**Universidad Central de Venezuela**

**Facultad de Ingeniería**

**Escuela de Ingeniería Eléctrica**

# Proyecto 2

**Periodo 2020-3**

**19/01/2021**

**Prof. Gilberto R. Noguera**

**Alumno: José Páez C.I.: 24 311 351**

# Planteamiento de los Problemas

## 1.)

Se Considerará $x_0 = [1, 1, \cdots, 1]$ y la matriz tridiagonal de orden n = 20 de la forma,

$$A = \begin{pmatrix} 4 & 2 & 0 & 0 & \cdot & 0 \\ 1 & 4 & 1 & 0 & \cdots & 0 \\ 0 & \vdots & \vdots & & \vdots & 0 \\ \vdots & \vdots & 0 & 1 & 4 & 1 \\ 0 & \cdots & 0 & 0 & 2 & 4 \end{pmatrix}.$$

A la cual se aplicarán los métodos de la potencia y la potencia inversa para hallar eigenvalores de A. Comparando los resultados.

## 2.)

Un problema importante que concierne a la probabilidad se expresa de la siguiente forma,

$$P[a \leq x \leq b] = \int_a^b f_X(x)dx$$

donde

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp \frac{-x^2}{2\sigma^2}$$

La función $f_X(x)$ es llamada función de densidad de probabilidad Gaussiana, con media cero y varianza $\sigma^2 = 9$. Se Utilizará la función x = np.random.normal(mu,sigma,1000) de Python para generar 1000 valores simulados que siguen una distribución normal, elegiendo los valores x tal que −1 ≤ x ≤ 1 para calcular, aplicando los reglas del Trapecio compuesta y Sipmson compueta,

$$P = P[-1 \leq x \leq 1]$$

.

## 3.)

Se Expresará la ecuación de tercer orden, $x''' + tx'' - tx' - 2x = t$ con condiciones iniciales: $x(0) = x''(0) = 0, x'(0) = 1$ como un conjunto de ecuaciones de primer orden y se resolverá en $t = 0.2; 0.4; 0.6$ con el metodo de Runge-Kutta con $h = 0.02$.

# Resolución

## 1.)

Se formó la matriz a partir de ciclos for y ciertas condiciones, haciendo posible variar las dimensiones de esta, pero resolviendo el problema con n = 20, tal como se indica.

Luego se aplicaron los métodos de la Potencia y Potencia Inversa para hallar el mayor y el menor, eigenvalor y eigen vector asociados, respectivamente.

- Para el método de la Potencia los resultados obtenidos fueron :

Eigenvalor = 6.7988406368112235

Eigenvector: [0.15394368 0.21485278 0.44588292 0.40772648 0.69312092 0.5607923 0.87474525 0.66248703 0.9787536 0.70700585 1. 0.69237751 0.93755205 0.61912909 0.79402457 0.49079667 0.57752209 0.31582922 0.30440204 0.10906011]

- Mientras que para el método de la Potencia Inversa los resultados obtenidos fueron :

Eigenvalor = 5.033340463100365

Eigenvector: [ 0.33171531 0.17138743 -0.15461375 -0.25127175 -0.10503551 0.19700302 0.3086067 -0.03755513 -0.34741393 -0.14194331 0.20073816 0.24565874 0.05311095 -0.21821789 -0.27860433 0.07427132 0.35535189 0.10932842 -0.24237841 -0.23455813]

## 2.)

Para determinar la probabilidad $P = P[-1 \leq x \leq 1]$, se halló el área debajo de la campana de Gauss en dicho intervalo, ello resolviendo la integral de dicha función, por dos métodos:

- A partir de una muestra de 1000 valores simulados que siguen una distribución normal, la seleccion de aquellos entre -1 y 1, el ordenamiento de estos de menor a mayor y su evaluación en la función Gaussiana.

Obteniendose como resultados:

Integral por Trapecio Compuesto = 0.261577027429894

Integral por Simpson Compuesto = 0.263582972184626

- A partir directamente de la función Gaussiana $f(x) = \frac{1}{c\sqrt{2\pi}} e^{-\frac{(x-b)^2}{2c^2}}$ donde la media μ = b y la varianza σ2 = c2, integrnado de -1 a 1.

