MATH 210 Final Exam

December 6, 2019

INSTRUCTIONS

- 120 minutes, 9 questions and 75 points
- Plain scientific calculators only (graphing/programmable calculators are **not** allowed)
- No cellphones, laptops or notes
- Use proper syntax when writing Python code
- It is **not** necessary to include a documentation string when writing a Python function
- Documentation and formula sheet provided

NAME:

STUDENT #:

SIGNATURE:

Question	Grade	Question	Grade
1	/18	6	/6
2	/10	7	/6
3	/5	8	/8
4	/10	9	/6
5	/6	Total	/75

1. (18 points; 3 points each) **Predict** the output for each of the following blocks of code. If the output is a float, then display the number to **only 4 decimal places**.

```
def fun(x):
    return x**2 - 3

x = 0
for n in range(0,5):
    x = fun(x)

print(x)
```



```
def fun(s):
    n = len(s)
    terms = [1/s[i] for i in range(0,n)]
    result = n/sum(terms)
    return result

x = [1,5,2,1,3]
y = fun(x)

print(y)
```

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

def fun(y,t):
    return y**2

t = np.linspace(0,0.5,100)
y = spi.odeint(fun,1,t)

print(y[-1,0])
```



```
import numpy as np

A = np.ones(3) - np.eye(3)
M = A @ A

print(M)
```

```
import numpy as np

x = np.linspace(0,1,11)
X = np.vander(x, N=4, increasing=True)
M = X[-2:,-2:]

print(M)
```



```
import numpy as np

x = np.array([0.0,1.0,2.0,3.0])
y = np.array([[1.0],[3.0],[2.0],[4.0]])
X = np.column_stack((np.ones(len(x)),x))
M = X.T @ X
b = X.T @ y
A = np.column_stack((M,b))

print(A)
```

2. (10 points) Answer **True** or **False**. No justification required.

(a) The trapezoid rule approximation $T_N(f)$ is the average of the left and right Riemann sums $R_N(f)$ and $L_N(f)$ with the same number N of subintervals over the same interval $[a, b]$.

(b) Suppose $|f''(x)| \leq K_2$ and $|g''(x)| \leq M_2$ for all $x \in [a, b]$. If $K_2 < M_2$, then $E_N^T(f) < E_N^T(g)$.

(c) Suppose we use Simpson's rule with $N = 10$ subintervals to approximate an integral. If we increase the number of subintervals to $N = 20$ then we expect the error to decrease approximately by a factor of 1/8.

(d) If $f(x)$ is increasing on the interval $[a, b]$ then the right Riemann sum approximation $R_N(f)$ is an over-estimate of the definite integral $I = \int_a^b f(x)dx$. In other words, if $f'(x) \geq 0$ for all $x \in [a, b]$ then $R_N(f) \geq I$.

(e) If $f(x)$ is concave up on the interval $[a, b]$, then the trapezoid rule approximation $T_N(f)$ is an over-estimate of the definite integral $I = \int_a^b f(x)dx$. In other words, if $f''(x) \geq 0$ for all $x \in [a, b]$, then $T_N(f) \geq I$.

(f) Given a set of N points $(x_1, y_1), \ldots, (x_N, y_n)$ (with $x_i \neq x_j$ for $i \neq j$), there is a unique polynomial of degree N which interpolates these points.

(g) The linear system of equations $X^T X \boldsymbol{a} = X^T \boldsymbol{y}$ which solves for the coefficients $\boldsymbol{a}$ of the polynomial of degree d which best fits the data (minimizing the sum of squared errors) is a system with $d+1$ equations.

(h) The function `scipy.integrate.odeint` returns the exact values of the solution $y(t)$ of a differential equation

$$y' = f(y, t) \quad , \quad y(t_0) = y_0$$

(i) If the step size h is chosen small enough, Euler's method will compute the exact values of the solution $y(t)$ of the differential equation

$$y' = f(y, t) \quad , \quad y(t_0) = y_0$$

(j) Python is cool. (Hint: this statement is `True`.)

3. (5 points) Consider the system of differential equations

$$\ddot{x} = -\frac{x}{(x^2+y^2)^{3/2}}$$
$$\ddot{y} = -\frac{y}{(x^2+y^2)^{3/2}}$$

Determine an order of the line numbers such that the resulting code plots the solution $x(t)$ versus $y(t)$ for $t \in [0, 10]$ with initial conditions $x(0) = 1, \dot{x}(0) = 0, y(0) = 0, \dot{y}(0) = 1$.

```
1         return dudt
2         dudt[1] = -u[0]/D
3     import matplotlib.pyplot as plt
4         dudt = np.zeros(4)
5     plt.show()
6         D = (u[0]**2 + u[2]**2)**(3/2)
7     u0 = [1,0,0,1]
8     import numpy as np
9         dudt[2] = u[3]
10    t = np.linspace(0,10,1000)
11        dudt[3] = -u[2]/D
12    import scipy.integrate as spi
13    plt.plot(U[:,0],U[:,2])
14    def f(u,t):
15    U = spi.odeint(f,u0,t)
16        dudt[0] = u[1]
```

4. (a) (4 points) Write a Python function called `k_sum` which takes input parameters `x` and `N` and returns the sum

$$\sum_{k=1}^{N} \frac{x^{2k+1}}{(k+1)^2}$$

Do **not** import any packages. Use only builtin Python datatypes and functions.

(b) (4 points) Write a Python function called `k_seq` which takes input parameters `a` and `N` and returns a Python list of length $N+1$ representing the sequence

$$x_0 = a \qquad x_{k+1} = x_k - \frac{x_k^2 - 2}{2x_k}$$

Do **not** import any packages. Use only builtin Python datatypes and functions.

(c) (2 points) Predict the output of the Python code (up to 4 decimal places):

```
s = k_seq(-2,100)
print(s[-1])
```

5. (6 points) Write a function called `max_sval` which takes an input parameter `A` and returns the maximum eigenvalue of AA^T. Use NumPy and SciPy and include any relevant `import` statements. Note that the eigenvalues of AA^T are real since the matrix is symmetric therefore the function should return a real number.

6. (6 points) Approximate **all** solutions of the equation $x = e^{-x}$ to 3 decimal places. Justify your answer and show all your work.

7. (a) (4 points) Use Euler's method with $h = 0.25$ to predict the value $y(1)$ where $y(t)$ is the unique solution of the initial value problem

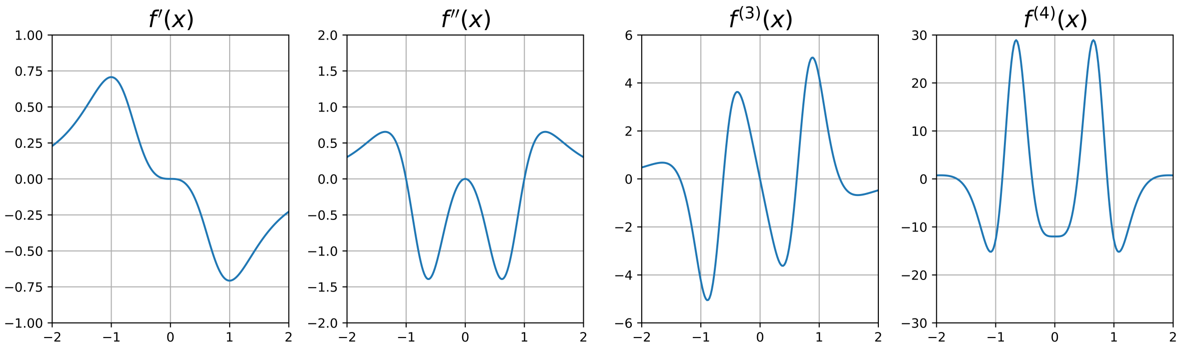
$$y' = \sin(y) \ \ , \ y(0) = 1$$

(b) (2 points) Is the approximation in part (a) an over-estimate or an under-estimate of the true value $y(1)$? In other words, if $\tilde{y}$ is the approximation in part (a), then $y(1) < \tilde{y}$ or $y(1) > \tilde{y}$? Justify your answer.

8. Consider the definite integral

$$I = \int_0^2 \frac{dx}{\sqrt{x^4+1}}$$

The figure below displays the derivatives of $f(x) = \dfrac{1}{\sqrt{x^4+1}}$:



(a) (4 points) Determine a value N which guarantees the error in the midpoint Riemann sum approximation satisfies $E_N^M(f) \leq 0.025$. Justify your answer.

(b) (4 points) Approximate I with error less than 0.025. Choose any method and justify your answer.

9. (6 points) Write a Python function called `linear_fit` which takes an input parameter `y` (a 1D NumPy array of length $N+1$) and returns the coefficient vector $\boldsymbol{a} = [a_0, a_1]^T$ of the linear model $y = a_0 + a_1 x$ which best fits (minimizing the sum of squared errors) the data

$$(0, y_0), (1, y_1), (2, y_2), \ldots, (N, y_N)$$

Use NumPy and SciPy and include any relevant `import` statements.

Extra workspace

Extra workspace