TAREA 13 - METODOS NUMERICOS

Guillermo Segura Gómez
Centro de Investigación en Matemáticas
Métodos Numéricos
19 de noviembre de 2023

1 Introducción

Las ecuaciones diferenciales aparecen en toda la física y matemáticas, desde el análisis hasta uno de los problemas mas importantes en la física, el péndulo simple. Este tipo de ecuaciones modelan cómo cambian las variables en función del tiempo o de otras variables y son vitales para describir y entender fenómenos físicos, biológicos y de ingeniería. Resolver ecuaciones diferenciales, especialmente las no lineales o aquellas que involucran sistemas complejos, puede ser una tarea muy complicada. Sobretodo por la limitada familia de ecuaciones diferenciales para las cuales existe una solución analítica.

En este contexto, los métodos numéricos para resolver ecuaciones diferenciales ofrecen una alternativa sumamente potente. En el presente trabajo exploramos las técnicas de Euler, Heun, Taylor y Runge-Kutta, para encontrar la solución de sistemas de ecuaciones que no pueden ser resueltos de manera analítica. Para poder encontrar este tipo de soluciones es requisito contar un problema bien planteado (Revisar capitulo burden) [1] lo cual limita la capacidad de soluciones para algunos sistemas pero fortalece el trasfondo matemático, asegurando su funcionalidad y correcta aproximación a las soluciones de las ecuaciones diferenciales.

2 Problema 1

Programar en C, el método de Euler, el método de Heun, el método de Taylor de $2o$ orden, y el método de Runge-Kutta de $4o$ orden (RK4), vistos en clase. Prueba tus programas con el problema de valor inicial y(0) = y, con $y(0) = 1$, en el intervalo [0, 4], cuya solución es $y(x) = e^x$. Gráfica las soluciones.

2.1 Pseudocódigo

Se presentan los pseudocódigos de las rutinas unidimensionales para resolver una ecuación diferencial de un problema bien planteado [1] de la forma

$$\frac{dy}{dt} = f(t, y), \quad a \leq x \leq b, \quad y(a) = \alpha \tag{1}$$

Se elaboraron los métodos de Euler, Heun, Taylor de segundo orden y Runge-Kutta de cuarto orden o RK4. Los métodos se encuentran en una biblioteca llama **EDO.h**.

```
procedure Euler(a, b, N: Real; alpha: Real; f: Function; var w: array of Real
    );
var
    h, t: Real;
    i: Integer;
begin
    h := (b - a) / N;
    t := a;
    w[0] := alpha;

    for i := 1 to N do
    begin
        w[i] := w[i-1] + h * f(t, w[i-1]);
        t := a + i * h;
    end;
```

```
end;
```

Listing 1: Método de Euler

```
procedure Heun(a, b, N: Real; alpha: Real; f: Function; var w: array of Real)
    ;
var
    h, t, w_star: Real;
    i: Integer;
begin
    h := (b - a) / N;
    t := a;
    w[0] := alpha;

    for i := 1 to N do
    begin
        w_star := w[i-1] + h * f(t, w[i-1]);
        w[i] := w[i-1] + (h / 2) * (f(t, w[i-1]) + f(t + h, w_star));
        t := a + i * h;
    end;
end;
```

Listing 2: Método de Heun

```
procedure Taylor2(a, b, N: Real; alpha: Real; f, dfdt, dfdy: Function; var w:
     array of Real);
var
    h, t: Real;
    i: Integer;
begin
    h := (b - a) / N;
    t := a;
    w[0] := alpha;

    for i := 1 to N do
    begin
        w[i] := w[i-1] + h * f(t, w[i-1]) + (h * h / 2) * (dfdt(t, w[i-1]) +
    dfdy(t, w[i-1]) * f(t, w[i-1]));
        t := a + i * h;
    end;
end;
```

Listing 3: Método de Taylor de segundo orden

```
procedure RK4(a, b, N: Real; alpha: Real; f: Function; var w: array of Real);
var
    h, t, k1, k2, k3, k4: Real;
    i: Integer;
begin
    h := (b - a) / N;
    t := a;
    w[0] := alpha;

    for i := 1 to N do
    begin
        k1 := h * f(t, w[i-1]);
        k2 := h * f(t + h / 2, w[i-1] + k1 / 2);
        k3 := h * f(t + h / 2, w[i-1] + k2 / 2);
```

```
        k4 := h * f(t + h, w[i-1] + k3);

        w[i] := w[i-1] + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
        t := a + i * h;
    end;
end;
```

Listing 4: Método RK4

2.2 Ejecución

Se presentan los resultados de manera conjunta en una tabla. También se presentan los resultados de la solución exacta para los sistemas en los que si se pueda solucionar la ecuación diferencial.

Para el ejemplo del burden

$$y' = y - t^2 + 1, \quad y(0) = 0.5 \tag{2}$$

```
guillermo_sego@MacBook-Air Tarea13 % make
gcc -g -o build/Debug/EDO_Solution.o -c EDO_Solution.c
gcc -g -o build/EDO_Solution build/Debug/EDO.o build/Debug/matrix.o build/
    Debug/EDO_Solution.o
guillermo_sego@MacBook-Air Tarea13 % ./build/EDO_Solution
t         Euler           Heun            Taylor2         RK4
0.00      0.500000        0.500000        0.500000        0.500000
0.20      0.800000        0.826000        0.830000        0.829293
0.40      1.152000        1.206920        1.215800        1.214076
0.60      1.550400        1.637242        1.652076        1.648922
0.80      1.988480        2.110236        2.132333        2.127203
1.00      2.458176        2.617688        2.648646        2.640823
1.20      2.949811        3.149579        3.191348        3.179894
1.40      3.451773        3.693686        3.748645        3.732340
1.60      3.950128        4.235097        4.306146        4.283409
1.80      4.428154        4.755619        4.846299        4.815086
2.00      4.865785        5.233055        5.347684        5.305363
```

Ahora para el sistema proporcionado en el problema de la tarea

$$y' = y, y(0) = 1 \tag{3}$$

```
guillermo_sego@MacBook-Air Tarea13 % make
gcc -g -o build/Debug/EDO_System.o -c EDO_System.c
gcc -g -o build/EDO_Solution build/Debug/EDO.o build/Debug/matrix.o build/
    Debug/EDO_Solution.o
guillermo_sego@MacBook-Air Tarea13 % ./build/EDO_Solution
t         Euler           Heun            Taylor2         RK4
0.00      1.000000        1.000000        1.000000        1.000000
0.40      1.400000        1.480000        1.480000        1.491733
0.80      1.960000        2.190400        2.190400        2.225268
1.20      2.744000        3.241792        3.241792        3.319507
1.60      3.841600        4.797852        4.797852        4.951819
2.00      5.378240        7.100821        7.100821        7.386794
2.40      7.529536        10.509215       10.509215       11.019126
2.80      10.541350       15.553639       15.553639       16.437598
```

