In [15]:
```
from sympy import*
x=Symbol('x')
y=input('Enter  the function')
f=lambdify(x,y)
a=float(input('Enter a value: '))
b=float(input('enter b value: '))
N=int(input('Enter the number of iterations: '))
for i in range(1,N+1):
    c=(a*f(b)-b*f(a))/(f(b)-f(a))
    if (f(a)*f(c)<0):
        b=c
    else:
        a=c
    print('iteration %d \t the root %0.3f \t function value %0.3f \n',(i,c,f(c)))
```

```
Enter  the functionx**3-9*x+1
Enter a value: 2
enter b value: 3
Enter the number of iterations: 5
iteration %d     the root %0.3f          function value %0.3f
 (1, 2.9, -0.7109999999999985)
iteration %d     the root %0.3f          function value %0.3f
 (2, 2.941554646405611, -0.02147327159597978)
iteration %d     the root %0.3f          function value %0.3f
 (3, 2.942783276646023, -0.0006245529337078892)
iteration %d     the root %0.3f          function value %0.3f
 (4, 2.9428189892140617, -1.8145117234524832e-05)
iteration %d     the root %0.3f          function value %0.3f
 (5, 2.94282002675138, -5.271525829186885e-07)
```

In [13]:
```
#Regula falsi method while loop
from sympy import*
x=Symbol('x')
y=input('Enter the function: ')
f=lambdify(x,y)
a=float(input('enter a value: '))
b=float(input('enter b value: '))
N=int(input('Enter tolarance: '))
x=a;
c=b;
i=0;
while(abs(x-c)>=N):
    x=c
    c=(a*f(b)-b*f(a))/(f(b)-f(a))
    if((f(a)*f(c)<0)):
        b=c
    else:
        a=c
        i=i+1
    print('iteration %d\t the root %0.3f\t function value %0.3f\n' %(i,c,f(c)));
    print('final value of root is %0.5f' %c)
```

```
Enter the function: x**3-9*x+1
enter a value: 2
enter b value: 3
Enter tolarance: 15
```

In [4]:
```
#Regula falsi method while loop
from sympy import*
x=Symbol('x')
y=input('Enter the function: ')
f=lambdify(x,y)
```

```
a=float(input('enter a value: '))
b=float(input('enter b value: '))
N=int(input('Enter tolarance: '))
x=a;
c=b;
i=0;
while(abs(x-c)>=N):
    x=c
    c=(a*f(b)-b*f(a))/(f(b)-f(a))
    if((f(a)*f(c)<0)):
        b=c
    else:
        a=c
        i=i+1
    print('iteration %d\t the root %0.3f\t function value %0.3f\n' %(i,c,f(c)));
    print('final value of root is %0.5f' %c)
```

```
Enter the function: x*e**x-2
enter a value: 0
enter b value: 1
Enter tolarance: 6
```

In [3]:
```
def regula_falsi(f,a,b,tol=6,maxiter=100):
    fa,fb=f(a),f(b)
    if fa*fb>0:
        raise ValueError("Function has same sign at both ends of interval ")
    for i in range(maxiter):
        c=(a*f(b)-b*f(a))/(f(b)-f(a))
        fc=f(c)
        if abs(fc)<tol:
            return c
        elif fc*fa<0:
            b,fb=c,fc
        else:
            a,fa=c,fc
    raise ValueError("Failed to converge")
    f=lambda x: x**3-x*2-1
    df=lambda x: 3*x**2-2*x
    a,b=3,2
    print("Using regula falsi method: x=",regula_falsi(f,a,b))
```

In [4]:
```
from sympy import*
x=Symbol('x')
g=input('Enter the function: ')
f=lambdify(x,g)
dg=diff(g);
df=lambdify(x,dg)
x0=float(input('enter the initial approximation: '));
n=int(input('Enetr the number of iterations: '));
for i in range(1,n+1):
    x1=(x0-(f(x0)/df(x0)))
    print('iteration %d \t the root %0.3f \t function value %0.3f \n'%(i,x1,f(x1))
    x0=x1
```

```
Enter the function: 3*x-cos(x)-1
enter the initial approximation: 1
Enetr the number of iterations: 10
iteration 1      the root 0.620          function value 0.046

iteration 2      the root 0.607          function value 0.000

iteration 3      the root 0.607          function value 0.000

iteration 4      the root 0.607          function value 0.000

iteration 5      the root 0.607          function value 0.000

iteration 6      the root 0.607          function value 0.000

iteration 7      the root 0.607          function value 0.000

iteration 8      the root 0.607          function value 0.000

iteration 9      the root 0.607          function value 0.000

iteration 10     the root 0.607          function value 0.000
```

In [5]:

```
from sympy import*
x=Symbol('x')
g=input('Enter the function: ')
f=lambdify(x,g)
dg=diff(g);
df=lambdify(x,dg)
x0=float(input('enter the initial approximation: '));
n=int(input('Enetr the number of iterations: '));
for i in range(1,n+1):
    x1=(x0-(f(x0)/df(x0)))
    print('iteration %d \t the root %0.3f \t function value %0.3f \n'%(i,x1,f(x1))
    x0=x1
```

```
Enter the function: x*exp(x)-2
enter the initial approximation: 1
Enetr the number of iterations: 5
iteration 1      the root 0.868          function value 0.067

iteration 2      the root 0.853          function value 0.001

iteration 3      the root 0.853          function value 0.000

iteration 4      the root 0.853          function value 0.000

iteration 5      the root 0.853          function value -0.000
```

In [6]:

```
from sympy import*
x=Symbol('x')
g=input('Enter the function: ')
f=lambdify(x,g)
dg=diff(g);
df=lambdify(x,dg)
x0=float(input('enter the initial approximation: '));
n=int(input('Enetr the number of iterations: '));
for i in range(1,n+1):
    x1=(x0-(f(x0)/df(x0)))
    print('iteration %d \t the root %0.3f \t function value %0.3f \n'%(i,x1,f(x1))
    x0=x1
```

```
Enter the function: x**4-x
enter the initial approximation: 1
Enetr the number of iterations: 4
iteration 1      the root 1.000          function value 0.000

iteration 2      the root 1.000          function value 0.000

iteration 3      the root 1.000          function value 0.000

iteration 4      the root 1.000          function value 0.000
```

In [7]:

```
from sympy import*
x=Symbol('x')
g=input('Enter the function: ')
f=lambdify(x,g)
dg=diff(g);
df=lambdify(x,dg)
x0=float(input('enter the initial approximation: '));
n=int(input('Enetr the number of iterations: '));
for i in range(1,n+1):
    x1=(x0-(f(x0)/df(x0)))
    print('iteration %d \t the root %0.3f \t function value %0.3f \n'%(i,x1,f(x1))
    x0=x1
```

```
Enter the function: x**4+x**3-7*x**2-x+5
enter the initial approximation: 3
Enetr the number of iterations: 7
iteration 1      the root 2.489          function value 12.950

iteration 2      the root 2.198          function value 2.934

iteration 3      the root 2.081          function value 0.373

iteration 4      the root 2.061          function value 0.010

iteration 5      the root 2.061          function value 0.000

iteration 6      the root 2.061          function value 0.000

iteration 7      the root 2.061          function value 0.000
```

In [2]:

```
#newtons forward interpolation
from sympy import*
import numpy as np
n=int(input('Enetr the number of data points: '))
x=np.zeros((n))
y=np.zeros((n,n))

#reading data points
print('enter data for x and y: ')
for i in range(n):
    x[i]=float(input('x['+str(i)+']='))
    y[i][0]=float(input('y['+str(i)+']='))

#generating forward difference table
for i in range(1,n):
    for j in range(0,n-i):
        y[j][i]=y[j+1][i-1]-y[j][i-1]
print('\nFORWARD DIFFERENCE TABLE\n');

for i in range(0,n):
    print('%0.2f' %(x[i]),end='')
```

```
    for j in range(0,n-i):
        print('\t\t%0.2f' %(y[i][j]),end='')
    print()

#obtaining the polynomial
t=symbols('t')
f=[]
p=(t-x[0])/(x[1]-x[0])
f.append(p)
for i in range(1,n-1):
    f.append(f[i-1]*(p-i)/(i+1))
    poly=y[0][0]
for i in range(n-1):
    poly=poly+y[0][i+1]*f[i]
simp_poly=simplify(poly)
print(simp_poly)

#if u want to interpolate at some point the next session will help
inter=input('Do you want to interpolate at a point(y/n)? ')
if inter=='y':
    a=float(input('enter the point'))
    interpol=lambdify(t,simp_poly)
    result=interpol(a)
    print('\nThe value of the function at ',a,'is\n',result);
```

```
Enetr the number of data points: 5
enter data for x and y:
x[0]=0
y[0]=1
x[1]=1
y[1]=7
x[2]=2
y[2]=23
x[3]=3
y[3]=55
x[4]=4
y[4]=109

FORWARD DIFFERENCE TABLE

0.00            1.00            6.00            10.00           6.00            0.
00
1.00            7.00            16.00           16.00           6.00
2.00            23.00           32.00           22.00
3.00            55.00           54.00
4.00            109.00
1.0*t**3 + 2.0*t**2 + 3.0*t + 1.0
Do you want to interpolate at a point(y/n)? y
enter the point0.5

The value of the function at  0.5 is
 3.125
```

In [3]:

```
#newtons forward interpolation
from sympy import*
import numpy as np
n=int(input('Enetr the number of data points: '))
x=np.zeros((n))
y=np.zeros((n,n))

#reading data points
print('enter data for x and y: ')
for i in range(n):
    x[i]=float(input('x['+str(i)+']='))
```

```
    y[i][0]=float(input('y['+str(i)+']='))

#generating forward difference table
for i in range(1,n):
    for j in range(0,n-i):
        y[j][i]=y[j+1][i-1]-y[j][i-1]
print('\nFORWARD DIFFERENCE TABLE\n');

for i in range(0,n):
    print('%0.2f' %(x[i]),end='')
    for j in range(0,n-i):
        print('\t\t%0.2f' %(y[i][j]),end='')
    print()

#obtaining the polynomial
t=symbols('t')
f=[]
p=(t-x[0])/(x[1]-x[0])
f.append(p)
for i in range(1,n-1):
    f.append(f[i-1]*(p-i)/(i+1))
    poly=y[0][0]
for i in range(n-1):
    poly=poly+y[0][i+1]*f[i]
simp_poly=simplify(poly)
print(simp_poly)

#if u want to interpolate at some point the next session will help
inter=input('Do you want to interpolate at a point(y/n)? ')
if inter=='y':
    a=float(input('enter the point'))
    interpol=lambdify(t,simp_poly)
    result=interpol(a)
    print('\nThe value of the function at ',a,'is\n',result);
```

```
Enetr the number of data points: 4
enter data for x and y:
x[0]=0
y[0]=1
x[1]=1
y[1]=2
x[2]=2
y[2]=1
x[3]=3
y[3]=10

FORWARD DIFFERENCE TABLE

0.00            1.00            1.00            -2.00           12.00
1.00            2.00            -1.00           10.00
2.00            1.00            9.00
3.00            10.00
2.0*t**3 - 7.0*t**2 + 6.0*t + 1.0
Do you want to interpolate at a point(y/n)? y
enter the point0.5

The value of the function at  0.5 is
 2.5
```

In [4]:

```
#newtons forward interpolation
from sympy import*
import numpy as np
n=int(input('Enetr the number of data points: '))
x=np.zeros((n))
```

```
y=np.zeros((n,n))

#reading data points
print('enter data for x and y: ')
for i in range(n):
    x[i]=float(input('x['+str(i)+']='))
    y[i][0]=float(input('y['+str(i)+']='))

#generating forward difference table
for i in range(1,n):
    for j in range(0,n-i):
        y[j][i]=y[j+1][i-1]-y[j][i-1]
print('\nFORWARD DIFFERENCE TABLE\n');

for i in range(0,n):
    print('%0.2f' %(x[i]),end='')
    for j in range(0,n-i):
        print('\t\t%0.2f' %(y[i][j]),end='')
    print()

#obtaining the polynomial
t=symbols('t')
f=[]
p=(t-x[0])/(x[1]-x[0])
f.append(p)
for i in range(1,n-1):
    f.append(f[i-1]*(p-i)/(i+1))
    poly=y[0][0]
for i in range(n-1):
    poly=poly+y[0][i+1]*f[i]
simp_poly=simplify(poly)
print(simp_poly)

#if u want to interpolate at some point the next session will help
#inter=input('Do you want to interpolate at a point(y/n)? ')
#if inter=='y':
a=float(input('enter the point'))
interpol=lambdify(t,simp_poly)
result=interpol(a)
print('\nThe value of the function at ',a,'is\n',result);
```

```
Enetr the number of data points: 4
enter data for x and y:
x[0]=50
y[0]=9
x[1]=150
y[1]=30
x[2]=250
y[2]=35
x[3]=350
y[3]=42

FORWARD DIFFERENCE TABLE

50.00           9.00            21.00           -16.00          18.00
150.00          30.00           5.00            2.00
250.00          35.00           7.00
350.00          42.00
3.0e-6*t**3 - 0.00215*t**2 + 0.5425*t - 13.125
enter the point100

The value of the function at  100.0 is
 22.625
```

In [5]:
```
#newtons forward interpolation
from sympy import*
import numpy as np
n=int(input('Enetr the number of data points: '))
x=np.zeros((n))
y=np.zeros((n,n))

#reading data points
print('enter data for x and y: ')
for i in range(n):
    x[i]=float(input('x['+str(i)+']='))
    y[i][0]=float(input('y['+str(i)+']='))

#generating forward difference table
for i in range(1,n):
    for j in range(0,n-i):
        y[j][i]=y[j+1][i-1]-y[j][i-1]
print('\nFORWARD DIFFERENCE TABLE\n');

for i in range(0,n):
    print('%0.2f' %(x[i]),end='')
    for j in range(0,n-i):
        print('\t\t%0.2f' %(y[i][j]),end='')
    print()

#obtaining the polynomial
t=symbols('t')
f=[]
p=(t-x[0])/(x[1]-x[0])
f.append(p)
for i in range(1,n-1):
    f.append(f[i-1]*(p-i)/(i+1))
    poly=y[0][0]
for i in range(n-1):
    poly=poly+y[0][i+1]*f[i]
simp_poly=simplify(poly)

#print(simp_poly)
#if u want to interpolate at some point the next session will help
#inter=input('Do you want to interpolate at a point(y/n)? ')
#if inter=='y':
a=float(input('enter the point '))
b=float(input('enter the point '))
interpol=lambdify(t,simp_poly)
result1=interpol(a)
result2=interpol(b)
print('\nThe value of the function at',a,'is\n',result1);
print('\nThe value of the function at',b,'is\n',result2);
```

```
Enetr the number of data points: 7
enter data for x and y:
x[0]=3
y[0]=4.8
x[1]=4
y[1]=8.4
x[2]=5
y[2]=14.5
x[3]=6
y[3]=23.6
x[4]=7
y[4]=36.2
x[5]=8
y[5]=52.8
x[6]=9
y[6]=73.9

FORWARD DIFFERENCE TABLE

3.00            4.80            3.60            2.50            0.50            -
0.00            -0.00           0.00
4.00            8.40            6.10            3.00            0.50            -
0.00            0.00
5.00            14.50           9.10            3.50            0.50            0.
00
6.00            23.60           12.60           4.00            0.50
7.00            36.20           16.60           4.50
8.00            52.80           21.10
9.00            73.90
enter the point 1
enter the point 10

The value of the function at 1.0 is
 3.10000000000021

The value of the function at 10.0 is
 100.0000000000002
```

In [1]:

```
from sympy import*
x=Symbol('x')
g=input("Enter the function: ")
f=lambdify(x,g)
a=float(input("Enter lower limits: "))
b=float(input("Enter upper limits: "))
n=int(input("Enter the number of strips: "))
h=(b-a)/n
k=1
sum=0
while(k<n):
    x=a+k*h
    sum=sum+f(x)
    k=k+1
Ia=(h/2)*(f(a)+f(b)+2*sum)
print("Approximate value of integration from Trapezoidal Rule is: %0.4f"%Ia)
```

```
Enter the function: 1/(1+x**2)
Enter lower limits: 0
Enter upper limits: 1
Enter the number of strips: 4
Approximate value of integration from Trapezoidal Rule is: 0.7828
```

In [3]:

```
from sympy import*
x=Symbol('x')
g=input("Enter the function: ")
```

```
Ie=integrate(g,(x,0,1))
f=lambdify(x,g)
a=float(input("Enter lower limits: "))
b=float(input("Enter upper limits: "))
n=int(input("Enter the number of strips: "))
h=(b-a)/n
k=1
sum=0
while(k<n):
    x=a+k*h
    sum=sum+f(x)
    k=k+1
Ia=(h/2)*(f(a)+f(b)+2*sum)
print("Exact value of integration is: %0.4f"%float(Ie))
print("Approximate value of integration from Trapezoidal Rule is: %0.4f"%Ia)
print("error=",abs(Ia-float(Ie)))
```

```
Enter the function: 1/(1+x**2)
Enter lower limits: 0
Enter upper limits: 1
Enter the number of strips: 4
Exact value of integration is: 0.7854
Approximate value of integration from Trapezoidal Rule is: 0.7828
error= 0.0026040457503894165
```

In [5]:

```
from sympy import*
x=Symbol('x')
g=input("Enter the function: ")
f=lambdify(x,g)
a=float(input("Enter lower limits: "))
b=float(input("Enter upper limits: "))
n=int(input("Enter the number of strips: "))
h=(b-a)/n
k=1
sum=0
while(k<n):
    x=a+k*h
    if(k%2==0):
        sum=sum+2*f(x)
    else:
        sum=sum+4*f(x)
    k=k+1
Ia=(h/3)*(f(a)+f(b)+sum)
print("Approximate value of integration from Simpson 1/3 Rule is: %0.4f"%Ia)
```

```
Enter the function: 1/(1+x)
Enter lower limits: 1
Enter upper limits: 3
Enter the number of strips: 8
Approximate value of integration from Simpson 1/3 Rule is: 0.6932
```

In [11]:

```
from sympy import*
x=Symbol('x')
g=input("Enter the function: ")
Ie=integrate(g,(x,1,3))
f=lambdify(x,g)
a=float(input("Enter lower limits: "))
b=float(input("Enter upper limits: "))
n=int(input("Enter the number of strips: "))
h=(b-a)/n
k=1
sum=0
while(k<n):
    x=a+k*h
    if(k%2==0):
```

```
        sum=sum+2*f(x)
    else:
        sum=sum+4*f(x)
    k=k+1
Ia=(h/3)*(f(a)+f(b)+sum)
print("Exact value of integration is: %0.4f"%float(Ie))
print("Approximate value of integration from Simpson 1/3 Rule is: %0.4f"%Ia)
print("error=%0.4f" %abs(Ia-float(Ie)))
```

```
Enter the function: 1/(1+x)
Enter lower limits: 1
Enter upper limits: 3
Enter the number of strips: 8
Exact value of integration is: 0.6931
Approximate value of integration from Simpson 1/3 Rule is: 0.6932
error=0.0000
```

In [13]:

```
from sympy import*
x=Symbol('x')
g=input("Enter the function: ")
f=lambdify(x,g)
a=float(input("Enter lower limits: "))
b=float(input("Enter upper limits: "))
n=int(input("Enter the number of strips: "))
h=(b-a)/n
k=1
sum=0
while(k<n):
    x=a+k*h
    if(k%3==0):
        sum=sum+2*f(x)
    else:
        sum=sum+3*f(x)
    k=k+1
Ia=(3*h/8)*(f(a)+f(b)+sum)
print("Approximate value of integration from Simpson 3/8 Rule is: %0.4f"%Ia)
```

```
Enter the function: 1/((1+x)**(1/2))
Enter lower limits: 0
Enter upper limits: 6
Enter the number of strips: 6
Approximate value of integration from Simpson 3/8 Rule is: 3.2991
```

In [15]:

```
from sympy import*
x=Symbol('x')
g=input("Enter the function: ")
Ie=integrate(g,(x,0,6))
f=lambdify(x,g)
a=float(input("Enter lower limits: "))
b=float(input("Enter upper limits: "))
n=int(input("Enter the number of strips: "))
h=(b-a)/n
k=1
sum=0
while(k<n):
    x=a+k*h
    if(k%3==0):
        sum=sum+2*f(x)
    else:
        sum=sum+3*f(x)
    k=k+1
Ia=(3*h/8)*(f(a)+f(b)+sum)
print("Exact value of integration is: %0.4f"%float(Ie))
print("Approximate value of integration from Simpson 3/8 Rule is: %0.4f"%Ia)
print("error=%0.4f" %abs(Ia-float(Ie)))
```

```
Enter the function: 1/((1+x)**(1/2))
Enter lower limits: 0
Enter upper limits: 6
Enter the number of strips: 100
Exact value of integration is: 3.2915
Approximate value of integration from Simpson 3/8 Rule is: 3.2858
error=0.0057
```

In [1]:
```
#Taylor series method to solve dy/dx=x**2+y**2 at x=0 and y=1
from sympy import*
x,y=symbols('x,y')
x0,x1,x2=0,0.1,0.2
yd0=1
yd1=x**2+y**2
yd2=2*x+2*y*yd1
yd3=2+2*y*yd2+yd1*2*yd1
yd1=yd1.subs({x:0,y:1})
yd2=yd2.subs({x:0,y:1})
yd3=yd3.subs({x:0,y:1})
ts=yd0+(x-x0)*yd1+(x-x0)**2*yd2/2+(x-x0)**3*yd3/6
tsx1=ts.subs(x,x1)
tsx2=ts.subs(x,x2)
print("Taylor series is: ",ts)
print('y(',x1,')=',round(tsx1,4))
print('y(',x2,')=',round(tsx2,4))
```

```
Taylor series is:  4*x**3/3 + x**2 + x + 1
y( 0.1 )= 1.1113
y( 0.2 )= 1.2507
```

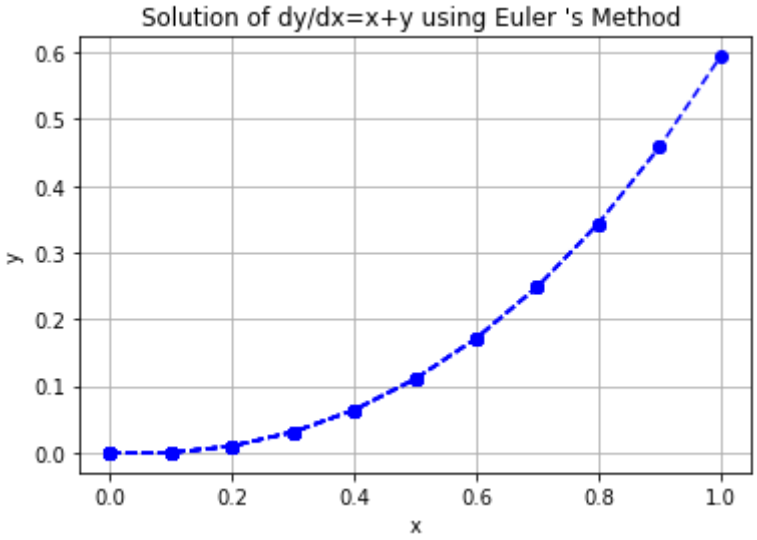
In [4]:
```
from sympy import*
x,y=symbols('x,y')
x0,x1,x2=1,1.1,1.2
yd0=0
yd1=x+y
yd2=1+y*yd1
yd3=yd2
yd1=yd1.subs({x:1,y:0})
yd2=yd2.subs({x:1,y:0})
yd3=yd3.subs({x:1,y:0})
ts=yd0+(x-x0)*yd1+(x-x0)**2*yd2/2+(x-x0)**3*yd3/6
tsx1=ts.subs(x,x1)
tsx2=ts.subs(x,x2)
print("Taylor series is: ",ts)
print('y(',x1,')=',round(tsx1,4))
print('y(',x2,')=',round(tsx2,4))
```

```
Taylor series is:  x + (x - 1)**3/6 + (x - 1)**2/2 - 1
y( 1.1 )= 0.1052
y( 1.2 )= 0.2213
```

In [6]:
```
#Eulers method
import matplotlib.pyplot as plt
import math
def f(x,y):
    return(x+y)
x0=float(input("Enter x0: "))
y0=float(input("Enter y0: "))
h=float(input("Enter h: "))
n=int(input("Enter n: "))
x0_plot=[]; y0_plot=[]
for i in range(n+1):
    y1=y0+h*f(x0,y0)
    x0_plot.append(x0); y0_plot.append(y0)
```

```
    plt.plot(x0_plot,y0_plot,'bo--')
    x0=x0+h
    y0=y1

    print("The value of y at x",round(x0,2), "is=", round(y0,4))

plt.xlabel('x')
plt.ylabel('y')
plt.title('Solution of dy/dx=x+y using Euler \'s Method ')
plt.grid(True)
def f(x):
    return -1-x+math.exp(x)
print("exact solution at one= ",round(f(1),4))
plt.show()
```

```
Enter x0: 0
Enter y0: 0
Enter h: 0.1
Enter n: 10
The value of y at x 0.1 is= 0.0
The value of y at x 0.2 is= 0.01
The value of y at x 0.3 is= 0.031
The value of y at x 0.4 is= 0.0641
The value of y at x 0.5 is= 0.1105
The value of y at x 0.6 is= 0.1716
The value of y at x 0.7 is= 0.2487
The value of y at x 0.8 is= 0.3436
The value of y at x 0.9 is= 0.4579
The value of y at x 1.0 is= 0.5937
The value of y at x 1.1 is= 0.7531
exact solution at one=  0.7183
```

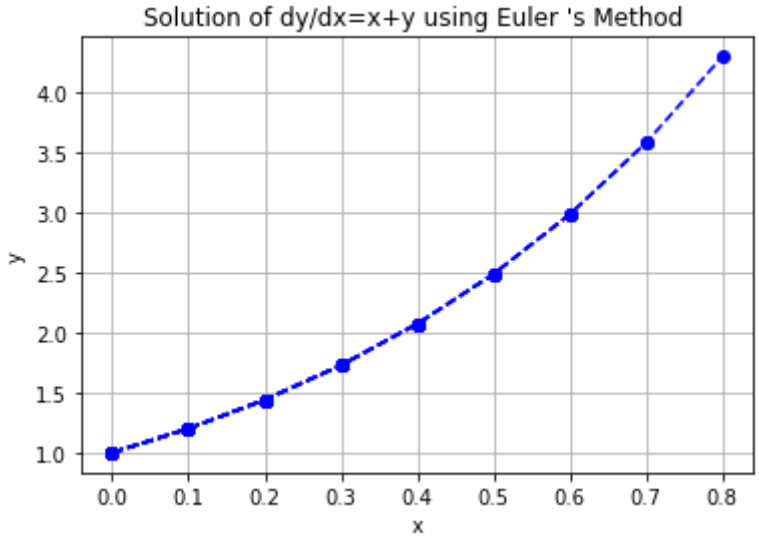


In [14]:

```
import matplotlib.pyplot as plt
import math
def f(x,y):
    return(2*y)
x0=float(input("Enter x0: "))
y0=float(input("Enter y0: "))
h=float(input("Enter h: "))
n=int(input("Enter n: "))
x0_plot=[]; y0_plot=[]
for i in range(n+1):
    y1=y0+h*f(x0,y0)
    x0_plot.append(x0); y0_plot.append(y0)
    plt.plot(x0_plot,y0_plot,'bo--')
    x0=x0+h
    y0=y1
```

```
    print("The value of y at x",round(x0,2), "is=", round(y0,4))

plt.xlabel('x')
plt.ylabel('y')
plt.title('Solution of dy/dx=x+y using Euler \'s Method ')
plt.grid(True)
def f(x):
    return -1-x+math.exp(2*x)
print("exact solution at one= ",round(f(1),4))
plt.show()
```

```
Enter x0: 0
Enter y0: 1
Enter h: 0.1
Enter n: 8
The value of y at x 0.1 is= 1.2
The value of y at x 0.2 is= 1.44
The value of y at x 0.3 is= 1.728
The value of y at x 0.4 is= 2.0736
The value of y at x 0.5 is= 2.4883
The value of y at x 0.6 is= 2.986
The value of y at x 0.7 is= 3.5832
The value of y at x 0.8 is= 4.2998
The value of y at x 0.9 is= 5.1598
exact solution at one=  5.3891
```

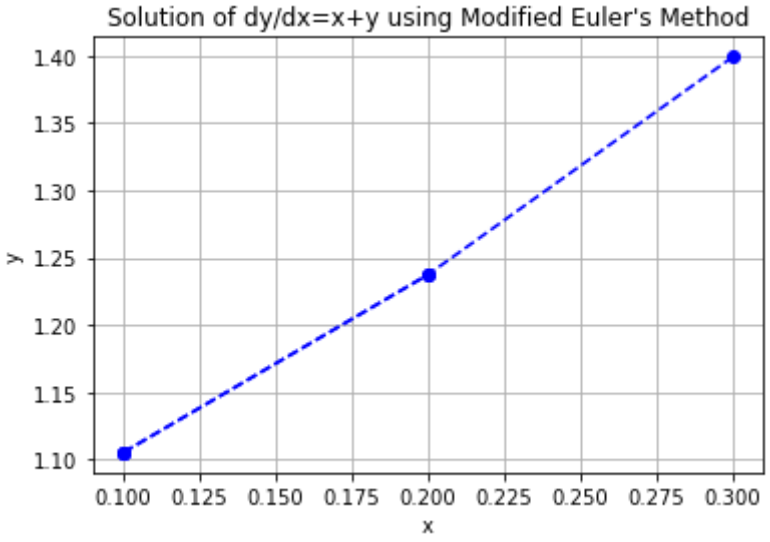


In [15]:

```
#Modified Eulers Method
import matplotlib.pyplot as plt
def f(x,y):
    return(x+y)
x0=float(input("Enter x0: "))
y0=float(input("Enter y0: "))
h=float(input("Enter h: "))
n=int(input("Enter n: "))
x0_plot=[]; y0_plot=[]
for i in range(n):
    k=f(x0,y0)
    y1=y0+h*k
    print('y(',round(x0+h,3),')=',round(y1,4))
    for j in range(1,4):
        y1=y0+(h/2)*(k+f(x0+i*h,y1))
        print('The',j,'the improved value of y(',round(x0+h,3),')=',round(y1,4))
    x0=x0+h
    y0=y1
    x0_plot.append(x0); y0_plot.append(y0)
    plt.plot(x0_plot,y0_plot,'bo--')
```

```
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Solution of dy/dx=x+y using Modified Euler\'s Method')
    plt.grid(True)
import math
def f(x):
    return -1-x+math.exp(x)
print("exact solution at one= ",round(f(1),4))
```

```
Enter x0: 0
Enter y0: 1
Enter h: 0.1
Enter n: 3
y( 0.1 )= 1.1
The 1 the improved value of y( 0.1 )= 1.105
The 2 the improved value of y( 0.1 )= 1.1053
The 3 the improved value of y( 0.1 )= 1.1053
y( 0.2 )= 1.2258
The 1 the improved value of y( 0.2 )= 1.2368
The 2 the improved value of y( 0.2 )= 1.2374
The 3 the improved value of y( 0.2 )= 1.2374
y( 0.3 )= 1.3811
The 1 the improved value of y( 0.3 )= 1.3983
The 2 the improved value of y( 0.3 )= 1.3992
The 3 the improved value of y( 0.3 )= 1.3992
exact solution at one=  0.7183
```



In [ ]:

```
#use Newton backward interpolation to obtain the interpolating polynomial and hence
from sympy import*
import numpy as np
import sys
print("This will use Newton's backward interpolation formula ")
#Reading number of unknowns
n=int(input("Enter number of data points: "))
#making numpy array of n & nx
x=np.zeros((n))
y=np.zeros((n,n))
print('enter the data for x and y')
for i in range(n):
    x[i]=float(input('x['+str(i)+']='))
    y[i][0]=float(input('y['+str(i)+']='))
#generating backward difference table
for i in range(1,n):
    for j in range(n-1,i-2,-1):
        y[j][i]=y[j][i-1]-y[j-1][i-1]
print('\nBACKWARD DIFFERENCE TABLE\n');
```

```
for i in range(0,n):
    print('%0.2f'%(x[i]),end='')
    for j in range(0,i+1):
        print('\t%0.2f'%(y[i][j]),end='')
    print()

#obtaining polynomial
t=symbols('t')
f=[]
p=(y-x[n-1])/(x[1]-x[0])
f.append(p)
for i in range(1,n-1):
    f.append(f[i-1]*(p+i)/(i+1))
poly=y[n-1][0]
print(poly)
for i in range(n-1):
    poly=poly+y[n-1][i+1]*f[i]
    simp_poly=simplify(poly)
print('\nTHE INTERPOLATING POLYNOMIAL IS\n');
print(simp_poly)

#if you want to interpolate at some point the next session will help
inter=input('Do you want to interpolate at a point(y/n)?')
if inter=='y':
    a=float(input('enter the point:'))
    interpol=lambdify(t,simp_poly)
    result=interpol(a)
    print('\nThe value of the function at',a,'is\n',result);
```

```
This will use Newton's backward interpolation formula
```

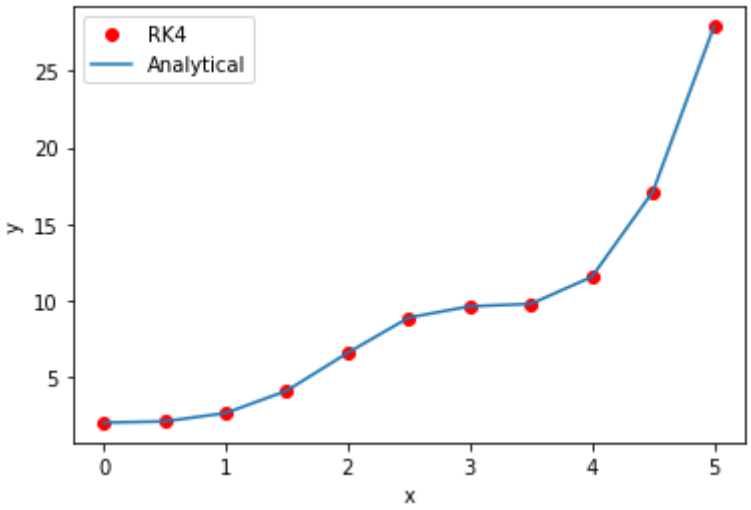
In [3]:

```
#solve by using RK fourth order method dy/dx=ysin^2(x),y(0)=2,h=0.5,xn=5
import math
import matplotlib.pyplot as plt
#functions
dy=lambda x,y:math.sin(x)**2*y
f=lambda x:2*math.exp(0.5*(x-math.sin(x)*math.cos(x)))

#initialization
xi=0;
xf=5;
h=0.5
n=int((xf-xi)/h)
x=0;y=2

#visualisation of code initialisation
print('x \t\t y \t\t f(x)');
print('%f\t % f\t%f'%(x,y,f(x)))
x_plot=[];
y_RK4=[];
y_analytical=[]

#RK4 method
for i in range(1,n+1):
    x_plot.append(x);
    y_RK4.append(y);
    y_analytical.append(f(x))

    #euler's method
    k1=dy(x,y)
    k2=dy(x+h/2,y+k1*h/2)
    k3=dy(x+h/2,y+k2*h/2)
    k4=dy(x+h,y+k3*h)
```

```
    #calculate new y estimation
    y=y+1/6*(k1+2*k2+2*k3+k4)*h
    #increase x by step size to calculate next y estimate
    x=x+h
    print('%f\t %f \t %f'%(x,y,f(x)))

#visualisation of code
x_plot.append(x);
y_RK4.append(y);
y_analytical.append(f(x))
plt.plot(x_plot,y_RK4,'ro',x_plot,y_analytical)
plt.xlabel('x');
plt.ylabel('y');
plt.legend(["RK4",'Analytical'])
```

```
x               y               f(x)
0.000000         2.000000      2.000000
0.500000        2.080616        2.080856
1.000000        2.626551        2.626948
1.500000        4.086093        4.087229
2.000000        6.566018        6.568909
2.500000        8.867524        8.871805
3.000000        9.606408        9.611892
3.500000        9.759067        9.765937
4.000000        11.531199       11.539866
4.500000        17.103444       17.117778
5.000000        27.886259       27.914673
```

Out[3]: <matplotlib.legend.Legend at 0x1f652aeef40>



In [5]:
```
#milne predictor corrector method
#Solve dy/dx=2y/x, Y(1)=2, Y(1.25)=3.13, Y(1.5)
x0=1
y0=2
y1=3.13
y2=4.5
y3=6.13
h=0.25
x1=x0+h
x2=x1+h
x3=x2+h
x4=x3+h
def f(x,y):
    return (2*y)/x
y10=f(x0,y0)
y11=f(x1,y1)
y12=f(x2,y2)
```

```
y13=f(x3,y3)
y4p=y0+(4*h/3)*(2*y11-y12+2*y13)
print('predicted value of y4 is %3.3f '%y4p )
y14=f(x4,y4p);
for i in range (1,4):
    y4=y2+(h/3)*(y12+4*y13+y14);
    print('corrected value of y4 after \t iteration %d is \t %3.4f\t'%(i,y4))
    y14=f(x4,y4);
def f(x):
    return 2*x**2
print("Exact value at 2=", round(f(2),4))
```

```
predicted value of y4 is 8.009
corrected value of y4 after      iteration 1 is          8.0027
corrected value of y4 after      iteration 2 is          8.0021
corrected value of y4 after      iteration 3 is          8.0021
Exact value at 2= 8
```

In [7]:
```
from sympy import*
x,y=symbols('x y ')
w1=integrate(x**2+y**2,(y,0,x),(x,0,1))
print(w1)
```

```
1/3
```

In [8]:
```
from sympy import*
x,y,z=symbols('x y z')
w2=integrate(x+y+z,(x,0,z),(y,x-z,x+z),(z,-1,1))
print(w2)
```

```
4*x/3
```

In [9]:
```
from sympy import*
x,y,z=symbols('x y z')
w2=integrate((x*y*z),(z,0,3-x-y),(y,0,3-x),(x,0,3))
print(w2)
```

```
81/80
```

In [1]:
```
from sympy import*
x,y=symbols('x y')
A=integrate(1,(y,0,sqrt(16-x**2)),(x,0,4))
print(A)
```

```
4*pi
```

In [3]:
```
from sympy import*
x,y=symbols('x y')
a=4
b=6
w3=4*integrate(1,(y,0,(b/a)*sqrt(a**2-x**2)),(x,0,a))
print(w3)
```

```
24.0*pi
```

In [4]:
```
from sympy import*
r,t,a=symbols('r t a')
A=2*integrate(r,(r,0,a*(1+cos(t))),(t,0,pi))
pprint(A)
```

```
     2
3·π·a
──────
  2
```

In [5]:
```
from sympy import*
x,y,z=symbols('x y z')
A=integrate(1,(z,0,5*(1-x/3-y/4)),(y,0,4*(1-x/3)),(x,0,3))
print(A)
```

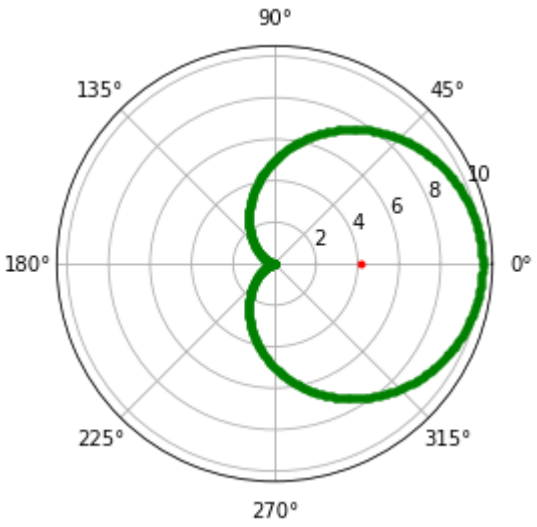
```
10
```

In [7]:
```
from sympy import*
x,y,z,a,b,c=symbols('x y z a b c')
A=integrate(1,(z,0,c*(1-x/a-y/b)),(y,0,b*(1-x/a)),(x,0,a))
print(A)
```

```
a*b*c/6
```

In [1]:
```
#center of gravity
import numpy as np
import matplotlib.pyplot as plt
import math
from sympy import*
r,t,a=symbols('r t a')
I1=integrate(cos(t)*r**2,(r,0,a*(1+cos(t))),(t,-pi,pi))
I2=integrate(r,(r,0,a*(1+cos(t))),(t,-pi,pi))
I=I1/I2
print(I)
I=I.subs(a,5)
plt.axes(projection='polar')
a=5
rad=np.arange(0,(2*np.pi),0.01)

#plotting the cardoid
for i in rad:
    r=a+(a*np.cos(i))
    plt.polar(i,r,'g.')
plt.polar(0,I,'r.')
plt.show()
```

```
5*a/6
```



In [2]:
```
from sympy import*
x=Symbol('x')
w1=integrate(exp(-x),(x,0,float('inf')))
print(w1)
```

```
1
```

In [3]:
```
from sympy import*
x=Symbol('x')
```

```
w1=integrate(exp(-x)*x**4,(x,0,float('inf')))
print(w1)
```

```
24
```

In [4]:
```
from sympy import*
t,s=symbols('t s')
#for infinity in sympy we use oo
w1=integrate(exp(-s*t)*cos(4*t),(t,0,oo))
display(w1)
```

$$\begin{cases} \frac{s}{16\left(\frac{s^2}{16}+1\right)} & \text{for } |\arg(s)| < \frac{\pi}{2} \\ \int_0^\infty e^{-st}\cos(4t)\,dt & \text{otherwise} \end{cases}$$

In [5]:
```
#beta and gamma functions
from sympy import beta,gamma
m=float(input('m: '));
n=float(input('n: '));
s=beta(m,n);
t=gamma(n)
print('gamma(',n,') is %3.3f '%t)
print('Beta(',m,n,') is %3.3f '%s)
```

```
m: 3
n: 5
gamma( 5.0 ) is 24.000
Beta( 3.0 5.0 ) is 0.010
```

In [7]:
```
from sympy import beta,gamma
m=float(input('m: '));
n=float(input('n: '));
s=beta(m,n);
t=gamma(n)
print('gamma(',n,') is %3.3f '%t)
print('Beta(',m,n,') is %3.3f '%s)
```

```
m: 2.5
n: 3.5
gamma( 3.5 ) is 3.323
Beta( 2.5 3.5 ) is 0.037
```

In [8]:
```
from sympy import beta,gamma
m=float(input('m: '));
n=float(input('n: '));
s=beta(m,n);
t=(gamma(m)*gamma(n))/gamma(m+n);
print(s,t)
if (abs(s-t)<=0.00001):
    print('beta and gamma are related ')
else:
    print('given values are wrong ')
```

```
m: 5
n: 7
0.000432900432900433 0.000432900432900433
beta and gamma are related
```

In [10]:
```
#to find gradient of scalar point function
from sympy.vector import CoordSys3D,Del,gradient
R= CoordSys3D ('R')
x,y,z=symbols('x y z')
A=R.x**2*R.y+2*R.x*R.z-4
```

```
delop=Del()
display (delop(A))
gradA=gradient(A)
print(f"\n Gradient of {A} is \n")
display(gradA)
```

$$(\frac{\partial}{\partial \mathbf{x_R}}(\mathbf{x_R}^2\mathbf{y_R}+2\mathbf{x_R}\mathbf{z_R}-4))\hat{\mathbf{i}}_\mathbf{R}+(\frac{\partial}{\partial \mathbf{y_R}}(\mathbf{x_R}^2\mathbf{y_R}+2\mathbf{x_R}\mathbf{z_R}-4))\hat{\mathbf{j}}_\mathbf{R}+(\frac{\partial}{\partial \mathbf{z_R}}(\mathbf{x_R}^2\mathbf{y_R}+$$

```
Gradient of R.x**2*R.y + 2*R.x*R.z - 4 is
```

$$(2\mathbf{x_R}\mathbf{y_R}+2\mathbf{z_R})\hat{\mathbf{i}}_\mathbf{R}+(\mathbf{x_R}^2)\hat{\mathbf{j}}_\mathbf{R}+(2\mathbf{x_R})\hat{\mathbf{k}}_\mathbf{R}$$

In [11]:
```
from sympy.vector import*
R= CoordSys3D ('R')
x,y,z=symbols('x y z')
A=R.x**2*R.y+2*R.x*R.z-4
delop=Del()
display (delop(A))
gradA=gradient(A)
print(f"\n Gradient of {A} is \n")
display(gradA)
```

$$(\frac{\partial}{\partial \mathbf{x_R}}(\mathbf{x_R}^2\mathbf{y_R}+2\mathbf{x_R}\mathbf{z_R}-4))\hat{\mathbf{i}}_\mathbf{R}+(\frac{\partial}{\partial \mathbf{y_R}}(\mathbf{x_R}^2\mathbf{y_R}+2\mathbf{x_R}\mathbf{z_R}-4))\hat{\mathbf{j}}_\mathbf{R}+(\frac{\partial}{\partial \mathbf{z_R}}(\mathbf{x_R}^2\mathbf{y_R}+$$

```
Gradient of R.x**2*R.y + 2*R.x*R.z - 4 is
```

$$(2\mathbf{x_R}\mathbf{y_R}+2\mathbf{z_R})\hat{\mathbf{i}}_\mathbf{R}+(\mathbf{x_R}^2)\hat{\mathbf{j}}_\mathbf{R}+(2\mathbf{x_R})\hat{\mathbf{k}}_\mathbf{R}$$

In [14]:
```
#to find divergence and curl of a vector point function
from sympy.vector import*
from sympy import symbols
R=CoordSys3D('R')
x,y,z=symbols('x y z')
A=R.x**2*R.y*R.z*R.i+R.y**2*R.z*R.x*R.i+R.z**2*R.x*R.y*R.i
delop=Del()
divA=delop.dot(A)
curlA=delop.cross(A)
display(divA)
display(curlA)
print(f"\n Divergence of {A} is \n")
display(divergence(A))
print(f"\n Curl of {A} is \n")
display(curl(A))
```

$$\frac{\partial}{\partial \mathbf{x_R}}(\mathbf{x_R}^2\mathbf{y_R}\mathbf{z_R}+\mathbf{x_R}\mathbf{y_R}^2\mathbf{z_R}+\mathbf{x_R}\mathbf{y_R}\mathbf{z_R}^2)$$

$$(\frac{d}{d\mathbf{y_R}}0-\frac{d}{d\mathbf{z_R}}0)\hat{\mathbf{i}}_\mathbf{R}+(-\frac{d}{d\mathbf{x_R}}0+\frac{\partial}{\partial \mathbf{z_R}}(\mathbf{x_R}^2\mathbf{y_R}\mathbf{z_R}+\mathbf{x_R}\mathbf{y_R}^2\mathbf{z_R}+\mathbf{x_R}\mathbf{y_R}\mathbf{z_R}^2))\hat{\mathbf{j}}_\mathbf{R}+(\frac{d}{d\mathbf{x}}$$

```
Divergence of (R.x**2*R.y*R.z + R.x*R.y**2*R.z + R.x*R.y*R.z**2)*R.i is
```

$$2\mathbf{x_R}\mathbf{y_R}\mathbf{z_R}+\mathbf{y_R}^2\mathbf{z_R}+\mathbf{y_R}\mathbf{z_R}^2$$

```
Curl of (R.x**2*R.y*R.z + R.x*R.y**2*R.z + R.x*R.y*R.z**2)*R.i is
```

$$(\mathbf{x_R}^2\mathbf{y_R} + \mathbf{x_R}\mathbf{y_R}^2 + 2\mathbf{x_R}\mathbf{y_R}\mathbf{z_R})\hat{\mathbf{j}}_\mathbf{R} + (-\mathbf{x_R}^2\mathbf{z_R} - 2\mathbf{x_R}\mathbf{y_R}\mathbf{z_R} - \mathbf{x_R}\mathbf{z_R}^2)\hat{\mathbf{k}}_\mathbf{R}$$

In [16]:
```
from sympy import*
var('x y')
p=x+2*y
q=x-2*y
f=diff(q,x)-diff(p,y)
soln=integrate(f,[x,0,1],[y,0,1])
print("I=",soln)
```

```
I= -1
```

In [18]:
```
import numpy as np
from scipy.linalg import null_space
A=np.array([[1,2,3],[4,5,6],[7,8,9]])
rank=np.linalg.matrix_rank(A)
print("Rank of the matrix ",rank)
ns=null_space(A)
print("Null space of matrix ",ns)
nullity=ns.shape[1]
print("Null space of matrix ",nullity)
if rank+nullity==A.shape[1]:
    print("Rank-nullity theorem holds good")
else:
    print("Rank-nullity theorem does not hold good")
```

```
Rank of the matrix  2
Null space of matrix  [[ 0.40824829]
 [-0.81649658]
 [ 0.40824829]]
Null space of matrix  1
Rank-nullity theorem holds good
```

In [19]:
```
import numpy as np
V=np.array([
    [1,2,3],
    [2,3,1],
    [3,1,2]])
basis=np.linalg.matrix_rank(V)
dimension=V.shape[0]
print("Basis of the matrix ",basis)
print("Dimension of the matrix ",dimension)
```

```
Basis of the matrix  3
Dimension of the matrix  3
```

In [3]:
```
from numpy import*
import sympy as sp
A=[[1,-1,1,1],[2,-5,2,2],[3,-3,5,3],[4,-4,4,4]]
AB=array(A)
S=shape(A)
n=len(A)
for i in range(n):
    if AB[i,i]==0:
        ab=copy(AB)
        for k in range (i+1,S[0]):
            if ab[k,i]!=0:
                ab[i,:]=AB[k,:]
                ab[k,:]=AB[i,:]
                AB=copy(ab)
    for j in range(i+1,n):
        Fact=AB[j,i]/AB[i,i]
        for k in range(i,n):
```

```
            AB[j,k]=AB[j,k]-Fact*AB[i,k]
display("REF of given matrix: ",sp.Matrix(AB))
temp={(0,0,0,0)}
result=[]
for idx,row in enumerate(map(tuple,AB)):
    if row not in temp:
        result.append(idx)
print("\n Basis are non-zero rows of A: ")
display(sp.Matrix(AB[result]))
```

'REF of given matrix: '

$$\begin{bmatrix} 1 & -1 & 1 & 1 \\ 0 & -3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Basis are non-zero rows of A:

$$\begin{bmatrix} 1 & -1 & 1 & 1 \\ 0 & -3 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{bmatrix}$$

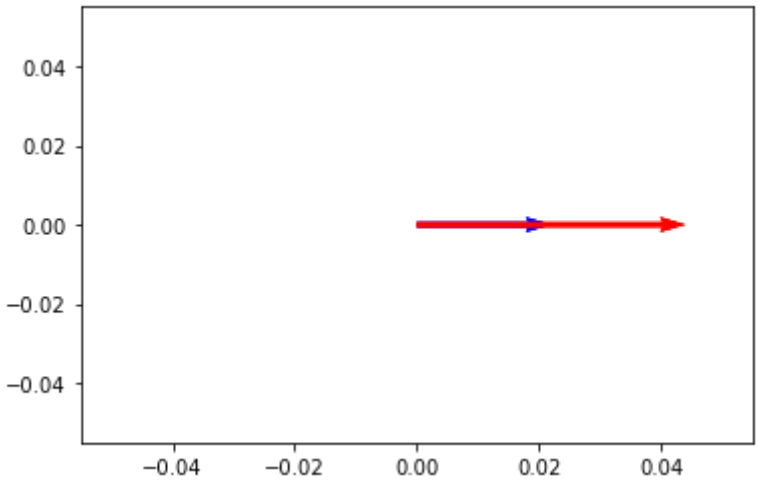
In [5]:
```
#inner product
import numpy as np
A=np.array([2,1,5,4])
B=np.array([3,4,7,8])
output=np.dot(A,B)
print(output)
```

```
77
```

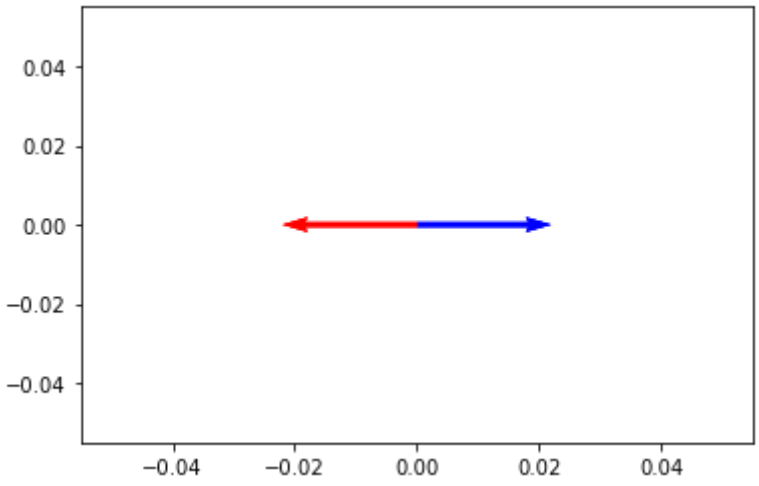
In [6]:
```
import numpy as np
A=np.array([2,1,5,4])
B=np.array([3,4,7,8])
output=np.dot(A,B)
print("Inner product is: ",output)
if output==0:
    print("given vectors are orthogonal ")
else:
    print("given vectors are not orthogonal")
```

```
Inner product is:  77
given vectors are not orthogonal
```

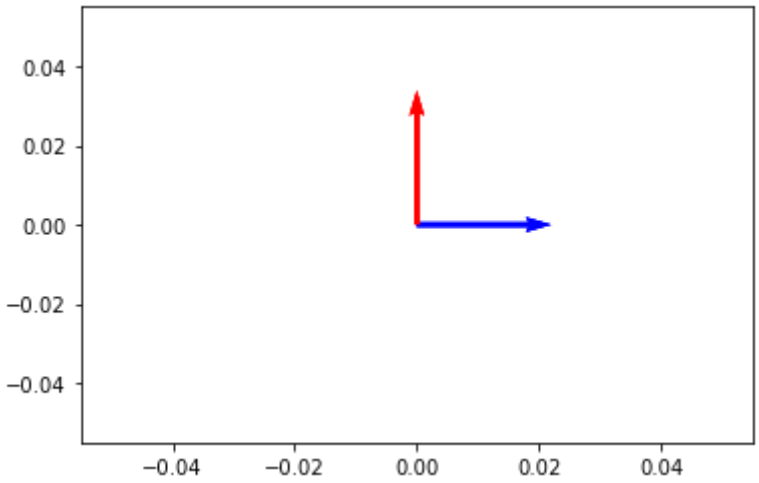
In [8]:
```
#horizontal stretch
#find image of vector(10,0) when it is stretched horizontally by 2 units
import numpy as np
import matplotlib.pyplot as plt
V=np.array([[10,0]])
origin=np.array([[0,0,0],[0,0,0]])
A=np.matrix([[2,0],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
plt.quiver(*origin,V[:,0],V[:,1],color=['b'],scale=50)
plt.quiver(*origin,V2[0,:],V2[1,:],color=['r'],scale=50)
plt.show()
```



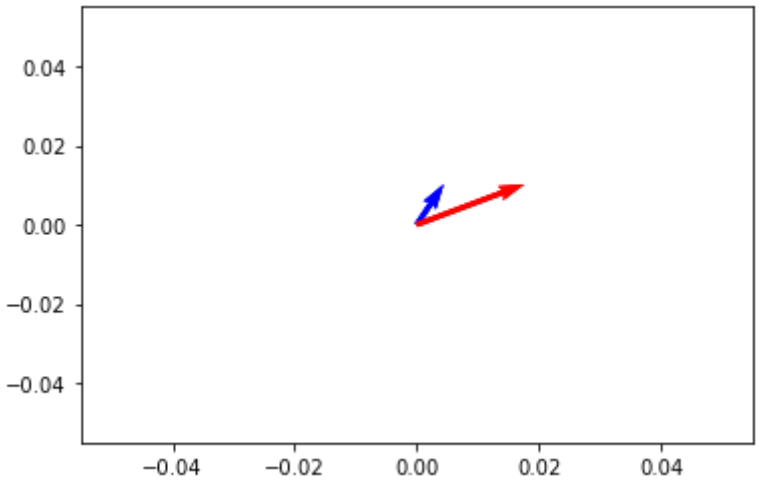
In [9]:
```
#reflection
#find the image of vector (10,0) when it is reflected about y axis
import numpy as np
import matplotlib.pyplot as plt
V=np.array([[10,0]])
origin=np.array([[0,0,0],[0,0,0]])
A=np.matrix([[-1,0],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
plt.quiver(*origin,V[:,0],V[:,1],color=['b'],scale=50)
plt.quiver(*origin,V2[0,:],V2[1,:],color=['r'],scale=50)
plt.show()
```



In [10]:
```
#rotation
#find the image of the vector when it is rotated by pi/2 radians
import numpy as np
import matplotlib.pyplot as plt
V=np.array([[10,0]])
origin=np.array([[0,0,0],[0,0,0]])
A=np.matrix([[0,-1],[1,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
plt.quiver(*origin,V[:,0],V[:,1],color=['b'],scale=50)
plt.quiver(*origin,V2[0,:],V2[1,:],color=['r'],scale=50)
plt.show()
```



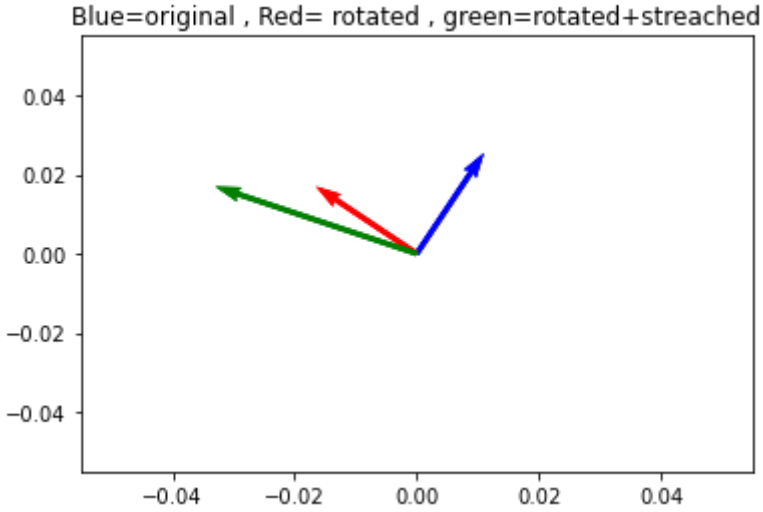
In [11]:
```
#shear transformation
#find image of (2,3) under shear transformation
import numpy as np
import matplotlib.pyplot as plt
V=np.array([[2,3]])
origin=np.array([[0,0,0],[0,0,0]])
A=np.matrix([[1,2],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
plt.quiver(*origin,V[:,0],V[:,1],color=['b'],scale=50)
plt.quiver(*origin,V2[0,:],V2[1,:],color=['r'],scale=50)
plt.show()
```



In [15]:
```
#find the image of the vector (2,3) when it is rotated by pi/2 radians by 2 units
import numpy as np
import matplotlib.pyplot as plt
V=np.array([[2,3]])
origin=np.array([[0,0,0],[0,0,0]])
A=np.matrix([[0,-1],[1,0]])
B=np.matrix([[2,0],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V3=B*V2
V2=np.array(V2)
V3=np.array(V3)
print("Image of given vector is: ",V3)
plt.quiver(*origin,V[:,0],V[:,1],color=['b'],scale=20)
plt.quiver(*origin,V2[0,:],V2[1,:],color=['r'],scale=20)
plt.quiver(*origin,V3[0,:],V3[1,:],color=['g'],scale=20)
```

```
plt.title('Blue=original , Red= rotated , green=rotated+streached')
plt.show()
```

```
Image of given vector is:  [[-6]
 [ 2]]
```



In [ ]: