Symbolic and Evolutionary Artificial Intelligence
Master Degree in Artificial Intelligence and Deep Learning
Year 2020/21

Optimal Transport

Luca Barsellotti
Marco Bongiovanni

1 Introduction

In recent years, Optimal Transport (OT) Theory has become attractive in Machine Learning and Computer Vision fields thanks to the need of approximate solvers designed for large scale problems and to its properties when used for comparing two distributions. Typical problems in which OT can be applied are Color Transfer [1], which consists in transferring the color style of an image into another one, Image Editing [2], which consists in editing an image by replacing a part of it using a patch of another image, Shape Reconstruction [3], which consists in capturing the shape and the appearance of real objects, training phase in Wasserstein Generative Adversarial Network (GAN) [4], Brain Decoding [5], which consists in averaging the brain imaging data, and Image Super-Resolution [6], namely the process of reconstructing a high-resolution image from a corresponding low-resolution images.

How can the OT Theory be applied to all these kinds of problems? OT can be defined as the problem of comparing two different probability distributions, consider all the possible ways to morph, transport or reshape the first into the second, associate a cost to every such transport and then take the least costly transport. The solution gives also information about the geometrical structure of the probability distributions. The OT problems can be metaphorically described as a worker with a shovel in hand that has to move a large pile of sand to erect a target pile with a specific shape using the minimum total effort. All the problems taken in account before involve moving distribution and for that reason the OT Theory becomes suitable to solve them.

2 Optimal Transport with Linear Programming

2.1 Optimal Transport of Discrete Distribution

Let imagine that there are some sources with available materials which should be transported to some destinations with required materials. We consider two **discrete distributions**, one representing the distribution of the sources and one representing the distribution of the destinations:

$$\forall k = 0, 1, \quad \mu_k = \sum_{i=0}^{n_k} p_{k,i}\delta_{x_{k,i}} \tag{1}$$

where:

- $k = 0$ indicates the sources;
- $k = 1$ indicates the destinations;
- n_0 indicates the number of source points;
- n_1 indicates the number of destination points;
- $x_{0,i} \in R^d$ indicates the **location** of the i-th source;

- $x_{1,i} \in R^d$ indicates the **location** of the i-th destination;
- $X_0 = (x_{0,i})_{i=1}^{n_0}$ indicates the **point cloud** of the sources;
- $X_1 = (x_{1,i})_{i=1}^{n_1}$ indicates the **point cloud** of the destinations;
- μ_0 indicates the **discrete distribution** of the sources;
- μ_1 indicates the **discrete distribution** of the destinations;
- $p_{0,i}$ indicates the available material in the i-th source (called **density weight**);
- $p_{1,i}$ indicates the required material in the i-th destination (called **density weight**);
- $\delta_{x_{0,i}}$ indicates the **Dirac** in the source location $x_{0,i}$;
- $\delta_{x_{1,i}}$ indicates the **Dirac** in the destination location $x_{1,i}$;

We define the **set of couplings** between μ_0, μ_1 as:

$$P = \{(\gamma_{i,j})_{i,j} \in (R^+)^{n_0 \times n_1}, \forall i, \sum_j \gamma_{i,j} = p_{0,i}, \forall j, \sum_i \gamma_{i,j} = p_{1,j}\} \tag{2}$$

where

- $\gamma_{i,j}$ is the **coupling**, namely the transported material, between the i-th source and the j-th destination. Each $\gamma_{i,j}$ is a positive real number and there is a coupling for each possible combination between a source and a destination;
- $\sum_j \gamma_{i,j} = p_{0,i}$ is the **constraint** which indicates that the material in output from the i-th source should be equal to the available material in that source;
- $\sum_i \gamma_{i,j} = p_{1,j}$ is the **constraint** which indicates that the material in input in the j-th destination should be equal to the required material in that destination;

Assume a toy example in which there are three sources ($n_0 = 3$) and four destinations ($n_1 = 4$) in R^2, where the point cloud of the sources is a Gaussian and the point cloud of the destinations is a Gaussian Mixture. The available and requested materials can be randomly generated and normalized, so that the sum for both will be 1: in this way the required material is equal to the available material. In MATLAB:

```
n0 = 3;
n1 = 4;

gauss = @(q,a,c)a*randn(2,q)+repmat(c(:), [1 q]);
```

```
X0 = randn(2,n0)*.3;
X1 = [gauss(n1/2,.5, [0 1.6]) gauss(n1/4,.3, [-1 -1]) ...
     gauss(n1/4,.3, [1 -1])];

normalize = @(a)a/sum(a(:));
p0 = normalize(rand(n0,1));
p1 = normalize(rand(n1,1));
```

The three locations of the sources in X_0 will be:

0.05856478057150	0.02076654631178	-0.4996929353644
0.2666586461648	0.7460463960892	-0.1247783834556

The density weights of the sources are:

0.5632732583192
0.1002242463506
0.3365024953302

The four locations of the destinations in X_1 will be:

-0.04211230323545	0.7280736002377	-1.034463175576	1.225448457091
1.644625973754	1.709726759319	-0.9794192591551	-1.206828117377

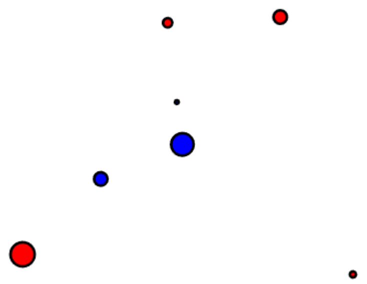
The density weights of the destinations are:

0.1762404573564
0.2532707705009
0.4541031797063
0.1163855924366

To plot them:

```
myplot = @(x,y,ms,col)plot(x,y, 'o', 'MarkerSize', ms, ...
     'MarkerEdgeColor', 'k', 'MarkerFaceColor', col, 'LineWidth', 2);

clf; hold on;
for i=1:length(p0)
    myplot(X0(1,i), X0(2,i), p0(i)*length(p0)*10, 'b');
end
for i=1:length(p1)
    myplot(X1(1,i), X1(2,i), p1(i)*length(p1)*10, 'r');
end
axis([min(X1(1,:)) max(X1(1,:)) min(X1(2,:)) max(X1(2,:))]); ...
     axis off;
```

The resulting plot is:



In the **Kantorovitch formulation** of the optimal transport the optimum is given by

$$\gamma^* \in \arg\min_{\gamma \in P} \sum_{i,j} \gamma_{i,j} C_{i,j} \tag{3}$$

where $C_{i,j} >= 0$ is the **cost** of moving some material (called **mass**) from $x_{0,i}$ to $x_{1,j}$.

The **optimal coupling** γ^* can be proved to be a sparse matrix with less than $n_0 + n_1 - 1$. An entry $\gamma^*_{i,j} \neq 0$ should be understood as a link between $x_{0,i}$ and $x_{1,j}$ where an amount of mass equal to $\gamma^*_{i,j}$ is transferred.

The matrix of the costs C can be implemented as we wish, for example

$$C_{i,j} = \left\| x_{0,i} - x_{1,j} \right\|^2 \tag{4}$$

Then, the L^2 **Wasserstein distance** is defined as:

$$W_2(\mu_0, \mu_1)^2 = \sum_{i,j} \gamma^*_{i,j} C_{i,j} \tag{5}$$

The **coupling constraints**:

$$\forall i, \sum_j \gamma_{i,j} = p_{0,i}, \forall j, \sum_i \gamma_{i,j} = p_{1,j} \tag{6}$$

can be expressed in matrix form as:

$$\Sigma(n_0, n_1)\gamma = [p_0; p_1] \tag{7}$$

where $\Sigma(n_0, n_1) \in R^{(n_0+n_1)\times(n_0 n_1)}$.

In MATLAB it can be seen as:

```
C = repmat( sum(X0.^2)', [1 n1] ) + ...
    repmat( sum(X1.^2), [n0 1] ) - 2*X0'*X1;

%Define a function handle that transform a x matrix in a column ...
    vector
flat = @(x)x(:);
%Define two function handles that generate an efficient sparse ...
    matrix
```

```
Cols = @(n0,n1)sparse( flat(repmat(1:n1, [n0 1])), ...
                flat(reshape(1:n0*n1,n0,n1) ), ...
                ones(n0*n1,1) );
Rows = @(n0,n1)sparse( flat(repmat(1:n0, [n1 1])), ...
                flat(reshape(1:n0*n1,n0,n1)' ), ...
                ones(n0*n1,1) );
% Define a function handle that expresses the coupling ...
    constraint matrix
Sigma = @(n0,n1)[Rows(n0,n1);Cols(n0,n1)];
```

In this case, the matrix C is a 3×4 matrix, containing:

1.908929831107	2.530687638813	2.747420259046	3.532780756861
0.8113990071162	1.428963111008	4.090741493382	5.264977371504
3.340171814211	4.872819784811	1.016390236232	4.146944450642

The shown function handles will be called on the optimal transport coupling γ^*, which can be obtained through the simplex algorithm. In MATLAB:

```
%Maximum number of iterations
maxit = 1e4;
%Tolerance
tol = 1e-9;
otransp = @(C,p0,p1)reshape( perform_linprog( ...
        Sigma(length(p0),length(p1)), ...
        [p0(:);p1(:)], C(:), 0, maxit, tol), [length(p0) ...
            length(p1)] );
gamma = otransp(C,p0,p1);
```

The optimal coupling γ^* is a 3×4 sparse matrix given by:

0.17624045735644	0.1530465241503	0.1176006843759	0.1163855924366
0	0.1002242463506	0	0
0	0	0.3365024953302	0

It is possible to check if the number of non-zero elements in the matrix is lower or equal to $n_0 + n_1 - 1$ (equal to 6):

```
fprintf('Number of non-zero: %d (n0+n1-1=%d)\n', ...
    full(sum(gamma(:)≠0)), n0+n1-1);
```

The number of non-zero elements is equal to 6, so equal to $n_0 + n_1 - 1$ and it is correct. It is possible also to check the constraints deviation:

```
fprintf('Constraints deviation (should be 0): %.2e, %.2e.\n', ...
    norm(sum(gamma,2)-p0(:)),  norm(sum(gamma,1)'-p1(:)));
```

To analyze the moving of the mass from the sources to the destinations it is useful to plot the evolution of μ_t for a varying value of $t \in [0, 1]$. What is μ_t? For any $t \in [0, 1]$, it is possible to define a distribution μ_t such that $t \mapsto \mu_t$

defines a geodesic for Wasserstein metric. A geodesic is a curve representing in some sense the shortest path between two points in a surface. Since the W_2 distance is a geodesic distance, this geodesic path solves the following variational problem:

$$\mu_t = \arg\min_{\mu} (1-t)W_2(\mu_0, \mu)^2 + tW_2(\mu_1, \mu)^2 \tag{8}$$

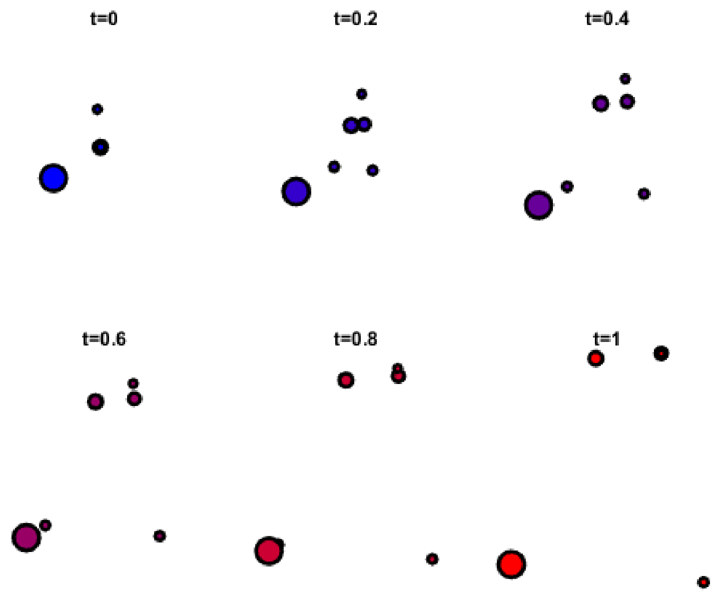
In practice, the **interpolated distribution** μ_t can be understood as an intermediate distribution between the source and the destination distributions. Let $\mu_k = \delta_{x_k}$, then $\mu_t = \delta_{x_k}$ where $x_t = (1-t)x_0 + tx_1$.

Once the optimal coupling γ^* has been computed, the interpolated distribution is obtained as

$$\mu_t = \sum_{i,j} \gamma^*_{i,j} \delta_{(1-t)x_{0,i}+tx_{1,j}} \tag{9}$$

Find the i, j with non-zero $\gamma^*_{i,j}$. In MATLAB:

```matlab
[I,J,gammaij] = find(gamma);

clf;
tlist = linspace(0,1,6);
for i=1:length(tlist)
    t=tlist(i);
    Xt = (1-t)*X0(:,I) + t*X1(:,J);
    subplot(2,3,i);
    hold on;
    for i=1:length(gammaij)
        myplot(Xt(1,i), Xt(2,i), gammaij(i)*length(gammaij)*6, ...
             [t 0 1-t]);
    end
    title(['t=' num2str(t,2)]);
    axis([min(X1(1,:)) max(X1(1,:)) min(X1(2,:)) max(X1(2,:))]); ...
        axis off;
end
```



2.2 Optimal Assignment

Assume a special case in which the number of the source points is equal to the one of the destination points (namely $n_0 = n_1 = n$) and moreover the weights of the sources and the destinations are all equal among them and constants (namely $p_{0,i} = 1/n, p_{1,i} = 1/n$). Let assume a toy example in MATLAB with $n = 4$:

```
n0 = 4;
n1 = n0;

X0 = randn(2,n0)*.3;
X1 = [gauss(n1/2,.5, [0 1.6]) gauss(n1/4,.3, [-1 -1]) ...
     gauss(n1/4,.3, [1 -1])];

p0 = ones(n0,1)/n0;
p1 = ones(n1,1)/n1;
```

So, x_0 is equal to:

-0.1422404202120	-0.6219079297018	-0.09457187428494	-0.2639644186302
0.2962145979051	0.3732078359950	0.4197881996802	-0.2767417366056

x_1 is equal to:

1.157201449266	-0.1548838173838	-0.6689947798329	0.3208020332562
1.894006602047	1.083517926325	-0.8973673207685	-1.270985745005

p_0 is equal to:

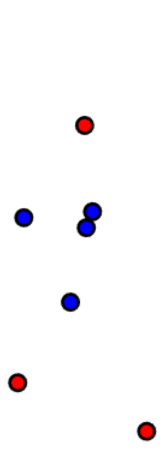
0.25
0.25
0.25
0.25

p_1 is equal to:

0.25
0.25
0.25
0.25

To display them:

```
clf; hold on;
myplot(X0(1,:), X0(2,:), 10, 'b');
myplot(X1(1,:), X1(2,:), 10, 'r');
axis equal; axis off;
```



In this case, the optimal transport coupling is simply a permutation matrix. This means that there exists an optimal permutation $\sigma^* \in \sum_n$ such that

$$\gamma^*_{i,j} = \begin{cases} 1 & \text{if} \quad j = \sigma^*(i), \\ 0 & \text{otherwise} \end{cases}$$

where $\sum_n$ is the set of permutations (bijections) of $\{1, ..., n\}$.

In this way the permutation solves the **optimal assigned problem**, defined as:

$$\sigma^* \in \arg\min_{\sigma \in \sum_n} \sum_i C_{i,\sigma(j)} \tag{10}$$

To compute the weight matrix $(C_{i,j})_{i,j}$ in MATLAB:

```
C = repmat( sum(X0.^2)', [1 n1] ) + ...
    repmat( sum(X1.^2), [n0 1] ) - 2*X0'*X1;
```

The resulting weight matrix is:

4.241488460651	0.6200063864332	1.702107951964	2.670525228533
5.478059069154	0.7226519459110	1.616578400440	3.592074606063
3.740256351429	0.4441746805123	2.064860339467	3.031252015094
6.731860773835	1.862204928165	0.5492257092135	1.330472951491

To solve the optimal transport:

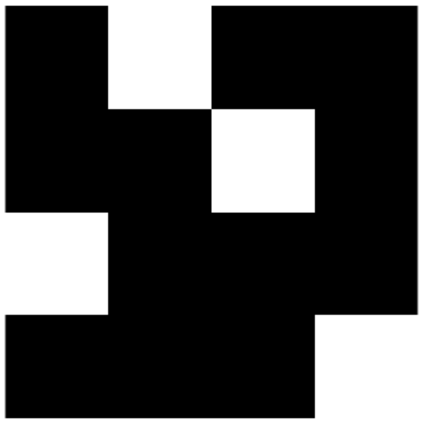
```
gamma = otransp(C,p0,p1);
```

The solutions is:

0	0.25	0	0
0	0	0.25	0
0.25	0	0	0
0	0	0	0.25

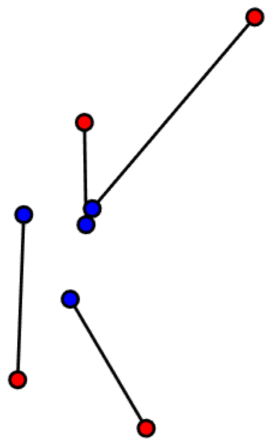
To show a binary permutation matrix of the solution γ:

```
clf;
imageplot(gamma);
```



To display the optimal assignment:

```
clf; hold on;
[I,J,¬] = find(gamma);
for k=1:length(I)
    h = plot( [X0(1,I(k)) X1(1,J(k))], [X0(2,I(k)) X1(2,J(k))], ...
        'k' );
    set(h, 'LineWidth', 2);
end
myplot(X0(1,:), X0(2,:), 10, 'b');
myplot(X1(1,:), X1(2,:), 10, 'r');
axis equal; axis off;
```



3 Optimal Transport in 1-D for Grayscale Image Equalization

3.1 Optimal Transport and Assignement

Let consider data $f \in R^{N\times d}$, which can correspond, for instance, to an image of N pixels with $d = 1$ for **grayscale image** and $d = 3$ for **color image**. We denote $f = (f_i)_{i=1}^{N}$ with $f_i \in R^d$ the elements of the data. To understand how this method works, let consider the case $d = 1$. To load an image $f \in R^N$ of $N = n \times n$ pixels:

```
n = 256;
f = rescale( load_image('lena', n) ); %matrix 256x256
```

To display it:

```
clf;
imageplot(f);
```

Let load another image:

```
g = rescale( mean(load_image('fingerprint', n),3) ); %matrix 256x256
clf;
imageplot(g);
```

The **discrete distribution** in R^d associated to this data f is the sum of Diracs:

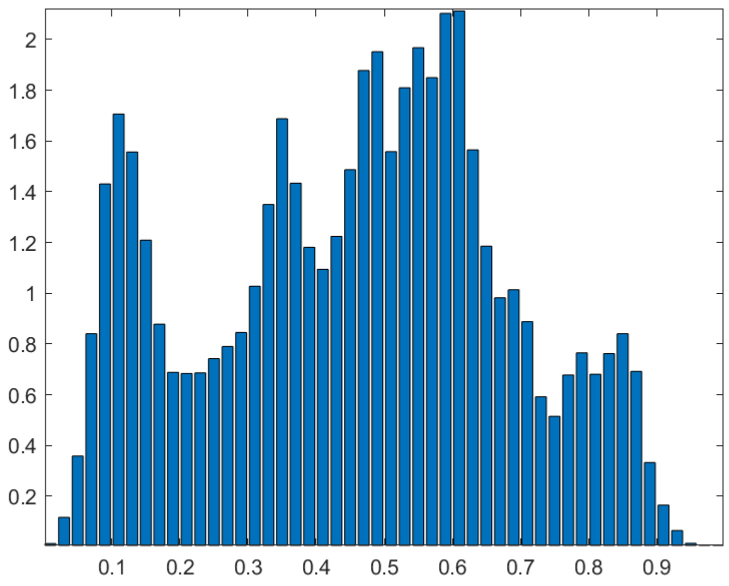
$$\mu_f = \frac{1}{N}\sum_{i=1}^{N}\delta_{f_i} \tag{11}$$

A convenient way to visualize the distribution μ_f is by computing the histogram $h \in R^Q$ composed using Q bins $[u_k, u_{k+1})$. The histogram is computed as $\forall k = 1, ..., Q, \quad h(p) = |\{i, f_i \in [u_k, u_{k+1})\}|$. In MATLAB:

```
Q = 50;
[h,t] = hist(f(:), Q);
```

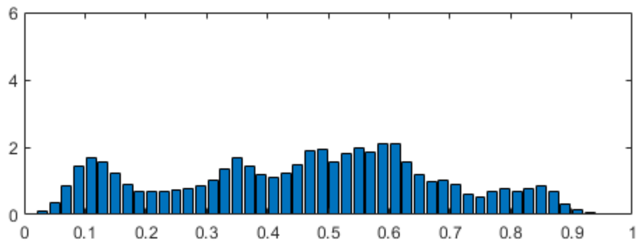
To make this curve an approximation of a continuous distribution, we normalize h by $\frac{Q}{N}$. To display the normalized histogram in MATLAB:

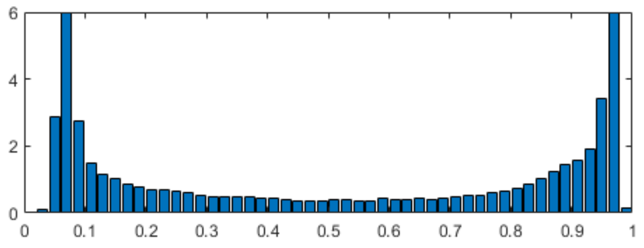
```
clf;
bar(t,h*Q/n^2); axis('tight');
```



Now, let show the comparison between the distributions of the two images:

```
Q = 50;
[h0,t] = hist(f(:), Q);
[h1,t] = hist(g(:), Q);
clf;
subplot(2,1,1);
bar(t,h0*Q/n^2); axis([0 1 0 6]);
subplot(2,1,2);
bar(t,h1*Q/n^2); axis([0 1 0 6]);
```





3.2 1-D Optimal Assignment

Notice that the histograms are computed just basing on the pixels intensity without considering any information regarding their position in the images.

An **optimal assignment** between two such vectors $f, g \in R^{N\times d}$ is a **permutation** $\sigma \in \sum_N$ which minimizes

$$\sigma^* \in \arg \min_{\sigma\in\sum_N} \sum_{i=1}^{N} C(f_i, g_{\sigma(i)}) \tag{12}$$

where $C(u, v) \in R$ is the **cost function**.

Let consider the L^p costs:

$$\forall(u, v) \in R^d \times R^d, \qquad C(u, v) = \|u - v\|^p \tag{13}$$

where $\|.\|$ is the Euclidean norm and $p \geq 1$.

The optimal assignment defines the L^p **Wasserstein distance** between the associated point cloud distributions:

$$W_p(\mu_f, \mu_g)^p = \sum_{i=1}^{N} \|f_i - g_{\sigma(i)}\|^p = \|f - g \circ \sigma\|_p^p \tag{14}$$

where $g \circ \sigma = (g_{\sigma(i)})_i$ is the re-ordered points cloud.

