Física Computacional: Resolución problemas 3 y 4, guía 1

Martín Famá

Fecha de entrega: 20/08/2020

1 Resumen

Las ecuaciones 1 y 2 representan un modelo de un sistema de reacciones químicas:

$$\frac{dx}{dt} = f^x(x, y, t) = a - (b+1)x + x^2 y \tag{1}$$

$$\frac{dy}{dt} = f^y(x, y, t) = bx - x^2 y \tag{2}$$

Siendo x e y las concentraciones de dos sustancias distintas, y a y b parámetros. Se resolvió el sistema numéricamente implementando el método Runge-Kutta de orden 4 (RK4). Primero se implemento el método con pasos h fijos, y luego se modificó para que los pasos sean adaptivos.

2 Problema 3 (pasos fijos)

El problema viene dado por dos variables dependientes, x e y, con el tiempo t la variable independiente. Sin embargo, notamos que en las ecuaciones 1 y 2 el tiempo no aparece explícitamente. El método RK4 entonces resulta:

$$k_1^x = f^x(x_j, y_j)$$
$$k_2^x = f^x(x_j + \frac{h}{2}k_1^x, y_j + \frac{h}{2}k_1^y)$$
$$k_3^x = f^x(x_j + \frac{h}{2}k_2^x, y_j + \frac{h}{2}k_2^y)$$
$$k_4^x = f^x(x_j + hk_3^x, y_j + hk_3^y)$$

$$k_1^y = f^y(x_j, y_j)$$
$$k_2^y = f^y(x_j + \frac{h}{2}k_1^x, y_j + \frac{h}{2}k_1^y)$$
$$k_3^y = f^y(x_j + \frac{h}{2}k_2^x, y_j + \frac{h}{2}k_2^y)$$
$$k_4^y = f^y(x_j + hk_3^x, y_j + hk_3^y)$$

Y obtenemos las siguientes ecuaciones para calcular los próximos valores de x e y:

$$x_{j+1} = x_j + \frac{h}{6}(k_1^x + 2k_2^x + 2k_3^x + k_4^x) + o(h^5) \tag{3}$$

$$y_{j+1} = y_j + \frac{h}{6}(k_1^y + 2k_2^y + 2k_3^y + k_4^y) + o(h^5) \tag{4}$$

A continuación se puede ver código de Python que implementa el método. Solo se incluye la función relevante, el resto de las dependencias se pueden encontrar en el archivo .py adjunto a la entrega.

```
#sistema nos resuelve el sistema de ecuaciones numéricamente
#se le pasan como argumento los parámetros a,b
#                            los valores iniciales x_0, y_0
#                            el paso h
#                          y hasta que tiempo t_f resolver
```

```python
def sistema(a, b, x_0, y_0, h, t_f):

    #devuelve las dos ecuaciones diferenciales evaluadas
    def f(x, y):
        return a - (b+1)*x + x**2*y, b*x - x**2*y

    #array que contiene los intervalos de tiempo
    t = np.arange(0, t_f, h)

    #aca guardaremos la trayectoria del sistema
    x = np.array([x_0])
    y = np.array([y_0])

    #loopeamos sobre todos los intervalos de tiempo
    for j in range(0, len(t)):

        #calculamos los k para el metodo RK4
        k_1_x, k_1_y = f(x[j], y[j])
        k_2_x, k_2_y = f(x[j]+h*k_1_x/2, y[j]+h*k_1_y/2)
        k_3_x, k_3_y = f(x[j]+h*k_2_x/2, y[j]+h*k_2_y/2)
        k_4_x, k_4_y = f(x[j]+h*k_3_x, y[j]+h*k_3_y)

        #agregamos los nuevos valores calculados de x e y a los
        #arrays que contienen la trayectoria
        x = np.append(x, x[j] + h*(k_1_x+2*k_2_x+2*k_3_x+k_4_x)/6) # + o(h^5)
        y = np.append(y, y[j] + h*(k_1_y+2*k_2_y+2*k_3_y+k_4_y)/6) # + o(h^5)

    return x, y
```

