# hw2

February 26, 2020

[10]:

```
!jupyter nbconvert --to script hw2.ipynb --output hw2
!jupyter nbconvert --to pdf hw2.ipynb --output hw2
```

```
[NbConvertApp] Converting notebook hw2.ipynb to script
[NbConvertApp] Writing 4197 bytes to hw2.py
[NbConvertApp] Converting notebook hw2.ipynb to pdf
[NbConvertApp] Writing 39232 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 37404 bytes to hw2.pdf
```

## 1 CMAM hw2

[2]:

```
import numpy as np
from collections import abc

def normalize_vector(vector: abc.Collection) -> np.ndarray:
    return vector/np.linalg.norm(vector)
```

### 1.1 Problem 1

[ ]:

```
```

### 1.2 Problem 2

[ ]:

```
```

### 1.3 Problem 3

[3]:

```
from functools import reduce
from collections import abc
from typing import Union
```

```
import numpy as np
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt
%matplotlib widget

class HermiteCubicCurve():
    """An implementation of a Hermite Cubic Curve from two end points and their␣
 ↪respective tangent vectors."""

    P0: abc.Collection
    M0: abc.Collection
    P1: abc.Collection
    M1: abc.Collection
    B: np.ndarray

    M_sub_H = np.array((
        (2, -2, 1, 1),
        (-3, 3, -2, -1),
        (0, 0, 1, 0),
        (1, 0, 0, 0)
    ))

    def __init__(self, P0: abc.Collection, M0: abc.Collection, P1: abc.
 ↪Collection, M1: abc.Collection):
        self.P0, self.M0, self.P1, self.M1 = P0, M0, P1, M1
        self.B = np.array((P0, P1, M0, M1))

    def get_point_from_u_value(self, u: Union[int, float]) -> np.ndarray:
        U = np.array((u**3, u**2, u, 1)).transpose()
        P = reduce(np.dot, (U, self.MsubH, self.B))
        return P

    def get_points(self, n: int = 50) -> np.ndarray:
        step = 1.0/n
        u = np.arange(0, 1+step, step)
        U = np.array((u**3, u**2, u, np.ones(len(u)))).transpose()
        P = reduce(np.dot, (U, self.M_sub_H, self.B))
        return P

    def get_tangent_vector_from_u_value(self, u: Union[int, float]) -> np.
 ↪ndarray:
        U_sup_u = np.array((3*u**2, 2*u, 1, 0)).transpose()
        V = reduce(np.dot, (U_sup_u, self.M_sub_H, self.B))
        return V

    def get_tangent_vectors(self, n: int = 50) -> np.ndarray:
        step = 1.0/n
        u = np.arange(0, 1+step, step)
        U_sup_u = np.array((3*u**2, 2*u, np.ones(len(u)), 0)).transpose()
        V = reduce(np.dot, (U_sup_u, self.M_sub_H, self.B))
        return V

##### PART A #####
P1 = np.array([4, 2, 6])
M1 = np.array([3, 1, -1])
P2 = np.array([2, 8, 4])
M2 = np.array([-1, 1, -1])

curve1 = HermiteCubicCurve(P1, M1, P2, M2)
points1 = curve1.get_points()

fig = plt.figure()
ax = plt.axes(projection = "3d")
ax.plot3D(points1[:, 0], points1[:, 1], points1[:, 2])
ax.set_title("Problem 3a - Hermite Cubic Curve")
plt.show()

VsupU = curve1.get_tangent_vector_from_u_value(.6)
VsupUsubNorm = normalize_vector(VsupU)
print(f"Promblem 3a - Tangent vector = [{VsupU[0]:.3}, {VsupU[1]:.3}, {VsupU[2]:
 ↪.3}], Unit tangent vector = [{VsupUsubNorm[0]:.3}, {VsupUsubNorm[1]:.3},␣
 ↪{VsupUsubNorm[2]:.3}]")

##### PART B #####
P3 = P2.copy() # C1 continuity
M3 = M2.copy() # C1 continuity
P4 = np.array([-2, 5, 4])
M4 = np.array([1, 2, -1])

curve2 = HermiteCubicCurve(P3, M3, P4, M4)
points2 = curve2.get_points()

fig = plt.figure()
ax = plt.axes(projection = "3d")
ax.plot3D(points1[:, 0], points1[:, 1], points1[:, 2])
ax.plot3D(points2[:, 0], points2[:, 1], points2[:, 2])
ax.set_title("Problem 3b - Joint Hermite Cubic Curves with C1 Continuity")
plt.show()
```

```
Canvas(toolbar=Toolbar(toolitems=[('Home', 'Reset original view', 'home', 'home'), ('Back', 'Ba
```

```
Promblem 3a - Tangent vector = [-3.72, 8.2, -2.44], Unit tangent vector =
[-0.399, 0.879, -0.262]
```

```
Canvas(toolbar=Toolbar(toolitems=[('Home', 'Reset original view', 'home', 'home'), ('Back', 'Ba
```

### 1.4 Problem 4

[9]:

```
from collections import abc
import numpy as np
from typing import Union

class BezierCurve():
    """An implementation of a Bezier Curve from an arbitrary number of control␣
↪points. Algorithm based on Bernstein Polynomial (implementation from https://
↪web.mit.edu/hyperbook/Patrikalakis-Maekawa-Cho/node9.html)"""
    # adapt for DeCasteljau's Algo https://web.mit.edu/hyperbook/
↪Patrikalakis-Maekawa-Cho/node13.html

    P_sub_I: np.ndarray
    n: int

    def __init__(self, *args: abc.Collection):
        self.P_sub_I = np.array(args)
        self.n = len(args) - 1

    def __solve_Bernstein_Polynomial_value(self, i, n, u):
        if (n == self.n - 1 and (i == -1 or i == self.n)): # n can only equal␣
↪self.n - 1 during tangent case
            return 0
        A_term = np.math.factorial(n)/(np.math.factorial(i)*np.math.
↪factorial(n-i))
        B_term = (1-u)**(n-i)
        C_term = u**i
        return A_term * B_term * C_term

    def get_point_from_u_value(self, u: Union[int, float]) -> np.ndarray:
        P = np.zeros(len(self.P_sub_I[0]))
        for i in range(self.n+1):
            B = self.__solve_Bernstein_Polynomial_value(i, self.n, u)
            P += self.P_sub_I[i]*B
        return P

    def get_points(self, n: int = 50) -> np.ndarray:
        step = 1.0/n
        u = np.arange(0, 1+step, step)
        P = np.array([self.get_point_from_u_value(u_val) for u_val in u])
        return P
    def get_tangent_vector_from_u_value(self, u: Union[int, float]) -> np.ndarray:
        V = np.zeros(len(self.P_sub_I[0]))
        for i in range(self.n+1):
            dB_over_dU = self.n * (self.__solve_Bernstein_Polynomial_value(i-1, self.n-1, u) - self.__solve_Bernstein_Polynomial_value(i, self.n-1, u))
            V += self.P_sub_I[i]*dB_over_dU
        return V

    def get_tangent_vectors(self, n: int = 50) -> np.ndarray:
        step = 1.0/n
        u = np.arange(0, 1+step, step)
        V = np.array([self.get_tangent_vector_from_u_value(u_val) for u_val in u])
        return V

##### PART A #####
A = np.array([1, 1, 1])
B = np.array([1, 3, 3])
C = np.array([3, 2, 1])
E = np.array([2, 4, 2])

curve1 = BezierCurve(A, B, C, E)
points1 = curve1.get_points()

fig = plt.figure()
ax = plt.axes(projection = "3d")
ax.plot3D(points1[:, 0], points1[:, 1], points1[:, 2])
ax.set_title("Problem 4a - Bezier Curve")
plt.show()

##### PART B #####
D = C.copy()

curve2 = BezierCurve(A, B, C, D, E)
points2 = curve2.get_points()

fig = plt.figure()
ax = plt.axes(projection = "3d")
ax.plot3D(points2[:, 0], points2[:, 1], points2[:, 2])
ax.set_title("Problem 4b - Bezier Curve with Duplicate C")
plt.show()

print(f"Promblem 4b - Degree of curve = {curve2.n}")
U = np.array([0, .5, .75, 1])
for u_val in U:
    VsupU = curve2.get_tangent_vector_from_u_value(u_val)
    print(f"Promblem 4b - Tangent vector @ {u_val} = [{VsupU[0]:.3}, {VsupU[1]:.
↪3}, {VsupU[2]:.3}]")

##### PART C #####
```

```
Canvas(toolbar=Toolbar(toolitems=[('Home', 'Reset original view', 'home', 'home'), ('Back', 'Ba
```

```
Canvas(toolbar=Toolbar(toolitems=[('Home', 'Reset original view', 'home', 'home'), ('Back', 'Ba
```

```
Promblem 4b - Degree of curve = 4
Promblem 4b - Tangent vector @ 0.0 = [0.0, 8.0, 8.0]
Promblem 4b - Tangent vector @ 0.5 = [2.5, 0.5, -1.5]
Promblem 4b - Tangent vector @ 0.75 = [-0.562, 2.94, 0.688]
Promblem 4b - Tangent vector @ 1.0 = [-4.0, 8.0, 4.0]
```

### 1.5 Problem 5

[ ]:

```

```