For the case $d = 1$, it is possible to compute explicitly an optimal assignment $\sigma^* \in \sum_N$ for any cost $C(u, v) = \varphi(|u - v|)$ where $\varphi : R \to R$ is a convex function. Thus, this is the case for the L^p optimal transport. This is obtained by computing two permutations $\sigma_f, \sigma_g \in \sum_N$ which order the values of the data:

$$f_{\sigma_{f(1)}} \leq f_{\sigma_{f(2)}} \leq ... \leq f_{\sigma_{f(N)}} \tag{15}$$

$$g_{\sigma_{g(1)}} \leq g_{\sigma_{g(2)}} \leq ... \leq g_{\sigma_{g(N)}} \tag{16}$$

We can interpret the two σ as the list of the indexes of the ordered pixel intensities from the two images f and g.

In MATLAB it is possible to compute σ_f, σ_g in $O(N \log(N))$ operations using a fast sorting algorithm (for example QuickSort):

```
[¬,sigmaf] = sort(f(:)); %column vector with 65536 elements
[¬,sigmag] = sort(g(:)); %column vector with 65536 elements
```

An optimal assignment is then obtained by assigning, for each k, the index $i = \sigma_f(k)$ to the index $\sigma_g(k)$, namely:

$$\sigma^* = \sigma_g \circ \sigma_f^{-1} \tag{17}$$

where σ_f^{-1} is the **inverse permutation**, which satisfies

$$\sigma_f^{-1} \circ \sigma_f = Id \tag{18}$$

where Id is the identity permutation (namely $Id = [1, 2, ..., N]$).

An inverse permutation is a permutation where the i-th element represents the rank of the i-th element of the original vector (image) in the ordered vector.

Example:

$$f = [4, 6, 8, 10, 2] \qquad (original\ vector) \tag{19}$$

$$\sigma_f = [5, 1, 2, 3, 4] \qquad (permutation\ to\ sort\ f) \tag{20}$$

$$f_{\sigma_f} = [2, 4, 6, 8, 10] \qquad (ordered\ vector) \tag{21}$$

$$\sigma_f^{-1} = [2, 3, 4, 5, 1] \qquad (inverse\ permutation) \tag{22}$$

Note that this optimal assignment σ^* is not unique when there are two pixels in f or g having the same value. To compute the inverse permutation σ_f^{-1}:

```
sigmafi = []; %row vector with 65536 elements
sigmafi(sigmaf) = 1:n^2;
```

To compute the optimal permutation σ^*:

```
sigma = sigmag(sigmafi); %column vector with 65536 elements
```

The optimal assignment is used to compute the **projection** on the set H_g of images m having the pixel distribution μ_g

$$H_g = \{m \in R^N, \mu_m = \mu_g\} \tag{23}$$

Indeed, for any $p > 1$, the L^p projector on this set is given by

$$\pi_g(f) = \arg\min_{m \in H_g} ||f - m||_p \tag{24}$$

is simply obtained by re-ordering the pixels of g using an optimal assignment $\sigma^* \in \sum_N$ between f and g, namely

$$\pi_g(f) = g \circ \sigma^*. \tag{25}$$

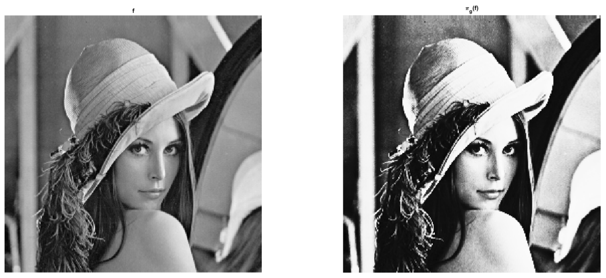
This projection $\pi_g(f)$ is called the **histogram equalization** of f using the histogram of g.

To compute the projection in MATLAB:

```
f1 = reshape(g(sigma), [n n]);
imageplot(f1)
```

To compare the original image with the equalized one:

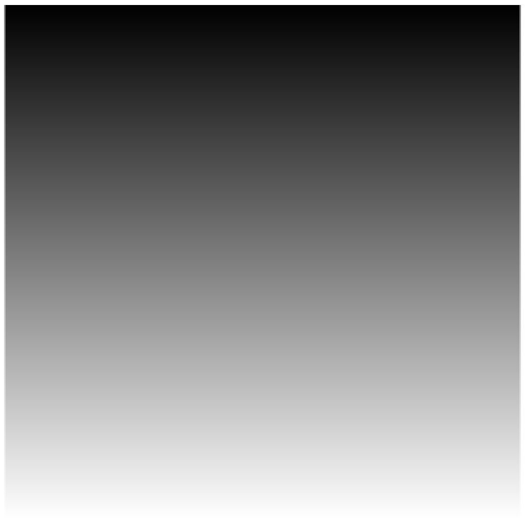
```
imageplot(f, 'f', 1,2,1);
imageplot(f1, '\pi_g(f)',  1,2,2);
```



The classical histogram equalization is done using a flat histogram in place of the histogram g. So, an image with flat histogram is artificially created in MATLAB:

```
for i=1:n
    for j=1:n
        g(i,j)=i/(n);
    end
end
imageplot(g);
```

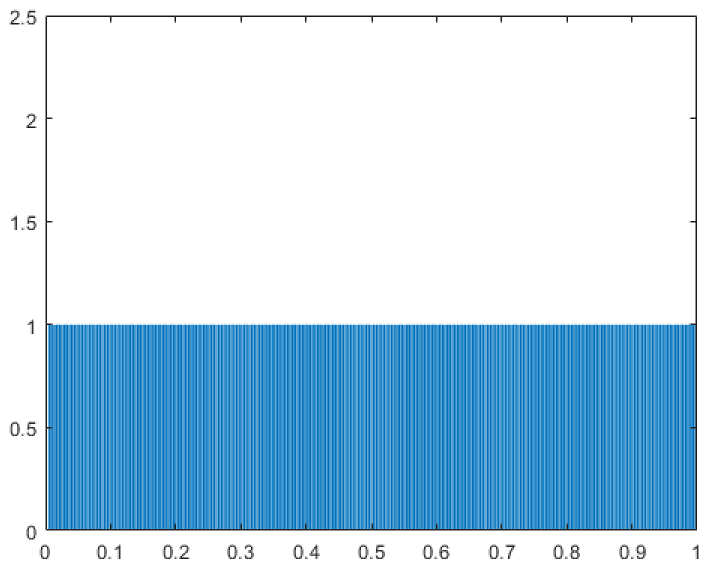
This will result in the following image:



To display its histogram:

```
Q = 256;
[h,t] = hist(g(:), Q);
clf;
bar(t,h*Q/n^2); axis('tight');
```

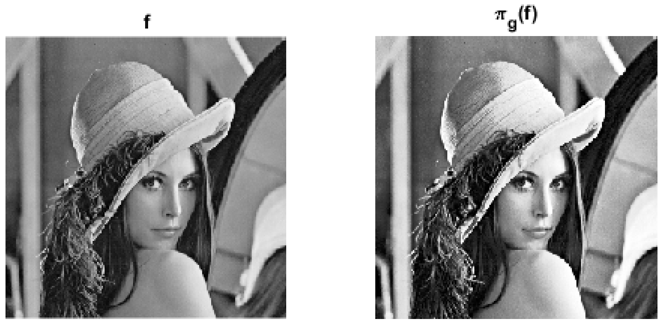
The image will have the following flat histogram:



To compute the histogram equalization in MATLAB:

```
[¬,sigmaf] = sort(f(:));
[¬,sigmag] = sort(g(:));
sigmafi = [];
sigmafi(sigmaf) = 1:n^2;
sigma = sigmag(sigmafi);
f1 = reshape(g(sigma), [n n]);
clf;
imageplot(f, 'f', 1,2,1);
imageplot(f1, '\pi_g(f)',  1,2,2);
```

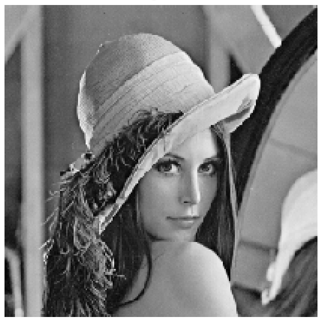
It will result in:



Let now compare this result with the histogram equalization done using the "histeq" function available in the "**Image Processing Toolbox**" of MATLAB.

```
J = histeq(f);
clf;
subplot(3,1,1);
imageplot(f, "Original:");
subplot(3,1,2);
imageplot(f1, "Optimal Tranport Equalization:");
subplot(3,1,3);
imageplot(J, "Image Processing Toolbox:");
```

Original:



Optimal Tranport Equalization:

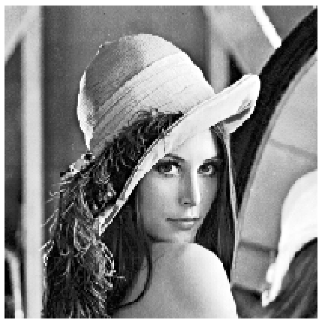
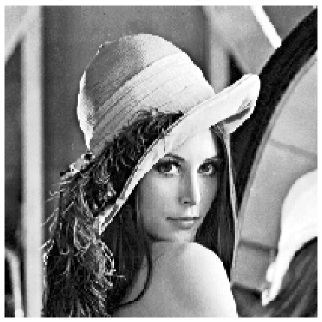


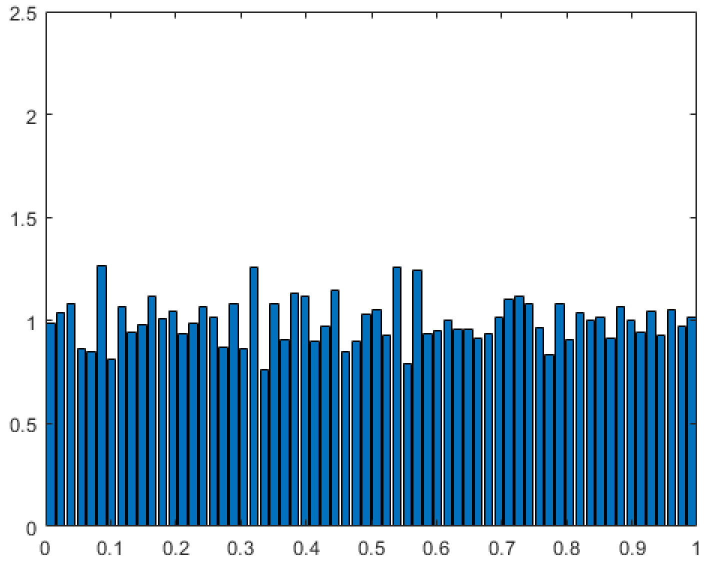
Image Processing Toolbox:



The histogram equalization in the "Image Processing Toolbox" is done choosing the greyscale transformation T that minimizes

$$|c_1(T(k)) - c_0(k)| \tag{26}$$

where c_0 is the cumulative histogram of the input image T and c_1 is the cumulative sum of the flat histogram for all intensities k. The histogram associated to the resulting image is not a perfectly flat histogram, as it happens using the Optimal Transport approach, but it tends to be flat.



3.3 Histogram Interpolation

Let introduce the **linearly interpolated image**, a sort of middle image between the original one and the equalized one:

$$\forall t \in [0,1], \qquad f_t = (1-t)f + tg \circ \sigma^*. \tag{27}$$

It is possible to show that the distribution μ_{f_t} is the **geodesic interpolation** in the L^2-Wasserstein space between the two distributions μ_f (obtained for $t=0$) and μ_g (obtained for $t=1$).

It is also possible to notice that the distribution μ_{f_t} is the **barycenter** between the two distributions since it has the following variational characterization:

$$\mu_{f_t} = \arg\min_{\mu} (1-t)W_2(\mu_f, \mu)^2 + tW_2(\mu_g, \mu)^2. \tag{28}$$

To define the **interpolation operator** in MATLAB:

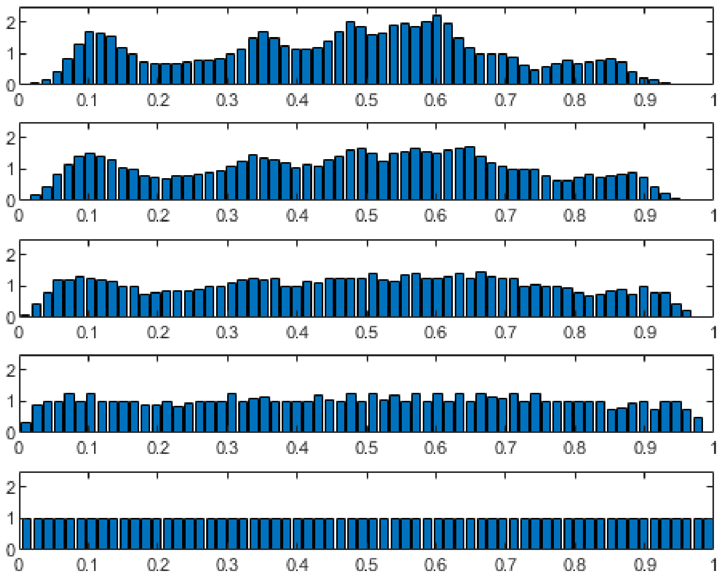
```
ft = @(t)reshape( t*f1 + (1-t)*f, [n n]);
```

The **midway equalization** is obtained for $t = 1/2$:

```
clf;
imageplot(ft(1/2));
```

To display the **progression of the interpolation** of the histograms:

```
p = 64;
tlist = linspace(0,1,5);
for i=1:length(tlist)
    a = ft(tlist(i));
    subplot(length(tlist), 1, i);
    [h,t] = hist(a(:), p);
    bar(t,h*p/n^2);
    axis([0 1 0 2.5]);
end
```



4 Entropic Regularization for Optimal Transport

This section describes the general methodology of regularizing the optimal transport (OT) linear program using entropy. This allows to derive fast computation algorithm based on iterative projections according to a **Kulback-Leiber(KL) divergence**.

Let assume two point clouds that do not necessarily have the same size and that are defined as:

$$x = (x_i)_{i=1}^{N_1}, y = (y_i)_{i=1}^{N_2} \tag{29}$$

where $x_i, y_i \in R^2$. In MATLAB, first the number of points in each cloud must be defined:

```
N = [5, 4];
```

Each point has a dimension equal to 2:

```
d = 2;
```

The point cloud x is defined as a point cloud with N_1 points inside a square.

```
x = rand(2,N(1))-.5;
```

The point cloud y is defined as a point cloud with N_2 points inside a ring.

```
theta = 2*pi*rand(1,N(2));
r = .8 + .2*rand(1,N(2));
y = [cos(theta).*r; sin(theta).*r];
```

x:

0.2922	0.1557	0.3491	0.1787	0.2431
0.4594	-0.4642	0.4339	0.2577	-0.1077

y:

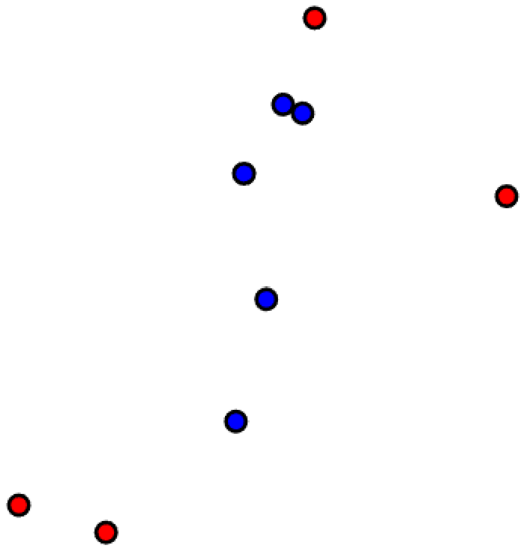
-0.4786	0.3845	-0.2234	0.9454
-0.7089	0.7120	-0.7883	0.1916

It is useful to define a shortcut to display the point clouds:

```
plotp = @(x,col)plot(x(1,:)', x(2,:)', 'o', 'MarkerSize', 10, ...
    'MarkerEdgeColor', 'k', 'MarkerFaceColor', col, 'LineWidth', 2);
```

So, call the shortcut to display the two defined point clouds:

```
clf; hold on;
plotp(x, 'b');
plotp(y, 'r');
axis('off'); axis('equal');
```



Two histograms $p \in \Sigma_{N_1}$, $q \in \Sigma_{N_2}$ has been considered as input for this method, where the simplexes in R_1^N, R_2^N are denoted as:

$$\Sigma_{N_1} \equiv \{p \in (R^+)^{N_1}, \sum_i p_i = 1\} \tag{30}$$

$$\Sigma_{N_2} \equiv \{q \in (R^+)^{N_2}, \sum_i q_i = 1\} \tag{31}$$

To show how this method works, let assume a toy example in which the two input measures are uniform measures (namely constant histograms). To define them in MATLAB:

```
p = ones(N(1),1)/N(1);
q = ones(N(2),1)/N(2);
```

p:

0.2
0.2
0.2
0.2
0.2

q:

0.25
0.25
0.25
0.25

Let assume the following **discrete regularized transport**:

$$W_\gamma(p,q) \equiv \min_{\pi \in \Pi(p,q)} < C, \pi > -\gamma E(\pi) \tag{32}$$

where the **polytope of coupling** is defined as

$$\Pi(p,q) \equiv \{\pi \in (R^+)^{N_1 \times N_2}, \pi I_1 = p, \pi^T I_2 = q\} \tag{33}$$

where $I_1 = (1, ..., 1)^T \in R_1^N, I_2 = (1, ..., 1)^T \in R_2^N$, and for $\pi \in (R^+)^{N_1 \times N_2}$ it is possible to define its **entropy** as:

$$E(\pi) \equiv -\sum_{i,j} \pi_{i,j}(\log(\pi_{i,j}) - 1). \tag{34}$$

The idea of the entropic regularization of optimal transport is to use $-E(\pi)$ as a regularization function to obtain approximate solutions to the original transport problem. Since the objective is an γ-strongly convex function, $W_\gamma(p,q)$ has a unique optimal solution.

When the **regularization strength** $\gamma = 0$ the problem can be reported to a classical (discrete) optimal transport. For the toy example, the following γ has been considered:

```
gamma = .01;
```

The cost matrix $C \in (R^+)^{N_1 \times N_2}$ defines the ground cost, namely $C_{i,j} = ||x_i - y_j||^2$ is the cost of moving mass from a bin indexed by i to a bin indexed by j. In MATLAB:

```
x2 = sum(x.^2,1); y2 = sum(y.^2,1);
C = repmat(y2,N(1),1)+repmat(x2.',1,N(2))-2*x.'*y;
```

C:

1.95941	0.0723	1.8230	0.4984
0.46235	1.4360	0.2488	1.0539
1.99147	0.0785	1.8220	0.4143
1.3666	0.24877	1.25602	0.5922
0.8823	0.6920	0.6809	0.5829

The regularized transportation problem can be re-written as a projection of ξ according to the **Kullbeck-Leibner divergence**:

$$W_\gamma(p,q) = \gamma \min_{\pi \in \Pi(p,q)} KL(\pi|\xi) \tag{35}$$

where ξ, defined as the **Gibbs kernel**, is:

$$\xi_{i,j} = e^{-\frac{C_{i,j}}{\gamma}} \tag{36}$$

In MATLAB:

```
xi = exp(-C/gamma);
```

ξ:

8.004e-86	0.0007	6.6897e-80	2.2476e-22
8.350e-21	4.289e-63	1.5647e-11	1.6917e-46
3.2421e-87	0.0003	7.4274e-80	1.0133e-18
4.4597e-60	1.5760e-11	2.8073e-55	1.9040e-26
4.7775e-39	8.7878e-31	2.6831e-30	4.8283e-26

The Kullback-Leibner divergence between $\pi, \xi \in (R^+)^{N_1 \times N_2}$ is defined as:

$$KL(\pi|\xi) = \sum_{i,j} \pi_{i,j}(\log(\frac{\pi_{i,j}}{\xi_i, j}) - 1). \tag{37}$$

The Kullback-Leibner divergence is a non-symmetric measure that represents the difference between two distributions, in this case π and ξ.

Given a convex set $\Pi \subset R^{N_i}$ the **projection according to the Kullback-Leibner divergence** is defined as:

$$P_\Pi^{KL}(\xi) = \arg\min_{\pi \in \Pi} KL(\pi|\xi) \tag{38}$$

4.1 Sinkhorn's Algorithm

It is possible to move the regularized optimal transport problem in the **Sinkhorn's Algorithm** framework by introducing:

$$\Pi_p^1 \equiv \{\pi \in (R^+)^{N_1 \times N_2}, \pi I_1 = p\} \tag{39}$$

$$\Pi_q^2 \equiv \{\pi \in (R^+)^{N_1 \times N_2}, \pi^T I_2 = q\} \tag{40}$$

rows and columns constraints, so that $\Pi(p,q) = \Pi_p^1 \cap \Pi_q^2$. It is possible to use the **Bregman iterative projections**:

$$\pi_{l+1} = P_{\Pi_p^1}^{KL}(\pi_l) \tag{41}$$

$$\pi_{l+2} = P_{\Pi_q^2}^{KL}(\pi_{l+1}) \tag{42}$$

Since the sets Π_p^1 and Π_q^2 are affine, these iterations are known to converge to the solution ($\pi_l \to P_{\Pi}^{KL}(\overline{\pi})$).

It is possible to write π_l using the **diagonal scaling of the kernel** ξ:

$$\pi_{2l} = diag(a_l)\xi diag(b_l) = P_{\Pi_q^2}^{KL}(\pi_{l-1}) \tag{43}$$

where l indicates the actual iteration of the Sinkhorn's algorithm. Each step of the algorithm contains two iterations and for that reason in the latter equation π_{2l} is used. The middle iteration is given by:

$$\pi_{2l+1} = diag(a_{l+1})\xi diag(b_l) = P_{\Pi_p^1}^{KL}(\pi_l) \tag{44}$$

But what are a and b during the iterations?

$$a_{l+1} = \frac{p}{\xi b_l} \tag{45}$$

$$b_{l+1} = \frac{q}{\xi^T a_{l+1}} \tag{46}$$

initialized with a vector $b_0 = 1_m$. So, at the end of the algorithm

$$\overline{\pi} = diag(a_e)\xi diag(b_e) \tag{47}$$

where e indicates the last iteration.

So, summing up, the algorithm is:

Algorithm 1: Sinkhorn's Algorithm

```
Initialize $b_0 = I_{N_2}$;
Set $e$ (max number of iterations);
for $l$=1:e do
    $a_l \leftarrow \frac{p}{\xi * b_{l-1}}$;
    $b_l \leftarrow \frac{q}{\xi^T * a_l}$;
end
$\overline{\pi} \leftarrow diag(a_e)\xi diag(b_e)$;
```

During the algorithm execution it is also possible to obtain the **errors in the satisfaction of the constraints**, namely

$$||\pi_l I_1 - p|| = \frac{||a_l \otimes (\xi b_l) - p||}{||p||} \tag{48}$$

$$||\pi_l I_2 - q|| = \frac{||b_{l-1} \otimes (\xi a_l) - q||}{||q||} \tag{49}$$

where $\otimes$ is the element-wise multiplication.

To implement the algorithm in MATLAB:

```
b = ones(N(2),1);
niter = 300;
```

```
Err_p = []; Err_q = [];
for i=1:niter
    a = p ./ (xi*b);
    Err_q(end+1) = norm( b .* (xi'*a) - q )/norm(q);
    b = q ./ (xi'*a);
    Err_p(end+1) = norm( a .* (xi*b) - p )/norm(p);
end

Pi = diag(a)*xi*diag(b);
```

a:

2.44234e+22
3.4833e-11
5.9052e+21
1.3659e+29
1.3040e+8

b:

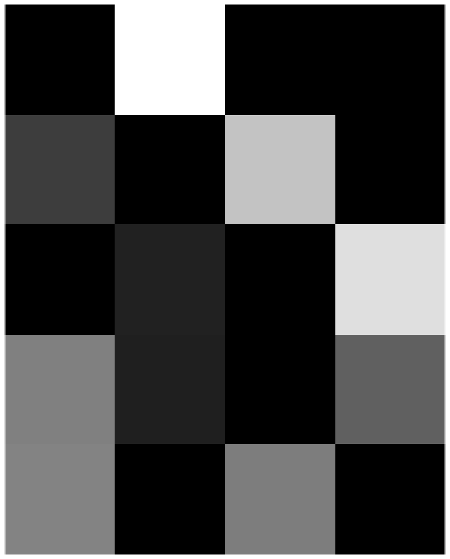
1.6414e+29
1.12937e-20
2.7933e+20
2.9102e-05

$\overline{\pi}$:

3.2088e-34	0.19985	4.5639e-37	0.0002
0.04773	1.6874e-93	0.1522	1.714e-61
3.1425e-36	0.02581	1.2251e-37	0.17412
0.09999	0.02431	1.0711e-05	0.07568
0.10226	1.2942e-42	0.09773	1.8324e-22

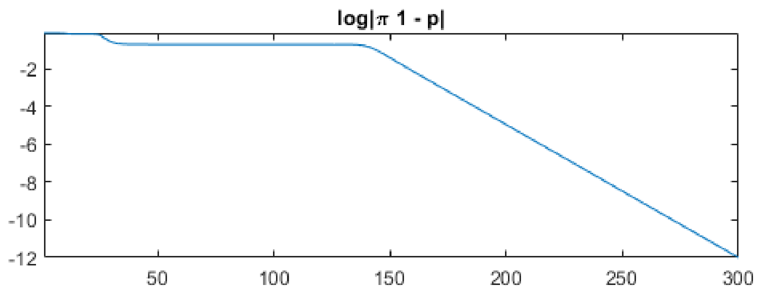
To display the coupling matrix in MATLAB:

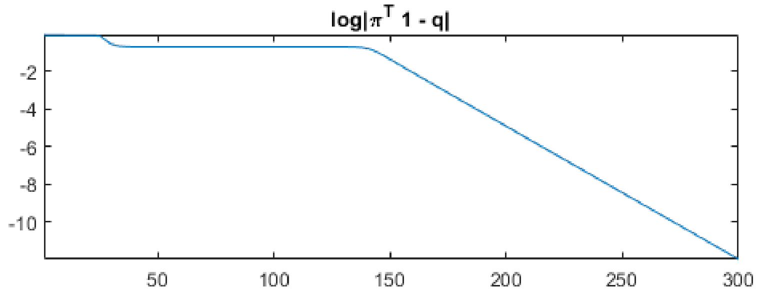
```
clf;
imageplot(Pi);
```



To display the **error evolution in a log-plot form**:

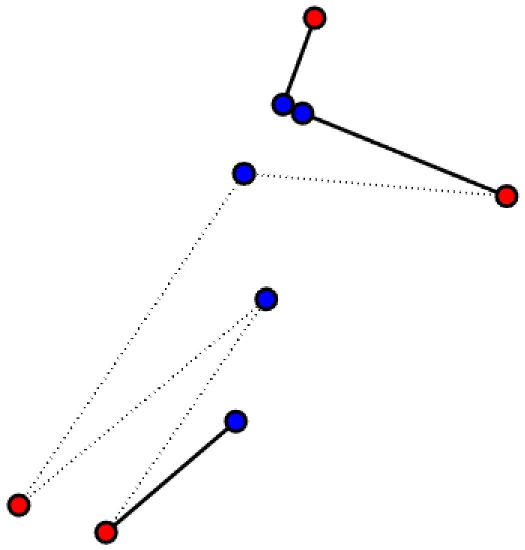
```
clf;
subplot(2,1,1);
plot(log10(Err_p)); axis tight; title('log|\pi 1 - p|');
subplot(2,1,2);
plot(log10(Err_q)); axis tight; title('log|\pi^T 1 - q|');
```





It is possible to show the connections among the points in the space. First, only the highest entries of the coupling matrix π has to be kept and used to draw a map between the two clouds. So, the strong connections are given by link (i, j) corresponding to large values of $\pi_{i,j}$. Then, the weaker connections are drawn, given by the small values of $\pi_{i,j}$.

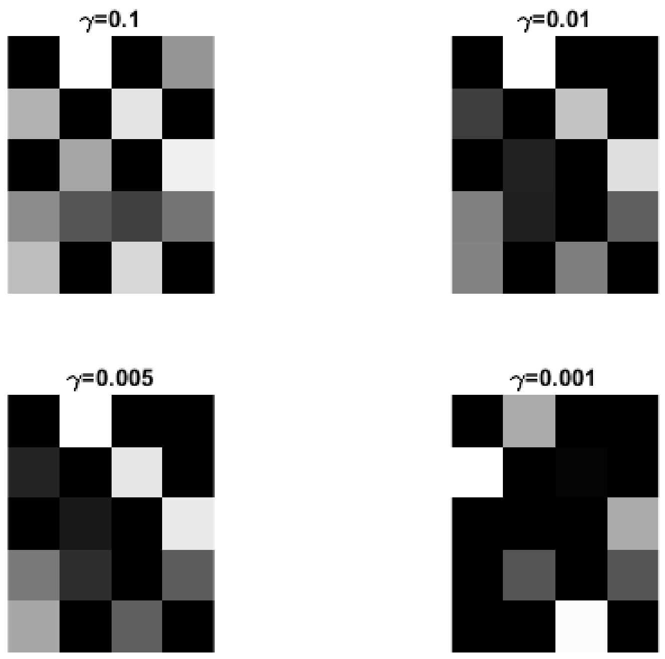
```
clf;
hold on;
A = sparse( Pi .* (Pi> min(1./N)*.7) ); [i,j,¬] = find(A);
h = plot([x(1,i);y(1,j)], [x(2,i);y(2,j)], 'k');
set(h, 'LineWidth', 2); % weaker connections.
A = sparse( Pi .* (Pi> min(1./N)*.3) ); [i,j,¬] = find(A);
h = plot([x(1,i);y(1,j)], [x(2,i);y(2,j)], 'k:');
set(h, 'LineWidth', 1);
plotp(x, 'b'); % plot the two point clouds.
plotp(y, 'r');
axis('off'); axis('equal');
```



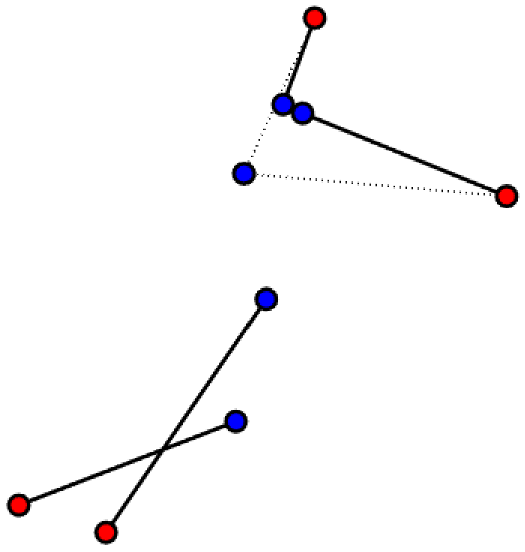
Now, let observe how the problem changes with different values of the regularization strength γ. Let get the same starting points and four different values of γ.

```
glist = [.1 .01 .005 .001 ];
niter = 300;
clf;
for k=1:length(glist)
    gamma = glist(k);
    xi = exp(-C/gamma);
    b = ones(N(2),1);
    for i=1:niter
        a = p ./ (xi*b);
        b = q ./ (xi'*a);
    end
    Pi = diag(a)*xi*diag(b);
    imageplot( Pi, ['\gamma=' num2str(gamma)], 2,2,k);
end
```

For a value of γ closer to zero, the problem becomes more similar to a discrete optimal transport as mentioned before.

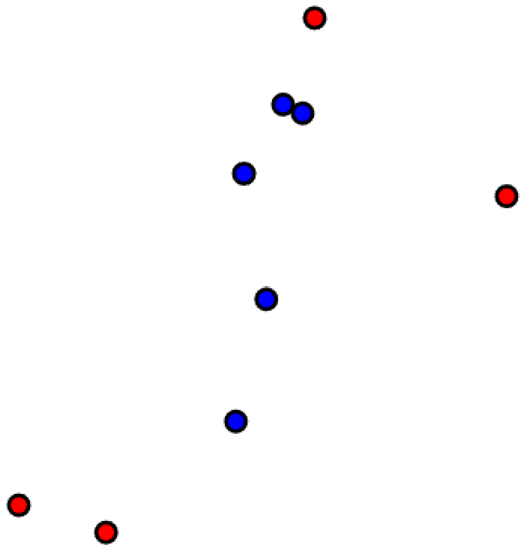


This phenomenon can be observed in the drawing of the case with $\gamma = 0.001$, where there are more strong links and less weak links.



4.2 Log-domain Sinkhorn's Algorithm

Let assume the same point clouds x, y of the previous example.



The Sinkhorn's Algprothm has been originally implemented using matrix-vector multiplication, which is unstable for small regularization strength γ. Now, let consider a **log-domain** implementation, which operates by iteratively updating the **Kantorovitch Dual Potentials** $(f, g) \in R^{N_1} \times R^{N_2}$. The updates are given by

$$f_i \leftarrow min_\gamma^q(C_{i,.} - g) \tag{50}$$

$$g_j \leftarrow min_\gamma^p(C_{.,j} - f) \tag{51}$$

where the regularized minimum operator is

$$min_\gamma^q(h) \equiv -\gamma \log \sum_i q_i e^{\frac{-h_i}{\gamma}} \tag{52}$$

$$min_\gamma^p(h) \equiv -\gamma \log \sum_i p_i e^{\frac{-h_i}{\gamma}} \tag{53}$$

In MATLAB:

```
p = ones(N(1),1)/N(1);
q = ones(1,N(2))/N(2);

minp = @(H,gamma)-gamma*log( sum(p .* exp(-H/gamma),1) );
minq = @(H,gamma)-gamma*log( sum(q .* exp(-H/gamma),2) );
```

The regularized *min* operator defined in this way is unstable, but it can be stabilized using the log-sum-exp technique, which relies on the fact for any constant $c \in R$:

$$min_\gamma^p(h + c) = min_\gamma^p(h) + c \tag{54}$$

and the stabilization is achieved using $c = min(h)$. In MATLAB:

```
minpp = @(H,gamma)minp(H-min(H,[],1),gamma) + min(H,[],1);
minqq = @(H,gamma)minq(H-min(H,[],2),gamma) + min(H,[],2);
```

So now let apply the Log-Domain Sinkhorn's Algorithm:

Algorithm 2: Sinkhorn's Algorithm

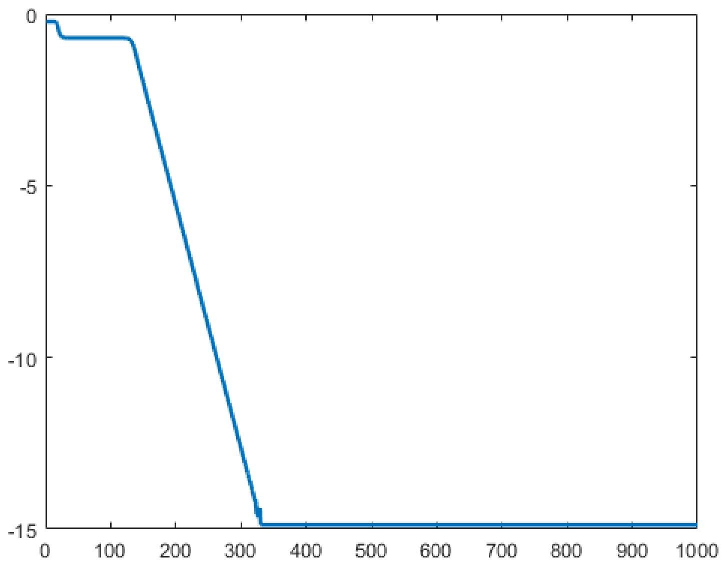
```
Initialize $f_0$ as a column vector of $N1$ zeros;
Set $e$ (max number of iterations);
for $l$=1:$e$ do
    $g_l \leftarrow min_\gamma^p(C - f_{l-1}) + min(C - f_{l-1})$;
    $f_l \leftarrow min_\gamma^p(C - g_l) + min(C - g_l)$;
end
$\overline{\pi} \leftarrow p \otimes e^{\frac{f+g-C}{\gamma}} \otimes q$;
```

```
gamma = .01
```

```
e = 1000;
f = zeros(N(1),1);
Err = [];
for it=1:e
    g = minpp(C-f,gamma);
    f = minqq(C-g,gamma);
    Pi = p .* exp((f+g-C)/gamma) .* q;
    Err(it) = norm(sum(Pi,1)-q,1);
end
```

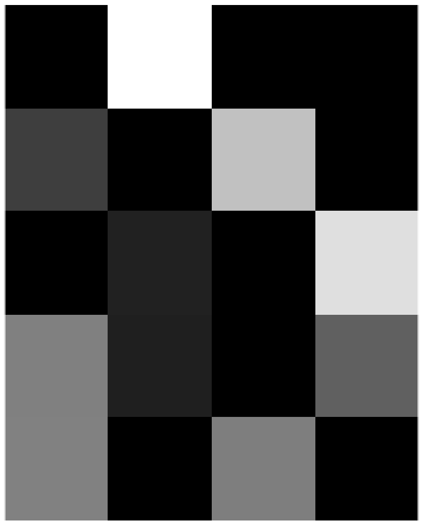
To display the error evolution in log-plot:

```
clf;
plot(log10(Err), 'LineWidth', 2);
```



To display the result:

```
imageplot(Pi);
```



4.3 Comparison between Classical Sinkhorn's Algorithm and Log-Domain Sinkhorn's Algorithm

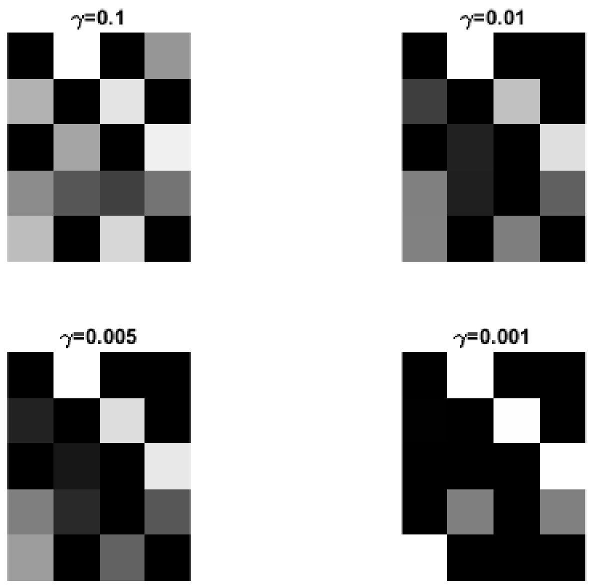
In this section the coupling matrices π obtained using the **classical version of the Sinkhorn's Algorithm** and the ones obtained exploiting the **Log-Domain version** have been compared. In particular, different values of γ have been used: $[0.1, 0.01, 0.005, 0.001]$.

As mentioned in the previous section, the classical version of the algorithm is unstable when small values of γ are used. For this reason the Log-domain version of the Sinkhorn's Algorithm has been introduced, which instead is stable for these values of the regularization strength γ.

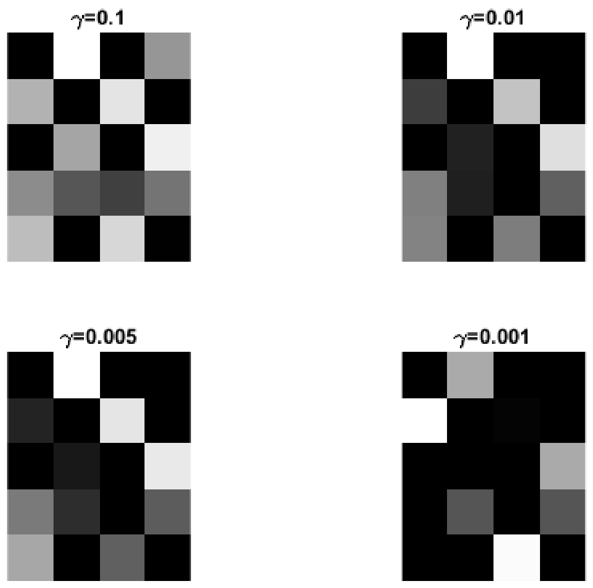
To compute the different coupling matrices π for the Log-domain version in MATLAB:

```
glist = [.1 .01 .005 .001 ];
e = 300; clf;
for k=1:length(glist)
    gamma = glist(k);
    f = zeros(N(1),1);
    for i=1:e
        g = minpp(C-f,gamma);
        f = minqq(C-g,gamma);
    end
    Pi = p .* exp((f+g-C)/gamma) .* q;
    imageplot( Pi , ['\gamma=' num2str(gamma)], 2,2,k);
end
```

Log-Domain Sinkhorn's Algorithm:



Classical Sinkhorn's Algorithm:



It is possible to notice by observing the previous figures that the resulting coupling matrices for $\gamma = [0.1, 0.01, 0.005]$ obtained using both the two different versions of the algorithm are the same. Instead, the more interesting case is the one with the smallest value of $\gamma = 0.001$ among the ones analyzed: the coupling matrices obtained from the two versions of the algorithm are very different. In particular, the one corresponding to the Log-domain version is more coherent with the other results of the algorithm (the ones with the different γ values). This can be noticed by the fact that three peaks (the white cells) are located in the same position in the matrices.

4.4 Transport Between Histograms

In the previous, a "special" case where p, q were uniform has been analyzed. Now, let assume a different scenario in which the histogram values p, q are not uniform, but the measures are defined on a uniform grid $x_i = y_i = \frac{i}{N}$, where N is their size. In this way, p, q are referred also as "**histograms**".

Let define the histograms p, q as translated **Gaussians** in 1-D. In MATLAB:

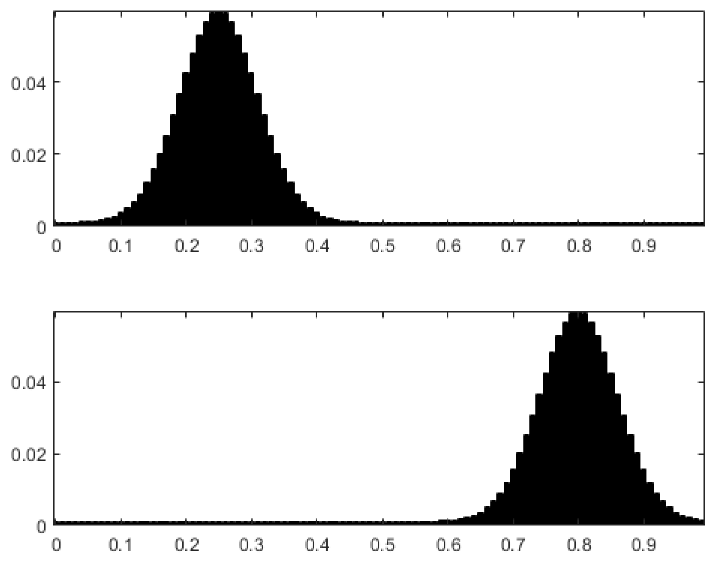
```
N = 100;
t = (0:N-1)'/N; %i/N

Gaussian = @(t0,sigma)exp( -(t-t0).^2/(2*sigma^2) );
sigma = .06;
p = Gaussian(.25,sigma);
q = Gaussian(.8,sigma);
```

To add some minimal mass and normalize them:

```
normalize = @(p)p/sum(p(:));
vmin = .02;
p = normalize( p+max(p)*vmin);
q = normalize( q+max(q)*vmin);
```

To plot them:

```
clf;
subplot(2,1,1);
bar(t, p, 'k'); axis tight;
subplot(2,1,2);
bar(t, q, 'k'); axis tight;
```



Let assume the **cost function** as a **1-D square Euclidean distance**:

$$C_{i,j} = (\frac{i}{N} - \frac{j}{N})^2 \tag{55}$$

So, the **Gibbs kernel** is a **Gaussian convolution** given by:

$$\xi = e^{\frac{C_{i,j}}{\gamma}} = e^{\frac{(\frac{i}{N} - \frac{j}{N})^2}{\gamma}} \tag{56}$$

```
gamma = (.03)^2;
[Y,X] = meshgrid(t,t);
xi = exp( -(X-Y).^2 / gamma); %matrix 100x100
```

ξ:

1	0.8948	...	0	0
0.8948	1	...	0	0
...	...	...	...	...
0	0	...	1	0.8948
0	0	...	0.8948	1

So now it is possible to execute the Sinkhorn's Algorithm:

```
b = ones(N,1);
```

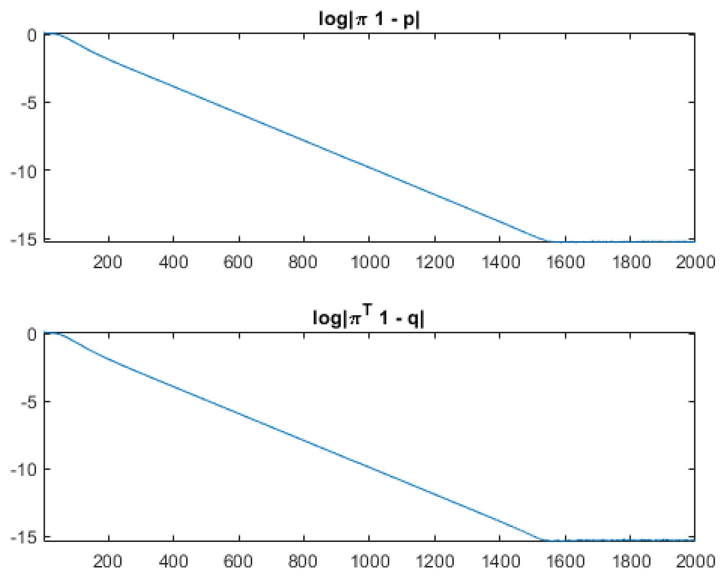
```
niter = 2000;
Err_p = []; Err_q = [];
for i=1:niter
    a = p ./ (xi*b);
    Err_q(end+1) = norm( b .* (xi*a) - q )/norm(q);
    b = q ./ (xi'*a);
    Err_p(end+1) = norm( a .* (xi'*b) - p )/norm(p);
end
Pi = diag(a)*xi*diag(b);
```

$\overline{\pi}$:

0.00045	0.00032	...	0	0
0.00032	0.00029	...	0	0
...	...	...	...	...
0	0	...	0.00034	0.00038
0	0	...	0.00036	0.00049

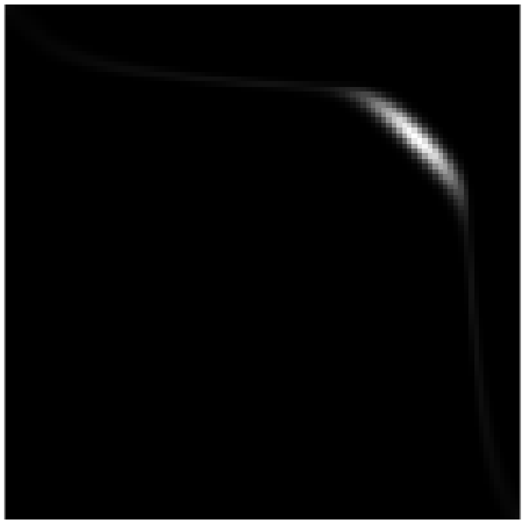
To display the error evolution in log-plot:

```
clf;
subplot(2,1,1);
plot(log10(Err_p)); axis tight; title('log|\pi 1 - p|');
subplot(2,1,2);
plot(log10(Err_q)); axis tight; title('log|\pi^T 1 - q|');
```



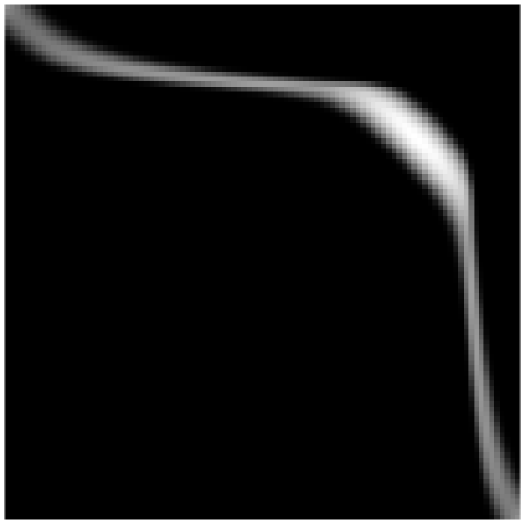
To display $\overline{\pi}$:

```
imageplot(Pi);
```



The plot can also be shown in the logarithmic plot to increment the readability of the low values:

```
imageplot(log(Pi+1e-5));
```



It is possible to compute an approximation of the transport plan between the two measures by computing the **barycentric projection map**, given by:

$$t_i \in [0,1] \longmapsto s_j \equiv \frac{\sum_j \pi_{i,j} t_j}{\sum_j \pi_{i,j}} = \frac{[a \otimes \xi(b \otimes t)]_j}{p_i} \tag{57}$$

where $\otimes$ and $\div$ are the entry-wise multiplication and division. This computation can be done using only multiplication with the kernel ξ:

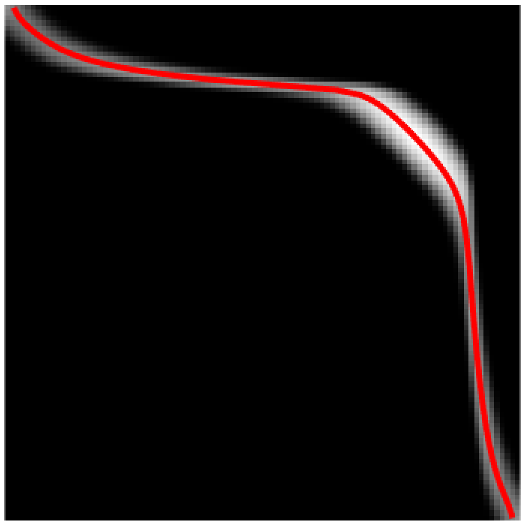
$$s = \frac{(\xi \times (b \otimes t)) \otimes a}{p} \tag{58}$$

In MATLAB:

```
s = (xi*(b.*t)) .* a ./ p;
```

To display the transport map over the coupling:

```
clf; hold on;
imagesc(t,t,log(Pi+1e-5)); colormap gray(256);
plot(s,t, 'r', 'LineWidth', 3);
axis image; axis off; axis ij;
```



5 Generative Adversarial Network using Optimal Transport

In the **Generative Adversarial Network** framework, in general, there are two models: a generative network, also called **Generator**, and a discriminative network, also called **Discriminator**. The objective of the Generator is, starting from a random sample extracted from the latent space z with an a priori distribution $p_z(z)$, to generate a sample x belonging to the distribution p_g which has to be as similar as possible to the distribution p_{data} of the real samples. On the other hand, the Discriminator is trained to output the probability that a certain sample x is belonging to the distribution p_{data}. The resulting loss function used in the classical GANs is called **minimax loss**, since the Generator wants to minimize the discriminator's estimate of the probability that a fake instance is actually fake and the Discriminator wants to maximize his estimate of the probability that a fake instance is actually and a real instance is actually real.

5.1 Wasserstein GAN

One of the most interesting application of the Optimal Transport in the Machine Learning field is the **Wasserstein Distance** applied to the **Generative Adversarial Models**, as described in the paper "Wasserstein GAN" by Arjovsky et al [4]. The core idea of this application is to exploit the ability of the Optimal Transport approach in evaluating the divergence among two distributions.

In the Wasserstein GAN the minimax loss function is not used in favour of the so called **Wasserstein loss**. This loss function depends on a modification of the GAN scheme in which the Discriminator does not output a probability on the veracity of an instance. Instead, it outputs a number for each instance which does not belong to the range $[0, 1]$, so it has not an upper bound on its value. The objective of the Discriminator's training is to output larger numbers for the real instances than for fake ones. Since the Discriminator is not giving a probability on each instance, in the Wasserstein GAN it is called **Critic**. The critic loss is given by:

$$D(x) - D(G(z)) \tag{59}$$

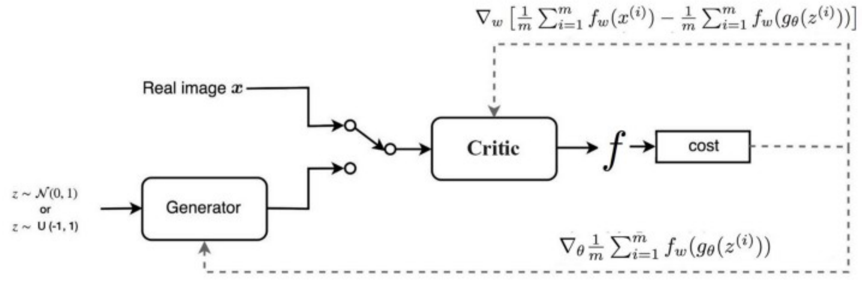
This difference between the output for the real instances and the fake ones is given by the so called **Earth Mover Distance** (or **Wasserstein-1**):

$$W(P_r, P_q) = \inf_{\gamma \in \Pi(P_r, P_g)} E_{(x,y)\ \gamma}[||x - y||] \tag{60}$$

where:

- P_r: distribution of the real instances. The instance x in the critic loss function is sampled from here.
- P_g: distribution of the fake instances. The instance $G(z)$ in the critic loss function is sampled from here.
- $\Pi(P_r, P_g)$: set of all joint distributions $\gamma(x, y)$ among P_r and P_g.

The idea is that $\gamma(x, y)$ indicates how much "mass" must be transported from x to y in order to transform the distribution P_r into the distribution P_g. The Earth Mover Distance is the "cost" of the optimal transport plan.


$\nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
Real image x
Critic
f
cost
$z \sim \mathcal{N}(0,1)$
or
$z \sim U(-1,1)$
Generator
$\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$

The advantages of the Wasserstein GANs are:

- **Robustness**: training a Wasserstein GAN does not require maintaining a careful balance in training of the discriminator and the generator, and does not require a careful design of the network architecture.
- **Resistance against Mode Collapse**: this phenomenon is typical of the GANs. It happens when the Generator learns only to produce a single

output or a small set of outputs. This is due to the vanishing gradient problem on the Discriminator, which can get stuck in a local minimum without learning to reject this kind of fake instance. This leads the Generator to produce always the same outputs because it will be sure that they will be recognized as real. The Wasserstein GAN does not suffer the vanishing gradient problem and for that reason is resistant against the Mode Collapse phenomenon.

- **Resistance against Mode Dropping**: this phenomenon happens in the GANs when the Generator is not able to generate fake instances from regions of the fake data distribution similar to certain regions of the real data distribution. This is drastically reduced in the Wasserstein GAN.
- **Continuos Estimation of the Earth Mover Distance**: this ability of the Wasserstein GAN is interesting because it allows to plot the learning curves to debug, to search the hyperparameters and to evaluate the sample quality.

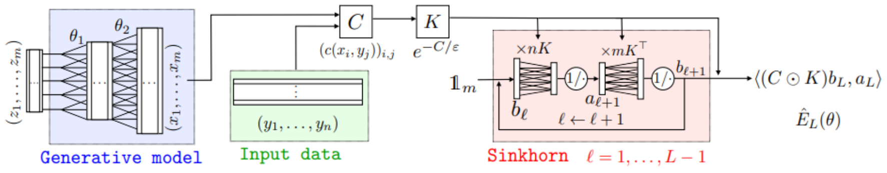
5.2 Sinkhorn's GAN

As well as the Wasserstein GANs, an application of the optimal transport for Generative Adversarial Network is given by the Sinkhorn's GANs. The main difference among WGAN and Sinkhorn's GAN consists in the loss it uses. This type of GANs exploit the so called **Sinkhorn Loss**. This loss corresponds to the result of the Sinkhorn's Algorithm, which has been analyzed in section 4, and compute the divergence between two distributions including an entropic regularization term. This approach can be seen as an interpolation between the Wasserstein (OT) loss (when $\gamma = 0$) and the Maximum Mean Discrepancy (MMD) loss (when $\gamma \to +\infty$). The latter is another way to compute the divergence among two distributions P, Q using a kernel k, as described in [7]:

$$M_k(P,Q) = ||\mu_P - \mu_Q||_H^2 = E_P[k(x,x')] - 2E_{P,Q}[k(x,y)] + E_Q[k(y,y')] \quad (61)$$

where:

- k: usually implemented as a Gaussian kernel, $k(x,x') = e^{||x-x'||^2}$
- μ_P: mean value of the distribution P
- μ_Q: mean value of the distribution Q
- $||.||_H$: infinity norm
- $E[.]$: expected value



The Sinkhorn's GAN benefits of all the advantages of both the Wasserstein and MMD approaches:

- **Sample Complexity**: MMD sample complexity is $O(\frac{1}{\sqrt{N}})$, while OT sample complexity is $O(\frac{1}{N^{\frac{1}{d}}})$. The sample complexity of the Sinkhorn's approach has not been proved but it empirically behaves in a similar manner of the MMD [8].
- **Diversified Samples**: empirically it has been shown that the MMD loss benefits of diversified samples.
- **Non-Strictly Positive Kernels**: The Sinkhorn loss, as is the case for the OT problem, can be defined with any cost C, whereas MMD loss is only meaningful when used with positive definite kernels k.
- **Mode Collapse and Mode Dropping**: The Sinkhorn loss is resistant against the Mode Collapse and Mode Dropping phenomenons as the Wasserstein loss, as described in section 5.1.

6 Appendix

6.1 Appendix A: Difference among discrete optimal transport problem and optimal assignment problem

The optimal assignment problem is a special case of the discrete optimal transport problem in which the number of source points is equal to the one of the destination points, namely $n_0 = n_1 = n$, and in which the weights of the sources and the destinations are all equal among them and constants, namely $p_{0,i} = 1/n, p_{1,i} = 1/n$. The optimal assignment problem can be represented as a problem in which there are n workers and n jobs. Each worker i has a cost to execute the job j. The objective of the problem is to minimize the total cost by assigning just one job to each worker. It can be seen as:

$$\begin{aligned}
&\min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}x_{i,j} \\
&s.t. \\
&\sum_{i=1}^{n} x_{i,j} = 1 \qquad \forall\, job\ j = 1, ..., n \\
&\sum_{j=1}^{n} x_{i,j} = 1 \qquad \forall\, worker\ j = 1, ..., n \\
&x_{i,j} \in \{0, 1\}
\end{aligned} \tag{62}$$

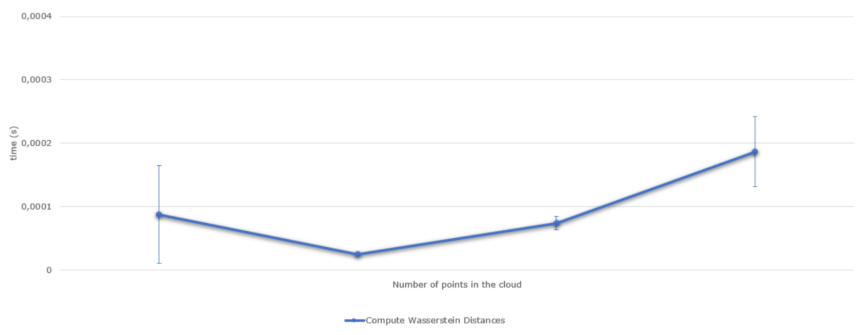
6.2 Appendix B: Computational Bottlenecks

All the performances in this section have been evaluated using four different numbers of points and 30 repetitions of the algorithm. The confidence intervals have been computed using a 0.05 significance level. All the algorithms have been executed on a Intel i7-8565 CPU using MATLAB.

6.2.1 Optimal Transport with Linear Programming

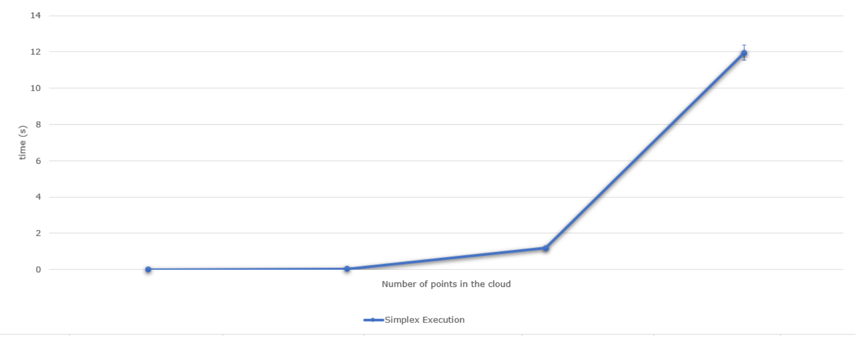
In the **Discrete Optimal Transport** there are two steps that have to be analyzed: the computation of the **Wasserstein Distances** and the execution of the **Simplex Algorithm** to solve the problem.

Compute the Wasserstein Distances:



The tested values are $N1 = [15, 30, 60, 120]$ and $N2 = [20, 40, 80, 160]$.

Solve the Simplex Algorithm:

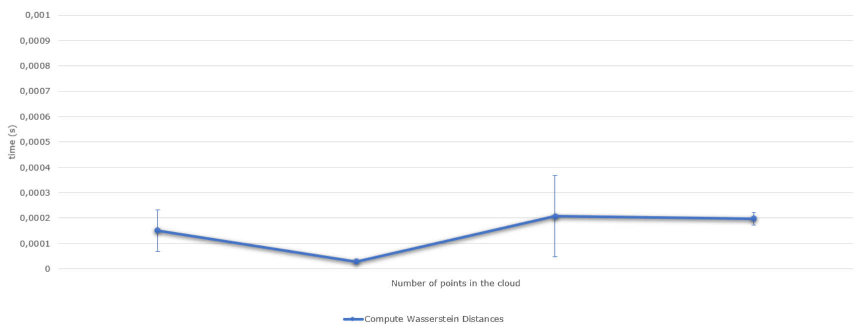


The tested values are $N1 = [15, 30, 60, 120]$ and $N2 = [20, 40, 80, 160]$.

It is possible to observe that the time required to solve the Simplex Algorithm is the main bottleneck of the Discrete Optimal Transport. This means that it is possible to improve the performance of this approach by using an iterative or approximated algorithm.

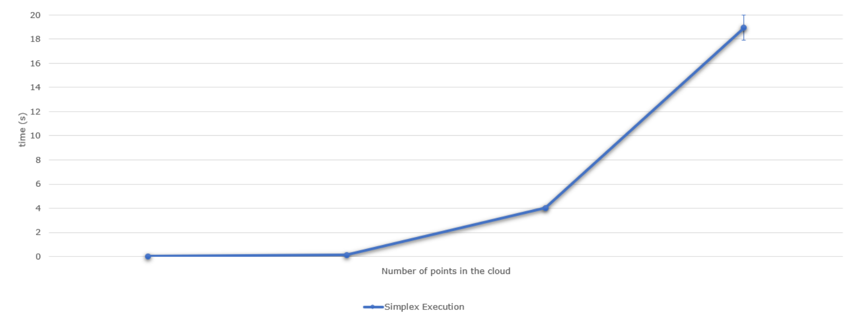
There are the same possible bottlenecks also in the execution of the **Optimal Assignment**.

Compute the Wasserstein Distances:



The tested values are $N = [20, 40, 80, 160]$.

Solve the Simplex Algorithm:



The tested values are $N = [20, 40, 80, 160]$.

The observations on the performances of the Optimal Assignment case are exactly the same of the Discrete Optimal Transport case.

6.2.2 Entropic Regularization for Optimal Transport

The main computational bottleneck of Sinkhorn's iterations is the vector-matrix multiplication against kernel ξ and ξ^T with complexity $O(nm)$, which are intrinsically GPU friendly and so they can easily be parallelized and speeded up. The success of the Sinkhorn's Algorithm to compute the Optimal Transport distances between histograms at large scale is given by the possibility of executing the algorithm multiple times in parallel handling matrix-matrix operations. Let assume $p_1, ..., p_N$ histograms in Σ_n and $q_1, ..., q_N$ in Σ_m. The goal is to compute all N approximate distances $W_\gamma(p_1, q_1), W_\gamma(p_2, q_2), ..., W_\gamma(p_N, q_N)$. In order to do so, $P = [p_1, p_2, ..., p_N]$ and $Q = [q_1, q_2, ..., q_N]$ can be used to represent the $n \times N$ and $m \times N$ matrices which store the histograms p_i and $q_i, \forall i = 0, 1, ..., N$. It can be noticed that all Sinkhorn iterations for all these N pairs of histograms can be computed in parallel:

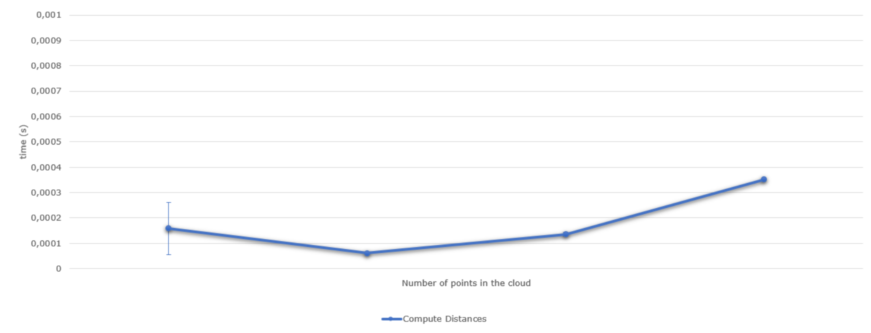
$$A_{l+1} = \frac{P}{\xi B_l} \tag{63}$$

$$B_{l+1} = \frac{Q}{\xi^T A_{l+1}} \tag{64}$$

initialized with $B_0 = 1_{m \times N}$ and where $\div$ corrisponds to the entrywise division of matrices.

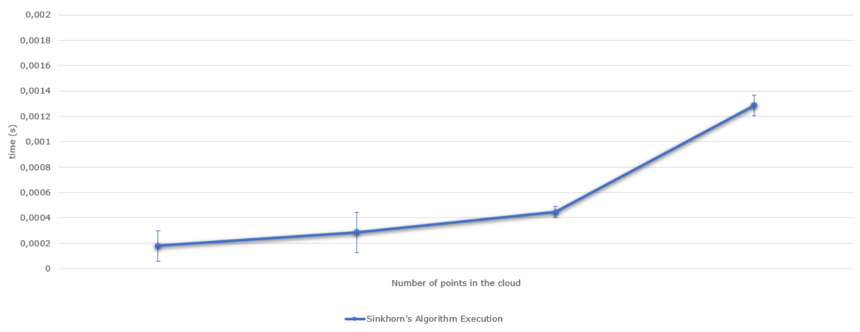
The time required to compute the distances and to perform the Sinkhorn's Algorithm, in both the classical and in the Log-Domain versions, has been evaluated.

Compute the distances:



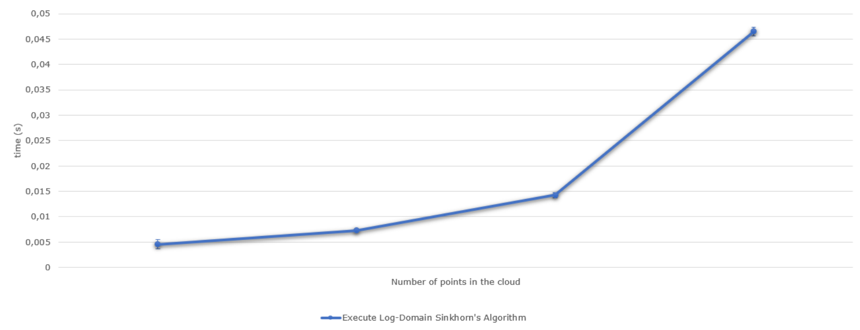
The tested values are $N1 = [15, 30, 60, 120]$ and $N2 = [20, 40, 80, 160]$ with 100 iterations in the algorithm.

Execute the Sinkhorn's Algorithm:



The tested values are $N1 = [15, 30, 60, 120]$ and $N2 = [20, 40, 80, 160]$ with 100 iterations in the algorithm.

Execute the Log-Domain Sinkhorn's Algorithm:



The tested values are $N1 = [15, 30, 60, 120]$ and $N2 = [20, 40, 80, 160]$ with 100 iterations in the algorithm.

It is possible to notice that the Sinkhorn's Algorithm in both its forms is drastically more efficient than the Simplex Algorithm. Moreover, there is a trade-off between the classical Sinkhorn's Algorithm and the Log-Domain version: the first one is faster but unstable for small values of the regularization strength γ as mentioned in 4.3, the second one is slower but stable also for small values of γ.

References

[1] Sira Ferradans, Nicolas Papadakis, Gabriel Peyré, and Jean-François Aujol. Regularized discrete optimal transport, 2013.

[2] Michaël Perrot, Nicolas Courty, Rémi Flamary, and Amaury Habrard. Mapping estimation for discrete optimal transport. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[3] Julie Digne, David Cohen-Steiner, Pierre Alliez, Fernando De Goes, and Mathieu Desbrun. Feature-Preserving Surface Reconstruction and Simplification from Defect-Laden Point Sets. *Journal of Mathematical Imaging and Vision*, pages 1–14, January 2013.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[5] Alexandre Gramfort, Gabriel Peyré, and Marco Cuturi. Fast optimal transport averaging of neuroimaging data, 2015.

[6] Soheil Kolouri and Gustavo K. Rohde. Transport-based single frame super resolution of very low resolution face images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4876–4884, 2015.

[7] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. *arXiv preprint arXiv:1705.08584*, 2017.

[8] Aude Genevay, Gabriel Peyré, and Marco Cuturi. Learning generative models with sinkhorn divergences, 2017.