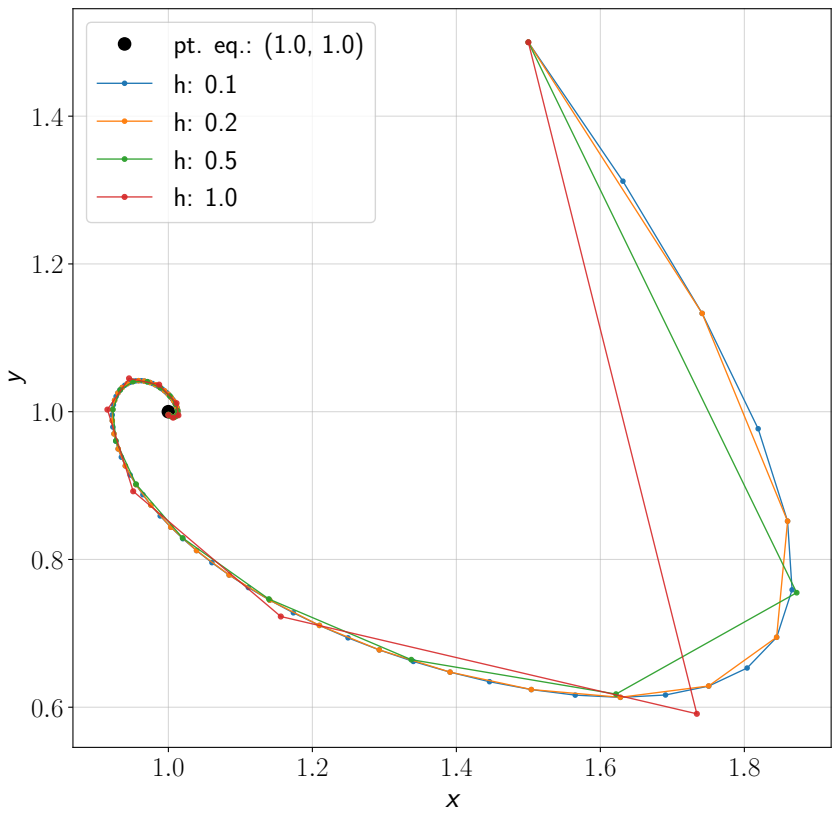
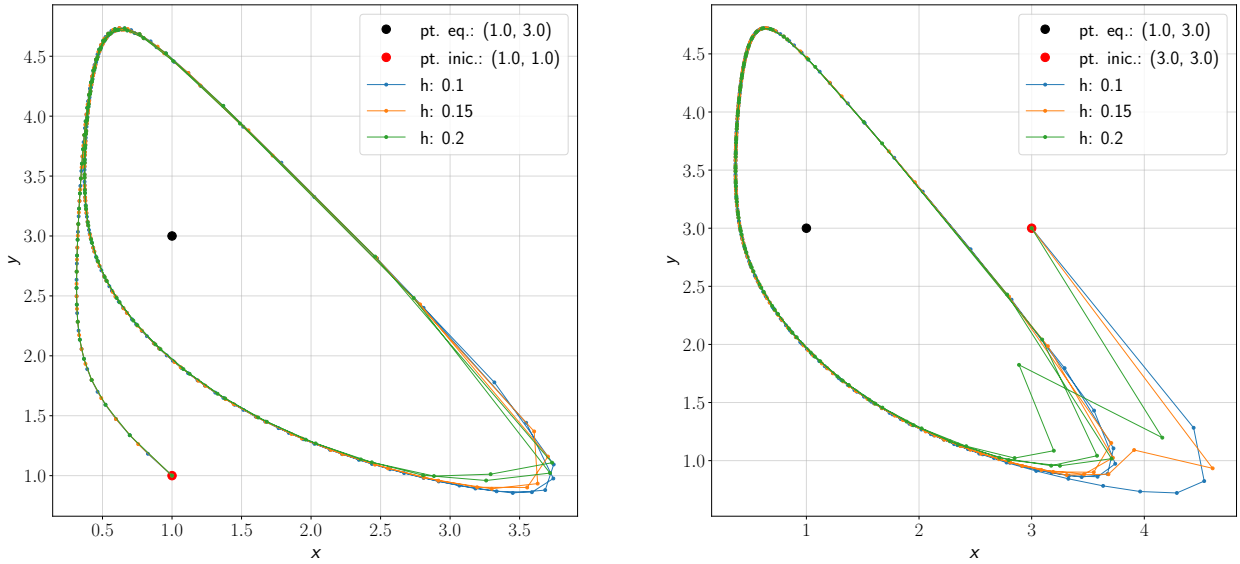