Obteniendose como resultados:

Integral por Trapecio Compuesto = 0.132004955265513

Integral por Simpson Compuesto = 0.255743622117717

## 3.)

La solución de este problema consistió en transformar la ecuación diferencial de orden tres (3)

$$x''' + tx'' - tx' - 2x = t$$

Dados

$$y = w1$$

$$y' = w'1 = w2$$

$$y'' = w''1 = w'2 = w3$$

$$y''' = w'''1 = w''2 = w'3 = w4$$

en tres (3) ecuaciones diferenciales de orden uno (1).

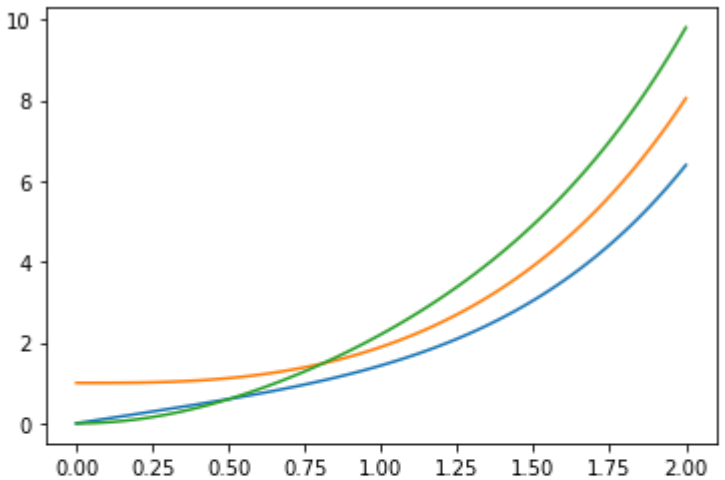
$$w'1 = w2$$

$$w'2 = w3$$

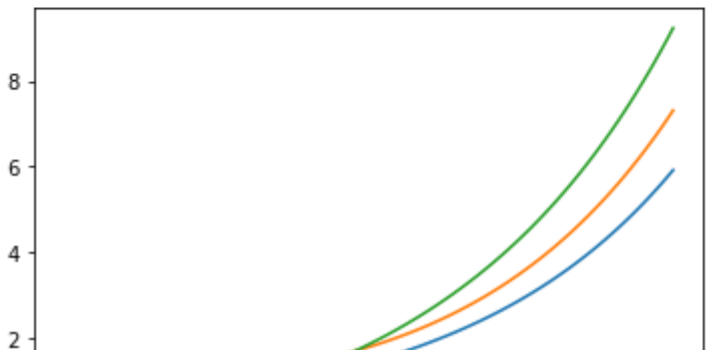
$$w'3 = -tw3 + tw2 + 2w1 + t$$

Luego se procedió a resolver dicho sistema con el método de Runge-Kutta para sistemas, con el parámetro t en 0.2, 0.4 y 0.6.

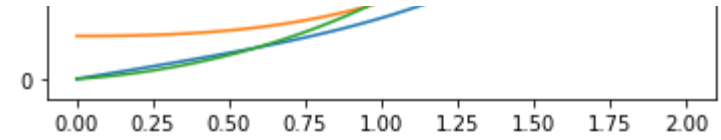
y expresando la solución en grafícas, para mejor visualización de la data, se obtuvo:

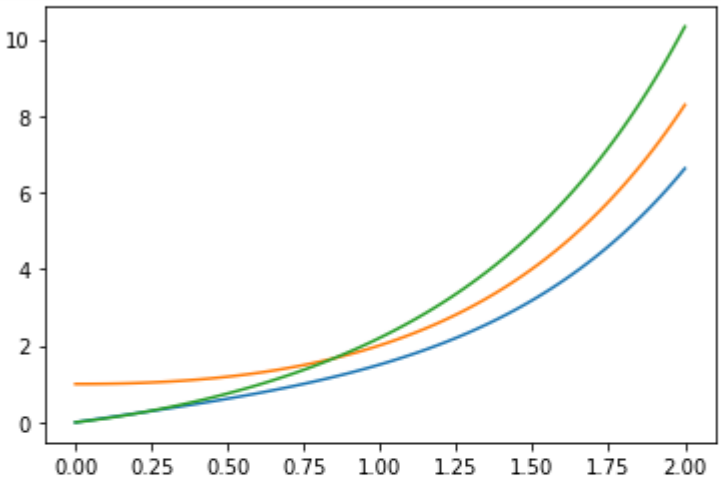



t = t

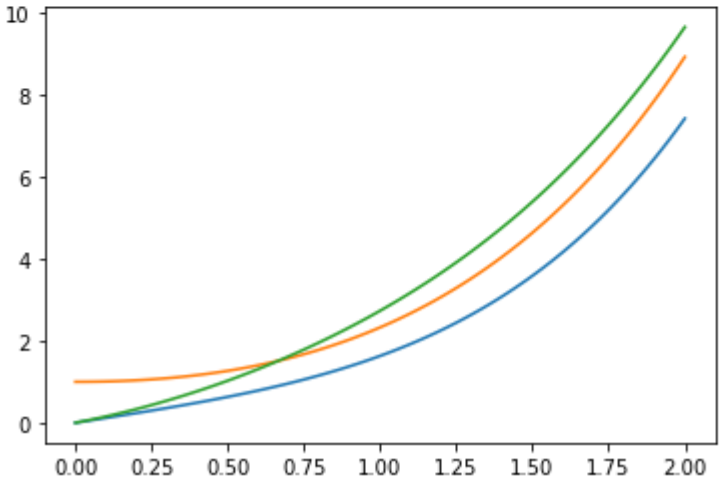



t = 0.2


0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


10
8
6
4
2
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

t = 0.4


10
8
6
4
2
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

t = 0.6

# Conclusiones

Al resolver el problema de Eigenvalores y Eigenvectores, los resultados corresponden a los esperado al estimar los autovalores con los circulos de Gershgorin (los autovalores se encuentran dentro de ellos), lo cual es un señal de que los algoritmos lograron encontrar los valores y vectores correctamente.

Para resolver la integral de la campana de Gauss se usaron 2 métodos, el sugerido por el profesor, a partir de un muestra de valores, y a partir de la misma función de Gauss, arrojando valores de areas cercanos, dadas las dimensiones de la campana.

Para el Sistema de ecuaciones formado a partir de de la ecuación diferencial ordianaria de orden 3, con t: 0.2, 0.4, 0.6, los resultados fueron parecidos debido a la cercanía de t, sin embargo existen diferencias en cuanto a los puntos de corte entre las funciones.

.

Se da por demostrado el inmenso poder de los métodos numéricos para resolver problemas de ciencia e ingeniería; tambien cómo el desarrollo de algoritmos a travez de la historia, nos permite hoy en día aproximar confiablemente resultados a cuestiones irresolubles (practicamente) por métodos analíticos, así como la aplicación en cuestiones reales de las técnicas compiladas en la cátedra de cálculo numérico.

# Bibliografía Consultada

Análisis numérico,10a. ed. Autores : Richard L. Burden ,J. Douglas Faires y Annette M. Burden.

Diferenciación e Integración Numérica. Autor: José Paéz

Autovalores y Autovectores. Autor: José Paéz

Ecuaciones Diferenciales Ordinarias. Autor: José Paéz

Numerical Methods in Engineering. Autor: Jaan Kiusalaas

Función gaussiana (https://es.wikipedia.org/wiki/Funci%C3%B3n_gaussiana)

# Códigos Desarrollados y Comentados

**Problema 1: Eigen-Valores y Eigen-Vectores**

In [1]:
```
import numpy as np

n=20 # n: Determina las dimensiones de la matriz A
A = np.zeros((n,n))
for i in range(n-1):
    A[i][i]=4
for i in range(int((n-1)/2)):
    A[i*2][((i+1)*2)-1]=2
    A[(i*2)+2][((i+1)*2)-1]=2
    A[((i+1)*2)-1][i*2]=1
    A[((i+1)*2)-1][(i*2)+2]=1

A[n-1][n-1]=4
A[n-1][n-2]=1
A[n-2][n-1]=2
#A[0][n-1]=0
print (A)  #A: Matriz de nxn

x0 = np.ones(n)

print(x0)  #b: Vector de dimensión n
```

```
[[4. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 4. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 2. 4. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 4. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 2. 4. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 4. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 2. 4. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 4. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 2. 4. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 4. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 2. 4. 2. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 4. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 2. 4. 2. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 4. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 2. 4. 2. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 4. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 2. 4. 2. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 4. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 2. 4. 2.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 4.]]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [2]:
```
# Método de la Potencia
import numpy as np
def Potencia(x,a):
    def normalizar(x):
        fac= abs(x).max()
        x_n= x / x.max()
        return fac, x_n


    for i in range(100):
        x = np.dot(a, x)
        lambda_1, x= normalizar(x)
    return lambda_1, x
```

In [3]:
```
# Prueba Método de la Potencia


lam,x = Potencia(x0,A)
print("Eigenvalor =",lam)
print("\nEigenvector:\n",x)
```

```
Eigenvalor = 6.7988406368112235

Eigenvector:
 [0.15394368 0.21485278 0.44588292 0.40772648 0.69312092 0.5607923
 0.87474525 0.66248703 0.9787536  0.70700585 1.         0.69237751
 0.93755205 0.61912909 0.79402457 0.49079667 0.57752209 0.31582922
 0.30440204 0.10906011]
```

In [4]:
```
#Método de la Potencia inversa
import numpy as np
import math
from random import random

from numpy import dot

def LUdecomp(a):
    n = len(a)
    for k in range(0,n-1):
        for i in range(k+1,n):
           if abs(a[i,k]) > 1.0e-9:
               lam = a [i,k]/a[k,k]
               a[i,k+1:n] = a[i,k+1:n] - lam*a[k,k+1:n]
               a[i,k] = lam
    return a

def LUsol(a,b):
    n = len(a)
    for k in range(1,n):
        b[k] = b[k] - dot(a[k,0:k],b[0:k])
    b[n-1] = b[n-1]/a[n-1,n-1]
    for k in range(n-2,-1,-1):
       b[k] = (b[k] - dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
    return b

def Potenciainv(a,s,tol=1.0e-6):
    n = len(a)
    aEstrella = a - np.identity(n)*s  # Formar [a*] = [a] - s[I]
    aEstrella = LUdecomp(aEstrella)       # Descomponer [a*]
    x = np.zeros(n)
    for i in range(n):             # Llenar [x] con números random
        x[i] = random()
    xMag = math.sqrt(np.dot(x,x)) # Normalizar [x]
    x =x/xMag
    for i in range(50):            # Comenzar iteraciones
        xViejo = x.copy()           # Salvar actual [x]
        x = LUsol(aEstrella,x)      # Resolver [a*][x] = [xViejo]
        xMag = math.sqrt(np.dot(x,x)) # Normalizar [x]
        x = x/xMag
        if np.dot(xViejo,x) < 0.0:  # Detectar cambio de signo en [x]
            sign = -1.0
            x=-x
        else: sign = 1.0
        if math.sqrt(np.dot(xViejo - x,xViejo - x)) < tol:
            return s + sign/xMag,x
    print("El método de la potencia inversa no converge")
```

In [5]:
```
#Prueba Método de la Potencia Inversa
s = 5.0

lam,x = Potenciainv(A,s)
print("Eigenvalor =",lam)
print("\nEigenvector:\n",x)
```

```
Eigenvalor = 5.033340463100365

Eigenvector:
 [ 0.33171531  0.17138743 -0.15461375 -0.25127175 -0.10503551  0.1970
0302
  0.3086067  -0.03755513 -0.34741393 -0.14194331  0.20073816  0.24565
874
  0.05311095 -0.21821789 -0.27860433  0.07427132  0.35535189  0.10932
842
 -0.24237841 -0.23455813]
```

**Problema 2: Integración**

In [12]:
```
import numpy as np

mu = 0
sigma = 9
n = 1000
y = []
x = np.random.normal(mu,sigma,n) #Generar n valores simulados  que  sig
uen  una  distribución  normal
for i in range(n-1):
    if  -1<x[i] and x[i]<1:
        y.append(x[i])  # Selecciona aquelloos entre -1 y 1
y.sort()  #Los ordena de menor a mayor

p = y
b = 0
c = 3
f = lambda t: (1/(c*sqrt(2*3.141592)))*exp(-(t-b)**2/(2*c**2))
nm = np.array([f(pi) for pi in p])
nm
```

Out[12]:
```
array([0.126030702486646, 0.126334127553166, 0.126678255126658,
       0.126679037520381, 0.127473172604629, 0.127690123513451,
       0.127733612033975, 0.127803322048374, 0.128124095600918,
       0.128367632858049, 0.128430341073680, 0.128578405475912,
       0.128670711678461, 0.129010775179137, 0.129031849158075,
       0.129714089203036, 0.129889375502795, 0.130359190425635,
       0.130467621260397, 0.130544087318443, 0.130636771551617,
       0.131133696697780, 0.131428445551457, 0.131466994898723,
       0.131532063738928, 0.131930404176520, 0.132010518776014,
       0.132369346625003, 0.132389661142455, 0.132397458380970,
       0.132675373314480, 0.132690429421528, 0.132699713769773,
       0.132700171673455, 0.132730153618805, 0.132913925711126,
       0.132915654239845, 0.132930480096543, 0.132964069637934,
       0.132968663000925, 0.132971037951989, 0.132979716516023,
       0.132979986966827, 0.132980567444159, 0.132976853888543,
       0.132963163166211, 0.132954967814144, 0.132943969378744,
       0.132934570440950, 0.132819971231442, 0.132781528692134,
       0.132635287832031, 0.132525550976804, 0.132509995189841,
       0.132435229353139, 0.132394798752935, 0.132259213349844,
       0.132147570690218, 0.132138248595665, 0.131974748156238,
       0.131944428669847, 0.131929115832210, 0.131890463968783,
       0.131786268561645, 0.131562494829214, 0.131365874402422,
       0.131201091338596, 0.131081981685710, 0.131073914083036,
       0.130832995823113, 0.130589203194296, 0.129998863858840,
       0.129637822043023, 0.129620117514832, 0.129613923491672,
       0.129464042613584, 0.129344392278667, 0.129018819053826,
       0.128500491895077, 0.128455327996141, 0.128147858261230,
       0.128102779694708, 0.127838050791671, 0.126873055916189,
       0.125951250933944], dtype=object)
```

In [13]:
```
# Método del Trapecio Compuesto
import numpy as np
a = -1
b = 1
n = len(nm)
h = (b - a) / (n -1)
x = np.linspace(a, b, n)
f = nm

I_trap= (h/2)*(f[0] +2*sum(f[1:n-1])+ f[n-1])

print(I_trap)
```

```
0.261577027429894
```

In [14]:
```
# Método de Simpson Compuesto:

import numpy as np
a =-1
b = 1
n =len(nm)
h = (b - a) / (n -1)
x = np.linspace(a, b, n)
f = nm

I_simp= (h/3)*(f[0] +2*sum(f[:n-2:2]) +4*sum(f[1:n-1:2]) + f[n-1])

print(I_simp)
```

```
0.263582972184626
```

***Medianto el uso de la función gaussiana:***

In [5]:
```
from sympy import *
from sympy.abc import  t
b=0
c=3
f=(1/(c*sqrt(2*3.141592)))*exp(-(t-b)**2/(2*c**2))  #Función gaussiana,
dada media=0 y varianza=9
f
```

Out[5]:
$$0.132980773966744e^{-\frac{t^2}{18}}$$

In [11]:
```
def trapecio_comp(func, a, b, h):

    def f(r): return func.subs('t', r).evalf()
    summ = sum([f(a+i*h) for i in range(int(b/h))])
    resultado = h / 2 * (f(a) + f(b)) + h*summ
    return resultado

a= -1
b= 1
h = (b-a)/100
tc= trapecio_comp(f, a, b, h)


print("Integral por Trapecio Compuesto =", tc)
```

```
Integral por Trapecio Compuesto = 0.133002232422777
```

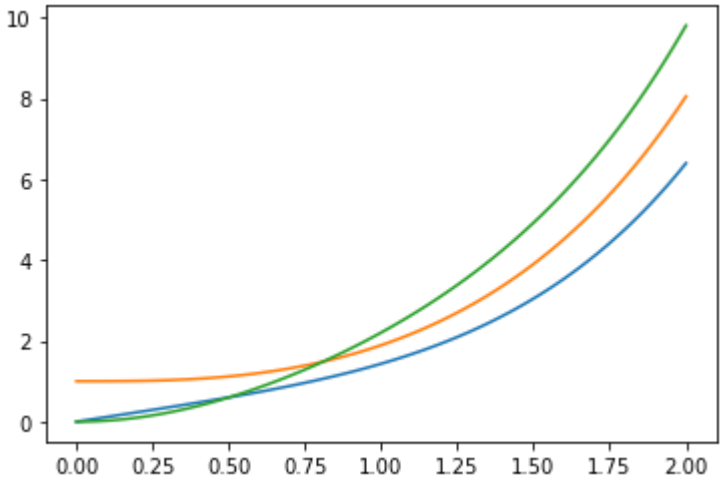
In [4]:
```
def simpson_comp(func, a, b, h):

    n = int(b//h)
    h = (b-a) / n
    def f(r): return func.subs('t', r).evalf()
    summ1 = sum([f(a+(2*i-2)*h) for i in range(2, int(n/2)+1)])
    summ2 = sum([f(a+(2*i-1)*h) for i in range(1, int(n/2)+1)])
    resultado = (h) / 3 * (f(a) + 2*summ1 + 4*summ2 + f(b))
    return resultado
a= -1
b= 1
h = (b-a)/94
sc= simpson_comp(f, a, b, h)

print("Integral por Simpson Compuesto =", sc)
```

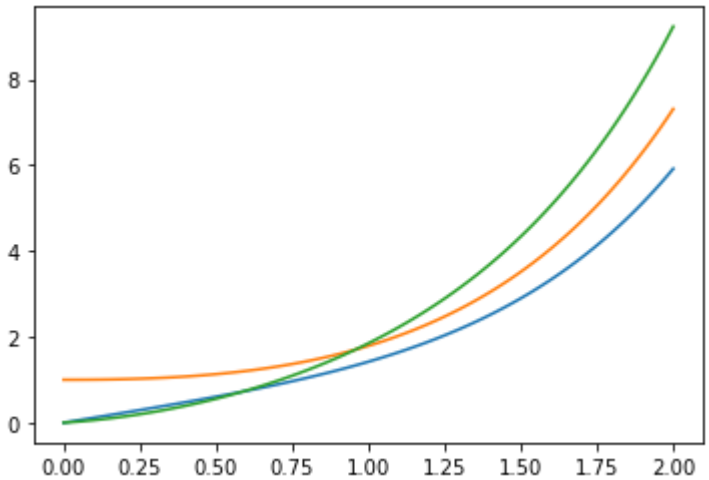
```
Integral por Simpson Compuesto = 0.255743622117717
```

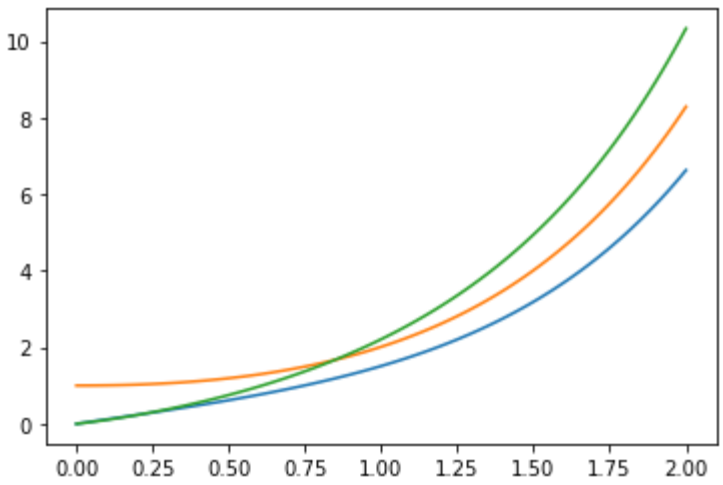
**Problema 3: Ecuaciones Diferenciales Ordinarias**

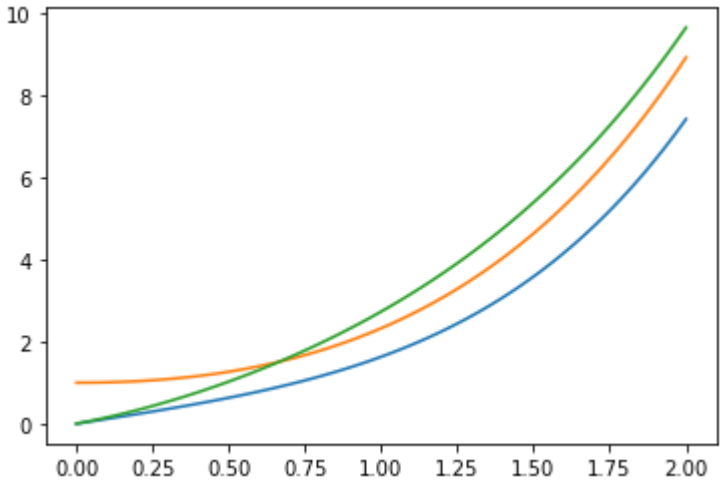
In [14]:
```
# Problema 3

