ANNEXE A : 1ER Modèle simple

```python
from random import *
from numpy import *
from matplotlib.pyplot import *
from numpy import linalg as LA
#constantes : lambda1 (paramètre de la Fatt) et alpha (paramètre de la Frep)
def MODELE_SIMPLE(lambda1,alpha):
    #Nombre d'ite´ration
    n=600
    #Nombre de Bo¨ıds
    nbr=2
    #Position initiale
    A0=array([-10,3])
    B0=array([10,0])
    #initialisation des matrices de stockage de position
    A=zeros((2,n))
    B=zeros((2,n))
    #initialisation de la vitesse initiale aléatoire
    j1=random.random()
    j2=random.random()
    V=array([[cos(j1*2*pi),cos(j2*2*pi)],[sin(j1*2*pi),sin(j2*2*pi)]])
    #Affectation des positions initiales
    A[:,0]=A0
    B[:,0]=B0
    #Boucle de calcul des positions succesives
    for i in range(n-1):
        V[:,0]=V[:,0]+(lambda1-(4*alpha)/LA.norm(A[:,i]-B[:,i])**2)*(B[:,i]-A[:,i])/2
        V[:,1]=V[:,1]+(lambda1-(4*alpha)/LA.norm(A[:,i]-B[:,i])**2)*(A[:,i]-B[:,i])/2
        A[:,i+1]=A[:,i]+V[ :,0]
        B[:,i+1]=B[:,i]+V[ :,1]

    #Affichage direct des trajectoires
    plot(A[0,:],A[1,:],'r')
    plot(B[0,:],B[1,:],"b")
    show()
```

Annexe B : Décroissance de vitesse

```python
from random import *
from numpy import *
from matplotlib.pyplot import *
from numpy import linalg as LA
#Coefficient de la décroissance des vitesses noté par K >1
def DECROISSANCE_VITESSE(lambda1,alpha,K):
    #Nombre d'ite´ration
    n=600
    #Nombre de Boids
    nbr=2
    #Position initiale
    A0=array([-10,3])
    B0=array([10,0])
    #initialisation des matrices de stockage de position
    A=zeros((2,n))
    B=zeros((2,n))
    #initialisation de la vitesse initiale aléatoire
    j1=random.random()
    j2=random.random()
    V=array([[cos(j1*2*pi),cos(j2*2*pi)],[sin(j1*2*pi),sin(j2*2*pi)]])
    #Affectation des positions initiales
    A[:,0]=A0
    B[:,0]=B0
    #Boucle de calcul des positions succesives
    for i in range(n-1):
        V[:,0]=V[ :,0]/K+(lambda1-(4*alpha)/LA.norm(A[:,i]-B[:,i])**2)*(B[:,i]-A[:,i])/2
        V[:,1]=V[ :,1]/K+(lambda1-(4*alpha)/LA.norm(A[:,i]-B[:,i])**2)*(A[:,i]-B[:,i])/2
        A[:,i+1]=A[:,i]+V[ :,0]
        B[:,i+1]=B[ :,i]+V[ :,1]

    #Affichage direct des trajectoires
    plot(A[0,:],A[1,:],'r')
    plot(B[0,:],B[1,:],"b")
    show()
```

Annexe C : Vitesse Limite

```
from random import *
from numpy import *
from matplotlib.pyplot import *
from numpy import linalg as LA
def VITESSE_LIMITE(lambda1,lambda2,K,Vlim):
    #Nombre d'itération
    n=600
    #Nombre de Boids
    nbr=2
    #Position initiale
    A0=array([-10,3])
    B0=array([10,0])
    #initialisation des matrices de stockage de position
    A=zeros((2,n))
    B=zeros((2,n))
    #initialisation de la vitesse initiale aléatoire
    j1=random.random()
    j2=random.random()
    V=array([[cos(j1*2*pi),cos(j2*2*pi)],[sin(j1*2*pi),sin(j2*2*pi)]])
    #Affectation des positions initiales
    A[:,0]=A0
    B[ :,0]=B0
    #direction de la vitesse limite
    Ulim=Vlim/LA.norm(Vlim)
    #Boucle de calcul des positions succesives
    for i in range(n-1):
        if LA.norm(V[:,0])> LA.norm(Vlim) or LA.norm(V[:,1])> LA.norm(Vlim):
            V[:,0]=V[ :,0]/K+(lambda1-(4*alpha)/LA.norm(A[:,i]-B[:,i])**2)*(B[:,i]-A[:,i])/2
            V[:,1]=V[ :,1]/K+(lambda1-(4*alpha)/LA.norm(A[:,i]-B[:,i])**2)*(A[:,i]-B[:,i])/2
            A[:,i+1]=A[:,i]+V[ :,0]
            B[:,i+1]=B[ :,i]+V[ :,1]
        elif LA.norm(V[:,0])<= LA.norm(Vlim) or LA.norm(V[:,1])<= LA.norm(Vlim):
            V[:,0]=V[ :,0]/K+(lambda1-(4*alpha)/LA.norm(A[:,i]-B[:,i])**2)*(B[:,i]-A[:,i])/2+[LA.norm(Vlim)-LA.norm(V[:,0])]*Ulim
            V[:,1]=V[ :,1]/K+(lambda1-(4*alpha)/LA.norm(A[:,i]-B[:,i])**2)*(A[:,i]-B[:,i])/2 + [LA.norm(Vlim)-LA.norm(V[:,0])]*Ulim
            A[:,i+1]=A[:,i]+V[ :,0]
            B[:,i+1]=B[ :,i]+V[ :,1]
    #Affichage direct des trajectoires
    plot(A[0,:],A[1,:],'r')
    plot(B[0,:],B[1,:],"b")
    show()
```

Annexe D : Généralisation des forces

```python
import numpy as np
import matplotlib.pyplot as plt
from math import *
def FattrGaussA(lambda1,lambda2,R0):
    t=np.linspace(0,10,1000)
    Fattr = lambda1*np.exp(-(((t/2)-R0/2)/lambda2)**2)
    plt.plot(t,Fattr)
    plt.xlabel("distance")
    plt.ylabel("intensité de la force")
    plt.show()
```

```
from random import *
from matplotlib.pyplot import *
from numpy import linalg as LA
import numpy as np
def GENERALISATION_FORCES(lambda1,lambda2,alpha,R0,K,Vlim):
    n=600
    nbr=2
    A0=array([-10,3])
    B0=array([10,0])
    A=zeros((2,n))
    B=zeros((2,n))
    j1=random.random()
    j2=random.random()
    V=array([[cos(j1*2*pi),cos(j2*2*pi)],[sin(j1*2*pi),sin(j2*2*pi)]])
    Ulim=Vlim/LA.norm(Vlim)
    A[:,0]=A0
    B[:,0]=B0
    for i in range(n-1):
        distAB=(B[:,i]-A[:,i])/2
        distBA=(A[:,i]-B[:,i])/2
        FattrAB=lambda1*np.exp(-((LA.norm(distAB)-R0/2)/lambda2)**2)*distAB/LA.norm(distAB)#Force d'attraction en A par B
        FattrBA=lambda1*np.exp(-((LA.norm(distBA)-R0/2)/lambda2)**2)*distBA/LA.norm(distBA)#Force d'attraction en B par A
        #on annule la répulsion pour une certaines distance
        if LA.norm(distAB)<10 :
            FrepAB=-(alpha/LA.norm(distAB)**2)*distAB
        else :
            FrepAB=0
        if LA.norm(distBA)<10 :
            FrepBA=-(alpha/LA.norm(distBA)**2)*distBA
        else:
            FrepBA=0
        V[:,0]=V[:,0]/k+FattrAB+FrepAB
        V[:,1]=V[:,1]/k+FattrBA+FrepBA
        if LA.norm(V[:,0])> LA.norm(Vlim) or LA.norm(V[:,1])> LA.norm(Vlim):
            V[:,0]=V[:,0]/K+FattrAB+ FrepAB
            V[:,1]=V[:,1]/K+FattrBA+ FrepBA
            A[:,i+1]=A[:,i]+V[:,0]
            B[:,i+1]=B[:,i]+V[:,1]
        elif LA.norm(V[:,0])<= LA.norm(Vlim) or LA.norm(V[:,1])<= LA.norm(Vlim):
            V[:,0]=V[:,0]/K+FattrAB+ FrepAB + [LA.norm(Vlim)-LA.norm(V[:,0])]*Ulim
            V[:,1]=V[:,1]/K+FattrBA+ FrepBA + [LA.norm(Vlim)-LA.norm(V[:,0])]*Ulim
            A[:,i+1]=A[:,i]+V[:,0]
            B[:,i+1]=B[:,i]+V[:,1]
        A[:,i+1]=A[:,i]+V[:,0]
        B[:,i+1]=B[:,i]+V[:,1]
    plot(A[0,:],A[1,:],"r")
    plot(B[0,:],B[1,:],"b")
    show()
```

Annexe E : Force extérieur

```python
from random import *
from matplotlib.pyplot import *
from numpy import linalg as LA
import numpy as np
#constantes : lambdap( paramètre de la Force (poteau) en position xp,yp
def FORCES_POTEAU(lambda1,lambda2,alpha,R0,K,Vlim,lambdap,xp,yp):
    n=600
    nbr=2
    A0=array([-10,3])
    B0=array([10,0])
    A=zeros((2,n))
    B=zeros((2,n))
    j1=random()
    j2=random()
    V=array([[cos(j1*2*pi),cos(j2*2*pi)],[sin(j1*2*pi),sin(j2*2*pi)]])
    Ulim=Vlim/LA.norm(Vlim)
    A[:,0]=A0
    B[:,0]=B0
    for i in range(n-1):
        distAB=(B[:,i]-A[:,i])/2
        distBA=(A[:,i]-B[:,i])/2
        FattrAB=lambda1*np.exp(-((LA.norm(distAB)-R0/2)/lambda2)**2)*distAB/LA.norm(distAB)#Force d'attraction en A par B
        FattrBA=lambda1*np.exp(-((LA.norm(distBA)-R0/2)/lambda2)**2)*distBA/LA.norm(distBA)#Force d'attraction en B par A
        P0=array([xp,yp])
        distAP=(P0-A[:,i])
        distBP=(P0-B[:,i])
        #on annule la répulsion pour une certaines distance
        if LA.norm(distAB)<10 :
            FrepAB=-(alpha/LA.norm(distAB)**2)*distAB
        else :
            FrepAB=0
        if LA.norm(distBA)<10 :
            FrepBA=-(alpha/LA.norm(distBA)**2)*distBA
        else:
            FrepBA=0
        #On annule l'influence de l'obstacle pour une certaines distance
        if LA.norm(distAP)<10 :
            poteauAP=-(lambdap/LA.norm(distAP)**2)*distAP
        else :
            poteauAP=0
        if LA.norm(distBP)<10 :
           poteauBP=-(lambdap/LA.norm(distBP)**2)*distBP
        else:
            poteauBP=0
        V[:,0]=V[:,0]/k+FattrAB+FrepAB +poteauAP
        V[:,1]=V[:,1]/k+FattrBA+FrepBA +poteauBP
        if LA.norm(V[:,0])> LA.norm(Vlim) or LA.norm(V[:,1])> LA.norm(Vlim):
            V[:,0]=V[:,0]/K+ FattrAB + FrepAB + poteauAP
            V[:,1]=V[:,1]/K+ FattrBA + FrepBA + poteauBP
            A[:,i+1]=A[:,i]+V[:,0]
            B[:,i+1]=B[:,i]+V[:,1]
        elif LA.norm(V[:,0])<= LA.norm(Vlim) or LA.norm(V[:,1])<= LA.norm(Vlim):
            V[:,0]=V[:,0]/K+FattrAB+ FrepAB + (LA.norm(Vlim)-LA.norm(V[:,0]))*Ulim
            V[:,1]=V[:,1]/K+FattrBA+ FrepBA + (LA.norm(Vlim)-LA.norm(V[:,0]))*Ulim
            A[:,i+1]=A[:,i]+V[:,0]
            B[:,i+1]=B[:,i]+V[:,1]
    plot(A[0,:],A[1,:],"r")
    plot(B[0,:],B[1,:],"b")
    plot(xp,yp,"v")
    show()
```

Annexe F : Modèle général

```
from matplotlib.pyplot import *
from numpy import linalg as LA
import numpy as np
from numpy import *
def MODELE_GENERAL(n,nbr,lambda1,lambda2,alpha,R0,K,Vlim,lambdap,xp,yp): #nbr = nombre de boids et n =nombre d'itération
    #Conditions initiales
    Boids0=20*np.random.rand(2,nbr)
    Vboids0=0.2*np.random.rand(2,nbr)
    #Initialisation
    Boids=zeros((2,n,nbr))
    Vboids=zeros((2,nbr))
    Fatt=zeros((2,nbr))
    Frep=zeros((2,nbr))
    Boids[:,0,:]=Boids0
    Vboids=Vboids0
    P0=array([xp,yp])#position du poteau
    Vinfty=array([1,Vlim])#Vlim vitesse à partir de laquel on tend vers Vinfty
    for i in range (n-1) : # i nombre de tour
        Fatt=zeros((2,nbr)) #on remet à zéro à chaque tour
        Frep=zeros((2,nbr))
        for j in range (nbr):# j numéro de boids
            k=0
            while (k<nbr):
                if(k!=j): # on cumule les interractions pour chaques boids
                    dist=Boids[:,i,k]-Boids[:,i,j]
                    norme=LA.norm(dist)
                    Fatt[:,j]=Fatt[:,j]+lambda1*np.exp(-((norme-R0/2)/lambda2)**2)*(dist/norme)
                    Frep[:,j]=Frep[:,j]-(alpha/norme)*(dist/norme)
                k=k+1
            distp=P0-Boids[:,i,j]
            if LA.norm(distp)<5:
                FpoteaujP=-(lambdap/LA.norm(distp)**2)*distp
            else :
                FpoteaujP =0
            Vboids[:,j]=Vboids[:,j]/K+Frep[:,j]+Fatt[:,j]+ FpoteaujP #mise en place de la vitesse
            if(LA.norm(Vboids[:,j])<Vlim): #recherche d'une vitesse inférieur à vlim
                Vboids[:,j]=Vboids[:,j]+[Vlim-LA.norm(Vboids[:,j])]*Vinfty
            Boids[:,i+1,j]=Boids[:,i,j]+Vboids[:,j]#calcul de la position
            plot(Boids[0,0 :i,j],Boids[1,0 :i,j],linewidth=0.1) #affichage des boids
    if lambdap!=0 :
        plot(xp,yp,"v")
    show()
```

Annexe G : Modèle 3D

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.pyplot import *
from numpy import linalg as LA
import numpy as np
from numpy import *
def MODELE_3D(n,nbr,lambda1,lambda2,alpha,R0,K,Vlim,lambdap,xp,yp,zp): #nbr = nombre de boids et n =nombre d'itération
    #Conditions initiales
    Boids0=20*np.random.rand(3,nbr)
    Vboids0=0.2*np.random.rand(3,nbr)
    #Initialisation
    Boids=zeros((3,n,nbr))
    Vboids=zeros((3,nbr))
    Fatt=zeros((3,nbr))
    Frep=zeros((3,nbr))
    Boids[:,0,:]=Boids0
    Vboids=Vboids0
    P0=array([xp,yp,zp])#position du poteau
    Vinfty=array([0,0,Vlim])#Vlim vitesse à partir de laquel on tend vers Vinfty
    for i in range (n-1) : # i nombre de tour
        Fatt=zeros((3,nbr)) #on remet à zéro à chaque tour
        Frep=zeros((3,nbr))
        theta1=random.rand()
        phi1=random.rand()
        if(random.rand()>0.5):# ajout d'une vitesse de direction aléatoire
            Vinfty=Vinfty+array([sin(pi*theta1)*cos(2*pi*phi1),sin(pi*theta1)*cos(2*pi*phi1),cos(pi*theta1)])/2
            Vinfty=Vinfty/LA.norm(Vinfty)
        for j in range (nbr):# j numéro de boids
            k=0
            while (k<nbr):
                if(k!=j): # on cumule les interractions pour chaques boids
                    dist=Boids[:,i,k]-Boids[:,i,j]
                    norme=LA.norm(dist)
                    Fatt[:,j]=Fatt[:,j]+lambda1*np.exp(-((norme-R0/2)/lambda2)**2)*(dist/norme)
                    Frep[:,j]=Frep[:,j]-(alpha/norme)*(dist/norme)
                k=k+1
            distp=P0-Boids[:,i,j]
            if LA.norm(distp)<5:
                FpoteaujP=-(lambdap/LA.norm(distp)**2)*distp
            else :
                FpoteaujP =0
            Vboids[:,j]=Vboids[:,j]/K+Frep[:,j]+Fatt[:,j]+ FpoteaujP #mise en place de la vitesse
            if(LA.norm(Vboids[:,j])<Vlim):#recherche d'une vitesse inférieur à Vlim
                theta2=random.rand()
                phi2=random.rand()
                if(random.rand()>0.5):
                    Vboids[:,j]=Vboids[:,j]+array([Vlim-LA.norm(Vboids[:,j])])*array([Vinfty+array([sin(pi*theta2)*cos(2*pi*phi2),sin(pi*theta2)*cos(2*pi*phi2),cos(pi*theta2)])])/2
                else:
                    Vboids[:,j]=Vboids[:,j]+[Vlim-LA.norm(Vboids[:,j])]*Vinfty
            Boids[:,i+1,j]=Boids[:,i,j]+Vboids[:,j]#calcul de la position
            gca(projection='3d').plot(Boids[0,0 :i,j],Boids[1,0 :i,j],Boids[2,0 :i,j],"--",linewidth=0.5)
    if lambdap!=0 :
        gca(projection='3d').plot([xp],[yp],[zp],"v")
    show()
```