Projet 3 Probabilité et Statistique: AKNOUCHE ANIS - HADDADI HACENE

1- DESCRIPTION DU MODELE :

 # 1-1 Matrice de transitions : M=([0.9, 0.1, 0] , [0 , 0.5, 0.5] , [0 , 0 , 1]) 1-2 Vérifier que la matrice de transition est stochastique : - Tous les coefficients de la matrice sont positifs - la somme de chaque ligne de la matrice est égale à 1 ligne 0 : 0.9 + 0.1 + 0 =1 ligne 1 : 0 + 0.5 + 0.5 =1 ligne 2 : 0 + 0 + 1 =1 Donc la matrice M de transition est stochastique 2- La distribution de la probabilité initiale : [0.9 , 0.1 , 0]

2- TIRAGE ALEATOIRE DES ETATS :

In [1]:

```
# Générer une séquence d'états aléatoire de taille n pour le modele SIR
import numpy as np
import random as rd

def generer_etats_aleatoire(n):
    """ en entrée: n : la taille de la séquence
        en sortie:  liste_etat : liste representant la sequence d'etat
                    seq : une chaine de charactere representant la sequence
                    proba : la probabilité de la séquence
                    k+1 : la taille de la séquence de sortie
                    p0 : la distribution de probabilité
    """
    #choix de l'état initial au hasard
    seq="" # la séquence d'etat
    liste_etat=[]
    matriceProba1=np.array([[0.9,0.1,0.0],[0.0,0.5,0.5],[0.0,0.0,1.0]])

    s=rd.randint(0,2)
    if(s==0):
        p0=[0.9,0.1,0.0]
        seq+="S "
        liste_etat.append('S')
    elif(s==1):
        p0=[0.0,0.5,0.5]
        seq+="I "
        liste_etat.append('I')
    elif(s==2):
        p0=[0.0,0.0,1]
        seq+="R "
        liste_etat.append('R')
    else:
        print("error")

    proba=1/3 # P(X0) probabilité qu'on soit dans un des états (S, I, R) au début
    matriceProba2=matriceProba1
    p1=p0
    i=s
    k=1
    for k in range(1,n):
        j=rd.randint(0,2)
        while(matriceProba1[i,j]==0):#pour eviter de génerer une séquence avec une probabilité n
ulle
            j=rd.randint(0,2)
        if(j==0):
            seq+="S "
            liste_etat.append('S')
        elif(j==1):
            seq+="I "
            liste_etat.append('I')
        elif(j==2):
            seq+="R "
            liste_etat.append('R')
        else:
            print("error")
```

```
        proba*=matriceProba1[i,j]   # proba de la séquence
        p0=np.dot(p1,matriceProba2) # la distribution de probabilité
        matriceProba2=np.dot(matriceProba2,matriceProba1)
        i=j


    return liste_etat, seq, proba, k+1, p0


for i in range(0,5): # on génére des sequences aléatoires de taille n=50

        liste_etat, seq, proba, k, p0=generer_etats_aleatoire(50)
        print("La séquence générée :"+seq)
        print("La proba de la séquence :"+str(proba))
        print("La distribution de probabilité après :"+str(k+1)+" périodes est de : "+str(p0))
        print(" ")
```

```
La séquence générée :R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R
R R R R R R R R R R R R R
La proba de la séquence :0.3333333333333333
La distribution de probabilité après :51 périodes est de : [0. 0. 1.]

La séquence générée :I I R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R
R R R R R R R R R R R R R
La proba de la séquence :0.08333333333333333
La distribution de probabilité après :51 périodes est de : [0.0000000e+00 8.8817842e-16 1.000000
0e+00]

La séquence générée :S S S I R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R
R R R R R R R R R R R R R
La proba de la séquence :0.013500000000000002
La distribution de probabilité après :51 périodes est de : [0.00515378 0.00128844 0.99355778]

La séquence générée :R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R
R R R R R R R R R R R R R
La proba de la séquence :0.3333333333333333
La distribution de probabilité après :51 périodes est de : [0. 0. 1.]

La séquence générée :S S I I R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R
R R R R R R R R R R R R R
La proba de la séquence :0.0075
La distribution de probabilité après :51 périodes est de : [0.00515378 0.00128844 0.99355778]
```

3- Probabilité d'une séquence :

1- La probabilité d'obtenir la séquence d'état : SSSSSSIIIIIIIIIIIR proba1=P(X0=S)*P(X1=S/X0=S)*P(X2=S/X1=S)*...*P(X6=S/X5=S)*P(X7=I/X6=S)*P(X8=I/X7=I)*...P(X17=I/X16=I)*P(X18=R/X17=I) proba1=(1/3)*0.9*0.9*0.9*0.9*0.9*0.1*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5*0.5 proba1=0.00000961083 2- La probabilité d'obtenir la séquence d'état : SSSSSSSSSSSSIIIR proba2=P(X0=S)*P(X1=S/X0=S)*P(X2=S/X1=S)*...*P(X11=S/X10=S)*P(X12=I/X11=S)*P(X13=I/X12=I)*P(X14=I/X13=I)*P(X15=R/X14=I) proba2=(1/3)*0.9*0.9*0.9*0.9*0.9*0.9*0.9*0.9*0.9*0.9*0.9*0.1*0.5*0.5*0.5 proba2=0.00065377207

4- Modélisation d'une population

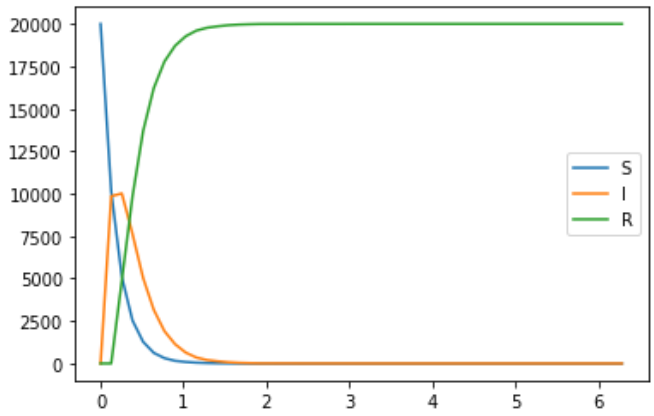
In [10]:

```
import matplotlib.pyplot as plt


def generer_etats_init_S(n):
    """ génére une sequence d'état à partir de l'état S (Sain)
        en entrée: n : la taille de la séquence
        en sortie:  liste_etat : liste representant la sequence d'etat
                    seq : une chaine de charactere representant la sequence
                    proba : la probabilité de la séquence
                    k+1 : la taille de la séquence de sortie
                    p0 : la distribution de probabilité
    """
    #choix de l'état initial au hasard
    seq="" # la séquence d'etat
    liste_etat=[]
```

```
    matriceProba1=np.array([[0.9,0.1,0.0],[0.0,0.5,0.5],[0.0,0.0,1.0]])

    s=0
    if(s==0):
        p0=[0.9,0.1,0.0]
        seq+="S "
        liste_etat.append('S')


    proba=1/3 # P(X0) probabilité qu'on soit dans un des états (S, I, R) au début
    matriceProba2=matriceProba1
    p1=p0
    i=s
    k=1
    for k in range(1,n):
        j=rd.randint(0,2)
        while(matriceProba1[i,j]==0):#pour eviter de génerer une séquence avec une probabilité n
ulle
            j=rd.randint(0,2)
        if(j==0):
            seq+="S "
            liste_etat.append('S')
        elif(j==1):
            seq+="I "
            liste_etat.append('I')
        elif(j==2):
            seq+="R "
            liste_etat.append('R')
        else:
            print("error")

        proba*=matriceProba1[i,j]  # proba de la séquence
        p0=np.dot(p1,matriceProba2) # la distribution de probabilité
        matriceProba2=np.dot(matriceProba2,matriceProba1)
        i=j


    return liste_etat, seq, proba, k+1, p0



# générer un ensemble de séquence
listeSeqPersonne=[] # la liste des ensembles d'etat de chaque personne
resultats=[]
listeEtatS=[]
listeEtatI=[]
listeEtatR=[]
n=50# taille de la sequence
nbPersonne=20000 # taille de la population
for i in range(0,nbPersonne):
    resultats=generer_etats_init_S(n)
    listeSeqPersonne.append(resultats[0])

matriceEtat=np.array(listeSeqPersonne)
for i in range(0,n):
    unique, counts = np.unique(matriceEtat[:,i], return_counts=True)
    dictio=dict(zip(unique, counts))
    if('S' in dictio):
        listeEtatS.append(dictio['S'])
    else:
        listeEtatS.append(0)
    if('I' in dictio):
        listeEtatI.append(dictio['I'])
    else:
        listeEtatI.append(0)
    if('R' in dictio):
        listeEtatR.append(dictio['R'])
    else:
        listeEtatR.append(0)
```

In [11]:

```
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatS, label="S")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatI, label="I")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatR, label="R")
plt.legend()
plt.show()
```



5- Modification du modèle :

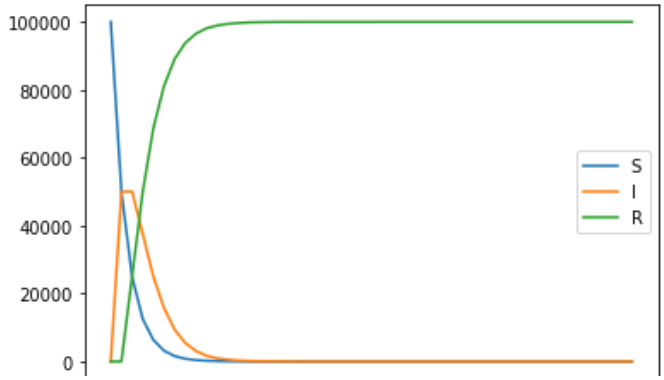
5.1- En modifiant la taille de la population :

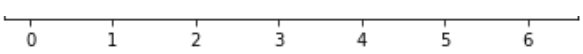
In [12]:

```
# en augmentant la taille de la population : pour n=100000 personnes
# générer un ensemble de séquence
listeSeqPersonne=[] # la liste des ensembles d'etat de chaque personne
resultats=[]
listeEtatS=[]
listeEtatI=[]
listeEtatR=[]
n=50# taille de la sequence
nbPersonne=100000 # taille de la population
for i in range(0,nbPersonne):
    resultats=generer_etats_init_S(n)
    listeSeqPersonne.append(resultats[0])

matriceEtat=np.array(listeSeqPersonne)
for i in range(0,n):
    unique, counts = np.unique(matriceEtat[:,i], return_counts=True)
    dictio=dict(zip(unique, counts))
    if('S' in dictio):
        listeEtatS.append(dictio['S'])
    else:
        listeEtatS.append(0)
    if('I' in dictio):
        listeEtatI.append(dictio['I'])
    else:
        listeEtatI.append(0)
    if('R' in dictio):
        listeEtatR.append(dictio['R'])
    else:
        listeEtatR.append(0)
```

In [13]:

```
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatS, label="S")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatI, label="I")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatR, label="R")
plt.legend()
plt.show()
```





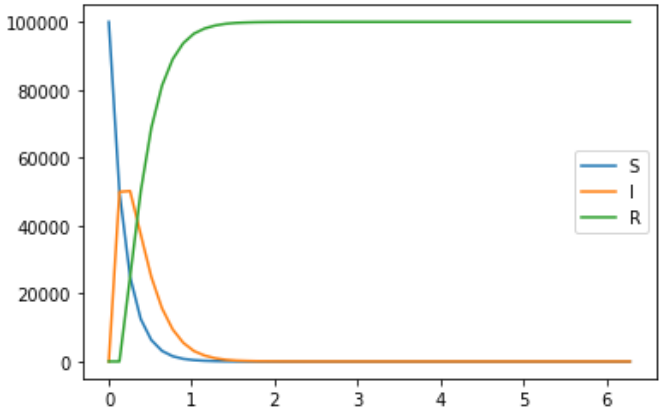
- Interprétation des résultats: Le résultat est proportionnel à la taille de la population, avec un pic d'individus infectés qui touche la moitié de la population comme pour la cas de n=20000 individus

In [14]:

```
# en deminuant la taille de la population : n=500 personnes
# générer un ensemble de séquence
listeSeqPersonne=[] # la liste des ensembles d'etat de chaque personne
resultats=[]
listeEtatS=[]
listeEtatI=[]
listeEtatR=[]
n=50# taille de la sequence
nbPersonne=100000 # taille de la population
for i in range(0,nbPersonne):
    resultats=generer_etats_init_S(n)
    listeSeqPersonne.append(resultats[0])

matriceEtat=np.array(listeSeqPersonne)
for i in range(0,n):
    unique, counts = np.unique(matriceEtat[:,i], return_counts=True)
    dictio=dict(zip(unique, counts))
    if('S' in dictio):
        listeEtatS.append(dictio['S'])
    else:
        listeEtatS.append(0)
    if('I' in dictio):
        listeEtatI.append(dictio['I'])
    else:
        listeEtatI.append(0)
    if('R' in dictio):
        listeEtatR.append(dictio['R'])
    else:
        listeEtatR.append(0)
```

In [15]:

```
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatS, label="S")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatI, label="I")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatR, label="R")
plt.legend()
plt.show()
```



- Interprétation des résultats: Meme chose que les résultats précédents, les résultats son proportionnels à la taille de la population

5.2 En modifiant la distribution de probabilité initiale

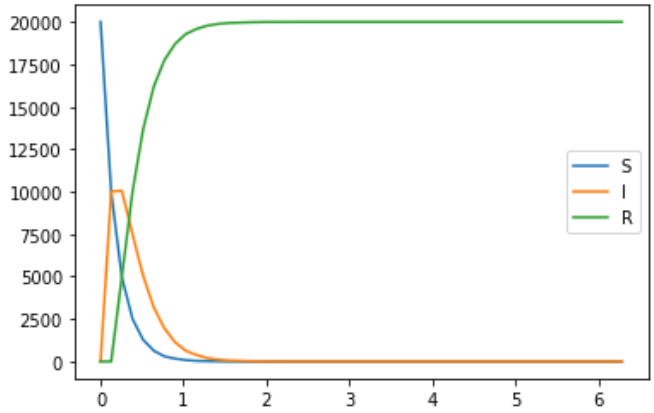
In [16]:

```
# dans le cas où tous les individus sont sains

# générer un ensemble de séquence
listeSeqPersonne=[] # la liste des ensembles d'etat de chaque personne
resultats=[]
listeEtatS=[]
listeEtatI=[]
listeEtatR=[]
```

```
n=50# taille de la sequence
nbPersonne=20000 # taille de la population
for i in range(0,nbPersonne):
    resultats=generer_etats_init_S(n)
    listeSeqPersonne.append(resultats[0])

matriceEtat=np.array(listeSeqPersonne)
for i in range(0,n):
    unique, counts = np.unique(matriceEtat[:,i], return_counts=True)
    dictio=dict(zip(unique, counts))
    if('S' in dictio):
        listeEtatS.append(dictio['S'])
    else:
        listeEtatS.append(0)
    if('I' in dictio):
        listeEtatI.append(dictio['I'])
    else:
        listeEtatI.append(0)
    if('R' in dictio):
        listeEtatR.append(dictio['R'])
    else:
        listeEtatR.append(0)
```

In [17]:

```
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatS, label="S")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatI, label="I")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatR, label="R")
plt.legend()
plt.show()
```



- Interprétation des résultats: le nombre d'individus sains déminue jusqu'à ce qu'il n y ait plus d'individu sain, et le nombre de cas infectés augmente jusqu'à ce que la moitié de la population soit infectée où la courbe atteint son pic de cas infectés, au meme moment le nombre de cas guéri augmente jusqu'à ce que toute la population soit guérie, et le nombre de cas infecté a déminué dès qu'il avait atteint son pic.

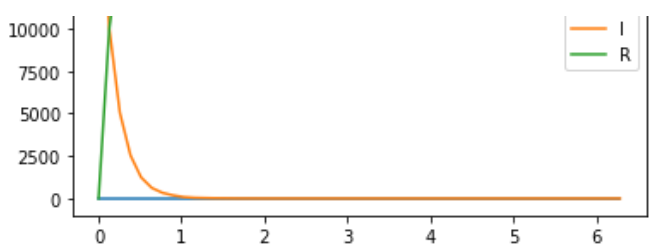
In [18]:

```
# Dans le cas où tous les individus sont inféctés
import matplotlib.pyplot as plt


def generer_etats_init_I(n):
    """ génére une sequence d'état à partir de l'état I (Infecte)
        en entrée: n : la taille de la séquence
        en sortie:  liste_etat : liste representant la sequence d'etat
                    seq : une chaine de charactere representant la sequence
                    proba : la probabilité de la séquence
                    k+1 : la taille de la séquence de sortie
                    p0 : la distribution de probabilité
    """
    #choix de l'état initial au hasard
    seq="" # la séquence d'etat
    liste_etat=[]
    matriceProba1=np.array([[0.9,0.1,0.0],[0.0,0.5,0.5],[0.0,0.0,1.0]])

    s=1
    if(s==1):
        p0=[0.0,0.5,0.5]
        seq+="I "
        liste_etat.append('I')
```

```
    proba=1/3 # P(X0) probabilité qu'on soit dans un des états (S, I, R) au début
    matriceProba2=matriceProba1
    p1=p0
    i=s
    k=1
    for k in range(1,n):
        j=rd.randint(0,2)
        while(matriceProba1[i,j]==0):#pour eviter de génerer une séquence avec une probabilité n
ulle
            j=rd.randint(0,2)
        if(j==0):
            seq+="S "
            liste_etat.append('S')
        elif(j==1):
            seq+="I "
            liste_etat.append('I')
        elif(j==2):
            seq+="R "
            liste_etat.append('R')
        else:
            print("error")

        proba*=matriceProba1[i,j]  # proba de la séquence
        p0=np.dot(p1,matriceProba2) # la distribution de probabilité
        matriceProba2=np.dot(matriceProba2,matriceProba1)
        i=j


    return liste_etat, seq, proba, k+1, p0



# générer un ensemble de séquence
listeSeqPersonne=[] # la liste des ensembles d'etat de chaque personne
resultats=[]
listeEtatS=[]
listeEtatI=[]
listeEtatR=[]
n=50# taille de la sequence
nbPersonne=20000 # taille de la population
for i in range(0,nbPersonne):
    resultats=generer_etats_init_I(n)
    listeSeqPersonne.append(resultats[0])

matriceEtat=np.array(listeSeqPersonne)
for i in range(0,n):
    unique, counts = np.unique(matriceEtat[:,i], return_counts=True)
    dictio=dict(zip(unique, counts))
    if('S' in dictio):
        listeEtatS.append(dictio['S'])
    else:
        listeEtatS.append(0)
    if('I' in dictio):
        listeEtatI.append(dictio['I'])
    else:
        listeEtatI.append(0)
    if('R' in dictio):
        listeEtatR.append(dictio['R'])
    else:
        listeEtatR.append(0)
```

In [19]:

```
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatS, label="S")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatI, label="I")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatR, label="R")
plt.legend()
plt.show()
```



- Interprétation des résultats: On remarque que le nombre d'individus infectés diminue jusqu'à atteindre 0, et le nombre d'individus guéris augmente de 0 jusqu'à ce que toute la population soit guérie. le nombre d'individus sains reste nul.

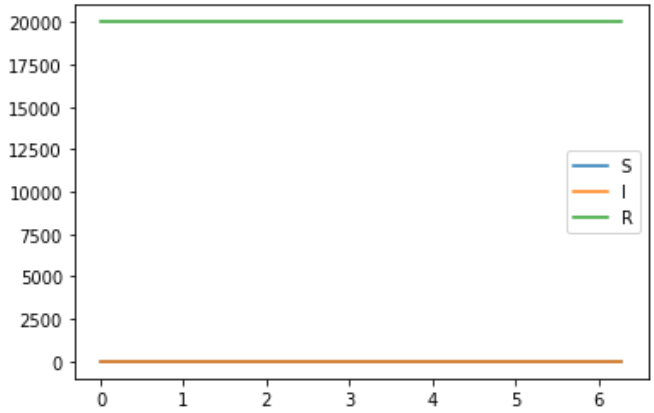
In [20]:

```
# Dans le cas où tous les individus sont Guéris
import matplotlib.pyplot as plt


def generer_etats_init_R(n):
    """ génére une séquence d'état à partir de l'état R (Guéri)
        en entrée: n : la taille de la séquence
        en sortie:  liste_etat : liste representant la sequence d'etat
                    seq : une chaine de charactere representant la sequence
                    proba : la probabilité de la séquence
                    k+1 : la taille de la séquence de sortie
                    p0 : la distribution de probabilité
    """
    #choix de l'état initial au hasard
    seq="" # la séquence d'etat
    liste_etat=[]
    matriceProba1=np.array([[0.9,0.1,0.0],[0.0,0.5,0.5],[0.0,0.0,1.0]])

    s=2
    if(s==2):
        p0=[0.0,0.0,1.0]
        seq+="R "
        liste_etat.append('R')


    proba=1/3 # P(X0) probabilité qu'on soit dans un des états (S, I, R) au début
    matriceProba2=matriceProba1
    p1=p0
    i=s
    k=1
    for k in range(1,n):
        j=rd.randint(0,2)
        while(matriceProba1[i,j]==0):#pour eviter de génerer une séquence avec une probabilité n
ulle
            j=rd.randint(0,2)
        if(j==0):
            seq+="S "
            liste_etat.append('S')
        elif(j==1):
            seq+="I "
            liste_etat.append('I')
        elif(j==2):
            seq+="R "
            liste_etat.append('R')
        else:
            print("error")

        proba*=matriceProba1[i,j]  # proba de la séquence
        p0=np.dot(p1,matriceProba2) # la distribution de probabilité
        matriceProba2=np.dot(matriceProba2,matriceProba1)
        i=j


    return liste_etat, seq, proba, k+1, p0



# générer un ensemble de séquence
listeSeqPersonne=[] # la liste des ensembles d'etat de chaque personne
resultats=[]
listeEtatS=[]
listeEtatI=[]
listeEtatR=[]
```

```
n=50# taille de la sequence
nbPersonne=20000 # taille de la population
for i in range(0,nbPersonne):
    resultats=generer_etats_init_R(n)
    listeSeqPersonne.append(resultats[0])

matriceEtat=np.array(listeSeqPersonne)
for i in range(0,n):
    unique, counts = np.unique(matriceEtat[:,i], return_counts=True)
    dictio=dict(zip(unique, counts))
    if('S' in dictio):
        listeEtatS.append(dictio['S'])
    else:
        listeEtatS.append(0)
    if('I' in dictio):
        listeEtatI.append(dictio['I'])
    else:
        listeEtatI.append(0)
    if('R' in dictio):
        listeEtatR.append(dictio['R'])
    else:
        listeEtatR.append(0)
```

In [21]:

```
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatS, label="S")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatI, label="I")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatR, label="R")
plt.legend()
plt.show()
```



-Interprétation des résultats : De l'état guéri on ne peut pas aller vers un autre état donc le nombre d'individus guéris est constant et représente toute la population, ainsi le nombre de cas sains et infectés sont nuls.

5.3 En modifiant les probabilités de transition :

-Representation du cas extreme 1: -En modifiant la probabilité d'un individu sain de devenir infécté (Si l'on met des mesures de distanciations sociales un individu sain a moins de chance de devenir infecté), on peut trés bien voir le résultat si la probabilité qu'un individu sain devienne infecté tende vers 0 et la probabilité de rester sain tende vers 1 Voici la nouvelle distribution de probabilité d'un individu sain : [1.0, 0.0, 0.0]

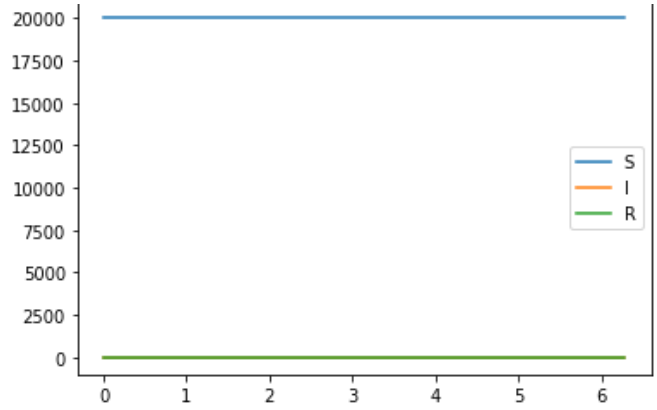
In [22]:

```
# Dans le cas où tous les individus sont Sains
import matplotlib.pyplot as plt


def generer_etats_init_S(n):
    """ génére une sequence d'état à partir de l'état S (Sain)
        en entrée: n : la taille de la séquence
        en sortie:  liste_etat : liste representant la sequence d'etat
                    seq : une chaine de charactere representant la sequence
                    proba : la probabilité de la séquence
                    k+1 : la taille de la séquence de sortie
                    p0 : la distribution de probabilité
    """
    #choix de l'état initial au hasard
    seq="" # la séquence d'etat
    liste_etat=[]
    matriceProba1=np.array([[1,0.0,0.0],[0.0,0.5,0.5],[0.0,0.0,1.0]])
```

```
    s=0
    if(s==0):
        p0=[1.0,0.0,0.0]
        seq+="S "
        liste_etat.append('S')


    proba=1/3 # P(X0) probabilité qu'on soit dans un des états (S, I, R) au début
    matriceProba2=matriceProba1
    p1=p0
    i=s
    k=1
    for k in range(1,n):
        j=rd.randint(0,2)
        while(matriceProba1[i,j]==0):#pour eviter de génerer une séquence avec une probabilité n
ulle
            j=rd.randint(0,2)
        if(j==0):
            seq+="S "
            liste_etat.append('S')
        elif(j==1):
            seq+="I "
            liste_etat.append('I')
        elif(j==2):
            seq+="R "
            liste_etat.append('R')
        else:
            print("error")

        proba*=matriceProba1[i,j]  # proba de la séquence
        p0=np.dot(p1,matriceProba2) # la distribution de probabilité
        matriceProba2=np.dot(matriceProba2,matriceProba1)
        i=j


    return liste_etat, seq, proba, k+1, p0



# générer un ensemble de séquence
listeSeqPersonne=[] # la liste des ensembles d'etat de chaque personne
resultats=[]
listeEtatS=[]
listeEtatI=[]
listeEtatR=[]
n=50# taille de la sequence
nbPersonne=20000 # taille de la population
for i in range(0,nbPersonne):
    resultats=generer_etats_init_S(n)
    listeSeqPersonne.append(resultats[0])

matriceEtat=np.array(listeSeqPersonne)
for i in range(0,n):
    unique, counts = np.unique(matriceEtat[:,i], return_counts=True)
    dictio=dict(zip(unique, counts))
    if('S' in dictio):
        listeEtatS.append(dictio['S'])
    else:
        listeEtatS.append(0)
    if('I' in dictio):
        listeEtatI.append(dictio['I'])
    else:
        listeEtatI.append(0)
    if('R' in dictio):
        listeEtatR.append(dictio['R'])
    else:
        listeEtatR.append(0)
```

In [23]:

```
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatS, label="S")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatI, label="I")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatR, label="R")
plt.legend()
plt.show()
```



-Interprétation des résultats : On remarque que le nombre de cas sains reste constant meme après plusieures périodes de temps, c'est le résultat d'un conffinement total de toute la population, si aucune personne n'est en contacte avec une autre personne, la probabilité d'etre infecté est nulle.-Représentation du cas extreme 2: -En modifiant la probabilité d'un individu sain de devenir infécté (Si l'on ne respecte aucunes mesures de distanciations sociales, un individu sain a plus de chance de devenir infecté, on peut trés bien voir le résultat si la probabilité qu'un individu sain devienne infecté tende vers 1 et la probabilité de rester sain tende vers 0) Voici la nouvelle distribution de probabilité d'un individu sain : [0.0, 1.0, 0.0]

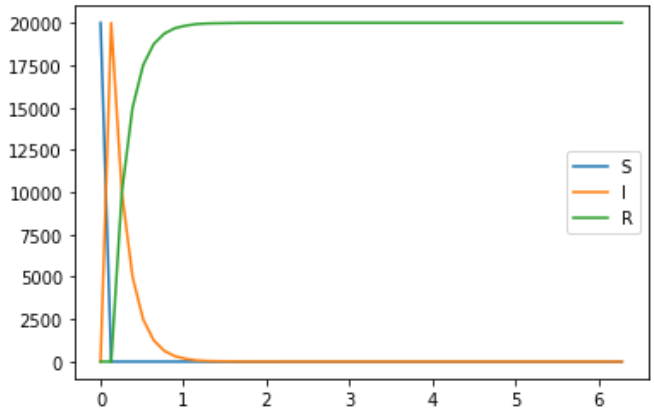
In [24]:

```
# Dans le cas où tous les individus sont Sains
import matplotlib.pyplot as plt
import math

def generer_etats_init_S(n):
    """ génére une sequence d'état à partir de l'état S (Sain)
        en entrée: n : la taille de la séquence
        en sortie:  liste_etat : liste representant la sequence d'etat
                    seq : une chaine de charactere representant la sequence
                    proba : la probabilité de la séquence
                    k+1 : la taille de la séquence de sortie
                    p0 : la distribution de probabilité
    """
    #choix de l'état initial au hasard
    seq="" # la séquence d'etat
    liste_etat=[]
    matriceProba1=np.array([[0.0,1.0,0.0],[0.0,0.5,0.5],[0.0,0.0,1.0]])

    s=0
    if(s==0):
        p0=[0.0,1.0,0.0]
        seq+="S "
        liste_etat.append('S')


    proba=1/3 # P(X0) probabilité qu'on soit dans un des états (S, I, R) au début
    matriceProba2=matriceProba1
    p1=p0
    i=s
    k=1
    for k in range(1,n):
        j=rd.randint(0,2)
        while(matriceProba1[i,j]==0):#pour eviter de génerer une séquence avec une probabilité n
ulle
            j=rd.randint(0,2)
        if(j==0):
            seq+="S "
            liste_etat.append('S')
        elif(j==1):
            seq+="I "
            liste_etat.append('I')
        elif(j==2):
            seq+="R "
            liste_etat.append('R')
        else:
            print("error")

        proba*=matriceProba1[i,j]  # proba de la séquence
        p0=np.dot(p1,matriceProba2) # la distribution de probabilité
        matriceProba2=np.dot(matriceProba2,matriceProba1)
        i=j
```

```
    return liste_etat, seq, proba, k+1, p0


# générer un ensemble de séquence
listeSeqPersonne=[] # la liste des ensembles d'etat de chaque personne
resultats=[]
listeEtatS=[]
listeEtatI=[]
listeEtatR=[]
n=50# taille de la sequence
nbPersonne=20000 # taille de la population
for i in range(0,nbPersonne):
    resultats=generer_etats_init_S(n)
    listeSeqPersonne.append(resultats[0])

matriceEtat=np.array(listeSeqPersonne)
for i in range(0,n):
    unique, counts = np.unique(matriceEtat[:,i], return_counts=True)
    dictio=dict(zip(unique, counts))
    if('S' in dictio):
        listeEtatS.append(dictio['S'])
    else:
        listeEtatS.append(0)
    if('I' in dictio):
        listeEtatI.append(dictio['I'])
    else:
        listeEtatI.append(0)
    if('R' in dictio):
        listeEtatR.append(dictio['R'])
    else:
        listeEtatR.append(0)
```

In [25]:

```
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatS, label="S")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatI, label="I")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatR, label="R")
plt.legend()
plt.show()
```



-Interprétation des résultats : Le nombre d'individus sains converger plus rapidement vers 0, puisque la chance de devenir infecté est plus grande (dans cet exemple elle vaut 1), on remarque aussi que le pic de cas infectés à rapidement évoluer, en moins d'un temps toute la population a été infecté.

5.4 Si un individu guéri peut devenir infecté :

 # Dans le cas ou un individu guéri peut devenir infecté, si il est en contact avec un individu infecté, donc on peut supposer que ce sont les memes probabilités pour un individu sain de devenir infecté: On pose la distribution de probabilités d'un individu guéri: [0.0, 0.1, 0.9]

In [27]:

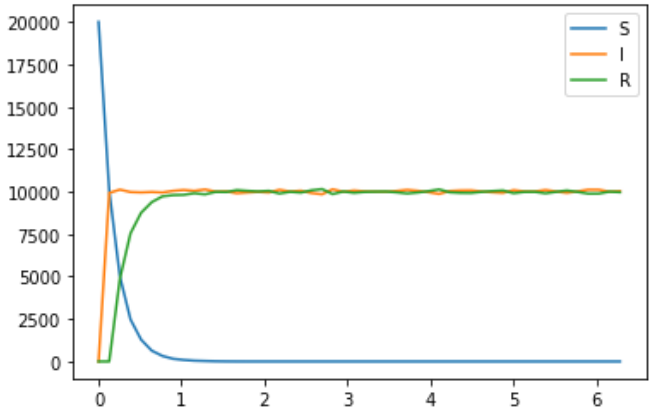
```
# Dans le cas où tous les individus sont Sains
import matplotlib.pyplot as plt
import math

def generer_etats_init_S(n):
    """ génére une sequence d'état à partir de l'état S (Sain)
```

```
        en entrée: n : la taille de la séquence
        en sortie:  liste_etat : liste representant la sequence d'etat
                    seq : une chaine de charactere representant la sequence
                    proba : la probabilité de la séquence
                    k+1 : la taille de la séquence de sortie
                    p0 : la distribution de probabilité
    """
    #choix de l'état initial au hasard
    seq="" # la séquence d'etat
    liste_etat=[]
    matriceProba1=np.array([[0.9,0.1,0.0],[0.0,0.5,0.5],[0.0,0.1,0.9]])

    s=0
    if(s==0):
        p0=[0.1,0.9,0.0]
        seq+="S "
        liste_etat.append('S')


    proba=1/3 # P(X0) probabilité qu'on soit dans un des états (S, I, R) au début
    matriceProba2=matriceProba1
    p1=p0
    i=s
    k=1
    for k in range(1,n):
        j=rd.randint(0,2)
        while(matriceProba1[i,j]==0):#pour eviter de génerer une séquence avec une probabilité n
ulle
            j=rd.randint(0,2)
        if(j==0):
            seq+="S "
            liste_etat.append('S')
        elif(j==1):
            seq+="I "
            liste_etat.append('I')
        elif(j==2):
            seq+="R "
            liste_etat.append('R')
        else:
            print("error")

        proba*=matriceProba1[i,j]  # proba de la séquence
        p0=np.dot(p1,matriceProba2) # la distribution de probabilité
        matriceProba2=np.dot(matriceProba2,matriceProba1)
        i=j


    return liste_etat, seq, proba, k+1, p0



# générer un ensemble de séquence
listeSeqPersonne=[] # la liste des ensembles d'etat de chaque personne
resultats=[]
listeEtatS=[]
listeEtatI=[]
listeEtatR=[]
n=50# taille de la sequence
nbPersonne=20000 # taille de la population
for i in range(0,nbPersonne):
    resultats=generer_etats_init_S(n)
    listeSeqPersonne.append(resultats[0])

matriceEtat=np.array(listeSeqPersonne)
for i in range(0,n):
    unique, counts = np.unique(matriceEtat[:,i], return_counts=True)
    dictio=dict(zip(unique, counts))
    if('S' in dictio):
        listeEtatS.append(dictio['S'])
    else:
        listeEtatS.append(0)
    if('I' in dictio):
        listeEtatI.append(dictio['I'])
    else:
        listeEtatI.append(0)
    if('R' in dictio):
        listeEtatR.append(dictio['R'])
```

```
    else:
        listeEtatR.append(0)
```

In [28]:

```
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatS, label="S")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatI, label="I")
plt.plot(np.linspace(0, 2*np.pi, n),listeEtatR, label="R")
plt.legend()
plt.show()
```



-Interprétation des résultats: le nombre d'individus sains déminue rapidement pour atteindre 0. Le nombre d'individus infectés à atteint la moitié de la population et est devenu constant, puis le nombre d'individus guéris a aussi augmenté pour atteindre la moitié de la population et devenir constant. le nombre de cas infectés et le nombre de cas guéris se chauvauchent et convergent vers la moitié du nombre de la population.

5.5 Critiques du modèle SIR et Améliorations :

-Entre l'état Sain 'S' et l'état Infecté 'I', pour certaines maladies, lorsque un individu sain est y exposé il faut parfois un certain temps avant que l'individu ne soit infecté, ce qui est appelée période de latence, du coup en devrait rajouter un autre état Exposé 'E' entre l'état S et I. -Après qu'un individu a été infecté, il peut décèder, représenté par l'état Décès 'D'. La maladie peut se terminer d'elle même et donner à l'individu une immunisation contre la réinfection, représenté par l'état Guéri 'R'. Dans le cas contraire, l'individu peut toujours être infectieux, il se retrouve isolé de la population par politique de quarantaine, représenté par l'état Quarantaine 'Q', sans négliger le fait que ce dernier peut éventuellement guérir et réintégrer l'état Sain 'S'.

In [ ]:

```
```