Computational Physics: Homework 3: Integration

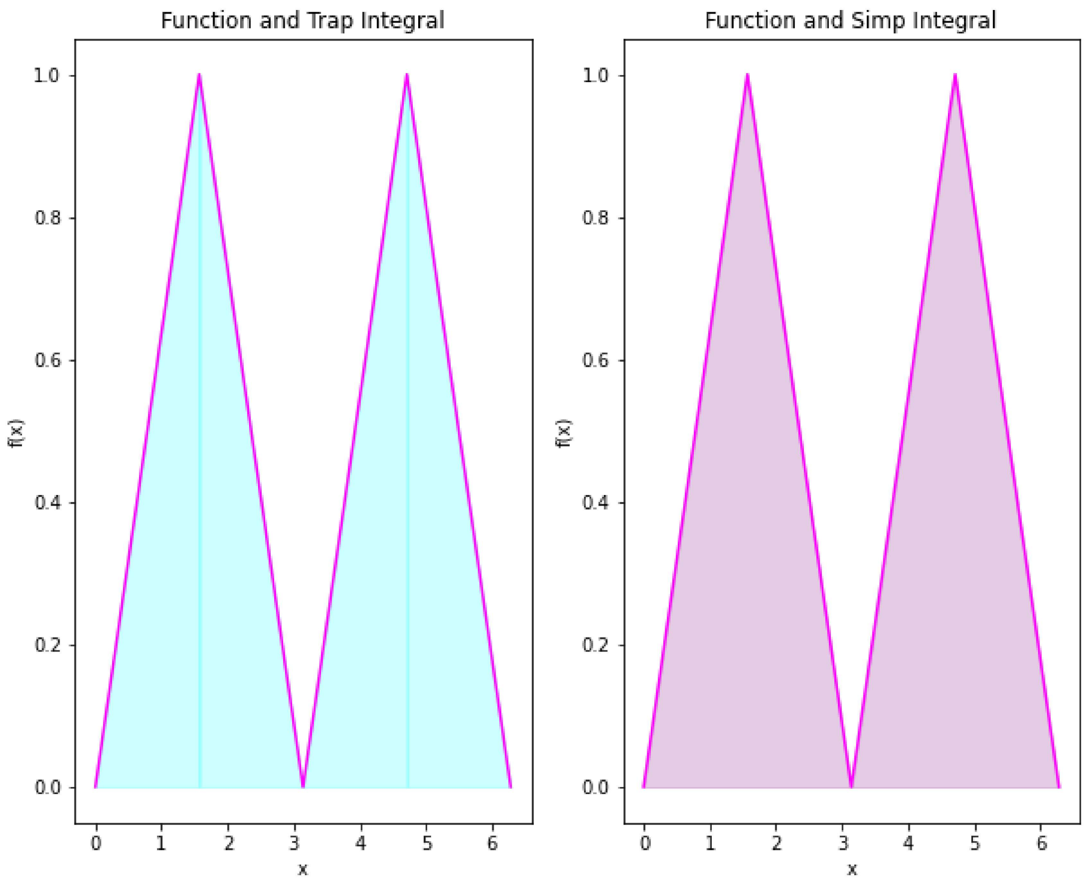
In [20]:
```
import numpy as np
import scipy.integrate
import matplotlib.pyplot as plt
import math
```

Problem 1:

In [21]:
```python
def f(x):
    return np.sin(x) **4

x = np.linspace(0, 2*np.pi, 5)
true_val = 3*np.pi/4

trap_val = np.trapz(f(x), x)
simp_val = scipy.integrate.simpson(f(x), x)

# Calculate the relative errors
trap_err = abs((trap_val - true_val) / true_val)
simp_err = abs((simp_val - true_val) / true_val)

print(f"Trapezoid = {trap_val}")
print(f"Simpson's = {simp_val}")
print(f"Trap err = {trap_err}")
print(f"Simp err = {simp_err}")

# function
plt.figure(figsize=(10,8))

plt.subplot(1,2,1)
plt.plot(x, f(x), label='f(x)', color='magenta')
# trap panels
for i in range(4):
    plt.fill_between(x[i:i+2], f(x[i:i+2]), alpha=0.2, color='cyan')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Function and Trap Integral')

plt.subplot(1,2,2)
plt.plot(x, f(x), label='f(x)', color='magenta')
# simp panels
for i in range(0, 5, 2):
    plt.fill_between(x[i:i+3], f(x[i:i+3]), alpha=0.2, color='purple')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Function and Simp Integral')
```

```
Trapezoid = 3.141592653589793
Simpson's = 4.1887902047863905
Trap err = 0.3333333333333333
Simp err = 0.7777777777777777
```

Out[21]:
```
Text(0.5, 1.0, 'Function and Simp Integral')
```


Function and Trap Integral
Function and Simp Integral
f(x)
x
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6

Problem 2:

In [22]:

```python
def f(x):
    return math.sin(math.exp(x))

def midpoint_rule(func, a, b, n):
    h = (b - a) / n
    I = 0
    for i in range(n):
        x_midpoint = a + (i + 0.5) * h
        I += f(x_midpoint)
    I *= h
    return I

def simpsons_rule(func, a, b, n):
    h = (b - a) / n
    I = f(a) + f(b)
    for i in range(1, n, 2):
        I += 4 * f(a + i * h)
    for i in range(2, n - 1, 2):
        I += 2 * f(a + i * h)
    I *= h / 3
    return I

true = 0.6061244734187699

# initial
n_mid = 1
n_simp = 2

mid_err = abs(midpoint_rule(f, 0, 3, n_mid) - true) / true
while mid_err > 1e-4:
    n_mid += 1
    mid_err = abs(midpoint_rule(f, 0, 3, n_mid) - true) / true

simp_err = abs(simpsons_rule(f, 0, 3, n_simp) - true) / true
while simp_err > 1e-4:
    n_simp += 2  # needs to be even
    simp_err = abs(simpsons_rule(f, 0, 3, n_simp) - true) / true

print("Final n_mid = ", n_mid)
print("Final n_simp = ", n_simp)
```

```
Final n_mid =  195
Final n_simp =  76
```

Problem 3:

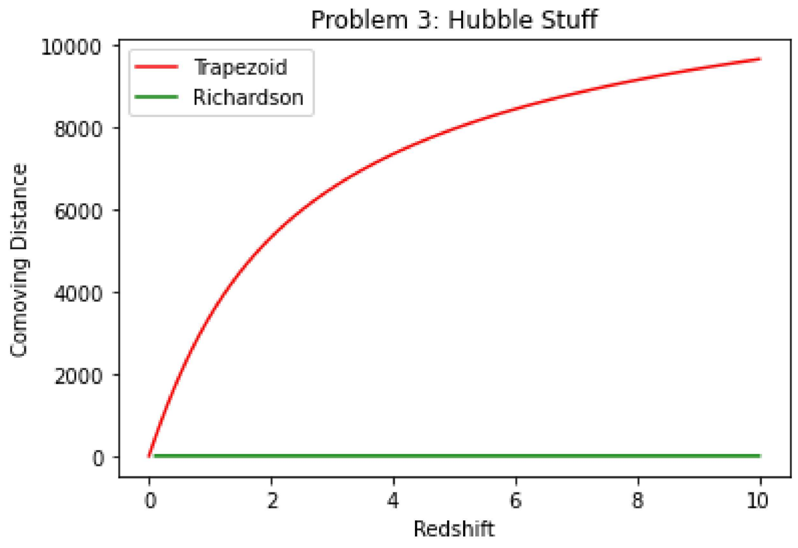
In [25]:
```
epsilon = 8.854187e-12  # F/m
charge_density = 1e-6    # C/m
observation_point = np.array([0, 3])  #observation point (xobs, yobs)

#parametrization of the wire
def x(s):
    return s

def y(s):
    return s**4

#function to calculate the distance between two points
def distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2)**2))

#midpoint rule integration
def midpoint_rule_integration(func, a, b, num_points):
    h = (b - a) / num_points
    integral = 0

    for k in range(1, num_points + 1):
        sk = a + (k - 0.5) * h
        integral += h * (func(sk) / distance(observation_point, np.array

    return integral

#electric potential using midpoint rule
num_points = 1000  # Adjust the number of points as needed
electric_potential = (1 / (4 * np.pi * epsilon)) * midpoint_rule_integra

print("Electric potential = ", electric_potential)
```

```
Electric potential =  6312.395581155265
```

Problem 4:

In [26]:
```
H0 = 67.66
c = 2.998e5
num_points = 100
num_panels = 200

Omega_r = 0.4165 / H0**2
Omega_k = 0
Omega_m = 0.3111 - 0.5 * Omega_r
Omega_vac = 0.6889 - 0.5 * Omega_r


def E(z):
    return np.sqrt(Omega_r*(1 + z)**4 + Omega_m * (1 +z)**3 + Omega_k *

def integrand(z):
    return 1 / E(z)

def trap(f, a, b, num_panels):
    h = (b - a) / num_panels
    I = 0.5 * (f(a) + f(b))
    for i in range(1, num_panels):
        I += f(a + i * h)
    return h * I

def rich(I, k):
    return (4**k * I[1] - I[0])/ (4**k - 1)

# comoving distance for each
z_vals = np.linspace(0, 10, num_points)
comoving_distances = []

for z in z_values:
    integrand_func = lambda z_prime: integrand(z_prime)
    integral = trap(integrand_func, 0, z, num_panels)
    comoving_distances.append(c / H0 * integral)

I = [trap(integrand_func, 0, z, num_panels) for z in z_vals[:2]]
k_vals = np.arange(1, len(z_vals))
richardson_distances = [rich(I, k) for k in k_vals]

plt.plot(z_vals, comoving_distances, label="Trapezoid", color = 'red')
plt.plot(z_vals[1:], richardson_distances, label="Richardson", color = '
plt.xlabel("Redshift")
plt.ylabel("Comoving Distance")
plt.title("Problem 3: Hubble Stuff")
plt.legend()
# why rich flat ):
```

Out[26]:
```
<matplotlib.legend.Legend at 0x2687d122310>
```

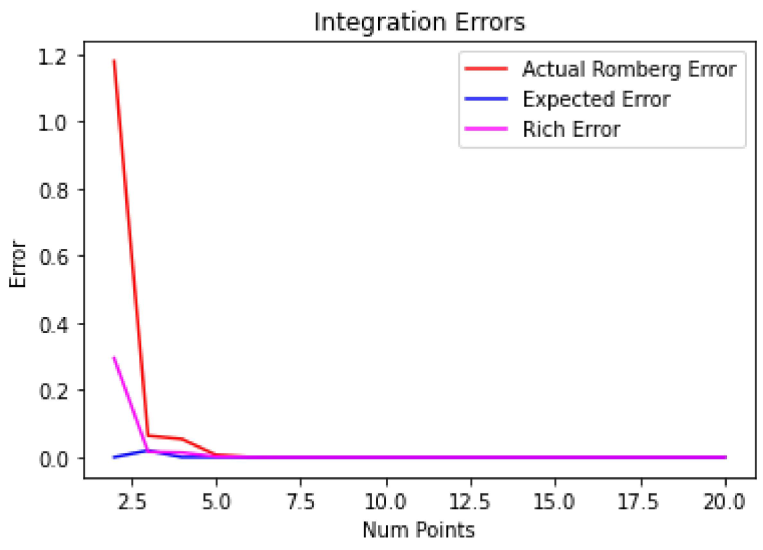

Problem 3: Hubble Stuff
Trapezoid
Richardson
Comoving Distance
Redshift
0
2000
4000
6000
8000
10000
0
2
4
6
8
10

Problem 5:

In [27]:
```
def func(x):
    return np.sin(np.exp(x))

# True value of the integral
true_value = 0.6440047437580401

num_points = np.arange(2, 21)

actual_errors = []
expected_errors = []
richardson_errors = []

# Calculate errors
for n in num_points:
    # Perform Romberg integration
    result = integrate.romberg(func, 0, np.pi, divmax=n)

    # Actual Romberg error using the bottom right point of the matrix
    actual_error = np.abs(result - true_value)
    actual_errors.append(actual_error)

    # Expected error using (R_i-1,j and R_i-1,j-1) for the current row
    if n > 2:
        prev_result = integrate.romberg(func, 0, np.pi, divmax=n-1)
        prev_prev_result = integrate.romberg(func, 0, np.pi, divmax=n-2)
        expected_error = np.abs(result - prev_result) / (2**(2*n) - 1)
        expected_errors.append(expected_error)
    else:
        expected_errors.append(0)

    richardson_error = 2 ** (-2) * np.abs(result - true_value)
    richardson_errors.append(richardson_error)

plt.plot(num_points, actual_errors, label="Actual Romberg Error", color
plt.plot(num_points, expected_errors, label="Expected Error", color = 'b
plt.plot(num_points, richardson_errors, label="Rich Error", color = 'mag
plt.xlabel("Num Points")
plt.ylabel("Error")
plt.title("Integration Errors")
plt.legend()
```

```
C:\Users\vince\anaconda3\lib\site-packages\scipy\integrate\_quadrature.
py:849: AccuracyWarning: divmax (2) exceeded. Latest difference = 1.586
517e+00
  warnings.warn(
C:\Users\vince\anaconda3\lib\site-packages\scipy\integrate\_quadrature.
py:849: AccuracyWarning: divmax (3) exceeded. Latest difference = 1.242
892e+00
  warnings.warn(
C:\Users\vince\anaconda3\lib\site-packages\scipy\integrate\_quadrature.
py:849: AccuracyWarning: divmax (1) exceeded. Latest difference = 2.009
865e+00
  warnings.warn(
C:\Users\vince\anaconda3\lib\site-packages\scipy\integrate\_quadrature.
py:849: AccuracyWarning: divmax (4) exceeded. Latest difference = 9.863
164e-03
  warnings.warn(
C:\Users\vince\anaconda3\lib\site-packages\scipy\integrate\_quadrature.
py:849: AccuracyWarning: divmax (5) exceeded. Latest difference = 4.745
078e-02
  warnings.warn(
C:\Users\vince\anaconda3\lib\site-packages\scipy\integrate\_quadrature.
py:849: AccuracyWarning: divmax (6) exceeded. Latest difference = 7.075
183e-03
  warnings.warn(
C:\Users\vince\anaconda3\lib\site-packages\scipy\integrate\_quadrature.
py:849: AccuracyWarning: divmax (7) exceeded. Latest difference = 5.497
926e-04
  warnings.warn(
C:\Users\vince\anaconda3\lib\site-packages\scipy\integrate\_quadrature.
py:849: AccuracyWarning: divmax (8) exceeded. Latest difference = 5.638
798e-06
  warnings.warn(
```

```
Out[27]: <matplotlib.legend.Legend at 0x2687d344970>
```


Integration Errors
Actual Romberg Error
Expected Error
Rich Error
Error
Num Points