Homework 2 - ME I4600, Computational Fluid Mechanics, Spring 2022, City College of New York

General imports

In [554…

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

Problem 1

In [555…

```python
def runge(x):
    '''
    Runge's function.

    Input(s):
        x: float
    Output(s):
        y: float
    '''
    return 1/(1 + 25*x**2)

def lpi(x, arr_x, arr_y, n=None):
    '''
    Lagrange polynomial interpolation.

    Input(s):
        x: (float) the point on the x-axis at which interpolation is desired
        arr_x: (list or NumPy array) array of points on the x-axis
        arr_y: (list or NumPly array) array of functions based on points from the x-axis
        n: (int, optional) order of interpolation
    Outputs(s):
        p: (float) interpolated point that is a function of 'x'
    '''

    # If no order was specified, set order to be equal to umber of points
    if not n:
        n = len(arr_x)

    # Intialize interpolation
    p = 0
    # Iterate through products
    for j in range(0, n):
        # Initialize inner sum
        s = 1
        # Iterate through sums
        for i in range(0, n):
            # i =/= j condition
            if i != j:
                s = s * (x - arr_x[i])/(arr_x[j] - arr_x[i])
        p += arr_y[j]*s

    return p
```

Problem 1a: Using the above data in the table, find the interpolated value at x=0.9. (10 pts)

In [558…

```python
# Define points along the x-axis
step_x = 0.2
arr_x_a = np.arange(-1, 1+step_x, step_x)
# Define functions of the arr_x points
```

```
arr_y_a = np.array([runge(x) for x in arr_x_a])

# Define point at which interpolation is desired
x = 0.9
# Calculate the interpolation
p_a = lpi(x, arr_x_a, arr_y_a)
```

Problem 1b: Use Runge's function to generate a table of 21 equally spaced data points. Interpolate these data using a Lagrange polynomial of order 20. Plot this polynomial and comment on the comparison between your result and the plot of part (a). (10 pts)

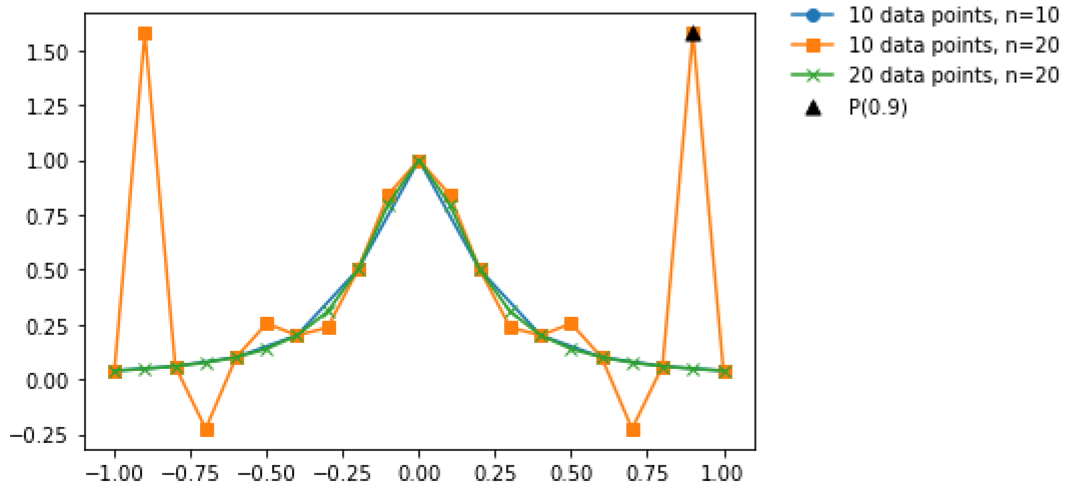
In [569…

```
# Define points along the x-axis
arr_x_b = np.linspace(-1, 1, 21)
# Define functions of the arr_x points
arr_y_b = np.array([runge(x) for x in arr_x_b])

# Define point at which interpolation is desired
x = 0.9
# Calculate the interpolation
p_b_10 = [lpi(i, arr_x_a, arr_y_a) for i in arr_x_b]
p_b_20 = [lpi(i, arr_x_b, arr_y_b) for i in arr_x_b]
print(arr_y_b[-2])
```

```
0.04705882352941175
```