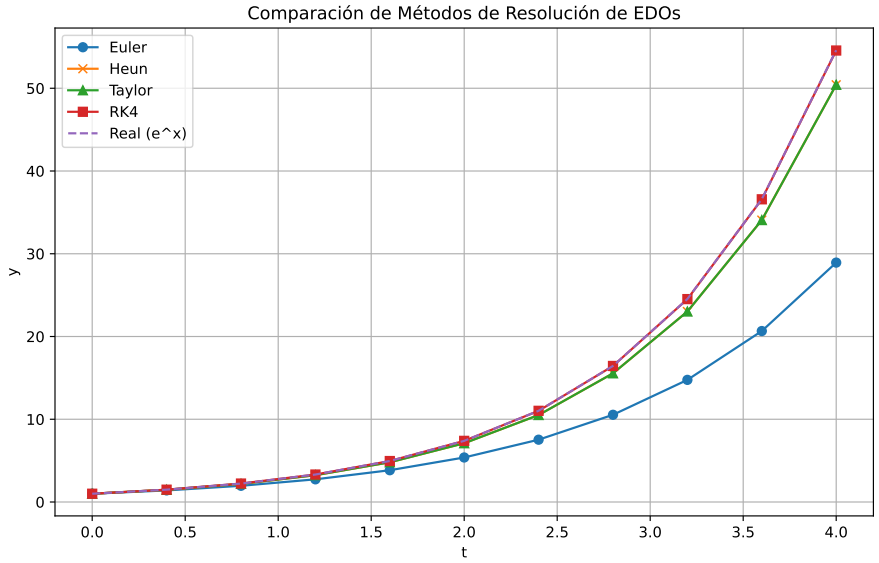

Figure 1: Solución del sistema 3 por cuatro métodos para ecuaciones diferenciales

3.20	14.757891	23.019385	23.019385	24.520513
3.60	20.661047	34.068690	34.068690	36.578067
4.00	28.925465	50.421662	50.421662	54.564722

La mejor solución en términos de exactitud es el método de Runge-Kutta de cuarto orden. Podemos apreciar gráfica de la la solución del sistema en la figura 1

3 Problema 2

Se define la función $y(x)$ de la forma

$$y(x) = \int_0^x \sqrt{1+t^3}dt. \tag{4}$$

dónde $0 \leq x \leq 2$. Nota: La integral $\int \sqrt{1+t^3}dt$ es elíptica y por lo tanto no es posible obtener una primitiva expresable en términos de funciones elementales. a) Utilizando la fórmula de cuadratura de gauss de 3 puntos aproxima el valor de la integral. b) La igualdad del ejercicio se puede escribir como un problema de Cauchy, es decir, un problema de una ecuación diferencial con condición inicial $y(0) = 0$, halla el valor de $y(2)$ utilizando los 4 métodos programados en el inciso 1 usando $h = 0.2$ (es equivalente a encontrar el valor de la integral del inciso a)). Realiza una tabla comparativa de los 4 métodos por cada incremento de h. ¿Qué piensas sobre los métodos? Grafica las aproximaciones de las soluciones.

Obtener una tabulación de la función $y(x)$ en el intervalo $0 \leq x \leq 2$ utilizando el método de Taylor de segundo orden con paso $h = 1/2$ para resolver de forma numérica un problema de valor inicial adecuado.

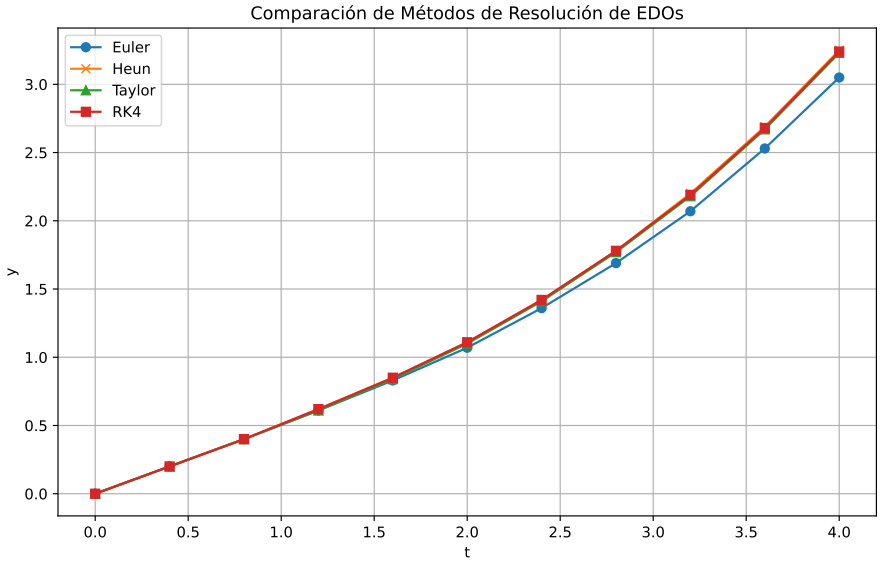

Figure 2: Solución para la integral planteada en el problema 2 utilizando diferentes métodos de solución de ecuaciones diferenciales

3.1 Ejecución

Para resolver este problema se utilizó la función de cuadratura gaussiana de tres puntos realizada en la tarea 11 para aproximar la solución de la integral. A si mismo se utilizaron los métodos implementados en el inciso anterior para calcular la misma aproximación de la integral. Por este motivo no se presentan los pseudocódigos. Todo se ejecuta y compara en el mismo script main().

```
guillermo_sego@MacBook-Air Tarea13 % make
gcc -g -o build/Debug/T13_P2.o -c T13_P2.c
gcc -g -o build/T13_P2 build/Debug/EDO.o build/Debug/matrix.o build/Debug/
    T13_P2.o
guillermo_sego@MacBook-Air Tarea13 % ./build/T13_P2
La integral por Cuadratura gaussiana de orden 3 es: 3.2418156124
 t       Euler           Heun            Taylor2         RK4
0.00     0.000000        0.000000        0.000000        0.000000
0.20     0.200000        0.200399        0.200000        0.200200
0.40     0.400798        0.403949        0.401994        0.403171
0.60     0.607099        0.617372        0.612948        0.615733
0.80     0.827644        0.850607        0.843286        0.847996
1.00     1.073571        1.114992        1.104828        1.111446
1.20     1.356413        1.421580        1.408884        1.417210
1.40     1.686747        1.780241        1.765372        1.775166
1.60     2.073735        2.199478        2.182749        2.193798
1.80     2.525221        2.686602        2.668256        2.680395
2.00     3.047983        3.247983        3.228205        3.241307
```

Comparamos los resultados con la aproximación de Gauss. Nuevamente el método que mejor aproxima la solución es el método de Runge-kutta de cuarto orden.

La solución para la integral se presenta en la figura 2.

4 Problema 3

Considera el siguiente problema sobre la dinámica depredador-presa de lices-conejos. La siguiente tabla muestra el índice de capturas de linces y conejos elaborada por la compañía Hudson Bay entre los años 1900 y 1920.

Año	Conejos	Linces	Año	Conejos	Linces
1900	30	4	1911	40.3	8
1901	47.2	6.1	1912	57	12.3
1902	70.2	9.8	1913	76.6	19.5
1903	77.4	35.2	1914	52.3	45.7
1904	36.3	59.4	1915	19.5	51.1
1905	20.6	41.7	1916	11.2	29.7
1906	18.1	19	1917	7.6	15.8
1907	21.4	13	1918	14.6	9.7
1908	22	8.3	1920	16.2	10.1
1909	25.4	9.1	1921	24.7	8.6
1910	27.1	7.4	1922	-	-

La dinámica del comportamiendo depredador-prese se puede analizar mediante ecuaciones diferenciales del tipo lotka-volterra, para este ejemplo, las ecuaciones diferenciales para e problema de valor inicial que definen el comportamiento de las especies a través del tiempo están dadas por,

$$\begin{cases} x'(t) = 0.4x(t) - 0.018x(t)y(t) \quad ; \quad x(0) = 30 \\ y'(t) = -0.8y(t) + 0.023x(t)y(t) \quad ; \quad y(0) = 4 \end{cases} \tag{5}$$

a) Grafica los puntos de la Tabla de datos medidos. b) Modifica los 4 metodos programados en el inciso 1, y resuelve el sistema de ecuaciones diferenciales con las condiciones iniciales establecidas, para t = [0, 25] c) En algunos casos, para el analisis de sistema de ecuaciones diferenciales se estudia el comportamiento del plano fase mediante las denominadas orbitas de la solución, el cual está dato por la grafica de las componentes del sistema, para nuestro ejercicio, debes graficar las coordenadas (x, y) = (conejos, presas)

