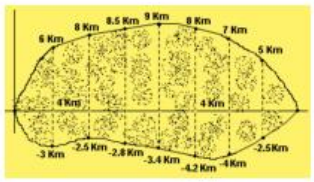
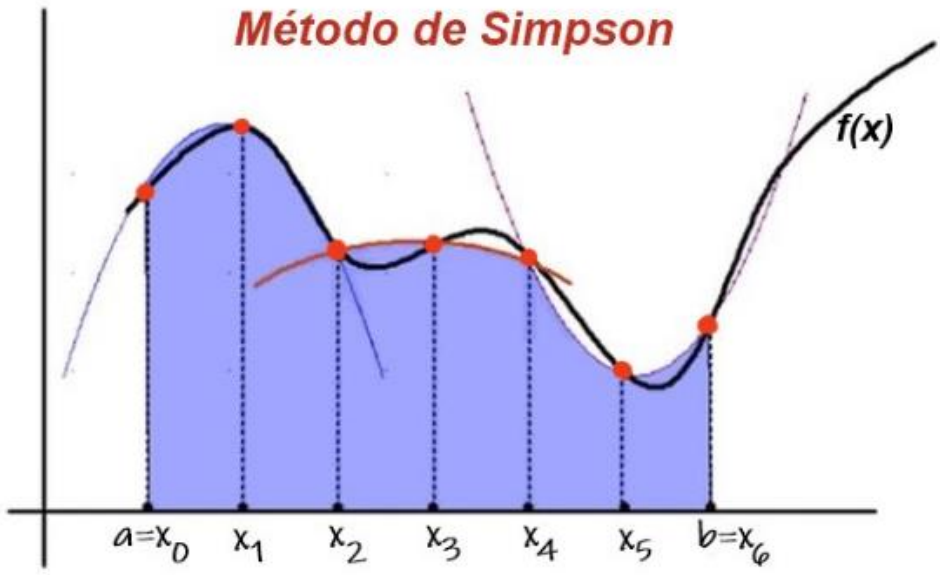
K.

k. En el siguiente gráfico se muestra delineada la zona de derrame de petróleo ocurrido en Caño Limón, donde las mediciones han sido obtenidas a distancias de 4Km. Con la fórmula de Simpson encuentre una aproximación del área total de afectación.



La regla de Simpson es un método de integración numérica. En otras palabras, es la aproximación numérica de integrales definidas.

Los valores extremos de dos sub-intervalos consecutivos definen tres puntos, por los que se ajusta una parábola, cuya ecuación es un polinomio de segundo grado.



Luego el área bajo la curva de la función en los dos intervalos consecutivos se aproxima por el área del polinomio de interpolación. Sumando la contribución al área bajo la parábola de todos los sub-intervalos sucesivos, se tiene el valor aproximado de la integral.

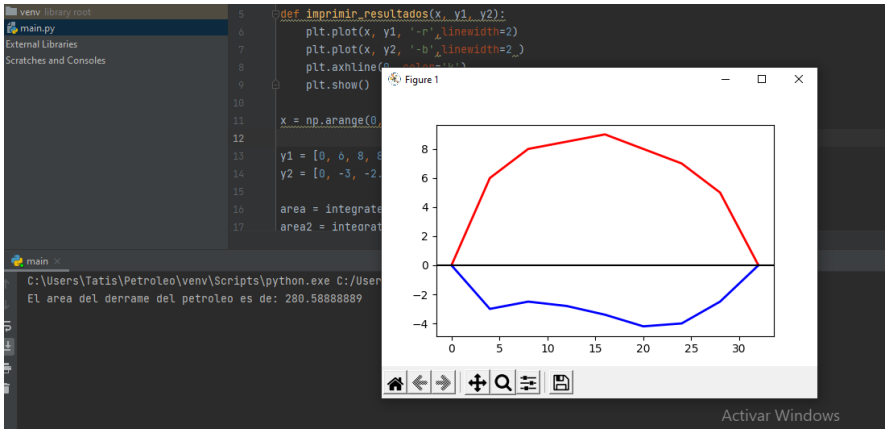
Por otra parte, como la integral de una parábola puede calcularse algebraicamente en forma exacta, entonces es posible encontrar una fórmula analítica para el valor aproximado de la integral definida. Es conocida como la fórmula de Simpson.