In [570…

```
fig, ax = plt.subplots()
ax.plot(arr_x_a, arr_y_a, marker='o', label='10 data points, n=10')
ax.plot(arr_x_b, p_b_10, marker='s', label='10 data points, n=20')
ax.plot(arr_x_b, p_b_20, marker='x', label='20 data points, n=20')
ax.scatter(x, p_a, marker='^', c='k', s=50, zorder=10, label='P(0.9)')
fig.legend(loc='upper right', bbox_to_anchor=(1.275, 0.925), frameon=False);
```



Problem 2

Problem 2a: Interpolate the data with the Lagrange polynomial (5 pts). Plot the polynomial and the data points (5 pts). Use the polynomial to predict the condition of the lakes in 2009 (5 pts). Discuss this prediction (5 pts).

In [581…

```
# List of years
interval = 2
years = np.arange(1993, 2009, interval)
# Toxin concentrations
conc = np.array([12, 12.7, 13, 15.2, 18.2, 19.8, 24.1, 28.1])
```

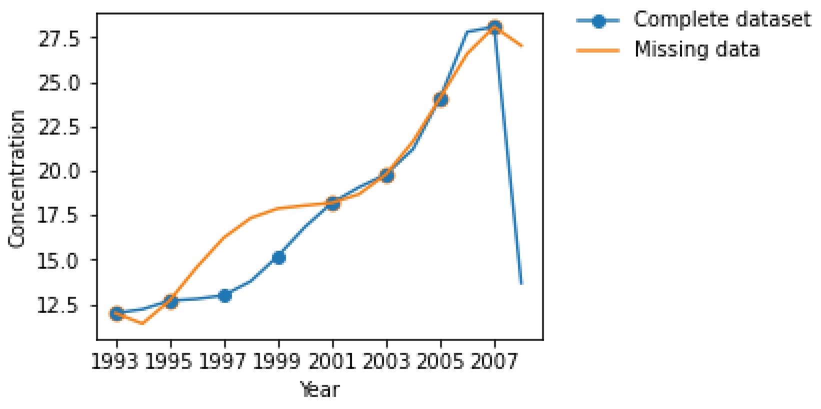
```
# Interpolate concentrations annually
years_annual = np.arange(1993, 2009, interval/2)
conc_p = [lpi(year, years, conc) for year in years_annual]

# Predict concentration in 2009
conc_p_2009 = lpi(2009, years, conc)
print('Toxin concentration prediction for 2009: {0:.2f}'.format(conc_p_2009))
```

```
Toxin concentration prediction for 2009: -38.40
```

Problem 2b: Interpolation may also be used to fill “holes” in the date. Say the data from 1997 and 1999 disappeared. Predict these values using the Lagrange polynomial fitted through the other known data points (10 pts).

In [584…

```
# Boolean mask
mask = np.where((years != 1997) & (years != 1999))
# List of years
years_missing = years[mask]
# Toxin concentrations
conc_missing = conc[mask]

# Interpolate concentrations annually
years_annual = np.arange(1993, 2009, interval/2)
conc_p_hole = [lpi(year, years_missing, conc_missing) for year in years_annual]

# Plot the concentrations
fig, ax = plt.subplots(figsize=(4, 3))
ax.plot(years_annual, conc_p, marker='o', markevery=interval, label='Complete dataset')
ax.scatter(years_missing, conc_missing, s=50, color='tab:orange')
ax.plot(years_annual, conc_p_hole, color='tab:orange', label='Missing data')
ax.set_xlabel('Year')
ax.set_xticks(years)
ax.set_ylabel('Concentration')
fig.legend(loc='upper right', bbox_to_anchor=(1.4, 0.925), frameon=False);
```



In [576…

```
for i in range(0, len(conc_p_hole)):
    print(years_annual[i], conc_p_hole[i])
```

```
1993.0 12.0
1994.0 11.406171874999998
1995.0 12.7
1996.0 14.569140624999996
1997.0 16.233333333333333
1998.0 17.343359375
1999.0 17.880000000000003
2000.0 18.052994791666666
2001.0 18.2
2002.0 18.685546875000004
2003.0 19.8
```

```
2004.0 21.658515625000003
2005.0 24.1
2006.0 26.586067708333335
2007.0 28.1
2008.0 27.045703125000003
```

Problem 3

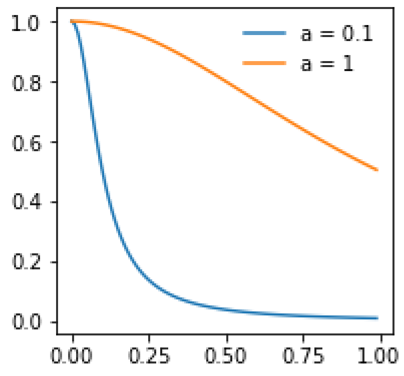
In [578…

```python
def lorentz(x, a):
    '''
    Lorentz profile.

    Input(s):
        x: float
        a: float
    Output(s):
        y: float
    '''
    return 1/(1 + x**2/(a**2))
```

Problem 3a: For a = 0.1 and 1, (a) plot the function from 0 to 1 (5 pts).

In [580…

```python
arr_x = np.arange(0, 1, 0.01)
y1 = [lorentz(x, 0.1) for x in arr_x]
y2 = [lorentz(x, 1) for x in arr_x]

fig, ax = plt.subplots(figsize=(3, 3))
ax.plot(arr_x, y1, label='a = 0.1')
ax.plot(arr_x, y2, label='a = 1')
ax.legend(frameon=False);
```



Problem 3b: Integrate this function from 0 to 1 using Newton-Cotes methods of various orders (n = 2, 3, and 6) with fixed number of intervals (number of intervals = 12) (10 pts).

In [588…

```python
# Define Newton-Cotes coefficients.
def ncc(n):
    '''
    Newton-Cotes coefficients generator.

    Input(s):
        n: int, order of Newton-Cotes coefficients
    Output(s):
        coeffs: array, array of floats with Newton-Cotes coefficients

    '''

    # Define array of values to set up coefficient matrix
    arr = np.arange(0, n+1, 1)
    # Define column vector to process matrix with
    B = [(n**(i+1))/(i+1) for i in range(0, n+1)]
    # Generate Vandermonde matrix
    M = np.vander(arr, max(arr)+1, increasing=True)
    # Generate coefficients
    coeffs = np.dot(np.linalg.inv(M.T), B)/n

    return coeffs
```

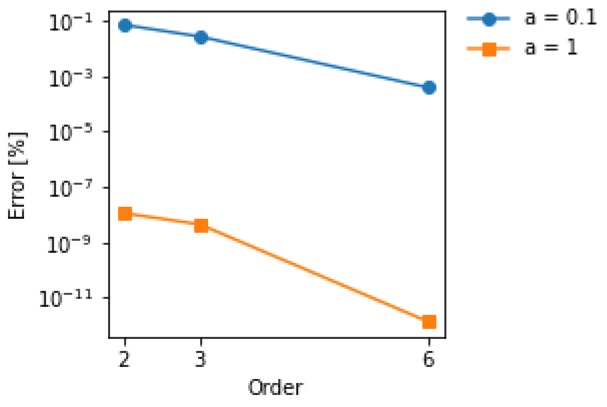
In [586…

```
def ncc_integration(n, lower, upper, a):

    # Define lower and upper bounds
    arr = np.linspace(lower, upper, n+1)
    # Generate Newton-Cotes coefficients
    coeffs = ncc(n)
    # Initialize integration
    s = 0
    # Perform integration
    s = (upper - lower)*np.nansum([coeffs[k]*lorentz(arr[k], a) for k in range(0, n+1)])

    return s
```

