Algoritmos Numéricos por Computadora

Profesor Ángel Fernando Kuri Morales

Otoño 2020

Emilia Sofia Spinola Campos

Celina Villers De Coss

Método de Runge-Kutta

Las ecuaciones diferenciales se utilizan para relacionar funciones que representan cantidades físicas y sus derivadas que representan razones de cambio. Existen diferentes tipos de ecuaciones diferenciales, pero en este caso se trabajará con las ecuaciones diferenciales ordinarias (EDO), las cuales contienen una función de una sola variable independiente y sus derivadas, es decir, la derivada es respecto a un solo término y son definidas de la forma siguiente:

$$F(x, y, y', y'', \ldots, y^{(n)}) = 0$$

En caso de ser lineales se hace uso de fórmulas para obtener soluciones exactas, de otra forma, se utilizan métodos de aproximación en forma de series o integrales. Las soluciones son catalogadas en generales o particulares; la solución general se da cuando las constantes en el resultado se dejan indicadas (como C_i o K_i), y la solución particular se obtiene gracias a la definición de condiciones iniciales, y por tanto se obtiene una única función que satisfaga la ecuación diferencial. Para una ecuación diferencial lineal de orden n, se requieren n condiciones iniciales para que exista una

única función que cumpla simultáneamente la ecuación diferencial y las condiciones. Por ejemplo:

Si se tiene la ecuación diferencial $y' = -\frac{y}{t} + 2$ con condición inicial $y(1) = 3$

Solución General	Solución Particular
$y = t + \frac{C}{t}$	$y = t + \frac{2}{t}$

Los métodos de Runge-Kutta son una serie de métodos iterativos, explícitos e implícitos, para aproximación numérica de ecuaciones diferenciales ordinarias, con el problema de valor inicial, desarrollados por los alemanes Carl Runge y Martin Wilhelm Kutta. El problema originalmente planteado por ambos fue para la solución de una ecuación diferencial ordinaria de cuarto orden, por lo que a este se le conoce como el método de Runge-Kutta .

Si se define la ecuación diferencial ordinaria como $y'(t) = f(t, y(t))$ con la condición de valor inicial de f como $(t_0\,, y_0) \in \Omega \subset \mathbb{R}^n$. Por lo tanto, el método Runge-Kutta de orden s se define como:

$$y_{n+1} = y_n + h \sum_{i=1}^{s} a_{ij} k_j$$

donde

$h =$ paso por iteración o incremento Δt_n entre los puntos t_n y t_{n+1}

$k_i =$ términos de aproximación evaluados en f de manera local y definidos por

$$k_i = f\left(t_n + hc_i\,, y_n + h \sum_{i=1}^{s} a_{ij} k_j\right), \qquad i = 1, \ldots, s$$

con a_{ij}, b_i, c_i coeficientes dependientes de la regla de cuadratura de integración numérica.

a_{ij}, se conoce como la matriz Runge-Kutta, b_i son los pesos y c_i los nodos, y son usados para la definición del tablero de Butcher, que debe cumplir la condición $\sum_{i=1}^{s} b_i = 1$. Los coeficientes se determinan de acuerdo a la ecuación $\sum_{j=1}^{i-1} a_{ij} = c_i$ para $i = 2, \ldots, s$. Y se escribe de la forma siguiente:

$$\begin{array}{c|cccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & & \ddots & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} \\ \hline & b_1 & b_2 & \cdots & b_{s-1} \quad b_s \end{array}$$

Donde, para el método de Runge-Kutta de cuarto orden, se define de manera directa como:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

Método de Runge-Kutta de cuarto orden (RK4)

Se define el problema inicial como $y' = f(t, y), y(t_0) = y_0$, por lo que el método RK4 es dado por

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

Donde

$k_1 = f(t_i, y_i)$

$$k_2 = f(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h)$$

$$k_3 = f(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h)$$

$$k_4 = f(t_i + h, y_i + k_3 h)$$

Así, el siguiente valor (y_{n+1}) es determinado por el presente valor (y_n) más el producto del tamaño del intervalo (h) por una pendiente estimada. La pendiente es un promedio ponderado de pendientes, donde k_1 es la pendiente al principio del intervalo, k_2 es la pendiente en el punto medio del intervalo, usando k_1 para determinar el valor de y en el punto $x_n + \frac{h}{2}$ usando el método de Euler **[Anexo 2]**. k_3 es otra vez la pendiente del punto medio, pero ahora usando k_2 para determinar el valor de y; k_4 es la pendiente al final del intervalo, con el valor de y determinado por k_3. Promediando las cuatro pendientes, se le asigna mayor peso a las pendientes en el punto medio:

$$\text{Pendiente} = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

Debido a que se trata de un método de cuarto orden, el error de convergencia es de $O(h^4)$.

Otra explicación más completa se hace mediante la **definición de integrales**, donde se quiere integrar la EDO $y' = f(t, y)$ desde t_n a t_{n+1} para obtener su solución:

$$y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} f\big(t, y(t)\big)dt$$

Se toma $y(t_n)$ cómo conocido para obtener la solución para $y(t_{n+1})$, a partir de la condición inicial. La integral se aproxima mediante la Regla de Simpson **[Anexo 3]**, de la forma:

$$\int_{t_n}^{t_{n+1}} f(t, y(t))dt \approx \frac{\Delta t}{6}\left(f^n + 4f^{n+\frac{1}{2}} + f^{n+1}\right)$$

pero surge el problema de no conocer $f^{n+\frac{1}{2}} = f\left(t_{n+\frac{1}{2}}, y^{n+\frac{1}{2}}\right)$ y $f^{n+1} = f(t_{n+1}, y^{n+1})$ por lo que se divide la integral en cuatro términos para la aproximación:

$$\int_{t_n}^{t_{n+1}} f(t, y(t))dt \approx \frac{\Delta t}{6}\left(f^n + 2\hat{f}^{n+\frac{1}{2}} + 2\bar{f}^{n+\frac{1}{2}} + \bar{f}^{n+1}\right)$$

Donde $2\hat{f}^{n+\frac{1}{2}}, 2\bar{f}^{n+\frac{1}{2}}, \bar{f}^{n+1}$ son aproximaciones de los términos faltantes.

Para obtener $\hat{f}^{n+\frac{1}{2}}$ se utiliza el método de Euler con paso de $\frac{1}{2}\Delta t$:

$$\hat{f}^{n+\frac{1}{2}} = f\left(t_{n+\frac{1}{2}}, y^n + \frac{1}{2}\Delta t f^n\right)$$

Para $\bar{f}^{n+\frac{1}{2}}$ se utiliza un método de Euler hacia atrás:

$$\bar{f}^{n+\frac{1}{2}} = f\left(t_{n+\frac{1}{2}}, y^n + \frac{1}{2}\Delta t \hat{f}^{n+\frac{1}{2}}\right)$$

Para $\bar{f}^{n+1}$ se utiliza un método de Crank-Nicolson **[Anexo 4]**:

$$\bar{f}^{n+1} = f\left(t_{n+1}, y^n + \Delta t \hat{f}^{n+\frac{1}{2}}\right)$$

La combinación de estos métodos genera un intervalo de tiempo de t_n a t_{n+1} y por lo tanto el método de Runge-Kutta se deriva en:

$$y_{i+1} = y_i + \frac{\Delta t}{6}\left(f^n + 2\hat{f}^{n+\frac{1}{2}} + 2\bar{f}^{n+\frac{1}{2}} + \bar{f}^{n+1}\right)$$

cuyos elementos fueron definidos anteriormente.

Ejemplos de Ecuaciones Diferenciales Ordinarias

Para la comprensión del método y el código implementado, se prepararon varios ejemplos para ser desarrollados y comparados con el uso de una calculadora online https://www.mathstools.com/section/main/runge_kutta_calculator#.X5YwxlhKjIW.

1. $y' = t\sqrt{y(t)}$ con $y(0) = 1.$ $tf = 10,\ n = 100$
2. $y' = t + \sqrt{y(t)}$ con $y(0) = 1.$ $tf = 10,\ n = 100$
3. $y' = t - \sqrt{y(t)}$ con $y(0) = 1.$ $tf = 10,\ n = 100$
4. $y' = y(t) + t^2 + \sin(t)$ con $y(0) = 1.$ $tf = 10,\ n = 100$
5. $y' = y(t) + t^2 - \sin\big(y(t)\big)$ con $y(0) = 1.$ $tf = 10,\ n = 100$

Problema Depredador - Presa (EDO)

Las ecuaciones de Lotka-Volterra, generan un modelo de ecuaciones diferenciales que modelan el comportamiento de sistemas biológicos en los que interactúan dos especies, una presa y un depredador.

Si se tienen los supuestos:

- Si los depredadores no estuvieran presentes, las presas crecerían proporcionalmente a su población, sin control o restricción alguna.
- El depredador caza a sus presas de manera proporcional al número de encuentros entre ambas especies.
- Si no hubiera presas, la población de depredadores decrece de manera proporcional a su población.
- La tasa de nacimiento de depredadores es proporcional al número de presas cazadas.

En este caso, se generan dos variables dependientes del tiempo:

$D(t)$ = población de depredadores, $P(t)$ = población de presas

Además de cuatro parámetros constantes:

$\alpha > 0$ tasa de crecimiento de la presa

$\beta > 0$ constante para medir el número de encuentros entre las especies

$\gamma > 0$ tasa de mortandad de depredadores

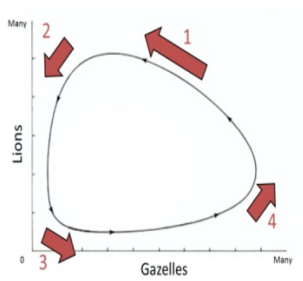
$\delta > 0$ constante para medir el beneficio de depredadores al cazar presas

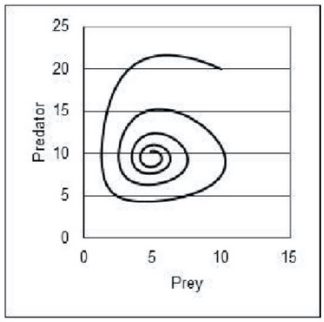
A partir de esta información, se deriva el siguiente sistema de ecuaciones de primer orden:

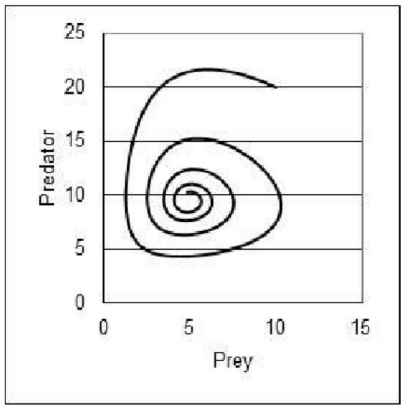
$$\frac{dP}{dt} = \alpha P - \beta PD, \qquad \frac{dD}{dt} = -\gamma D + \delta PD$$

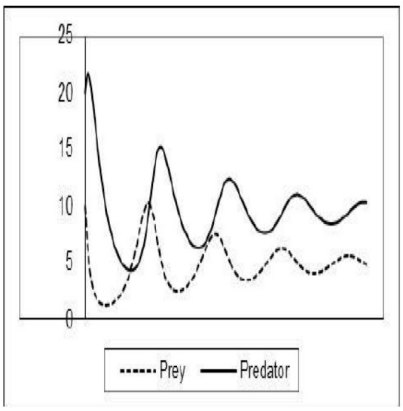
La solución a este sistema sería una pareja de funciones $P(t)$, $D(t)$ que cumplan este par de ecuaciones.

A continuación se muestran ejemplos gráficos para demostrar el comportamiento de este modelo.







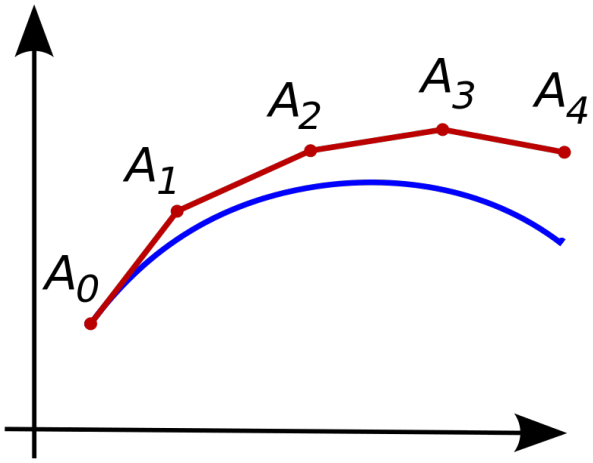


Anexos

[1] La cuadratura numérica es un sinónimo de integración numérica. El problema que plantea es el cálculo de la solución aproximada de la integral definida: $\int_a^b f(x)dx$ que puede ser definido como un problema de valor inicial para una ecuación diferencial ordinaria de la forma $y'(x) = f(x)$, $y(a) = 0$, donde encontrar $y(b)$ equivale al cálculo de la integral y puede ser realizado mediante el método de Runge-Kutta.