Código:

II.

```python
import matplotlib.pyplot as plt
import numpy as np
from scipy import integrate

def imprimir_resultados(x, y1, y2):
    plt.plot(x, y1, '-r', linewidth=2)
    plt.plot(x, y2, '-b', linewidth=2)
    plt.axhline(0, color='k')
    plt.show()

x = np.arange(0,33,4).tolist()

y1 = [0, 6, 8, 8.5, 9, 8, 7, 5,0]
y2 = [0, -3, -2.5, -2.8, -3.4, -4.2, -4, -2.5, 0]

area = integrate.simpson(y1, x)
area2 = integrate.simpson(x, y2)
print("El area del derrame del petroleo es de: {:.8f}".format(area + area2))

imprimir_resultados(x,y1, y2)
```

Resultados:



II. Considere el problema de valor inicial

$$y' = t\exp(3t) - 40y,\ t \in [1,2],\ y(1) = 10.$$

a. G2: Utilice Euler para aproximar las soluciones en t=0.4;0.01;1.55 y estime el error de truncamiento

El Método de Euler es un procedimiento de primer orden, lo cual supone que el error local es proporcional al cuadrado del tamaño del paso, y el error universal es proporcional al tamaño del paso.

Definiciones de funciones a usar

```python
import math
import numpy as np
import matplotlib.pyplot as plt

def funcion(x,y):
    ec = x * math.exp(3*x) - 40*y
    return ec

def cont_digitos(n):
    cant = 1
    while n > 9:
        n = n / 10
        cant = cant + 1
    return cant

def digitos(n):
    return list(map(int, str(n)))
```

```python
def error(n, capacidad):
    exponente = cont_digitos(n)
    notacion = int(n * (pow(10, exponente - 1)))
    arreglo = digitos(notacion)

    truncamiento = 0
    for i in range(capacidad, len(arreglo)):
        truncamiento = (truncamiento * 10) + arreglo[i]

    valor = truncamiento / 10
    error = exponente - capacidad

    print("El error de redondeo es ", valor, "x10^", error)
```

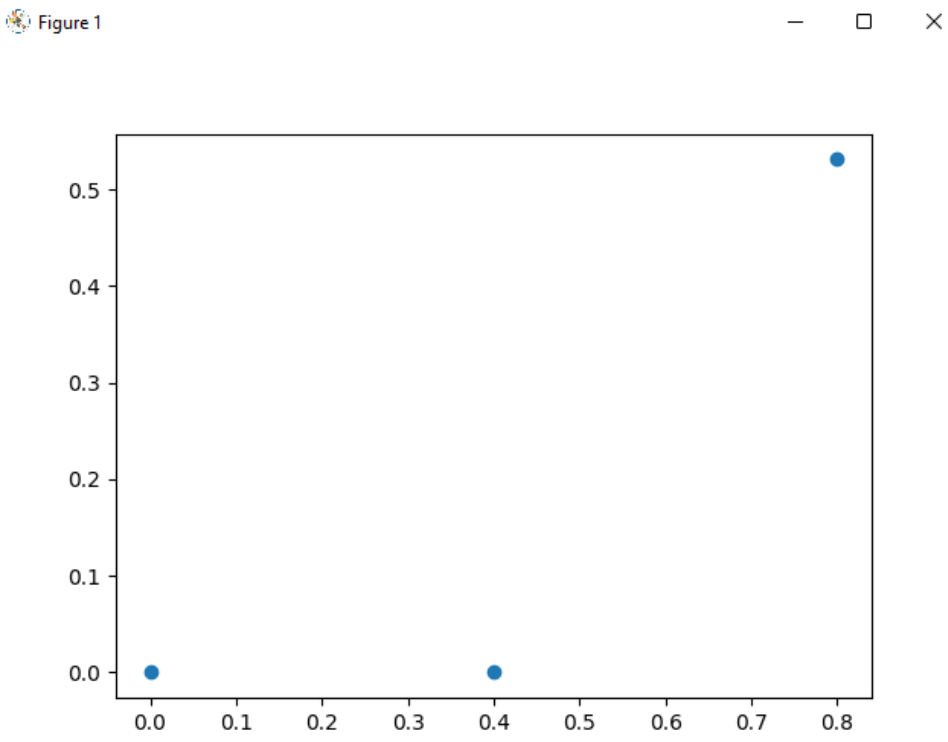
PROCEDIMIENTO

```
h =  float( input("Tamaño de paso: "))
s =  float( input("Hasta que valor? "))
n = int((s / h) + 1)
x = np.zeros(n)
y = np.zeros(n)

print(x[0], y[0])

for i in np.arange(1,n):
    y[i] = y[i - 1] + (funcion(x[ i - 1], y[i - 1])) * h
    x[i] = x[i - 1] + h
    print(x[i],y[i])

error(n,int(h))
plt.scatter(x,y)
plt.show()
```

