Universidad Alfonso X el Sabio
Martina García González, Lucía Mielgo Torres
Grado en Ingeniería Matemática
AMPLIACIÓN DE LOS MÉTODOS NUMÉRICOS

Taller 7: Métodos Runge-Kutta

28 de enero de 2025

1. Introducción

1. Desarrolle una función en Python para calcular la integral de una ecuación $\frac{dy}{dt} = f(y,t)$ donde $t \in [t_0, t_f]$ con $y(t_0) = y_0$ mediante el método Runge-Kutta de orden 4, dando como variables $f(y,t), t_0, t_f, y_0$, y N siendo N el número de pasos.

2. Implemente computacionalmente un programa que resuelva la ecuación diferencial $\frac{dy}{dt} = 4e^{0{,}8t} - 0{,}5y$, donde $t \in [0,4]$, $y(0) = 2$, mediante el método Runge-Kutta de orden 4, utilizando los números de paso de $h = 1$, $h = 0{,}5$ y $h = 0{,}1$. Compare los resultados con los obtenidos en el taller 6.

2. Métodos y modelos matemáticos

El método de Runge-Kutta de orden 4 (RK4) es un método numérico utilizado para resolver ecuaciones diferenciales ordinarias de la forma $\frac{dy}{dt} = f(t,y)$

Dado un intervalo $[t_0, t_f]$ y un valor inicial $y(t_0) = y_0$, el método RK4 calcula una aproximación de $y(t_f)$ siguiendo estos pasos:

- Definir el tamaño del paso h:

  El intervalo se divide en N pasos, donde h se calcula como:

$$h = \frac{t_f - t_0}{N} \tag{1}$$

- Iterar para cada paso:

  Para cada paso i desde 0 hasta $N-1$:

- Calcular el valor actual de t:

$$t_i = t_0 + i \cdot h \tag{2}$$

- Calcular las pendientes:

$$k_1 = h \cdot f(t_i, y_i) \tag{3}$$

$$k_2 = h \cdot f\left(t_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right) \tag{4}$$

$$k_3 = h \cdot f\left(t_i + \frac{h}{2}, y_i + \frac{k_2}{2}\right) \tag{5}$$

$$k_4 = h \cdot f(t_i + h, y_i + k_3) \tag{6}$$

- Actualizar el valor de y:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{7}$$

- Al finalizar la iteración, el valor y_N será la aproximación de la solución $y(t_f)$.

3. Resultados

Los resultados del taller anterior con h=0'1, h=0'01 y h=0'001 son los siguientes:

Cuadro 1: Comparación de los Métodos para $h = 0{,}1$

t	**Solución Analítica**	**Método de Heun**
0.0	2.000000	2.000000
0.1	2.308790	2.309157
0.2	2.636362	2.637113
0.3	2.984620	2.985770
0.4	3.355606	3.357177

Los resultados que nos da con método de Runge-Kutta para h=1, h=0'5 y h=0'1 son los siguientes:

Cuadro 2: Comparación de los Métodos para $h = 0{,}01$

t	Solución Analítica	Método de Heun
0.0	2.000000	2.000000
0.01	2.023089	2.023089
0.02	2.046479	2.046481
0.03	2.070174	2.070177
0.04	2.094182	2.094187

Cuadro 3: Comparación de los Métodos para $h = 0{,}001$

t	Solución Analítica	Método de Heun
0.0	2.000000	2.000000
0.001	2.002308	2.002308
0.002	2.004621	2.004621
0.003	2.006939	2.006939
0.004	2.009261	2.009261

Cuadro 4: Comparación de los Métodos para $h = 1$

t	Solución Analítica	Método de Runge-Kutta
0.0	2.000000	2.000000
1.0	6.194631	6.201037
2.0	14.843922	14.862484
3.0	33.677172	33.721348
4.0	75.338963	75.439172

Cuadro 5: Comparación de los Métodos para $h = 0{,}5$

t	Solución Analítica	Método de Runge-Kutta
0.0	2.000000	2.000000
0.5	3.751521	3.751699
1.0	6.194631	6.195042
1.5	9.707042	9.707772
2.0	14.843922	14.845106

Cuadro 6: Comparación de los Métodos para $h = 0{,}1$

t	Solución Analítica	Método de Runge-Kutta
0.0	2.000000	2.000000
0.1	2.308790	2.308790
0.2	2.636362	2.636362
0.3	2.984620	2.984620
0.4	3.355606	3.355606

4. Discusión

Comparando los resultados obtenidos con el método de Runge-Kutta y los del método de Heun, podemos notar que ambos métodos producen resultados similares, especialmente para tamaños de paso más pequeños (h=0.1 o menores). Esto se debe a que tanto el método de Runge-Kutta de cuarto orden como el de Heun (una versión mejorada del método de Euler) son métodos de integración numérica precisos que tienden a converger hacia la solución analítica conforme se disminuye el tamaño del paso.

Sin embargo, para tamaños de paso grandes (h=1), la diferencia entre el método de Heun y el método de Runge-Kutta puede ser más notable, ya que el método de Runge-Kutta tiene un orden de precisión más alto (cuarto orden) y, por lo tanto, puede manejar mejor los errores acumulativos. En general, los resultados parecen razonables y consistentes con lo que se esperaría, dado que ambas soluciones tienden hacia la solución analítica al reducir h.

5. Conclusiones

Podemos concluir que el método de Runge-Kutta de cuarto orden ofrece una mayor precisión en la aproximación de la solución de ecuaciones diferenciales en comparación con métodos de orden inferior, como el de Heun. A medida que se reduce el tamaño del paso uh, tanto el método de Heun como el de Runge-Kutta convergen hacia la solución analítica, lo que confirma que la elección de un tamaño de paso adecuado es crucial para obtener resultados precisos en métodos numéricos. Sin embargo, Runge-Kutta demuestra ser más robusto y menos sensible al tamaño del paso, lo que lo hace preferible para problemas donde la precisión es clave.

6. Referencias

Douglas J. Faires and Richard L. Burden, *Análisis Numérico*, Cengage Learning, 1998.

7. Anexos

Las funciones utilizadas para realizar los ejercicios son las siguientes.

Función del Método de Runge-Kutta

```
def runge_kutta_4(f, t0, tf, y0, N):
    # Calcula el tamaño del paso h basándose en los límites de integración
    h = (tf - t0) / N

    # Crea un array de valores de t que va desde t0 hasta tf con N+1 puntos
```

```
t_values = np.linspace(t0, tf, N + 1)

# Inicializa un array para almacenar los valores de y, con tamaño N+1
y_values = np.zeros(N + 1)

# Asigna el valor inicial y0 al primer elemento de y_values
y_values[0] = y0

# Itera sobre el número de pasos N
for i in range(N):
    # Obtiene el valor actual de t y y
    t = t_values[i]
    y = y_values[i]

    # Calcula las pendientes k1, k2, k3, y k4 usando la función f
    k1 = h * f(t, y)  # Pendiente en el inicio del intervalo
    k2 = h * f(t + h/2, y + k1/2)  # Pendiente en el medio del intervalo usando k1
    k3 = h * f(t + h/2, y + k2/2)  # Pendiente en el medio del intervalo usando k2
    k4 = h * f(t + h, y + k3)  # Pendiente al final del intervalo usando k3

    # Actualiza el valor de y para el siguiente paso
    y_values[i + 1] = y + (k1 + 2*k2 + 2*k3 + k4) / 6

# Devuelve los arrays de valores de t y de y
return t_values, y_values
```