Ejercicios de ecuaciones diferenciales de primer grado

Se basa en las siguientes fórmulas iterativas:

a) $K_1 = f(x_n, y_n)$
b) $K_2 = f(x_n + h/2, y_n + h*K_1/2)$
c) $K_3 = f(x_n + h/2, y_n + h*K_2/2)$
d) $K_4 = f(x_n + h, y_n + (K_3 * h)$

Calcular la aproximación de y, incluyendo el valor de x:

e) $Y_{n+1} = y_n + (1/6) * (K_1 + 2* K_2 + 2*K_3 + K_4) * h$
f) $X_{n+1} = x_n + h$

Ejercicios

1. y'= x, con su condición inicial y (0) =0 y valor h= 0.3

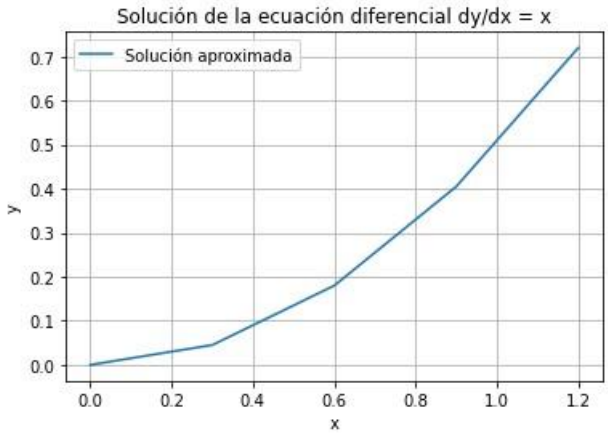
n	Xn	Yn	k1	k2	k3	k4	Yn+1
0	0	0	0	0.045	0.045	0.3	0.024
1	0.3	0.024	0.3	0.135	0.135	0.6	0.264
2	0.6	0.264	0.6	0.225	0.225	0.9	0.664
3	0.9	0.664	0.9	0.315	0.315	1.2	1.224
4	1.2	1.224	1.2	0.405	0.405	1.5	1.944

Código en Python de manera numérica

```
import numpy as np

import matplotlib.pyplot as plt


def runge_kutta_4(f, x0, y0, h, n):
    """
    Método de Runge-Kutta de cuarto orden para resolver ecuaciones diferenciales ordinarias.

    :param f: Función que describe la ecuación diferencial (dy/dx = f(x, y)).
    :param x0: Valor inicial de x.
    :param y0: Valor inicial de y.
    :param h: Tamaño del paso.
    :param n: Número total de pasos.
    :return: Arrays con los valores de x y y.
    """
    x_values = [x0]
    y_values = [y0]
```

```
    for i in range(n):
        k1 = h * f(x_values[-1], y_values[-1])
        k2 = h * f(x_values[-1] + h/2, y_values[-1] + k1/2)
        k3 = h * f(x_values[-1] + h/2, y_values[-1] + k2/2)
        k4 = h * f(x_values[-1] + h, y_values[-1] + k3)

        x_values.append(x_values[-1] + h)
        y_values.append(y_values[-1] + (k1 + 2*k2 + 2*k3 + k4) / 6)

    return np.array(x_values), np.array(y_values)

def f(x, y):
    """
    Definición de la función diferencial: dy/dx = x

    :param x: Valor de x.
    :param y: Valor de y.
    :return: Valor de la función en (x, y).
    """
    return x

# Parámetros iniciales
x0 = 0
y0 = 0
h = 0.3  # Tamaño del paso
n = 4  # Número total de pasos

# Resolver la ecuación diferencial
x_values, y_values = runge_kutta_4(f, x0, y0, h, n)

# Graficar la solución
```

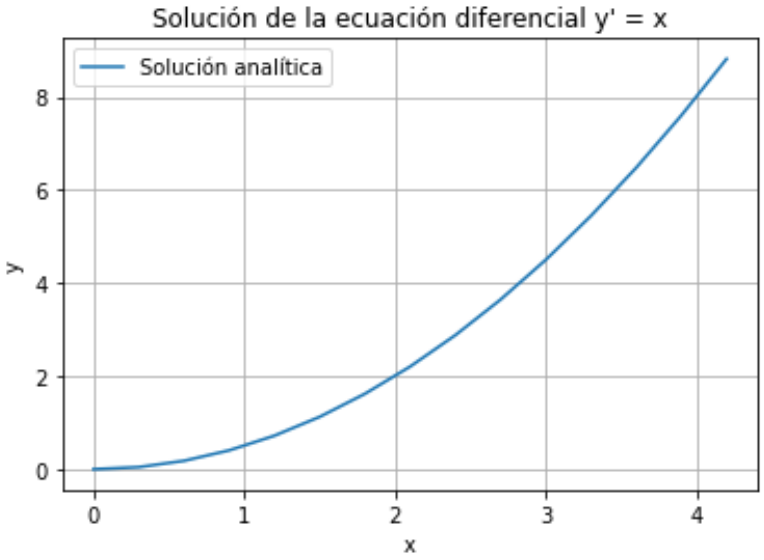
```
plt.plot(x_values, y_values, label='Solución aproximada')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Solución de la ecuación diferencial dy/dx = x')
plt.legend()
plt.grid(True)
plt.show()
```



Código en Python de manera analítica

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

# Definir la solución analítica de la ecuación diferencial
def analytical_solution(x):
    return x**2 / 2

# Condiciones iniciales y tamaño de paso
x0 = 0
y0 = 0
h = 0.3
xmax = 4

# Generar los valores de x
x_values = np.arange(x0, xmax + h, h)
```

```
y_values = analytical_solution(x_values)


# Graficar la solución
plt.plot(x_values, y_values, label='Solución analítica')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Solución de la ecuación diferencial y\' = x')
plt.legend()
plt.grid(True)
plt.show()
```



2. y'= 3*(x^2), con su condición inicial y (0) = -3 y valor h= 0.2

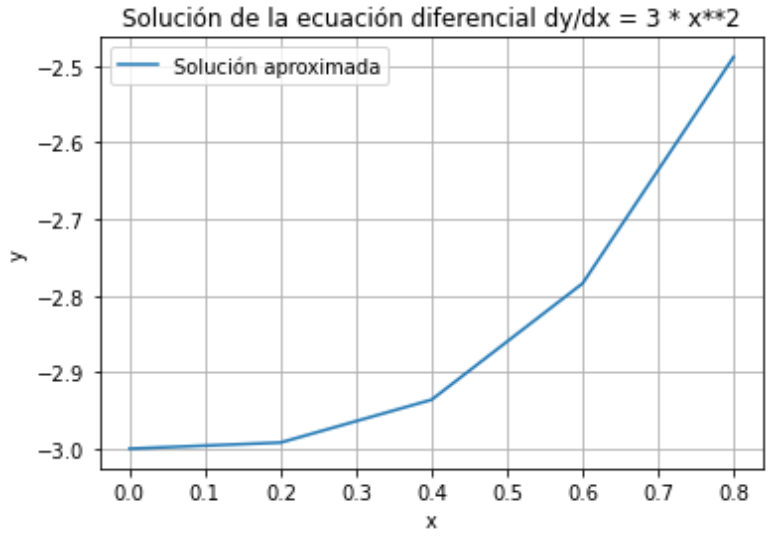
n	Xn	Yn	k1	k2	k3	k4	Yn+1
0	0	-3	0	0.03	0.03	0.12	-2.992
1	0.2	-2.992	0.12	0.27	0.27	0.48	-2.936
2	0.4	-2.936	0.48	0.75	0.75	1.08	-2.784
3	0.6	-2.784	1.08	1.47	1.47	1.92	-2.488
4	0.8	-2.488	1.92	2.43	2.43	3	-2

Código en Python de manera numérica

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

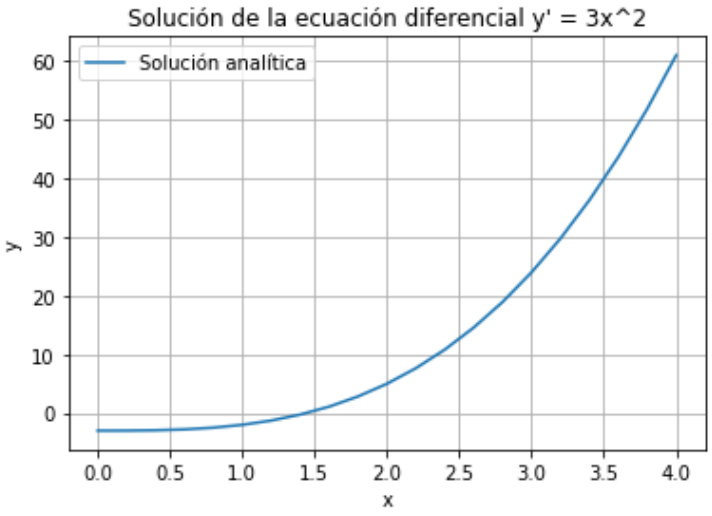
```
def runge_kutta_4(f, x0, y0, h, n):
    """
    Método de Runge-Kutta de cuarto orden para resolver ecuaciones diferenciales ordinarias.

    :param f: Función que describe la ecuación diferencial (dy/dx = f(x, y)).
    :param x0: Valor inicial de x.
    :param y0: Valor inicial de y.
    :param h: Tamaño del paso.
    :param n: Número total de pasos.
    :return: Arrays con los valores de x y y.
    """
    x_values = [x0]
    y_values = [y0]

    for i in range(n):
        k1 = f(x_values[-1], y_values[-1])
        k2 = f(x_values[-1] + h/2, y_values[-1] + ((h*k1)/2))
        k3 = f(x_values[-1] + h/2, y_values[-1] + ((h*k2)/2))
        k4 = f(x_values[-1] + h, y_values[-1] + h*k3)

        x_values.append(x_values[-1] + h)
        y_values.append(y_values[-1] + (h/6)*(k1 + 2*k2 + 2*k3 + k4))

    return np.array(x_values), np.array(y_values)

def f(x, y):
    """
    Definición de la función diferencial: dy/dx = 3 * x**2

    :param x: Valor de x.
    :param y: Valor de y.
    :return: Valor de la función en (x, y).
```

```
    """
    return 3 * x**2

# Parámetros iniciales
x0 = 0
y0 = -3
h = 0.2  # Tamaño del paso
n = 4  # Número total de pasos

# Resolver la ecuación diferencial
x_values, y_values = runge_kutta_4(f, x0, y0, h, n)

# Graficar la solución
plt.plot(x_values, y_values, label='Solución aproximada')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Solución de la ecuación diferencial dy/dx = 3 * x**2')
plt.legend()
plt.grid(True)
plt.show()
```



Código en Python de manera analítica

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

# Definir la solución analítica de la ecuación diferencial
def analytical_solution(x):
    return x**3 - 3

# Condiciones iniciales y tamaño de paso
x0 = 0
y0 = -3
h = 0.2
xmax = 4

# Generar los valores de x
x_values = np.arange(x0, xmax + h, h)
y_values = analytical_solution(x_values)

# Graficar la solución
plt.plot(x_values, y_values, label='Solución analítica')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Solución de la ecuación diferencial y\' = 3x^2')
plt.legend()
plt.grid(True)
plt.show()
```



3. y'= y, con su condición inicial y (0) =1 y valor h= 0.04

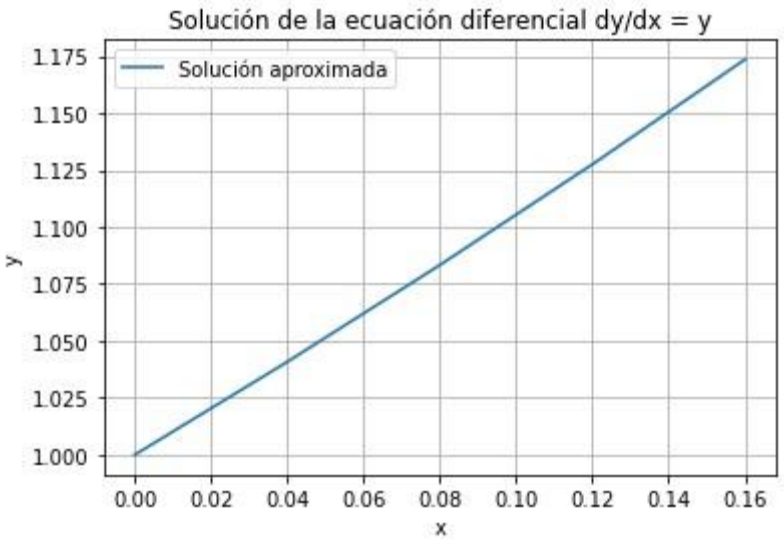
n	Xn	Yn	k1	k2	k3	k4	Yn+1
0	0	1	1	1.02	1.0204	1.040816	1.04081077
1	0.04	1.04081077	1.04081077	1.06162699	1.06204331	1.08329251	1.08328707
2	0.08	1.08328707	1.08328707	1.10495281	1.10538612	1.12750251	1.12749685
3	0.12	1.12749685	1.12749685	1.15004679	1.15049778	1.17351676	1.17351087
4	0.16	1.17351087	1.17351087	1.19698108	1.19745049	1.22140889	1.22140275

Código en Python de manera numérica

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


def runge_kutta_4(f, x0, y0, h, n):
    """
    Método de Runge-Kutta de cuarto orden para resolver ecuaciones diferenciales ordinarias.

    :param f: Función que describe la ecuación diferencial (dy/dx = f(x, y)).
    :param x0: Valor inicial de x.
    :param y0: Valor inicial de y.
    :param h: Tamaño del paso.
    :param n: Número total de pasos.
```

```
    :return: Arrays con los valores de x y y.
    """
    x_values = [x0]
    y_values = [y0]

    for i in range(n):
        k1 = f(x_values[-1], y_values[-1])
        k2 = f(x_values[-1] + h/2, y_values[-1] + ((h*k1)/2))
        k3 = f(x_values[-1] + h/2, y_values[-1] + ((h*k2)/2))
        k4 = f(x_values[-1] + h, y_values[-1] + h*k3)

        x_values.append(x_values[-1] + h)
        y_values.append(y_values[-1] + (h/6)*(k1 + 2*k2 + 2*k3 + k4))

    return np.array(x_values), np.array(y_values)

def f(x, y):
    """
    Definición de la función diferencial: dy/dx = y

    :param x: Valor de x.
    :param y: Valor de y.
    :return: Valor de la función en (x, y).
    """
    return y

# Parámetros iniciales
x0 = 0
y0 = 1
h = 0.04  # Tamaño del paso
n = 4  # Número total de pasos
```

```
# Resolver la ecuación diferencial
x_values, y_values = runge_kutta_4(f, x0, y0, h, n)

# Graficar la solución
plt.plot(x_values, y_values, label='Solución aproximada')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Solución de la ecuación diferencial dy/dx = y')
plt.legend()
plt.grid(True)
plt.show()
```



Código en Python de manera analítica

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

# Definir la solución analítica de la ecuación diferencial
def analytical_solution(x):
    return np.exp(x)

# Condiciones iniciales y tamaño de paso
x0 = 0
```

```
y0 = 1

h = 0.04

xmax = 4


# Generar los valores de x

x_values = np.arange(x0, xmax + h, h)

y_values = analytical_solution(x_values)


# Graficar la solución

plt.plot(x_values, y_values, label='Solución analítica')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Solución de la ecuación diferencial y\' = y')

plt.legend()

plt.grid(True)

plt.show()
```