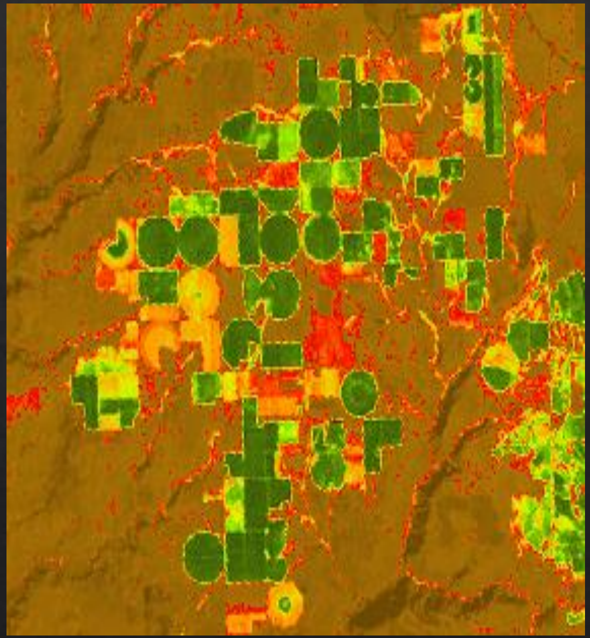
Transport optimal
Application en analyse des images satellitaires

Objectifs:

- Elaboration des histogrammes de fréquences des NDVI à partir d'une base de données fournie par le Centre National de Cartographie et de Télédétection
- Elaborer une relation entre des histogrammes de référence et un histogramme dont on veut estimer le rendement en se basant sur la théorie du transport optimal



Numéro d'inscription : 18175

Plan

Base de Données

Traçage des histogrammes de fréquence et détermination des histogrammes de référence

Extrapolation du motif

Détermination des coordonnées Barycentriques de Wasserstein

Détermination d'une relation liant la valeur moyenne des NDVI et le rendement

Echantillonnage des données et traçage en 2D

Détermination d’un coefficient liant les distributions de références et les distributions intermédiaires

Calcul d’erreur et comparaison

Bases de données

000081 veut dire annee 2000 et jour 81			
		21/03/2000	06/04/
	date	81	97
		2000081	2000
X	Y	B1	B2
10.0183661013	6.662023246	.74639999866485	.3764999
10.0160207728	6.659677918	0.635100007057	.3362999
10.0183661013	6.659677918	0.635100007057	0.3491999
10.0207114298	6.659677918	.70229995250701	0.3491999
10.0230567584	6.659677918	.73079997301101	.3612999
10.0160207728	6.657332589	.71169996261596	.3619999
10.0183661013	6.657332589	.69669997692108	.3458999
10.0207114298	6.657332589	.69669997692108	.3458999
10.0230567584	6.657332589	.73759996891021	.3483999
10.0113301153	6.654987261	.61369997262954	.3610999
10.0136754443	6.654987261	.69059997797012	.3455999

NDVI : indice de végétation par différence normalisée

PIR: Réflectance dans la bande proche Infra-rouge

R: Réflectance dans la bande rouge

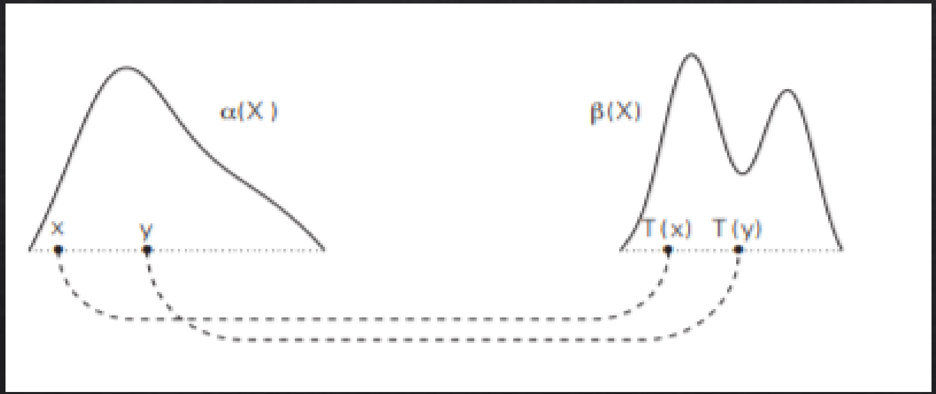
NDVI: $\frac{PIR-R}{PIR+R}$

Transport optimal

- l'étude du transfert **optimal** de matière et à l'allocation **optimale** de ressources

- Formulation de Monge : recherche du plan qui réalise $\inf\left\{\int_X c(x,T(x))\,\mathrm{d}\mu(x)\,\middle|\,T_*(\mu)=\nu\right\}$

Brouette de Monge



- Formulation de Kantorovitch : recherche de la mesure qui vérifie $\inf\left\{\int_{X\times Y} c(x,y)\,\mathrm{d}\gamma(x,y)\,\middle|\,\gamma\in\Gamma(\mu,\nu)\right\}$

Transport optimal: barycentres de Wasserstein

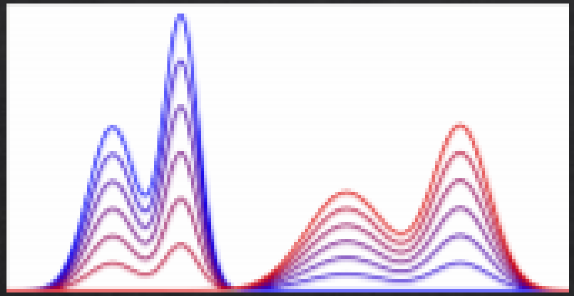
- L'espace de Wasserstein (sur R^d), $P_2(R^d)$ est par définition l'ensemble des mesures de probabilité sur R^d, de second moment fini, muni de la métrique W_2 définie par le problème de transport optimal quadratique $W_2^2(\mu,\nu) := \inf\left\{ \int_{\mathbb{R}^d \times \mathbb{R}^d} |x-y|^2 \mathrm{d}\gamma(x,y),\ \gamma \in \Pi(\mu,\nu) \right\},\ \forall (\mu,\nu) \in \mathscr{P}_2(\mathbb{R}^d)^2$
- $\Pi(\mu,\nu)$: l'ensemble des plans de transport entre μ et ν c'est-à-dire l'ensemble des mesures de probabilité sur $R^d \times R^d$ ayant μ et ν comme marginal
- Soient $\nu_1,\ldots\nu_N$ des éléments $P_2(R^d)$ et $\lambda=(\lambda_1,\ldots,\lambda_N)\in R+N$ des poids positifs normalisés par $\sum_{i=1}^{N} \lambda i=1$, un barycentre dans l'espace de Wasserstein des mesures νi avec les poids λi est un minimiseur de $J_\lambda(\mu) := \sum_{i=1}^{N} \lambda_i W_2^2(\nu_i,\mu)$

◈ Dans le cadre de cette étude on prendra **<u>d=1</u>** ;il suffit donc de minimiser $\alpha W_2^2(\nu, \mu)$.

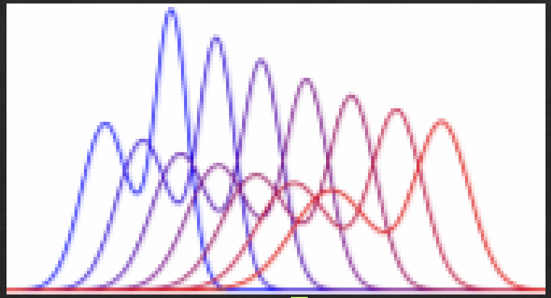
On considèrera le poids α comme le coefficient barycentrique

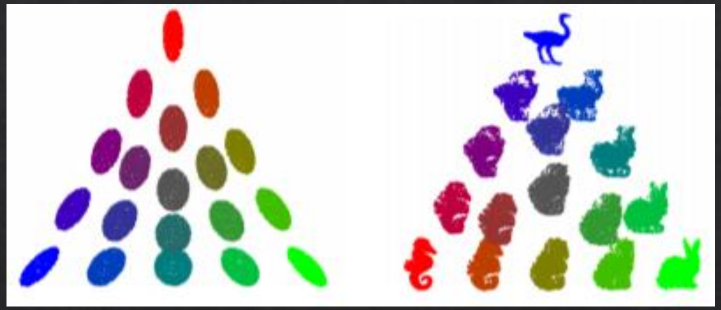
Applications :

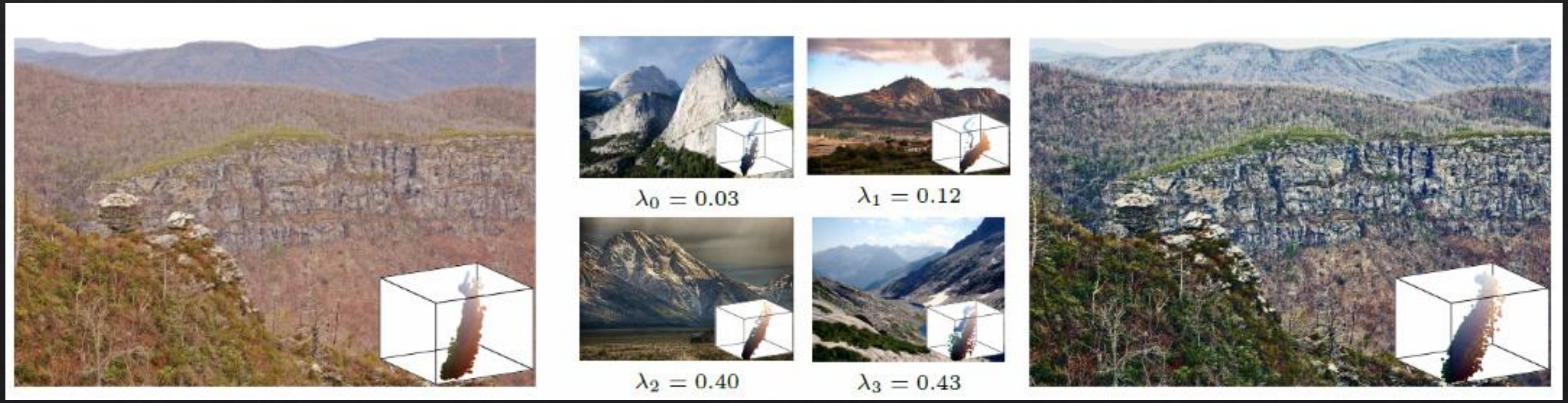
Interpolation avec L2



Interpolation de Wasserstein





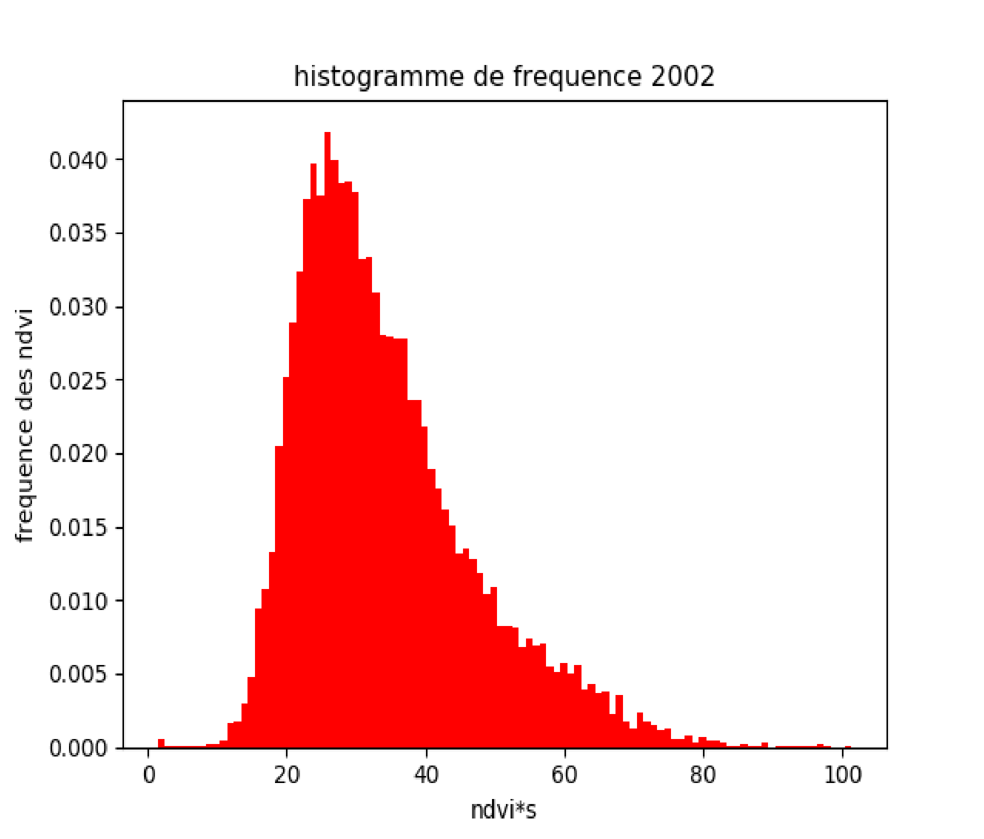


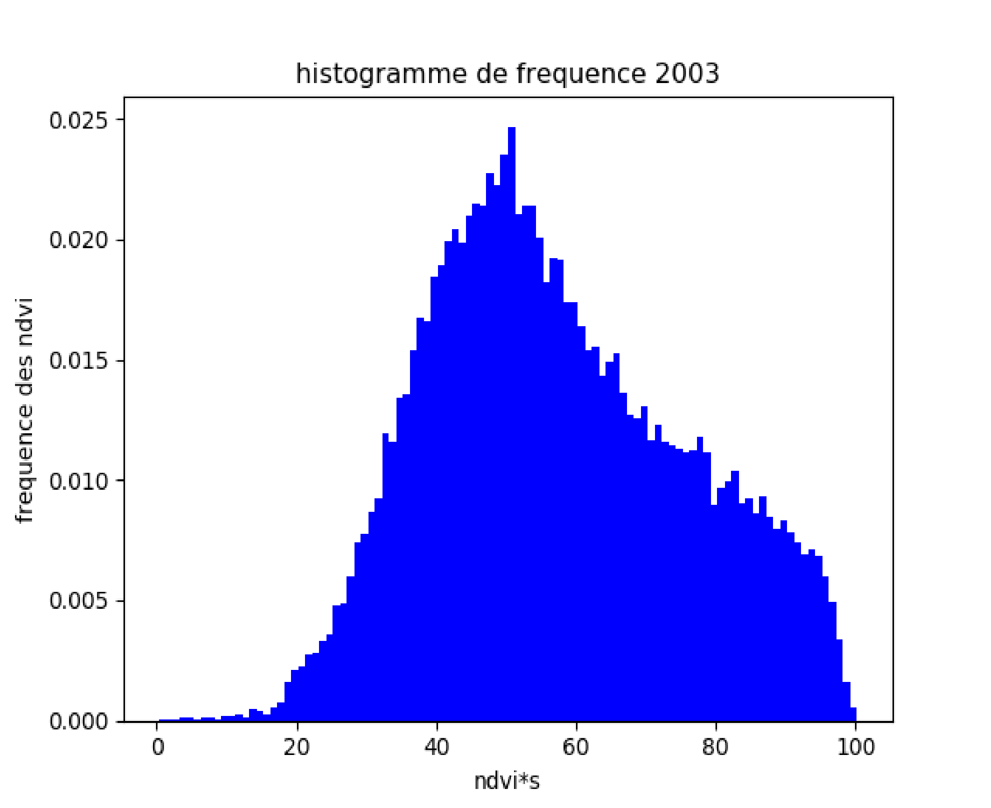
Source: Bonneel et al., 2016

Histogrammes de référence

année	Rdt Blé dur Q/ha
2000	9.0
2001	12.0
2002	1.0
2003	24.0
2004	20.0
2005	21.0
2006	15.0
2007	22.0
2008	11.0
2009	17.0
2010	4.0
2011	19.0
2012	22.0
2013	7.0
2014	23.0
2015	13.6
2016	4.1
2017	19.3

Fréquence élevée pour des valeurs élevées de NDVI





Fréquence élevée pour des valeurs faibles de NDVI

Curve fit

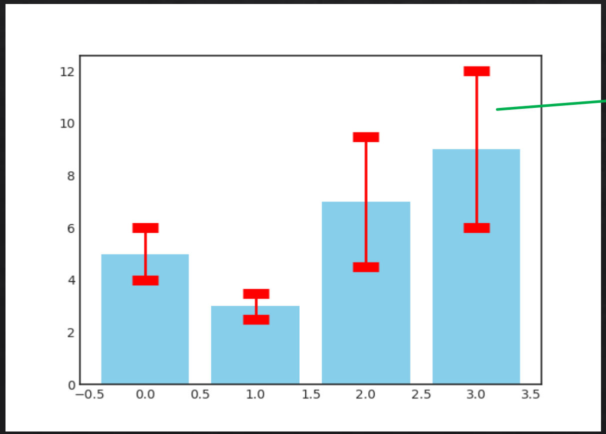
Ajustement non linéaire

$$Ae^{\frac{(xi-\mu i)^2}{2\sigma^2}}$$

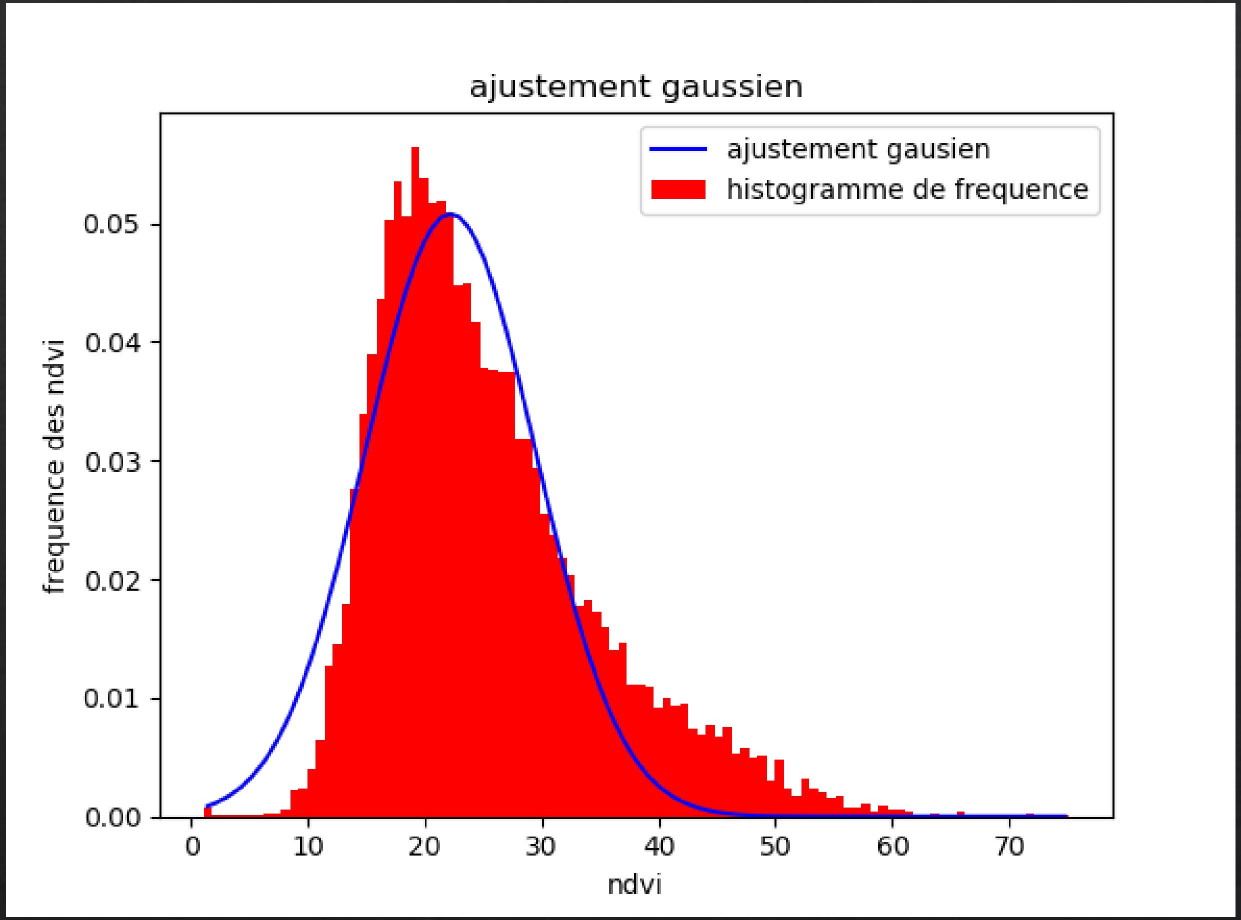
Donnée réelle

$$F(\mu,\sigma)=\sum_{i=1}^{N}(yi-f(xi))^2$$

: *recherche du minimum de F en calculant sa matrice hessienne*

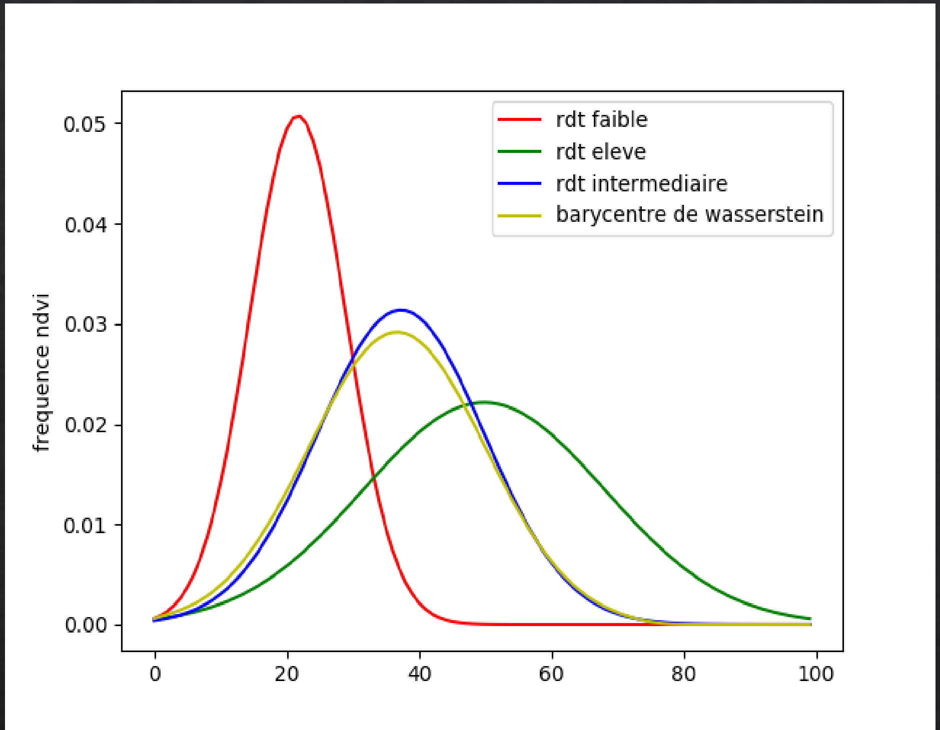


Barre d'erreur

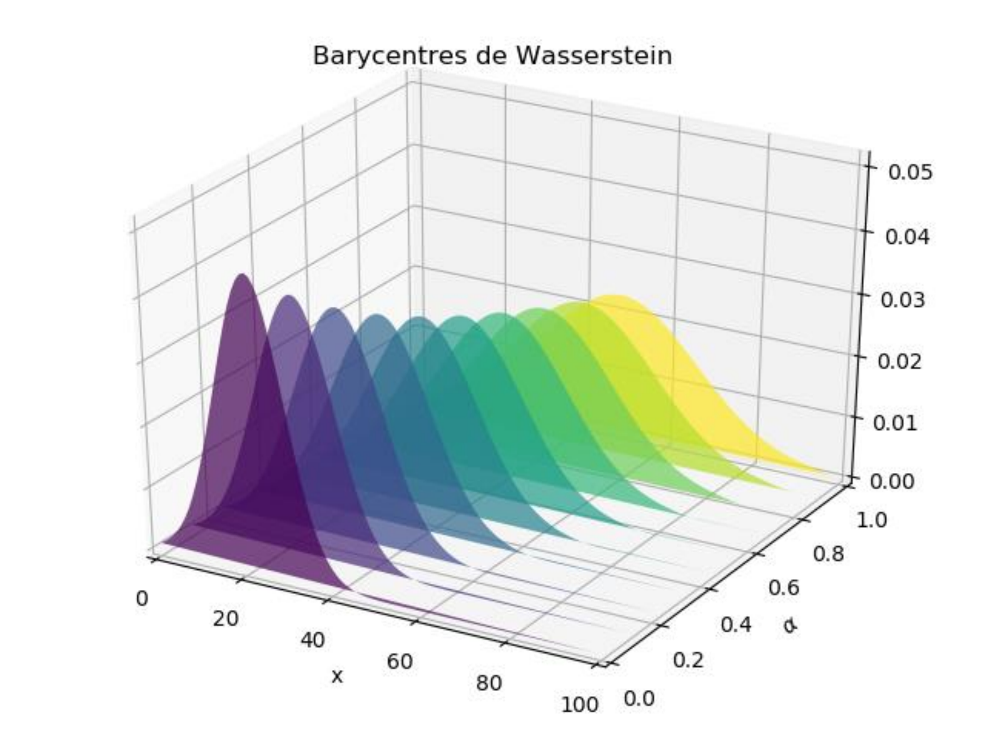


Approximation de l'histogramme par une gaussienne

Barycentres de Wasserstein



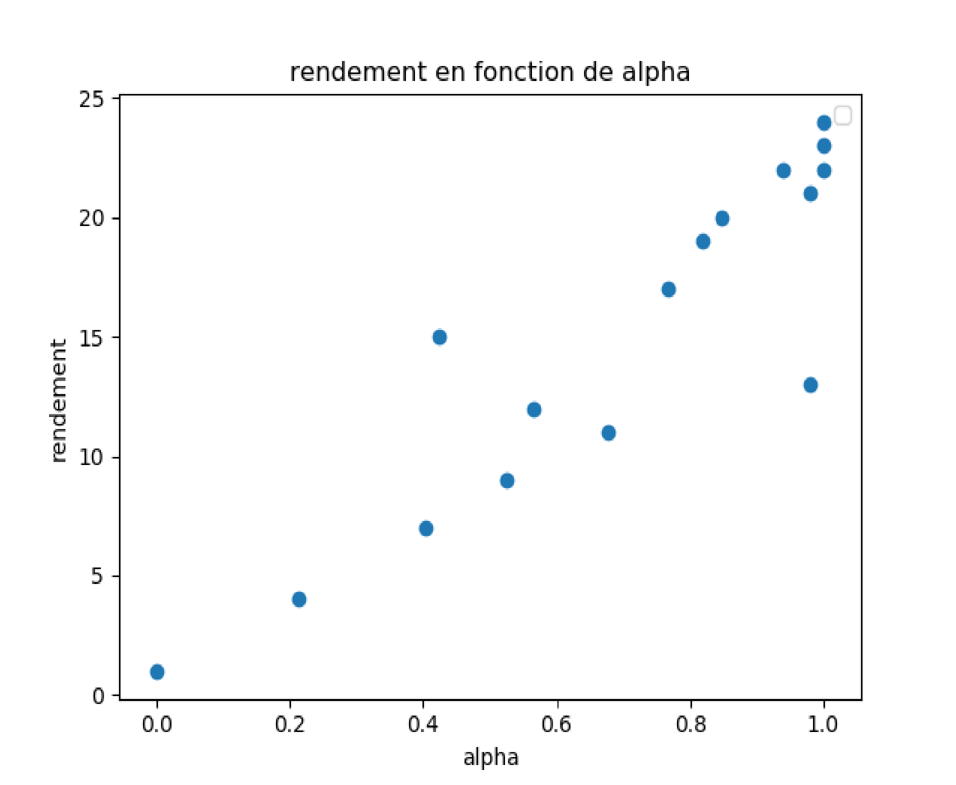
Détermination du barycentre le plus proche de la courbe intermédiaire



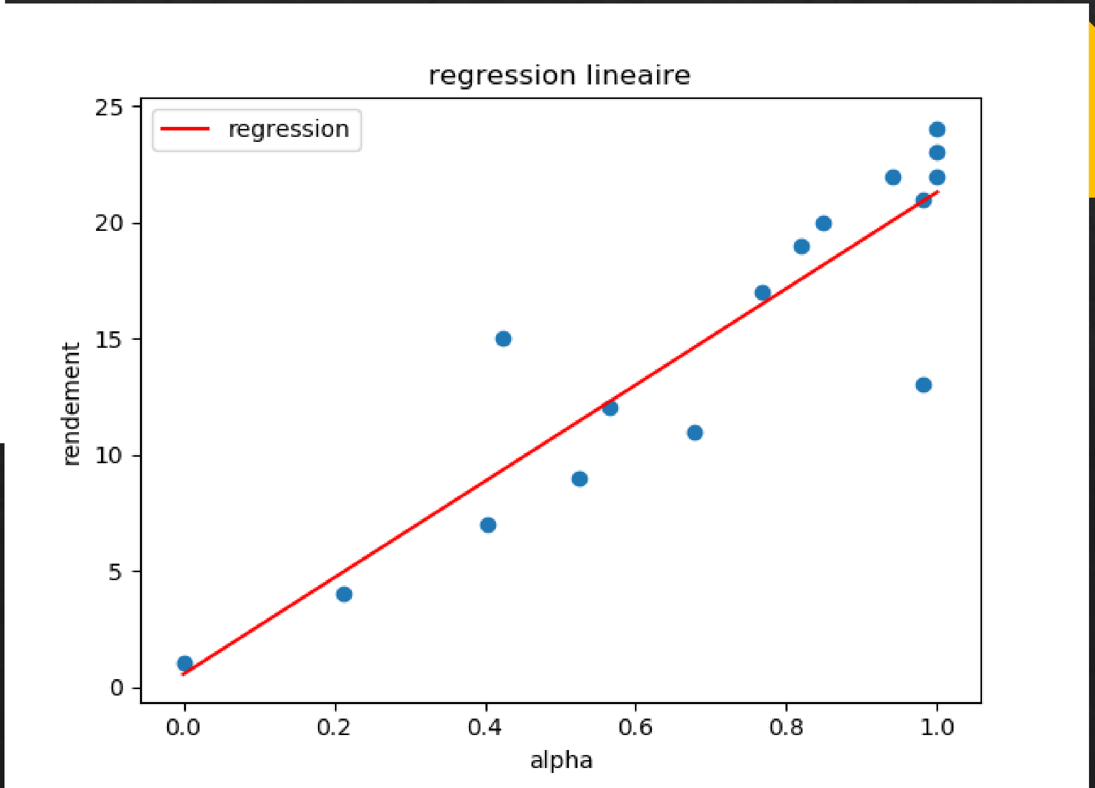
Interpolation barycentrique
Alpha représente le coefficient barycentrique

Le but est de déterminer les alpha de toutes les courbes de la base en cherchant le barycentre de Wasserstein le plus proche de chaque courbe

Rendement en fonction des coefficients barycentriques

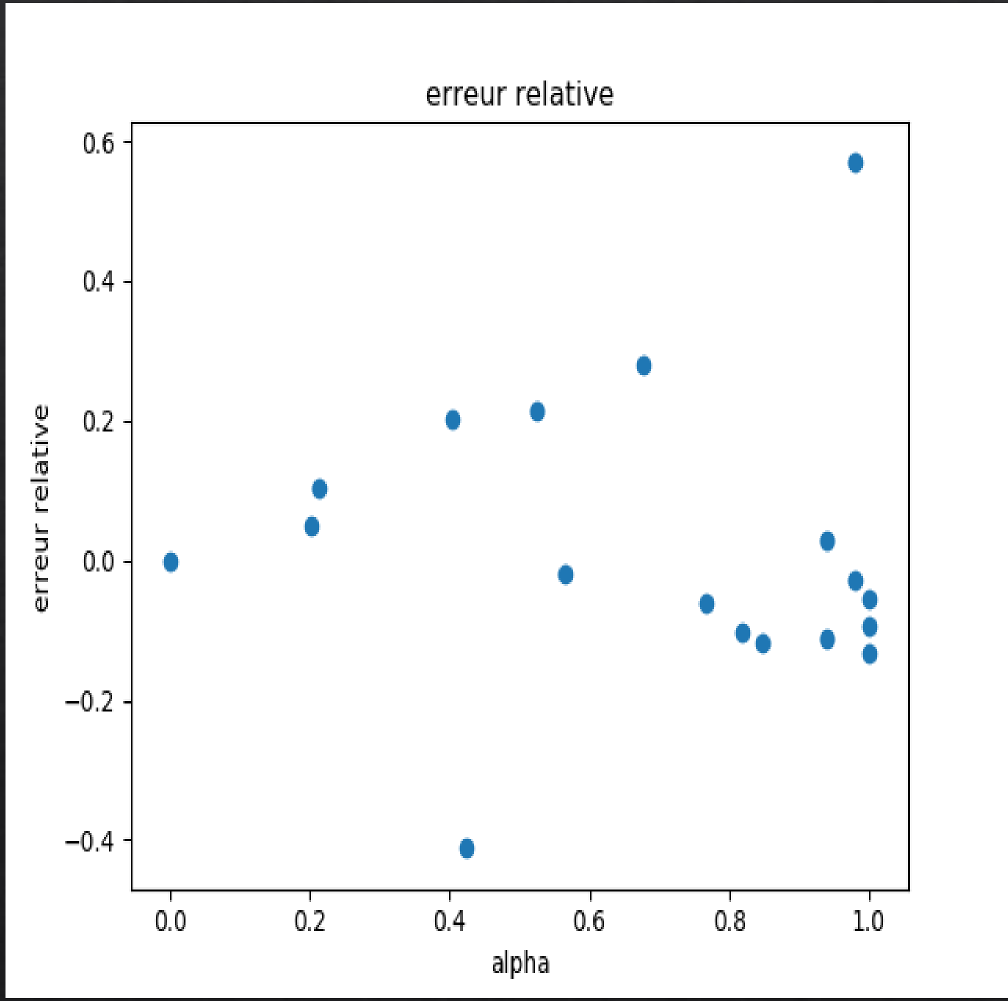


1



2

$$\frac{image\ du\ point\ par\ la\ droite\ - valeur\ mesurée}{|valeur\ mesurée|}$$



Moyenne de |erreur|:10.2%

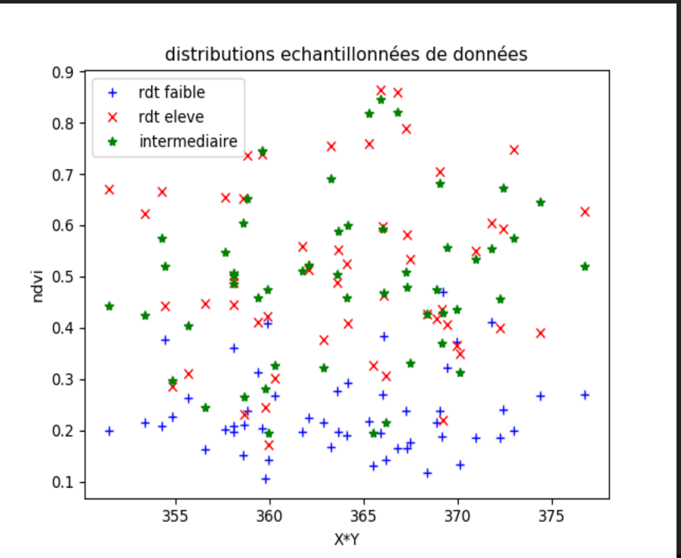
Prévisions

	Alpha	Rdt réel Q/ha	Rdt simulé Q/ha
Année 2016	0.2020202	4.100	4.195
Année 2017	0.93939393	19.300	19.508

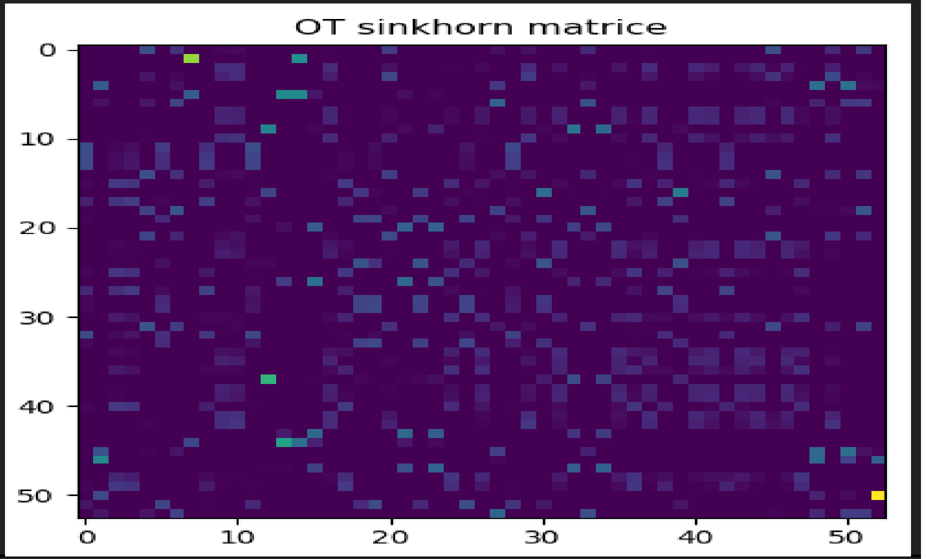
Erreur année 2016	Erreur année 2017
2.32%	1.07%

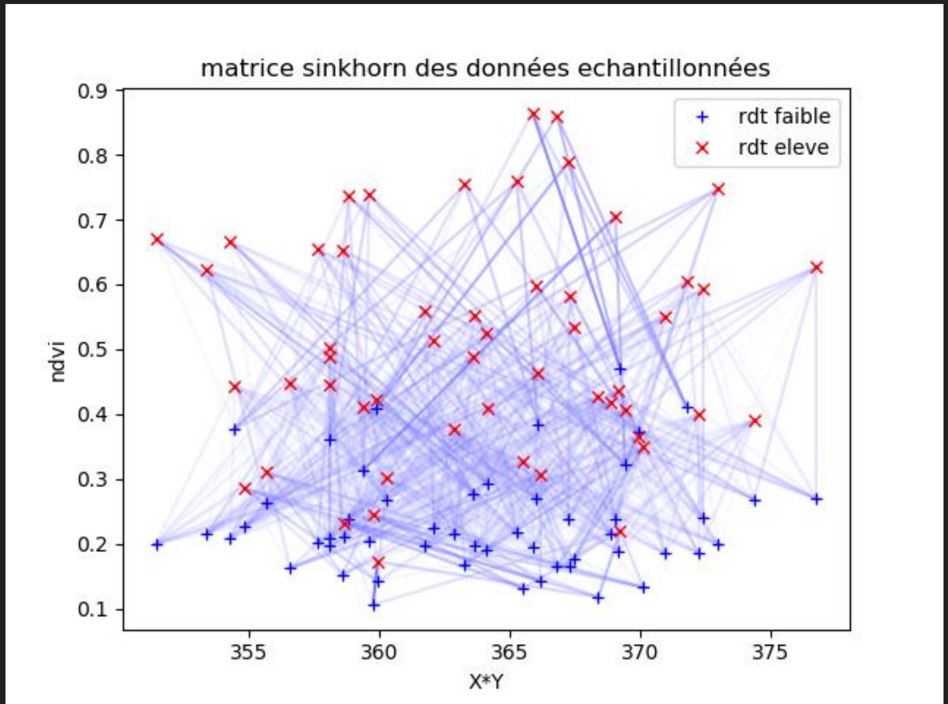
Transport entre distributions empiriques

Distribution empirique des données réelles

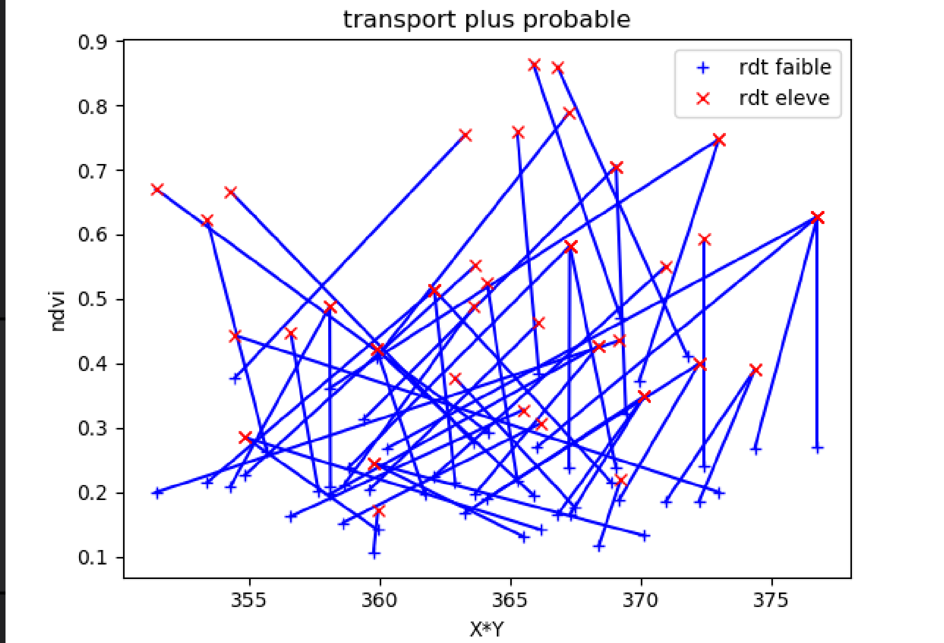


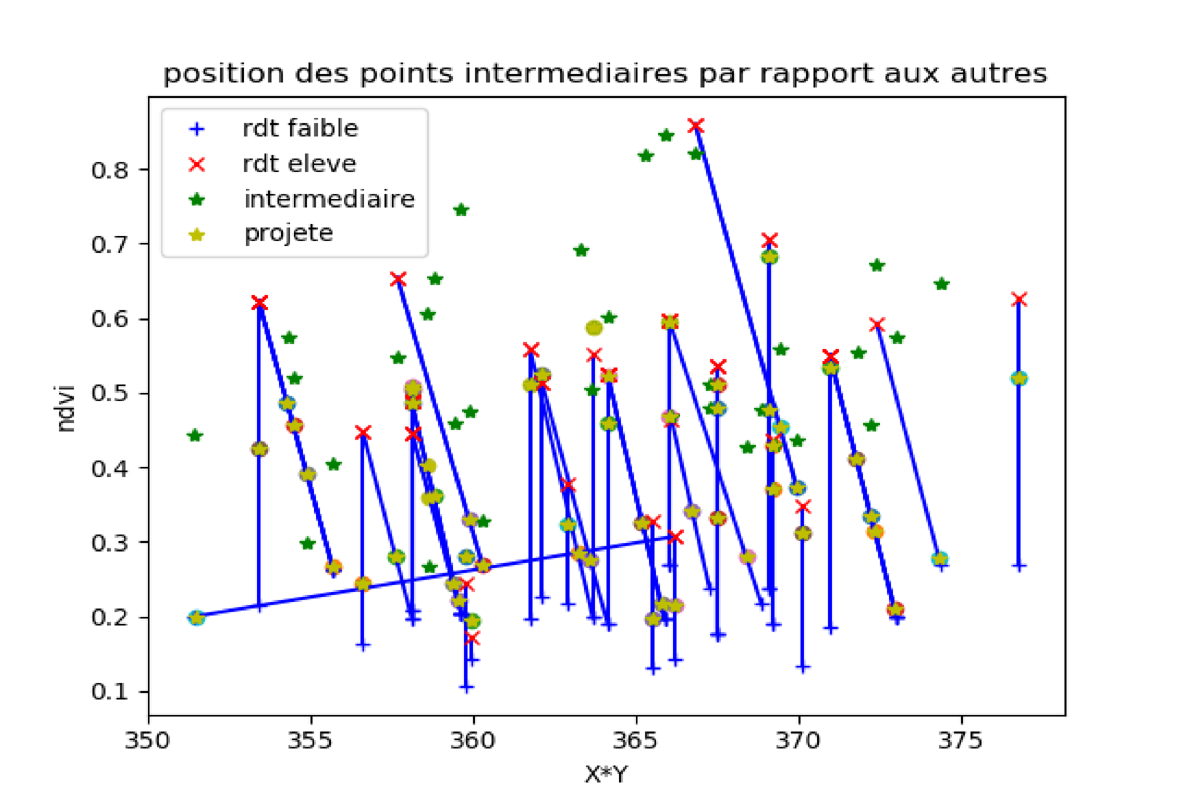
Les droits les plus sombres représentent le transport le plus probable





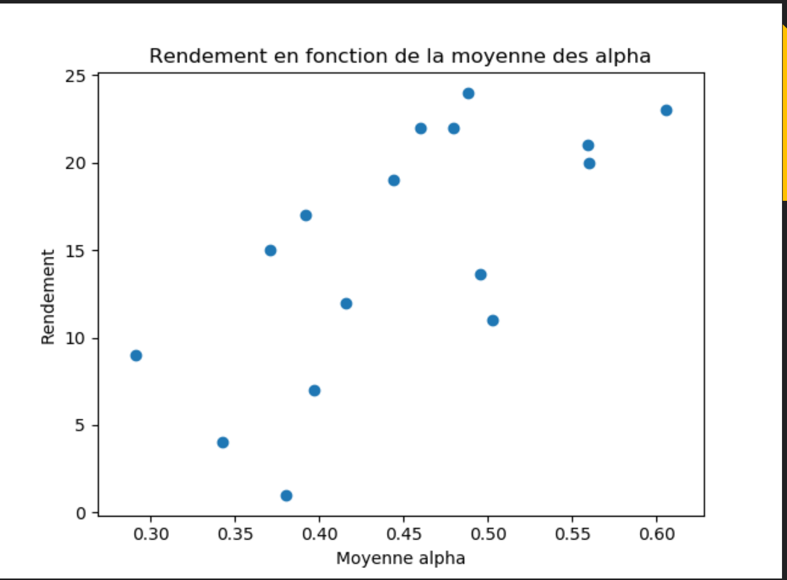
Extraction des droites





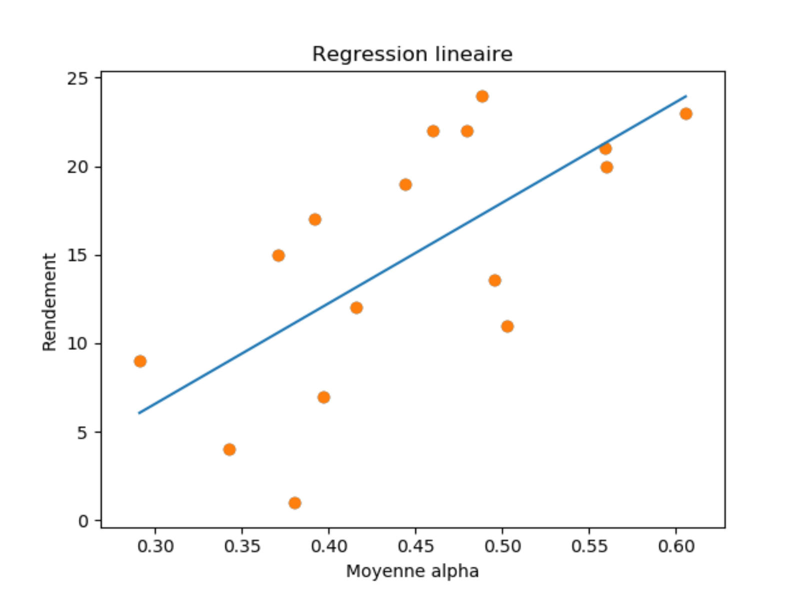
1

Projection sur la droite la plus proche



2

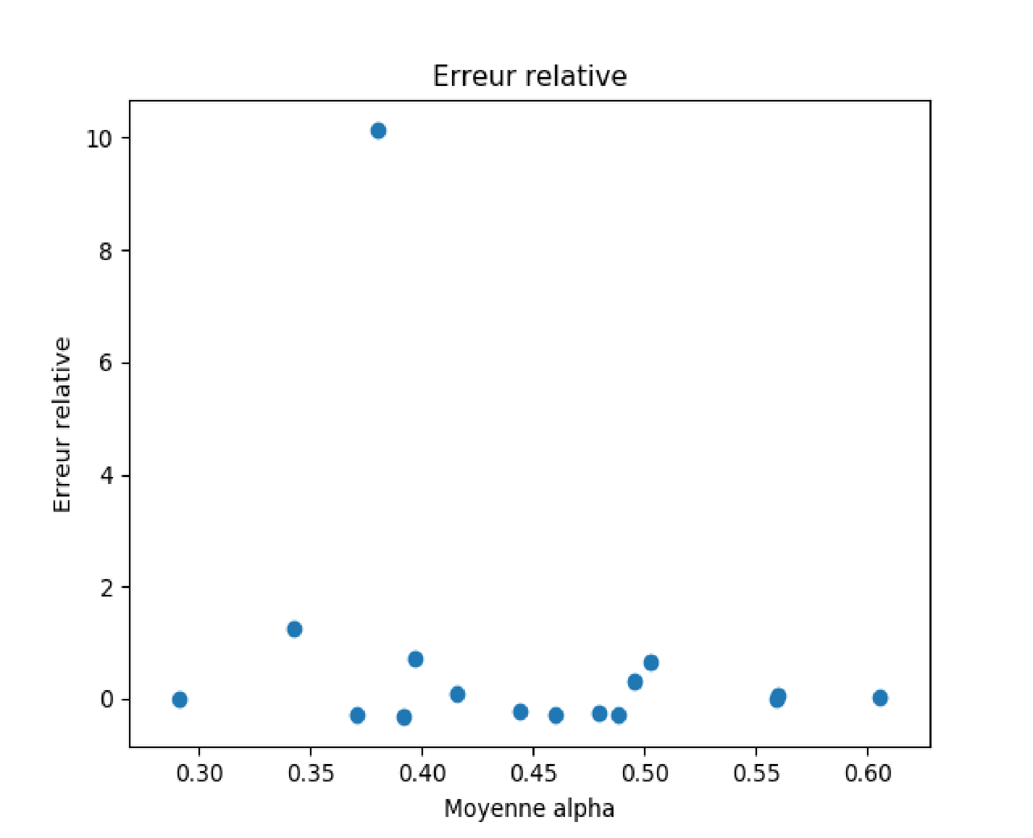
Alpha est la distance entre le projeté au point appartenant à la distribution intermédiaire et le point de rendement faible



3

Régression linéaire

$$\frac{image\ du\ point\ par\ la\ droite\ -\ valeur\ mesurée}{|valeur\ mesurée|}$$



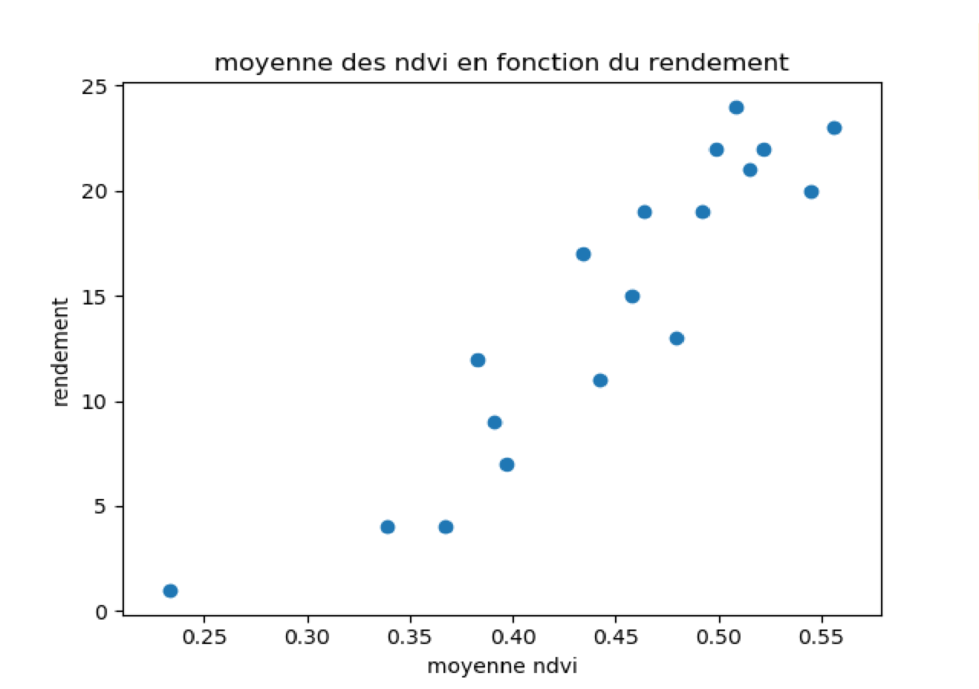
Moyenne de |erreur|:82,4%

Prévisions

	Moyenne alpha	**Rdt réel Q/ha**	**Rdt simulé Q/ha**
Année 2016	0.32942849	4.1000	9.3007
Année 2017	0.41315749	19.3000	15.0559

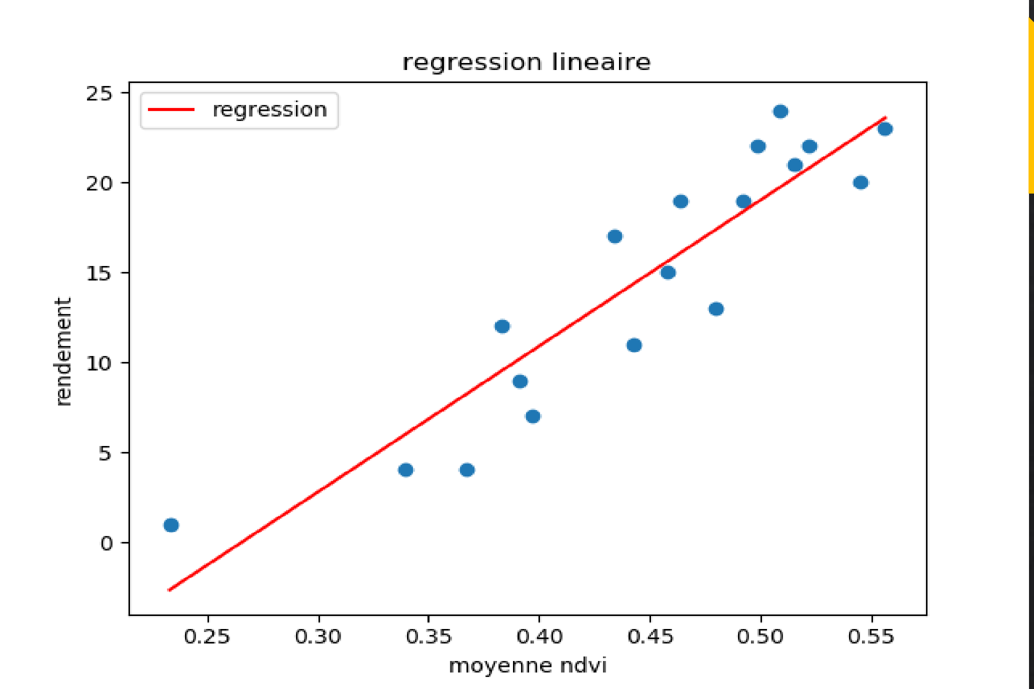
|Erreur| année 2016	**|Erreur| année 2017**
101,3%	32,5%

Rendement en fonction de la moyenne des NDVI



1

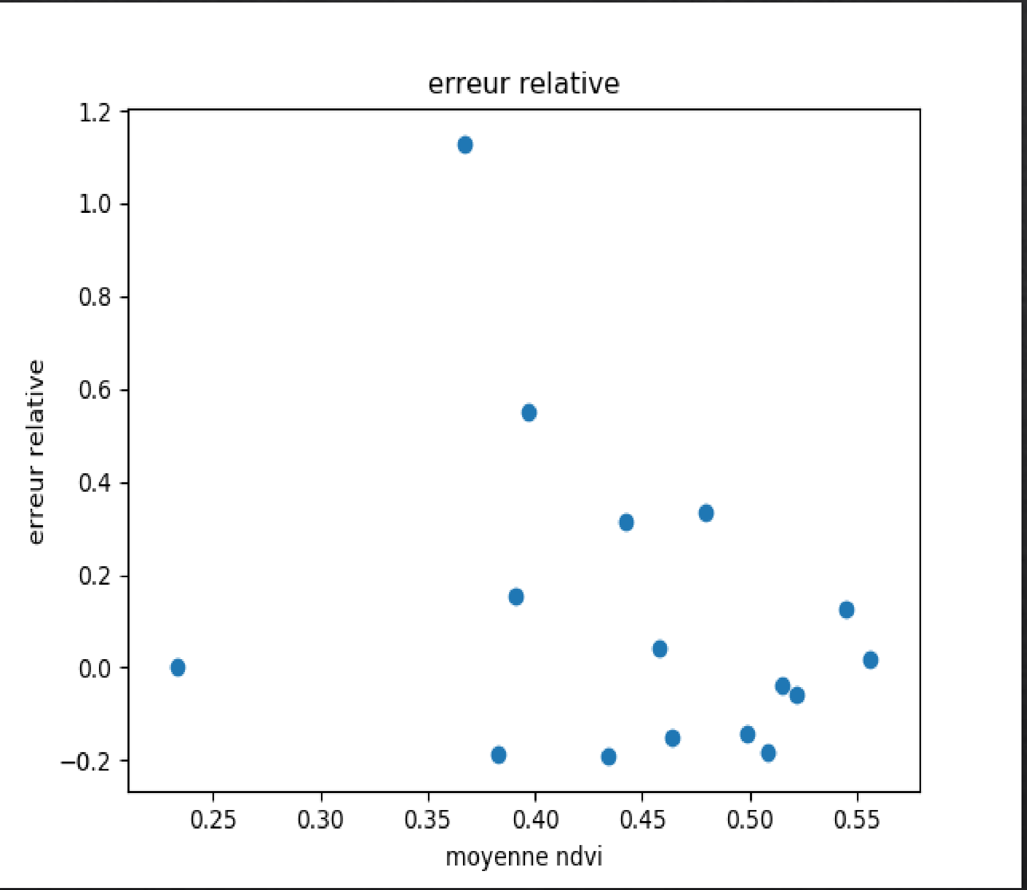
Rendement moyen de chaque année en fonction des moyennes des NDVI correspondants



2

Ajustement linéaire des points obtenus

$$\frac{\textit{image du point par la droite} - \textit{valeur mesurée}}{|\textit{valeur mesurée}|}$$



Moyenne de |erreur|:10,7%

Prévisions

	moyenne NDVI	Rdt réel Q/ha	Rdt simulé Q/ha
Année 2016	0.3392905581	4.100	6.305
Année 2017	0.4923113120	19.300	18.363

|Erreur| année 2016	|Erreur| année 2017
53.7%	4.8%

Comparaison des résultats/Conclusion

Méthode du barycentre de Wasserstein 1D

Erreur année 2016	Erreur année 2017
2,32%	1,07%

Moyenne de |erreur|:10,2%

C'est la méthode la plus précise

Méthode du transport entre données empiriques

|Erreur| année 2016	|Erreur| année 2017
101,3%	32,5%

Moyenne de |erreur|:82,4%

Méthode de la moyenne des NDVI

|Erreur| année 2016	|Erreur| année 2017
53%	4,8%

Moyenne de |erreur|:10,7%

Programmes informatiques

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
dataset = pd.read_excel("hitogram_mod13q1_200081_to201897_mod.xlsx",sheet_name="histogram_mod13q1_zaghouan")
type(dataset["Unnamed: 1"].values)
f=dataset.iloc[3:,8].values
l=[]
for i in range(len(f)):
    if f[i][0]=='0' and f[i][1]=='.':
        l.append(float(f[i]))
n=len(l)
k=[l[i]*134 for i in range(len(l))]
kk= plt.hist(k,100,normed=1,color='blue')
plt.title('histogramme de frequence 2003')
plt.xlabel("ndvi*s")
plt.ylabel("frequence des ndvi")
plt.show()
x = (kk[1][:-1] + kk[1][1:])/200
plt.show()
k=list(kk[0])
mean = sum(x*k)
sigma = sum(k*(x - mean)**2)
def gauss(x, a, x0, sigma):
    return a*np.exp(-(x-x0)**2/(2*sigma**2))
popt,pcov = curve_fit(gauss, x, k, p0=[1, mean, sigma] )
plt.plot(x*100, gauss(x, *popt), label='ajustement gausien',color='red')
plt.title('ajustement gaussien')
plt.xlabel("ndvi")
plt.ylabel("frequence des ndvi")
plt.legend()
plt.show()
```

Traçage des histogrammes de référence et ajustement gaussien

1/3

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
n = 100
x = np.arange(n, dtype=np.float64)
def gauss_fonction(x, a, x0, sigma):
    return a*np.exp(-(x-x0)**2/(2*sigma**2))
list1=[0.05075076,21.70256786,7.29220092]
list2=[2.21551895e-02,4.98199892e+01,1.82610047e+01]
list3=[3.13771210e-02,3.72675660e+01,1.26022234e+01]
a1=gauss_fonction(x,*list1)
a2=gauss_fonction(x,*list2)
a3=gauss_fonction(x,*list3)
plt.plot(x,a1,'r',label="rdt faible")
plt.plot(x,a2,'g',label="rdt eleve ")
plt.plot(x,a3,'b',label="rdt intermediaire ")
A = np.vstack((a1, a2)).T
n_distributions = A.shape[1]
def distance_euclidienne(X, Y, squared=False):
    XX = np.einsum('ij,ij->i', X, X)[:, np.newaxis]
    YY = np.einsum('ij,ij->i', Y, Y)[np.newaxis, :]
    distance = np.dot(X, Y.T)
    distance *= -2
    distance += XX
    distance += YY
    np.maximum(distance, 0, out=distance)
    if X is Y:
        distance.flat[::distance.shape[0] + 1] = 0.0
    return distance if squared else np.sqrt(distance, out=distance)
def dist(x1, x2=None, metric='sqeuclidean'):
    if x2 is None:
        x2 = x1
    if metric == "sqeuclidean":
        return distance_euclidienne(x1, x2, squared=True)
    return cdist(x1, x2, metric=metric)
def dist0(n, method='lin_square'):
    res = 0
    if method == 'lin_square':
        x = np.arange(n, dtype=np.float64).reshape((n, 1))
        res = dist(x, x)
    return res
```

Traçage des courbes de références et la courbe intermédiaire et détermination du barycentre le plus proche

2/3

```python
M =dist0(n)
M /= M.max()
def Barregeometrique(poids, toutesdistribT):
    assert(len(poids) == toutesdistribT.shape[1])
    return np.exp(np.dot(np.log(toutesdistribT), poids.T))
def Moyennegeometrique(toutesdistribT):
    return np.exp(np.mean(np.log(toutesdistribT), axis=1))
def barycentre(A, M, reg, poids=None, numItermax=1000,
               stopThr=1e-4, verbose=False, log=False):
    #retourne le barycentre de wasserstein pour un poids donné
    if poids is None:
        poids = np.ones(A.shape[1]) / A.shape[1]
    else:
        assert(len(poids) == A.shape[1])
    if log:
        log = {'err': []}
    K = np.exp(-M / reg)
    cpt = 0
    err = 1
    UKv = np.dot(K, np.divide(A.T, np.sum(K, axis=0)).T)
    u = (Moyennegeometrique(UKv) / UKv.T).T
    while (err > stopThr and cpt < numItermax):
        cpt = cpt + 1
        UKv = u * np.dot(K, np.divide(A, np.dot(K, u)))
        u = (u.T * Barregeometrique(poids, UKv)).T / UKv
        if cpt % 10 == 1:
            err = np.sum(np.std(UKv, axis=1))
            if log:
                log['err'].append(err)
            if verbose:
                if cpt % 200 == 0:
                    print(
                        '{:5s}|{:12s}'.format('It.', 'Err') + '\n' + '-' * 19)
                print('{:5d}|{:8e}|'.format(cpt, err))
    if log:
        log['niter'] = cpt
        return Barregeometrique(poids, UKv), log
    else:
        return Barregeometrique(poids, UKv)
reg = 1e-3
def diffmax(l1,l2):
```

Interpolation barycentrique

3/3

```python
def diffmax(l1,l2):
    #retourne en valeur absolue la difference maximale de deux listes
    M=0
    for i in range(100):
        if abs(l1[i]-l2[i])>M:
            M=abs(l1[i]-l2[i])
    return(M)
n_alpha = 100
alpha_list = np.linspace(0, 1, n_alpha)
def w(i):
     #retourne le barycentre relatif à i
    B_wass = np.zeros(n_alpha)
    alpha = alpha_list[i]
    poids = np.array([1 - alpha, alpha])
    B_wass = barycentre(A, M, reg, poids)
    return(B_wass)
def proche(l1) :
    #retourne l'indice dans alpha_list de l'alpha de la courbe la plus proche
    m=diffmax(l1,w(0))
    a=0
    for i in range(100):
        if diffmax(l1,w(i))<m:
            m=diffmax(l1,w(i))
            a=i
    return(a)
print(alpha_list[proche(a3)])
plt.plot(x,w(proche(a3)),'y',label="barycentre de wasserstein")
plt.legend()
plt.ylabel("frequence ndvi")
plt.show()
```

1/3

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pylab as pl
from matplotlib.collections import PolyCollection
from scipy.spatial.distance import cdist
n = 100
x = np.arange(n, dtype=np.float64)
def gauss_fonction(x, a, x0, sigma):
    return a*np.exp(-(x-x0)**2/(2*sigma**2))
list1=[0.05075076,21.70256786,7.29220092]
list2=[2.21551895e-02,4.98199892e+01,1.82610047e+01]
a1=gauss_fonction(x,*list1)
a2=gauss_fonction(x,*list2)
A = np.vstack((a1, a2)).T
n_distributions = A.shape[1]
def distance_euclidienne(X, Y, squared=False):
    XX = np.einsum('ij,ij->i', X, X)[:, np.newaxis]
    YY = np.einsum('ij,ij->i', Y, Y)[np.newaxis, :]
    distance = np.dot(X, Y.T)
    distance *= -2
    distance += XX
    distance += YY
    np.maximum(distance, 0, out=distance)
    if X is Y:
        distance.flat[::distance.shape[0] + 1] = 0.0
    return distance if squared else np.sqrt(distance, out=distance)
def dist(x1, x2=None, metric='sqeuclidean'):
    if x2 is None:
        x2 = x1
    if metric == "sqeuclidean":
        return distance_euclidienne(x1, x2, squared=True)
    return cdist(x1, x2, metric=metric)
def dist0(n, method='lin_square'):
    res = 0
    if method == 'lin_square':
        x = np.arange(n, dtype=np.float64).reshape((n, 1))
        res = dist(x, x)
    return res
M =dist0(n)
M /= M.max()
def Barregeometrique(poids, toutesdistribT):
```

```python
    assert(len(poids) == toutesdistribT.shape[1])
    return np.exp(np.dot(np.log(toutesdistribT), poids.T))
def Moyennegeometrique(toutesdistribT):
    return np.exp(np.mean(np.log(toutesdistribT), axis=1))
def barycentre(A, M, reg, poids=None, numItermax=1000,
               stopThr=1e-4, verbose=False, log=False):
    #retourne le barycentre de wasserstein pour un poids donné
    if poids is None:
        poids = np.ones(A.shape[1]) / A.shape[1]
    else:
        assert(len(poids) == A.shape[1])
    if log:
        log = {'err': []}
    K = np.exp(-M / reg)
    cpt = 0
    err = 1
    UKv = np.dot(K, np.divide(A.T, np.sum(K, axis=0)).T)
    u = (Moyennegeometrique(UKv) / UKv.T).T
    while (err > stopThr and cpt < numItermax):
        cpt = cpt + 1
        UKv = u * np.dot(K, np.divide(A, np.dot(K, u)))
        u = (u.T * Barregeometrique(poids, UKv)).T / UKv
        if cpt % 10 == 1:
            err = np.sum(np.std(UKv, axis=1))
            if log:
                log['err'].append(err)
            if verbose:
                if cpt % 200 == 0:
                    print(
                        '{:5s}|{:12s}'.format('It.', 'Err') + '\n' + '-' * 19)
                print('{:5d}|{:8e}|'.format(cpt, err))
    if log:
        log['niter'] = cpt
        return Barregeometrique(poids, UKv), log
    else:
        return Barregeometrique(poids, UKv)
reg = 1e-3
n_alpha = 10
alpha_list = np.linspace(0, 1, n_alpha)
B_l2 = np.zeros((n, n_alpha))
B_wass = np.copy(B_l2)
```

2/3

```python
reg = 1e-3
n_alpha = 10
alpha_list = np.linspace(0, 1, n_alpha)
B_l2 = np.zeros((n, n_alpha))
B_wass = np.copy(B_l2)
for i in range(0, n_alpha):
    alpha = alpha_list[i]
    poids = np.array([1 - alpha, alpha])
    B_l2[:, i] = A.dot(poids)
    B_wass[:, i] =barycentre(A, M, reg, poids)
cmap = pl.cm.get_cmap('viridis')
verts = []
zs = alpha_list
for i, z in enumerate(zs):
    ys = B_wass[:, i]
    verts.append(list(zip(x, ys)))
ax = pl.gcf().gca(projection='3d')
poly = PolyCollection(verts, facecolors=[cmap(a) for a in alpha_list])
poly.set_alpha(0.7)
ax.add_collection3d(poly, zs=zs, zdir='y')
ax.set_xlabel('ndvi')
ax.set_xlim3d(0, 100)
ax.set_ylabel('alpha')
ax.set_ylim3d(0, 1)
ax.set_zlabel('frequence ndvi')
ax.set_zlim3d(0, B_l2.max() * 1.01)
pl.title('Barycentres de Wasserstein')
fig, ax = plt.subplots()
pl.tight_layout()
pl.show()
```

3/3

```python
import pandas as pd
import numpy as np
from math import sqrt
import matplotlib.pylab as pl
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
dataset = pd.read_excel("hitogram_mod13q1_200081_to201897_mod.xlsx",sheet_name="histogram_mod13q1_zaghouan")
type(dataset["Unnamed: 1"].values)
xy= dataset.iloc[3::1000,1:3].values
z1=dataset.iloc[3::1000,8].values
z2=dataset.iloc[3::1000,10].values
z3=dataset.iloc[3::1000,38].values
Z1,Z2,X,Y,Z3=[],[],[],[],[]
for i in range(len(z1)):
    if z1[i][0]=='0' and z1[i][1]=='.'and z2[i][0]=='0' and z2[i][1]=='.':
        X.append(float(xy[i][0]))
        Y.append(float(xy[i][1]))
        Z1.append(float(z1[i]))
        Z2.append(float(z2[i]))
        Z3.append(float(z3[i]))
X = np.array(X).reshape(len(X),1)
Y = np.array(Y).reshape(len(Y),1)
Z1 = np.array(Z1).reshape(len(Z1),1)
Z2 = np.array(Z2).reshape(len(Z2),1)
Z3 = np.array(Z3).reshape(len(Z3),1)
```

1/5

```python
def sinkhorn(a, b, M, reg, method='sinkhorn', numItermax=1000,
             stopThr=1e-9, verbose=False, log=False, **kwargs):
    if method.lower() == 'sinkhorn':
        def sink():
            return sinkhorn_knopp(a, b, M, reg, numItermax=numItermax,
                                  stopThr=stopThr, verbose=verbose, log=log, **kwargs)
    elif method.lower() == 'greenkhorn':
        def sink():
            return greenkhorn(a, b, M, reg, numItermax=numItermax,
                              stopThr=stopThr, verbose=verbose, log=log)
    elif method.lower() == 'sinkhorn_stabilized':
        def sink():
            return sinkhorn_stabilized(a, b, M, reg, numItermax=numItermax,
                                       stopThr=stopThr, verbose=verbose, log=log, **kwargs)
    elif method.lower() == 'sinkhorn_epsilon_scaling':
        def sink():
            return sinkhorn_epsilon_scaling(
                a, b, M, reg, numItermax=numItermax,
                stopThr=stopThr, verbose=verbose, log=log, **kwargs)
    else:
        print('Warning : unknown method using classic Sinkhorn Knopp')

        def sink():
            return sinkhorn_knopp(a, b, M, reg, **kwargs)
    return sink()
Gs = sinkhorn(a, b, M, lambd)
A=np.einsum('ij,ij->i',X,Y)
def plot2D(xs, xt,A, G, thr=1e-8, **kwargs):
    if ('color' not in kwargs) and ('c' not in kwargs):
        #trace les graphes de transport entre les points
        kwargs['color'] = 'k'
    mx = G.max()
    for i in range(len(xs)):
        for j in range(len(xt)):
            if G[i, j] / mx > thr:
                plt.plot([A[i], A[j]], [xs[i],xt[j]],
                         alpha=G[i, j] / mx, **kwargs)
def maxx(xs, xt,A, G, thr=1e-8, **kwargs):
    if ('color' not in kwargs) and ('c' not in kwargs):
        kwargs['color'] = 'k'
```

2/5

Algorithme de la distribution empirique de données

```python
        x=(y3*(y2-y1)+x3*(x2-x1)-(y2-y1)*y1+((((y2-y1)**2)*x1)/(x2-x1)))/((x2-x1)+(((y2-y1)**2)/(x2-x1)))
        y=(y3*(y2-y1)+x3*(x2-x1)-(x2-x1)*x1+((((x2-x1)**2)*y1)/(y2-y1)))/((y2-y1)+(((x2-x1)**2)/(y2-y1)))
        return([x,y])
def dist_drt(x,y,x1,y1,x2,y2):
    if x1==x2:
        return(abs(x1-x))
    else:
        a=(y2-y1)/(x2-x1)
        b=y1-a*x1
        return(abs(a*x+y+b/np.sqrt(a*a+1)))
        #distance d'un point à une droite definie par 2
        #points de coordonnées (x1,y1) et(x2,y2)
def position(A,Z1,Z2,Z3,G, thr=1e-8):
    X1,X2=[],[]
    posy=[]
    posx=[]
    mx = G.max()
    for i in range(n):
        m=300
        for j in range(n):
            for k in range(n):
                l=projt(A[i],Z3[i],A[j],Z1[j],A[k],Z2[k])
                if dist_drt(A[i],Z3[i],A[j],Z1[j],A[k],Z2[k])<m and G[j, k] / mx > thr and \
                abs((dist_pt(l[0],l[1],A[j],Z1[j])+dist_pt(l[0],l[1],A[k],Z2[k]))-dist_pt(A[j],Z1[j],A[k],Z2[k]))<=0.1:
                    #il faut que le projet appartienne à la droite et soit entre les deux points
                        m=dist_drt(A[i],Z3[i],A[j],Z1[j],A[k],Z2[k])
                        jj=j
                        kk=k
        X1.append(jj)
        X2.append(kk)
        posx.append(projt(A[i],Z3[i],A[jj],Z1[jj],A[kk],Z2[kk])[0])
        posy.append(projt(A[i],Z3[i],A[jj],Z1[jj],A[kk],Z2[kk])[1])
    return(posx,posy,X1,X2)
a,b,c,d=position(A,Z1,Z2,Z3,Gs, thr=1e-8)
plt.figure(6)
aa,bb,cc,dd=[],[],[],[]
for i in range(n):
    plt.plot([A[c[i]], A[d[i]]], [Z1[c[i]],Z2[d[i]] ],'b')
    plt.scatter(a[i],b[i])
    aa.append(A[c[i]])
    bb.append(Z1[c[i]])
```

3/5

```python
    mx = G.max()
    l1=[]

    for i in range(len(xs)):
        a=0
        for j in range(len(xt)):
            if G[i, j] / mx > thr and abs(G[i, j] / mx)>a:
                    a=G[i, j] / mx
                    b=j

        l1.append(b)
    return(l1)
plt.figure(3)
plt.imshow(Gs, interpolation='nearest')
plt.title('OT sinkhorn matrice')
plt.figure(4)
plot2D(Z1, Z2,A, Gs, color=[.5, .5, 1])
plt.plot(A,Z1, '+b', label='rdt faible')
plt.plot(A,Z2, 'xr', label='rdt eleve')
plt.legend()
plt.title('matrice sinkhorn des données echantillonnées')
plt.xlabel("X*Y")
plt.ylabel("ndvi")
plt.figure(5)
l=maxx(Z1, Z2,A, Gs, color=[.5, .5, 1])
for i in range(n):
    plt.plot([A[i],A[l[i]]],[Z1[i],Z2[l[i]]])
plt.title('transport plus probable')
plt.xlabel("X*Y")
plt.ylabel("ndvi")
plt.scatter(A,Z1,'b')
plt.scatter(A,Z2,'r')
def dist_pt(x1,y1,x2,y2):
    return(np.sqrt((x2-x1)**2+(y2-y1)**2))
    #distance entre 2 points
def projt(x3,y3,x1,y1,x2,y2):
    #retourne les coordonées du projete
    if x1==x2:
        return([x1,y3])
    else:
```

4/5

5/5

```python
aa,bb,cc,dd=[],[],[],[]
for i in range(n):
    plt.plot([A[c[i]], A[d[i]]], [Z1[c[i]],Z2[d[i]] ],'b')
    plt.scatter(a[i],b[i])
    aa.append(A[c[i]])
    bb.append(Z1[c[i]])
    cc.append(A[d[i]])
    dd.append(Z2[d[i]])
plt.plot(aa,bb,'+b',label='rdt faible')
plt.plot(cc,dd,'xr',label='rdt eleve')
plt.plot(A,Z3,'*g',label='intermediaire')
plt.plot(a,b,'*y',label='projete')
plt.title('position des points intermediaires par rapport aux autres')
plt.xlabel("X*Y")
plt.ylabel("ndvi")
plt.legend()
moy=0
for i in range(n):
    moy+=dist_pt(a[i],b[i],A[c[i]],Z1[c[i]])/n
print(moy)
```

Régression linéaire

```python
l1 = np.array(l1).reshape(3,1)
l2 = np.array(l2).reshape(3,1)
plt.scatter(l,k)
reglin = LinearRegression()
reglin.fit(l1,l2)
reglin = LinearRegression()
reglin.fit(l1,l2)
x = list(sorted(l1))
y = reglin.predict(x)
plt.plot(x, y,label='regression')
plt.scatter(l,k)
plt.title('regression lineaire')
plt.xlabel("moyennes alpha")
plt.ylabel("rendement")
plt.show()
plt.figure(3)
def droite(l):
    kk=[0]
    a=0
    for i in range (1,len(l)):

        kk.append((l[i]*float(reglin.coef_)+float(reglin.intercept_)-k[i])/k[i])
        a+=(l[i]*float(reglin.coef_)-k[i])/k[i]
    print(abs(a)/len(kk))
    return(kk)
z=droite(l)
plt.scatter(l,z)
plt.title('erreur relative')
plt.xlabel("moyenne alpha")
plt.ylabel("erreur relative")
plt.show()
```