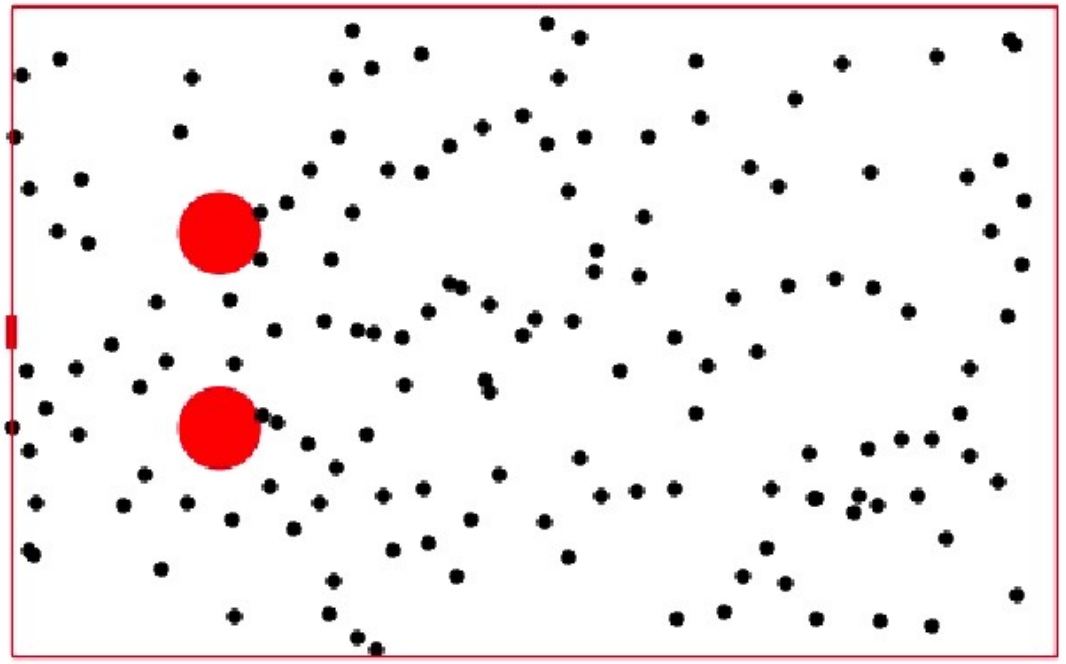
Modélisation et optimisation d'évacuations de personnes



Antoine Germain
Candidat n°14029

Sommaire

Introduction

Introduction

Dangers des mouvements de foule :

Lieux à forte densité de piétons :
manifestations, festivals, rues passantes…

https://www.letelegramme.fr/monde/la-mecque-plus-de-deux-millions-de-musulmans-entament-le-pelerinage-09-08-2019-12357281.php

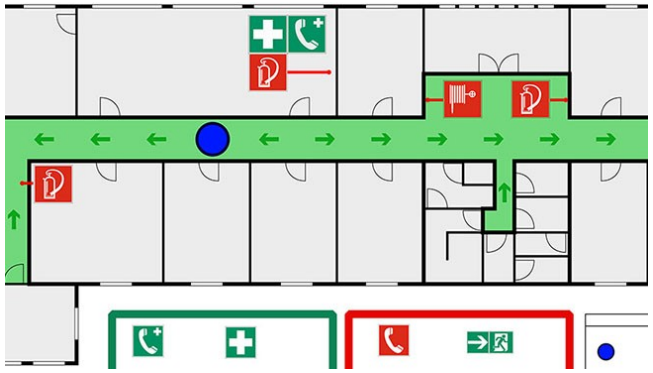
Introduction

Dangers des mouvements de foule :

Lieux à forte densité de piétons :
manifestations, festivals, rues passantes…

https://www.letelegramme.fr/monde/la-mecque-plus-de-deux-millions-de-musulmans-entament-le-pelerinage-09-08-2019-12357281.php

https://www.groupe-api-incendie.fr/plan-evacuation/



Évacuation d'urgence des bâtiments :
incendies, attaques terroristes,
bombardements…

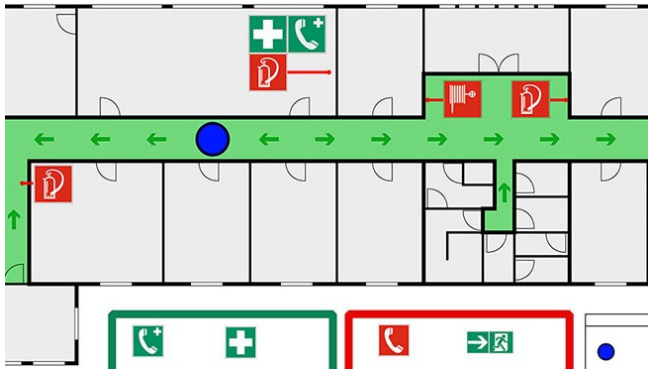
Introduction

Dangers des mouvements de foule :

Lieux à forte densité de piétons :
manifestations, festivals, rues passantes…

https://www.letelegramme.fr/monde/la-mecque-plus-de-deux-millions-de-musulmans-entament-le-pelerinage-09-08-2019-12357281.php

https://www.groupe-api-incendie.fr/plan-evacuation/



Évacuation d'urgence des bâtiments :
incendies, attaques terroristes,
bombardements…

Intérêts des modélisations : - comprendre les accidents passés
- prévenir les futurs accidents

Caractéristiques du modèle

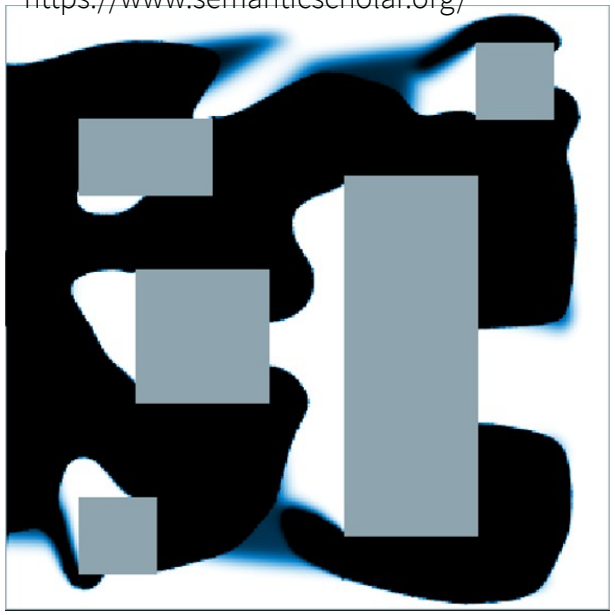
Caractéristiques du modèle

Modèles macroscopiques :

Foule considérée dans son ensemble.

Utilisation de la mécanique des fluides.

https://www.semanticscholar.org/



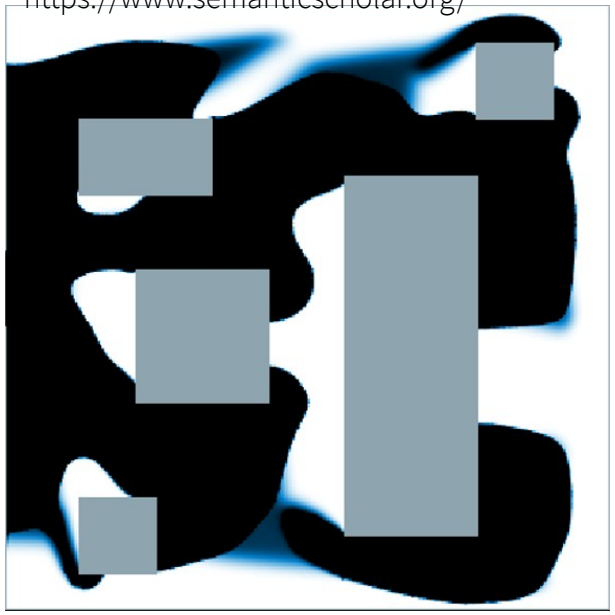
Caractéristiques du modèle

Modèles macroscopiques :

Foule considérée dans son ensemble.

Utilisation de la mécanique des fluides.

https://www.semanticscholar.org/

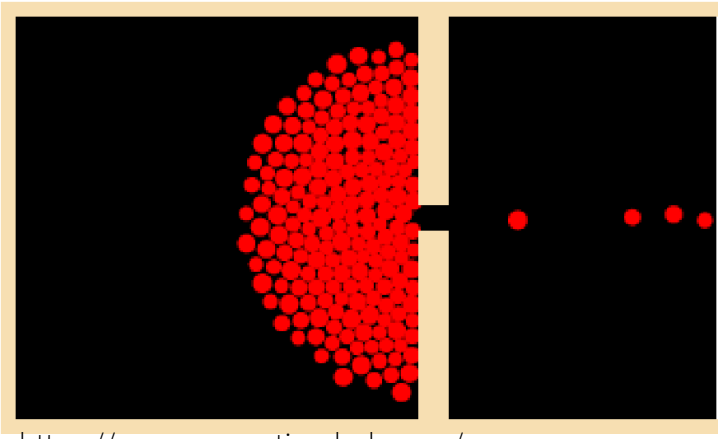


Modèles microscopiques :

Piétons considérés individuellement.

Utilisation de la mécanique classique,

d'un automate cellulaire, du gradient…



https://www.semanticscholar.org/

Caractéristiques du modèle

Choix d'un modèle microscopique : foule de n piétons.

Caractéristiques du modèle

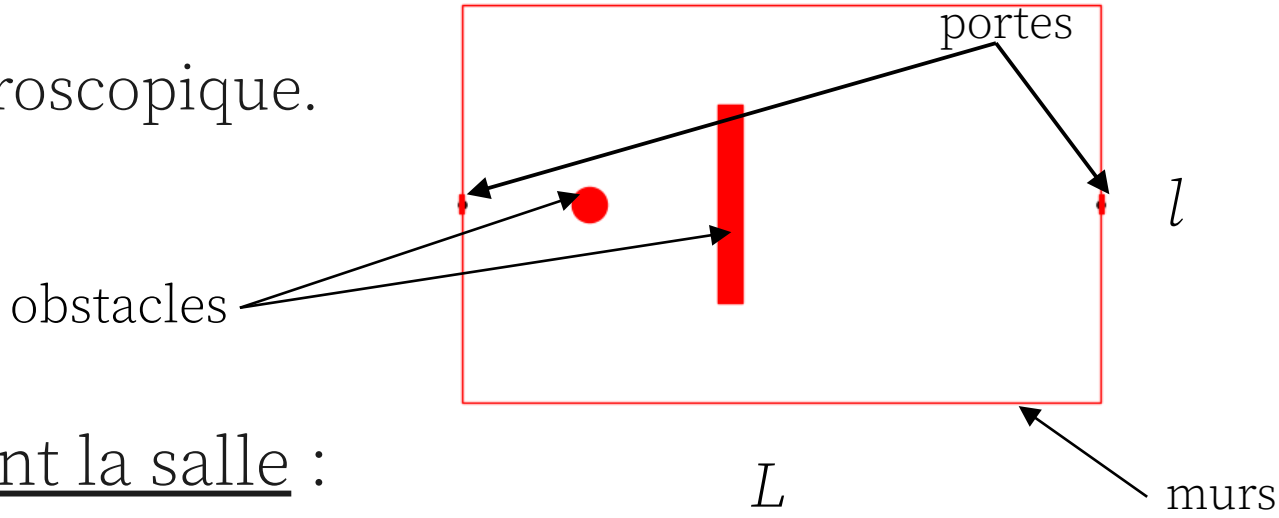
Choix d'un modèle microscopique : foule de n piétons.

Hypothèses concernant le piéton :

- Vitesse de marche constante
- Pas de traversée d'obstacles
- Pas de chevauchement entre piétons
- Connaissance du chemin le plus court pour atteindre la sortie

Caractéristiques du modèle

Choix d'un modèle microscopique.



<u>Hypothèses concernant la salle</u> :

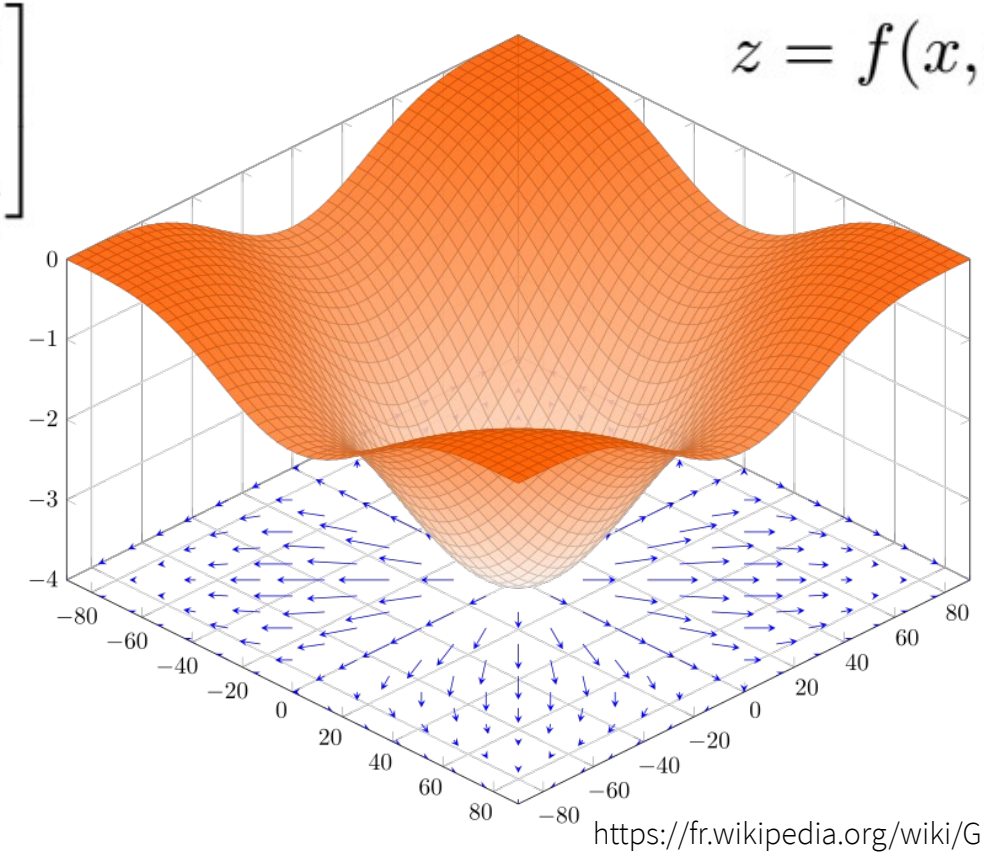
- Salle rectangulaire de longueur L et de largeur l
- Une ou plusieurs portes : objectifs des piétons
- Murs et obstacles infranchissables
- Ralentissements à proximité des murs négligés

Méthode de descente de gradient

Méthode de descente de gradient

- Fonction vectorielle : $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$

- Gradient dans la direction de plus grande pente de f :

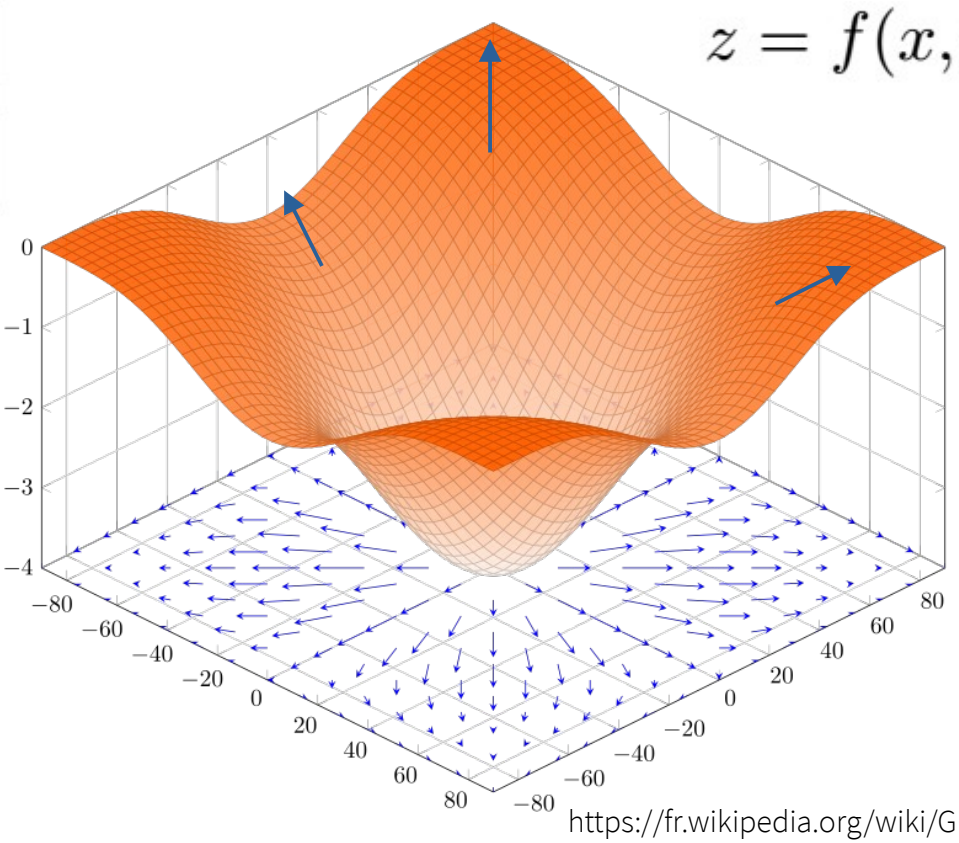


https://fr.wikipedia.org/wiki/Gradient

Méthode de descente de gradient

- Fonction vectorielle : $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$

- Gradient dans la direction de plus grande pente de f :



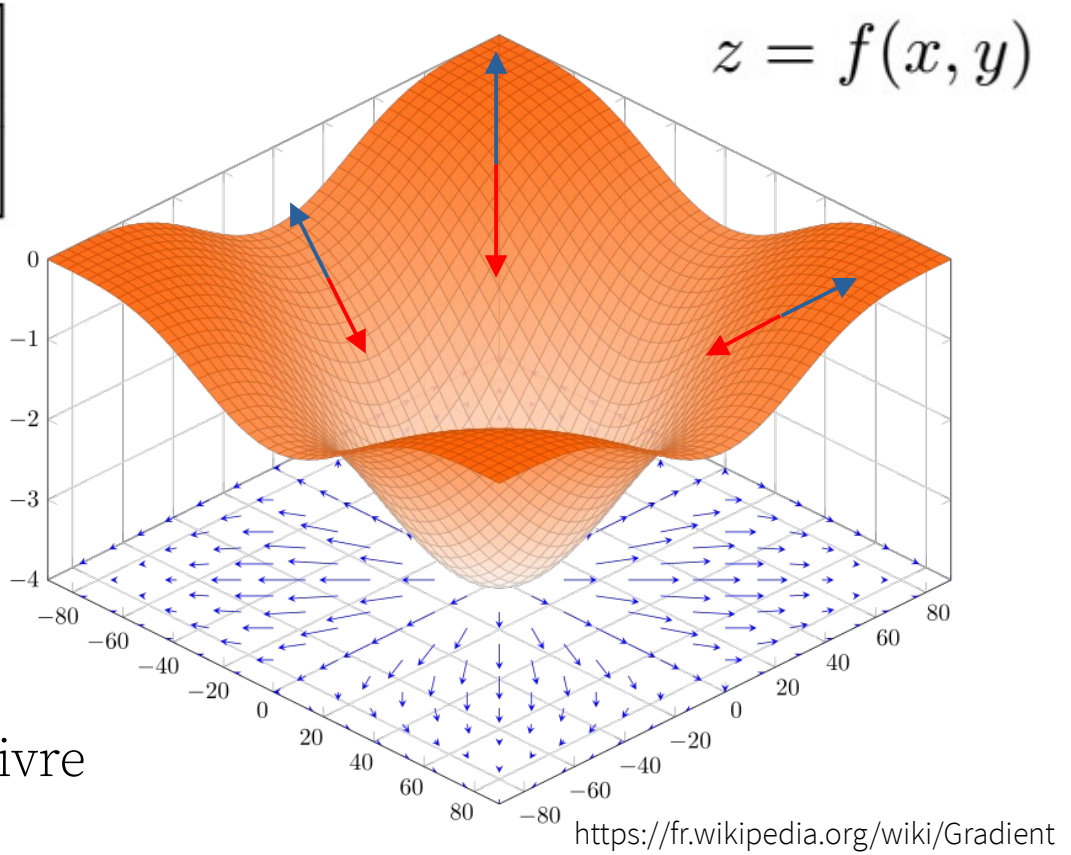
→ Gradient

https://fr.wikipedia.org/wiki/Gradient

Méthode de descente de gradient

- Fonction vectorielle : $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$

- Gradient dans la direction de plus grande pente de f :



Gradient

Direction à suivre

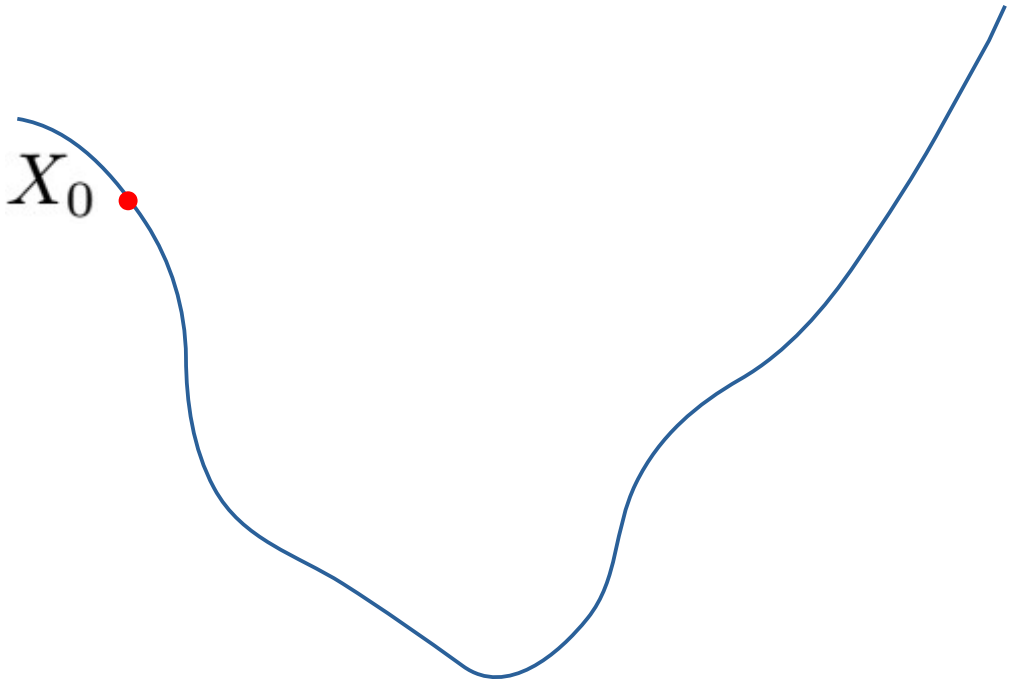
https://fr.wikipedia.org/wiki/Gradient

- Pour trouver le minimum d'une fonction, se diriger dans le sens inverse du gradient.

Méthode de descente de gradient

Algorithme de descente de gradient :

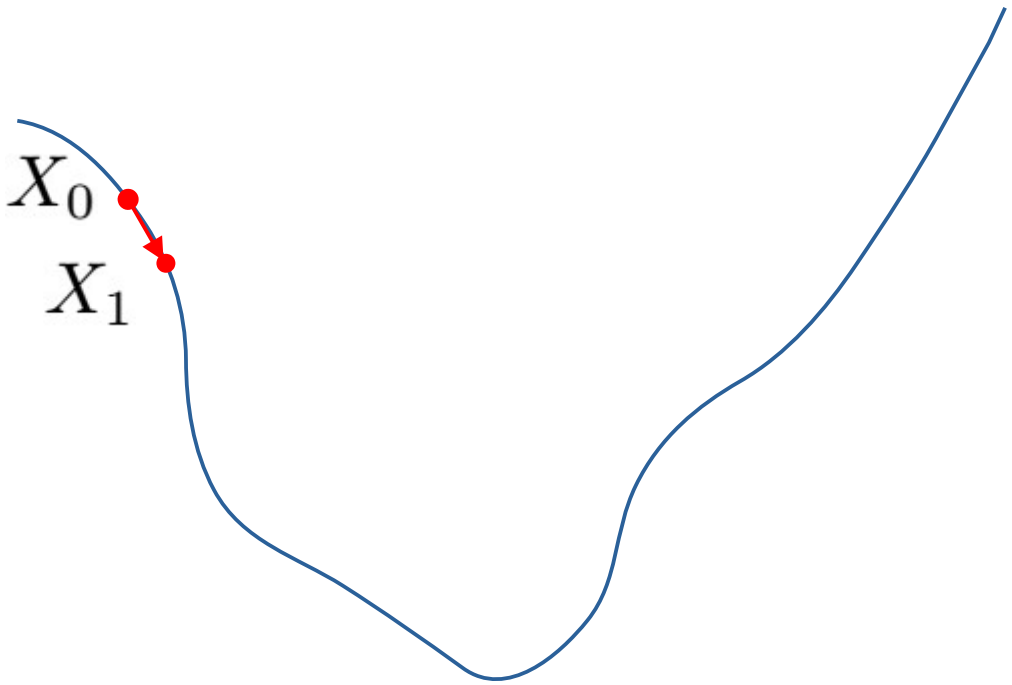
But : trouver un minimum de $f : (x_1, ..., x_n) \longmapsto f(x_1, ..., x_n)$



Méthode de descente de gradient

Algorithme de descente de gradient :

But : trouver un minimum de $f : (x_1, ..., x_n) \longmapsto f(x_1, ..., x_n)$

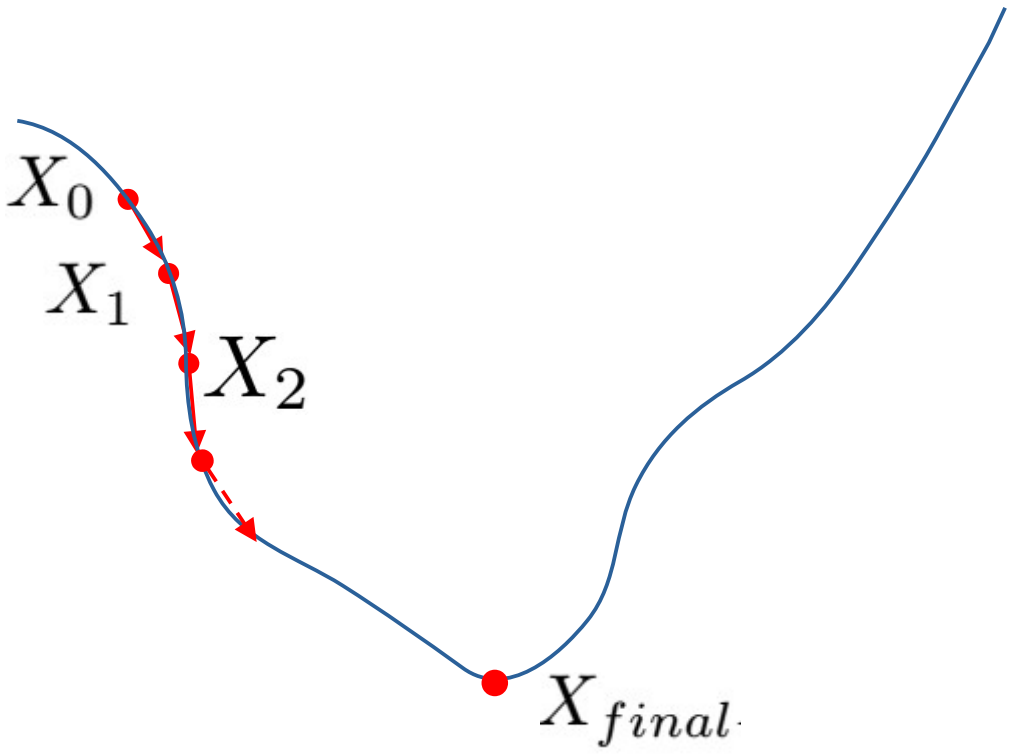


Calcul de ∇f en X_0.
Déplacement dans le sens inverse de ∇f : point X_1.

Méthode de descente de gradient

Algorithme de descente de gradient :

But : trouver un minimum de $f : (x_1, ..., x_n) \longmapsto f(x_1, ..., x_n)$



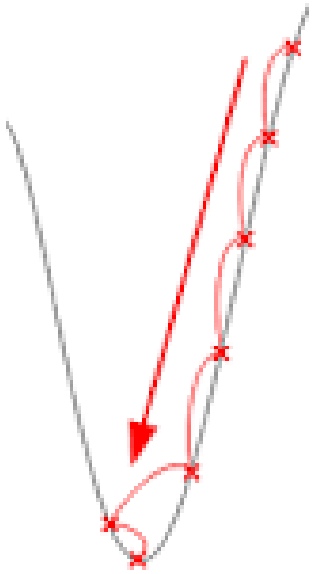
Calcul de ∇f en X_0.
Déplacement dans le sens inverse de ∇f : point X_1.

En itérant, on atteint X_{final} tel que le gradient de f en ce point soit presque nul.

Méthode de descente de gradient

Avantages : - implémentation simple

- localisation du minimum global d'une fonction convexe



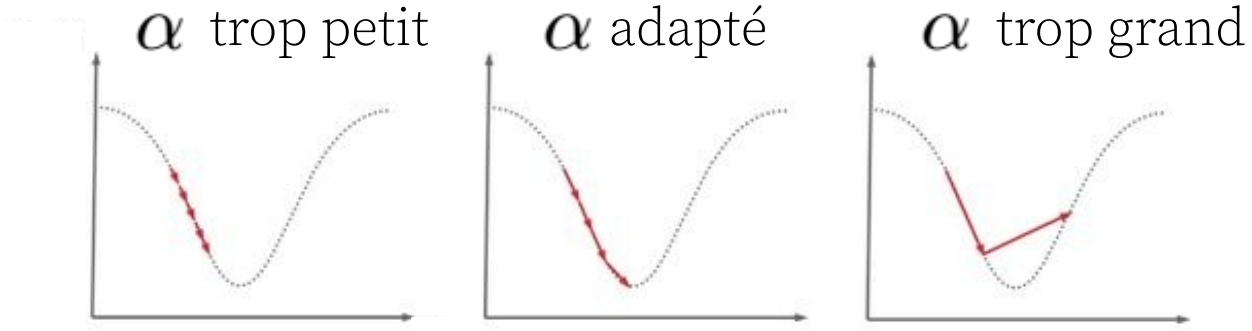
Méthode de descente de gradient

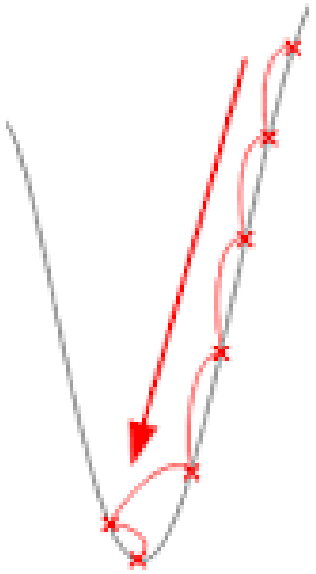
Avantages : - implémentation simple

- localisation du minimum global d'une fonction convexe

Inconvénients : - pas α constant

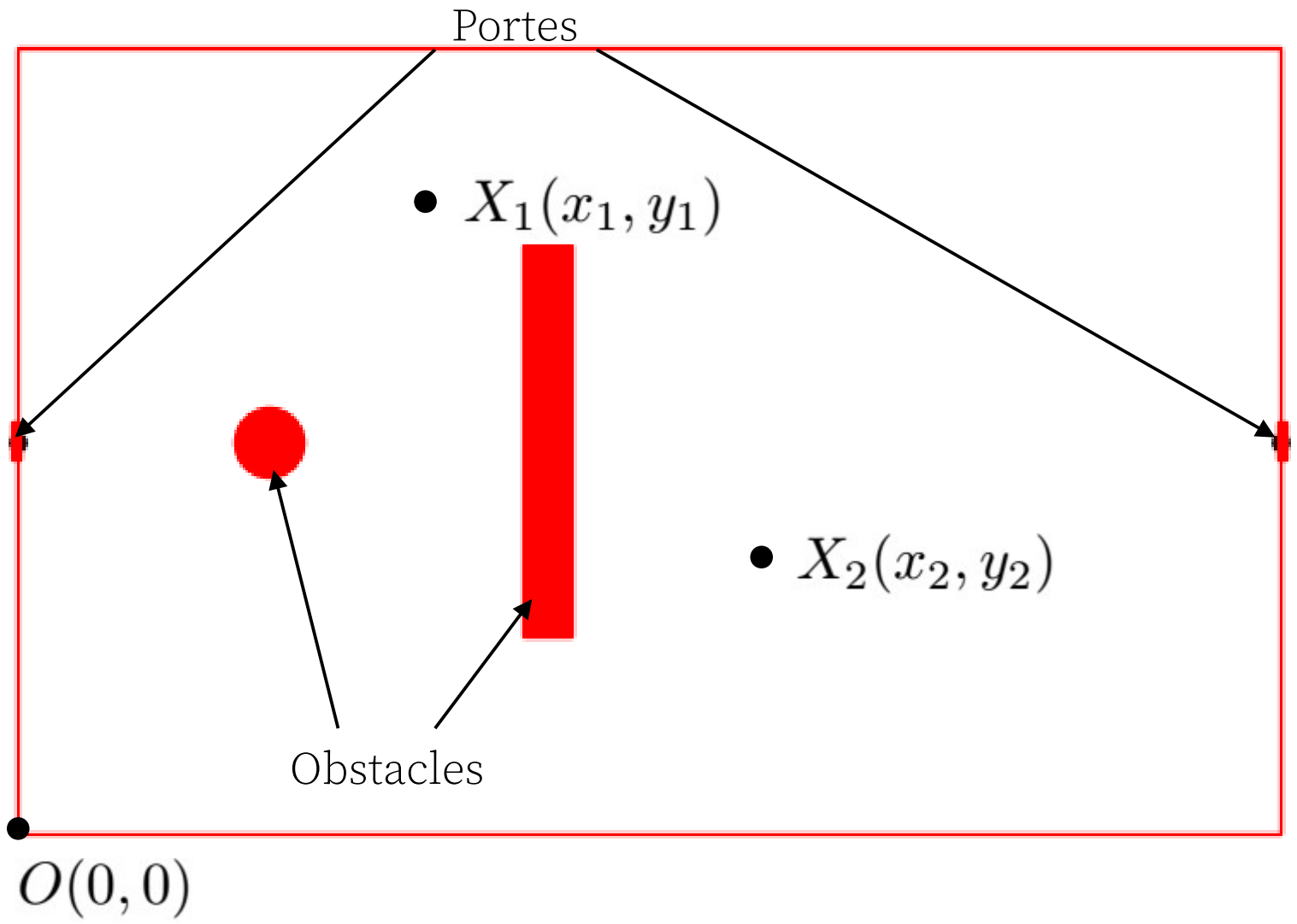
- convergence lente



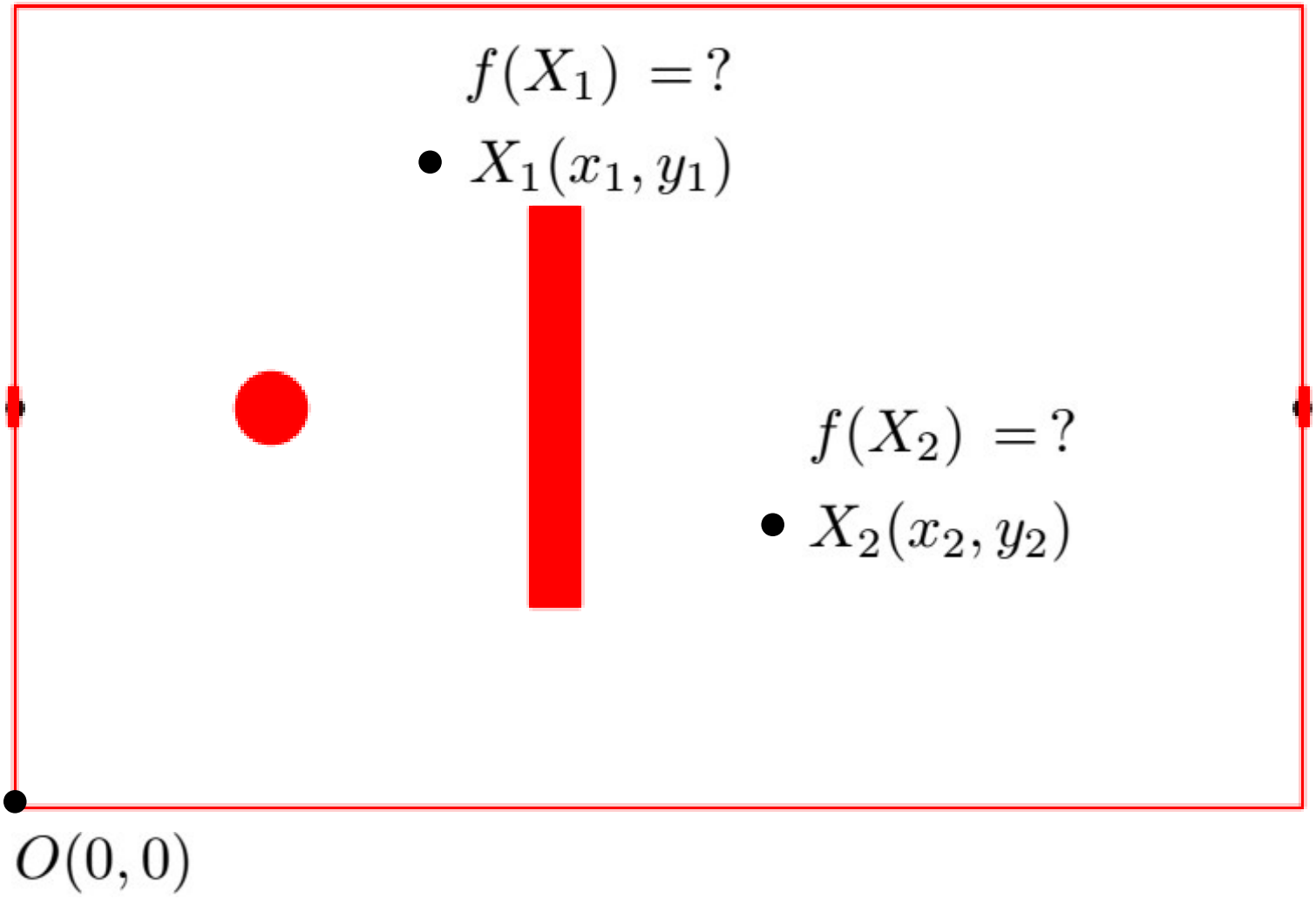


Application à la modélisation des foules

Application à la modélisation des foules



Application à la modélisation des foules

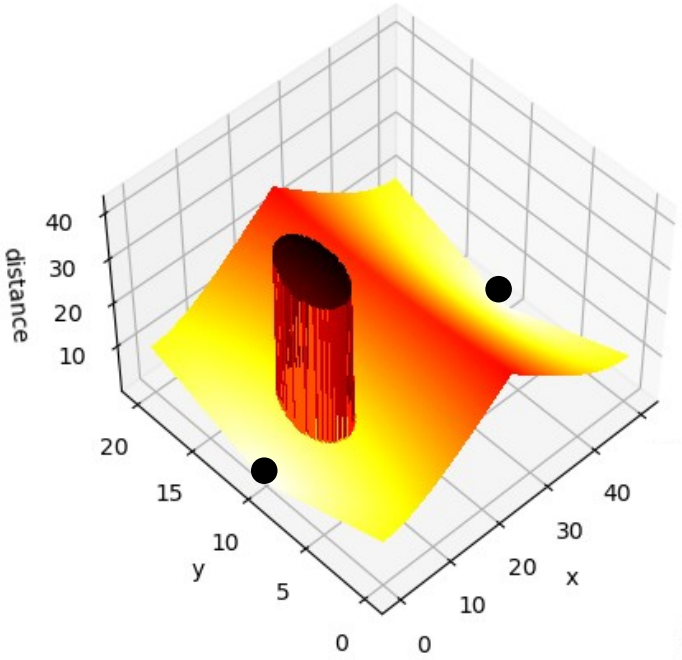


Application à la modélisation des foules

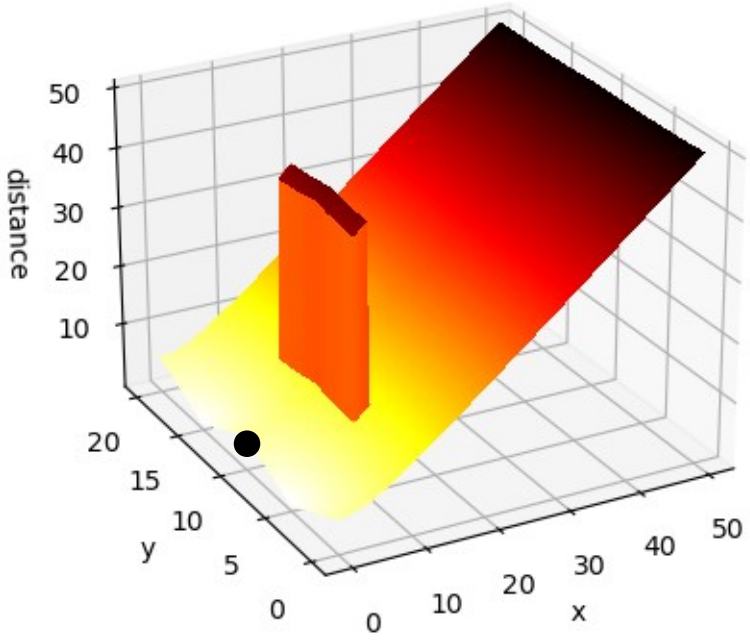
Première implémentation des obstacles : ajout d'un coût constant.

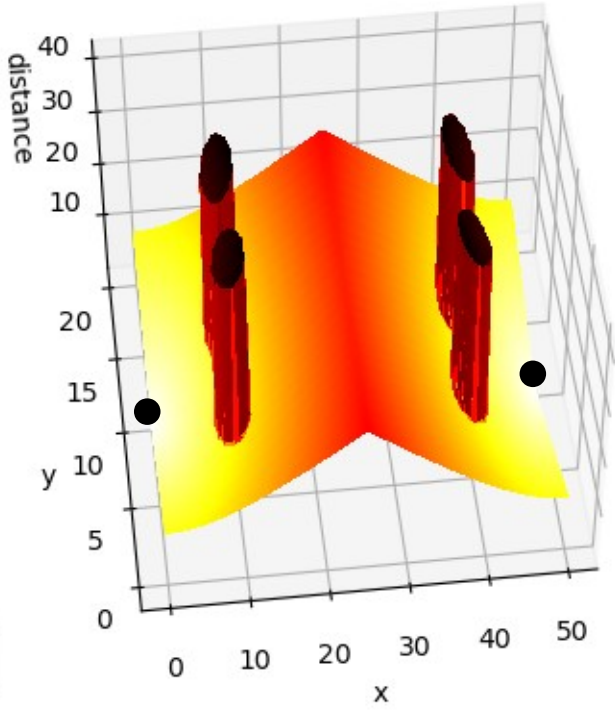
```
def fcout(l,portes,obstacles):
    x=l[0]
    y=l[1]
    return min([sqrt((x-portes[i][0])**2+(y-portes[i][1])**2)
    +passageobstacleconstant(obstacles,x,y) for i in range(len(portes))])

def passageobstacleconstant(obstacles,x,y):
    cout=0
    for elt in obstacles:
        if elt[0]=='cercle':                                    #obstacles ronds
            if (x-elt[1])**2+(y-elt[2])**2<elt[3]**2:
                cout+=30    #valeur arbitraire
        if elt[0]=='rectangle':                                 #obstacles rectangulaires
            if elt[1]<x<elt[1]+elt[3] and elt[2]<y<elt[2]+elt[4]:
                cout+=30    #valeur arbitraire
    return cout
```

Application à la modélisation des foules



$$distance = \text{coût}(x, y)$$

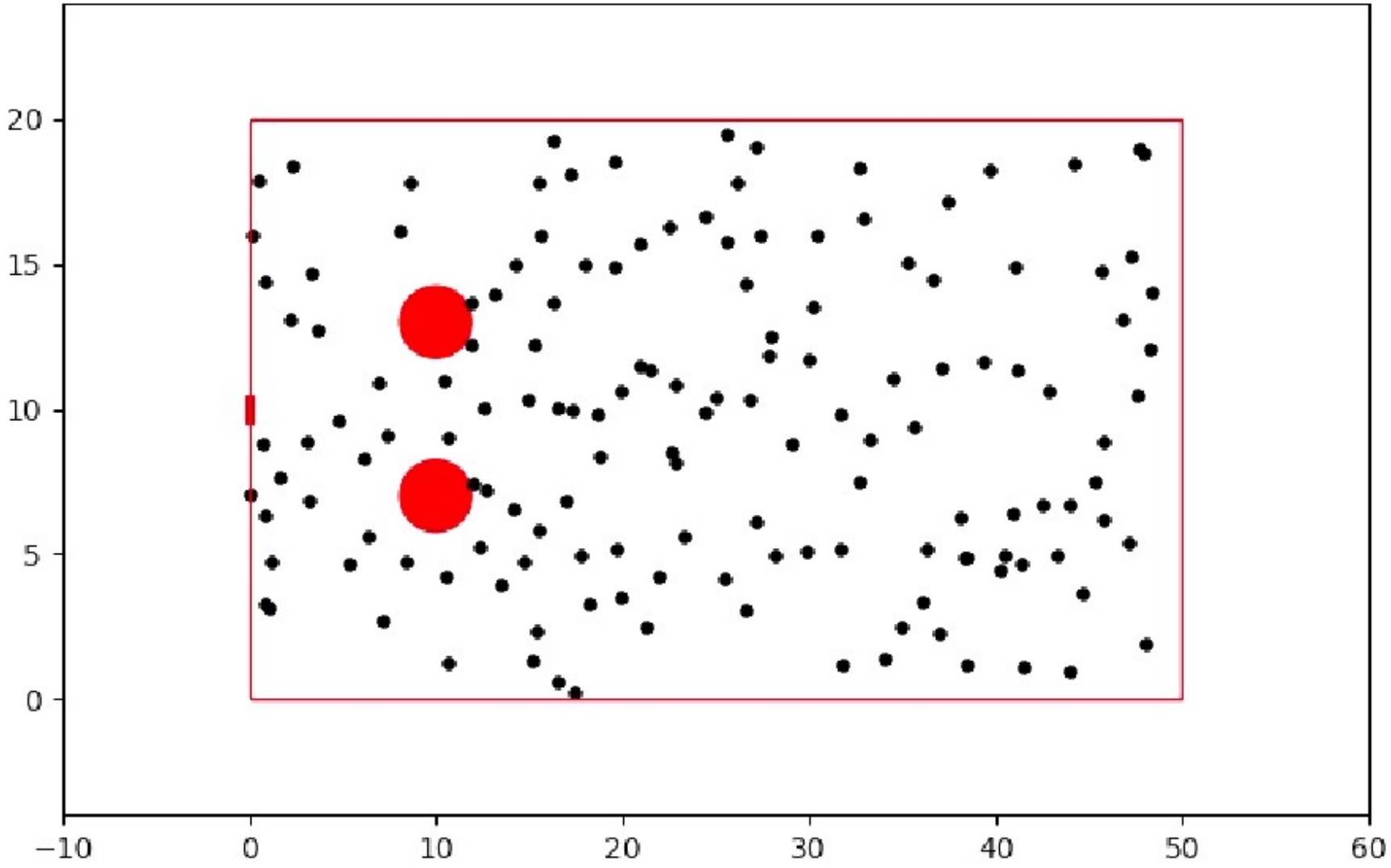




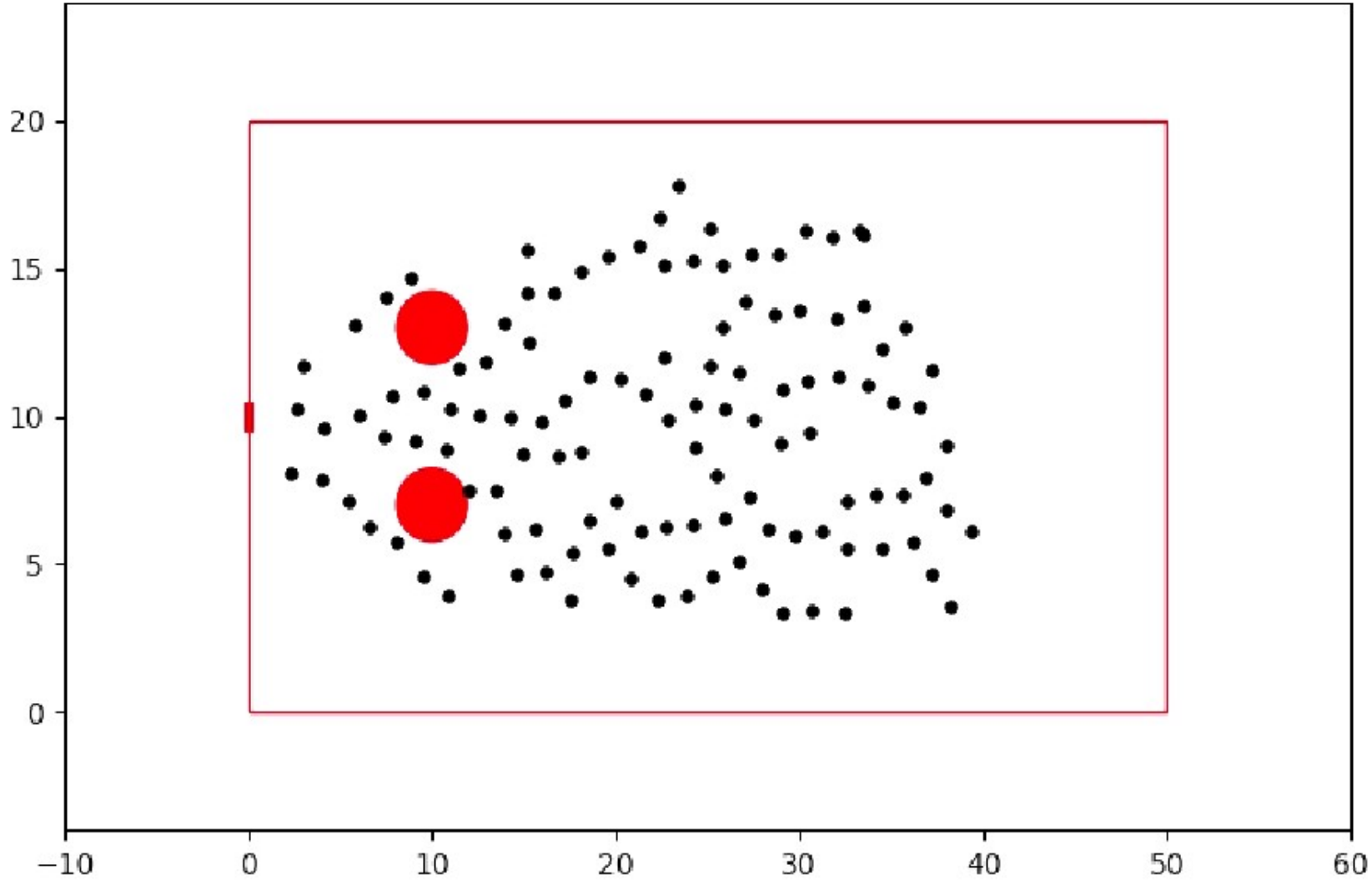
Représentation de la distance en fonction de x et y dans 3 salles

● Portes

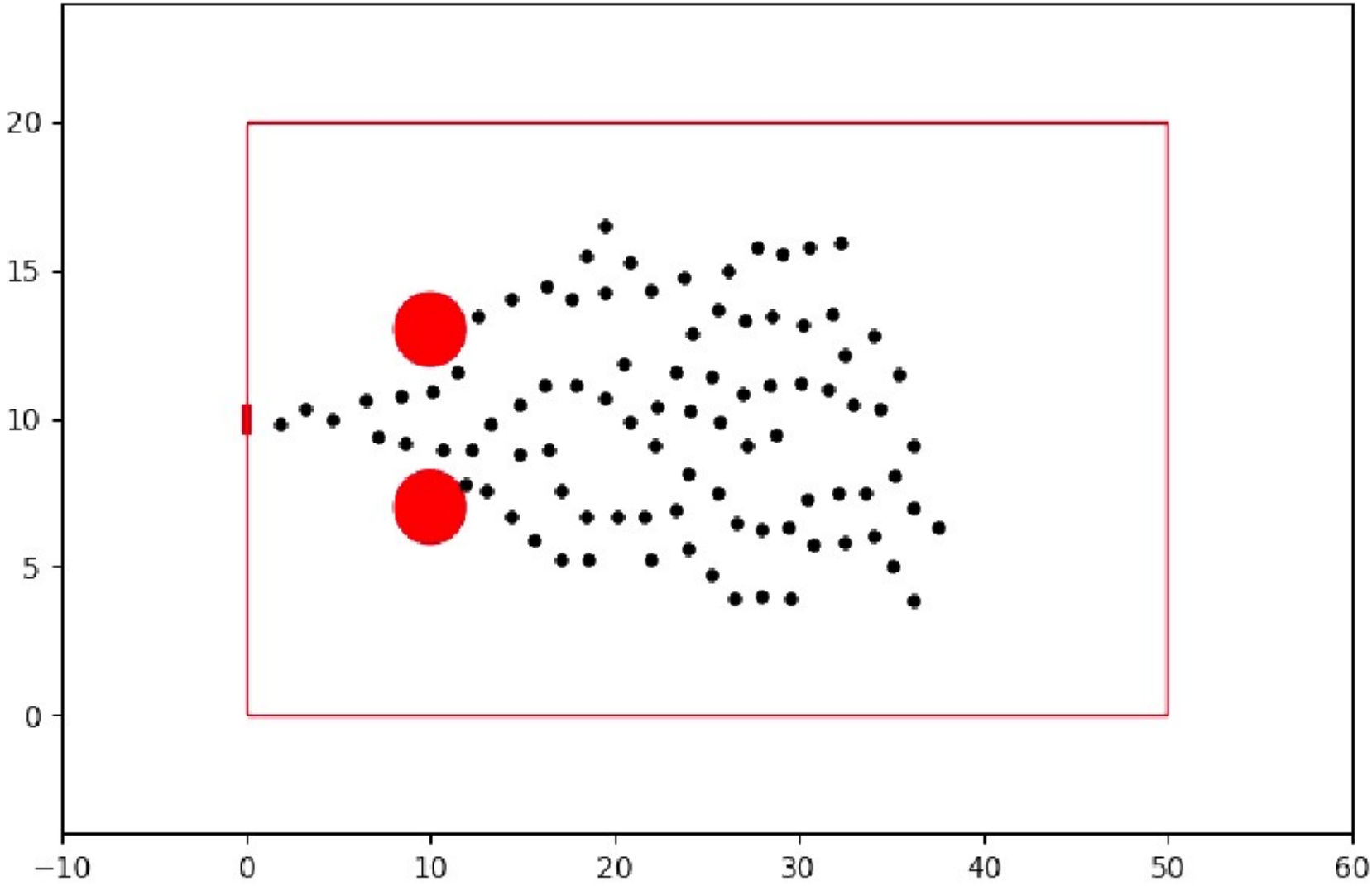
Application à la modélisation des foules



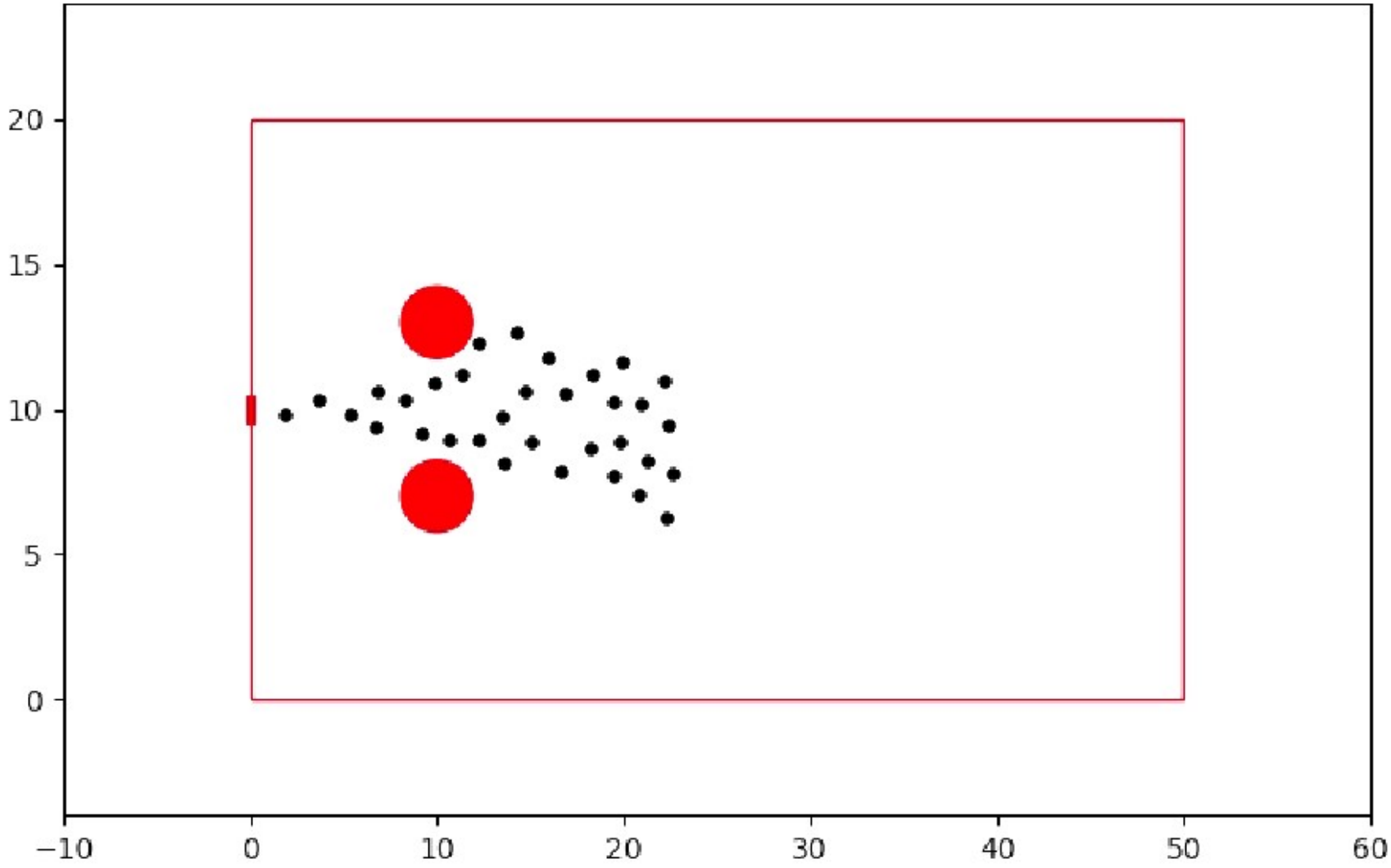
Application à la modélisation des foules



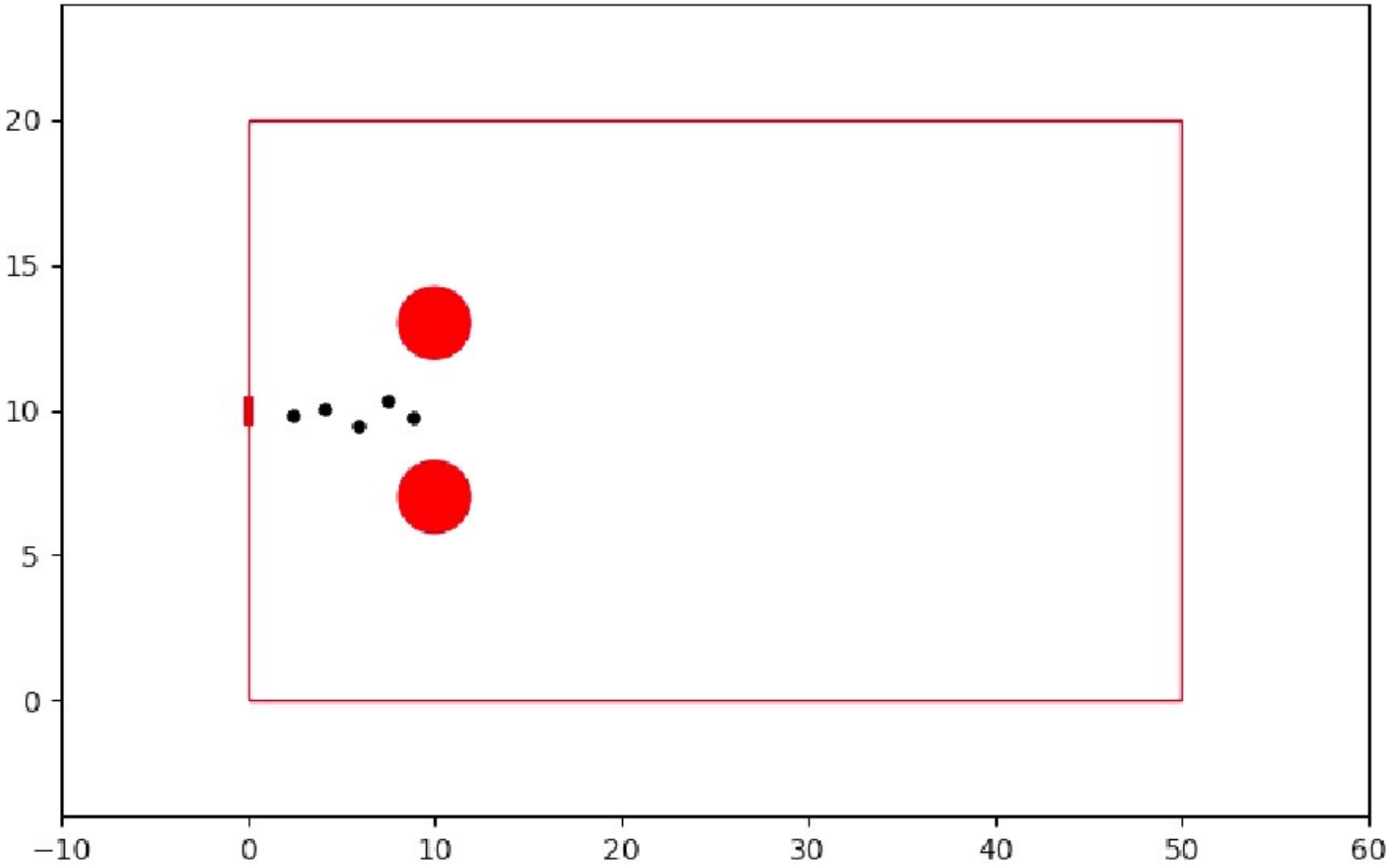
Application à la modélisation des foules



Application à la modélisation des foules

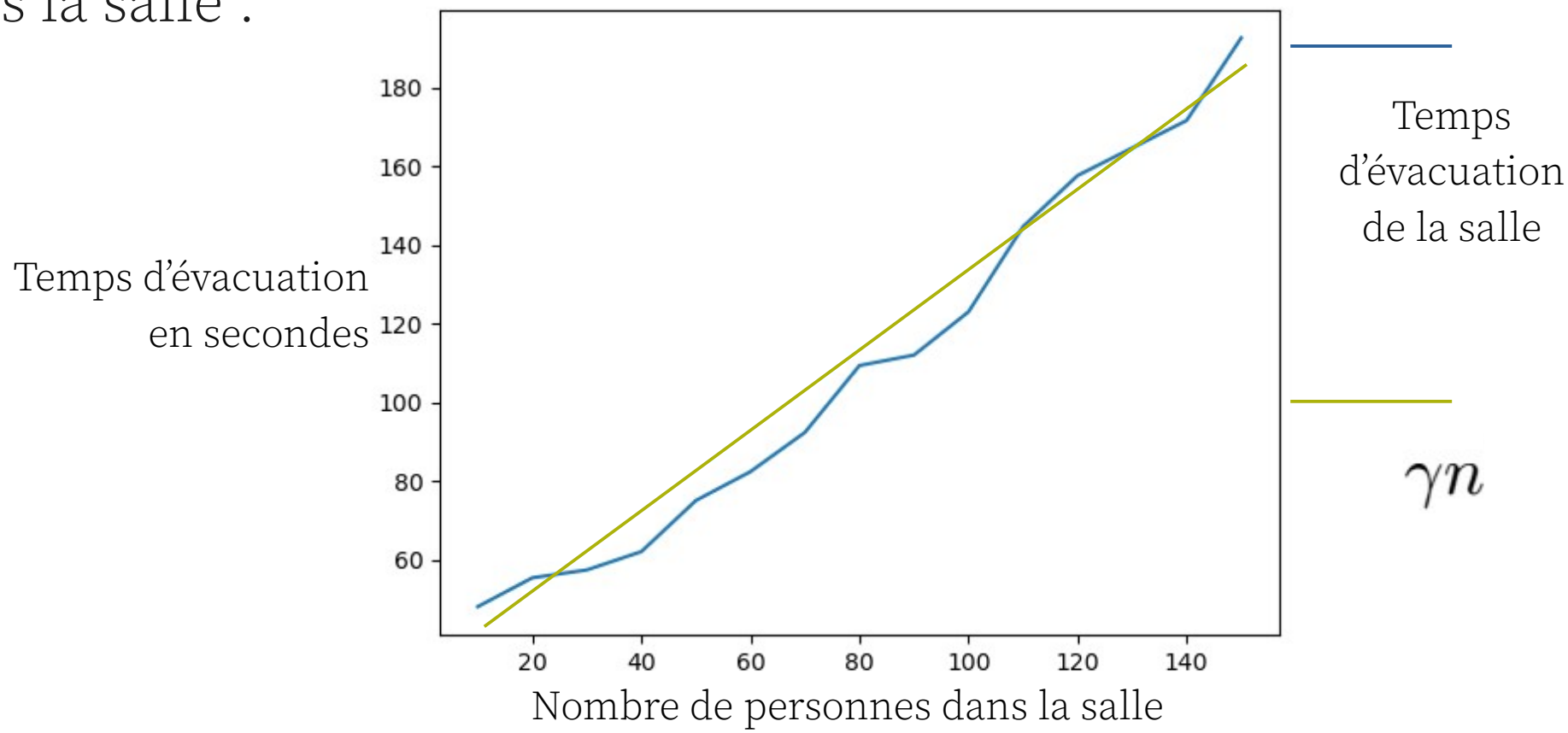


Application à la modélisation des foules



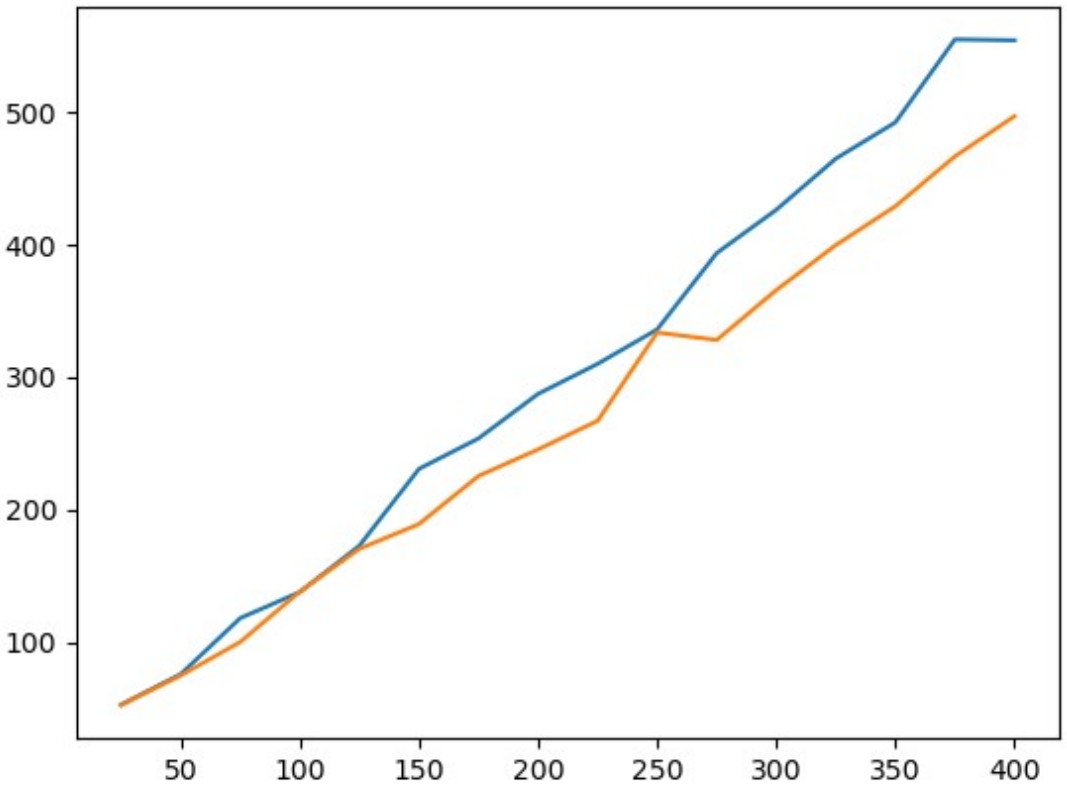
Application à la modélisation des foules

Temps d'évacuation en fonction du nombre de personnes dans la salle :

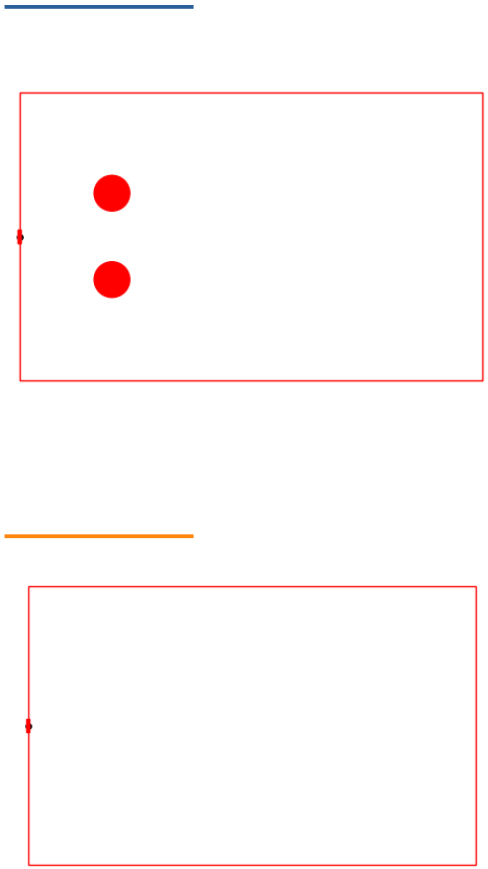


Application à la modélisation des foules

Temps d'évacuation en secondes



Nombre de personnes dans la salle



Application à la modélisation des foules

Points positifs :

- piétons ne se chevauchant pas
- piétons ne traversant pas les murs ou les obstacles

Points négatifs :

- rares blocages
- pas d'anticipation des obstacles
- piétons éloignés les uns des autres
- pas de phénomène de congestion
- pas de ralentissement à proximité des obstacles

Application à la modélisation des foules

Points positifs :

- piétons ne se chevauchant pas
- piétons ne traversant pas les murs ou les obstacles

Points négatifs :

- rares blocages
- pas d'anticipation des obstacles
- piétons éloignés les uns des autres
- pas de phénomène de congestion
- pas de ralentissement à proximité des obstacles

Application à la modélisation des foules

Première amélioration :

Ajout d'un léger déplacement aléatoire afin d'empêcher les blocages.

Application à la modélisation des foules

Points positifs :

- piétons sortant tous
- piétons ne se chevauchant pas
- piétons ne traversant pas les murs ou les obstacles

Points négatifs :

- pas d'anticipation des obstacles
- piétons éloignés les uns des autres
- pas de phénomène de congestion
- pas de ralentissement à proximité des obstacles

Application à la modélisation des foules

Deuxième amélioration :

Autre implémentation des obstacles : ajout d'un coût non constant.

$$\frac{Q}{e^{\alpha((x-x_o)^2+(y-y_o)^2)}}$$

Q : hauteur de la cloche

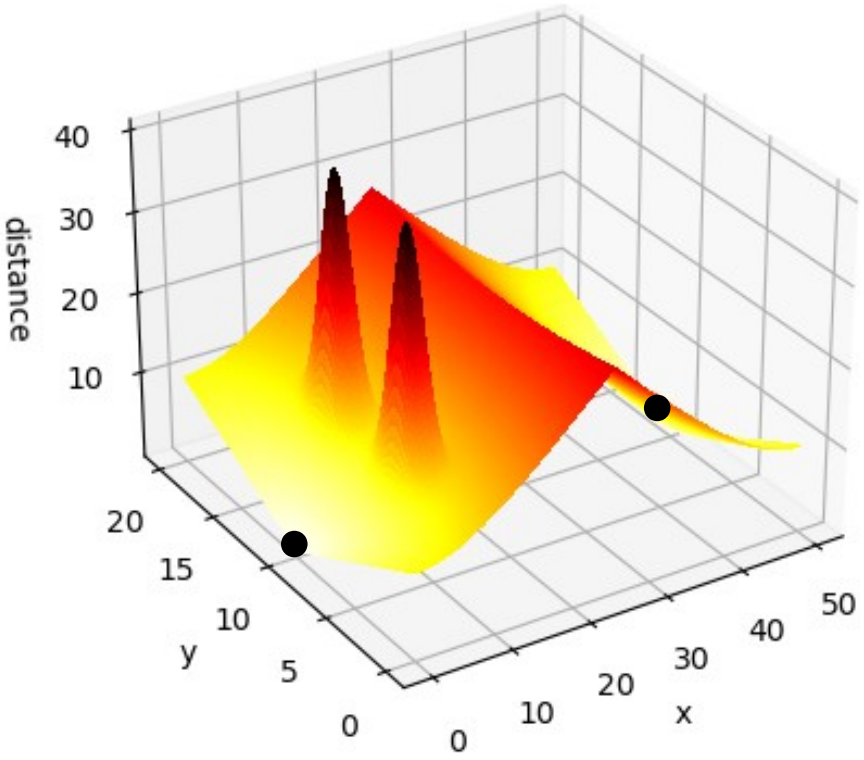
α : étalement de la cloche

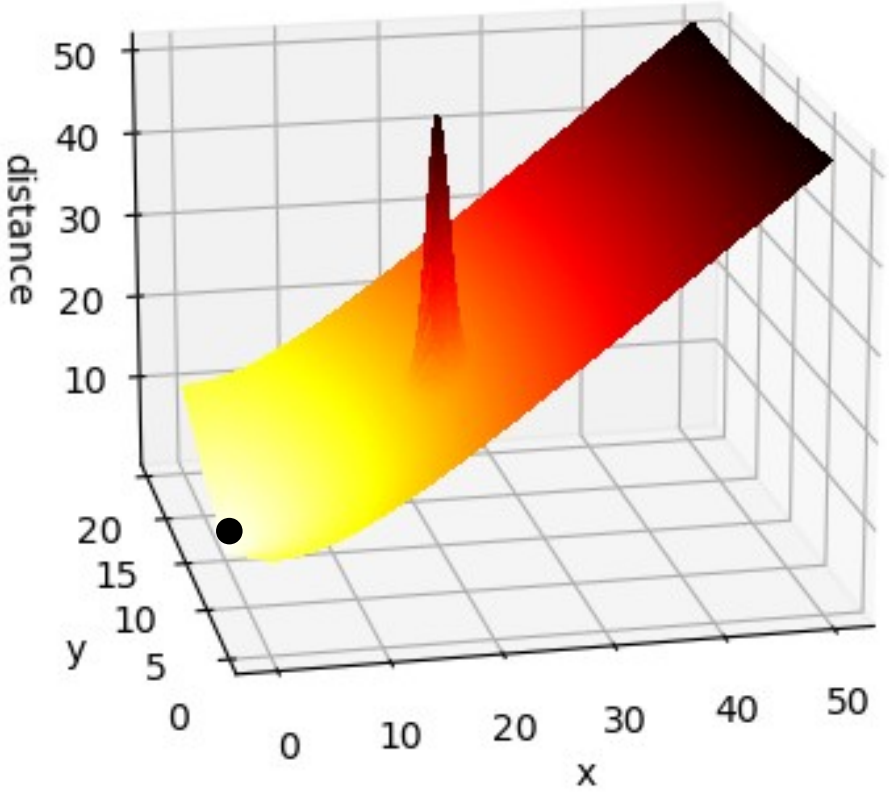
```
def passageobstacleexp(obstacles,x,y):
    cout=0
    for elt in obstacles:
        if elt[0]=='cercle':
            cout+=30*exp(-elt[3]*((x-elt[1])**2+(y-elt[2])**2))
    return cout
```

Application à la modélisation des foules

Représentation de la distance en fonction de x et y dans 2 salles

$$distance = \text{coût}(x, y)$$





● Portes

Application à la modélisation des foules

Points positifs :

- piétons sortant tous
- piétons ne se chevauchant pas
- piétons ne traversant pas les murs ou les obstacles
- légère anticipation des obstacles

Points négatifs :

- piétons éloignés les uns des autres
- pas de phénomène de congestion
- pas de ralentissement à proximité des obstacles

Application à la modélisation des foules

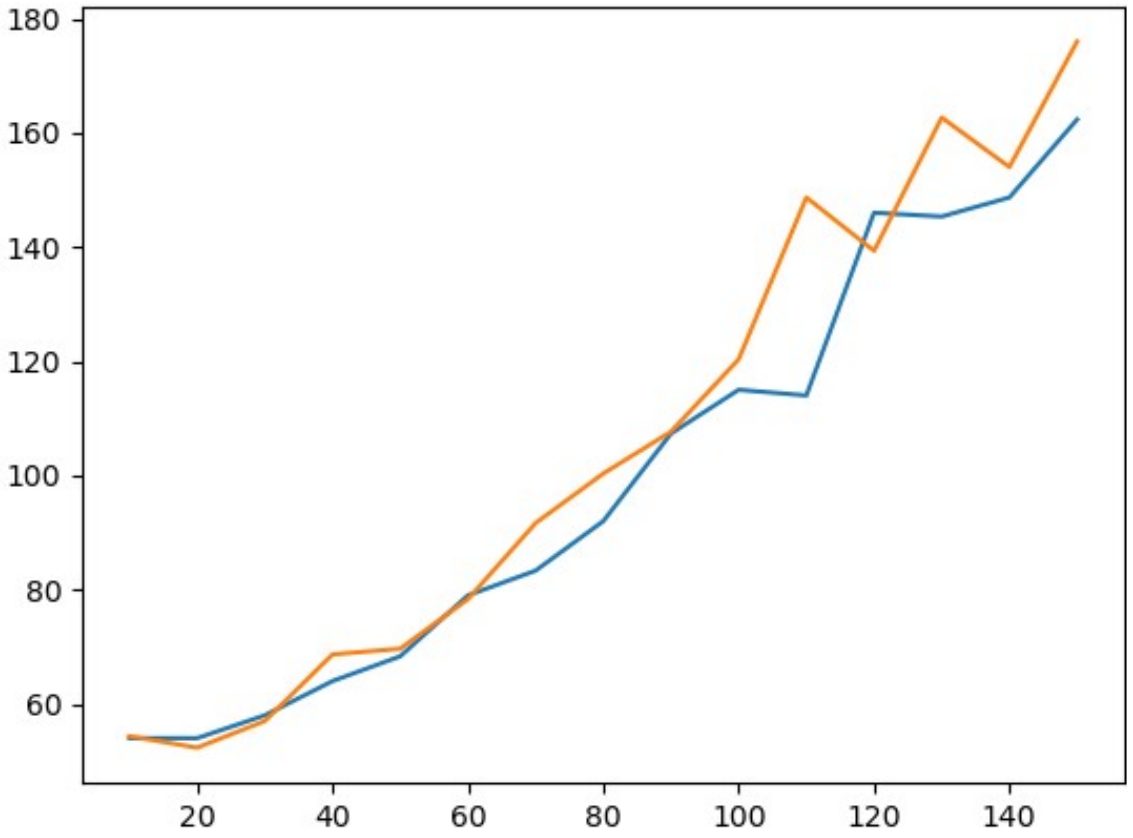
Troisième amélioration :

Prise en compte des autres piétons pour ajuster la trajectoire :

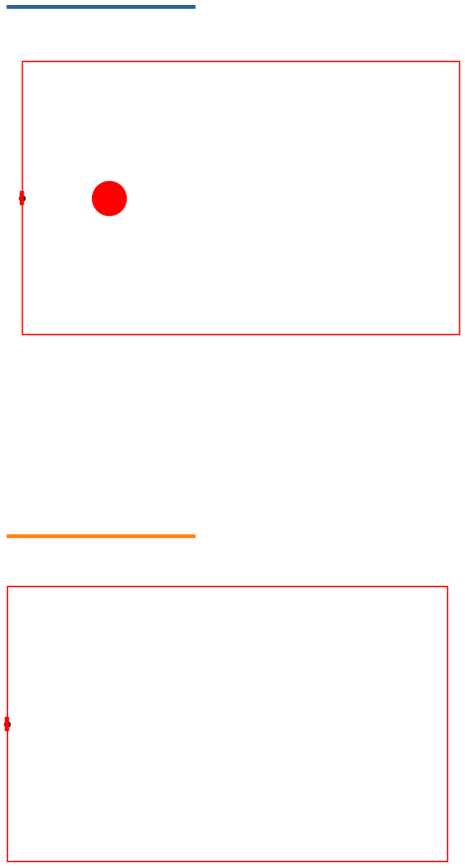
```
def coutvoisins(voisins,x,y):
    cout=0
    for elt in voisins:
        cout+=0.5*exp(-3*((x-elt[0])**2+(y-elt[1])**2))
    return cout
```

Application à la modélisation des foules

Temps d'évacuation en secondes



Nombre de personnes dans la salle



Application à la modélisation des foules

Points positifs :

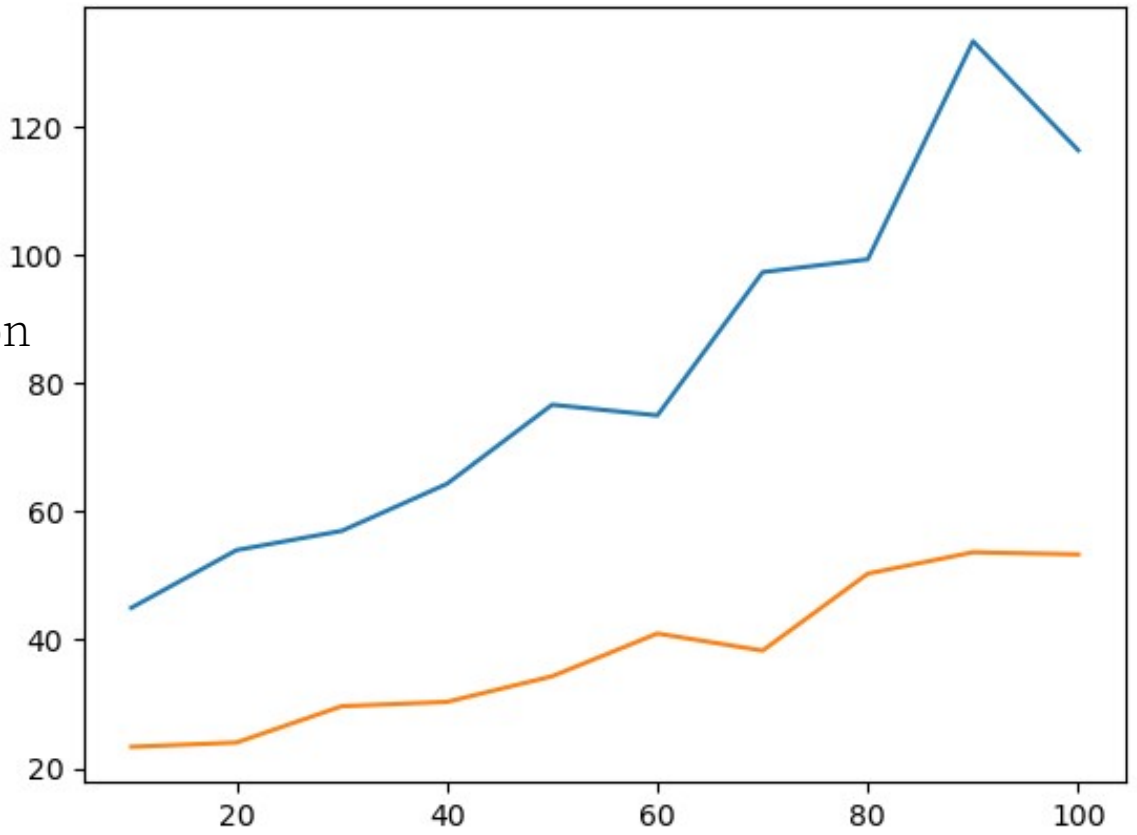
- piétons sortant tous
- piétons ne se chevauchant pas
- piétons ne traversant pas les murs ou les obstacles
- légère anticipation des obstacles
- réduction du temps d'évacuation avec un obstacle

Points négatifs :

- piétons éloignés les uns des autres
- pas de ralentissement à proximité des obstacles

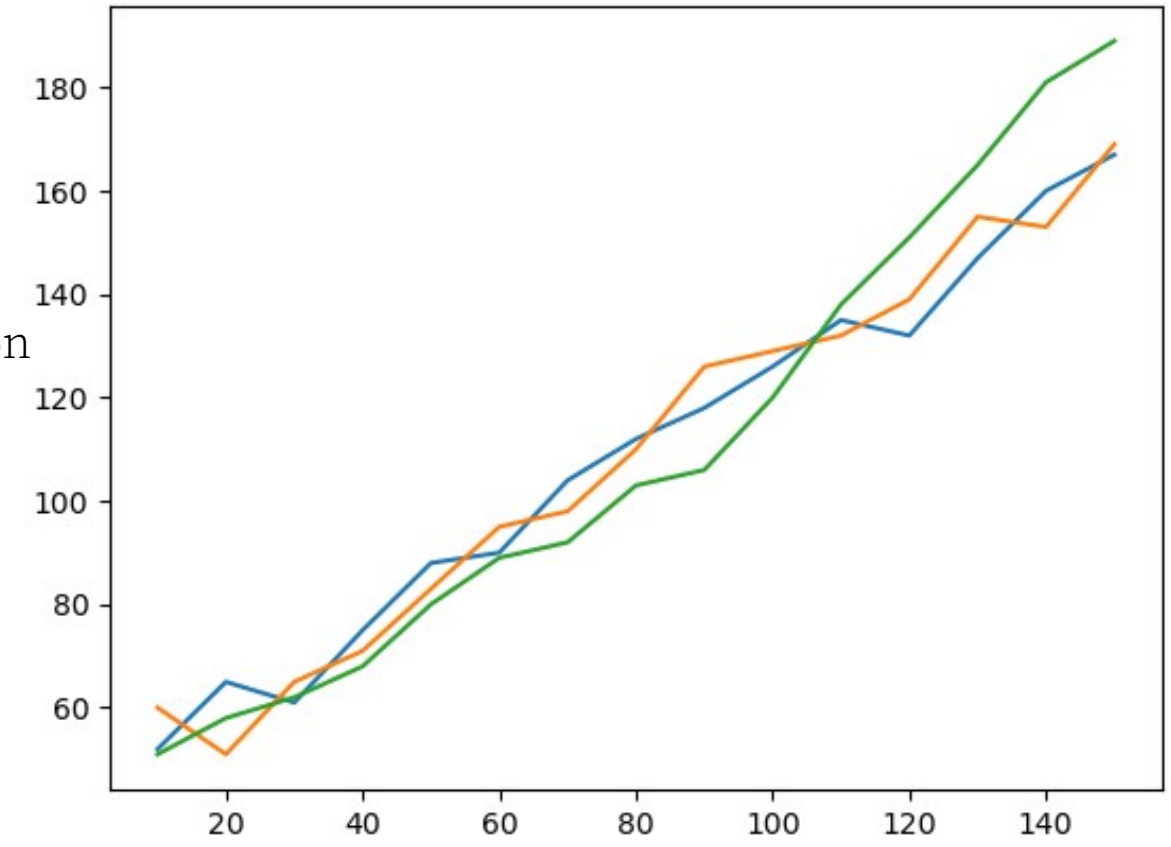
Application à la modélisation des foules

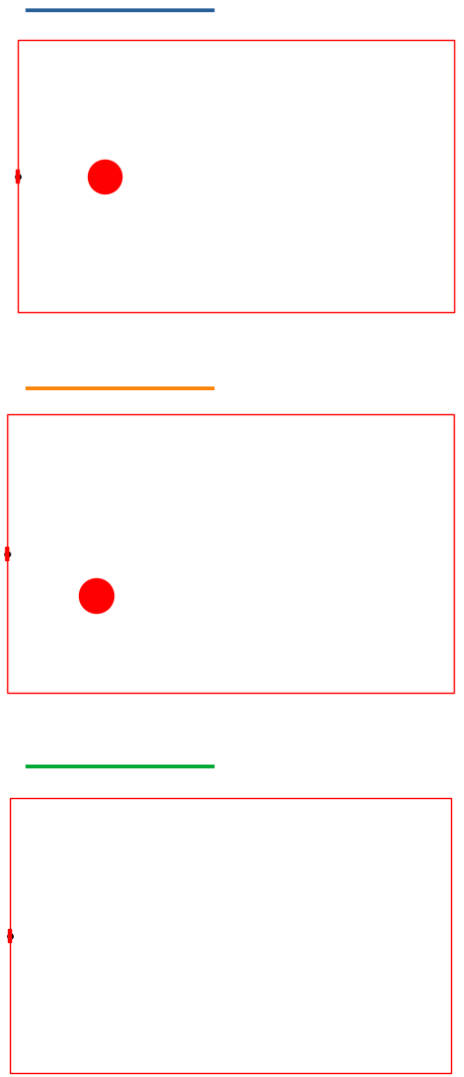
Temps d'évacuation en secondes



Nombre de personnes dans la salle

Application à la modélisation des foules





Conclusion

Conclusion

Modélisation en accord avec les résultats expérimentaux.

Conclusion

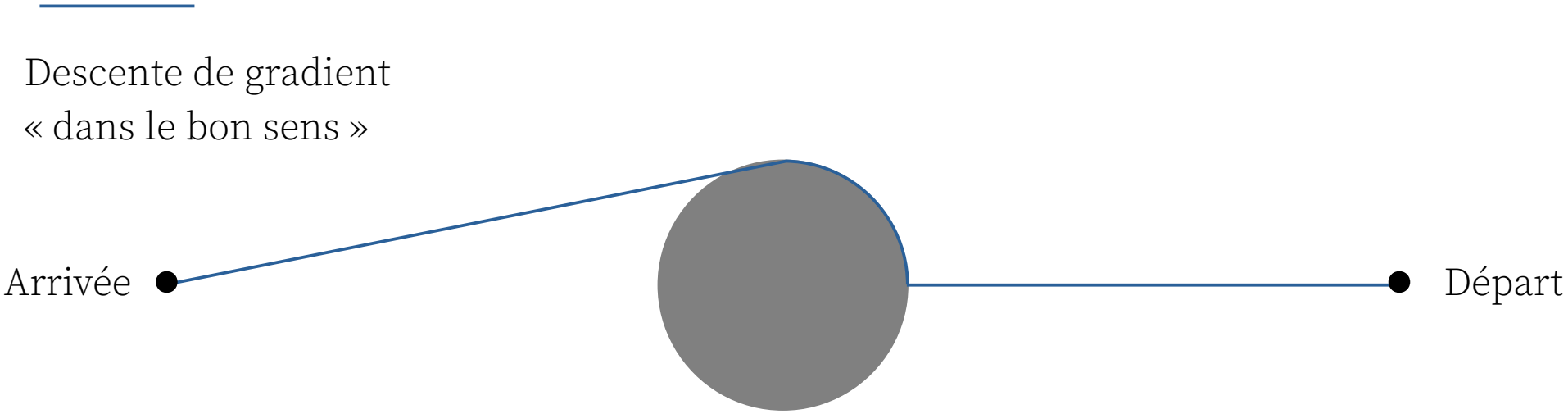
Modélisation en accord avec les résultats expérimentaux.

Modélisation peu réaliste :
- anticipation des obstacles,
- interactions des piétons entre eux,
- hypothèses de travail…

Conclusion

Modélisation en accord avec les résultats expérimentaux.

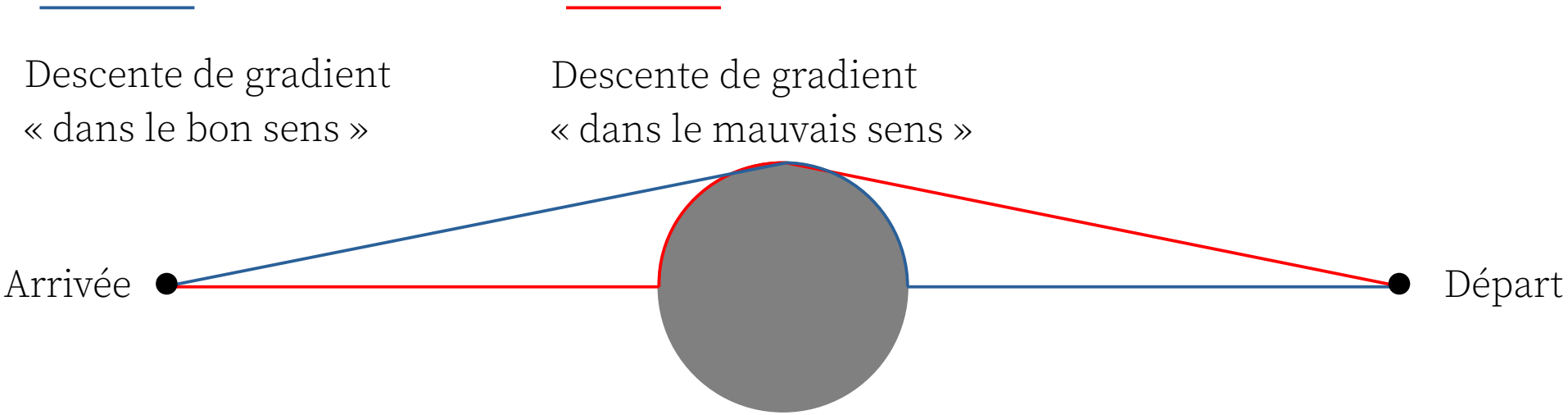
Modélisation peu réaliste :
- anticipation des obstacles,
- interactions des piétons entre eux,
- hypothèses de travail…



Conclusion

Modélisation en accord avec les résultats expérimentaux.

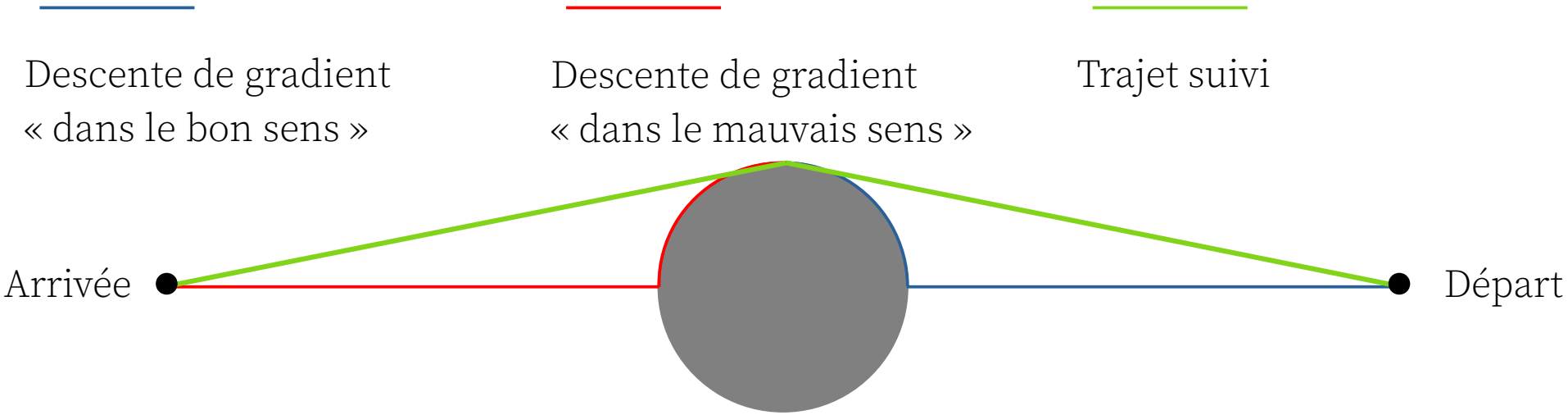
Modélisation peu réaliste : - anticipation des obstacles,
- interactions des piétons entre eux,
- hypothèses de travail…



Conclusion

Modélisation en accord avec les résultats expérimentaux.

Modélisation peu réaliste : - anticipation des obstacles,
- interactions des piétons entre eux,
- hypothèses de travail…



Conclusion

Applications : - sécurité,

- architecture,

- jeu vidéo,

- cinéma…

Conclusion

Applications : - sécurité,

Exemple : formations « gestion des mouvements de foule »

https://www.cnfce.com/formation-gestion-des-mouvements-de-foule

- architecture,

- jeu vidéo,

- cinéma…

Conclusion

Applications : - sécurité,

Exemple : formations « gestion des mouvements de foule »

https://www.cnfce.com/formation-gestion-des-mouvements-de-foule

- architecture,

Exemple : partie D:2 du guide de l'UEFA pour des stades de qualité : Contrôle des flux de circulation

https://fr.uefa.com

- jeu vidéo,

- cinéma…

Conclusion

Applications : - sécurité,

Exemple : formations « gestion des mouvements de foule » https://www.cnfce.com/formation-gestion-des-mouvements-de-foule

- architecture,

Exemple : partie D:2 du guide de l'UEFA pour des stades de qualité : Contrôle des flux de circulation https://fr.uefa.com

https://www.jeuxvideo.com/

Foule dans le jeu Assassin's Creed Unity

- jeu vidéo,

- cinéma…

Conclusion

Applications : - sécurité,

Exemple : formations « gestion des mouvements de foule »

https://www.cnfce.com/formation-gestion-des-mouvements-de-foule

- architecture,

Exemple : partie D:2 du guide de l'UEFA pour des stades de qualité : Contrôle des flux de circulation

https://fr.uefa.com

https://www.jeuxvideo.com/

Foule dans le jeu Assassin's Creed Unity

- jeu vidéo,

- cinéma…

Gladiator, foule de 35 000 personnes à partir de 2 000 figurants

https://djayesse.over-blog.com/gladiator-ridley-scott-2000.html

Bibliographie

Bibliographie

- Image des Maths : https://images.math.cnrs.fr/ , Modélisation de mouvements de foule.
- Pour La Science : https://www.pourlascience.fr , La foule en équation.
- Wikipédia : https://fr.wikipedia.org/ , Gradient, Algorithme du gradient.
- Thèses de Patrick Simo Kanmeugne et Philippe Pécol.
- Lionel Uhl : 1001 codes Python pour la modélisation - spécial classes prépas.

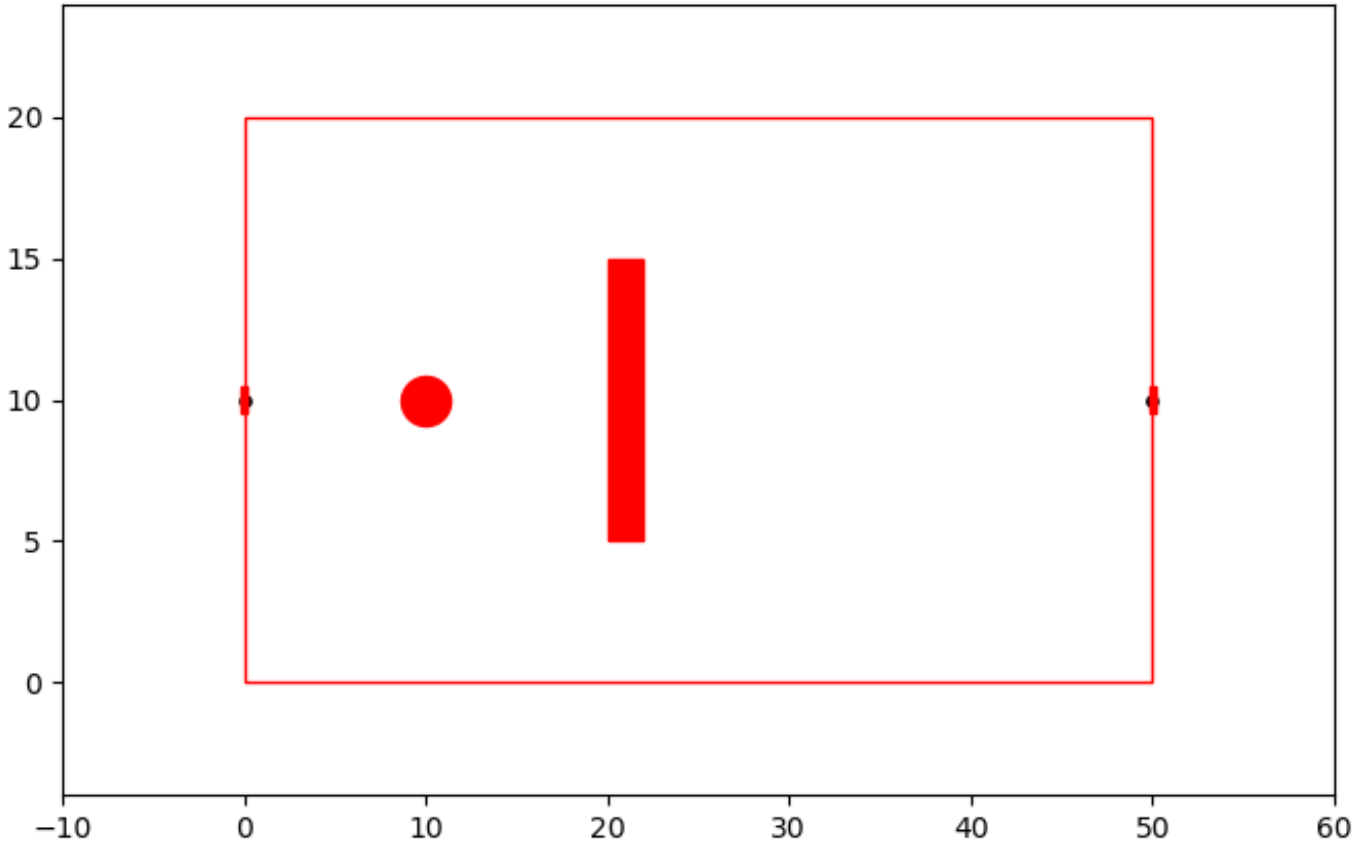
Annexe

Annexe

```
def gradientNDmodif(f, nuplet, portes,obstacles,voisins,pas, eps, max):
    j=0
    normegradient=1
    coordonnees=deepcopy(nuplet) #stockage des variables modifiées à chaque étape
    tableau=[nuplet]
    while j<max:
        for i in range(len(nuplet)):
            tplus=deepcopy(nuplet) #copie profonde des variables
            tplus[i]+=eps
            tmoins=deepcopy(nuplet) #copie profonde des variables
            tmoins[i]-=eps
            deriveepartielle=(f(tplus,portes,obstacles,voisins)-f(tmoins,portes,obstacles,voisins))/(2*eps)
            if abs(pas*deriveepartielle)>eps:
                if pas*deriveepartielle<0:
                    deplacement=-eps
                else:
                    deplacement=eps
            else:
                deplacement=pas*deriveepartielle
            coordonnees[i]-=deplacement
        nuplet=deepcopy(coordonnees) #récupération des variables modifiées
        j+=1
        tableau.append(nuplet)
        if tableau[-1]==tableau[-2]:
            for i in range(len(nuplet)):
                nuplet[i]-=pas
    return tableau
```

Annexe

```
>>> trajet(50,20,80,[[0,10],[50,10]],[['cercle',10,10,3],['rectangle',20,5,2,10]])
```



Annexe

```
def fcout(l,portes,obstacles,voisins):
    x=l[0]
    y=l[1]
    return min([sqrt((x-portes[i][0])**2+(y-portes[i][1])**2)
    +passageobstacleexp(obstacles,x,y)+coutvoisins(voisins,x,y) for i in range(len(portes))])

def passageobstacleconstant(obstacles,x,y):
    cout=0
    for elt in obstacles:
        if elt[0]=='cercle':                                    #obstacles ronds
            if (x-elt[1])**2+(y-elt[2])**2<elt[3]**2:
                cout+=30    #valeur arbitraire
        if elt[0]=='rectangle':                                 #obstacles rectangulaires
            if elt[1]<x<elt[1]+elt[3] and elt[2]<y<elt[2]+elt[4]:
                cout+=30    #valeur arbitraire
    return cout

def passageobstacleexp(obstacles,x,y):
    cout=0
    for elt in obstacles:
        if elt[0]=='cercle':
            cout+=30*exp(-elt[3]*(0.1*(x-elt[1])**2+1.8*(y-elt[2])**2))
    return cout

def coutvoisins(voisins,x,y):
    cout=0
    for elt in voisins:
        cout+=0.5*exp(-3*((x-elt[0])**2+(y-elt[1])**2))
    return cout
```

Annexe

```
def trajet(longueur, largeur,n,portes,obstacles):
    X=[uniform(0,longueur) for i in range(n)]
    Y=[uniform(0,largeur) for i in range(n)]
    for i in range(len(X)):
        while passageobstacleconstant(obstacles,X[i],Y[i])!=0:
            X[i]=X[i]+1
    trajectoires=[]
    for i in range(len(X)):
        print(i)
        trajectoires.append(gradientNDmodif(fcout,[X[i],Y[i]],portes,obstacles,[],0.01,0.05,5000))
    print([trajectoires[i][-1] for i in range(len(X))])
    dist=[0.3 for i in range(len(X))]
    listeX=[X]
    listeY=[Y]
    abs=[0]
    ord=[0]
    absancien=X.copy()
    ordancien=Y.copy()
    marqueur=0
    while abs!=[]:
        print(marqueur)
        marqueur+=1
        abs=[]
        ord=[]
        for i in range(len(trajectoires)):
            j=0
            while j<len(trajectoires[i])-1 and ((trajectoires[i][j][0]-X[i])**2+(trajectoires[i][j][1]-Y[i])**2)<dist[i]**2:
                j+=1
            a,o=trajectoires[i][j]
            distporte=min([(a-portes[k][0])**2+(o-portes[k][1])**2 for k in range(len(portes))])
            if distporte>2:
                ref=True
                for k in range(len(absancien)):#y-a-t'il quelqu'un devant à l'instant précédent :
                    if i!=k and ((a-absancien[k])**2+(o-ordancien[k])**2)<2 and fcout([absancien[k],ordancien[k]],portes,obstacles,voisins)<fcout([a,o],portes,obstacles,voisins):
                        ref=False
                if ref:#si il n'y a personne, on avance
                    c=a
                    d=o
                    dist[i]+=0.3
                else:#sinon, on ne bouge pas
                    c=absancien[i]
                    d=ordancien[i]
                absancien[i]=c
                ordancien[i]=d
                abs.append(c)#on ajoute les coordonnées calculées si personne devant, on ne bouge pas sinon
                ord.append(d)
            else:
                absancien[i]=-100
                ordancien[i]=-100
        listeX.append(abs)
        listeY.append(ord)
    affichage2(longueur,largeur,listeX,listeY,portes,obstacles)#affichage de l'avancement des piétons
```

Annexe

```
def trajetaleatoire(longueur, largeur,n,portes,obstacles):
    X=[uniform(0,longueur) for i in range(n)]
    Y=[uniform(0,largeur) for i in range(n)]
    for i in range(len(X)):
        while passageobstacleconstant(obstacles,X[i],Y[i])!=0:
            X[i]=X[i]+1
    dist=[0.3 for i in range(len(X))]
    listeX=[X]
    listeY=[Y]
    abs=X.copy()
    ord=Y.copy()
    absancien=X.copy()
    ordancien=Y.copy()
    marqueur=0
    m=[True for i in range(len(X))]
    while abs!=[]:
        print(marqueur)
        marqueur+=1
        trajectoires=[]
        count=0
        for i in range(len(X)):
            voisins=[[absancien[j],ordancien[j]] for j in range(len(absancien))]
            voisins.pop(i)
            if m[i]:
                trajectoires.append(gradientNDmodif(fcout,[abs[i-count],ord[i-count]],portes,obstacles,voisins,0.01+0.00005*marqueur,0.05,100))
            else:
                trajectoires.append([])
                count+=1
        abs=[]
        ord=[]
        for i in range(len(trajectoires)):
            if m[i]:
                j=0
                while j<len(trajectoires[i])-1 and ((trajectoires[i][j][0]-X[i])**2+(trajectoires[i][j][1]-Y[i])**2)<dist[i]**2:
                    j+=1
                a,o=trajectoires[i][j]
                distporte=min([(a-portes[k][0])**2+(o-portes[k][1])**2 for k in range(len(portes))])
                if distporte>2:
                    ref=True
                    for k in range(len(absancien)):#y-a-t'il quelqu'un devant à l'instant précédent :
                        if i!=k and ((a-absancien[k])**2+(o-ordancien[k])**2)<2 and fcout([absancien[k],ordancien[k]],portes,obstacles,voisins)<fcout([a,o],portes,obstacles,voisins):
                            ref=False
                    if ref:#si il n'y a personne, on avance
                        c=a+uniform(-0.01,0.01)
                        d=o+uniform(-0.01,0.01)
                        dist[i]+=0.3
                    else:#sinon, on ne bouge pas
                        c=absancien[i]
                        d=ordancien[i]
                    absancien[i]=c
                    ordancien[i]=d
                    abs.append(c)#on ajoute les coordonnées calculées si personne devant, on ne bouge pas sinon
                    ord.append(d)
                else:
                    absancien[i]=-100
                    ordancien[i]=-100
                    m[i]=False
        listeX.append(abs)
        listeY.append(ord)
    affichage2(longueur,largeur,listeX,listeY,portes,obstacles)#affichage de l'avancement des piétons
```

Annexe

```
def affichage2(longueur, largeur, listeX,listeY,portes,obstacles):
    dimension=10/(longueur+largeur)
    listeXY=[]
#liste des listes des couples (x,y) de chacun de mes points :
    for i in range(len(listeX)):
        listeXY.append([[listeX[i][j],listeY[i][j]] for j in range(len(listeX[i]))])
    plt.close('all')
#dimension de la salle :
    fig=plt.figure(figsize=[8,5])
    axSalle=plt.subplot2grid((1,1),(0,0))
    axSalle.axis([-0.2*longueur, longueur*1.2,-0.2*largeur, largeur*1.2])
#affichage des portes :
    for i in range(len(portes)):
        axSalle.add_patch(patches.Rectangle((portes[i][0]-0.125,portes[i][1]-0.5),0.25,1,edgecolor = 'red',facecolor ='red',fill=True))
#affichage des murs de la salle :
    axSalle.add_patch(patches.Rectangle((0,0),longueur,largeur,edgecolor = 'red',fill=False))
#affichage des obstacles rectangulaires :
    ronds=[]
    for elt in obstacles:
        if elt[0]=='rectangle':
            axSalle.add_patch(patches.Rectangle((elt[1],elt[2]),elt[3],elt[4],edgecolor = 'red',facecolor='red',fill=True))
        else:
            ronds.append(elt)
#affichage des obstacles ronds :
    axSalle.scatter([ronds[i][1] for i in range(len(ronds))],[ronds[i][2] for i in range(len(ronds))],[300*ronds[i][3] for i in range(len(ronds))], color='red')
    listeT=np.array([np.array([listeXY[0][i][0],listeXY[0][i][1]]) for i in range(len(listeXY[0]))],dtype=object)
    #print(listeXY)
#affichage des personnes au départ :
    personnes=axSalle.scatter(listeT[:, 0], listeT[:, 1], s=100*dimension, color=[0,0,0], marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=1, linewidths=None, edgecolors=None,
plotnonfinite=False, data=None)
#fonction d'animation :
    def animation(iter):
        if iter!=len(listeXY)-1:
            listeT=np.array([np.array([listeXY[iter][i][0],listeXY[iter][i][1]]) for i in range(len(listeXY[iter]))],dtype=object)
        else:
            listeT=np.array(portes,dtype=object)
        #print(listeT)
        personnes.set_offsets(listeT)
        #scat.set_array(array) pour changer les couleurs
        return personnes,
    ani=anim.FuncAnimation(fig, animation, frames=len(listeX),interval=100,repeat=False)
    plt.show()
```

Annexe

```
def nappe3D(longueur, largeur, portes, obstacles, n):
    x=np.linspace(0, longueur, n)
    y=np.linspace(0, largeur, n)
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    Z=[]
    for i in range(n):
        z=[]
        for j in range(n):
            z.append(fcout([x[i],y[j]],portes,obstacles,[]))
        Z.append(z)
    Z=np.array(Z)
    Z=Z.T
    X,Y=np.meshgrid(x,y)
    surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='hot_r', linewidth=0, antialiased=False)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('distance')
    plt.show()
```

Annexe

```
def temps(longueur, largeur,listen,portes,obstacles):
    X=listen
    Y=[]
    for i in range(len(listen)):
        Y.append(trajetcomparaison(longueur, largeur,listen[i],portes,obstacles)/3)
    plt.plot(X,Y)
    plt.show()

def comparaisonobstacles(longueur, largeur,listen,portes,listeobstacles):
    X=listen
    for j in range(len(listeobstacles)):
        Y=[]
        for i in range(len(listen)):
            Y.append(trajetaleatoirecomparaison(longueur, largeur,listen[i],portes,listeobstacles[j][0])/3)
        plt.plot(X,Y,label=listeobstacles[j][1])
    plt.show()

def comparaisonportes(longueur, largeur,listen,listeportes,obstacles):
    X=listen
    for j in range(len(listeportes)):
        Y=[]
        for i in range(len(listen)):
            Y.append(trajetcomparaison(longueur, largeur,listen[i],listeportes[j],obstacles)/3)
        plt.plot(X,Y)
    plt.show()
```

Annexe

```
>>> comparaisonportes(50,20,[10,20,30,40,50,60,70,80,90,100],[[[0,10]],[[0,10],[50,10]]],[])
```

```
>>> comparaisonobstacles(50,20,[25,50,75,100,125,150,175,200,225,250,275,300,325,350,375,400],[[0,10]],[[[['cercle
',10,7,2],['cercle',10,13,2]],'2 obstacles circulaires'],[[],'aucun obstacle']])
```

```
>>> trajetaleatoire(50,20,2,[[0,10]],[['cercle',10,7,2]])
```

```
>>> nappe3D(50,20,[[0,10]],[['cercle',20,10,3]],300)
```

Annexe

- Algorithme du gradient : direction du gradient
- Algorithme du gradient conjugué : direction du gradient conjugué
- Algorithme de Newton : direction de Newton
- Algorithme de quasi-Newton : direction de quasi-Newton

Annexe

- Direction du gradient : $d = -\nabla f(x)$
- Direction du gradient conjugué : $d = -\nabla f(x) + \beta\, d\_$
- Direction de Newton : $d = -\left(\nabla^2 f(x)\right)^{-1} \nabla f(x)$
- Direction de quasi-Newton : $d = -M^{-1}\nabla f(x)$

$$M \sim \nabla^2 f(x)$$

2 000 figurants pour créer par ordinateur une foule de 35 000 spectateurs virtuels qui réagissaient de façon crédible aux scènes de combat dans l'arène.

En effet, pendant le tournage, seulement les deux premiers rangs du Colisée étaient occupés par des figurants, les milliers de spectateurs supplémentaires furent rajoutés en post-production à Londres.

Méthode de descente de gradient

Algorithme de descente de gradient :

But : trouver un minimum de $f : (x_1, ..., x_n) \longmapsto f(x_1, ..., x_n)$

Soit $\mathcal{E} > 0$, $\alpha > 0$.

En partant d'un point X_0, on calcule ∇f et on se dirige dans sa direction sur une certaine distance α (le pas). On obtient alors un nouveau point X_1.

En itérant le procédé jusqu'à atteindre un gradient de norme inférieur à $\mathcal{E}$, on trouve X_{final} tel que la pente de f en X_{final} soit presque nulle.

Application à la modélisation des foules

Un piéton est représenté par un couple de coordonnées (x,y).

L'origine (0,0) du repère se situe en bas à gauche de la salle.

f : fonction traduisant la distance d'un piéton à l'objectif le plus proche en prenant en compte les obstacles.