Salida

Ejemplo con t = 0.4

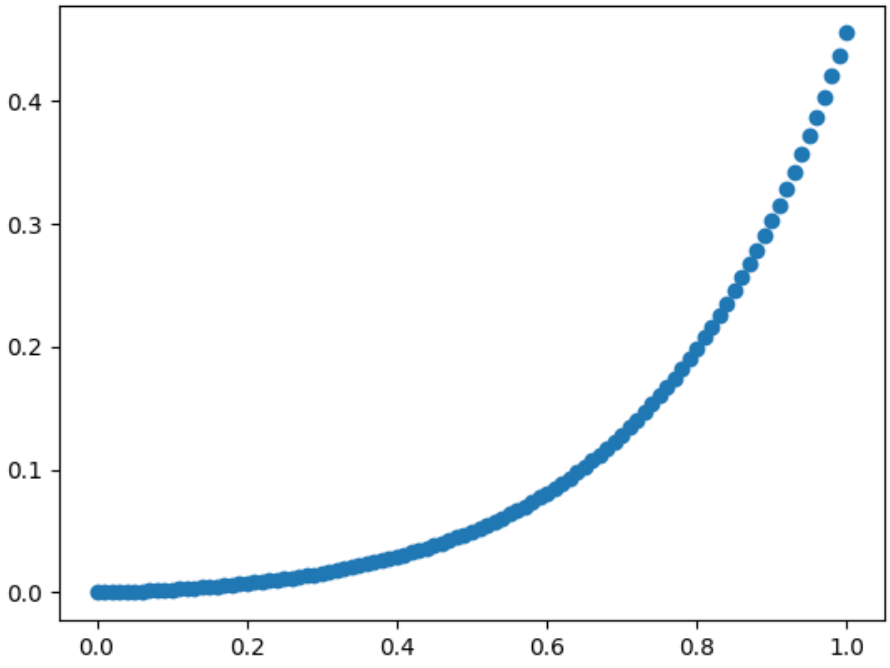
```
"C:\Users\juanf\PycharmProjects\Ecuac
Tamaño de paso: .4
Hasta que valor? 1
0.0 0.0
0.4 0.0
0.8 0.5312187076378477
El error de redondeo es  0.3 x10^ 1
```



Ejemplo con t = 0.01

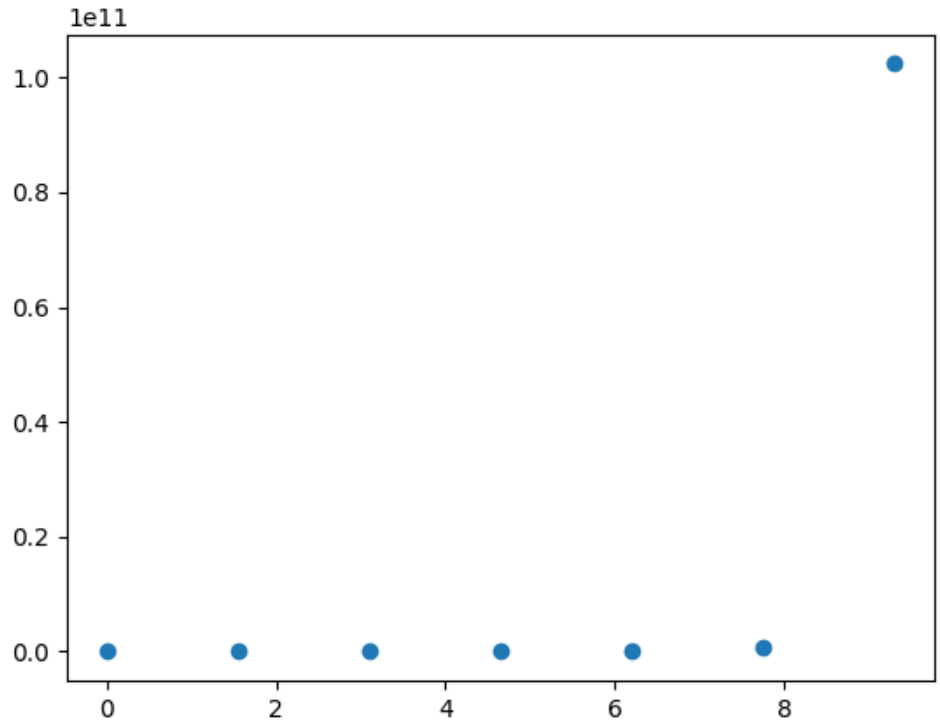
```
Tamaño de paso: 0.01
Hasta que valor? 1
0.0 0.0
0.01 0.0
0.02 0.0001030454533953517
0.03 0.000274194581346283
0.04 0.0004927690339193329
0.05 0.00074666016098335
0.060000000000000005 0.001028913217954151
0.07 0.0013356783486455771
0.08 0.0016649816511570665
0.09 0.002015988310951364
0.09999999999999999 0.002388560992230741
0.10999999999999999 0.002782995402914448
0.11999999999999998 0.0031998621830588267
0.12999999999999998 0.0036399126073077036
0.13999999999999999 0.004104022596432057
0.15 0.004593159735725321
0.16 0.005108364119670446
0.17 0.0056507375153108975
0.18000000000000002 0.006221437540594546
0.19000000000000003 0.006821674876289474
0.20000000000000004 0.007452712323497782
0.21000000000000005 0.008115864994879688
0.22000000000000006 0.008812501213382935
0.23000000000000007 0.00954404386371423
0.24000000000000007 0.01031197204468763
0.25000000000000006 0.01111782293235791
0.26000000000000006 0.011963193800946435
0.27000000000000001 0.012849744170863186
0.28000000000000001 0.013779198066544388
0.29000000000000001 0.014753346374913693
0.30000000000000001 0.0157740493001686 2
0.31000000000000001 0.016843238913572027
0.32000000000000001 0.0179629217987585 7
0.33000000000000001 0.019135181794209123
0.34000000000000014 0.020362182835278048
0.35000000000000014 0.02164617189864545
```

El error de redondeo es 1010.0 x10^ 3



Ejemplo con t = 1.55

```
Tamaño de paso: 1.55
Hasta que valor? 10
0.0 0.0
1.55 0.0
3.1 251.26542784901693
4.65 37229.99119644371
6.2 5974008.76549811
7.75 785328404.2492278
9.3 102402278199.43587
El error de redondeo es  0.0 x10^ 0
```



IV. Dado el sistema de ecuaciones diferenciales que corresponden a una muestra estudio del sistema depredador presa de capturas de linces y conejos entre los años 1900 y 1920:

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

Utilizando:

b. G2 y G3 y G4: Runge- Kutta de orden 4

* Encuentre la solución numérica del sistema de ecuaciones diferenciales con una evolución por año.

* Realice la gráfica que nos muestra la evolución de las presas.

* Compare la solución con los datos reales y evalúe el error total promedio y el error local, en que año se produce el mayor error.

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	-	-

Informe:

Runge-Kutta:

- El método Runge-Kutta es un método propuesto por los matemáticos Karl Runge y Martin Wilhelm en 1900 para resolver ecuaciones diferenciales ordinarias no lineales .El método de Runge-Kutta de 4to orden busca reducir el error de la curvatura para el método de Euler utilizando varias evaluaciones del campo de direcciones y realizando una ponderación.

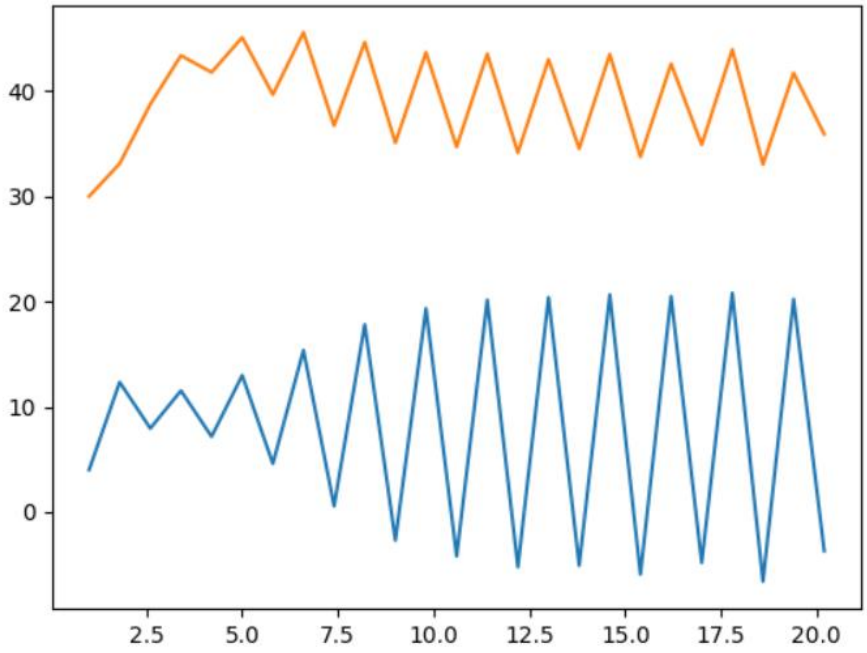
Resultados:

Código:

```python
import numpy as np
import matplotlib.pyplot as plt

def g(x,y,t):
    return -0.8*y*t+0.023*x*t*y*t
def f(x,y,t):
    return 0.4*x*t-0.018*x*t*y*t

conejosV = 0

def runge_kutta_sis(f,g,a,b,x0,y0,h):
    t = np.arange(a, b + h, h)
    n = len(t)
    x = np.zeros(n); y = np.zeros(n);
    x[0] = x0; y[0] = y0;
    er = np.zeros(n)
    for i in range (0,n-1):
        k1 = h*f(x[1],y[1],t[1])
        l1 = h*g(x[1],y[1],t[1])
        k2 = h*f(x[i]+h/2,y[i]+k1*h/2,t[1]+h/2)
        l2 = h*g(x[i]+h/2,y[i]+l1*h/2,t[1]+h/2)
        k3 = h*f(x[i]+h/2,y[i]+k2*h/2,t[1]+h/2)
        l3 = h*g(x[i]+h/2,y[i]+l2*h/2,t[1]+h/2)
        k4 = h*f(x[i]+h,y[i]+k3*h,t[1]+h)
        l4 = h*f(x[i]+h,y[i]+l3*h,t[1]+h)
        x[i+1] = x[i]+(h/6)*(l1+2*l2*2+l3+l4)
        y[i+1] = y[i] +(h/6)*(k1+2*k2*2+k3+k4)
        er[i] = abs(x[i]-y[i])
        conejos = x[i]
        conejosV = conejosV + conejos
    z = conejosV/20
    error = 34.4 - z
    print("Error local: ", er)
    print("error Total Promedio:",error)
    plt.plot(t,y,t,x)
    plt.show()

runge_kutta_sis(f,g,1,20,30,4,0.8)
```

Salida:

Gráfica:



Comparando con los datos reales de la tabla los datos de la gráfica están muy cerca e incluso podría decirse que son los mismos datos que los de la tabla

Error local:

```
Error local:  [26.         20.77243466 30.81247953 31.85752394 34.6203552  32.12054525
 35.08091731 30.20891645 36.15326419 26.83201738 37.79414255 24.34727963
 38.9031445  23.39097684 39.38536138 22.62347113 39.62230282 22.85304001
 39.67304357 22.10698906 39.72872775 23.14295614 39.64114209 21.49445116
  0.        ]
```

Error promedio:

```
error Total Promedio: -12.65701831565125
```