In [590…

```
# Define number of intervals
N = 12
# Define x-values
arr_x = np.linspace(0, 1, N)
# Initialize list of integrations and errors
ints, errors = [], []
# Define actual value (from Wolfram)
actual = [0.147113, np.pi/4]
# Define orders of interest
ns = [2, 3, 6]
# Perform integrations
for j, a in enumerate([0.1, 1]):
    # Initialize list to capture coefficient-specific errors
    error = []
    for n in ns:
        s = np.nansum([ncc_integration(n, arr_x[i], arr_x[i+1], a) for i in range(0, len(a
        print('When a is {0}, the integral of function with order {1} is: {2:.6f}'.format
        ints.append(s)
        # Catalogue error
        error.append(100*np.abs((s - actual[j])/actual[j]))
    # Append to master error list
    errors.append(error)

# Plotting
fig, ax = plt.subplots(figsize=(3,3))
ax.plot(ns, errors[0], marker='o', label='a = 0.1')
ax.plot(ns, errors[1], marker='s', label='a = 1')
ax.set_yscale('log')
ax.set_xticks(ns)
ax.set_xlabel('Order')
ax.set_ylabel('Error [%]')
fig.legend(loc='upper right', bbox_to_anchor=(1.3, 0.925), frameon=False);
```

```
When a is 0.1, the integral of function with order 2 is: 0.147009
When a is 0.1, the integral of function with order 3 is: 0.147074
When a is 0.1, the integral of function with order 6 is: 0.147112
When a is 1, the integral of function with order 2 is: 0.785398
When a is 1, the integral of function with order 3 is: 0.785398
When a is 1, the integral of function with order 6 is: 0.785398
```

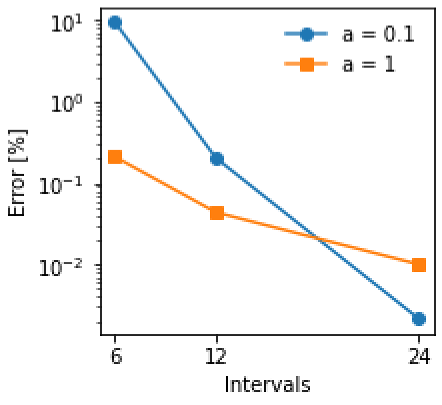


Problem 3c: Integrate this function from 0 to 1 using trapezoidal rule (n = 1) with variable number of intervals (number of intervals = 6, 12, and 24) (10 pts).

In [592…

```python
# Define order
n = 1
# Define Lorentz coefficients
Ns = [6, 12, 24]
# Intialize list of actuals and errors
actual, errors = [0.147113, np.pi/4], []
# Perform integrations
for j, a in enumerate([0.1, 1]):
    error = []
    for N in Ns:
        arr_x = np.linspace(0, 1, N)
        s = np.nansum([ncc_integration(n, arr_x[i], arr_x[i+1], a) for i in range(0, len(a
        print('Integral of function with {0} intervals is: {1:.6f}'.format(N, s))
        error.append(100*np.abs((s - actual[j])/actual[j]))
    errors.append(error)

# Plotting
fig, ax = plt.subplots(figsize=(3, 3))
ax.plot(Ns, errors[0], marker='o', label='a = 0.1')
ax.plot(Ns, errors[1], marker='s', label='a = 1')
ax.set_yscale('log')
ax.set_xticks(Ns)
ax.set_xlabel('Intervals')
ax.set_ylabel('Error [%]')
ax.legend(frameon=False);
```

```
Integral of function with 6 intervals is: 0.161237
Integral of function with 12 intervals is: 0.147413
Integral of function with 24 intervals is: 0.147110
Integral of function with 6 intervals is: 0.783732
Integral of function with 12 intervals is: 0.785054
Integral of function with 24 intervals is: 0.785319
```



Problem 4: Integrate the function of Problem 3 using Gauss quadrature of various numbers of Gauss points

(n ≤ 5) with single interval (20 pts). Plot the error versus the number of Gauss points. (15 pts)

