Topic 1

December 30, 2021

1 Topic 1 - Some basic operations in Python for scientific computing

1.0.1 1. Program to accept two numbers from the user to add, subtract, multiply and divide it and print the results.

[59]:
```
n1=int(input('Enter a number: '))
n2=int(input('Enter a number: '))
a=n1+n2;
s=n1-n2;
p=n1*n2;
d=n1/n2;
print('The sum of the two numbers = ',a)
print('The difference of the two numbers = ',s)
print('The product of the two numbers = ',p)
print('The quotient of the two numbers = ',d)
```

```
Enter a number: 5
Enter a number: 8
The sum of the two numbers =  13
The difference of the two numbers =  -3
The product of the two numbers =  40
The quotient of the two numbers =  0.625
```

1.0.2 2. Program to check if a number is even or odd.

[60]:
```
a= int(input('Enter a number: '))
if(a%2==0):
    print(a,' is an even number')
else:
    print(a,' is an odd number')
```

```
Enter a number: 674
674  is an even number
```

1.0.3 3. Program to check whether a given integer is positive or negative.

[96]:
```
b= int(input('Enter an integer: '))
if(b>0):
    print(b,' is positive')
elif(b<0):
    print(b,' is negative')
else:
    print("zero")
```

```
Enter an integer: 7
7  is positive
```

1.0.4 4. Program to check the biggest of three numbers entered by the user.

[62]:
```
a1=int(input('Enter a number: '))
a2=int(input('Enter a number: '))
a3=int(input('Enter a number: '))
if(a1>=a2) and (a1>=a3):
    largest=a1
elif(a2>=a1) and (a2>=a3):
    largest=a2
else:
    largest=a3

print("")
print('The largest of the three numbers is',largest)
```

```
Enter a number: 5
Enter a number: 8
Enter a number: 3

The largest of the three numbers is 8
```

1.0.5 5. Write a program to compute the factorial of a number.

[63]:
```
fact=1;
f=int(input('Enter the number: '))
while(f>0):
    fact=fact*f
    f=f-1
print('The factorial is ',fact)
```

```
Enter the number: 7
The factorial is  5040
```

1.0.6 6. Program to generate the multiplication table of a number (from 1 to 10)

[64]:
```
x=int(input('Enter the number: '))
for i in range(1,11):
    print(x,' x ',i,' = ',x*i)
```

```
Enter the number: 7
7  x  1  =  7
7  x  2  =  14
7  x  3  =  21
7  x  4  =  28
7  x  5  =  35
7  x  6  =  42
7  x  7  =  49
7  x  8  =  56
7  x  9  =  63
7  x  10  =  70
```

1.0.7 7. Lists

[65]:
```
list=[]  #empty list

#Adding numbers to the list
nm=int(input('How many elements are to be added to the list: '))
while(nm>0):
    m=int(input('Enter the element to be added: '))
    list.append(m)
    nm=nm-1
print("")
print('Updated list: ')
print(list)
```

```
How many numbers are to be added to the list: 5
Enter the element to be added: 6
Enter the element to be added: 8
Enter the element to be added: 3
Enter the element to be added: 5
Enter the element to be added: 9

Updated list:
[6, 8, 3, 5, 9]
```

[66]:
```
print('First element in the list: ', list[0])    #accessing an element from the␣
 ↪list using index number
print('Third element in the list: ', list[2])
```

```
First element in the list:  6
Third element in the list:  3
```

[67]:
```
list.remove(8)    #removing an element from the list
print('Updated list: ',list)
```

```
Updated list:  [6, 3, 5, 9]
```

1.0.8 Arrays

[93]:
```
from numpy import *
ar=array([[1,2,3],[4,5,6],[7,8,9]])
print(ar)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

[69]:
```
ab=array([[1,2],[3,4]],dtype=float)
print(ab)
```

```
[[1. 2.]
 [3. 4.]]
```

[70]:
```
print(zeros(5))
```

```
[0. 0. 0. 0. 0.]
```

[71]:
```
print(ones(5))
```

```
[1. 1. 1. 1. 1.]
```

[72]:
```
print(identity(5))
```

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

[73]:
```
# arange (from,to,stepsize)
# linspace(from,to,no.of points)

print(arange(2,20,3))
print("")
print(linspace(2,10,5))
```

```
[ 2  5  8 11 14 17]

[ 2.  4.  6.  8. 10.]
```

1.0.9 Matrices

[74]:
```
matrix1=array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix1)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

[94]:
```
matrix1[1]
```

[94]:
```
array([4, 5, 6])
```

[98]:
```
matrix1[:,0]  #printing first column
```

[98]:
```
array([1, 4, 7])
```

[77]:
```
matrix1[1,2]   #accessing an element
```

[77]:
```
6
```

[95]:
```
print(sin(matrix1))
```

```
[[ 0.84147098  0.90929743  0.14112001]
 [-0.7568025  -0.95892427 -0.2794155 ]
 [ 0.6569866   0.98935825  0.41211849]]
```

[79]:
```
# Adding - Subtracting - Multiplying - Dividing matrices (element wise)

m1=array([[1,1,3],[2,3,5],[3,4,2]])
m2=array([[2,8,4],[7,1,3],[0,7,3]])
print('The two matrices are: ')
print("")
print(m1,' and ')
print("")
print(m2)
```

```
The two matrices are:

[[1 1 3]
 [2 3 5]
 [3 4 2]]  and

[[2 8 4]
 [7 1 3]
 [0 7 3]]
```

[80]:
```
print(m1+m2)   #Adding
```

```
[[ 3  9  7]
 [ 9  4  8]
 [ 3 11  5]]
```

[81]:
```
print(m1-m2)   #difference
```

```
[[-1 -7 -1]
 [-5  2  2]
 [ 3 -3 -1]]
```

[82]:
```
print(m1*m2)   #product
```

```
[[ 2  8 12]
 [14  3 15]
 [ 0 28  6]]
```

[83]:
```
print(m2/m1)   #quotient
```

```
[[2.         8.         1.33333333]
 [3.5        0.33333333 0.6       ]
 [0.         1.75       1.5       ]]
```

1.0.10 Solving a system of linear equations.

The system of linear equations can be solved using the function *numpy.linalg.solve*().

1.0.11 1. Solve the system of equations:

1.0.12 $4x + 3y = 20$ and $-5x + 9y = 26$

[2]:
```
from numpy import *
A=matrix([[4,3],[-5,9]])
B=matrix([[20],[26]])
print(A)
print('=')
print(B)
```

```
[[ 4  3]
 [-5  9]]
=
[[20]
 [26]]
```

[2]:
```
print(linalg.solve(A,B))
```

```
[[2.]
 [4.]]
```

1.0.13 2. Solve the system of equations:

1.0.14 $2x - 3y + 5z = 11$

1.0.15 $5x + 2y - 7z = -12$

1.0.16 $-4x + 3y + z = 5$

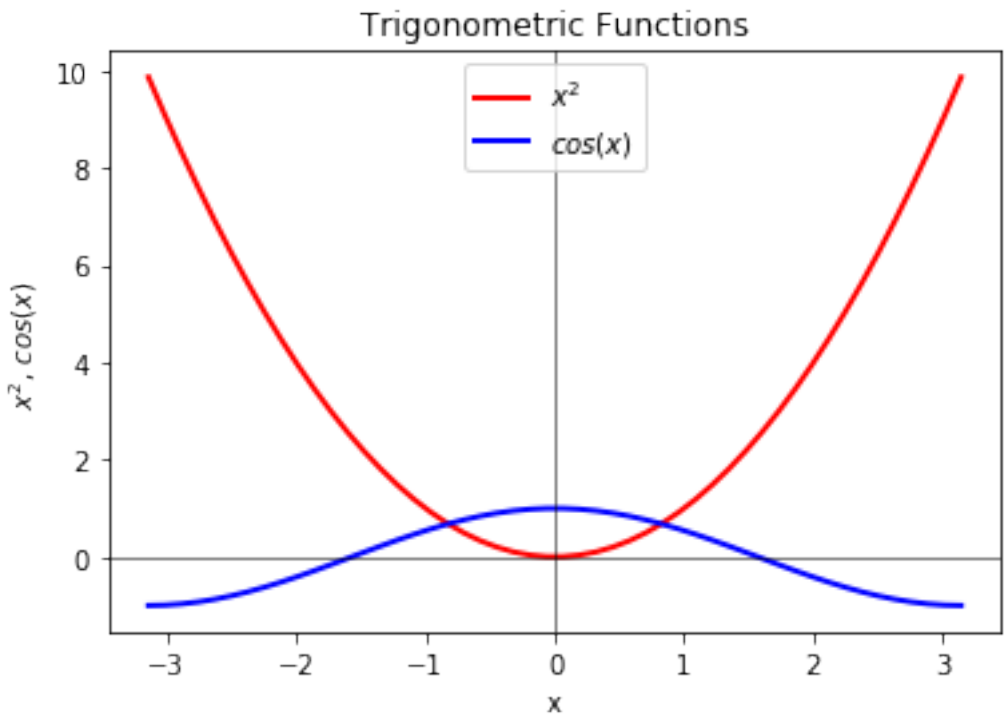
[68]:
```
A1=matrix([[2,-3,5],[5,2,-7],[-4,3,1]])
B1=matrix([[11],[-12],[5]])
print(linalg.solve(A1,B1))
```

```
[[1.]
 [2.]
 [3.]]
```

1.0.17 Scalar and Vector Plotting

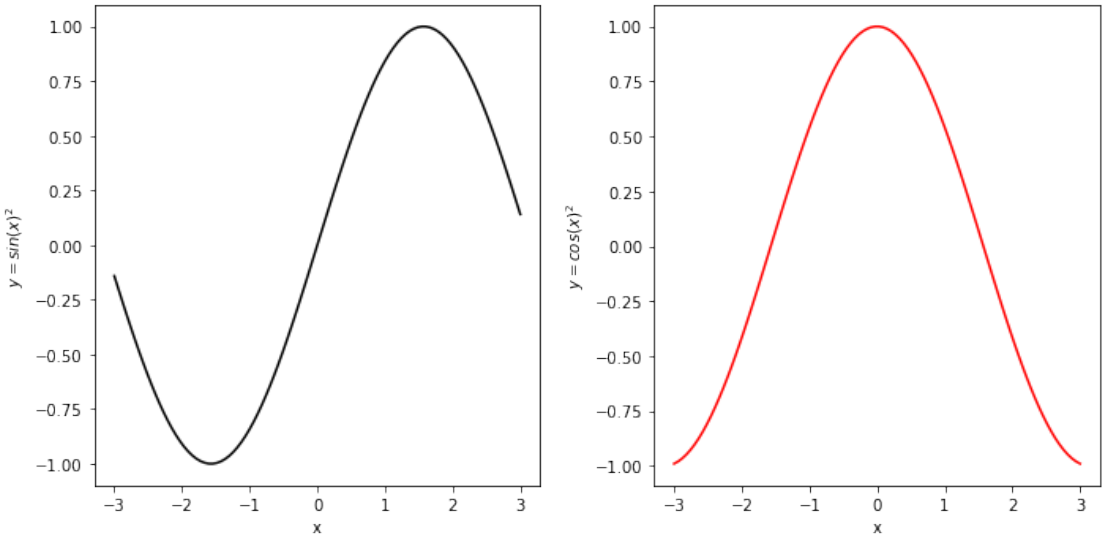
Q1. Plot the graphs of the functions x^2 and $Cos(x)$ in a single plot.

[2]:
```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-np.pi, np.pi)
Y1 = X**2
Y2 = np.cos(X)


plt.plot(X, Y1, lw=2, color='red',label='$x^2$')
plt.plot(X, Y2, lw=2, color='blue',label='$cos(x)$')


plt.title('Trigonometric Functions')
plt.xlabel('x')
plt.ylabel('$x^2$, $cos(x)$')
plt.axhline(0, lw=0.5, color='black')
plt.axvline(0, lw=0.5, color='black')
plt.legend()
None
```



Q2. Plot the graphs of the trigonometric functions $sin(x)$ and $Cos(x)$ in as subplots.

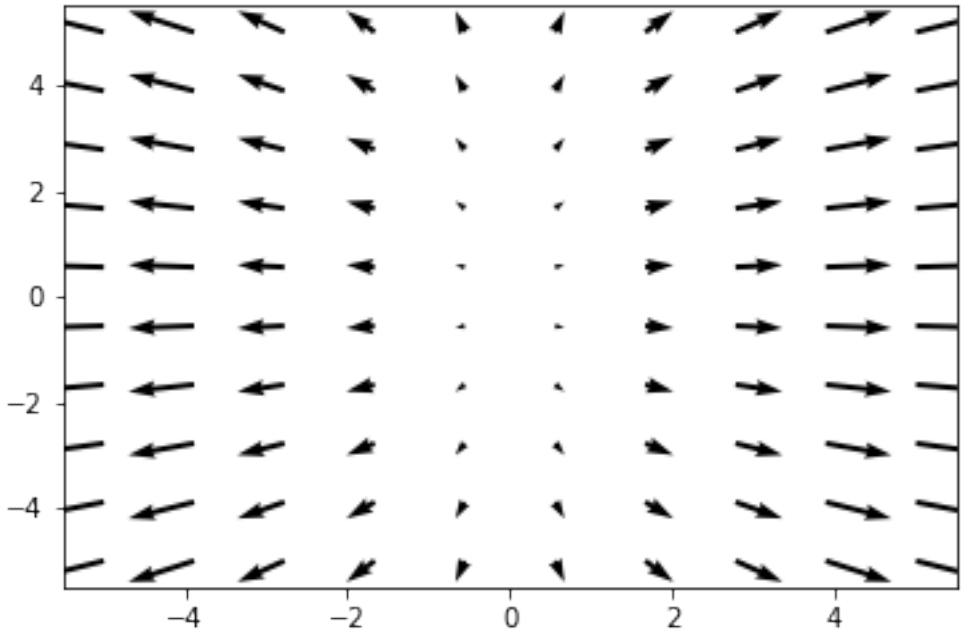
[46]:
```
plt.figure(figsize=(10,5))
xvalues = linspace(-3, 3, 100)

plt.subplot(1, 2, 1)
yvalues = sin(xvalues)
plt.plot(xvalues, yvalues, color='black')
plt.xlabel('x')
plt.ylabel('$y=sin(x)^2$')

plt.subplot(1, 2, 2)
yvalues = cos(xvalues)
plt.plot(xvalues, yvalues, color='red')
plt.xlabel('x')
plt.ylabel('$y=cos(x)^2$')
plt.tight_layout()
```



1.0.18 Vector Plot

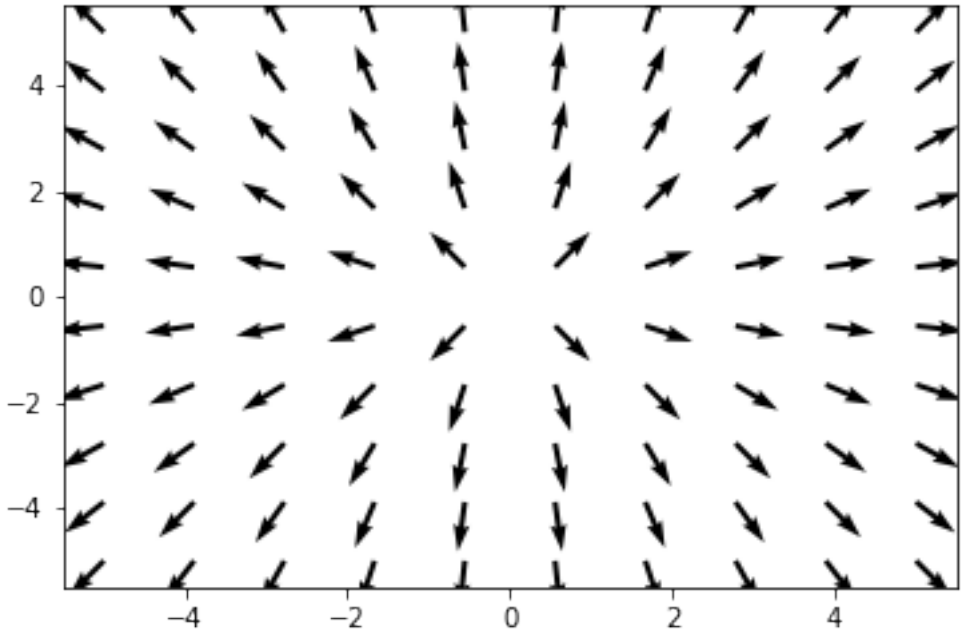
Q1. Plot the vector field for $f(x, y) = 2xi + \frac{y}{2}j$

[44]:
```
# Vector Plot of f(x,y)=2xi+(y/2)j
import numpy as np
import matplotlib.pyplot as plt
x,y=np.meshgrid(np.linspace(-5,5,10),np.linspace(-5,5,10))
u=2*x
v=y/2
plt.quiver(x,y,u,v)
plt.show()
```



Q2. Plot the vector field for $f(x,y) = \frac{x}{\sqrt{x^2+y^2}}i + \frac{y}{\sqrt{x^2+y^2}}j$

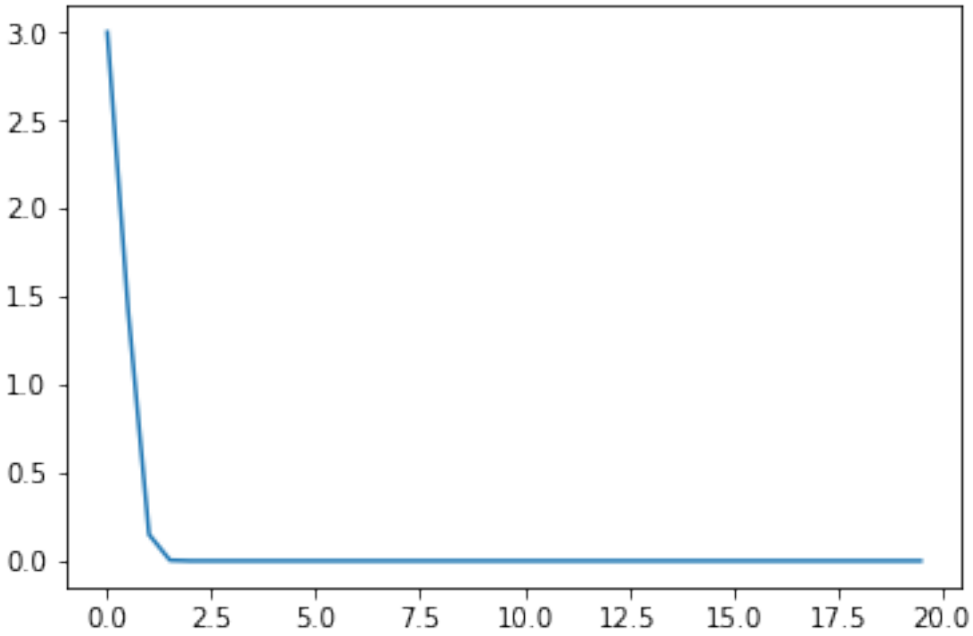
[14]:
```
import numpy as np
import matplotlib.pyplot as plt
x,y=np.meshgrid(np.linspace(-5,5,10),np.linspace(-5,5,10))
u=x/np.sqrt(x**2+y**2)
v=y/np.sqrt(x**2+y**2)
plt.quiver(x,y,u,v)
plt.show()
```



1.0.19 Solution of initial value problems

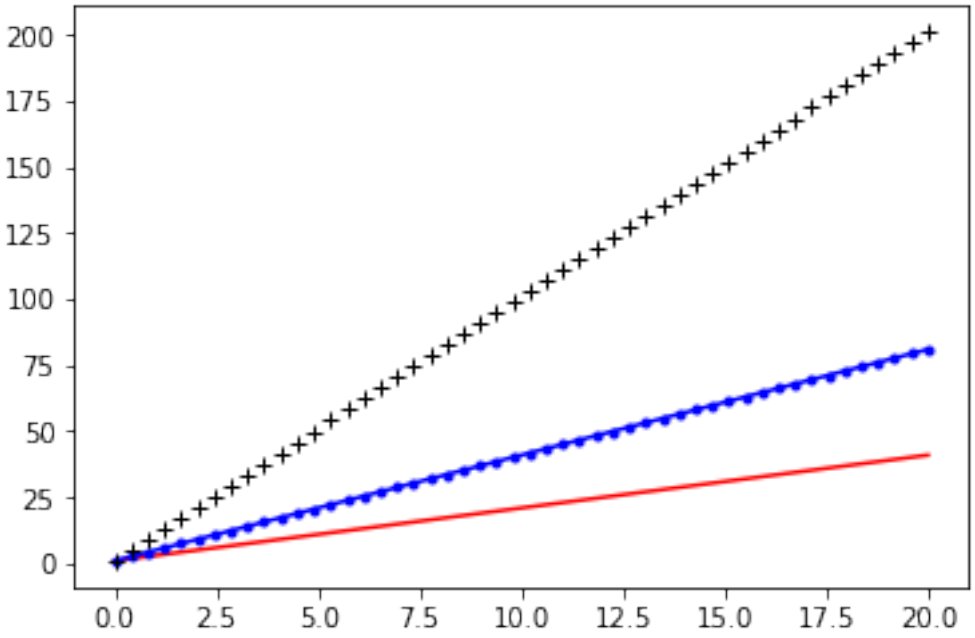
1. Solve the IVP $\frac{dy}{dx} = -0.3y, y(0) = 5$

[64]:
```
from numpy import *
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def model(y,x):        # define a function that returns dy/dx
    dydx=-0.3*y
    return dydx

y0=5                  # intial condition
x=arange(0,20,0.5)    # timepoints

y1=odeint(model,y0,x) # solve ODE

plt.plot(x,y1)
plt.show()
```



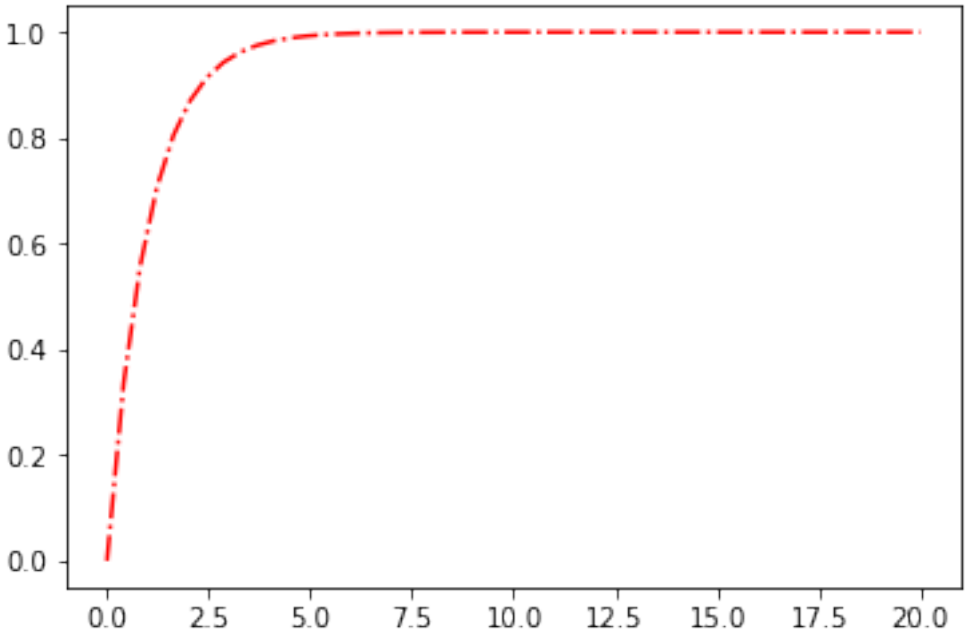
2. Solve the IVP $\frac{dy}{dx} = 2k, y(0) = 1$

[57]:
```
def model(y,t,k):        # define a function that returns dy/dx
    dydx=2*k
    return dydx
y0=1                  # intial condition
t=linspace(0,20,50)    # timepoints
# solve ODE
k=1
yk1=odeint(model,y0,t,args=(k,))
k=2
yk2=odeint(model,y0,t,args=(k,))
k=5
yk3=odeint(model,y0,t,args=(k,))
plt.plot(t,yk1,'r')
plt.plot(t,yk2,'b.-')
plt.plot(t,yk3,'k+')
plt.show()
```



3. Solve the IVP $\frac{dy}{dx} = 1 - y, y(0) = 0$

[27]:
```
from numpy import *
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def model(y,x):        # define a function that returns dy/dx
    dydx=1-y
    return dydx

y0=0                  # intial condition
x=linspace(0,20)  # timepoints

yx=odeint(model,y0,x) # solve ODE

plt.plot(x,yx,'r-.')
plt.show()
```



1.0.20 Solving higher order differential equations

To solve higher order differential equations, we have to convert it to a system of first order differential equations using the following method and then solve by using *odeint*. **Example**: Solve $y''+2y'+y = 0$; ——————————— (1) with intial conditions $[y(0) = 5, y'(0) = 3]$

Here, take $y' = t$ ——————————— (2) Substitute (2) in (1), we get $t' + 2t + y = 0$ i.e. $t' = -2t - y$ ———————————————-(3)

Equations (2) and (3) constitute the system of first order differential equations.

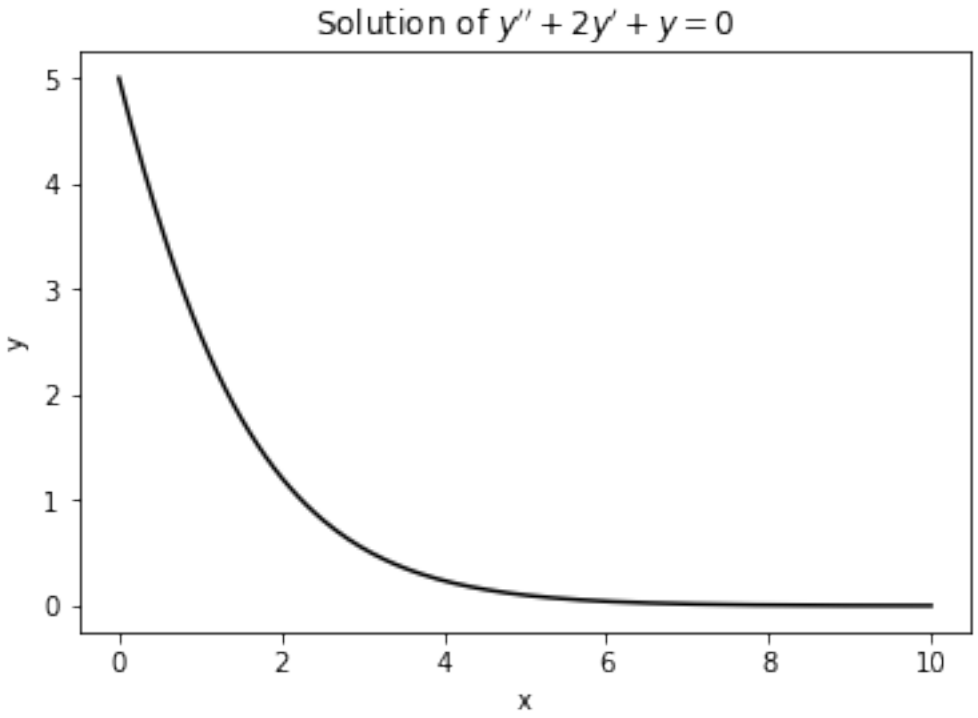
Now, let u be a vector, $u = [y, t]$. This implies $u[0] = y$ and $u[1] = t$. Then, (2) and (3) can be written as: $y' = u[1]$ $t' = -2u[1] - u[0]$ ———————————————————————

Step 1 : Define a function which returns the system of differential equations. **Step 2** : Assign the initial conditions and define a range for independent variable x. **Step 3** : Using in built function *odeint* solve the D.Eqns. Syntax - $us = odeint(func, IC, x)$ **Step 4** : The solution us will be an array with 2 columns. First column for $u[0] = y$ and second column for $u[1] = t$. (With number of rows depending on the number of values defined for x). Therefore, $Sol = us[:, 0]$ will extract the values obtained for y from us. **Step 5** : Plot the solution for Sol.

1. Solve the differential equation $y'' + 2y' + y = 0$ with the initial conditions, $y(0) = 5, y'(0) = -3$.

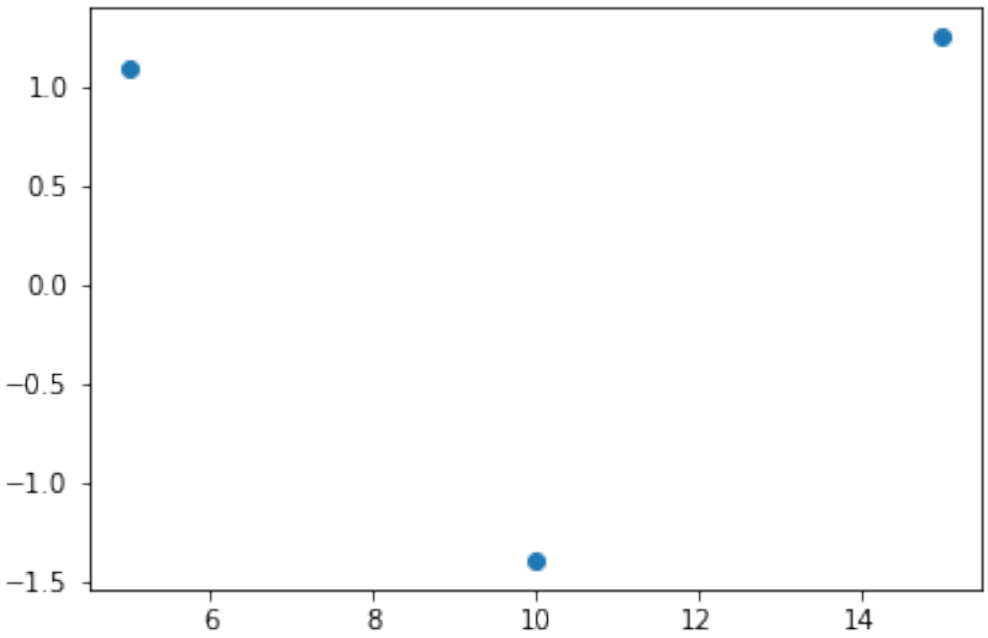
[2]:
```
from numpy import *
from scipy.integrate import odeint
import matplotlib.pyplot as plt
def du_dx(u,x):
```

```
    return [u[1],-2*u[1]-u[0]]
u0=[5,-3]
xs=linspace(0,10,200)
us=odeint(du_dx,u0,xs)
ys=us[:,0]
plt.plot(xs,ys,color="black")
plt.title("Solution of $y''+2y'+y=0$")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



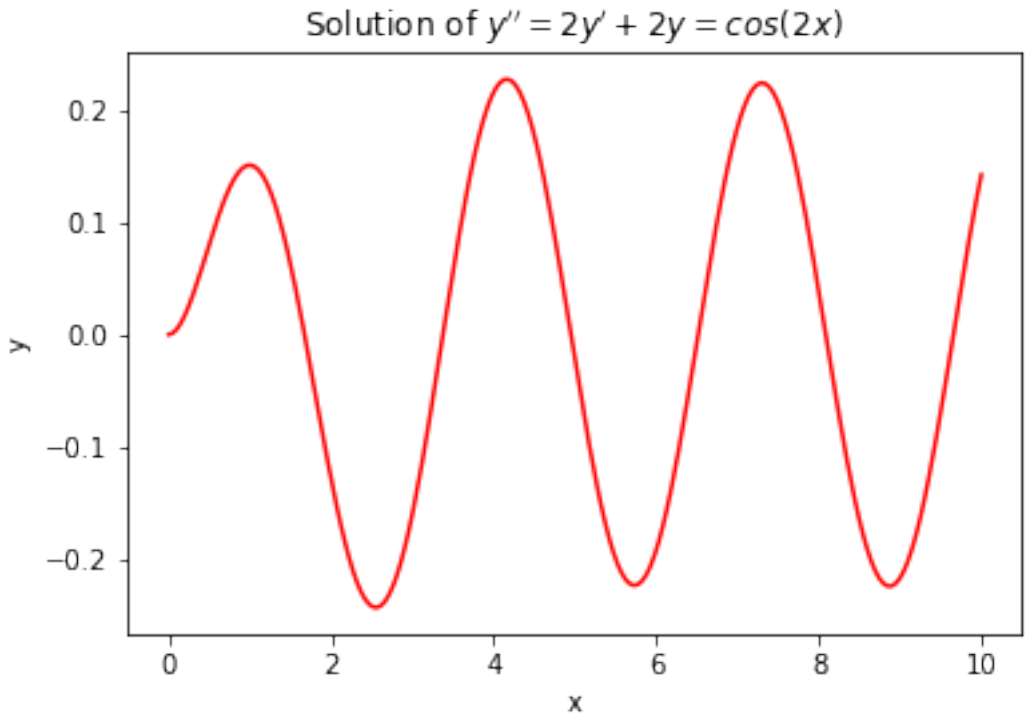
[4]:
```
print(us[5,0],us[10,0],us[15,0])
x=[5,10,15]; y=[1.088,-1.391,1.249]
import matplotlib.pyplot as plt
plt.scatter(x, y)
plt.show()
```

```
4.279980910683981 3.6330919852019443 3.0623797122019942
```



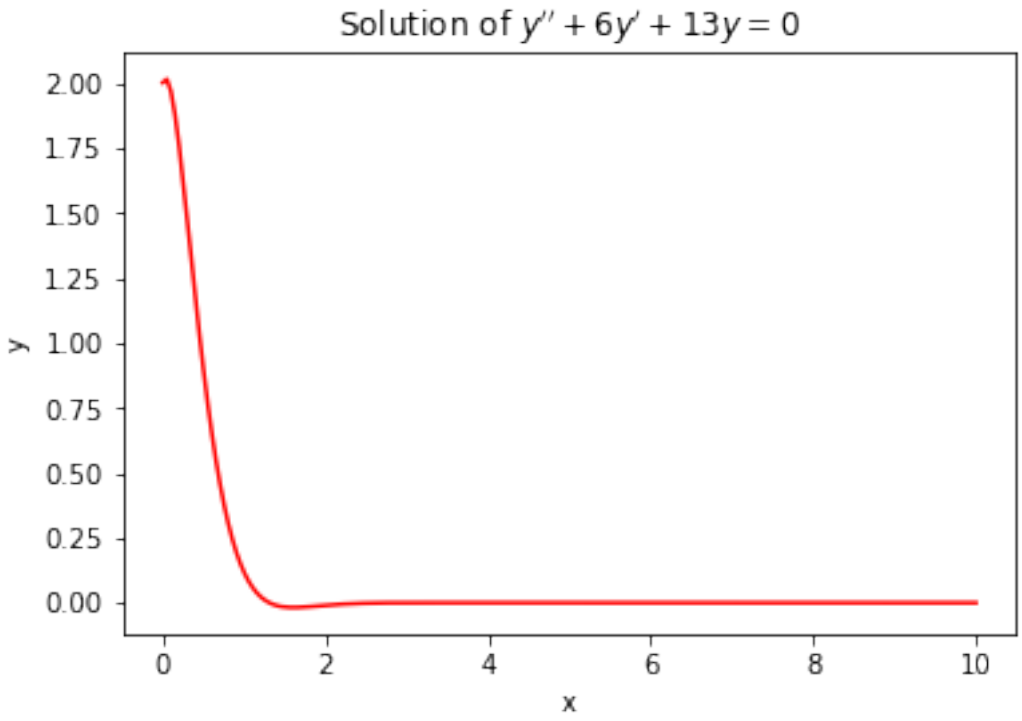
2. Solve the differential equation $y'' + 2y' + 2y = cos(2x)$ with the initial conditions, $y(0) = 0, y'(0) = 0$.

[29]:
```
from numpy import *
from scipy.integrate import odeint
import matplotlib.pyplot as plt
def du_dx(u,x):
    return [u[1],-2*u[1]-2*u[0]+cos(2*x)]
u0=[0,0]
xs=linspace(0,10,200)
us=odeint(du_dx,u0,xs)
ys=us[:,0]
plt.plot(xs,ys,color="red")
plt.title("Solution of $y''=2y'+2y=cos(2x)$")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



3. Solve the differential equation $y'' + 6y' + 13y = 0$ with the initial conditions, $y(0) = 2, y'(0) = 1$.

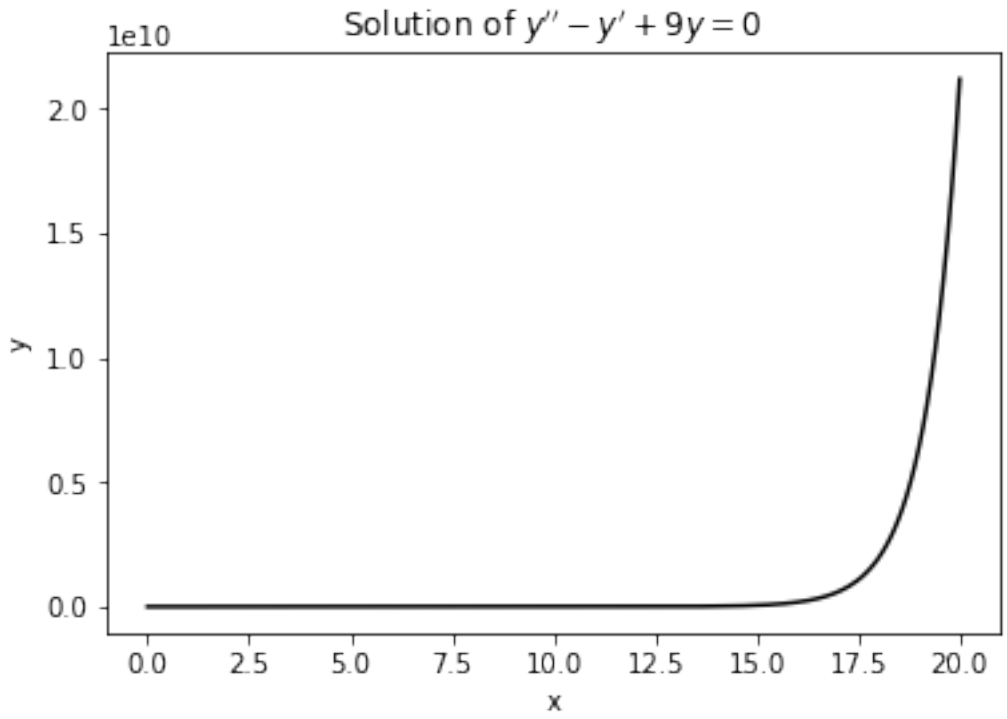
[90]:
```
from numpy import *
from scipy.integrate import odeint
import matplotlib.pyplot as plt
def du_dx(u,x):
    return [u[1],-6*u[1]-13*u[0]]
u0=[2,1]
xs=linspace(0,10,200)
us=odeint(du_dx,u0,xs)
ys=us[:,0]
plt.plot(xs,ys,color="red")
plt.title("Solution of $y''+6y'+13y=0$")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



4. Solve the differential equation $y'' + 3y' - 5y = 0$ with the initial conditions, $y(0) = 0, y'(0) = 5$.

[19]:
```
from numpy import *
from scipy.integrate import odeint
import matplotlib.pyplot as plt
def du_dx(u,x): # defining a function to return the differential equation
    return [u[1],-3*u[1]+5*u[0]]
u0=[0,5]  # initial conditions
xs=linspace(0,20,200)  #timepoints (x-axis)
us=odeint(du_dx,u0,xs) #solving the d.e to obtain solution array 'us'
# Array 'us' has two columns. The first column gives solution 'y'.
# Each row (r) contains value of y for a timepoint, t=r.
ys=us[:,0] #Extracting the first column (Solution set - y)
plt.plot(xs,ys,color="black") # Plotting the solution-set(yaxis) against␣
 ↪timepoints(xaxis)
plt.title("Solution of $y''-y'+9y=0$")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
#############
print(us[5,0],us[10,0],us[15,0])
```

```
# To extract the solution y for a timepoint t=5(say), then you may use the code␣
↪us[5,0].
```




```
1.5776655470660519 3.0645281101696122 5.603295179886854
```

```
[ ]:
```