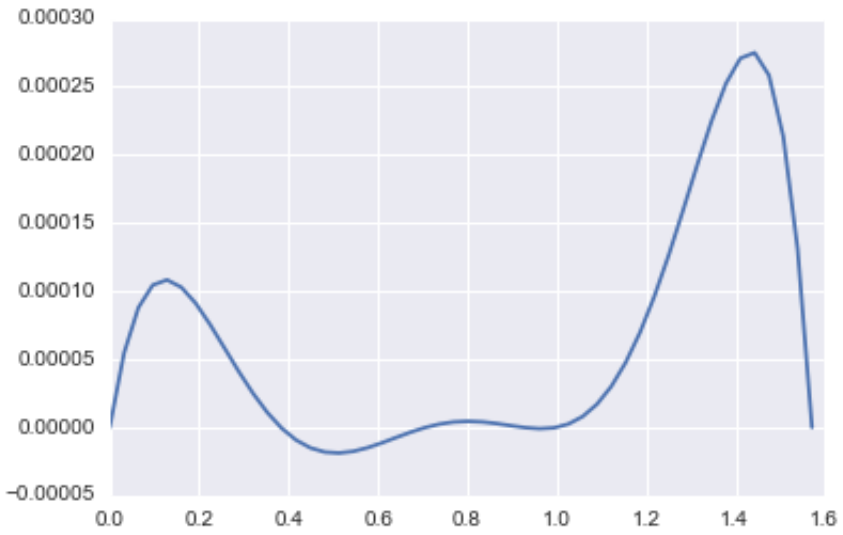
$$f(t) - p_{n-1}(t) \leq \frac{cos(t)}{5!} \prod_{k=0}^{4} (t - k\pi)$$

It is not possible to get an analytical solution, thus, we might want to loose some bound, thus, $cos(t) < 1$,
Thus,

$$max|f(t) - p_{n-1}(t)| \leq \frac{Mh^n}{4n} < \frac{(\pi/8)^5}{20} \approx 0.00046695$$

And if we draw the graph of the function, we will get

```
import operator

p = np.poly1d([1/120.0])
for i in range(5):
    p*= np.poly1d([1,-np.sin(i*np.pi/8.0)])

f = lambda n: np.cos(n)*p(n)

x = np.linspace(0, np.pi/2)
plt.plot(x, f(x))
```



To reach 10^{-10}, we need $\frac{(\pi/n)^n}{n} < 10^{-10}$
We get the smallest integer to be 14. Thus, we need 13 degree polynomial for error to be less than 10^{-10}

Problem 4: (10 points)

Suppose that Lagrange interpolation at a given set of nodes $x_1, \ldots, x_n$ is used to derive a quadrature rule. Prove that the corresponding weights are given by the integrals of the Lagrange basis functions, $w_i = \int_a^b l_i(x)dx, i = 1, \ldots, n$.

the Lagrange polynomial interpolating the
corresponding values of the integrand function f is

$$p(x) = \sum_{i=1}^{n} f(x_i) l_i(x)$$

We then have

$$\int_a^b p(x)dx = \int_a^b \sum_{i=1}^{n} f(x_i) l_i(x)dx = \sum_{i=1}^{n} f(x_i) \int_a^b l_i(x)dx$$

according to de definition of quadrature rules, the weights are

$$w_i = \int_a^b l_i(x)dx$$

Problem 5: (15 points)

The forward difference formula

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

and the backward difference formula

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

are both first-order accurate approximations to the first derivative of a function f : R → R. What order accuracy results if we average these two approximations? Support your answer with an

error analysis.

The function then becomes

$$\tilde{f}'(x) = \frac{f(x+h) - f(x-h)}{2h}$$

If we do a taylor approximation around x point for f,
we will get

$$f(x+h) = f(x) + f'(x)h + f''(x)h^2/2 + f'''(x)h^3/6 + O(h^4)$$

$$f(x-h) = f(x) - f'(x)h + f''(x)h^2/2 - f'''(x)h^3/6 + O(h^4)$$

$$\tilde{f}'(x) = f'(x) + h^2 f'''(x)/6 + O(h^4)$$

As we can see, the order of accuracy becomes the second order.

Problem 6: (15 points)

Suppose that the first-order accurate, forward difference approximation to the derivative of a function at a given point produces the value -0.8333 for h = 0.2 and the value -0.9091 for h = 0.1. Use Richardson extrapolation to obtain a better approximate value for the derivative.

So, we get the following data

$$\begin{aligned} A(x+h) &= A(x) + Kh + O(h^2) \\ A(x+2h) &= A(x) + 2Kh + O(h^2) \\ A(0.2) &= -0.8333 \\ A(0.1) &= -0.9091 \\ B(0.2) &= \frac{2A(0.1) - A(0.2)}{2-1} \end{aligned}$$

B(0.2) = 0.9849
Thus, it is a better approximated value for the derivative.

Problem 7: (20 points)

Compute the interpolant of the function

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

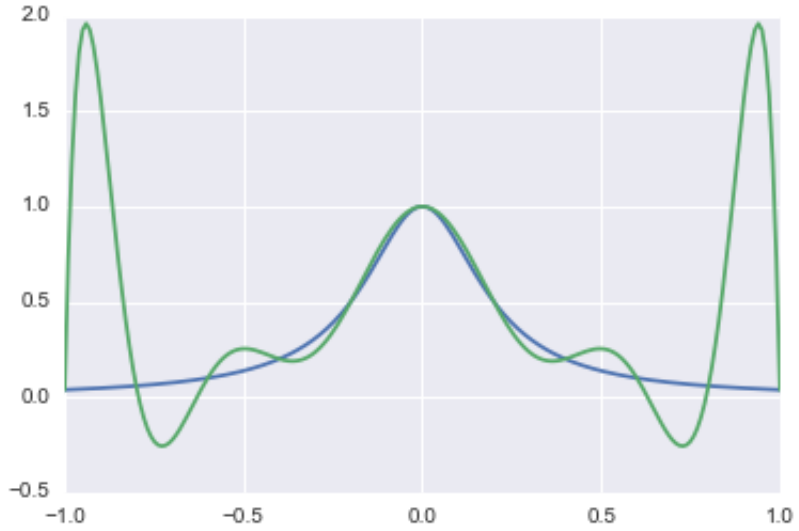
on the interval [−1, 1] using

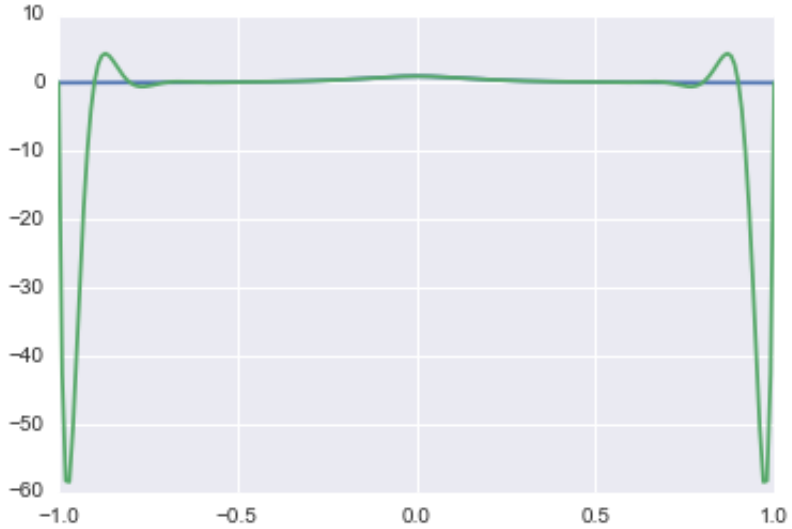
(a) Lagrange polynomials with uniform points
(b) Cubic splines with uniform points
(c) Lagrange polynomials with n Chebyshev points (i.e. $x_i = cos\theta_i$ with $\theta_i = \frac{2i-1}{n}\pi, where i = 1, \cdots, n$)

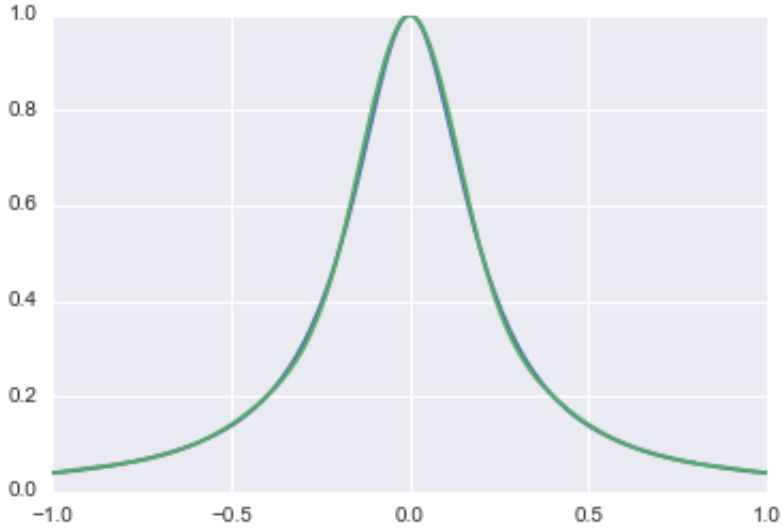
Compare your results graphically by plotting both interpolants and the original function for n = 11 and n = 21.

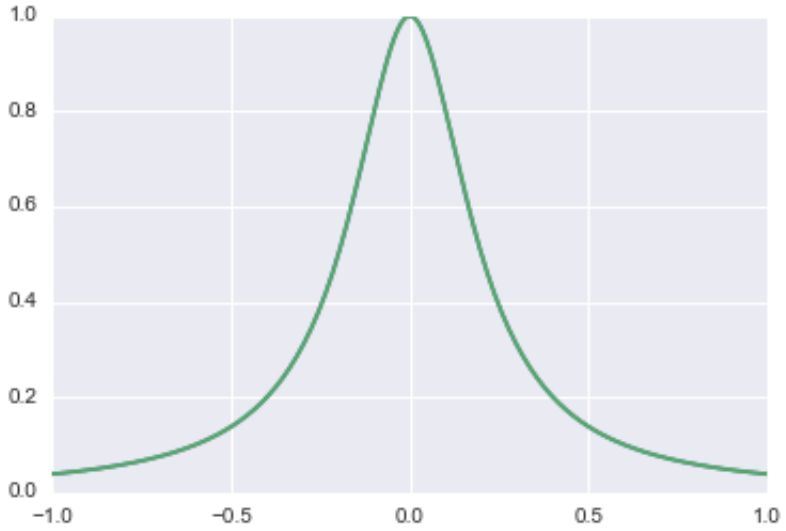
```
def plot(f, _f, start=-1, end=1):
    x = np.linspace(start, end, 200)
    y = f(x)
    plt.plot(x,y)
    y = _f(x)
    plt.plot(x,y)
    plt.show()

f = lambda x: 1/(1+25*(x**2))

# n = 11 and linear Lagrange
n = 11
xs = np.linspace(-1,1, n)
ys = f(xs)
p = inter.lagrange(xs, ys)
plot(f, p)

# n = 21 and linear Lagrange
n = 21
xs = np.linspace(-1,1, n)
ys = f(xs)
```

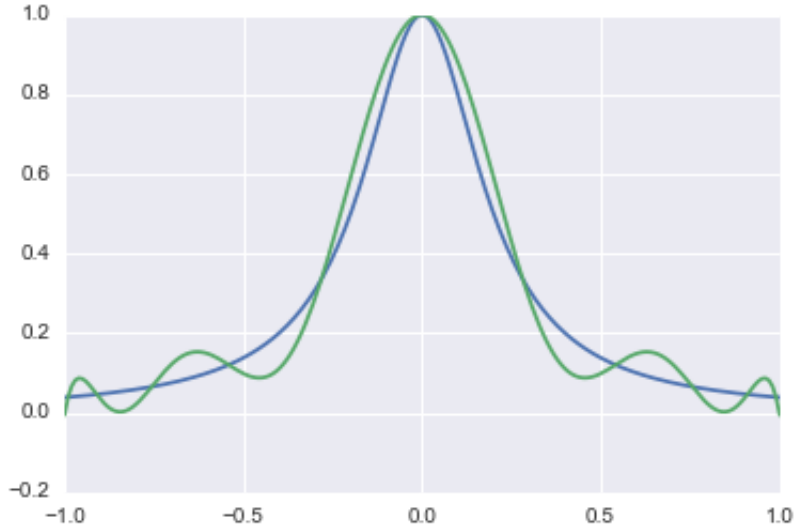
```
p = inter.lagrange(xs, ys)
plot(f, p)

# n = 11 and cubic spline
n = 11
xs = np.linspace(-1,1, n)
ys = f(xs)
tck = inter.splrep(xs, ys)
s = lambda x: inter.splev(x, tck)
plot(f, s)

# n = 21 and cubic spline
n = 21
xs = np.linspace(-1,1, n)
ys = f(xs)
tck = inter.splrep(xs, ys)
s = lambda x: inter.splev(x, tck)
plot(f, s)

# n = 11 and Chebyshev Lagrange
n = 11
xs = np.cos(np.linspace(1,2*n-1,n)/n/2*np.pi)
ys = f(xs)
p = inter.lagrange(xs, ys)
plot(f, p)

# n = 21 and Chebyshev Lagrange
n = 21
xs = np.cos(np.linspace(1,2*n-1,n)/n/2*np.pi)
ys = f(xs)
p = inter.lagrange(xs, ys)
plot(f, p)
```

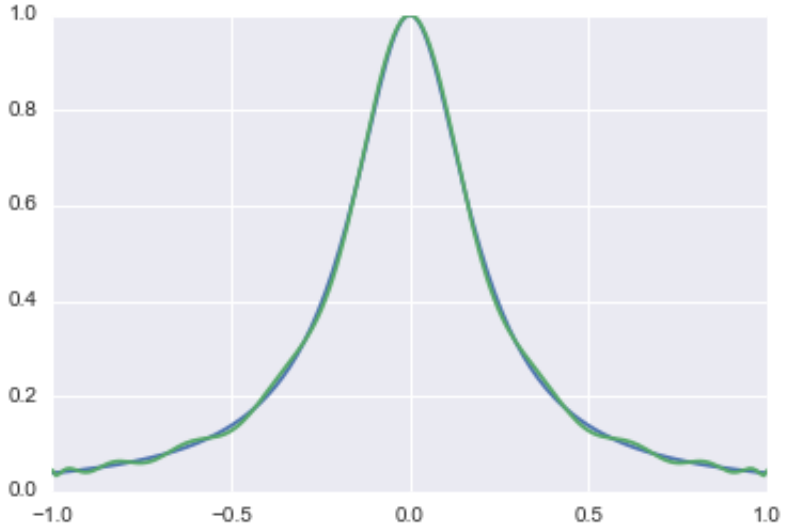

2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
1.0


10
0
−10
−20
−30
−40
−50
−60
−1.0
−0.5
0.0
0.5
1.0


1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0







The most accurate is 21 points cubic spline and the least accurate is the 21 linear lagrange.