TD 4 - Apprentissage supervisé

Execice 1 : Algorithme de descente du gradient

Soit la fonction f : $\mathbb{R}^2 \rightarrow \mathbb{R}$ définie par :

$$f(x) = {x_1}^2 + {x_2}^2 + 1 \tag{1}$$

en notant les coordonnées d'un point de $\mathbb{R}^2$, $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

1. Formaliser l'algorithme de descente de gradient pour rechercher le minimum de cette fonction à partir : du point initial
   $x^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

   et un pas $\alpha^{(1)} = \dfrac{0.1}{n}$

$\nabla f(x_1, x_2) = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix}$,

CN1 : La fonction admet un **point critique** en (0,0).

$H = \nabla^2 f(x_1, x_2) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$

$$\begin{cases} Det(H) = 4 \\ Tr(H) = 4 \end{cases}$$

CN2 : Les valeurs propres sont de même signe et la trace est positive donc les valeurs propres sont positives donc (0,0) est un **minimum local**.

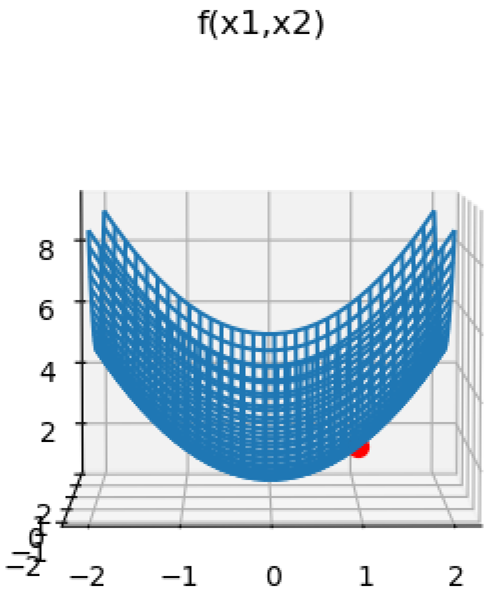
In [3]:
```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:
```
%matplotlib notebook
```

In [3]:
```
def afficherDescenteGradient(points):
    '''Afficher la fonction et les points correspondant à la descente de gradient'''
    x1 = np.outer(np.linspace(-2, 2, 32), np.ones(32))
    x2 = x1.copy().T # transpose
    f = x1**2+x2**2+1
    # Creating figure
    fig = plt.figure(figsize =(5,4))
    ax = plt.axes(projection ='3d')
    # Creating plot
    surf = ax.plot_wireframe(x1,x2,f, rstride=1, cstride=1)
    plt.title("f(x1,x2)")
```

```
    for (a,b) in points:
        ax.scatter(a,b,a*a+b*b+1,marker='o',s=50,color='red')
    plt.show()

points= [(1,0)]
afficherDescenteGradient(points)
print()
```

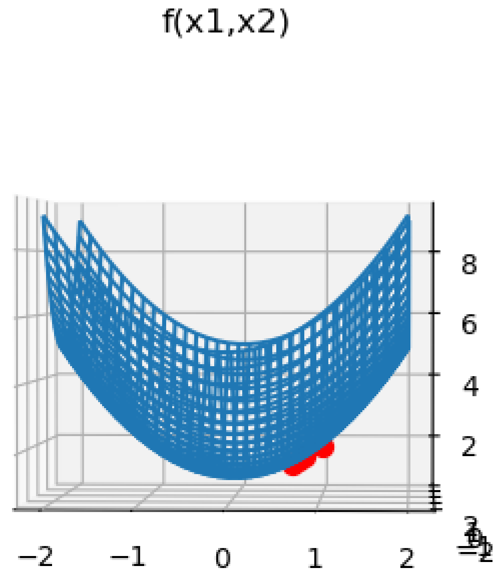


On peut donc appliquer l'algorithme de descente du gradient en partant du point x.

In [4]:
```
points= [(1,0)]
for step in range(1,4):
    alpha=0.1/step
    x1,x2=points[step-1]
    f = (x1-2*x1*alpha,x2-2*x2*alpha)
    points.append(f)
    print(f'step {step+1} : {f}')
```

```
step 2 : (0.8, 0.0)
step 3 : (0.72, 0.0)
step 4 : (0.6719999999999999, 0.0)
```

In [5]:
```
afficherDescenteGradient(points)
print()
```

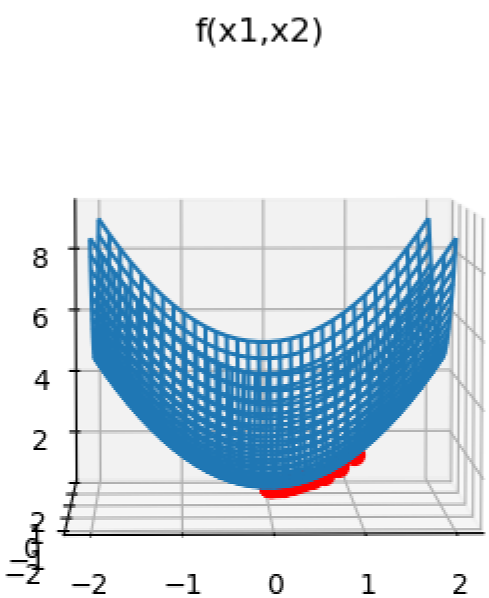


Remarque : Le point se rapproche bien de (0,0) mais utiliser un pas adaptatif nous ralenti.

In [6]:
```
step=1
points= [(1,0)]
f=(c1,c2)=points[0]
while c1>0.01: #on s'arrete quand on est suffisament proche
    alpha=0.1
    x1,x2=points[step-1]
    f =(c1,c2) = (x1-2*x1*alpha,x2-2*x2*alpha)
    points.append(f)
    print(f'step {step+1} : {f}')
    step+=1
```

```
step 2 : (0.8, 0.0)
step 3 : (0.64, 0.0)
step 4 : (0.512, 0.0)
step 5 : (0.4096, 0.0)
step 6 : (0.32768, 0.0)
step 7 : (0.26214400000000004, 0.0)
step 8 : (0.20971520000000005, 0.0)
step 9 : (0.16777216000000003, 0.0)
step 10 : (0.13421772800000004, 0.0)
step 11 : (0.10737418240000003, 0.0)
step 12 : (0.08589934592000002, 0.0)
step 13 : (0.06871947673600001, 0.0)
step 14 : (0.05497558138880001, 0.0)
step 15 : (0.04398046511104001, 0.0)
step 16 : (0.035184372088832, 0.0)
step 17 : (0.028147497671065603, 0.0)
step 18 : (0.022517998136852482, 0.0)
step 19 : (0.018014398509481985, 0.0)
step 20 : (0.014411518807585589, 0.0)
step 21 : (0.011529215046068471, 0.0)
step 22 : (0.009223372036854777, 0.0)
```

In [7]:
```
afficherDescenteGradient(points)
print()
```



Execice 2 : Hyperplan séparateur

Soit un problème à 2 classes, sont donnés en apprentissage:

- Pour la classe (1), les points $c_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ et $c_2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$
- Pour la classe (2), les points $c_3 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$ et $c_4 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$

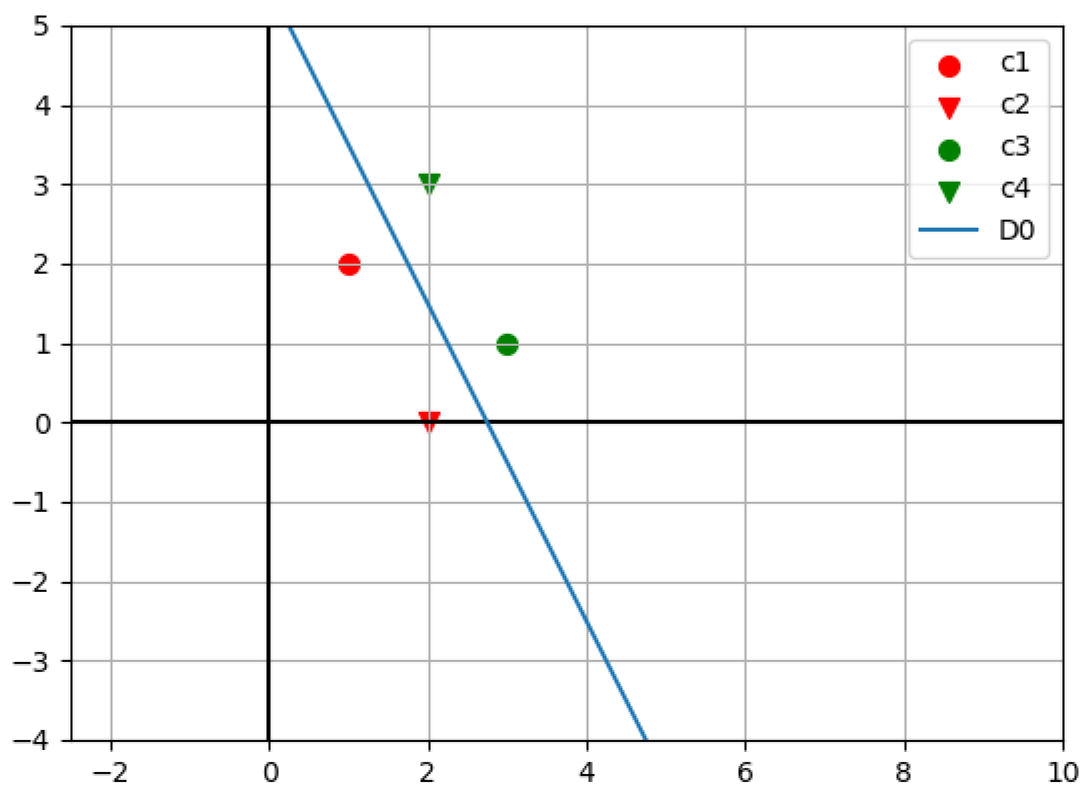
1. En notant les coordonnées d'un point de $\mathbb{R}^2, x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, vérifier que la droite D d'équation $-4x_1 - 2x_2 + 11 = 0$, sépare les deux classes.

In [5]:
```
def afficherDroites(c1,c2,c3,c4,c5,droites=[],mode='droites'):
    '''Afficher les 4 points et les droites de séparation'''
    fig, ax = plt.subplots()
    #axis
    ax.set_aspect('equal')
    ax.grid(True, which='both')
    ax.axhline(y=0, color='k')
    ax.axvline(x=0, color='k')
    #intervalle
    x=np.linspace(-2.5,10,200)
    ax.set_xlim([-2.5, 10])
    ax.set_ylim([-4, 5])
    #plot fist droite
    (x1,y1,label1)=c1
    (x2,y2,label2)=c2
    (x3,y3,label3)=c3
    (x4,y4,label4)=c4
    (x5,y5,label5)=c5
    ax.scatter(x1,y1,marker='o',s=50,color='red' if label1==1 else 'green',label="c1")
```

```python
    ax.scatter(x2,y2,marker='v',s=50,color='red' if label2==1 else 'green',label="c2")
    ax.scatter(x3,y3,marker='o',s=50,color='red' if label3==1 else 'green',label="c3")
    ax.scatter(x4,y4,marker='v',s=50,color='red' if label4==1 else 'green',label="c4")
    if (x5,y5)!=(0,0):
        ax.scatter(x5,y5,marker='o',s=50,color='yellow',label="c5")
    for i,(a,b) in enumerate(droites):
        if mode=='droites':
            ax.plot(x, a*x+b,label=f'D{i}')
        if mode=='centres':
            a1,a2,_=a
            b1,b2,_=b
            ax.scatter(a1,a2,marker='X',s=50,color='olive' ,label=f'centre1')
            ax.scatter(b1,b2,marker='X',s=50,color='darksalmon' ,label=f'centre2')
    ax.legend()
    # Creating plot
    plt.show()
    print()




c1=(x1,y1,label1)=(1,2,1)
c2=(x2,y2,label2)=(2,0,1)
c3=(x3,y3,label3)=(3,1,-1)
c4=(x4,y4,label4)=(2,3,-1)
c5=(0,0,0)
droites=[(-2,5.5)]
afficherDroites(c1,c2,c3,c4,c5,droites,'droites')
```



$D : -4x_1 - 2x_2 + 11 = 0$

$\Leftrightarrow D : x_2 = -2x_1 + 5.5$

1. On pose $y_1 = y_2 = 1$ et $y_3 = y_4 = -1$ et D_0 la droite qui passe par $\begin{bmatrix} 0 \\ 2 \end{bmatrix}$ et $\begin{bmatrix} 3 \\ 0 \end{bmatrix}$

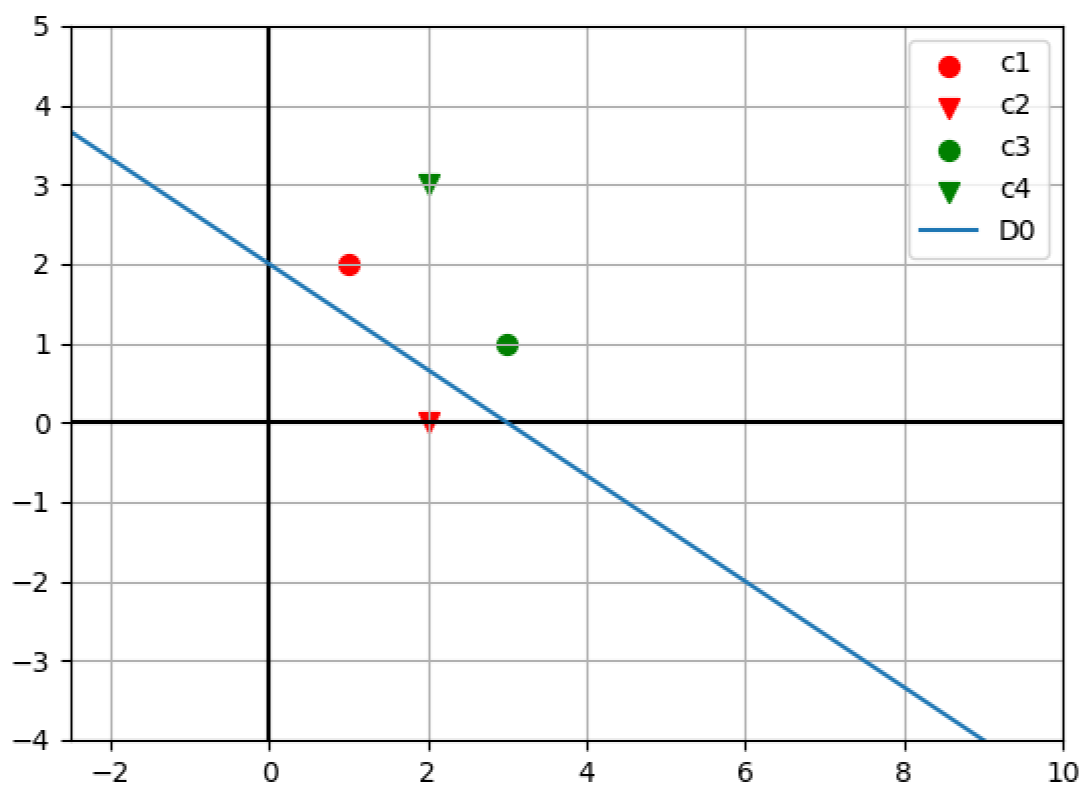
En prenant $\alpha = 0.1$ calculer les 3 premières droites obtenues par l'algorithme du perceptron donné dans l'algorithme 1.

$$\begin{cases} x_1 = 0 \Longrightarrow x_2 = 2 \\ x_1 = 3 \Longrightarrow x_2 = 0 \end{cases}$$

$$D_0 : x_2 = \frac{-2}{3} x_1 + 2$$

$$\Leftrightarrow D_0 : -2x_1 - 3x_2 + 6 = 0$$

In [6]:
```
c1=(x1,y1,label1)=(1,2,1)
c2=(x2,y2,label2)=(2,0,1)
c3=(x3,y3,label3)=(3,1,-1)
c4=(x4,y4,label4)=(2,3,-1)
c5=(0,0,0)
droites=[(-2/3,2)]
afficherDroites(c1,c2,c3,c4,c5,droites,'droites')
```



Remarque : Seulement c_1 est mal classé selon la droite D1

Appliquons l'algorithme d'apprentissage "online" du perceptron

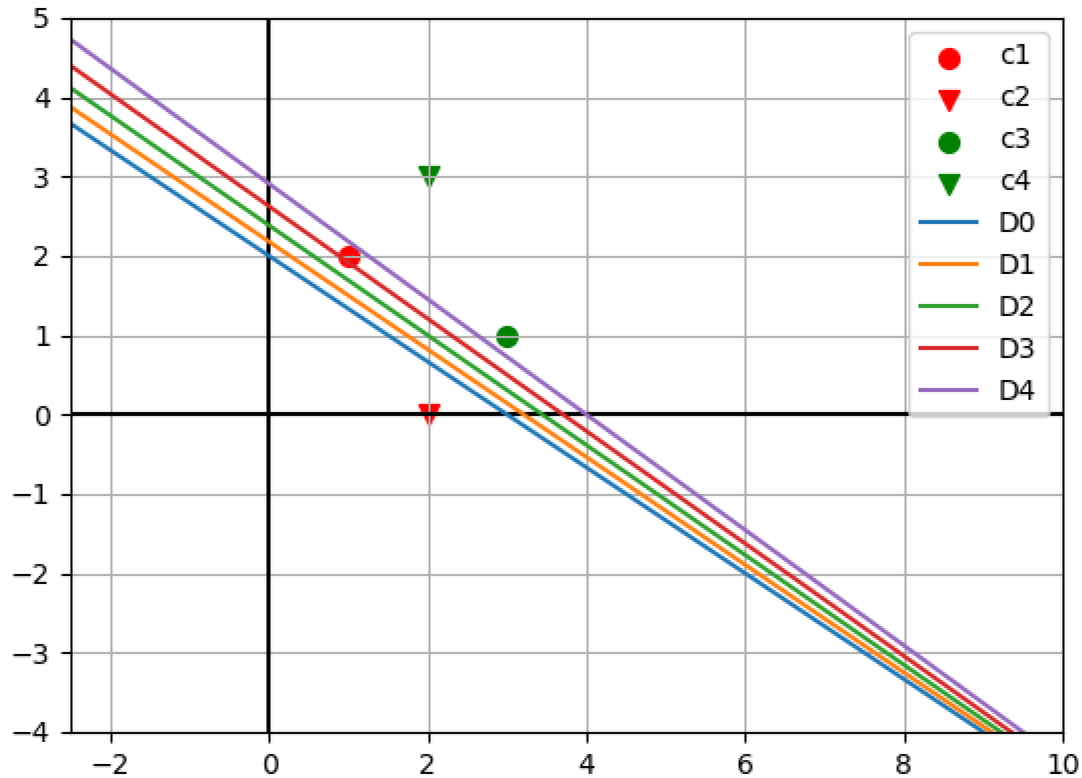
En partant de D_0, posons $W^{(1)} = \begin{bmatrix} -2 \\ -3 \end{bmatrix}$, $W_0 = 6$ et $\alpha = 0.1$

In [7]:
```
def algo_perceptron(c1,c2,c3,c4,c5,alpha,W,w0):
    nb_calcul=0
    points=[c1,c2,c3,c4]
    w1,w2=W
    cpt=0
    i=0
    while cpt!=4: # on s'arrete si les 4 points sont bien classés
        a,b,y=points[i]
        nb_calcul+=1
        if (w1*a+w2*b+w0)*y <= 0: # condition de l'algo
            w1=w1+y*a*alpha
            w2=w2+y*b*alpha
            w0=w0+alpha*y
            print(f'x{i+1} est mal classé : nouveau W={round(w1,2),round(w2,2)} et W0={r
            droites.append((-w1/w2,-w0/w2)) #on rajoute la droite correspondantes a la l
            cpt=0
        else:
            cpt+=1
        i=(i+1)%4
    print(f'Tous les points sont bien classés apres {nb_calcul} opérations pour W={w1,w2
    return droites

c1=(x1,y1,label1)=(1,2,1)
c2=(x2,y2,label2)=(2,0,1)
c3=(x3,y3,label3)=(3,1,-1)
c4=(x4,y4,label4)=(2,3,-1)
c5=(0,0,0)
W=(w1,w2)=(-2,-3)
w0=6
droites=[(-w1/w2,-w0/w2)]
alpha=0.1
droites=algo_perceptron(c1,c2,c3,c4,c5,alpha,W,w0)
```

```
x1 est mal classé : nouveau W=(-1.9, -2.8) et W0=6.1
x1 est mal classé : nouveau W=(-1.8, -2.6) et W0=6.2
x1 est mal classé : nouveau W=(-1.7, -2.4) et W0=6.3
x1 est mal classé : nouveau W=(-1.6, -2.2) et W0=6.4
Tous les points sont bien classés apres 17 opérations pour W=(-1.5999999999999996, -2.19
99999999999993) et w0=6.399999999999999
```

In [8]:
```
afficherDroites(c1,c2,c3,c4,c5,droites,'droites')
```



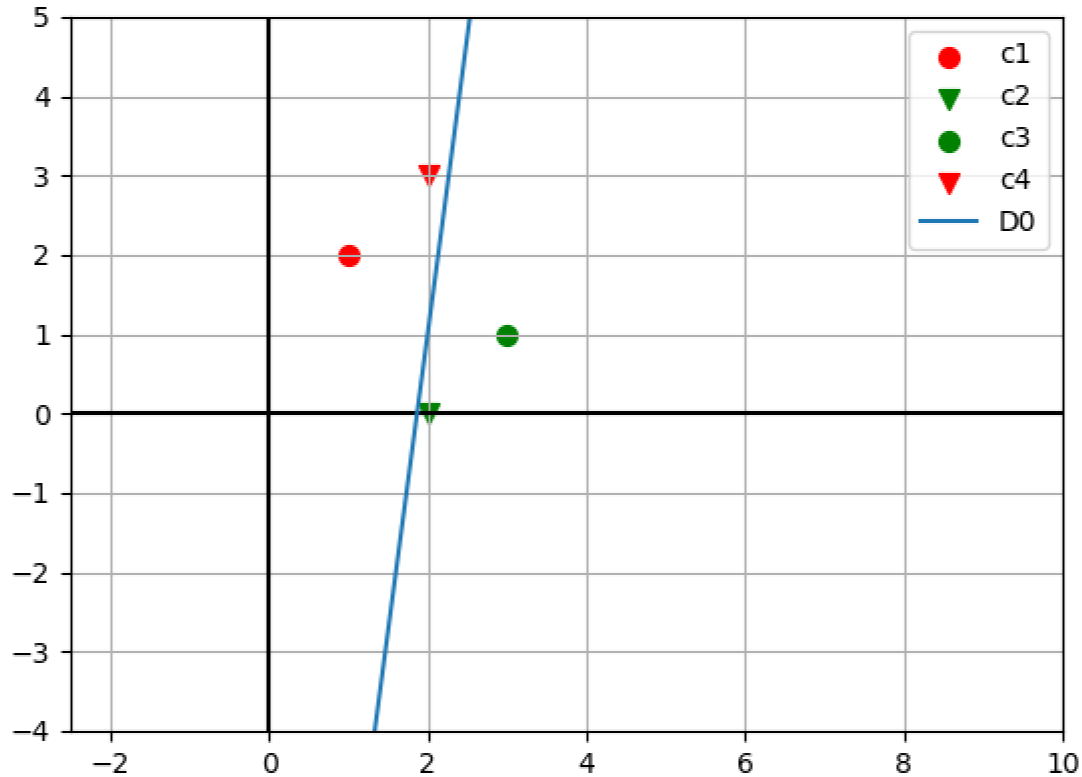
Remarque : Seulement c_1 était mal classé, réessayons en modifiant les étiquettes de telle sorte que c_4 et c_1 soit de la même classe

In [9]:
```
c1=(x1,y1,label1)=(1,2,1)
c2=(x2,y2,label2)=(2,0,-1)
c3=(x3,y3,label3)=(3,1,-1)
c4=(x4,y4,label4)=(2,3,1)
droites=[algo_perceptron(c1,c2,c3,c4,c5,alpha,W,w0)[-1]] #seulement la derniere
```

```
x1 est mal classé : nouveau W=(-1.9, -2.8) et W0=6.1
x2 est mal classé : nouveau W=(-2.1, -2.8) et W0=6.0
x4 est mal classé : nouveau W=(-1.9, -2.5) et W0=6.1
x1 est mal classé : nouveau W=(-1.8, -2.3) et W0=6.2
x2 est mal classé : nouveau W=(-2.0, -2.3) et W0=6.1
x4 est mal classé : nouveau W=(-1.8, -2.0) et W0=6.2
x2 est mal classé : nouveau W=(-2.0, -2.0) et W0=6.1
x4 est mal classé : nouveau W=(-1.8, -1.7) et W0=6.2
x2 est mal classé : nouveau W=(-2.0, -1.7) et W0=6.1
x4 est mal classé : nouveau W=(-1.8, -1.4) et W0=6.2
x2 est mal classé : nouveau W=(-2.0, -1.4) et W0=6.1
x4 est mal classé : nouveau W=(-1.8, -1.1) et W0=6.2
x2 est mal classé : nouveau W=(-2.0, -1.1) et W0=6.1
x4 est mal classé : nouveau W=(-1.8, -0.8) et W0=6.2
x2 est mal classé : nouveau W=(-2.0, -0.8) et W0=6.1
x4 est mal classé : nouveau W=(-1.8, -0.5) et W0=6.2
x2 est mal classé : nouveau W=(-2.0, -0.5) et W0=6.1
x2 est mal classé : nouveau W=(-2.2, -0.5) et W0=6.0
x2 est mal classé : nouveau W=(-2.4, -0.5) et W0=5.9
x4 est mal classé : nouveau W=(-2.2, -0.2) et W0=6.0
x2 est mal classé : nouveau W=(-2.4, -0.2) et W0=5.9
x2 est mal classé : nouveau W=(-2.6, -0.2) et W0=5.8
```

```
x2 est mal classé : nouveau W=(-2.8, -0.2) et W0=5.7
x4 est mal classé : nouveau W=(-2.6, 0.1) et W0=5.8
x2 est mal classé : nouveau W=(-2.8, 0.1) et W0=5.7
x2 est mal classé : nouveau W=(-3.0, 0.1) et W0=5.6
x4 est mal classé : nouveau W=(-2.8, 0.4) et W0=5.7
x2 est mal classé : nouveau W=(-3.0, 0.4) et W0=5.6
Tous les points sont bien classés apres 66 opérations pour W=(-3.000000000000001, 0.4000
000000000006) et w0=5.600000000000001
```

In [10]:
```
afficherDroites(c1,c2,c3,c4,c5,droites,'droites')
```



Execice 3 : Perceptron VS Kmeans

On considère le jeu de données 2D suivant dont on associe une étiquette y :

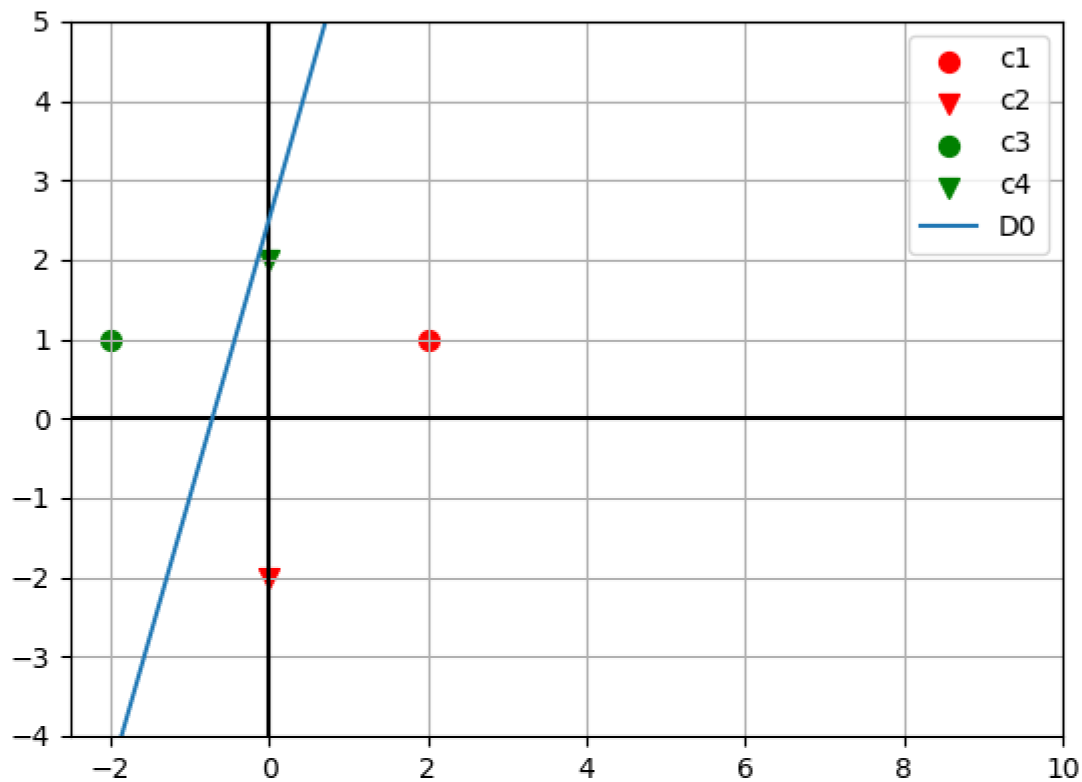
- Pour la classe (1), les points $c_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ et $c_2 = \begin{bmatrix} 0 \\ -2 \end{bmatrix}$
- Pour la classe (-1), les points $c_3 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$ et $c_4 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$
- $w_1 = -0.7$, $w_2 = 0.2$ et $w_0 = -0.5$
- $\alpha = 1$

$D_0 : -0.7x_1 + 0.2x_2 - 0.5 = 0$

$\Leftrightarrow D_0 : x_2 = 3.5x_1 + 2.5$

In [11]:
```
c1=(x1,y1,label1)=(2,1,1)
```

```
c2=(x2,y2,label2)=(0,-2,1)
c3=(x3,y3,label3)=(-2,1,-1)
c4=(x4,y4,label4)=(0,2,-1)
c5=(0,0,0)
W=(w1,w2)=(-0.7,0.2)
w0=-0.5
droites=[(-w1/w2,-w0/w2)]
alpha=1
afficherDroites(c1,c2,c3,c4,c5,droites,'droites')
```



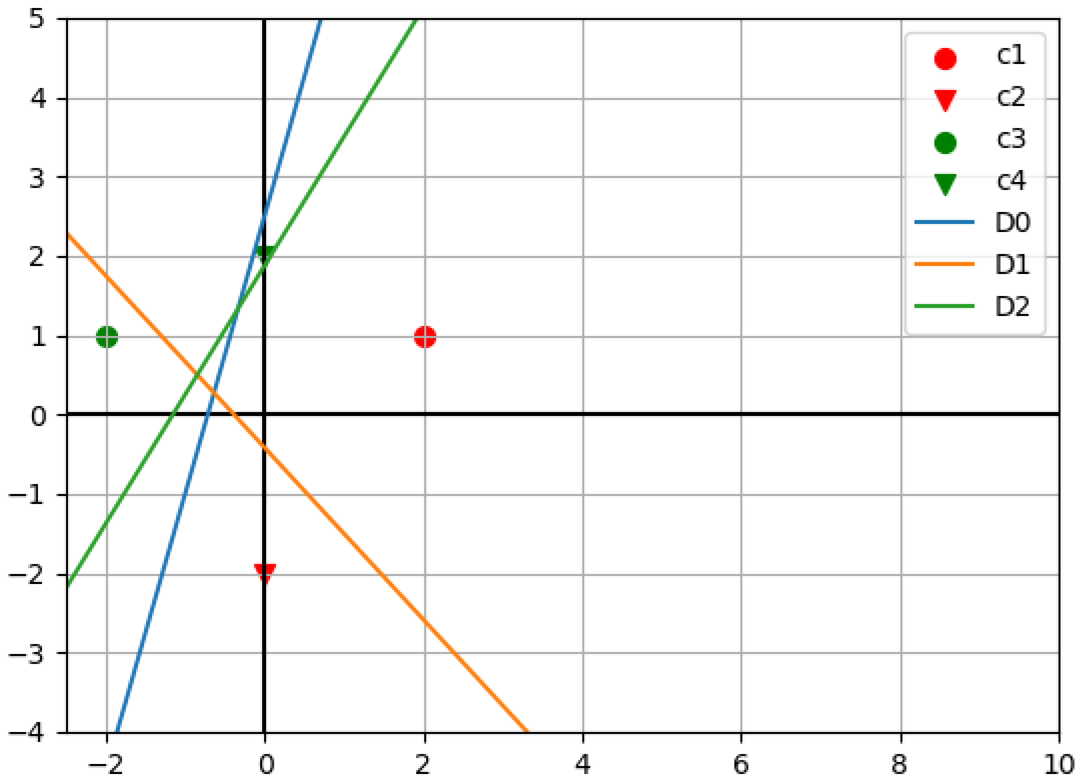
1. Calculer les poids w_1, w_2 et w_0 tels que le perceptron vérifie la base d'apprentissage

In [12]:
```
droites=algo_perceptron(c1,c2,c3,c4,c5,alpha,W,w0) #seulement la derniere
```

```
x1 est mal classé : nouveau W=(1.3, 1.2) et W0=0.5
x2 est mal classé : nouveau W=(1.3, -0.8) et W0=1.5
Tous les points sont bien classés apres 6 opérations pour W=(1.3, -0.8) et w0=1.5
```

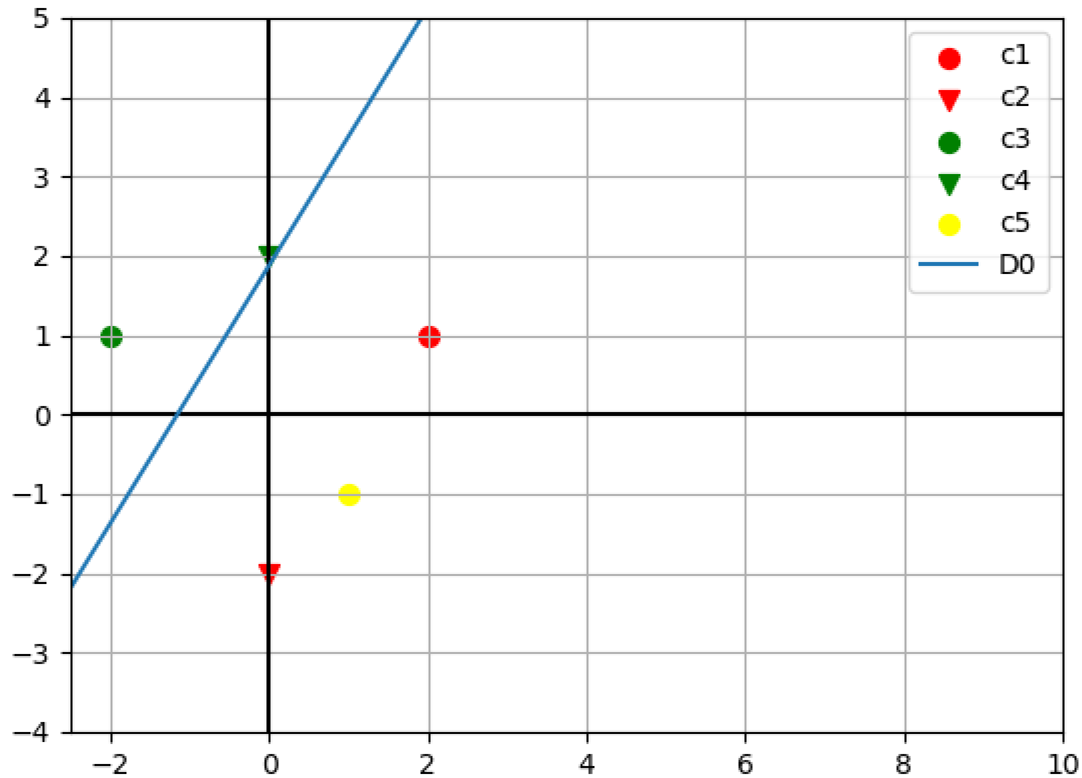
1. Représenter les données et l'hyperplan séparateur

In [13]:
```
afficherDroites(c1,c2,c3,c4,c5,droites,'droites')
```



1. On souhaite classer le point $c_5 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ A quelle classe affecteriez vous ce nouveau point?

In [14]:
```
afficherDroites(c1,c2,c3,c4,(1,-1,0),[droites[-1]],'droites')
```

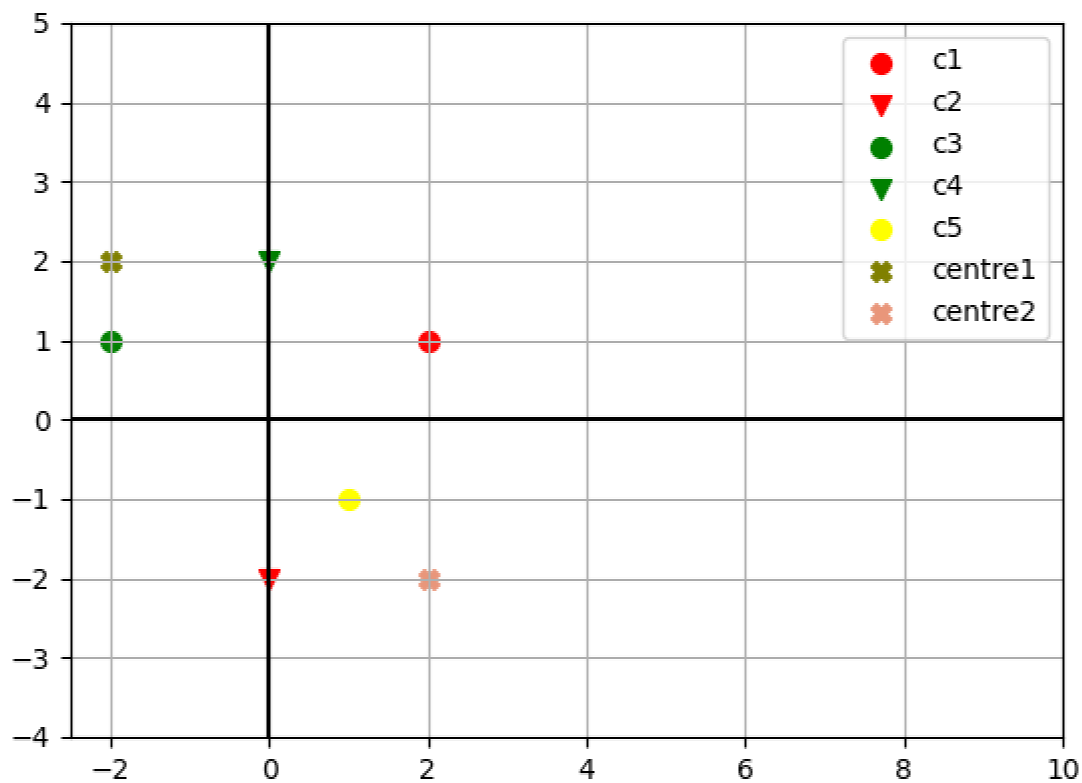


On le classerait dans la classe 1 (rouge), car le point est en dessous de la droite.

Classification non supervisée : Kmeans

1. Pour centres initiaux, les points $centre_1 = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$ et $centre_2 = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$

In [15]:
```
c_1=(-2,2,0)
c_2=(2,-2,0)
c5=(1,-1,0)
centres=[(c_1,c_2)]
points=[c1,c2,c3,c4,c5]
afficherDroites(c1,c2,c3,c4,c5,centres,'centres')
```

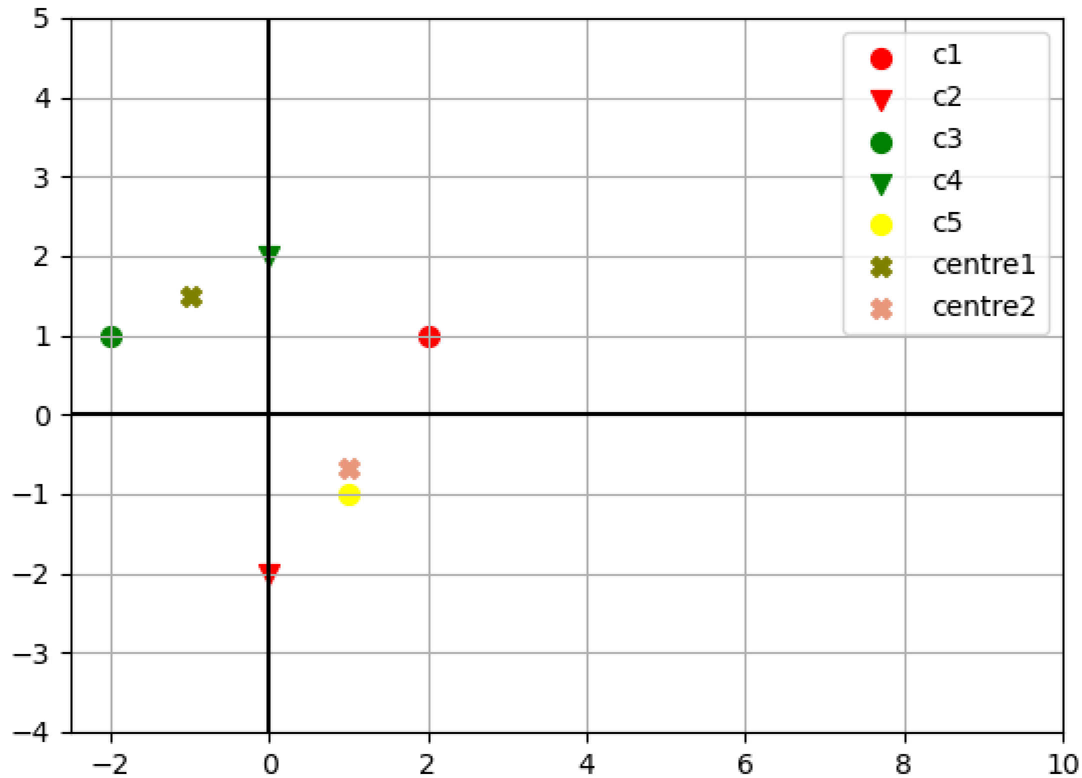


In [16]:
```
def euclidean_distance(a,b):
    '''Retourne la distance euclidienne entre a et b au carrée'''
    a1,a2,_=a
    b1,b2,_=b
    return (a1-b1)**2+(a2-b2)**2
def les_voisins(centres,points):
    '''Retourne un dictionnaire centre : point les plus proches'''
    c1,c2=centres[0]
    les_voisins={'c1':[],'c2':[]}
    for (a,b,c) in points:
        les_voisins['c1' if (euclidean_distance((a,b,c),c1) < euclidean_distance((a,b,c)
    return les_voisins
```

In [17]:
```
c_1=(-2,2,0)
c_2=(2,-2,0)
V=les_voisins(centres,points)
newC1= sum(a for (a,b) in V['c1'])/len(V['c1']),sum(b for (a,b) in V['c1'])/len(V['c1'])
newC2= sum(a for (a,b) in V['c2'])/len(V['c2']),sum(b for (a,b) in V['c2'])/len(V['c2'])
d1,d2=euclidean_distance(newC1,c_1),euclidean_distance(newC2,c_2)
while (d1+d2>0.05): #les centres ont convergé
    centres=[(newC1,newC2)]
    V=les_voisins(centres,points)
    newC1= sum(a for (a,b) in V['c1'])/len(V['c1']),sum(b for (a,b) in V['c1'])/len(V['c
    newC2= sum(a for (a,b) in V['c2'])/len(V['c2']),sum(b for (a,b) in V['c2'])/len(V['c
    c_1,c_2=centres[0]
    d1,d2=euclidean_distance(newC1,c_1),euclidean_distance(newC2,c_2)
    print(f'Nouveaux centres : ({newC1[0]},{newC1[1]}) et ({newC2[0]},{newC2[1]})')
```

```
Nouveaux centres : (-1.0,1.5) et (1.0,-0.6666666666666666)
```

In [18]:
```
afficherDroites(c1,c2,c3,c4,c5,centres,'centres')
```



In [19]:
```
print(les_voisins(centres,points))
```

```
{'c1': [[-2, 1], [0, 2]], 'c2': [[2, 1], [0, -2], [1, -1]]}
```

Le point c_5 serait aussi dans la classe 1 (rouge).