In [1]:
```
#Program-1(a): Rank of the matrix
import numpy as np
A=np.array([[0,2,3,4],[2,3,5,4],[4,8,13,12]])
print("A=",A)
rank=np.linalg.matrix_rank(A)
print("\n the rank of the given matrix=",rank)
```

```
A= [[ 0  2  3  4]
 [ 2  3  5  4]
 [ 4  8 13 12]]

 the rank of the given matrix= 2
```

In [2]:
```
# program-1(b): Rank of the matrix
import numpy as np
A=np.array([[1,2,4,3],[2,4,6,8],[4,8,12,16],[1,2,3,4]])
print("A=",A)
rank=np.linalg.matrix_rank(A)
print("\n the rank of the given matrix=",rank)
```

```
A= [[ 1  2  4  3]
 [ 2  4  6  8]
 [ 4  8 12 16]
 [ 1  2  3  4]]

 the rank of the given matrix= 2
```

In [3]:
```
#Program-2(a):Test the consistency of the given system of equations
#x+2y-z=1 ; 2x+y+4z=2 ; 3x+3y+4z=1
import numpy as np
A=np.matrix([[1,2,-1],[2,1,4],[3,3,4]])
B=np.matrix([[1],[2],[1]])
AB=np.concatenate((A,B),axis=1)
rA=np.linalg.matrix_rank(A)
rAB=np.linalg.matrix_rank(AB)
n=A.shape[1]
if(rA==rAB):
    print("The system of equations is consistent")
    if(rA==n):
        print("The system has unique solution")
        print(np.linalg.solve(A,B))
    else:
            print("The system has infinitely many solutions")
else:
    print("The system of equations is inconsistent")
```

```
The system of equations is consistent
The system has unique solution
[[ 7.]
 [-4.]
 [-2.]]
```

In [4]:
```
#Program-2(b):Test the consistency of the given system of equations
#5x+y+3z=20 ; 2x+5y+2z=18 ; 3x+2y+z=14
import numpy as np
A=np.matrix([[5,1,3],[2,5,2],[3,2,1]])
B=np.matrix([[20],[18],[14]])
AB=np.concatenate((A,B),axis=1)
rA=np.linalg.matrix_rank(A)
rAB=np.linalg.matrix_rank(AB)
n=A.shape[1]
if(rA==rAB):
    print("The system of equations is consistent")
    if(rA==n):
        print("The system has unique solution")
        print(np.linalg.solve(A,B))
    else:
            print("The system has infinitely many solutions")
else:
    print("The system of equations is inconsistent")
```

```
The system of equations is consistent
The system has unique solution
[[3.]
 [2.]
 [1.]]
```

In [5]:
```
#Program-3(a): Solution of system of linear equations by Guass Siedel Method
#20x+y-2z=12 ; 3x+20y-z=-18 ,;2x-3y+20z=25
f1 = lambda x,y,z :(17-y+2*z)/20
f2 = lambda x,y,z :(-18-3*x+z)/20
f3 = lambda x,y,z :(25-2*x+3*y)/20
x0=0
y0=0
z0=0
count=1
e=float(input('Enter tolerable error:'))
print('\ncount\tx\ty\tz\n')
condition = True
while condition:
    x1 = f1(x0, y0, z0)
    y1 = f2(x1, y0, z0)
    z1 = f3(x1, y1, z0)
    print('%d\t%0.4f\t%0.4f\t%0.4f\n'%(count,x1,y1,z1))
    e1 = abs(x0-x1);
    e2 = abs(y0-y1);
    e3 = abs(z0-z1);
    count+=1
    x0=x1
    y0=y1
    z0=z1
    condition = e1>e and e2>e and e3>e
print('\nsolution : x=%0.3f,y=%0.3f and z=%0.3f\n'%(x1,y1,z1,))
```

```
Enter tolerable error:0.01

count   x       y       z

1       0.8500  -1.0275 1.0109

2       1.0025  -0.9998 0.9998

3       1.0000  -1.0000 1.0000


solution : x=1.000,y=-1.000 and z=1.000
```

In [6]:
```
#Program-3(b): Solution of system of linear equations by Guass Siedel Method
#5x+2y+z=12 ; x+4y+2z=15 ; x+2y+5z=20
f1 = lambda x,y,z :(12-2*y-z)/5
f2 = lambda x,y,z :(15-x-z)/4
f3 = lambda x,y,z :(20-x-2*y)/5
x0=0
y0=0
z0=0
count=1
e=float(input('Enter tolerable error:'))
print('\ncount\tx\ty\tz\n')
condition = True
while condition:
    x1 = f1(x0, y0, z0)
    y1 = f2(x1, y0, z0)
    z1 = f3(x1, y1, z0)
    print('%d\t%0.4f\t%0.4f\t%0.4f\n'%(count,x1,y1,z1))
    e1 = abs(x0-x1);
    e2 = abs(y0-y1);
    e3 = abs(z0-z1);
    count+=1
    x0=x1
    y0=y1
    z0=z1
    condition = e1>e and e2>e and e3>e
print('\nsolution : x=%0.3f,y=%0.3f and z=%0.3f\n'%(x1,y1,z1,))
```

```
Enter tolerable error:0.01

count   x       y       z

1       2.4000  3.1500  2.2600

2       0.6880  3.0130  2.6572

3       0.6634  2.9199  2.6994

4       0.6922  2.9021  2.7007


solution : x=0.692,y=2.902 and z=2.701
```

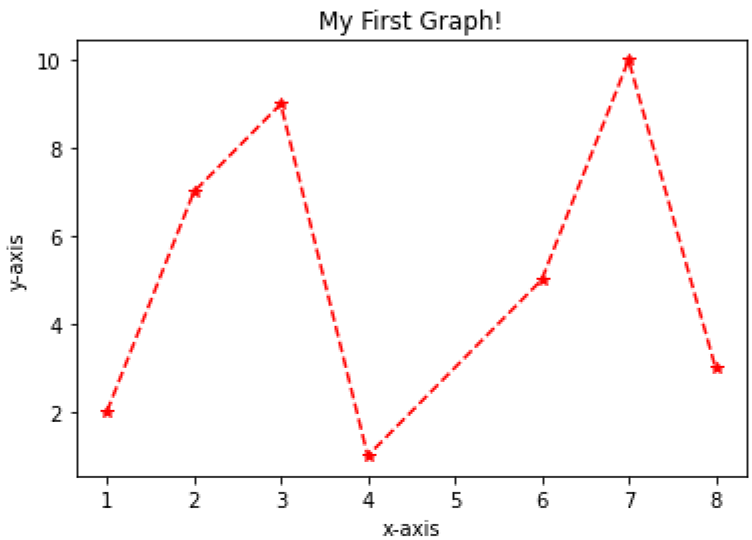
In [7]:
```
#Program-4(a):Finding the Largest eigen value and corresponding eigen vector by
#Given A=[[2,0,1],[0,2,0],[1,0,2]]
import numpy as np
x=np.array([1,0,0])
a=np.array([[2,0,1],[0,2,0],[1,0,2]])
def normalize(x):
    f=abs(x).max()
    xn=x/x.max()
    return f,xn
for i in range(40):
    x=np.dot(a,x)
    lambda1,x=normalize(x)
print('Eigenvalue:',lambda1)
print('Eigenvector:',x)
```

```
Eigenvalue: 3.0
Eigenvector: [1. 0. 1.]
```

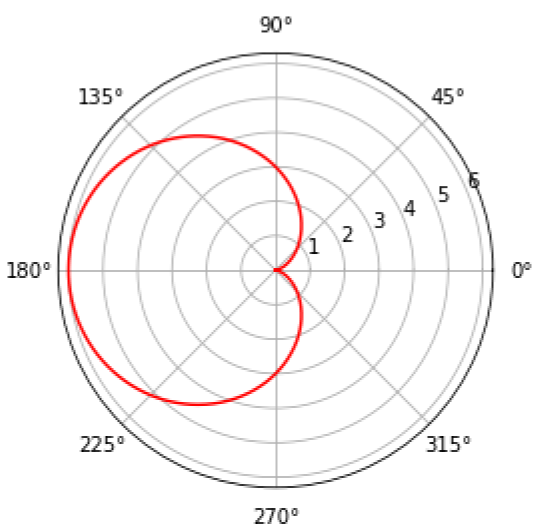
In [8]:
```
#Program-4(b):Finding the Largest eigen value and corresponding eigen vector by
#Given A=[[6,-2,2],[-2,3,-1],[2,-1,3]] , X=[1,1,1]
import numpy as np
x=np.array([1,1,1])
a=np.array([[6,-2,2],[-2,3,-1],[2,-1,3]])
def normalize(x):
    f=abs(x).max()
    xn=x/x.max()
    return f,xn
for i in range(100):
    x=np.dot(a,x)
    lambda1,x=normalize(x)
print('Eigenvalue:',lambda1)
print('Eigenvector:',x)
```

```
Eigenvalue: 8.0
Eigenvector: [ 1.  -0.5  0.5]
```

In [4]:
```
#Program-5(a): Python program for plotting a line
import numpy as np
import matplotlib.pyplot as plt
x= [1,2,3,4,6,7,8]
y= [2,7,9,1,5,10,3]
plt.plot(x,y,'r--*')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('My First Graph!')
plt.show()
```



In [10]:
```
#Program-5(b):python program to plot cardioid r=3(1+cos theta)
from pylab import*
theta=linspace(0,2*np.pi,1000)
r1=3-3*cos(theta)
polar(theta,r1,'r')
show()
```



In [11]:
```
#Program-6(a): Finding the angle between two polar curves r1=4*(1+cos(t)) and r
from sympy import *
r,t=symbols('r,t')
r1=4*(1+cos(t));
r2=5*(1-cos(t));
dr1=diff(r1,t)
dr2=diff(r2,t)
t1=r1/dr1
t2=r2/dr2
q=solve (r1-r2,t)
w1=t1.subs({t:float (q[1])})
w2=t2.subs({t:float (q[1])})
y1=atan(w1)
y2=atan(w2)
w=abs(y1-y2)
print('Angle between curves in radian is %0.3f'%(w))
```

```
Angle between curves in radian is 1.571
```

In [3]:
```
#Program-6(b): Finding the radius of curvature for r=asin(nt) at t=pi/2 and n=1
from sympy import*
t,r,a,n=symbols('t,r,a,n')
r=a*sin(n*t)
r1=Derivative(r,t).doit()
r2=Derivative(r1,t).doit()
rho=(r**2+r1**2)**1.5/(r**2+2*r1**2-r*r2)
rho1=rho.subs(t,pi/2)
rho1=rho1.subs(n,1)
print("The radius of curvature is")
display(simplify (rho1))
```

```
The radius of curvature is
```

$$\frac{\left(a^2\right)^{1.5}}{2a^2}$$

In [13]:
```
#Program-7(a): Finding partial derivatives and jacobian of functions
#Prove that if u=e^x(x*cos(y)-y*sin(y)) then uxx+uyy=0
from sympy import *
x,y=symbols('x,y')
u=exp(x)*(x*cos(y)-y*sin(y))
display(u)
dux=diff(u,x)
duy=diff(u,y)
uxx=diff(dux,x)
uyy=diff(duy,y)
w=uxx+uyy
w1=simplify(w)
print('Ans:',float(w1))
```

$$\left(x\cos\left(y\right) - y\sin\left(y\right)\right)e^{x}$$

```
Ans: 0.0
```

