Math 151A Homework #5

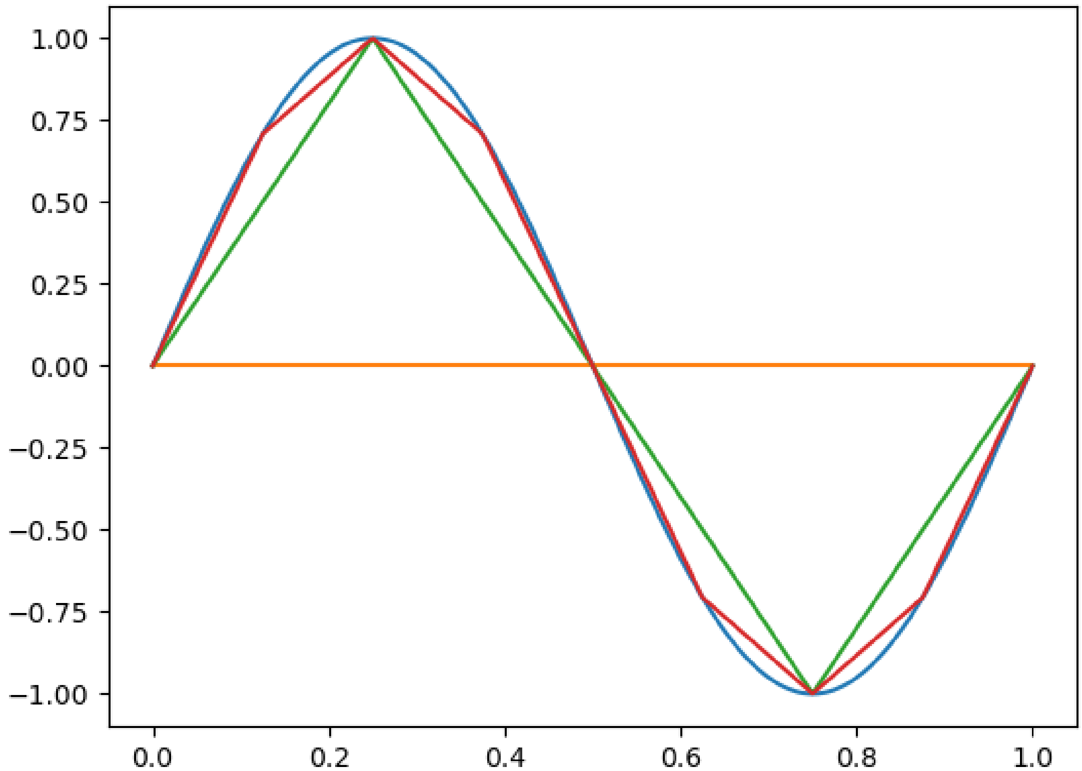
Nathan Solomon

November 15, 2024

Problem 0.1.

The formulas for $P_{1,2}, P_{1,4}$ are given by this Python code:

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

def f(x):
    return math.sin(2 * math.pi * x)

def P(n, x):
    m = math.floor(n * x)
    fract = n * x - m
    return f(m / n) + fract * (f((m+1) / n) - f(m / n))

x_data = np.linspace(0, 1, 400)
plt.plot(x_data, [f(x) for x in x_data])
for n in [2, 4, 8]:
    y_data = [P(n, x) for x in x_data]
    plt.plot(x_data, y_data)
plt.show()
```



Even in the supremum norm, $\lim_{n\to\infty} |f(x) - P_{1,n}| = 0$, because $P_{1,n}$ converges uniformly to f (in the sup norm).

Problem 0.2.

A degree 3 polynomial has the form $f(x) = ax^3 + bx^2 + cx + d$, where a, b, c, d are real numbers and $a \neq 0$ (because if $a = 0$, the degree of the polynomial would be less than 3). Let $\{x_0, x_1, x_2, x_3\}$ be four points in the domain of f, satisfying $x_0 < x_1 < x_2 < x_3$. f is its own natural clamped cubic spline interpolant, because $f(x_i) = f(x_i)$ and $f'(x_i) = f'(x_i)$ for any $i \in \{0, 1, 2, 3\}$. However, f is not its own natural cubic spline, because $a \neq 0$ implies f'' is either linearly increasing or linearly decreasing, which means $f''(x_0)$ and $f''(x_n)$ cannot both be zero.

Problem 0.3.

$$s(x) := \begin{cases} s_0(x) := a_0x^3 + b_0x^2 + c_0x + d_0 & 0.1 \le x < 0.2 \\ s_1(x) := a_1x^3 + b_1x^2 + c_1x + d_1 & 0.2 \le x \le 0.3 \end{cases}$$

Now to find these 8 constants, we need 8 equations:

$$\begin{aligned}
-0.29004996 &= 0.001a_0 + 0.01b_0 + 0.1c_0 + d_0 \\
-0.56079734 &= 0.008a_0 + 0.04b_0 + 0.2c_0 + d_0 \\
-0.56079734 &= 0.008a_1 + 0.04b_1 + 0.2c_1 + d_1 \\
-0.81401972 &= 0.027a_1 + 0.09b_1 + 0.3c_1 + d_1 \\
s''(0.1) = 0 \Rightarrow 0 &= 0.6a_0 + 2b_0 \\
s''(0.3) = 0 \Rightarrow 0 &= 1.8a_1 + 2b_1 \\
s_0'(0.2) = s_1'(0.2) \Rightarrow 0 &= 0.12a_0 + 0.4b_0 + c_0 - 0.12a_1 - 0.4b_1 - c_1 \\
s_0''(0.2) = s_1''(0.2) \Rightarrow 0 &= 1.2a_0 + 2b_0 - 1.2a_1 - 2b_1
\end{aligned}$$

Here is the code and output for the rest of this question. Note that $f'(0.2) = s'(0.2)$, which is a result of the definition of a cubic spline. Osculating polynomial interpolation would also ensure this is true.

```
import scipy.integrate as integrate
import numpy as np

A = np.matrix([
    [.001, .01, .1, 1,     0,   0,  0, 0],
    [.008, .04, .2, 1,     0,   0,  0, 0],
    [   0,   0,  0, 0,  .008, .04, .2, 1],
    [   0,   0,  0, 0,  .027, .09, .3, 1],
    [  .6,   2,  0, 0,     0,   0,  0, 0],
    [   0,   0,  0, 0,   1.8,   2,  0, 0],
    [ .12,  .4,  1, 0,  -.12, -.4, -1, 0],
    [ 1.2,   2,  0, 0,  -1.2,  -2,  0, 0]
])
B = np.matrix([[-0.29004996, -0.56079734, -0.56079734, -0.81401972, 0, 0, 0, 0]]).T

X = np.linalg.inv(A) @ B
a_0 = X[0, 0]
b_0 = X[1, 0]
c_0 = X[2, 0]
d_0 = X[3, 0]
a_1 = X[4, 0]
b_1 = X[5, 0]
c_1 = X[6, 0]
d_1 = X[7, 0]
print(f"{a_0=}\n{b_0=}\n{c_0=}\n{d_0=}\n\n{a_1=}\n{b_1=}\n{c_1=}\n{d_1=}")

def f(x):
    return x**2 * np.cos(x) - 3 * x
def f_prime(x):
    return 2 * x * np.cos(x) - x**2 * np.sin(x) - 3
def s(x):
    if x < 0.2:
        return a_0 * x**3 + b_0 * x**2 + c_0 * x + d_0
    return     a_1 * x**3 + b_1 * x**2 + c_1 * x + d_1
def s_prime(x):
    if x < 0.2:
        return 6 * a_0 * x**2 + 2 * b_0 * x + c_0
    return     6 * a_1 * x**2 + 2 * b_1 * x + c_1

print(f"\n{f(0.18)=}")
```

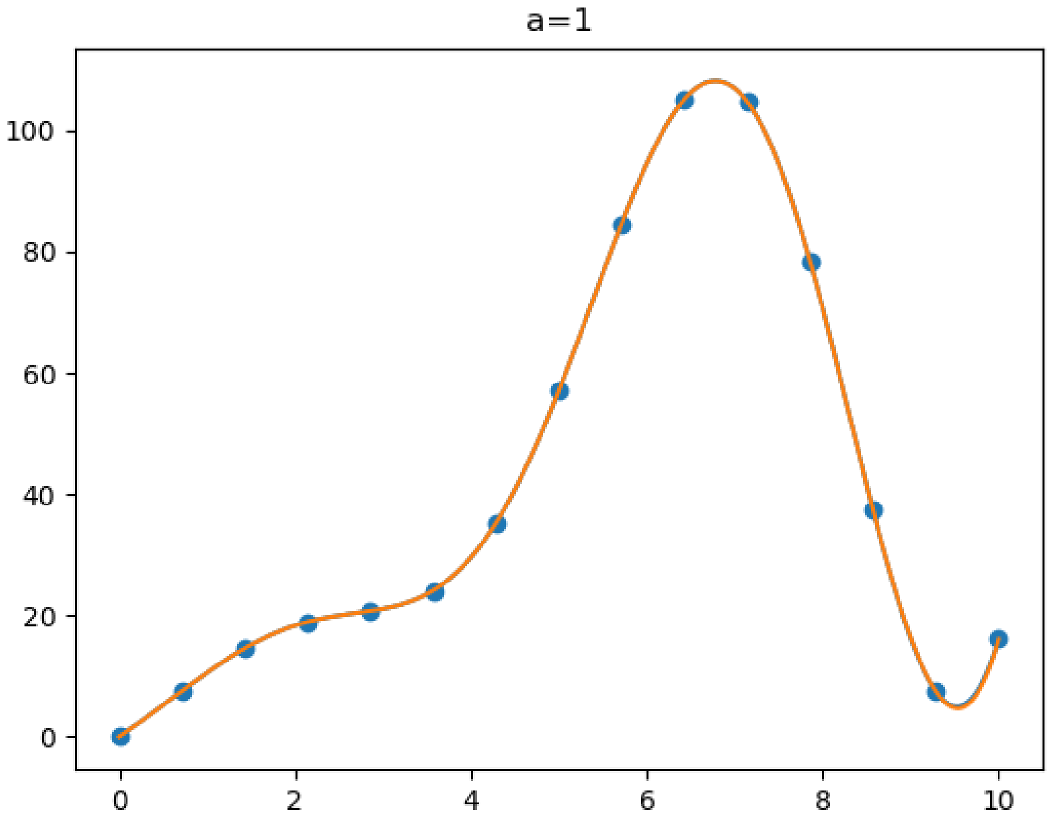
```
print(f"{s(0.18)=}")
print(f"Relative error = {abs((f(0.18) - s(0.18)) / f(0.18))}")
print(f"\n{f_prime(0.18)=}")
print(f"{s_prime(0.18)=}")
print(f"Relative error = {abs((f_prime(0.18) - s_prime(0.18)) / f_prime(0.18))}")

print(f"\n{f(0.2)=}\n{s(0.2)=}")

exact_integral  = integrate.quad(f, 0.1, 0.3)[0]
approx_integral = integrate.quad(s, 0.1, 0.3)[0]
print(f"\n{exact_integral=}")
print(f"{approx_integral=}")
print(f"Relative error = {abs((exact_integral - approx_integral) / approx_integral)}")
```

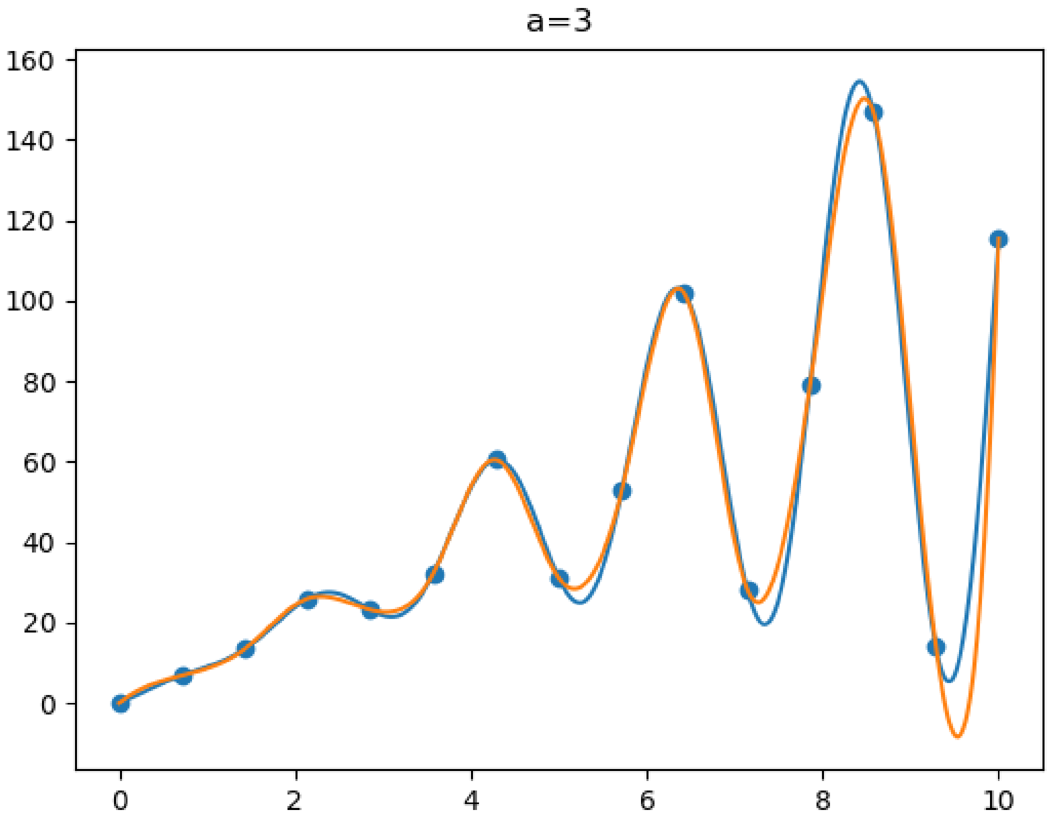
```
a_0=4.38124999999998
b_0=-1.3143749999999947
c_0=-2.6198488000000006
d_0=-0.01930258000000029

a_1=-4.381249999999994
b_1=3.9431249999999807
c_1=-3.671348799999997
d_1=0.050797419999999965

f(0.18)=-0.508123464353665
s(0.18)=-0.5079096640000003
Relative error = 0.00042076457527223696

f_prime(0.18)=-2.651616828775273
s_prime(0.18)=-2.2413088000000028
Relative error = 0.1547388085347094

f(0.2)=-0.5607973368863504
s(0.2)=-0.5607973400000005

exact_integral=-0.11157403517621209
approx_integral=-0.1115022805000001
Relative error = 0.0006435265349750621
```

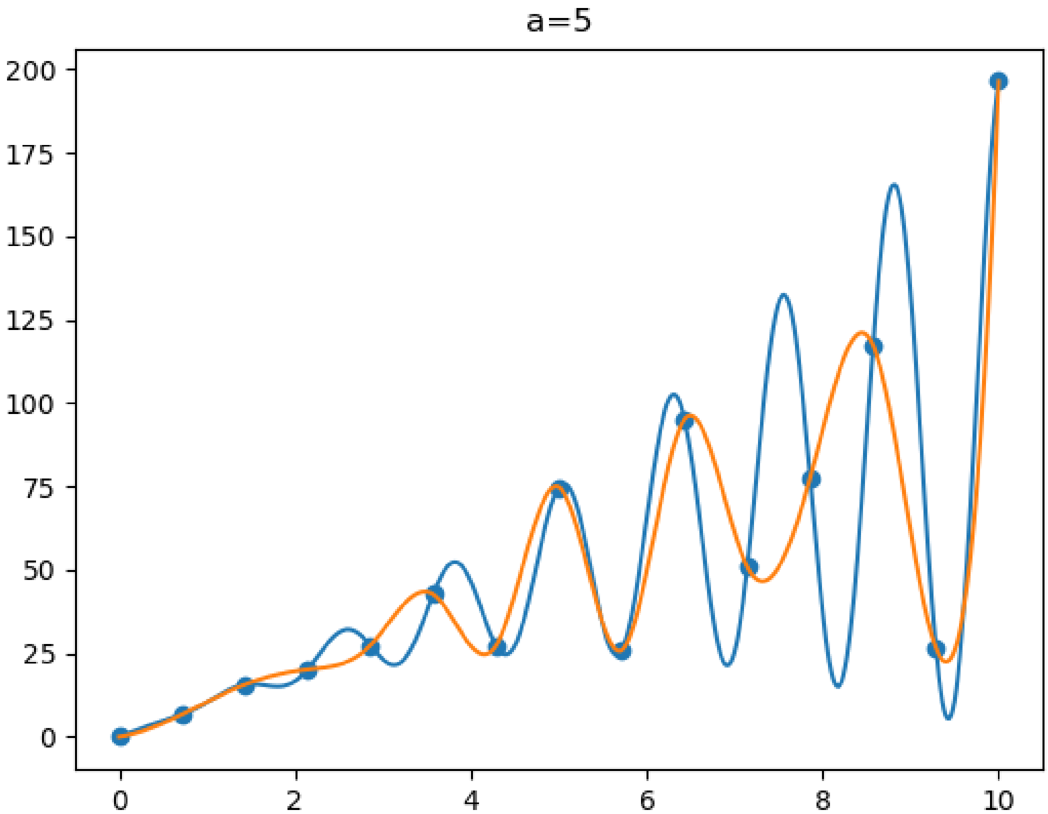
Problem 0.4.

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import CubicSpline

def f(a, x):
    return np.cos(a * x) * x**2 + 10 * x

xvals = np.linspace(0, 10, 15)
x_points = np.linspace(0, 10, 500)
```

```
for a in [1, 3, 5]:
    fvals = [f(a, x) for x in xvals]
    plt.scatter(xvals, fvals)
    spln = CubicSpline(xvals, fvals)
    y_points = [f(a, x) for x in x_points]
    plt.plot(x_points, y_points)
    y_points = [spln(x) for x in x_points]
    plt.plot(x_points, y_points)
    plt.show()
```




a=3
160
140
120
100
80
60
40
20
0
0
2
4
6
8
10



As a increases, the spline approximations get worse. That's because we're pushing the Nyquist-Shannon sampling limit.