import numpy as np
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

def runge_kutta_sistema(f, g, e, x0, y0, z0, a, b, h):
    t = np.arange(a, b + h, h)
    n = len(t)
    x = np.zeros(n)
    y = np.zeros(n)
    z = np.zeros(n)
    x[0] = x0
    y[0] = y0
    z[0] = z0
    for i in range(n - 1):
        k1 = h * f(x[i], y[i], z[i], t[i])
        l1 = h * g(x[i], y[i], z[i], t[i])
        m1 = h * e(x[i], y[i], z[i], t[i])

        k2 = h * f(x[i] + k1 / 2, y[i] + l1 / 2, z[i] + m1 / 2, t[i] +
h / 2)
        l2 = h * g(x[i] + k1 / 2, y[i] + l1 / 2, z[i] + m1 / 2, t[i] +
h / 2)
        m2 = h * e(x[i] + k1 / 2, y[i] + l1 / 2, z[i] + m1 / 2, t[i] +
h / 2)

        k3 = h * f(x[i] + k2 / 2, y[i] + l2 / 2, z[i] + m2 / 2, t[i] +
h / 2)
        l3 = h * g(x[i] + k2 / 2, y[i] + l2 / 2, z[i] + m2 / 2, t[i] +
h / 2)
        m3 = h * e(x[i] + k2 / 2, y[i] + l2 / 2, z[i] + m2 / 2, t[i] +
h / 2)

        k4 = h * f(x[i] + k3, y[i] + l3, z[i] + m3, t[i] + h)
        l4 = h * g(x[i] + k3, y[i] + l3, z[i] + m3, t[i] + h)
        m4 = h * e(x[i] + k3, y[i] + l3, z[i] + m3, t[i] + h)

        x[i + 1] = x[i] + (1 / 6) * (k1 + 2 * k2 + 2 * k3 + 2 * k4)
        y[i + 1] = y[i] + (1 / 6) * (l1 + 2 * l2 + 2 * l3 + 2 * l4)
        z[i + 1] = z[i] + (1 / 6) * (m1 + 2 * m2 + 2 * m3 + 2 * m4)
    plt.plot(t, x, t, y, t, z)
    plt.show()
```

In [23]:
```
a = 0
b = 2
x0 = 0
y0 = 1
z0 = 0
h = 0.02

# f = w'1 = y
# g = w'2 = z
# e = w'3 = -t*z +t*y + 2*x +t
f = lambda x,y,z,t: y ;
g = lambda x,y,z,t: z ;
e = lambda x,y,z,t: -t*z +t*y + 2*x +t ;
runge_kutta_sistema(f, g, e, x0, y0, z0, a, b, h)

# t = 0.2
e = lambda x,y,z,t: -0.2*z +0.2*y + 2*x +0.2
runge_kutta_sistema(f, g, e, x0, y0, z0, a, b, h)
# t = 0.4
e = lambda x,y,z,t: -0.4*z +0.4*y + 2*x +0.4
runge_kutta_sistema(f, g, e, x0, y0, z0, a, b, h)
# t = 0.6
e = lambda x,y,z,t: -0.6*t*z +0.6*y + 2*x +0.6
runge_kutta_sistema(f, g, e, x0, y0, z0, a, b, h)
```


10
8
6
4
2
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


8
6
4
2
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


10
8
6
4
2
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


10
8
6
4
2
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00