PYL800 - Group 1 Presentation

Hriday Sabharwal - 2020PH10697
Khushvind Maurya - 2021MT10238
Asmit Singh - 2021MT10887
Mohit Raj Modi - 2021MT10919

January 15, 2024

Outline

Problem 1

a. Plot y = A.sin(wt)

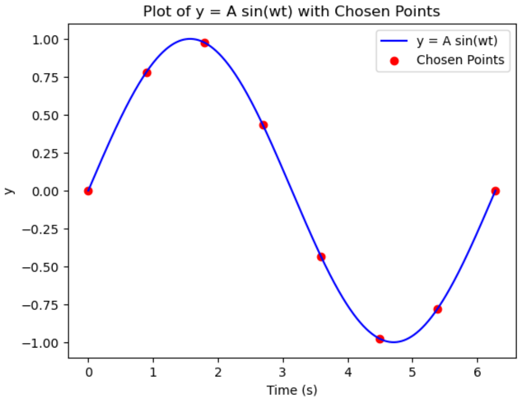

Fig. 1.1

Problem 1

b. Use linear and quadratic splines and c. Plot the spline fitted curve

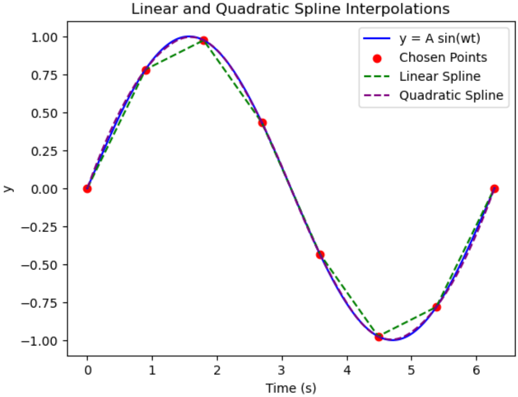

Fig. 1.2

Problem 1

d. Evaluate R^2 with increased data points

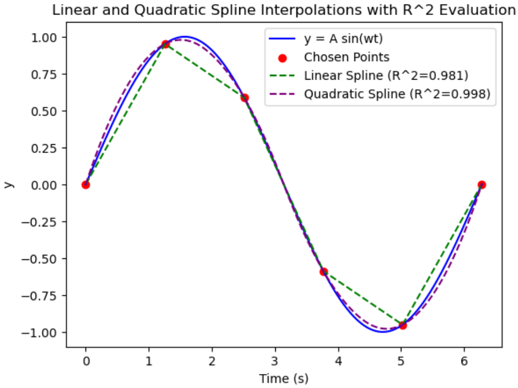

Fig. 1.3

Problem 1 - Code

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
from sklearn.metrics import r2_score

A = 1       # Amplitude
w = 1       # Angular frequency

# Time Period
T = 2 * np.pi / w

# Choose 20 points within one period
t_points = np.linspace(0, T, 6)
inc_t_points = np.linspace(0, T, 20)
y_points = A * np.sin(w * t_points)

t = np.linspace(0, T, 1000)
y = A * np.sin(w * t)
```

Problem 1 - Code

```
# Linear spline interpolation
linear_interp = interp1d(t_points, y_points, kind='linear')

# Quadratic spline interpolation
quadratic_interp = interp1d(t_points, y_points, kind='quadratic')

# R^2 for linear spline
linear_r2 = r2_score(A * np.sin(w * inc_t_points), linear_interp(inc_t_points))

# R^2 for quadratic spline
quadratic_r2 = r2_score(A * np.sin(w * inc_t_points), quadratic_interp(inc_t_po
```

Problem 1 - Code

```
# Plotting
fig, ax = plt.subplots()
ax.plot(t, y, label='y = A sin(wt)', color='blue')
ax.scatter(t_points, y_points, color='red', label='Chosen Points')

ax.plot(t, linear_interp(t), label=f'Linear Spline (R^2={linear_r2:.3f})', line

ax.plot(t, quadratic_interp(t), label=f'Quadratic Spline (R^2={quadratic_r2:.3f

ax.set_xlabel('Time (s)')
ax.set_ylabel('y')
ax.legend()

plt.title('Linear and Quadratic Spline Interpolations with R^2 Evaluation')
plt.show()
```

Problem 2

Linear:

$\vec{P} = \vec{P}_0(1-t) + \vec{P}_1 t$

Quadratic:

$\vec{A} = \vec{P}_0(1-t) + \vec{P}_1 t \;;\; \vec{B} = \vec{P}_1(1-t) + \vec{P}_2 t \;;\; \vec{P} = \vec{A}(1-t) + \vec{B}t$

$\Rightarrow \vec{P} = \vec{P}_0(1-t)^2 + 2\vec{P}_1 t(1-t) + \vec{P}_2 t^2$

$= \vec{P}_0(1-2t+t^2) + \vec{P}_1(2t-2t^2) + \vec{P}_2(t^2)$

Cubic:

$\vec{A} = \vec{P}_0(1-t) + \vec{P}_1 t \;;\; \vec{B} = \vec{P}_1(1-t) + \vec{P}_2 t \;;\; \vec{C} = \vec{P}_2(1-t) + \vec{P}_3 t$

$\vec{D} = \vec{A}(1-t) + \vec{B}t \;;\; \vec{E} = \vec{B}(1-t) + \vec{C}t \;;\; \vec{P} = \vec{D}(1-t) + \vec{E}t$

$\Rightarrow \vec{P} = \vec{A}(1-t)^2 + 2\vec{B}t(1-t) + \vec{C}t^2$

$= \vec{P}_0(1-t)^3 + \vec{P}_1 t(1-t)^2 + 2\vec{P}_1 t(1-t)^2 + 2\vec{P}_2 t^2(1-t) + \vec{P}_2 t^2(1-t) + \vec{P}_3 t^3$

$= \vec{P}_0(1-3t+3t^2-t^3) + \vec{P}_1(3t-6t^2+3t^3) + \vec{P}_2(3t^2-3t^3) + \vec{P}_3(t^3)$

Problem 3

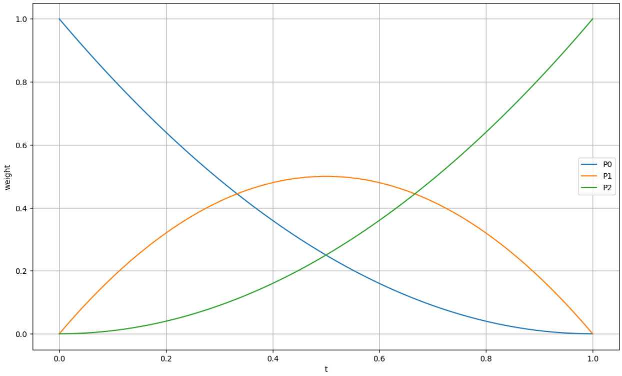

Fig. 3.1 Quadratic weights

Problem 3

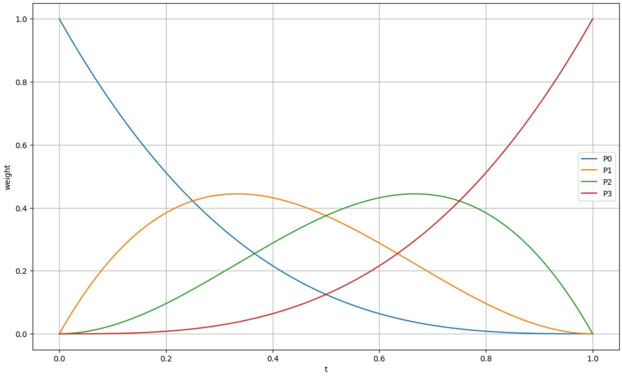

Fig. 3.2 Cubic weights

Problem 3 - Code

```
import matplotlib.pyplot as plt
import numpy as np

#weights for linear curve
wt=[np.polynomial.Polynomial([1,-1]),np.polynomial.Polynomial([0,1])]