In [14]:
```
#Program-7(b):If u=xy/z ,v=yz/x ,w=zx/y then find the jacobian of u,v,w w.r.t x
from sympy import *
x,y,z=symbols('x,y,z')
u=x*y/z
v=y*z/x
w=z*x/y
dux=diff(u,x)
duy=diff(u,y)
duz=diff(u,z)
dvx=diff(v,x)
dvy=diff(v,y)
dvz=diff(v,z)
dwx=diff(w,x)
dwy=diff(w,y)
dwz=diff(w,z)
J=Matrix([[dux,duy,duz],[dvx,dvy,dvz],[dwx,dwy,dwz]])
print("The jacobian matrix is")
display(J)
Jac=det(J).doit()
print('J=',Jac)
```

The jacobian matrix is

$$\begin{bmatrix} \frac{y}{z} & \frac{x}{z} & -\frac{xy}{z^2} \\ -\frac{yz}{x^2} & \frac{z}{x} & \frac{y}{x} \\ \frac{z}{y} & -\frac{xz}{y^2} & \frac{x}{y} \end{bmatrix}$$

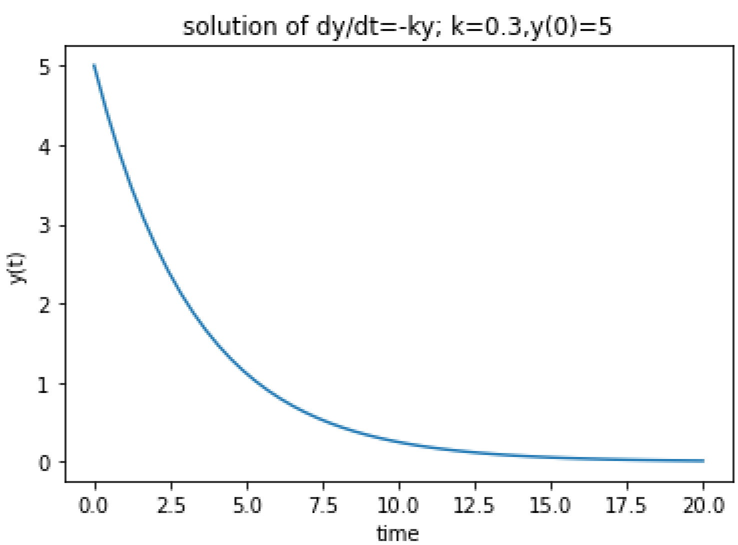
J= 4

In [15]:
```
#Program-8: Find the maxima and minima of f(x,y)=x^2+y^2+3x-3y+4
import sympy
from sympy import symbols,solve,Derivative,pprint
x,y=symbols('x,y')
f=x**2+y**2+3*x+3*y+4
d1=Derivative(f,x).doit()
d2=Derivative(f,y).doit()
criticalpoints1=solve(d1)
criticalpoints2=solve(d2)
s1=Derivative(f,x,2).doit()
s2=Derivative(f,y,2).doit()
s3=Derivative(Derivative(f,y),x).doit()
print('function value is')
q1=s1.subs({y:criticalpoints1,x:criticalpoints2}).evalf()
q2=s2.subs({y:criticalpoints1,x:criticalpoints2}).evalf()
q3=s3.subs({y:criticalpoints1,x:criticalpoints2}).evalf()
delta=s1*s2-s3**2
print(delta,q1)
if(delta>0 and s1<0):
    print("f takes maximum")
elif(delta>0 and s1>0):
    print("f takes minimum")
if (delta<0):
    print("The point is a saddle point")
if(delta==0):
    print("Further tests required")
```

```
function value is
4 2.00000000000000
f takes minimum
```

In [16]:
```
#Program-9:Solution of first order differential equation
#Solve dy/dt=-ky with parameter k=0.3 and y(0)=5
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
#Function returns dy/dt
def model(y,t):
    k=0.3
#dy/dt=-ky
    return -k*y
#intial condition
y0=5
#values for time
t=np.linspace(0,20)
#solve ODE
y=odeint(model,y0,t)

plt.plot(t,y)
plt.title('solution of dy/dt=-ky; k=0.3,y(0)=5')
plt.xlabel('time')
plt.ylabel('y(t)')
plt.show()
```



In [ ]: