UJIAN AKHIR SEMESTER GENAP TAHUN AKADEMIK 2023/2024

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNIK DAN INFORMATIKA

UNIVERSITAS MULTIMEDIA NUSANTARA

IF420 - Analisis Numerik

Petunjuk:

- Gunakan file template jawaban UAS ini untuk **SOAL B**
- Setiap butir pertanyaan memiliki bobot penilaiannya masing-masing
- Usahakan untuk mengerjakan setiap butir pertanyaan sesuai dengan perintah yang diberikan dan di tempat yang disediakan
- Tulis nama dan NIM Anda di dalam blok cell di bawah ini! Tampilkan dengan perintah print!

In [1]:
```
# input nama dan nim Anda di sini, lalu tampilkan dengan perintah print
nama = 'Farrelius Kevin'
nim = '00000081783'
print(f'Nama:\n {nama}')
print(f'NIM:\n {nim}')
```

```
Nama:
 Farrelius Kevin
NIM:
 00000081783
```

SOAL 1 (Total Bobot: 20%)

Soal 1 A (Bobot: 10%)

Buatlah sebuah fungsi **my_num_diff_w_smoothing(x,y,n)** dengan output [dy, X], dimana x dan y adalah 1D numpy array dengan panjang yang sama dan n adalah suatu bilangan bulat positif.
Pertama, fungsi tersebut akan membuat sebuah vektor yang terdiri atas titik-titik data y yang telah diperhalus (smoothed) dengan menerapkan formula

$$ysmooth[i] = np.mean(y[i - n : i + n])$$

Selanjutnya, fungsi tersebut akan menghitung dy, yakni turunan dari vektor y-smoothed tersebut dengan menggunakan metode **central difference**. Selain mengembalikan dy, fungsi tersebut juga akan mengembalikan output berupa 1D array X yang memiliki ukuran panjang yang sama dengan dy dan merepresentasikan nilai-nilai x dimana dy berlaku.
Asumsikan bahwa data x berada dalam urutan naik tanpa ada dua nilai yang sama (tidak ada duplikasi) dan elemen-elemen x bisa jadi tidak memiliki jarak yang sama satu dengan lainnya. Perhatikan bahwa output dy akan memiliki 2n+2 titik lebih sedikit daripada y, serta asumsikan bahwa panjang dari y jauh lebih besar daripada 2n+2.

In [2]:
```
# my_num_diff_w_smoothing function (Max Points: 10%)
# import all needed libraries
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
%matplotlib inline

def my_num_diff_w_smoothing(x, y, n):
    # put your codes here
    ysmooth = np.zeros(len(y))
    for i in range(n, len(y) - n):
        ysmooth[i] = np.mean(y[i - n:i + n + 1])

    dy = np.zeros(len(y) - 2 * n)
    for i in range(n, len(y) - n):
        dy[i - n] = (ysmooth[i + n] - ysmooth[i - n]) / (x[i + n] - x[i - n])

    X = x[n:len(x) - n]

    return [dy, X]
```

Soal 1 B (Bobot: 10%)

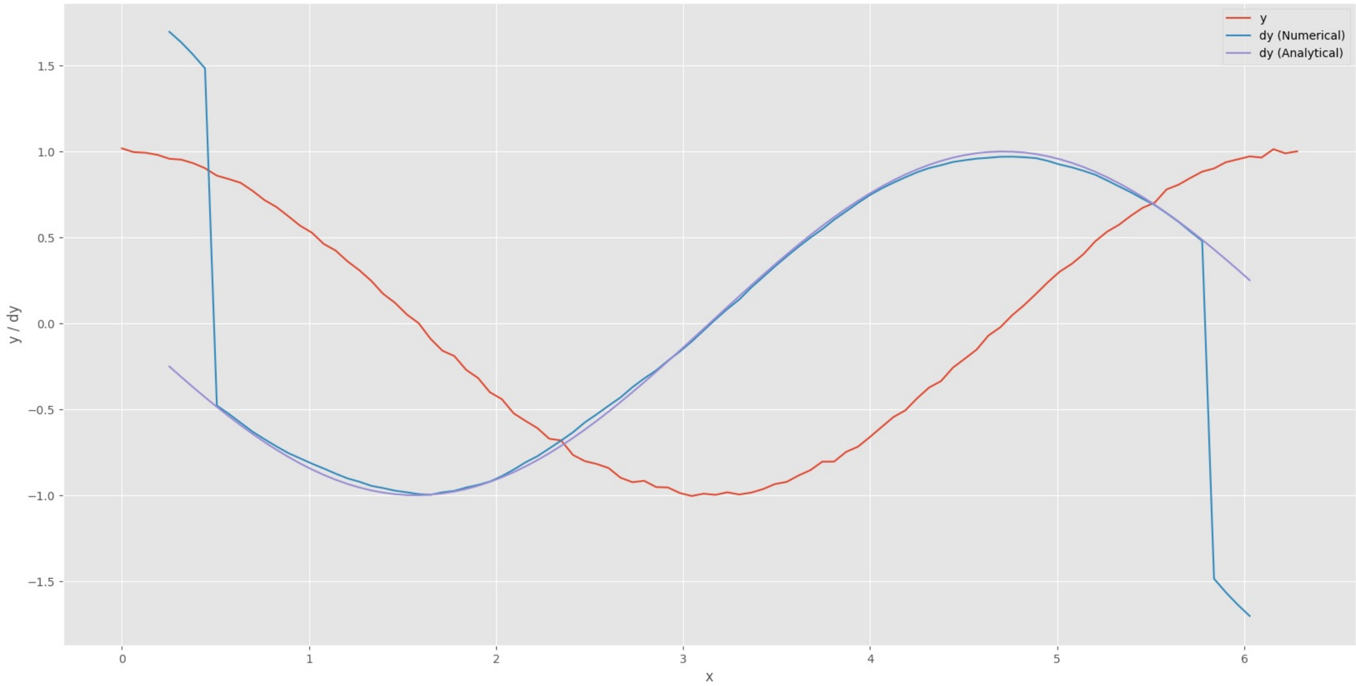
Lakukan pengecekan terhadap fungsi yang telah dibuat dengan nilai x, y, dan n yang didefinisikan sebagai berikut:
$x = np.linspace(0, 2 * np.pi, 100)$

$y = np.cos(x) + np.random.randn(len(x))/100$

$n = 4$

Plot hasilnya untuk solusi fungsi tersebut serta bandingkan dengan plot turunan fungsi tersebut secara analitik, yakni **dy = -np.sin(x)**

In [3]:
```
# test case (Max Points: 5%)
# put your codes here
x = np.linspace(0, 2*np.pi, 100)
y = np.cos(x) + np.random.randn(len(x))/100
n = 4

dy, X = my_num_diff_w_smoothing(x, y, n)
dy_analytical = -np.sin(x[n:len(x) - n])


# plot the estimation and analytic results (Max Points: 5%)
# put your codes here
plt.figure(figsize=(20, 10))
plt.plot(x, y, label='y')
plt.plot(X, dy, label='dy (Numerical)')
plt.plot(x[n:len(x) - n], dy_analytical, label='dy (Analytical)')
plt.xlabel('x')
plt.ylabel('y / dy')
plt.legend()
plt.show()
```



SOAL 2 (Total Bobot: 10%)

Kita telah mengetahui bahwa beberapa fungsi dapat dinyatakan sebagai jumlahan tak hingga atas polinomial (ingat kembali Deret Taylor). Fungsi-fungsi lainnya, terutama fungsi-fungsi periodik, dapat ditulis sebagai jumlahan tak hingga fungsi sinus dan cosinus, seperti ditunjukkan dalam persamaan **Deret Fourier** berikut:

$$f(x) = \frac{A_0}{2} + \sum_{n=1}^{\infty} A_n cos(nx) + B_n sin(nx)$$

Dapat ditunjukkan bahwa nilai-nilai dari A_n dan B_n bisa dihitung dengan menggunakan dua rumus berikut:

$$A_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)cos(nx)dx$$

$$B_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)sin(nx)dx$$

Sama seperti Deret Taylor, fungsi-fungsi juga dapat diperkirakan dengan memotong Deret Fourier pada suatu n=N. Deret Fourier dapat digunakan untuk memperkirakan beberapa fungsi khusus, seperti step function, dan banyak dipakai dalam aplikasi teknik seperti dalam pemrosesan sinyal.

Buatlah sebuah fungsi **my_fourier_coef(f,n)**, dengan output $[A_n, B_n]$, dimana f adalah suatu obyek fungsi yang memiliki periode 2π. Fungsi my_fourier_coef harus mampu menghitung koefisien ke-n Fourier, A_n dan B_n, dalam Deret Fourier untuk f, yang didefinisikan dengan dua rumus di atas. Anda dapat menggunakan fungsi **quad** dari scipy.integrate untuk melakukan integrasi.

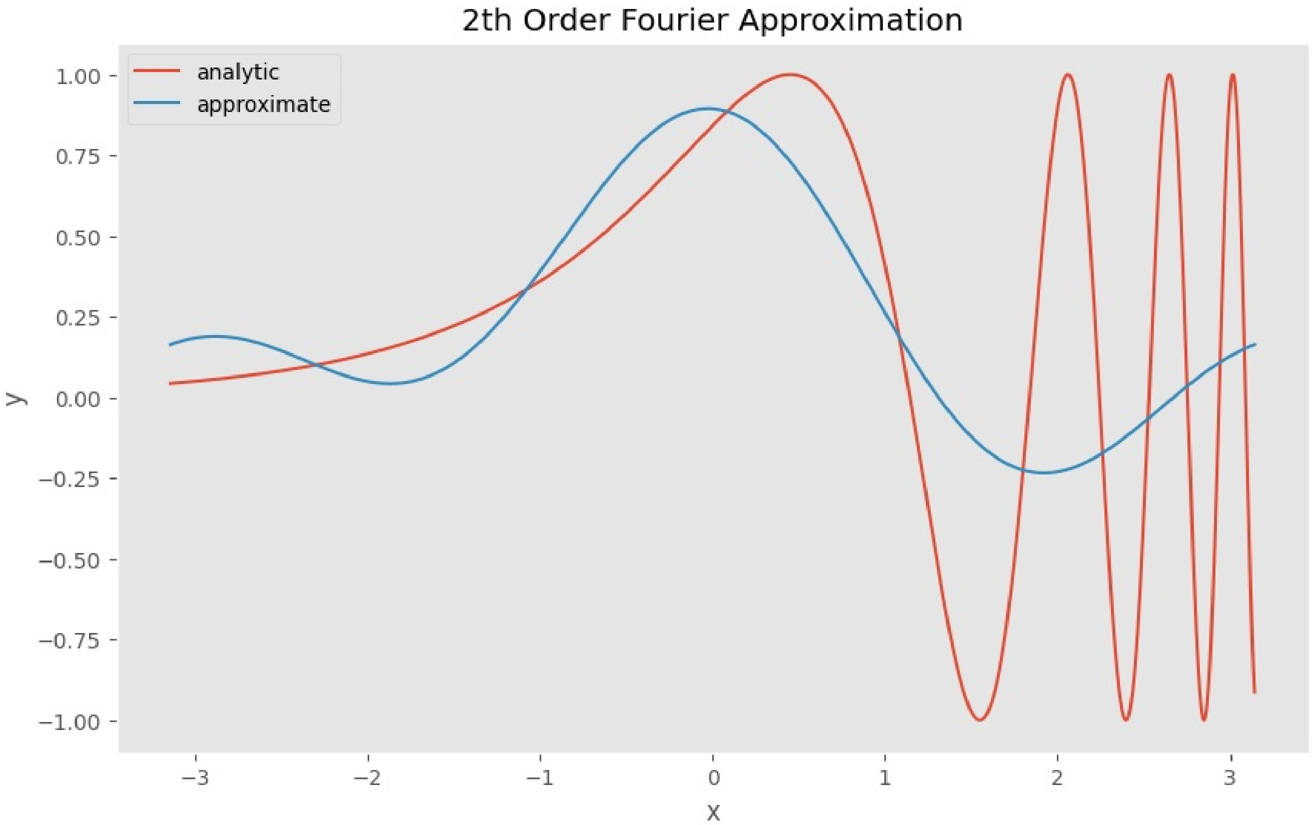
Setelah itu, gunakan fungsi **plot_results(f,N)** yang telah disediakan untuk mengecek fungsi my_fourier_coef yang Anda siapkan dengan detail input berikut:

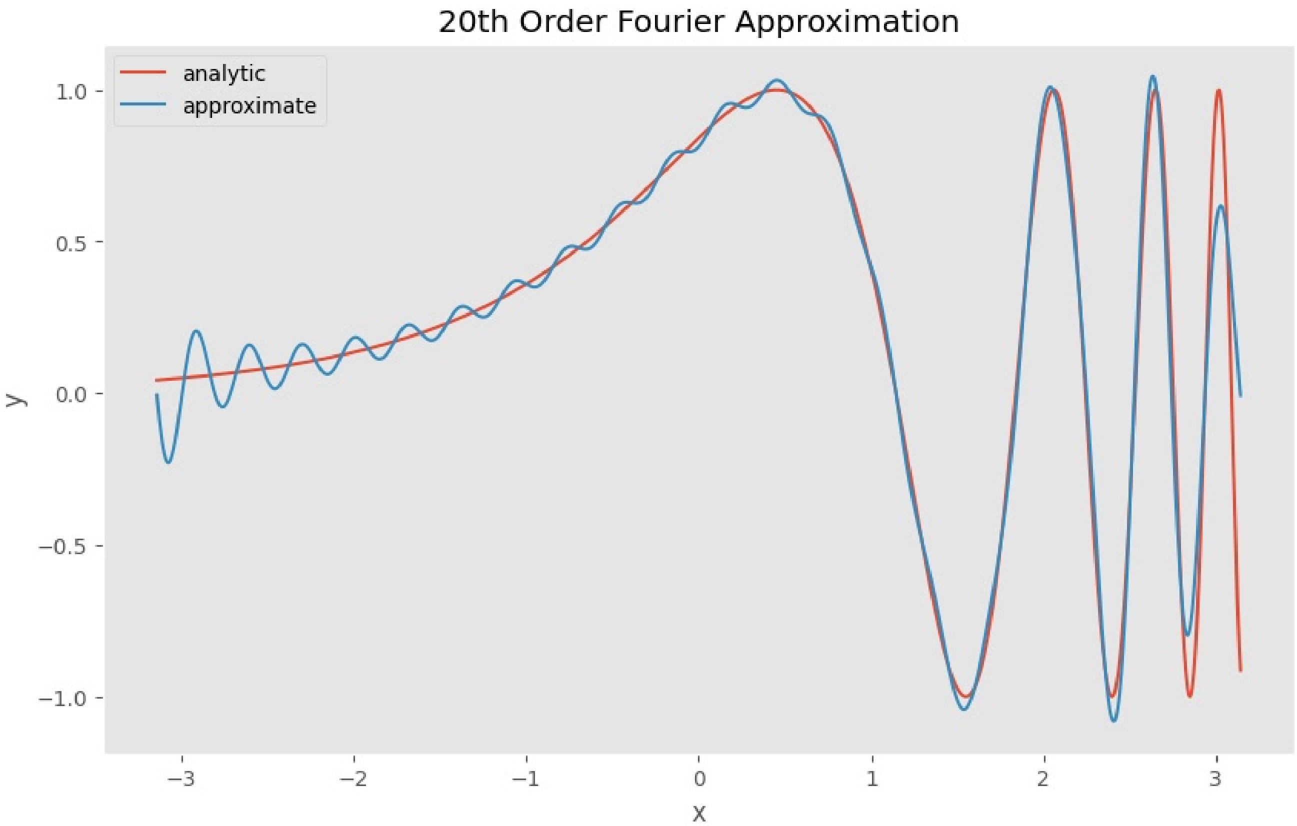
(1) $f = np.sin(np.exp(x)), N = 2$

(2) Obyek fungsi yang sama seperti di poin (1) namun dengan $N = 20$

In [4]:
```
# DON'T CHANGE CODE in this cell

# import any needed libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

# this is a given code to plot the results
def plot_results(f, N):
    x = np.linspace(-np.pi, np.pi, 10000)
    [A0, B0] = my_fourier_coef(f, 0)
    y = A0*np.ones(len(x))/2
    for n in range(1, N):
        [An, Bn] = my_fourier_coef(f, n)
        y += An*np.cos(n*x)+Bn*np.sin(n*x)
    plt.figure(figsize = (10,6))
    plt.plot(x, f(x), label = 'analytic')
    plt.plot(x, y, label = 'approximate')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
    plt.legend()
    plt.title(f'{N}th Order Fourier Approximation')
    plt.show()
```

In [5]:
```
# my_fourier_coef(f,n) function (Max Points: 5%)
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

def my_fourier_coef(f, n):
    def a_n(n):
        return (1/np.pi) * quad(lambda x: f(x) * np.cos(n*x), -np.pi, np.pi)[0]

    def b_n(n):
        return (1/np.pi) * quad(lambda x: f(x) * np.sin(n*x), -np.pi, np.pi)[0]

    A_n = a_n(n)
    B_n = b_n(n)
    return A_n, B_n

def plot_results(f, N):
    x = np.linspace(-np.pi, np.pi, 10000)
    [A0, B0] = my_fourier_coef(f, 0)
    y = A0*np.ones(len(x))/2
    for n in range(1, N+1):
        [An, Bn] = my_fourier_coef(f, n)
        y += An*np.cos(n*x) + Bn*np.sin(n*x)
    plt.figure(figsize=(10, 6))
    plt.plot(x, f(x), label='analytic')
    plt.plot(x, y, label='approximate')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
    plt.legend()
    plt.title(f'{N}th Order Fourier Approximation')
    plt.show()


# Test case 1 (Max Points: 3%)
# put your codes here
f = lambda x: np.sin(np.exp(x))
N = 2
plot_results(f, N)


# Another test case 2 - different order (Max Points: 2%)
# put your codes here
N = 20
plot_results(f, N)
```





SOAL 3 (Total Bobot: 20%)

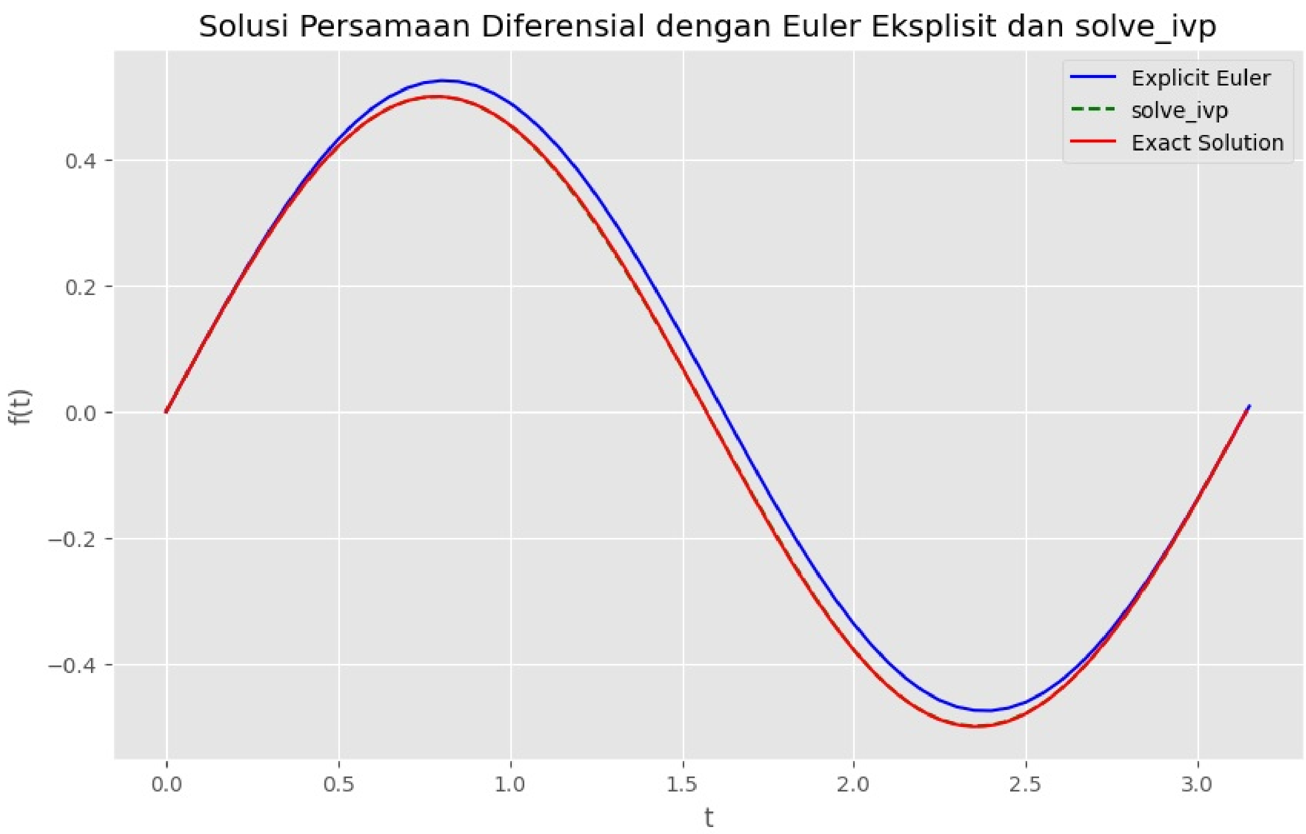
Persamaan diferensial $\frac{df(t)}{dt} = cos(2t)$ dengan kondisi awal $f_0 = 0$ memiliki solusi exact $f(t) = 0.5 * sin(2t)$. Carilah perkiraan solusi terhadap permasalahan nilai awal ini di antara 0 hingga $np.pi$ dengan kenaikan langkah sebesar 0.05 menggunakan dua pendekatan, yakni:

- Explicit Euler Method
- Fungsi built-in **solve_ivp** yang tersedia dalam module scipy.integrate

Selanjutnya, plot hasil perkiraannya baik dengan metode **eksplisit Euler** maupun dengan **solve_ivp** serta solusi **exact**-nya di dalam satu canvas gambar yang sama.

In [6]:
```
# import libraries and define initial parameters (Max Points: 5%)
import numpy as np
import matplotlib.pyplot as plt
```

```python
from scipy.integrate import solve_ivp

def f(t, y):
    return np.cos(2 * t)

def exact_solution(t):
    return 0.5 * np.sin(2 * t)


# Explicit Euler Method (Max Points: 5%)
def explicit_euler(f, t_span, y0, h):
    t_start, t_end = t_span
    t = np.arange(t_start, t_end + h, h)
    n = len(t)
    y = np.zeros(n)
    y[0] = y0
    for i in range(n - 1):
        y[i + 1] = y[i] + h * f(t[i], y[i])
    return t, y

t_span = (0, np.pi)
y0 = 0
h = 0.05
t_explicit, y_explicit = explicit_euler(f, t_span, y0, h)


# using built-in function - solve_ivp (Max Points: 5%)
sol = solve_ivp(f, t_span, [y0], dense_output=True, method='RK45', t_eval=np.arange(t_span[0], t_span[1], h))
t_solve, y_solve = sol.t, sol.y[0]

t_exact = np.linspace(t_span[0], t_span[1], 100)
y_exact = exact_solution(t_exact)


# plot the results - exact solution, approx with Explicit Euler, approx with solve_ivp (Max Points: 5%)
plt.figure(figsize=(10, 6))
plt.plot(t_explicit, y_explicit, 'b-', label='Explicit Euler')
plt.plot(t_solve, y_solve, 'g--', label='solve_ivp')
plt.plot(t_exact, y_exact, 'r-', label='Exact Solution')
plt.xlabel('t')
plt.ylabel('f(t)')
plt.title('Solusi Persamaan Diferensial dengan Euler Eksplisit dan solve_ivp')
plt.legend()
plt.show()
```



SOAL 4 (Total Bobot: 10%)

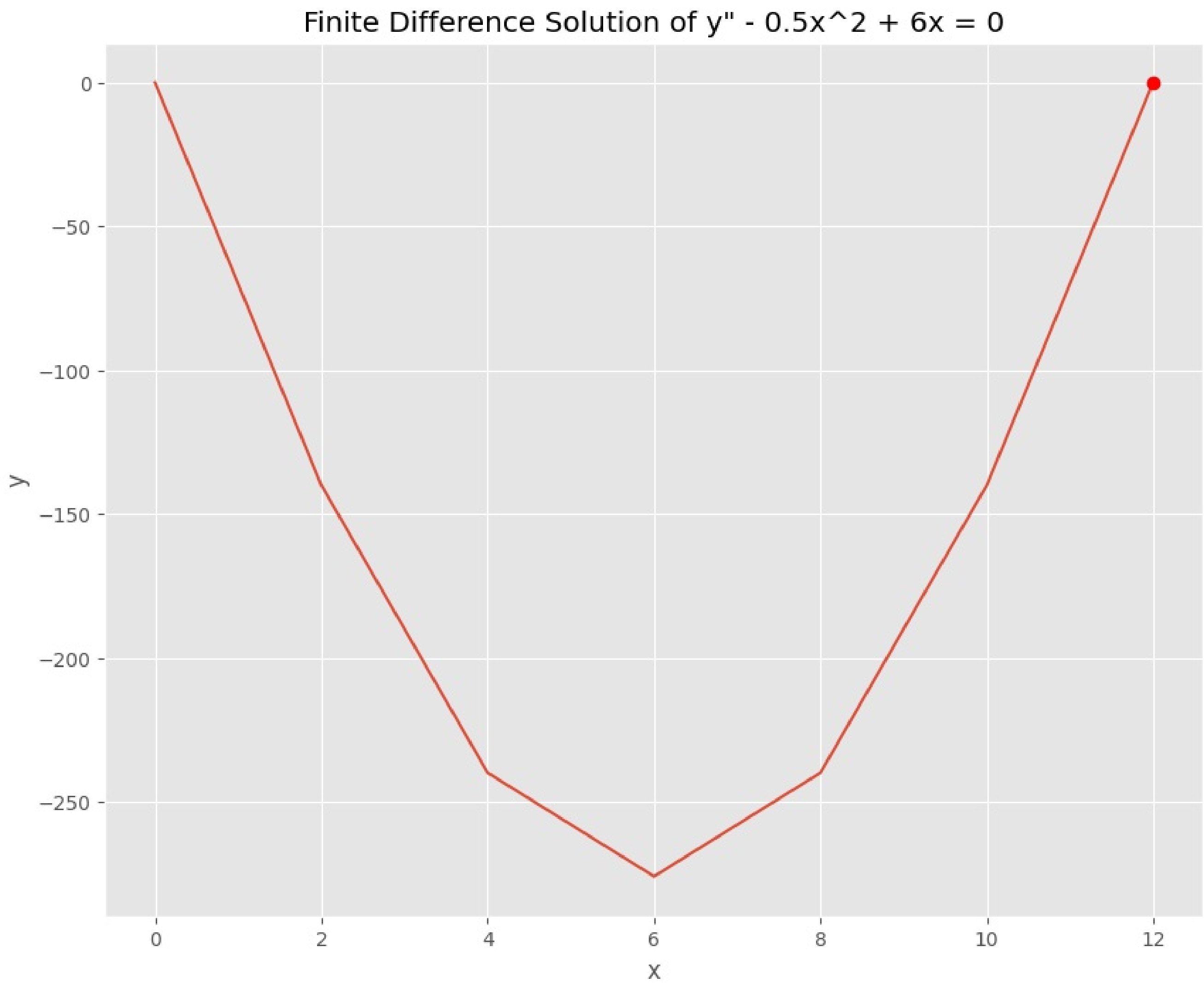
Gunakan metode Finite Difference untuk menyelesaikan permasalahan nilai batas linear berikut: y^"-0.5x^2+6x=0 dengan kondisi batas

$y(0)=0$ dan $y(12)=0$. Ambil $n=6$.

Plot hasilnya bersama dengan kondisi batas yang diberikan!

In [7]:
```
import numpy as np
import matplotlib.pyplot as plt

# initial parameters
n = 6
h = (12-0) / n
x = np.linspace(0, 12, n+1)


# Get A and b (Max Points: 5%)
# put your codes here
# Get A
A = np.zeros((n+1, n+1))
A[0, 0] = 1
A[n, n] = 1

for i in range(1, n):
    A[i, i-1] = 1
    A[i, i] = -2
    A[i, i+1] = 1
print(A)


# Get b
# put your codes here
b = np.zeros(n+1)
b[0] = 0
b[-1] = 0

for i in range(1, n):
    b[i] = (-0.5 * x[i]**2 + 6 * x[i]) * h**2
print(b)


# solve the linear equations and plot the results (Max Points: 5%)
# put your codes here
y = np.linalg.solve(A, b)


# plot the results
# put your codes here
plt.figure(figsize=(10, 8))
plt.plot(x, y)
plt.plot(12, 0, 'ro')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Finite Difference Solution of y" - 0.5x^2 + 6x = 0')
plt.show()
```

```
[[ 1.  0.  0.  0.  0.  0.  0.]
 [ 1. -2.  1.  0.  0.  0.  0.]
 [ 0.  1. -2.  1.  0.  0.  0.]
 [ 0.  0.  1. -2.  1.  0.  0.]
 [ 0.  0.  0.  1. -2.  1.  0.]
 [ 0.  0.  0.  0.  1. -2.  1.]
 [ 0.  0.  0.  0.  0.  0.  1.]]
[ 0. 40. 64. 72. 64. 40.  0.]
```



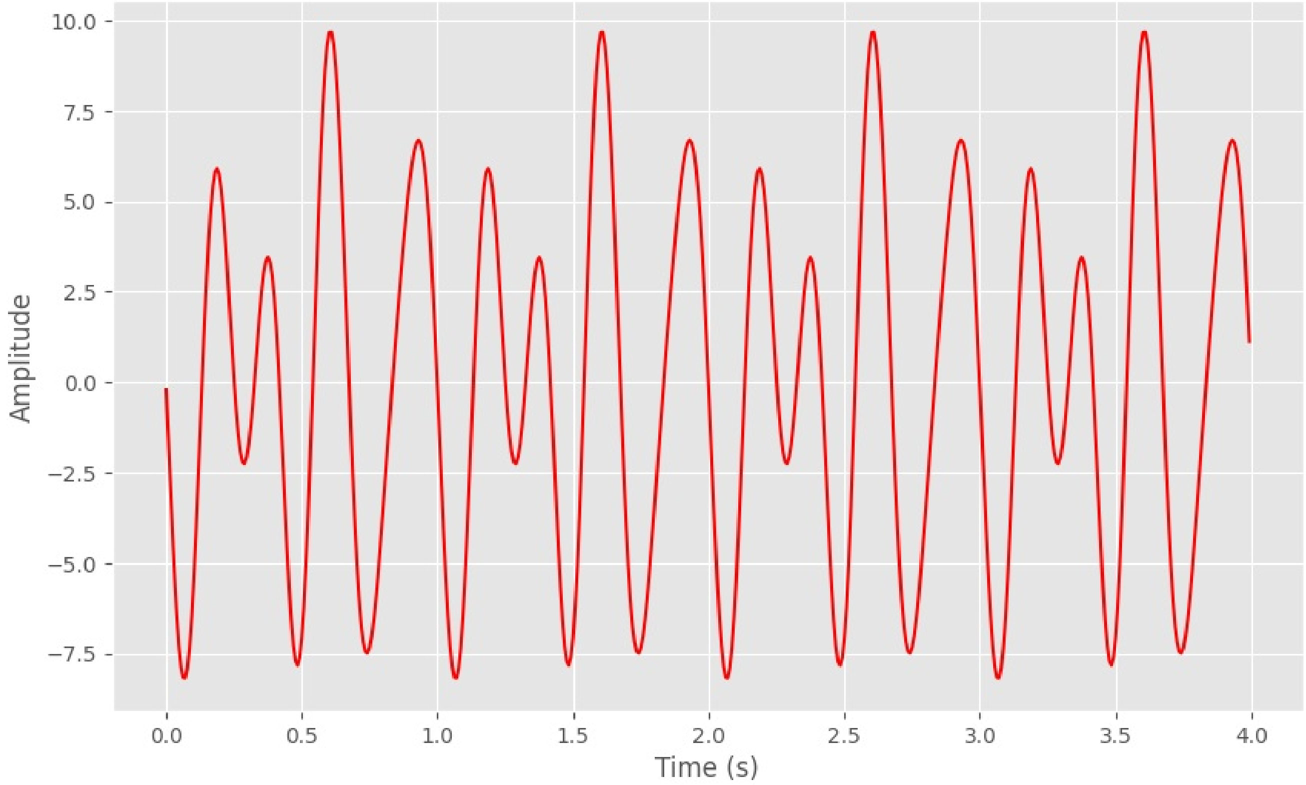
Soal 5 (Bobot: 20%)

Soal 5 A (Bobot: 10%)

Bangkitkan tiga sinyal berikut:
Sinyal 1 adalah gelombang sinus dengan 3 Hz, amplitude 5 dan phase shift 3,
Sinyal 2 adalah gelombang sinus dengan 4 Hz, amplitude 4 dan phase shift -2,
Sinyal 3 adalah gelombang sinus dengan 5 Hz, amplitude 3 dan phase shift 2.
Tambahkan ketiga gelombang ini bersama dengan sampling rate 128 Hz dan plot hasil gabungannya untuk 4 detik.

In [8]:
```
# generate the combined signals (Max Points: 10%)
# put your codes here
import numpy as np
import matplotlib.pyplot as plt
```

```
sr = 128 # Sampling Rate
ts = 1.0 / sr
t = np.arange(0, 4, ts) # 4 Detik

#Sinyal 1 gelombang sinus 3 Hz, amplitude 5, phase shift 3,
s1 = 5 * np.sin(2 * np.pi * 3 * t + 3)
#Sinyal 2 gelombang sinus 4 Hz, amplitude 4, phase shift -2,
s2 = 4 * np.sin(2 * np.pi * 4 * t - 2)
#Sinyal 3 gelombang sinus 5 Hz, amplitude 3, phase shift 2.
s3 = 3 * np.sin(2 * np.pi * 5 * t + 2)

# Kombinasi Signal
combined_signal = s1 + s2 + s3

# Plot Kombinasi Signal
plt.figure(figsize=(10, 6))
plt.plot(t, combined_signal, 'r')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show()
```



Soal 5 B (Bobot: 10%)

Tulis kembali fungsi **Fast Fourier Transform/ FFT(x)** yang telah kita bahas di pertemuan terakhir kelas. Fungsi tersebut akan menghitung FFT dari sinyal yang ada dan mengembalikan nilai-nilai FFT-nya.
Terapkan fungsi FFT ini untuk sinyal yang baru saja kita buat di soal 5 A dan plot hasilnya!

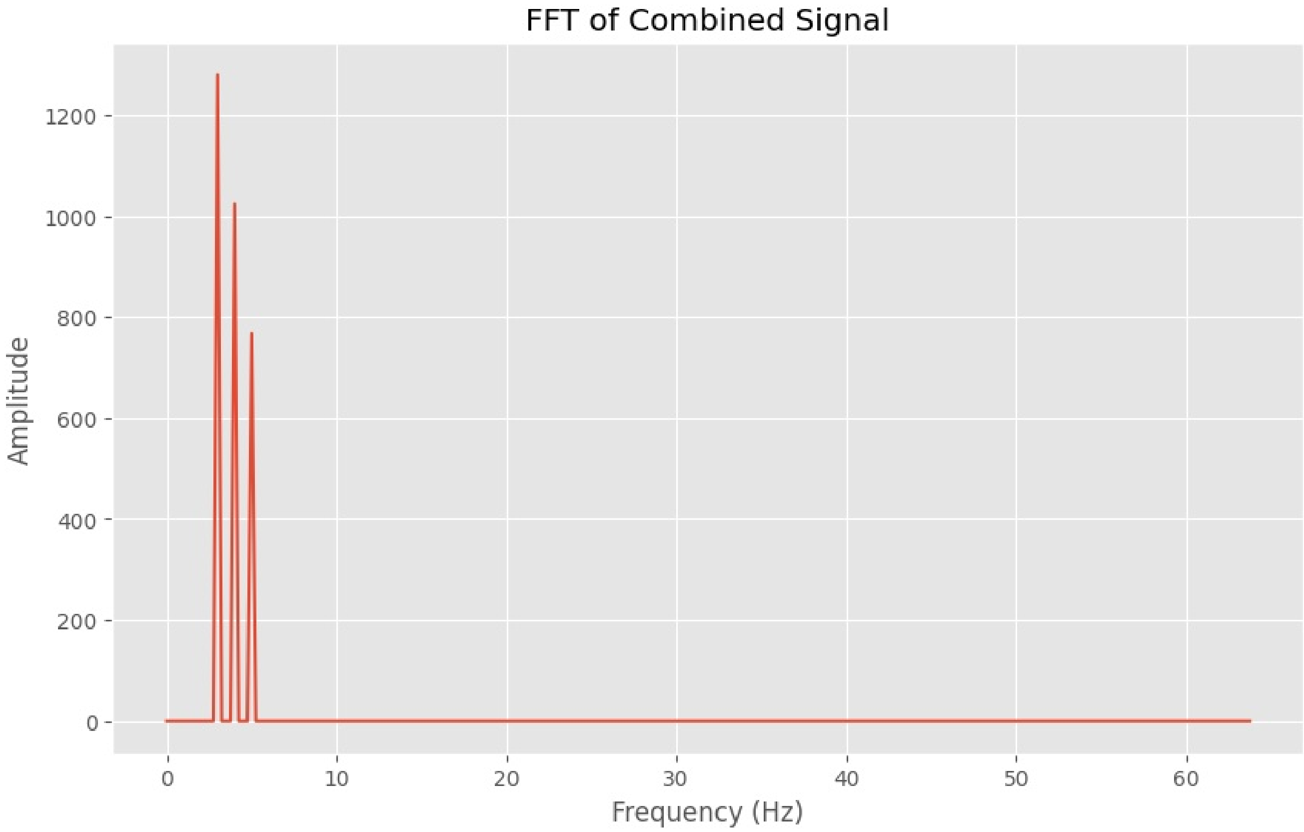
In [9]:

```
# FFT function (Max Points: 5%)
import numpy as np
import matplotlib.pyplot as plt

def FFT(x):
    N = len(x)
    if N <= 1:
        return x
    even = FFT(x[0::2])
    odd = FFT(x[1::2])
    factor = np.exp(-2j * np.pi * np.arange(N) / N)
    return np.concatenate([even + factor[:N//2] * odd, even + factor[N//2:] * odd])

sr = 128  # Sampling rate
ts = 1.0 / sr
t = np.arange(0, 4, ts)  # 4 Detik

# Sinyal 1 gelombang sinus 3 Hz, amplitude 5, phase shift 3,
s1 = 5 * np.sin(2 * np.pi * 3 * t + 3)
# Sinyal 2 gelombang sinus 4 Hz, amplitude 4, phase shift -2,
s2 = 4 * np.sin(2 * np.pi * 4 * t - 2)
# Sinyal 3 gelombang sinus 5 Hz, amplitude 3, phase shift 2.
s3 = 3 * np.sin(2 * np.pi * 5 * t + 2)
```

```
# Kombinasi Signal
combined_signal = s1 + s2 + s3

# Compute the FFT of the combined signal
fft_result = FFT(combined_signal)
```

In [10]:

```
# plot the results (Max Points: 5%)
# put your codes here
plt.figure(figsize=(10, 6))
frequencies = np.fft.fftfreq(len(fft_result), ts)
plt.plot(frequencies[:len(frequencies)//2], np.abs(fft_result)[:len(frequencies)//2])
plt.title('FFT of Combined Signal')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.show()
```



Loading [MathJax]/jax/element/mml/optable/BasicLatin.js