[2] El método de Euler es un procedimiento de integración numérica para resolver ecuaciones diferenciales ordinarias (EDO) definidas como problema de valor inicial. El método de Euler es un método de primer orden, por lo que el error local es proporcional al cuadrado del tamaño del paso, y el error global es proporcional al tamaño del paso.

Para la solución se puede pensar en la ecuación diferencial como una fórmula que nos permite calcular la pendiente de la recta tangente a la curva en cualquier punto de la curva, una vez que el punto ha sido calculado. Aunque la curva es desconocida en un principio, se conoce el punto inicial A_0, por lo que de la ecuación diferencial se puede calcular la pendiente de la curva en el punto A_0y como resultado la recta tangente a la curva. Si se da un pequeño paso sobre dicha recta, se toma un nuevo punto A_1 y se supone que pertenece a la curva. Siguiendo el mismo razonamiento, se vuelve a calcular la pendiente de la recta tangente a la curva en el punto A_1. Luego de varios pasos se forma una curva poligonal $A_0A_1A_2$.... El error entre ambas curvas se puede minimizar si se dan pasos muy pequeños al avanzar sobre la recta tangente a la curva, (h pequeña).



Aproximación por el método de Euler

[3] La regla de Simpson es un método de integración numérica que se utiliza para obtener la aproximación de la integral:

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

Para la aproximación de una integral definida en un intervalo $[a, b]$ se sigue que sobre cada subintervalo en el que se divide $[a, b]$, se aproxima f por un polinomio de primer grado, para luego calcular la integral como suma de las áreas de los trapecios formados en esos subintervalos. El método utilizado para la regla de Simpson realiza lo mismo pero aproximando los subintervalos de f mediante polinomios de segundo grado.

[4] Crank Nicolson es un método de diferencias finitas usado para la resolución numérica de ecuaciones en derivadas parciales. Si la ecuación en derivadas parciales es $\frac{\delta u}{\delta t} = \left(u, x, t, \frac{\delta u}{\delta x}, \frac{\delta^2 u}{\delta x^2}\right)$, entonces, tomando $u(i\Delta x, n\Delta t) = u_i^n$, combinamos el método de Euler implicito y explicito en la etapa de tiempo $n + 1$:

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} = F_i^n\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}\right) \qquad \text{(Euler explícito)}$$

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} = F_i^{n+1}\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}\right) \qquad \text{(Euler implícito)}$$

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} = \frac{1}{2}\left[F_i^{n+1}\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}\right) + F_i^n\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}\right)\right] \qquad \text{(Crank-Nicolson)}$$

Se trata de un método implícito, en el cual para obtener el valor de u en el "siguiente instante de tiempo", debe resolverse un sistema de ecuaciones algebraicas.

Links para complementar:

https://mathworld.wolfram.com/Runge-KuttaMethod.html

https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods#:~:text=In%20numerical%20analysis%2C%20the%20Runge,solutions%20of%20ordinary%20differential%20equations.

http://campus.usal.es/~mpg/Personales/PersonalMAGL/Docencia/MetNumTema4Teo(09-10).pdf

https://www.mathstools.com/section/main/runge_kutta_calculator#.X5YgX4hKjIV

http://hplgit.github.io/INF5620/doc/pub/._main_decay007.html

Códigos

Runge-Kutta Lineal

```
import math
import sys
import matplotlib.pyplot as plt
import numpy

def check_int(x):
    while True:
        try:
            x=int(input("\""+x+"\")\t"))
            return x
        except:
            print("Debe ser un número")
            print("Intente de nuevo")
    ##EndWhile
##EndFunction

def RungeKutta4(f, t0, y0, tf, n):
    tiempo = [0] * (n + 1)
    Ysol = [0] * (n + 1)
    h = (tf - t0) / float(n)
    tiempo[0] = t = t0
    Ysol[0] = y = y0
    for i in range(1, n + 1):
        k1 = h * f(t, y)
        k2 = h * f(t + 0.5 * h, y + 0.5 * k1)
```

```
        k3 = h * f(t + 0.5 * h, y + 0.5 * k2)
        k4 = h * f(t + h, y + k3)
        tiempo[i] = t = t0 + i * h
        Ysol[i] = y = y + (k1 + k2 + k2 + k3 + k3 + k4) / 6
    return tiempo, Ysol


# RungeKutta4(f, 0, 1, 10, 100)


while True:
    # Definir la Ecuación Diferencial Ordinaria a ser resuelta
    def function(t, y):
        return y + t**2 - math.sin(y)

    print("Indique la condición inicial y(t0) = y0 \t")
    t0 = check_int("t0")
    y0 = check_int("y0")
    print("Indique el tiempo final (tf) de evaluación \t")
    tf = check_int("tf")
    print("Indique el número de pruebas (n) \t")
    n = check_int("n")

    RKt, RKy = RungeKutta4(function, t0, y0, tf, n)

    plt.plot(RKt, RKy)
    plt.title("Solución Aproximada de la Ecuación Diferencial Ordinaria")
    plt.grid(True)
    plt.xlabel("t")
    plt.ylabel("y(t)")
```

```
    plt.show()

    for t, y in list(zip(RKt, RKy))[::tf]:
        print("%4.1f %10.5f" % (t, y))



    while True:
        Resp=input("Desea continuar iterando? (\"S\"/\"N\")\t").upper()
        if Resp=="N":
            print("*** FIN DE PROGRAMA ***\n\n")
            sys.exit()              ## Exit program
        if Resp=="S":
            break                   # Exit option loop
    #endWhile
```

Runge-Kutta Depredador/Presa

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

def check_int(x):
    while True:
        try:
            x=int(input("\""+x+"\")\t"))
            return x
        except:
            print("Debe ser un número")
            print("Intente de nuevo")
```

```
    ##EndWhile
##EndFunction

def check_float(x):
    while True:
        try:
            x=float(input("\""+x+"\")\t"))
            return x
        except:
            print("Debe ser un número")
            print("Intente de nuevo")
    ##EndWhile
##EndFunction

def RungeKutta2D(y, t, h):                  #edited; no need for input f
        """ Runge-Kutta 4 method """
        k1 = h*f(t, y)
        k2 = h*f(t+0.5*h, y+0.5*k1)
        k3 = h*f(t+0.5*h, y+0.5*k2)
        k4 = h*f(t+h, y+k3)
        return (k1 + 2*k2 + 2*k3 + k4)/6

def f(t, y):
        alpha = 1.1
        beta = 0.4
        gamma = 0.4
        delta = 0.1
        x1, x2 = y[0], y[1]
        fxd = x1*(alpha - beta*x2)
```

```
    fyd = -x2*(gamma - delta*x1)
    return numpy.array([fxd, fyd], float)



while True:
    print("Indique el tiempo inicial (t0) \t")
    t0 = check_int("t0")
    print("Indique el tiempo final (tf) \t")
    tf = check_int("tf")
    print("Indique la condiciones iniciales y(t0) = [y0a y0b] \t")
    a = check_int("y0a")
    b = check_int("y0b")
    y0 = numpy.array([a, b], float)
    print("Indique el número de step (h) \t")
    h = check_float("h")

    time = numpy.arange(t0, tf, h)
    ec1, ec2  = [], []

    for t in time:
        ec1.append(y0[0])         #edited
        ec2.append(y0[1])         #edited
        y0 += RungeKutta2D(y0, t, h)            #edited; no need for input f

    plt.subplot(1, 2, 1)
    plt.plot(time, ec1, "r", label="Presas")
    plt.plot(time, ec2, "b", label="Depredadores")
    plt.grid(True)
    plt.xlabel("t")
```

```
plt.ylabel("y(t)")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(ec1, ec2)
plt.grid(True)
plt.xlabel("Presas")
plt.ylabel("Depredadores")

plt.show()

while True:
    Resp=input("Desea continuar iterando? (\"S\"/\"N\")\t").upper()
    if Resp=="N":
        print("*** FIN DE PROGRAMA ***\n\n")
        sys.exit()              ## Exit program
    if Resp=="S":
        break                   # Exit option loop
#endWhile
```