#calculate weights for nth order curve
def wts(new_wt,n,ctr=1):
    global wt
    if ctr<n:
        temp=[np.polynomial.Polynomial([0])]*(len(new_wt)+1)
        for i in range(len(wt)):
            for j in range(len(new_wt)):
                temp[i+j]=np.polynomial.polynomial.polyadd(temp[i+j],(np.polyno
        return wts(temp,n,ctr+1)
    return new_wt

t=np.linspace(0,1,num=100)
```

Problem 3 - Code

```
def plot(x,wt,n):
    wt=wts(wt,n)
    for i in range(n+1):
        plt.plot(x,wt[i][0](x),label='P'+str(i))
    plt.legend()
    plt.xlabel('t')
    plt.ylabel('weight')
    plt.grid()
    plt.show()

plot(t,wt,3)
```

Problem 4

It appears that loops come to fruition upon the following condition
$x_2 < x_0 < x_3 < x_1$ or $x_1 < x_3 < x_0 < x_2$
and
y_1, y_2 lie on the same side of the line containing y_0, y_3

Problem 4

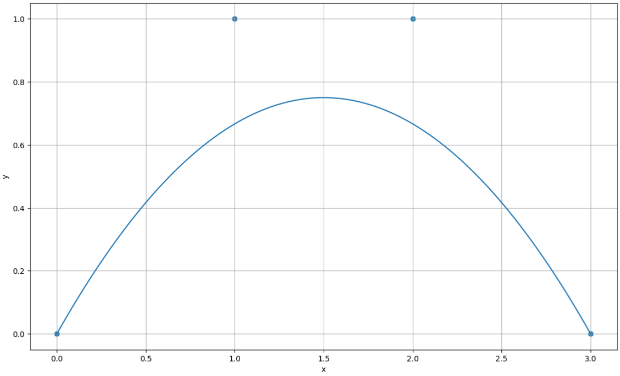

Fig. 4.1 $x_0 < x_1 < x_2 < x_3$

Problem 4

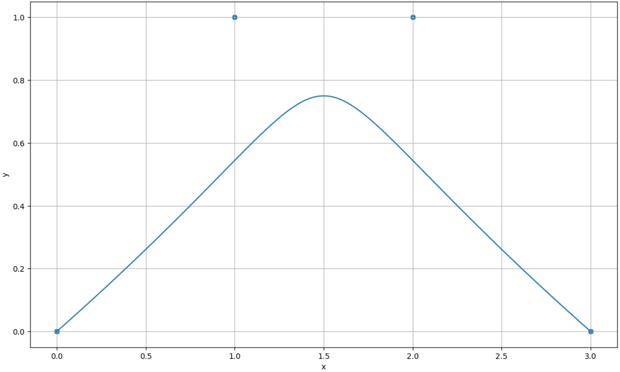

Fig. 4.2 $x_0 < x_2 < x_1 < x_3$

Problem 4

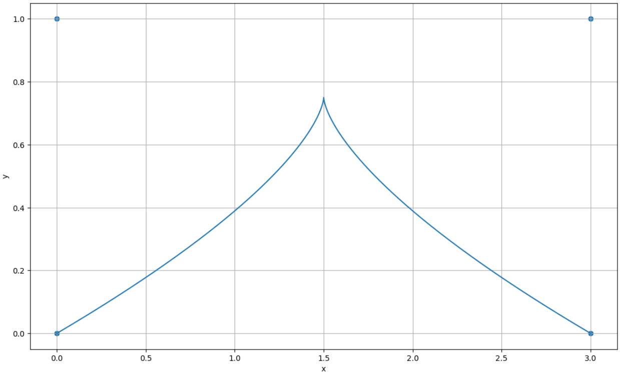

Fig. 4.3 $x_0 = x_2 < x_1 = x_3$

Problem 4

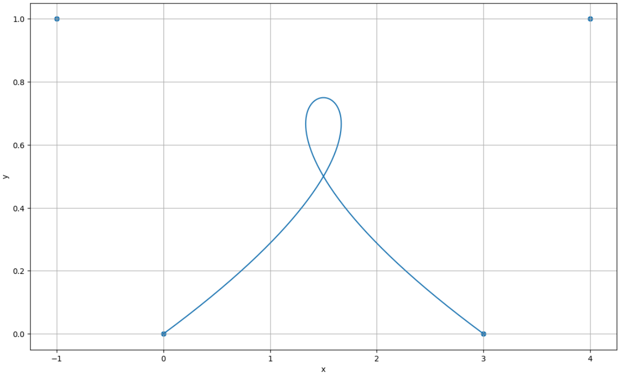

Fig. 4.4 $x_2 < x_0 < x_3 < x_1$

Problem 4

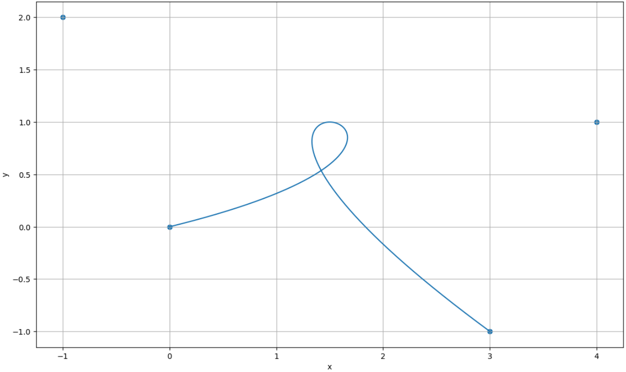

Fig. 4.5 y_1, y_2 lie on same side

Problem 4

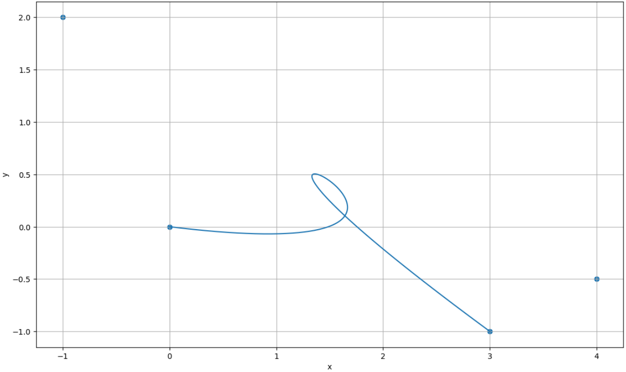

Fig. 4.6 y_1, y_2 lie on same side

Problem 4

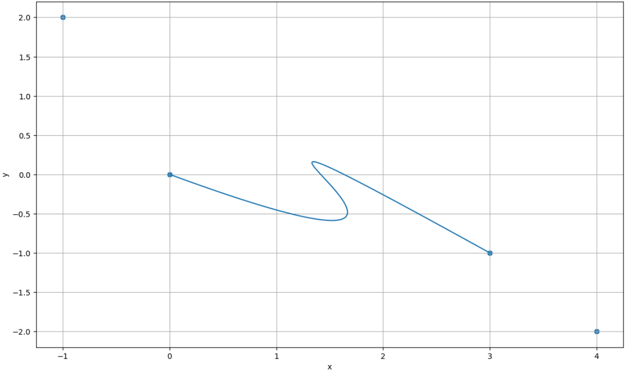

Fig. 4.7 y_1, y_2 lie on opposite sides

Problem 4 - Code

```
import matplotlib.pyplot as plt
import numpy as np

#weights for linear curve
wt=[np.polynomial.Polynomial([1,-1]),np.polynomial.Polynomial([0,1])]

#calculate weights for nth order curve
def wts(new_wt,n,ctr=1):
    global wt
    if ctr<n:
        temp=[np.polynomial.Polynomial([0])]*(len(new_wt)+1)
        for i in range(len(wt)):
            for j in range(len(new_wt)):
                temp[i+j]=np.polynomial.polynomial.polyadd(temp[i+j],(np.polyno
        return wts(temp,n,ctr+1)
    return new_wt

#array containing points P_0, P_1 ... P_n
P=np.array([np.array([0,0]),np.array([4,-2]),np.array([-1,2]),np.array([3,-1])]

t=np.linspace(0,1,num=100)
```

Problem 4 - Code

```
def plot(x,wt,n):          #n=order of curve
    wt=wts(wt,n)
    new_x=np.zeros(len(x))
    new_y=np.zeros(len(x))
    for i in range(n+1):
        new_x=new_x+P[i][0]*wt[i][0](x)
        new_y=new_y+P[i][1]*wt[i][0](x)
    plt.plot(new_x,new_y)
    plt.scatter(np.transpose(P)[0],np.transpose(P)[1])
    plt.grid()
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

plot(t,wt,3)
```