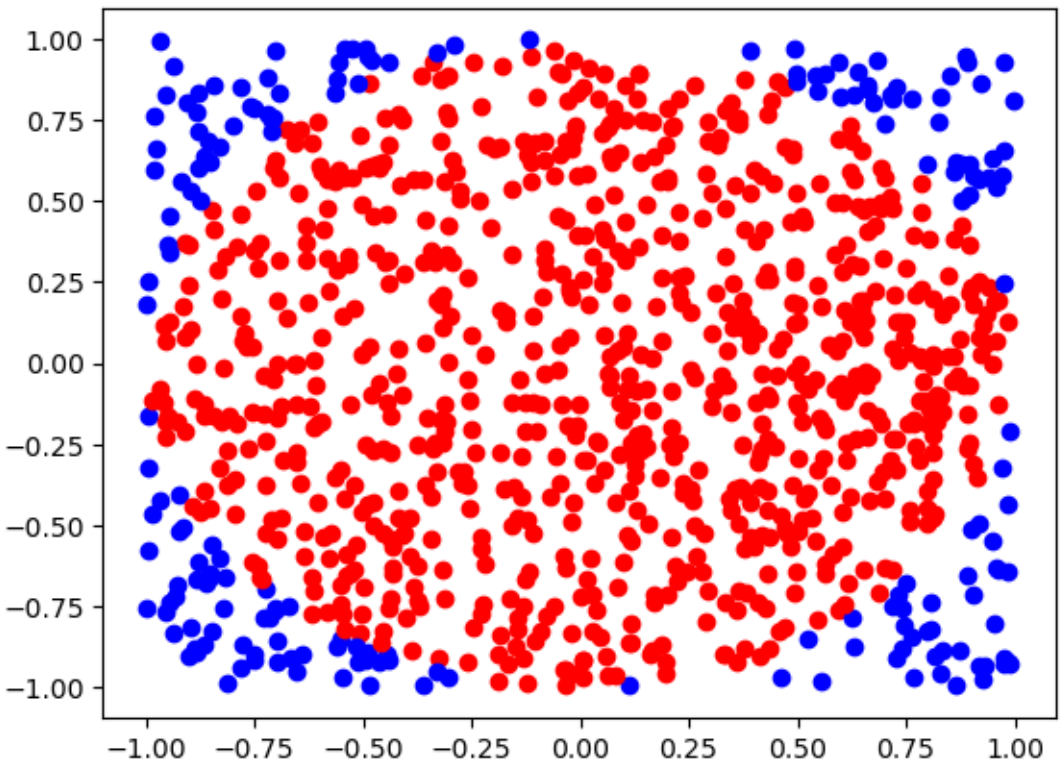
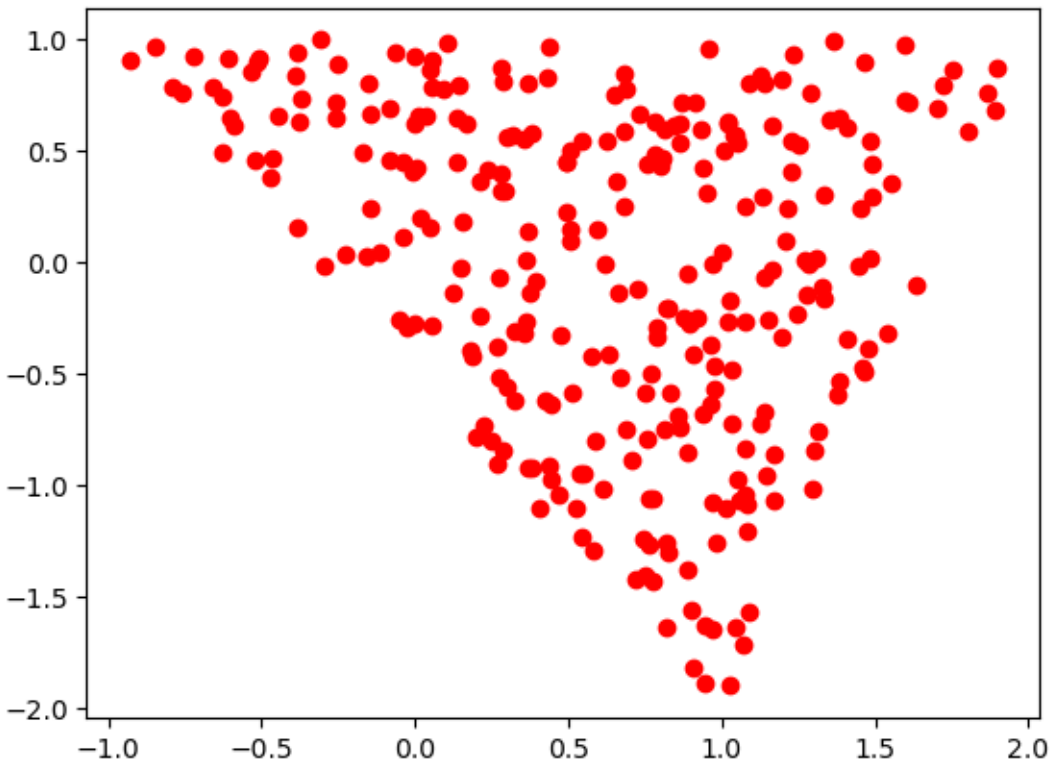
Computational Revision

February 26, 2024

[13]:
```
import random
import numpy as np
import matplotlib.pyplot as plt

n = 1000

points = np.random.uniform(-1,1,(n,2))

for point in points:
    sqdist = point[0]**2 + point[1]**2
    if sqdist <= 1:
        plt.scatter(point[0],point[1],color='r')
    else:
        plt.scatter(point[0],point[1],color='b')
```



[18]:
```
import random
import numpy as np
import matplotlib.pyplot as plt

n=1000
points = np.random.uniform(-2,2,(n,2))

A = [-1,1]
B = [2,1]
C = [1,-2]

def isInside(point,A,B,C):
    x1 = A[0]
    y1 = A[1]
    x2 = B[0]
    y2 = B[1]
    x3 = C[0]
    y3 = C[1]
    x = point[0]
    y = point[1]
     # Calculate the cross products (c1, c2, c3) for the point relative to each␣
 ↪edge of the triangle
    c1 = (x2 - x1) * (y - y1) - (y2 - y1) * (x - x1)
    c2 = (x3 - x2) * (y - y2) - (y3 - y2) * (x - x2)
    c3 = (x1 - x3) * (y - y3) - (y1 - y3) * (x - x3)
    # Check if all cross products have the same sign (inside the triangle) or␣
 ↪different signs (outside the triangle)
    if (c1 < 0 and c2 < 0 and c3 < 0) or (c1 > 0 and c2 > 0 and c3 > 0):
        return True
    else:
        return False

def montecarlo(isInside, points, A,B,C):
    for point in points:
        if isInside(point,A,B,C):
            plt.scatter(point[0], point[1], color='r')

montecarlo(isInside,points,A,B,C)
```

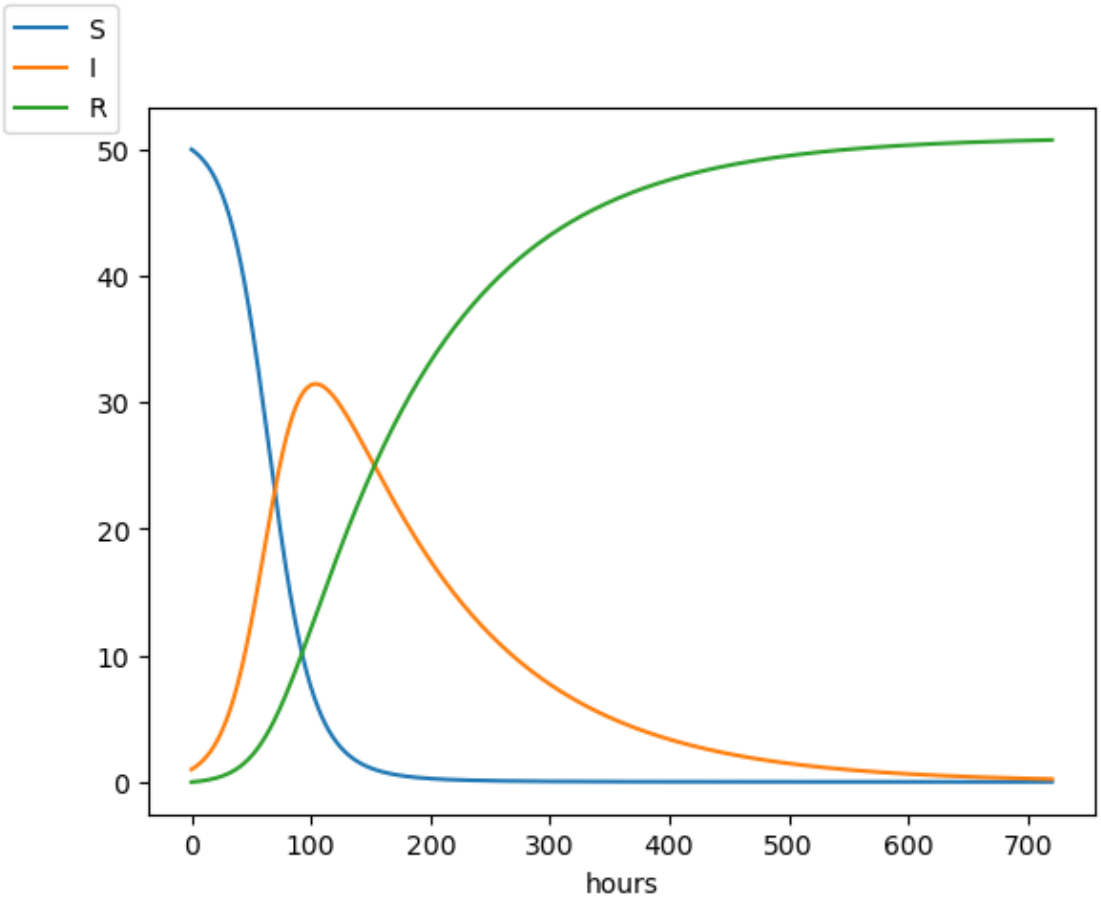


[21]:
```
import random
import numpy as np

a = 0
b = 1
n = 1000
points = np.random.uniform(a,b,n)

def func(x):
    return x**2 + 2*x

def montecarlo(func,points,a,b,n):
    integralsum = 0
    for point in points:
        integralsum += func(point)
    print((b-a)*integralsum/n)

montecarlo(func,points,a,b,n)
```

```
1.3379905552085618
```

[27]:
```
import numpy as np
import matplotlib.pyplot as plt

beta = 10./(40*8*24)
gamma = 3./(15*24)
dt = 0.1
D = 30
N_t = int(D*24/dt)

t = np.linspace(0,N_t*dt,N_t+1)
S = np.zeros(N_t+1)
I = np.zeros(N_t+1)
R = np.zeros(N_t+1)

S[0] = 50
I[0] = 1
R[0] = 0


for n in range(N_t):
    S[n+1] = S[n] - beta*S[n]*I[n]*dt
    I[n+1] = I[n] + beta*S[n]*I[n]*dt - gamma*I[n]*dt
    R[n+1] = R[n] + gamma*I[n]*dt

fig = plt.figure()
l1, l2, l3 = plt.plot(t, S, t, I, t, R)
fig.legend((l1, l2, l3), ('S', 'I', 'R'), 'upper left')
plt.xlabel('hours')
plt.show()
plt.savefig('tmp.pdf'); plt.savefig('tmp.png')
```

```
C:\Users\harry\AppData\Local\Temp\ipykernel_21180\1957819969.py:27:
MatplotlibDeprecationWarning: Passing the loc parameter of __init__()
positionally is deprecated since Matplotlib 3.6; the parameter will become
keyword-only two minor releases later.
  fig.legend((l1, l2, l3), ('S', 'I', 'R'), 'upper left')
```

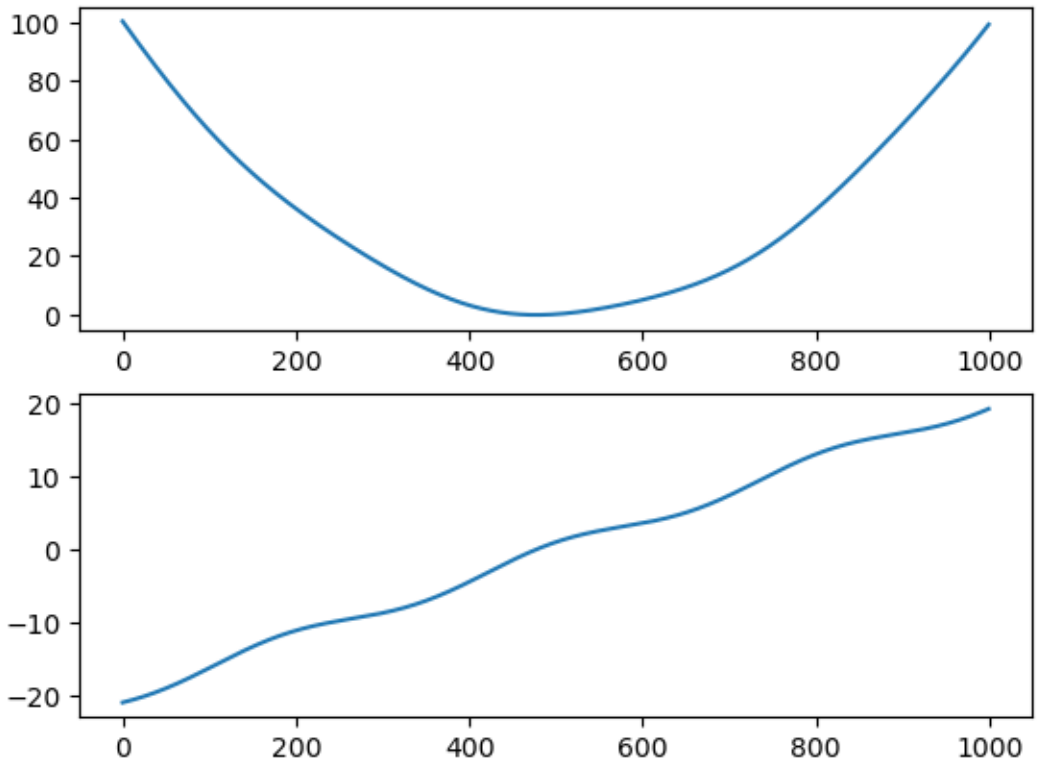


<Figure size 640x480 with 0 Axes>

[38]:
```
import numpy as np

def func(x):
    return x**2 + np.sin(x)

def tpd(func,x0,h=1e-6):
    return((func(x0+h)-func(x0-h))/(2*h))

x = np.linspace(-10,10,1000)
y = func(x)

dydx = tpd(func,x)

plt.subplot(2,1,1)
plt.plot(y)

plt.subplot(2,1,2)
```

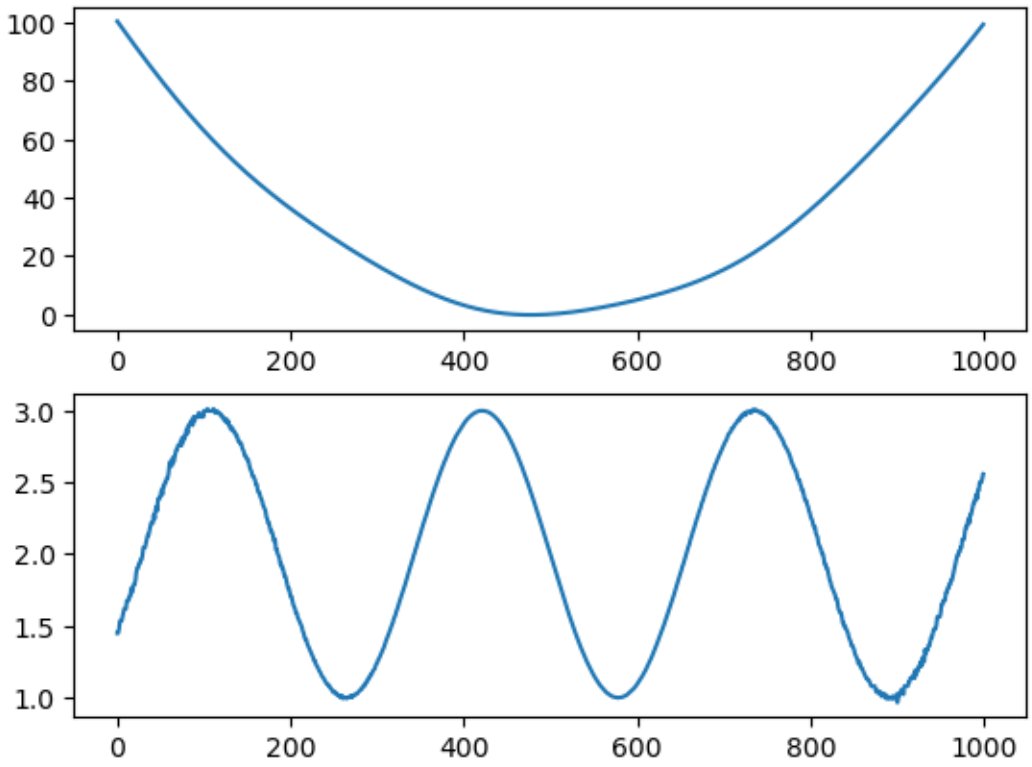
```
plt.plot(dydx)
```

[38]: [<matplotlib.lines.Line2D at 0x247a04c2650>]



[39]:
```
import numpy as np

def func(x):
    return x**2 + np.sin(x)

def der(func,x0,h=1e-6):
    return((func(x+h)+func(x-h)-2*func(x))/(h**2))

x = np.linspace(-10,10,1000)
y = func(x)

dydx = der(func,x)

plt.subplot(2,1,1)
plt.plot(y)

plt.subplot(2,1,2)
plt.plot(dydx)
```

[39]: [<matplotlib.lines.Line2D at 0x247a19c1dd0>]



[46]:
```
tol = 1e-6
n = 1000

def func(x):
    return x**2 - 4

def bisection(func,a,b,n,tol):
    if func(a)*func(b)>= 0:
        raise ValueError("Wrong a and b")
        return

    c = a
    for i in range(n):
        c = (a+b)/2
        if(func(c)==0 or abs(func(c))<tol):
            break

        elif(func(c)*func(a)<0):
            b = c
        else:
```

```
            a = c
    return c

print(bisection(func,0,3,n,tol))
```

```
2.000000238418579
```

[49]:
```
tol = 1e-6
n = 1000
x0 = 1.5

def func(x):
    return x**2 - 4

def der(x):
    return 2*x

def newton(func,der,x,tol,n):
    for i in range(n):
        if (abs(func(x))<tol):
            return x,i
        else:
            x = x-(func(x)/der(x))
    return x,i

root, num_itr = newton(func,der,x0,tol,n)
print(f"Root is approximately: {root} after {num_itr} iterations")
```

```
Root is approximately: 2.0000000000001203 after 4 iterations
```

[ ]:
```

```