Fig. 1: Simulación para distintos pasos h, con $a = b = 1$

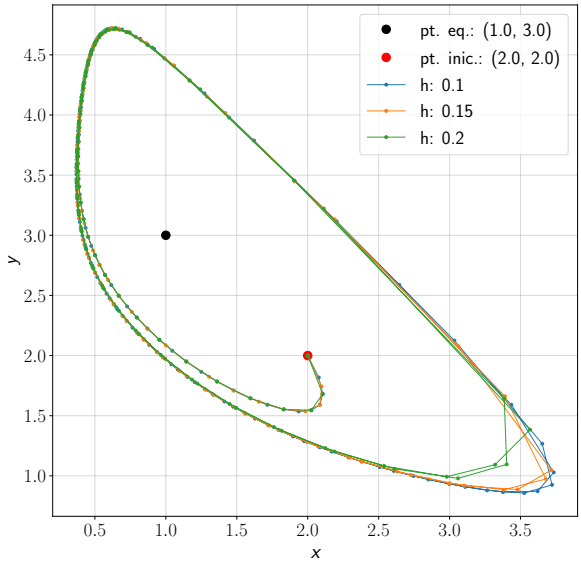
A la izquierda (Fig. 1) se pueden ver los resultados del método con parámetros $a = b = 1$, y para distintos pasos h. El punto de equilibrio en este caso es $(x_{eq} = 1,\ y_{eq} = 1)$. Vemos que para todos los valores de h, la solución converge.

En la siguiente página, se puede ver (Fig. 2) la evolución del sistema con $a = 1$, $b = 3$ para distintos puntos iniciales y distintos pasos h. En este caso, el sistema tiene un equilibrio en $(x_{eq} = 1,\ y_{eq} = 3)$, pero es inestable. El sistema tiende a un ciclo límite. En las figuras 2d y 2e, aún estando la condición inicial muy cercana al punto de equilibrio, el sistema evoluciona al mismo ciclo límite. En la figura 2f, la condición inicial *es* el equilibrio, y ahí el sistema permanece en ese estado pues no hay perturbaciones.

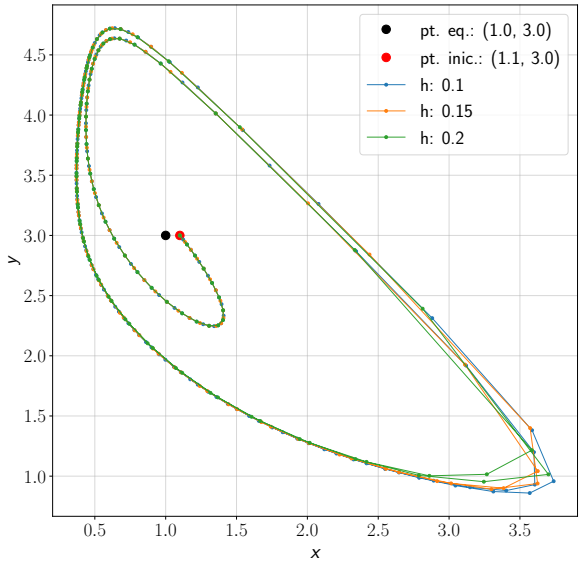


(a)

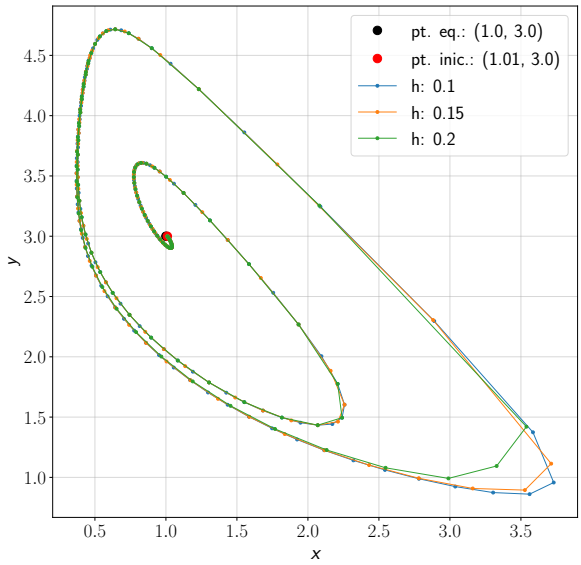
(b)



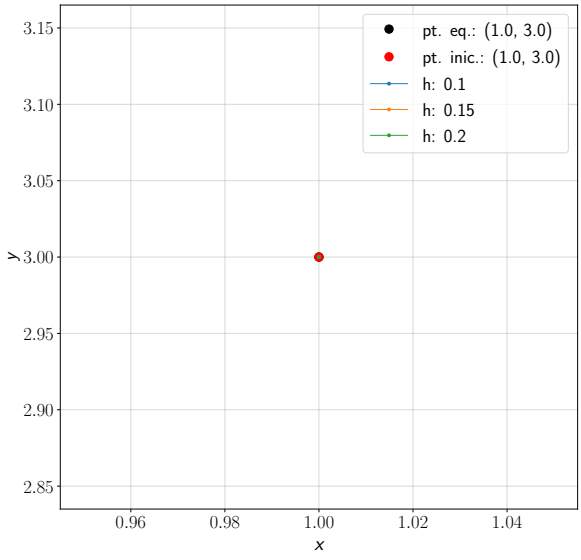
(c)



(d)



(e)



(f)

Fig. 2: El método aplicado al sistema $a = 1$, $b = 3$ con distintos puntos iniciales (marcados con punto rojo). En todos los casos el sistema tiende a un ciclo límite, salvo en (f), que está en equilibrio (inestable).

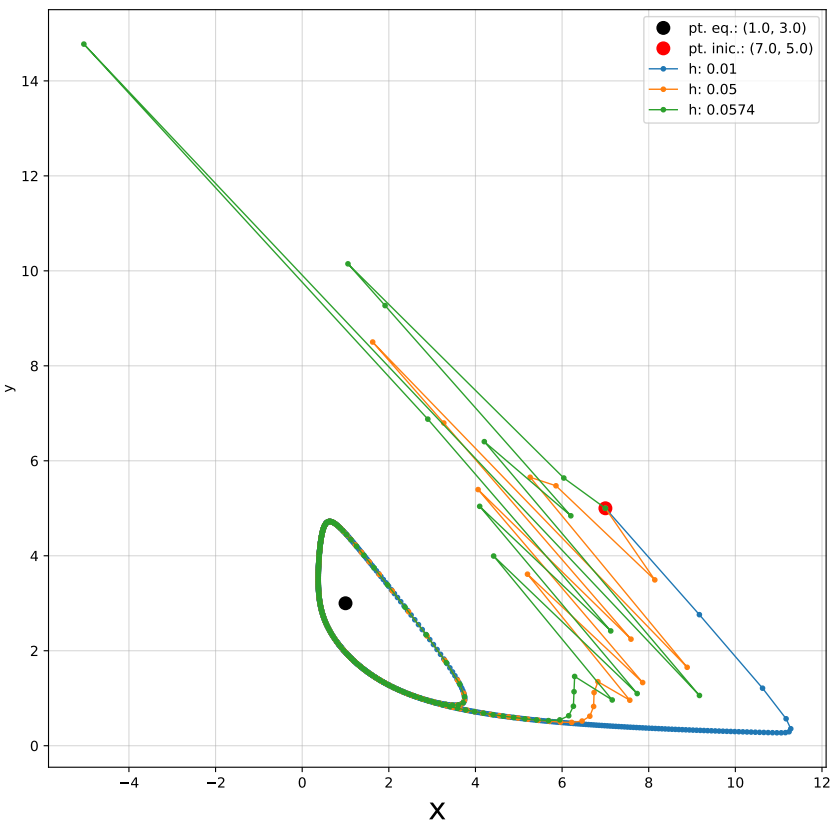


Fig. 3: El modelo presenta comportamiento errático, eventualmente divergiendo si h es lo suficientemente grande ($a = 1$, $b = 3$)