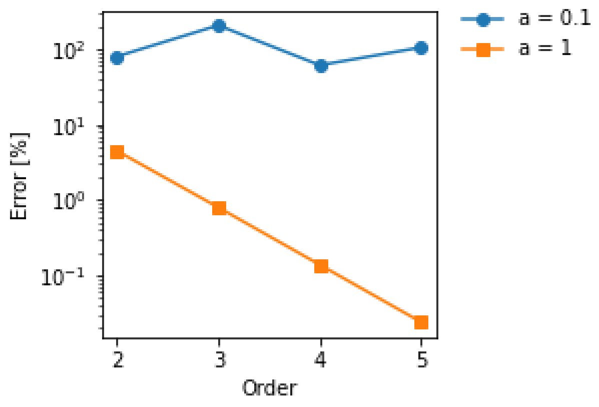
In [594…

```
def gq(n, a):
    '''
    Gauss quadrature for use with the Lorentz profile.

    Input(s):
        n: int
        a: int, for Lorentz profile
    Output(s):
        s: float
    '''

    # Catalogue abscissas and weights for orders (n = 2 to n = 5)
    if n == 2:
        arr_x = [-1, -np.sqrt(3)/3, np.sqrt(3)/3, 1]
        w = [1, 1, 1, 1]
    elif n == 3:
        arr_x = [-1, -np.sqrt(15)/5, 0, np.sqrt(15)/5, 1]
        w = [1, 5/9, 8/9, 5/9, 1]
    elif n == 4:
        arr_x = [-1,
                 -np.sqrt(525 + 70*np.sqrt(30))/35,
                 -np.sqrt(525 - 70*np.sqrt(30))/35,
                 np.sqrt(525 - 70*np.sqrt(30))/35,
                 np.sqrt(525 + 70*np.sqrt(30))/35,
                 1]
        w = [1,
             (18-np.sqrt(30))/36,
             (18+np.sqrt(30))/36,
             (18+np.sqrt(30))/36,
             (18-np.sqrt(30))/36, 1]
    elif n == 5:
        arr_x = [-1,
                 -np.sqrt(245 + 14*np.sqrt(70))/21,
                 -np.sqrt(245 - 14*np.sqrt(70))/21,
                 0,
                 np.sqrt(245 - 14*np.sqrt(70))/21,
                 np.sqrt(245 + 14*np.sqrt(70))/21,
                 1]
        w = [1,
             (322 - 13*np.sqrt(70))/900,
             (322 + 13*np.sqrt(70))/900,
             128/225,
             (322 + 13*np.sqrt(70))/900,
             (322 - 13*np.sqrt(70))/900,
             1]
    else:
        return None

    # Get quadrature
    s = np.nansum([w[k]*lorentz(arr_x[k], a) for k in range(1, n+1)])

    return s
```

In [596…

```
# Initialize quadrature orders
ns = [2, 3, 4, 5]
# Intialize list of actuals and errors
actual, errors = [0.147113, np.pi/4], []

# Iterate over each value of a
for j, a in enumerate([0.1, 1]):
    error = []
    for n in ns:
        value = gq(n, a)/2
        print('Gauss quadrature for Lorentz profile with coefficient {0} and order {1} is:
        error.append(100*np.abs(actual[j] - value)/actual[j])
    errors.append(error)

fig, ax = plt.subplots(figsize=(3, 3))
ax.plot(ns, errors[0], marker='o', label='a = 0.1')
ax.plot(ns, errors[1], marker='s', label='a = 1')
ax.set_xticks(ns)
ax.set_xlabel('Order')
ax.set_ylabel('Error [%]')
ax.set_yscale('log')
fig.legend(loc='upper right', bbox_to_anchor=(1.3, 0.925), frameon=False);
```

```
Gauss quadrature for Lorentz profile with coefficient 0.1 and order 2 is: 0.029126
Gauss quadrature for Lorentz profile with coefficient 0.1 and order 3 is: 0.453552
Gauss quadrature for Lorentz profile with coefficient 0.1 and order 4 is: 0.056556
Gauss quadrature for Lorentz profile with coefficient 0.1 and order 5 is: 0.303252
Gauss quadrature for Lorentz profile with coefficient 1 and order 2 is: 0.750000
Gauss quadrature for Lorentz profile with coefficient 1 and order 3 is: 0.791667
Gauss quadrature for Lorentz profile with coefficient 1 and order 4 is: 0.784314
Gauss quadrature for Lorentz profile with coefficient 1 and order 5 is: 0.785586
```



In [ ]: