



# Métodos de Runge-Kutta

Camargo Badillo Luis Mauricio

Audiciones Fase 3  
**Programa de Inducción a Matemáticas Aplicadas y Computación**

# Índice

<b>1. Introducción a los Métodos de Runge-Kutta</b>	<b>2</b>
<b>2. Método de Runge-Kutta de Grado 4</b>	<b>3</b>
2.1. Funcionamiento de RK4 . . . . .	3
2.2. Visualización de RK4 . . . . .	4
2.2.1. Pendientes a lo Largo del Intervalo . . . . .	4
2.2.2. Aproximación Obtenida . . . . .	5
<b>3. Aplicación de RK4</b>	<b>5</b>
3.1. Oscilador de Van der Pol . . . . .	5
3.2. Van der Pol en Términos de Ecuaciones de Primer Orden . . . . .	6
3.3. Solución con RK4 . . . . .	6
3.3.1. Ejemplo 1 . . . . .	7
3.3.2. Ejemplo 2 . . . . .	9
3.3.3. Ejemplo 3 . . . . .	10
3.4. Implementación en Python . . . . .	11
<b>4. Conclusión</b>	<b>12</b>

Recordemos que una **ecuación diferencial** es una ecuación que relaciona una (o más) funciones con sus derivadas. Estas ecuaciones son de gran utilidad, pues nos ayudan a modelar muchos fenómenos naturales, el objeto de estudio de muchas disciplinas como la física, la ingeniería, la química, la biología, la astronomía, etc.; las ecuaciones diferenciales también modelan otros fenómenos de la economía y otras ciencias matemáticas. Es por esta razón por la que la resolución de ecuaciones diferenciales es un área de tanto interés en el mundo actual.

Usualmente, al tener una ecuación diferencial, el objetivo es encontrar la función (o las funciones) que la satisfagan. No obstante, dependiendo de la complejidad de la ecuación y de su forma, puede ser difícil o incluso imposible encontrar la solución exacta y explícita de forma analítica, un problema que se presenta sin importar si buscamos la solución manualmente o le asignamos la tarea a una computadora. De hecho, son bastantes las aplicaciones que involucran ecuaciones diferenciales cuya solución no se puede obtener de forma analítica; las ecuaciones que definen la forma del ala de un avión son un ejemplo.

Es por esta problemática por lo que nos apoyamos de los métodos numéricos para resolver este tipo de ecuaciones diferenciales. Recordemos que los métodos numéricos son técnicas que nos ayudan a aproximar procesos matemáticos y que se han vuelto especialmente populares luego del *boom* en poder computacional que se ha experimentado en las últimas décadas; la resolución de esta clase de ecuaciones diferenciales son tan solo una de las muchas aplicaciones de los métodos numéricos.

Evidentemente, dada la naturaleza de los métodos numéricos, no tendremos una solución exacta como la analítica, pero ciertamente podemos tener una solución lo suficientemente cercana como para poder confiar en ella. En el mundo actual, podemos constatar que las soluciones obtenidas son perfectamente válidas: miles de aviones vuelan todos los días a pesar de que las ecuaciones que modelan la aerodinámica deben ser resueltas numéricamente.

Los **métodos de Runge-Kutta** son algunos de los métodos numéricos más ampliamente utilizados para la resolución de estas ecuaciones diferenciales, dado un problema de valor inicial. Estos métodos proporcionan soluciones más precisas que aquellas dadas por otros métodos numéricos más sencillos, como el método de Euler. A lo largo de este documento, se estará dando un panorama general de estos métodos, sus diferentes tipos y cómo utilizarlos, cerrando con una aplicación.

## 1. Introducción a los Métodos de Runge-Kutta

Los métodos de Runge-Kutta son una familia de métodos numéricos iterativos desarrollados por los matemáticos alemanes Carl Runge y Wilhelm Kutta alrededor de los inicios del siglo XX.

Estos métodos tienen como objetivo obtener una curva aproximada a la solución de un problema de valor inicial que involucra una ecuación diferencial *ordinaria*, especialmente cuando se trata del tipo de problemas cuya solución analítica es difícil o imposible de obtener, como ya fue establecido en la introducción de este texto. Debido a su relativamente rápida convergencia en relación a otros métodos, los métodos de Runge-Kutta son ampliamente preferidos, a pesar de que son computacionalmente más demandantes,

De forma similar a otros métodos numéricos utilizados para las ecuaciones diferenciales, los métodos de Runge-Kutta son iterativos, lo que quiere decir que una primera estimación es utilizada para obtener una segunda más precisa y así sucesivamente.

## 2. Método de Runge-Kutta de Grado 4

El método más utilizado de los Runge-Kutta es el de grado 4, frecuentemente conocido como «RK4»; incluso algunas veces es simplemente llamado el método de Runge-Kutta.

### 2.1. Funcionamiento de RK4

Sea la siguiente ecuación diferencial:

$$\frac{dy}{dt} = f(t, y) \quad (2.1)$$

... con la condición de valor inicial:

$$y(t_0) = y_0$$

Aquí conocemos  $f$  y las condiciones inicales  $t_0, y_0$ , deseando aproximar a  $y$ , una función desconocida sobre  $t$ . Utilizaremos el método RK4 para obtener esta aproximación.

Antes que nada, debemos:

- Definir el intervalo para  $t$  sobre el cual estaremos aproximando la solución.
- Definir el tamaño  $h$  de los pasos por iteración que estaremos tomando sobre ese intervalo.

El número de iteraciones que realizaremos dependerá de la razón entre el intervalo para  $t$  y el tamaño  $h$  que elijamos. Aquí, estamos básicamente convirtiendo el intervalo continuo de  $t$  en uno discreto a través de  $h$ , sobre el cual estaremos iterando en el método.

Comenzamos definiendo los dos valores más importantes que se estarán determinando durante cada iteración  $n \in [0, 1, 2, \dots]$ :

$$y_{n+1} = y_n + \frac{h}{6} (k_1 + 2k_2 + 2k_3 + k_4) \quad (2.2)$$

$$t_{n+1} = t_n + h$$

Es en la definición de  $y_{n+1}$  que se nos presentan los valores  $k_1, k_2, k_3, k_4$ , que son los que caracterizan al método RK4 y donde realizaremos la mayor cantidad de cálculos involucrados. Estos cuatro valores se definen para cada iteración  $n$  como:

$$k_1 = f(t_n, y_n) \quad (2.3)$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right) \quad (2.4)$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right) \quad (2.5)$$

$$k_4 = f(t_n + h, y_n + hk_3) \quad (2.6)$$

## 2.2. Visualización de RK4

Puede que hasta el momento la descripción anterior del método de Runge-Kutta de grado 4 (RK4) haya parecido bastante abstracta, pero describámosla y visualicémosla para que su funcionamiento quede más claro.

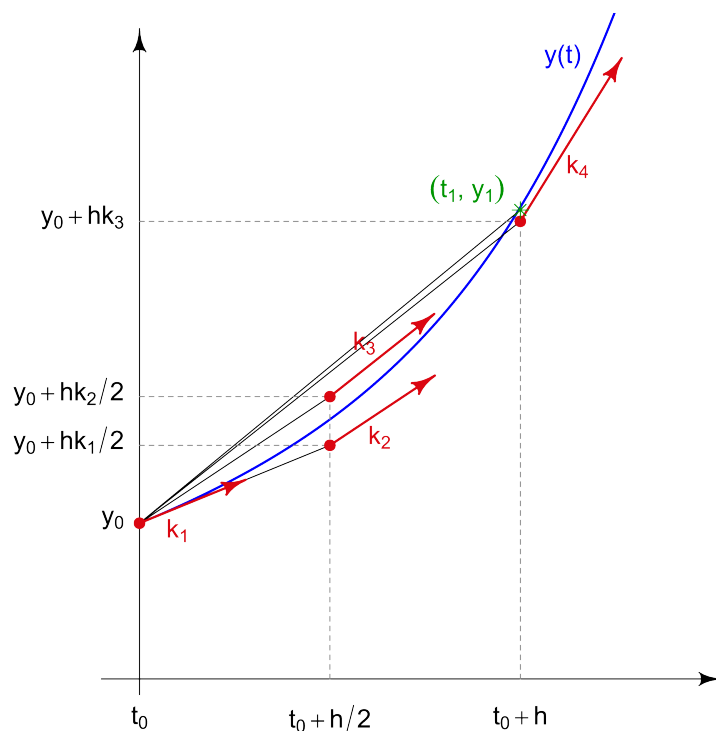


Figura 1: Aproximación de la solución a  $\frac{dy}{dt} = y + t^3$  con  $y_0$  y un intervalo para  $t$  arbitrarios. En azul, la solución exacta; en rojo, las pendientes correspondientes a los valores  $k_i$ ; en verde, uno de los puntos de la solución, que corresponde a la aproximación obtenida durante la primera iteración con paso  $h$ . Gráfico obtenido de [https://en.wikipedia.org/wiki/File:Runge-Kutta\\_slopes.svg](https://en.wikipedia.org/wiki/File:Runge-Kutta_slopes.svg).

La figura 2.2 representa el funcionamiento de RK4 para la obtención de la solución a la ecuación diferencial  $\frac{dy}{dt} = y + t^3$  bajo la condición  $y(t_0) = y_0$ , con  $y_0$  y el intervalo para  $t$  arbitrarios.

La obtención analítica de la solución exacta para esta ecuación diferencial es, de hecho, posible y es por ello por lo que pudo ser trazada como la línea azul, por lo que estrictamente hablando no es necesaria la implementación de un método numérico para obtenerla. Sin embargo, se ha utilizado simplemente para ejemplificar y visualizar el funcionamiento del método RK4.

### 2.2.1. Pendientes a lo Largo del Intervalo

Lo primero que notamos es que los valores  $k_i$  representan pendientes. Si regresamos a su definición en la sección anterior ((2.3), (2.4), (2.5) y (2.6)), notamos que son simplemente evaluaciones de  $f(t, y)$  en valores concretos para  $t$  y  $y$ , y que la ecuación diferencial de la que

partimos establece que esa función describe la derivada (véase (2.1)), por lo que la razón de ser de este resultado se vuelve evidente.

Analizando un poco más las definiciones de estas  $k_i$ , y con ayuda de la figura 2.2, notaremos también que estas pendientes no son del todo arbitrarias:

- $k_1$  es la pendiente al inicio del intervalo.
- $k_2$  es la pendiente en el punto medio del intervalo (de ahí que involucre  $\frac{h}{2}$ ), apoyándose de  $k_1$ .
- $k_3$ , similar a  $k_2$ , es la pendiente en el punto medio del intervalo, pero esta vez apoyándose de  $k_2$ .
- $k_4$  es la pendiente al final del intervalo

### 2.2.2. Aproximación Obtenida

Al final de cada iteración  $n$  obtenemos  $y_{n+1}$ , que no es más que la aproximación de  $y(t_{n+1})$  según el método.

Notemos que este valor es determinado por la  $y_n$  actual correspondiente a la iteración, sumada con una media ponderada de cuatro incrementos, cada uno correspondiendo al producto de las cuatro pendientes anteriormente descritas por la longitud del paso  $h$ . Esta media ponderada favorece sobre todo a las pendientes  $k_2$  y  $k_3$ , las del medio del intervalo.

Visto de otra forma, las pendientes  $k_i$  descritas anteriormente van guiando al método a lo largo del intervalo, lo que le permite obtener una aproximación relativamente precisa de un punto de la solución al final de él.

## 3. Aplicación de RK4

Ya se introdujo a los métodos numéricos de Runge-Kutta y se describió el más popular, RK4. A continuación, para seguir con el desarrollo de este método, se describirá una aplicación en la cual este método puede ser aplicado.

### 3.1. Oscilador de Van der Pol

Recordemos que un oscilador es un sistema dinámico que presenta una especie de variación estable y periódica a lo largo del tiempo. Uno de los osciladores más sencillos es el oscilador armónico amortiguado, que es descrito por la siguiente ecuación diferencial ordinaria de segundo orden:

$$\frac{d^2y}{dx^2} + \mu \frac{dy}{dx} + y = 0$$

Aquí,  $\mu$  es un parámetro escalar que controla el amortiguamiento del sistema (sobreamortiguamiento, amortiguamiento crítico o subamortiguamiento). Las soluciones a esta ecuación diferencial pueden ser obtenidas analíticamente de forma relativamente sencilla, así que no nos enfocaremos en esta aplicación.

En lugar, preferiremos como sujeto de estudio al **oscilador de Van der Pol**. Este tipo de oscilador, nombrado en honor al físico e ingeniero eléctrico neerlandés Balthasar van der Pol, tiene una amortiguación no lineal, lo que resulta en un comportamiento curioso que a lo largo del tiempo es descrito por la siguiente ecuación diferencial ordinaria de segundo orden:

$$\frac{d^2x}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x = 0 \quad (3.1)$$

...donde  $x$  representa la coordenada de la posición (en función de  $t$ ) y  $\mu$  es un parámetro escalar que controla la no linealidad y la fuerza del amortiguamiento. Esta ecuación es algunas veces llamada simplemente la «ecuación de Van der Pol».

Esta ecuación tiene muchas aplicaciones, que van desde la física y la electrónica hasta la biología. Originalmente fue utilizada para describir las oscilaciones de un circuito de tubos de vacío, pero más recientemente ha sido utilizada para describir el comportamiento de las neuronas y para el desarrollo de un modelo que describe la interacción entre dos placas geológicas, solo por citar un par de ejemplos.

Debido a que la ecuación de Van der Pol no es lineal, es imposible la obtención de una solución analítica exacta en términos de funciones elementales conocidas. Por ello, el único camino que tenemos para analizar su solución son los métodos numéricos. Utilizaremos el método RK4 para encontrar numéricamente las soluciones a esta ecuación.

### 3.2. Van der Pol en Términos de Ecuaciones de Primer Orden

Antes de poder utilizar el método de Runge-Kutta de grado 4 para la obtención de las soluciones numéricas de la ecuación de Van der Pol, debemos notar que no podemos aplicarlo de manera directa sobre la ecuación (3.1), pues, recordemos que, tal como fue descrito en la sección 2.1, el método necesita de una ecuación de *primer* orden de la forma de (2.1).

No obstante, podemos establecer  $y = \frac{dx}{dt}$ , y así podemos reescribir la ecuación (3.1) como un sistema de dos ecuaciones diferenciales ordinarias de primer orden:

$$\begin{cases} \frac{dx}{dt} = y \\ \frac{dy}{dt} = \mu(1 - x^2)y - x \end{cases} \quad (3.2)$$

De esta forma, ya tenemos dos ecuaciones de la forma necesaria para utilizar el método RK4.

### 3.3. Solución con RK4

Para encontrar las soluciones al sistema (3.2) se tendrá que resolver cada una de las ecuaciones de forma independiente con valores iniciales distintos para  $x$  y  $y$ , pero cuidando que el valor inicial y el intervalo de  $t$ , así como el tamaño  $h$  de los pasos, sean los mismos para ambas ecuaciones.

Si bien se podría realizar el procedimiento del método RK4 a mano, es más conveniente el desarrollo de un programa que lo realice computacionalmente, ahorrándonos tiempo y esfuerzo. Después de todo, la utilización de los métodos numéricos para la obtención de soluciones

a ecuaciones diferenciales ordinarias (y muchas otras aplicaciones) se ha vuelto especialmente viable en las últimas décadas luego de los avances en la velocidad de procesamiento computacional.

El programa fue realizado en Python, un lenguaje de programación popular para este tipo de aplicaciones. La implementación de RK4 en Python es sumamente sencilla y solo se utilizaron los paquetes *numpy* para un manejo más elegante de los cálculos, y *matplotlib* para la graficación de la solución obtenida. El código se encuentra en la sección 3.4 y también está disponible en el [repositorio de GitHub de este proyecto](#).

El funcionamiento del programa es simple:

1. Se obtienen los datos que el usuario debe definir antes de la aplicación de RK4.
2. En base a esos datos, se definen las estructuras necesarias para la aplicación de RK4.
3. Una vez se tienen las estructuras, se procede a la aplicación de RK4, iterando a lo largo del conjunto discreto de  $t$ .
4. Cuando se termina la aplicación del método, se guarda la solución final y luego se representa en dos gráficas: una donde se trazan individualmente  $x$  y  $y$  contra  $t$ , y otra donde se traza  $x$  contra  $y$ .

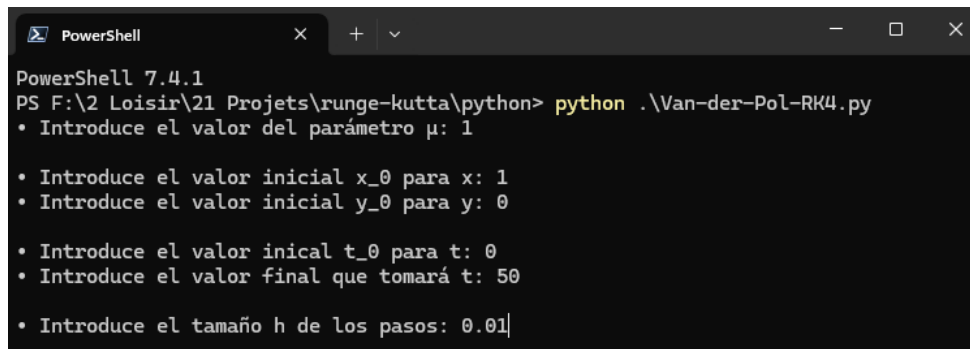
### 3.3.1. Ejemplo 1

Ya que tenemos un programa que realiza los cálculos de forma sencilla, podemos experimentar con los parámetros y demás valores que uno define antes de comenzar la aplicación de RK4.

Para un primer ejemplo, utilizaremos:

$$\begin{aligned}\mu &= 1 \\ x_0 &= 1 & y_0 &= 0 \\ t_0 &= 0 & t_{\text{final}} &= 50 \\ h &= 0.01\end{aligned}$$

Introducimos estos datos al programa:



```
PowerShell 7.4.1
PS F:\2 Loisir\21 Projets\runge-kutta\python> python .\Van-der-Pol-RK4.py
• Introduce el valor del parámetro μ: 1

• Introduce el valor inicial x_0 para x: 1
• Introduce el valor inicial y_0 para y: 0

• Introduce el valor inicial t_0 para t: 0
• Introduce el valor final que tomará t: 50

• Introduce el tamaño h de los pasos: 0.01|
```

Figura 2: Introducción de los datos del ejemplo 1 al programa desde PowerShell

Posteriormente, el programa aproxima la solución a la ecuación de Van der Pol correspondiente



(en su forma de sistema de ecuaciones de primer orden) utilizando el método RK4 con la configuración que hemos definido según estos datos y finalmente nos muestra las gráficas correspondientes a la solución encontrada:

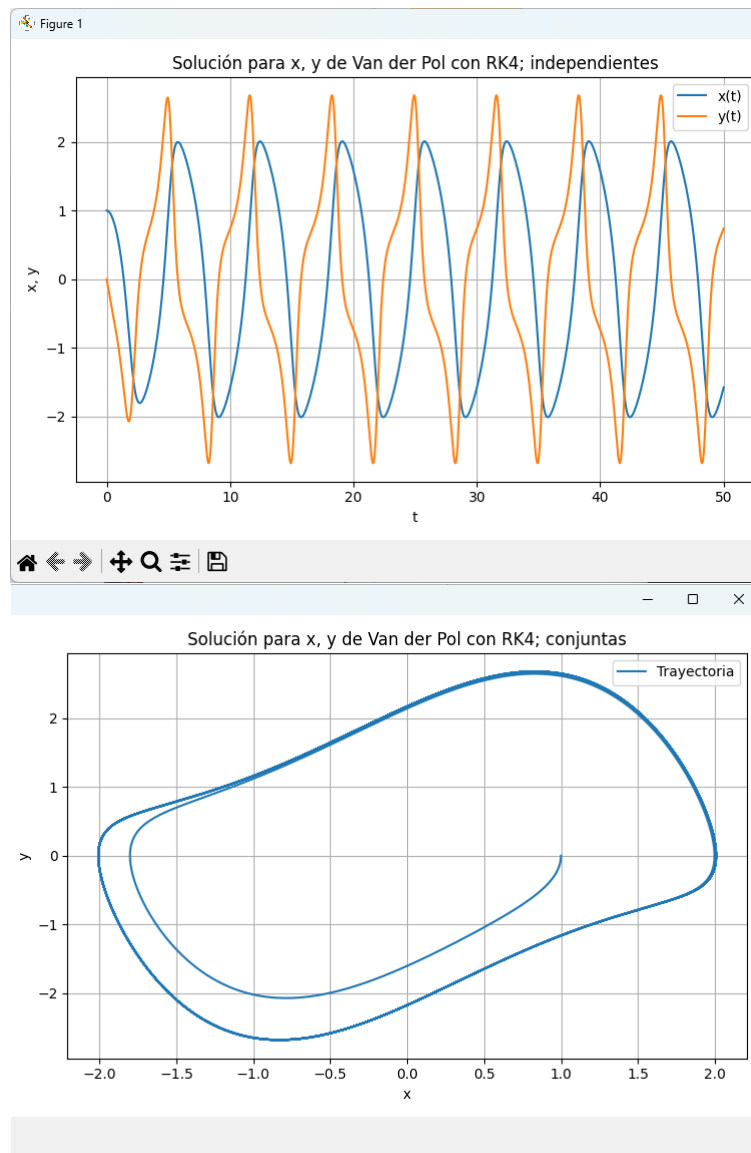


Figura 3: Gráficas de las soluciones halladas correspondientes al ejemplo 1

Si buscamos en diversas fuentes sobre el oscilador de Van der Pol y las soluciones a su ecuación (como las que están en la sección de referencias), comprobaremos que efectivamente toman la forma de las gráficas que hemos obtenido con el programa. El programa funciona correctamente.

Un análisis del comportamiento de las soluciones a la ecuación de Van der Pol en sí está fuera del alcance de este proyecto, así que enfoquémonos solamente en el análisis del procedimiento numérico. Podemos observar que, efectivamente, se aproximó la solución en el intervalo  $t \in [0, 50]$  con un paso de  $h = 0.01$ . El paso es lo suficientemente pequeño como para resultar en una solución que, al menos a primera vista, parece perfectamente suave sin cambios bruscos o «artefactos».

### 3.3.2. Ejemplo 2

Para resaltar la naturaleza numérica de estas soluciones, se reutilizarán todos los datos del ejemplo 1 para obtener otra solución, pero esta vez optando por un tamaño de paso más largo  $h = 0.75$ . Con ello, el programa resulta en:

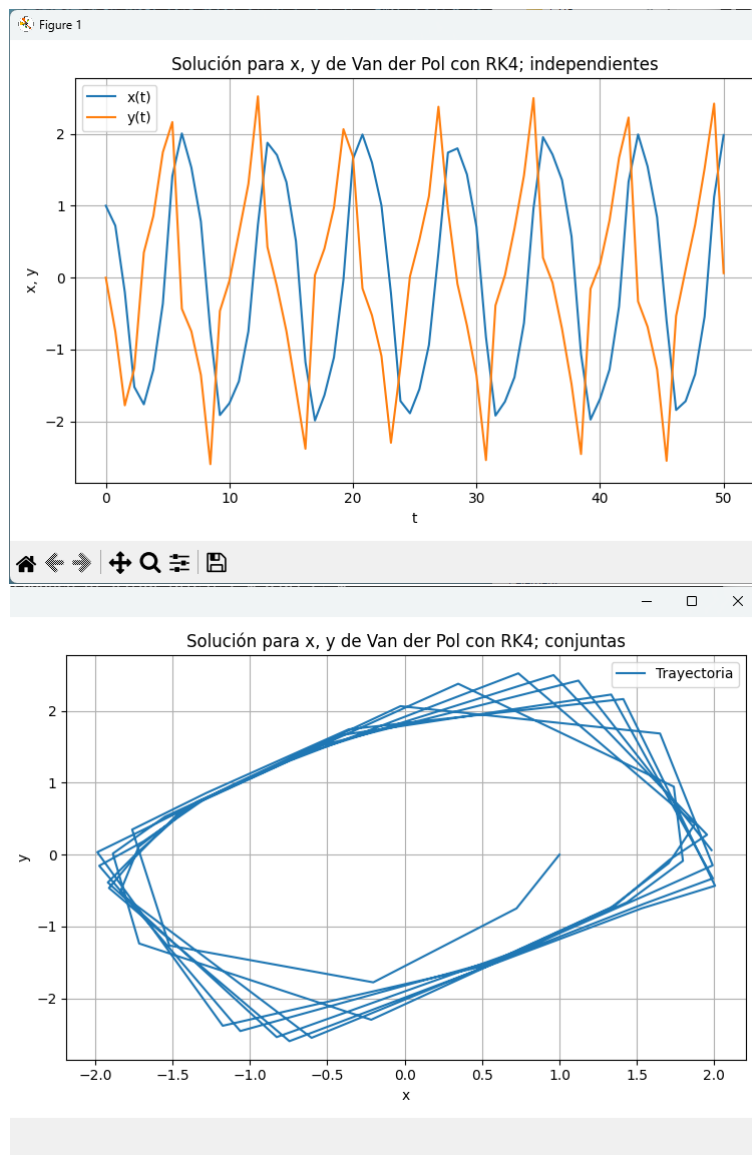


Figura 4: Aplicación de RK4 con los mismos datos del ejemplo 1, pero  $h = 0.75$

Observemos que esta aproximación continúa teniendo la misma tendencia que la del ejemplo 1 (la solución es la misma, después de todo), pero, esta vez, como el paso es más largo, se perciben los segmentos de recta individuales que unen los puntos de aproximación obtenidos numéricamente.

Aquí también podemos incluso notar de forma más clara que la gráfica conjunta de la aproximación obtenida anteriormente (figura 3.3.1) se compone de una línea que pasa varias veces por puntos muy cercanos; en el ejemplo 1 parecería que es una sola línea que solo pasa una vez por una cierta vecindad, pero en este segundo ejemplo vemos que en realidad pasa varias

ocasiones.

### 3.3.3. Ejemplo 3

Como último ejemplo, se mostrará que el programa también funciona para el problema generado por otros datos. Esta vez, se utilizará:

$$\begin{aligned}\mu &= 4 \\ x_0 &= 1 & y_0 &= 2.5 \\ t_0 &= 10 & t_{\text{final}} &= 30 \\ h &= 0.05\end{aligned}$$

La solución aproximada generada por el programa es:

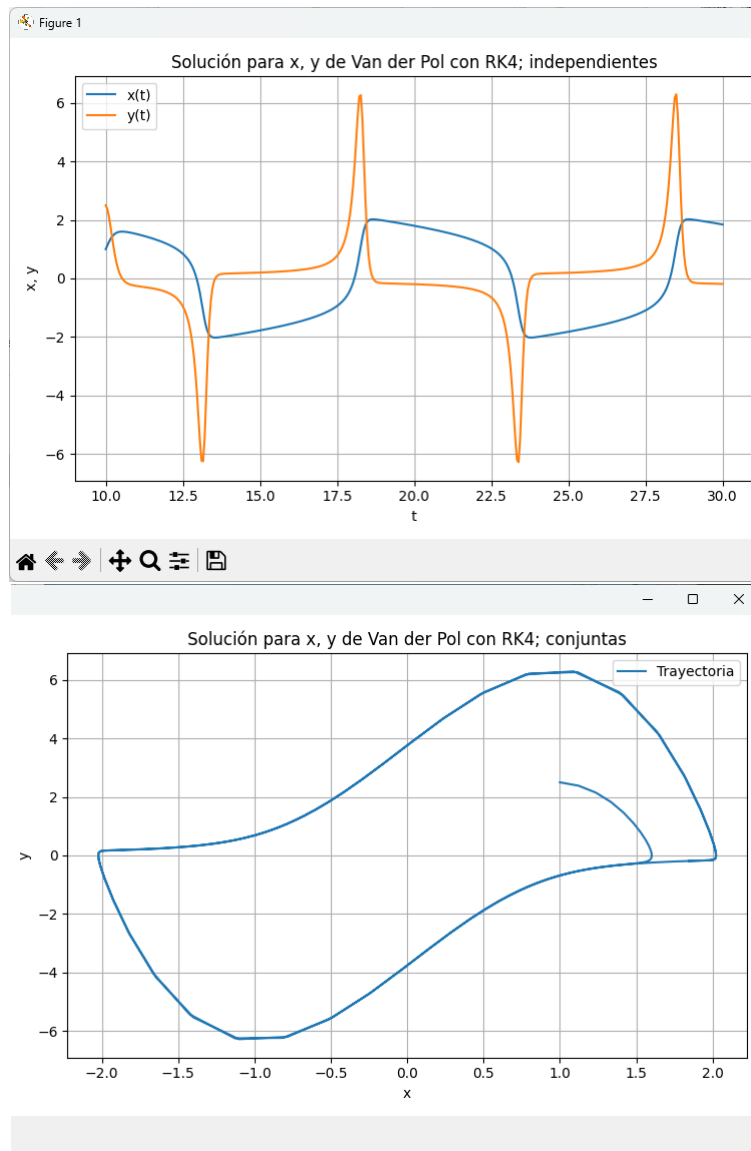


Figura 5: Aplicación de RK4 con los datos correspondientes al ejemplo 3

Notemos que el programa una vez más logra dar una solución aproximada satisfactoria que corresponde al comportamiento de la solución a la ecuación de Van der Pol.

### 3.4. Implementación en Python

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Retorna la evaluacion segun la ecuacion de Van der Pol
5 def van_der_pol(x, y, m):
6     return [y, m*(1 - x**2)*y - x]
7
8 # Aplica RK4 para encontrar las soluciones a Van der Pol
9 def runge_kutta(t0, x0, y0, m, h, t_final):
10     total_steps = int((t_final - t0) / h)
11
12     # Arreglos para la discretizacion de t y los valores de x, y
13     t = np.linspace(t0, t_final, total_steps)
14     x = np.zeros(total_steps)
15     y = np.zeros(total_steps)
16
17     x[0] = x0
18     y[0] = y0
19
20     # Aproximacion de los puntos hasta cubrir todos los valores de
21     # t
22     for i in range(1, total_steps):
23         # Pendientes
24         k1 = h * np.array(van_der_pol(x[i-1], y[i-1], m))
25         k2 = h * np.array(van_der_pol(x[i-1] + 0.5*k1[0], y[i-1] + 0.5*
26         k1[1], m))
27         k3 = h * np.array(van_der_pol(x[i-1] + 0.5*k2[0], y[i-1] + 0.5*
28         k2[1], m))
29         k4 = h * np.array(van_der_pol(x[i-1] + k3[0], y[i-1] + k3[1], m
30         ))
31
32         # Aproximacion del punto siguiente
33         x[i] = x[i-1] + (1/6) * (k1[0] + 2*k2[0] + 2*k3[0] + k4[0])
34         y[i] = y[i-1] + (1/6) * (k1[1] + 2*k2[1] + 2*k3[1] + k4[1])
35
36     return t, x, y
37
38 # Grafica la solucion
39 def plot_solution():
40     fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
41
42     # Grafica 1: x, y contra t
43     ax1.plot(t, x, label="x(t)")
44     ax1.plot(t, y, label="y(t)")
45     ax1.set_xlabel("t")
46     ax1.set_ylabel("x, y")
47     ax1.legend()
48     ax1.grid(True)
49     ax1.set_title("Solucion para x, y de Van der Pol con RK4;
50     independientes")
51
52     # Grafica 2: x contra y
53     ax2.plot(x, y, label="Trayectoria")
54     ax2.set_xlabel("x")
55     ax2.set_ylabel("y")
56     ax2.set_title("Solucion para x, y de Van der Pol con RK4; conjuntas")
```

```

52     ax2.legend()
53     ax2.grid(True)
54
55     plt.tight_layout()
56     plt.show()
57
58 # Definicion de mu, h y valores iniciales
59 m = float(input("- Introduce el valor del parametro mu: "))
60 x0 = float(input("\n- Introduce el valor inicial x_0 para x: "))
61 y0 = float(input("- Introduce el valor inicial y_0 para y: "))
62 t0 = float(input("\n- Introduce el valor inicial t_0 para t: "))
63 t_final = float(input("- Introduce el valor final que tomara t: "))
64 h = float(input("\n- Introduce el tamano h de los pasos: "))
65
66 # Resolucion con RK4 y graficacion
67 t, x, y = runge_kutta(t0, x0, y0, m, h, t_final)
68 plot_solution()

```

## 4. Conclusión