El tamaño del paso h no solo dicta el error cometido en la simulación, sino que también puede terminar dando resultados divergentes, como es el caso en el sistema $a = 1$, $b = 3$. A la izquierda (Fig. 3), se puede ver comportamiento errático del modelo, siendo cada vez más errático al aumentar h. En este caso, con ($x_0 = 7$, $y_0 = 5$), se encontró que $h = 0.0574$ está justo por debajo del límite en donde el modelo diverge explosivamente.

3 Problema 4 (pasos adaptivos)

Para implementar pasos adaptativos, se consideró el siguiente criterio: en cada paso, se calcularon las soluciones de x e y en un caso con h y en otro con $h/2$. Es decir se calculó x_1, y_1 tomando como paso h, y x_2, y_2 con $h/2$ (se toman dos pasos en esta instancia, para que el tiempo total simulado sea $2h/2 = h$). Se tomó la diferencia:

$$\Delta_x = x_2 - x_1 \qquad\qquad \Delta_y = y_2 - y_1 \tag{5}$$

Luego, tomando una tolerancia ϵ (por ejemplo $\epsilon = 10^{-5}$), se aplicó la siguiente evaluación:

a) Si $\Delta_x < \epsilon/2$ y $\Delta_y < \epsilon/2$: se aceptan x_2, y_2 y se multiplica el paso h por 1.5.

b) Si $\epsilon/2 < \Delta_x < \epsilon$ y $\epsilon/2 < \Delta_y < \epsilon$: se aceptan x_2, y_2 y se deja el paso h igual.

c) De lo contrario: rechazamos x_2, y_2, dividimos h por 1.5 y repetimos el procedimiento.

A continuación esta el código de Python implementado (nuevamente, se adjunta un archivo completo .py con la entrega).

```
#sistemaAdapt ahora tiene pasos adaptativos
#se le pasan los mismos argumentos, mas la
#tolerancia (err)
def sistemaAdapt(a, b, x_0, y_0, h, t_f, err):
    #devuelve las dos ecuaciones diferenciales evaluadas
    def f(x, y):
        return a - (b+1)*x + x**2*y, b*x - x**2*y

    def RK4(h_, x, y):
        k_1_x, k_1_y = f(x             , y)
        k_2_x, k_2_y = f(x + h_*k_1_x/2, y + h_*k_1_y/2)
        k_3_x, k_3_y = f(x + h_*k_2_x/2, y + h_*k_2_y/2)
        k_4_x, k_4_y = f(x + h_*k_3_x  , y + h_*k_3_y)

        #calculamos los (posibles) nuevos valores de x, y
        #digo posibles porque mas adelante vamos a chequear
        #si cumplen con la cota del error para ser aceptados
        x_ = x + h_*(k_1_x+2*k_2_x+2*k_3_x+k_4_x)/6 # + o(h^5)
        y_ = y + h_*(k_1_y+2*k_2_y+2*k_3_y+k_4_y)/6 # + o(h^5)

        return x_, y_

    #array que contiene los intervalos de tiempo
    t = np.array([0.0])

    #aca guardaremos la trayectoria del sistema
    x = np.array([x_0])
    y = np.array([y_0])

    #en este while loopeamos hasta que el tiempo llegue a t_f
    #puede tener una cantidad arbitraria de pasos, ya que el
    #largo del paso "h" se va modificando
    j = 0 #indice
    while t[-1] < t_f:
        while True:
            #proximos valores con paso h
            x_1, y_1 = RK4(h, x[j], y[j])
            #proximos valores con paso h/2
```

```
            #hay que hacer dos pasos aca
            x_2, y_2 = RK4(h/2, x[j], y[j])
            x_2, y_2 = RK4(h/2, x_2, y_2)
            dx = np.abs(x_2 - x_1)
            dy = np.abs(y_2 - y_1)
            delta = max(dx, dy)

            #aca chequeamos si se cumplen las cotas de errores
            if delta <= err/2:
                t = np.append(t, t[-1]+h/2)
                h *= 1.5
                break #salimos del while
            elif delta > err/2 and delta <= err:
                t = np.append(t, t[-1]+h/2)
                break #salimos del while

            #si ninguna condicion se cumplio, rechazamos x_2, y_2 y
            #reducimos el paso
            h /= 1.5

        #si llegamos hasta aca, nos aseguramos que queremos aceptar x_2, y_2
        x = np.append(x, x_2) # + o(h^5)
        y = np.append(y, y_2) # + o(h^5)
        j += 1

    return x, y, t
```

En la próxima página (Fig. 4) se puede ver la evolución temporal de x e y usando este método, de nuevo para el sistema $a = 1$, $b = 3$, y con una tolerancia de $\epsilon = 10^{-5}$. Sobre el eje temporal, las líneas rojas marcan los intervalos de tiempo, que se puede ver que varían. En particular, notamos que los pasos se achican en regiones donde x e/o y varían rápidamente, mientras que los pasos son más largos en regiones donde ocurre lo contrario.

En este caso, se simuló hasta un tiempo $t_f = 10$, y el método tomó 169 pasos. Esto es el 16.9% de los 1000 pasos que el método con pasos fijos $h = 0.01$ toma. El método tiene el potencial para reducir el número de pasos pues en regiones donde no se necesita un paso demasiado corto para mantener precisión, los pasos más largos ahorran ciclos del algoritmo. Es importante tener en cuenta igual que podría pasar lo contrario: el método se topa con una región en donde achica el paso h de manera drástica para mantener la precisión, y podríamos tener una cantidad total de pasos enorme. Resumiendo, el método tiene la ventaja de mantener cierto nivel de precisión, pero a priori no sabemos que tan caro (en cuanto a tiempo de ejecución) será.

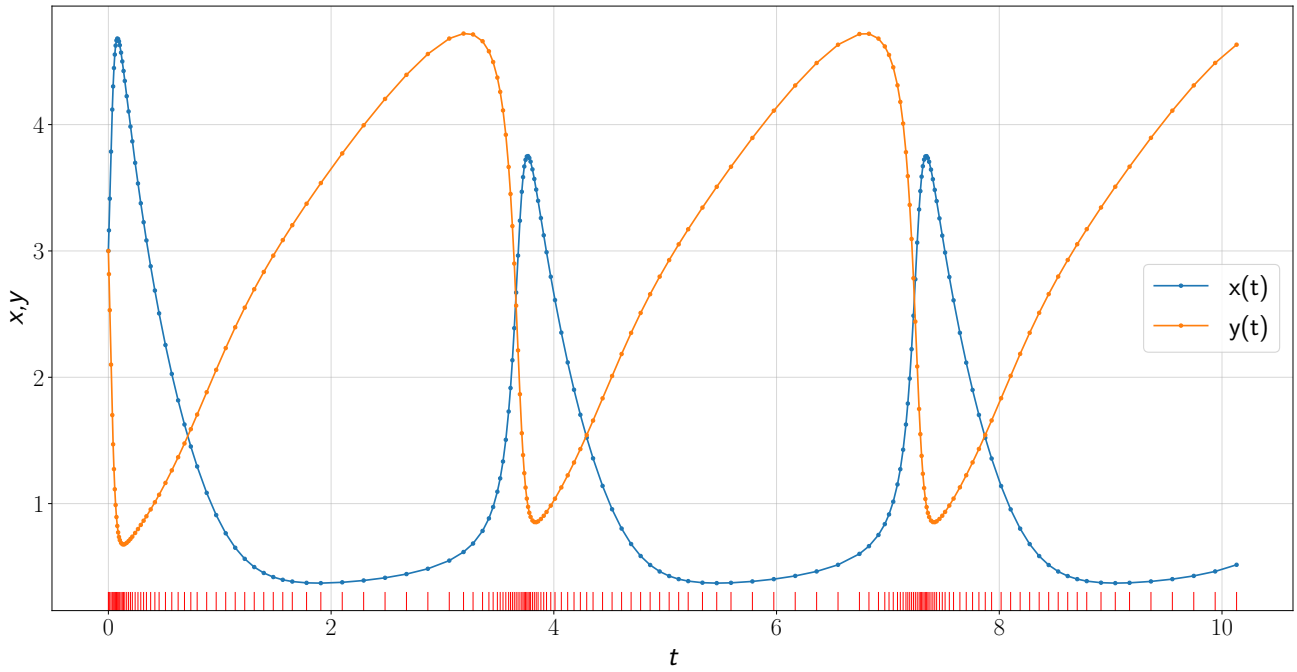


Fig. 4: Evolución temporal de x e y en el sistema $a = 1$, $b = 3$ con pasos adaptativos. Las líneas rojas representan los pasos de tiempo, que varían según el criterio planteado.

Abajo, se pueden ver (Fig. 5) las trayectorias en el espacio de fases del mismo sistema (con $t_f = 10$), tomando tolerancias distintas (entre 10^{-5} y 10^{-1}). Vemos que según la tolerancia, el sistema tiende a un ciclo levemente distinto, debido al error introducido. Al ir disminuyendo la tolerancia, la trayectoria se asemeja cada vez más a la trayectoria verdadera (analítica). En la leyenda, también se pueden ver la cantidad de pasos que tomo el método con la respectiva tolerancia.

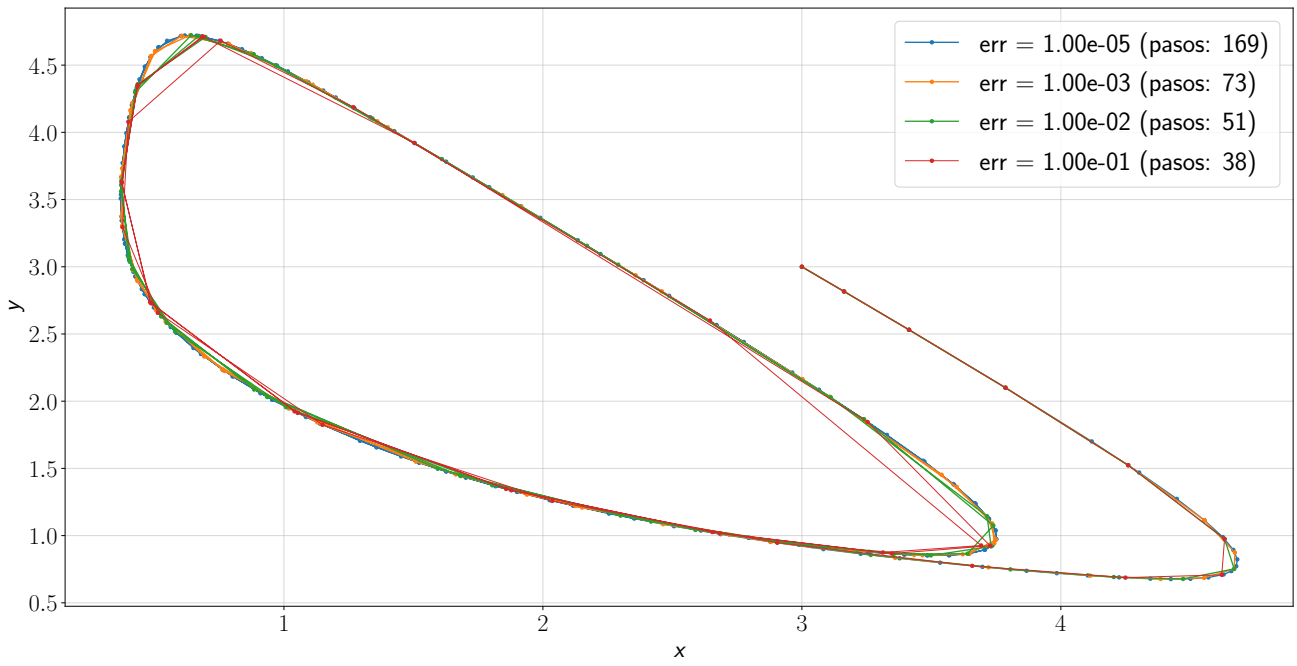


Fig. 5: Trayectorias en el espacio de fases del sistema $a = 1$, $b = 3$ con pasos adaptativos, con distintas tolerancias.