4.1 Ejecución

Presentamos la gráfica en python para los puntos del inciso a.

```
import matplotlib.pyplot as plt
# Datos de la tabla
aos = [1900, 1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910,
    1911, 1912, 1913, 1914, 1915, 1916, 1917, 1918, 1920, 1921]
conejos = [30, 47.2, 70.2, 77.4, 36.3, 20.6, 18.1, 21.4, 22, 25.4, 27.1,
    40.3, 57, 76.6, 52.3, 19.5, 11.2, 7.6, 14.6, 16.2, 24.7]
linces = [4, 6.1, 9.8, 35.2, 59.4, 41.7, 19, 13, 8.3, 9.1, 7.4, 8, 12.3,
    19.5, 45.7, 51.1, 29.7, 15.8, 9.7, 10.1, 8.6]

# Graficar
plt.figure(figsize=(10, 6))

plt.plot(aos, conejos, 'o-', label='Conejos')
plt.plot(aos, linces, 's-', label='Linces')
```

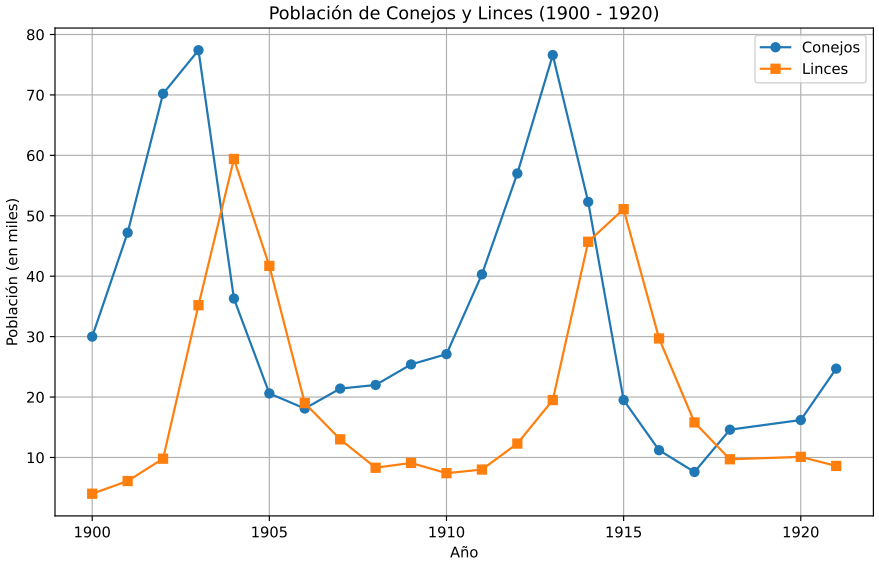

Figure 3: Grafica de las capturas de los conejos y liebres. Datos medidos

```
plt.title('Población de Conejos y Linces (1900 - 1920)')
plt.xlabel('Ao')
plt.ylabel('Población (en miles)')
plt.legend()
plt.grid()
plt.savefig("Problema2.pdf")
plt.show()
```

Listing 5: Gráfica de puntos

Para poder implementar los métodos se realizó la modificación de las funciones unidimensionales para que funcionales como funciones multidimensionales, pudiendo resolver sistemas de ecuaciones diferenciales. Las funciones modificadas se encuentran en la misma biblitoeca **EDO.h**. Para probar la solución, se utilizó el mismo ejemplo del libro burden mostrado en el inciso 1. La prueba para una dimensión se encuentra en el script *EDO_System1d*.

```
gcc -g -o build/Debug/EDO_System1d.o -c EDO_System1d.c
gcc -g -o build/EDO_System1d build/Debug/EDO.o build/Debug/matrix.o build/
    Debug/EDO_System1d.o
guillermo_sego@192 Tarea13 % ./build/EDO_System1d
t         Euler           Heun            Taylor2         RK4
0.00      0.500000        0.500000        0.500000        0.500000
0.20      0.800000        0.826000        0.830000        0.829293
0.40      1.152000        1.206920        1.215800        1.214076
0.60      1.550400        1.637242        1.652076        1.648922
0.80      1.988480        2.110236        2.132333        2.127203
1.00      2.458176        2.617688        2.648646        2.640823
1.20      2.949811        3.149579        3.191348        3.179894
1.40      3.451773        3.693686        3.748645        3.732340
1.60      3.950128        4.235097        4.306146        4.283409
1.80      4.428154        4.755619        4.846299        4.815086
2.00      4.865785        5.233055        5.347684        5.305363
```

Se puede corroborar que las soluciones son las mismas que con los casos unidimensionales. Se resolvió el sistema de ecuaciones planteado en el problema utilizando estos métodos. La ejecución se encuentra en el script *EDO_System* y se muestra a continuación.

```
make
gcc -g -o build/Debug/EDO_System.o -c EDO_System.c
gcc -g -o build/EDO_System build/Debug/EDO.o build/Debug/matrix.o build/Debug
    /EDO_System.o
guillermo_sego@192 Tarea13 % ./build/EDO_System
 t       x(Euler)        y(Euler)        x(Heun)         y(Heun)         x(
    Taylor)       y(Taylor)       x(RK4)  y(RK4)
0.00     30.000000       4.000000        30.000000       4.000000
    30.000000       4.000000        30.000000      4.000000
2.50     54.600000       2.900000        66.037350       5.102275
    39.020000       3.338750        65.992549      7.887898
5.00     102.074700      6.204550        112.388626      43.386095
    44.129067       4.491312        54.921786      53.042852
7.50     175.649659      30.211786       33.005435       -60.332230
    45.376282       7.876446        16.768211      52.425241
10.00    112.498273      274.922882      361.783628      -245.428387
    45.569062       14.701660       10.752640      13.578293
12.50    -1166.779182    1503.457216     520956.955864   -657098.644468
    45.601864       27.704296       20.522180      4.476400
15.00    76605.557773    -102370.105614  6822484811654074368.000000
    -8717619452031865856.000000     45.586461      52.291606        47.229295
         3.941935
17.50    353048567.910302        -450819474.687684
    20594148759611177443746374470832244274560160906592909893924822595718348 8.000000

    -26314745637280952497665617380562118112340001683583275894434355696631808 0.000000
            45.480053     98.624824       84.495500       26.525052
20.00    7162253352702405.000000         -9151766819842814.000000
    17098142032144854062596920436575782133727036931299760160149585284763559517623058422650519560632

    -21847625929962869079984953891180166059762324967771915760191136752753437161407241317831219438588
         44.825915       185.035350      58.735609       39.239142
22.50    29496272664856558789440254771 20.000000
    -37689681738427555529362631557 12.000000         inf     nan     38.486542
         336.045013      23.940830       40.826969
25.00    50026730814374585890493748461046937166156561888100132428185 6.000000
        -63923044929478638022314127445351600635327124026980424902246 4.000000
      nan     nan     -52.024750      439.389139      18.605993
    16.705934
```

Las soluciones de algunos métodos divergen en su mayoría acercándose a la posición final. Las gráficas de orbitas las observamos a continuación:

Es evidente que la única gráfica que convergió a una orbita fue la del método de runge-kutta de cuarto orden confirmando que es el método mas poderoso explorado para resolver sistemas de ecuaciones diferenciales.

5 Conclusión

En este trabajo, hemos explorado algunos métodos numéricos para la resolución de ecuaciones diferenciales. Se utilizaron técnicas como Euler, Heun, Taylor y Runge-Kutta, las

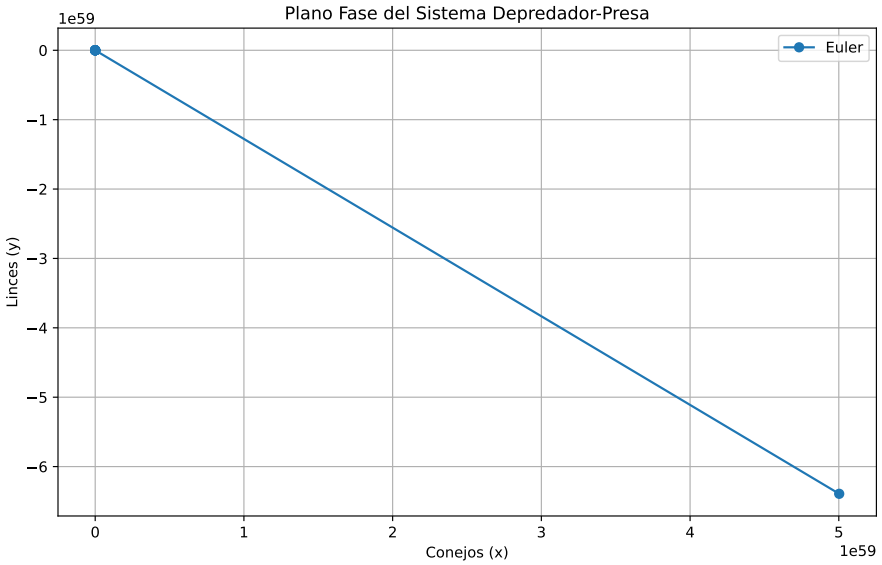

Figure 4: Orbita de la solución para Euler

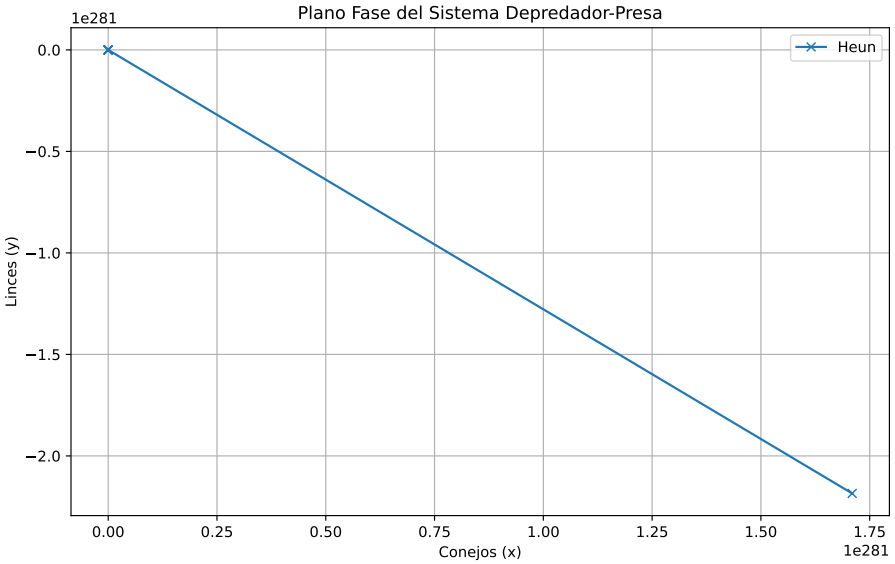

Figure 5: Orbita de la solución para Heun

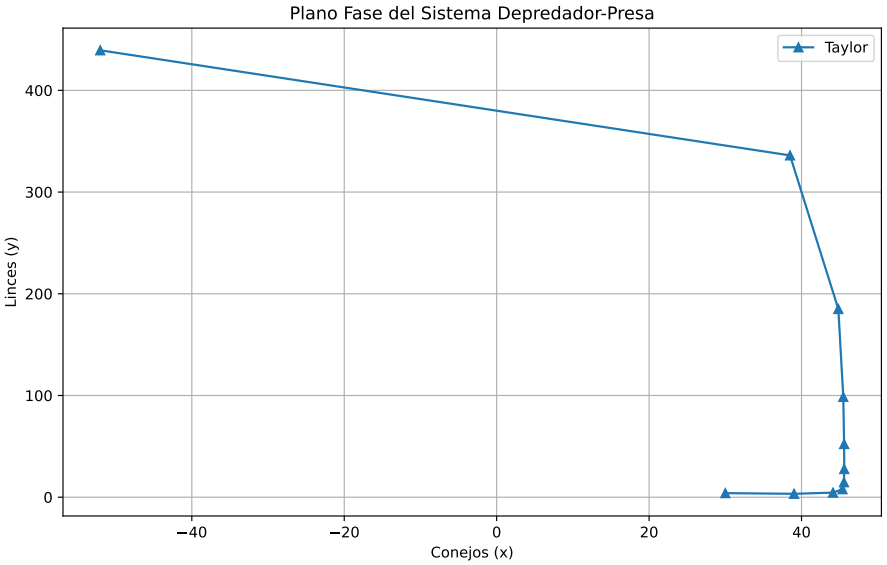

Figure 6: Orbita de la solución para Taylor de segundo orden

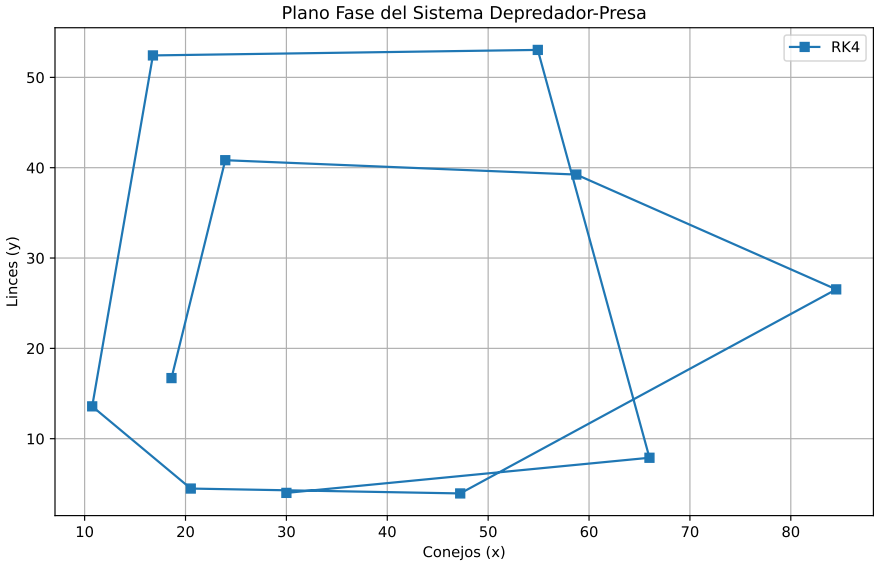

Figure 7: Orbita de la solución para RK4

cuales nos permitieron encontrar muy buenas aproximaciónes a diferentes problemas bien planteados. Estas metodologías han demostrado ser cruciales para aproximarse a la solución de ecuaciones que, por su naturaleza complicada es díficil a un análisis analítico.

El método de Runge-kutta de orden 4 fue el mejor en término de exactitud para aproximar la solución de las ecuaciones diferenciales. Incluso este tipo de planteamientos permitió definir un problema en el que se pudieron utilizar métodos de solución de ecuaciones diferenciales para calcular una integral numérica.

Además, los métodos pueden ser aplicados a la solución de sistemas de ecuaciones diferenciales como el problema de depredador-presa. La aplicación práctica de estos métodos en la simulación de interacciones depredador-presa, ilustradas por las ecuaciones de Lotka-Volterra, revela no el gran potencial en la modelización de sistemas biológicos planteados como sistemas de ecuaciones diferenciales.

En conclusión, este trabajo resalta la importancia y la aplicabilidad de estas técnicas numéricas en el análisis de ecuaciones diferenciales. El método de Runge-kutta es muy superior, por lo cual si se desea escoger un método, se tiene que priorizar este.

References

[1] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical analysis*. Cengage learning, 2015.