Random Osborne Algorithm for Matrix Balancing

Optimal transport report

Alexi Canesse[1,2]

Optimal transport course
Part of the MVA program at ENS Paris-Saclay.

Computer Science and Mathematics
École Normale Supérieur Paris-Saclay
École Normale Supérieur de Lyon
Paris, France
17th January 2024

1 alexi.canesse@ens-lyon.fr, ENS de Lyon, France
2 alexi.canesse@ens-paris-saclay.fr, ENS Paris-Saclay, France

Table of content

Abstract

We dived in matrix balancing, a crucial task with diverse applications such as an increased stability in eigen-spaces decomposition, and introduces Osborn*'s algorithm along with its variants. The focus is on presenting both the theoretical foundations and practical implementation aspects. A comprehensive guide to achieving an efficient implementation is provided, supported by clean, openly available code.*

Theoretical results, fundamental to the article, are presented: near-linear convergence. They are showcased through numerical experiments, demonstrating relative success in illustrating their complexities. Furthermore, a connection is established between Osborn*'s and* Sinkhorn*'s algorithms, leading to a thorough comparative analysis across various types of kernels for matrix balancing. The findings highlight instances where* Sinkhorn *encounters challenges, contrasting with* Osborn*'s consistent convergence across variants. Leveraging these insights, a compelling demonstration is presented where* Osborn *excels while* Sinkhorn *struggles and fails to converge in a constructed kernel.*

1 Introduction

The article we dived into for our project [AP23] introduces Osborn's algorithm, a matrix balancing algorithm widely employed in preconditioning matrices for tasks such as eigen-decomposition. The authors provided a significant contribution by proving near-linear runtime in the sparsity of the input matrix. This achievement does not enhances the algorithm's efficiency but gives us a better understanding of its performances.

1.1 Presentation of the problem

The matrix balancing problem arises from the recognition that matrices encountered in practical applications may have varying scales across rows and columns. These imbalances can lead to numerical instability and adversely impact the accuracy of numerical computations such as eigenvalues/eigenvectors decomposition [CD00; Che01] or solving linear systems [Che01]. Matrix balancing algorithms aim to address these issues by adjusting the scaling of rows and columns, thereby enhancing the overall numerical properties of the matrix.

Other problems can be reduced to matrix balancing, which gives direct applications. For instance, in optimal transport with problems where decision variables are probability distributions [Alt22]; the approximating Min-Mean-Cycle which can be solved in near-linear runtime in the number of edges [AP22] using results from matrix balancing and in particular, the main result of the studied article.

Notation 1.1 *Let c and r respectively be the column-wise sum and the row-wise sum of matrices ie.*

$$c : \begin{cases} \mathcal{M}_{n,m}(\mathbb{K}) & \to & \mathbb{K} \\ A & \mapsto & A^\top \mathbf{1} \end{cases} \quad \text{and} \quad r : \begin{cases} \mathcal{M}_{n,m}(\mathbb{K}) & \to & \mathbb{K} \\ A & \mapsto & A\mathbf{1}. \end{cases}$$

Definition 1.1 *Let $A \in \mathcal{M}_n(\mathbb{R}_+)$ be a non negative square matrix, $\varepsilon \geq 0$ and $k \in \mathbb{N}^\star$. The matrix A is (ε, k)-balanced if*

$$\frac{||c(A) - r(A)||_k}{\sum_{i,j} a_{i,j}} \leq \varepsilon. \tag{1}$$

Furthermore, if A is $(0, k)$-balanced, we say that A is balanced.

Definition 1.2 *The ε, k-**approximate matrix balancing problem** is: given a square non-negative matrix $K \in \mathcal{M}_n(\mathbb{R}_+)$, $\varepsilon \geq 0$ and $k \in \mathbb{N}^\star$, find a vector $x \in \mathbb{R}^n$ such that $\mathcal{D}(e^x)K\mathcal{D}(e^{-x})$ is (ε, k)-balanced.*

1.2 Related work

Iterative algorithms The Osborn algorithm, introduced by Osborn in [Osb60] and later discussed in [PR71], is implemented in Scipy as a default matrix balancing method. Other iterative methods have been introduces. Notably, the Sinkhorn-Knopp algorithm [SK67], a special case of Bregman's balancing method [LS81], iteratively rescales each row and column until convergence. However, it converges linearly, which is impractical for large and sparse matrices, as discussed by Soules et al. in [Sou91]. Parlett introduced an approach in [PR71] to approximately balance matrices by ensuring the diagonal consists only of powers of 2. The advantage of this method is the elimination of floating-point errors when computing the balanced matrix on base-two computers, making it "good enough" in practice. Some efforts are also made to compare balancing algorithms on real use cases [SZ90].

Issues in Matrix Balancing While matrix balancing is generally effective, there are corner cases where it can worsen the conditioning of matrices. Watkins et al. highlighted some of these cases in [Wat06]. Additionally, James et al. explained, in [JLL14], these issues and proposed modifications to LAPACK to mitigate them.

Theoretical bounds [KK96] establishes an upper bound on the norm of the scaling factors and presents a polynomial-time complexity bound for the computation of scaling factors with a prescribed accuracy.

Characterizations of Non-Negative Balancable Matrices [Osb60] and [Eav+85] offer characterizations of non-negative balancable matrices, contributing to the theoretical understanding of the properties of such matrices. Additionally, [KKS97] provides insights into the complexity aspects of this problem.

Tensor Matrix Balancing Sugiyama et al. extended matrix balancing to tensors in [SNT17], providing a perspective on balancing operations in multi-dimensional spaces. This work contributes to the broader understanding of matrix balancing techniques applied beyond traditional matrices.

1.3 Contributions of the paper

Their main contribution is Theorem 2.2. It exhibits a variant of Osborn's algorithm with near-linear runtime in the input sparsity. It also shows that improving the runtime dependence in ε can be improve from ε^{-2} to ε^{-1} without an additional factor n.

Theorem 1.1 *Let $K \in \mathcal{M}_n(\mathbb{R}_+)$ be a balanceable non negative square matrix and $\varepsilon \geq 0$. Random* Osborn *solves $(\varepsilon, 1)$-approximate matrix balancing problem in T operations where there exists $c > 0, \delta > 0$ such that*

$$\mathbb{E}(T) = \mathcal{O}\left(\frac{m}{\varepsilon} \min\left\{\frac{1}{\varepsilon}; d\right\} \log \kappa\right) \quad \textit{and} \quad \mathbb{P}\left(T \leq c\frac{m}{\varepsilon} \min\left\{\frac{1}{\varepsilon}; d\right\} \log \kappa \log \frac{1}{\delta}\right) \geq 1 - \delta$$

where m is the number of nonzero entries in K, d is the diameter of the graph associated to K and $\kappa = \sum_{i,j} K_{i,j} / \min_{i,j} K_{i,j}$.

1.4 Our contributions

We conducted numerical experiments to empirically validate the theorems established in the original article.

We establishment of a meaningful connection between Osborn' and Sinkhorn' algorithms. Through extensive comparative analyses, we identify scenarios where Sinkhorn faces challenges. We showcase Osborn's consistent convergence across variants and construct a kernel where Osborn excels while Sinkhorn fails to converge, a demonstration of our understanding of their strengths.

2 Main body

2.1 Notations

For convenience and clarity, we consolidate the notations used throughout our discussions. These notations adhere to standard conventions, ensuring consistency and facilitating a concise representation of key elements:

- n denotes the size of the matrix, representing both the number of rows and columns.
- m signifies the count of non-zero coefficients within the matrix.
- $K \in \mathcal{M}_n(\mathbb{R}_+)$ is the matrix requiring balancing.
- ε serves as the parameter for the accuracy of the balancing process.
- G_K represents the graph associated with K, and d stands for its diameter.
- The conditioning number of K, denoted by κ, is computed as the sum of all elements in K divided by the minimum non-zero element in K.
- The exp function is applied in an element-wise manner for both vectors and matrices.

2.2 Presentation of the method

The algorithm operates iteratively, selecting column/row pairs and adjusting the scaling factor to ensure that their sums are equal.

```
osborn(K, ε):
    x = 0 ∈ ℝ^n
    while not(is_balanced(𝔻(e^x)K𝔻(e^{-x}), ε)):
        Choose k ∈ [n] # This is where variants differ
        x += (log(c_k(𝔻(e^x)K𝔻(e^{-x}))) - log(r_k(𝔻(e^x)K𝔻(e^{-x}))))/2
    return x
```

Figure 1: Pseudo-code of Osborn's algorithm.

There are *many* way to choose the next coordinate to update and hence many variants of the algorithm. The article focuses on four of them:

- **Cyclic Osborn** Cycle through the coordinates. (*eg.* 1, 2, 3, 1, 2, 3, 1, …).
- **Random-Reshuffle Cyclic Osborn** Cycle through the coordinates using a new random permutation for each cycle. (*eg.* 2, 1, 3, 1, 2, 3, 1, 3, 2, …).
- **Greedy Osborn** Choose k where the imbalance is maximal *ie.*

$$k = \text{argmax}_k \left| \sqrt{r_k(\mathbb{D}(e^x)K\mathbb{D}(e^{-x}))} - \sqrt{c_k(\mathbb{D}(e^x)K\mathbb{D}(e^{-x}))} \right|.$$

- **Random Osborn** Uniformly sample k independently between each call.

2.2.1 Implementation

The implementations were carried out in Python, providing us with a transparent and customizable framework for thorough experimentation. The source code for our implementations is available on GitHub. The implementation uses sparse representation for matrices and aimed to provide an implementation using data structures as close as possible as those used to compute the theoretical results. However, we were not able to find a detailed presentation of those and had to do our best to match the article. We did not give any focus on bit complexity. The authors recommanded a log-domain implementation to enable a logarithmic bit-complexity for some variants and increase numerical stability. However, we decided not to follow through in order to increase speed because we did not worry about bit-complexity.

At each update of x, we update the balanced matrix by multiplying the corresponding row and column. With the sparse representation, this operation is proportional to m. It is even $\mathcal{O}(m/n)$ on average! To compute the optimization function, the sum of elements in DAD^{-1} is computed in $\mathcal{O}(m)$ thanks to the sparse representation and $||c(DAD^{-1}) - r(DAD^{-1})||_1$ is also computed in $\mathcal{O}(m)$. Indeed, we use $||c(DAD^{-1}) - r(DAD^{-1})||_1 = \sum_j \left|\sum_i (DAD^{-1} - (DAD^{-1})^\top)_{i,j}\right|$. **Those techniques are not presented in the article and we had to come up with on our own.**

2.3 Matrix balancing as a convex optimization problem

A key part of the proofs of convergence given on the paper and other works rely mainly on a seeing the matrix balancing problem as an optimization problem. Let $\Phi : x \mapsto$

$\log \sum_{i,j} e^{x_i - x_j} K_{i,j}$. The gradient of this function is

$$\nabla \Phi(x) = \frac{r\left(\mathcal{D}(e^x) K \mathcal{D}(e^x)\right) - c\left(\mathcal{D}(e^x) K \mathcal{D}(e^x)\right)}{\sum_{i,j} \left(\mathcal{D}(e^x) K \mathcal{D}(e^x)\right)_{i,j}}.$$

We can clearly notice a connection with Equation (1) and see that x is a solution to the ε, k-matrix balancing problem if and only if $||\nabla \Phi(x)||_k \leq \varepsilon$. The function Φ is convex and approximately solving the convex optimization problem $\min_x \Phi(x)$ solves the matrix balancing problem.

Alexandre d'Aspremont says in his MVA course on convex optimization that writing a problem as a convex optimization problem is almost solving it, which shows how important this statement is.

2.4 Theoretical guarantees

In this section, we present the theorems from the article without providing proofs, as their derivation relies heavily on the content of the original article.

Lemma 2.1 *A matrix $K \in \mathcal{M}_n(\mathbb{R}_+^\star)$ is balanceable if and only if its associated graph is strongly connected [Osb60].*

Theorem 2.1 *Let $K \in \mathcal{M}_n(\mathbb{R}_+)$ be a balanceable non negative square matrix and $\varepsilon \geq 0$. Greedy Osborn solves $(\varepsilon, 1)$-approximate matrix balancing problem in*

$$\mathcal{O}\left(\frac{n^2}{\varepsilon} \min\left\{\frac{1}{\varepsilon}; d\right\} \log \kappa\right)$$

where d is the diameter of the graph associated to K and $\kappa = \sum_{i,j} K_{i,j} / \min_{i,j} K_{i,j}$.

Theorem 2.2 *Let $K \in \mathcal{M}_n(\mathbb{R}_+)$ be a balanceable non negative square matrix and $\varepsilon \geq 0$. Random Osborn solves $(\varepsilon, 1)$-approximate matrix balancing problem in T operations where there exists $c > 0, \delta > 0$ such that*

$$\mathbb{E}(T) = \mathcal{O}\left(\frac{m}{\varepsilon} \min\left\{\frac{1}{\varepsilon}; d\right\} \log \kappa\right) \quad \text{and} \quad \mathbb{P}\left(T \leq c\frac{m}{\varepsilon} \min\left\{\frac{1}{\varepsilon}; d\right\} \log \kappa \log \frac{1}{\delta}\right) \geq 1 - \delta$$

where m is the number of nonzero entries in K, d is the diameter of the graph associated to K and $\kappa = \sum_{i,j} K_{i,j} / \min_{i,j} K_{i,j}$.

Theorem 2.3 *Let $K \in \mathcal{M}_n(\mathbb{R}_+)$ be a balanceable non negative square matrix and $\varepsilon \geq 0$. Random cyclic Osborn solves $(\varepsilon, 1)$-approximate matrix balancing problem in T operations where there exists $c > 0, \delta > 0$ such that*

$$\mathbb{E}(T) = \mathcal{O}\left(\frac{mn}{\varepsilon} \min\left\{\frac{1}{\varepsilon}; d\right\} \log \kappa\right) \quad \text{and} \quad \mathbb{P}\left(T \leq c\frac{mn}{\varepsilon} \min\left\{\frac{1}{\varepsilon}; d\right\} \log \kappa \log \frac{1}{\delta}\right) \geq 1 - \delta$$

where m is the number of nonzero entries in K, d is the diameter of the graph associated to K and $\kappa = \sum_{i,j} K_{i,j} / \min_{i,j} K_{i,j}$.

2.5 Numerics

Computing numeric experiments was not easy. Getting the actual number of arithmetic operations is a complex endeavour, especially with a high-level language such as Python. We decided do use the running time to estimate how the number of arithmetic operations will evolve as a function of given parameters. This is not a precise measurement. However, all algorithms presented are are basically the same and this measure should be a fair assessment of convergence speed.

2.5.1 Data generation

In order to conduct numerical experiments, it is useful to be able to generate matrices with given parameters such as value of κ and sparsity. For that, we start by generating a (n, n) random matrix with m non-zero values using `scipy.sparse.random`. Then we assign values to those non-zero entries according to an exponential distribution of scale 1. We then want to modify those values to get the expected value for κ. To do so, we add a value x to all non-zeros entries. If the generated matrix was K, to find x, we solve

$$\kappa = \frac{\sum_{i,j} K_{i,j} + m \times x}{K_{\min} + x}.$$

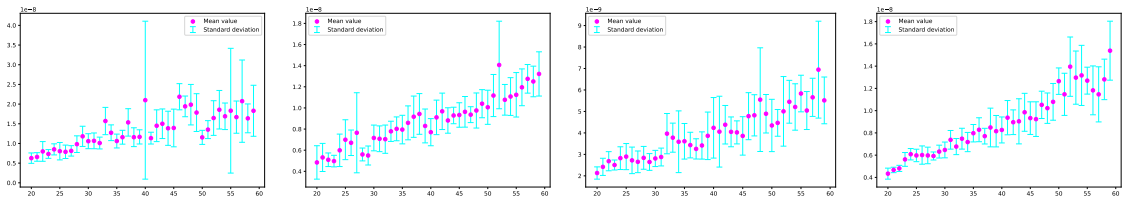
We then find that the value to add is

$$x = \frac{\kappa \times K_{\min} - \sum_{i,j} K_{i,j}}{m - \kappa}.$$

Sometimes, x is negative and K end up not being non-negative and we just start again from the start. Finally, we want the matrix to be balanceable. Hence, we start again until its associated graph is strongly connected (Lemma 2.1).

2.5.2 Complexity as a function of n

In order to experimentally show the convergence speed of theorems presented in Section 2.4, we fixed the parameters m and ε and let n in $[\![20, 60]\!]$. Each matrix is generated using an exponential distribution according to the scheme[3] presented in Section 2.5.1. The Figure 2 shows the results.



(a) Random. (b) Random cyclic. (c) Greedy. (d) Cyclic.

Figure 2: Plot of $\text{avg}(\text{time}/(m \min(d, 1/\varepsilon) \log(\kappa)/\varepsilon))$ as a function of n, using various variations with $\varepsilon = 10^{-4}$ and $m = 175$. Matrices are generated according to the framework presented in Section 2.5.1. Forall $n \in [\![20, 60]\!]$, we generate 10 matrices of size n and sparsity m.

We can assess the convergence depicted in Figure 2 by comparing it to the outcomes presented in Section 2.4. As outlined in Theorem 2.3, our anticipation was a linear

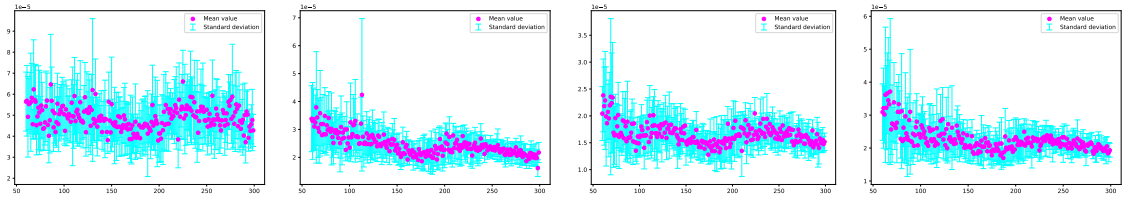
3 Note that here κ is not fixed.

progression concerning n in Figure 2b and Figure 2d. Remarkably, this linear trend is observed, even though it was originally expected only in terms of expectation. Referring to Theorem 2.1, an evolution in $\mathcal{O}(n^2/m)$ is expected in Figure 2c, and this prediction aligns with the observed outcomes. These numerical results are indeed promising.

The final validation involves examining Theorem 2.2, suggesting that Figure 2a should remain constant. Contrary to this expectation, the figure does not exhibit constancy. However, the growth rate is noticeably lower compared to the other variants, and there appears to be a tendency for the curve to plateau as n increases which is encouraging.

2.5.3 Complexity as a function of m

In order to experimentally show the convergence speed of theorems presented in Section 2.4, we fixed the parameters n and ε and let m in $[\![60, 300]\!]$. Each matrix is generated using an exponential distribution according to the scheme[4] presented in Section 2.5.1. The Figure 3 displays the results.



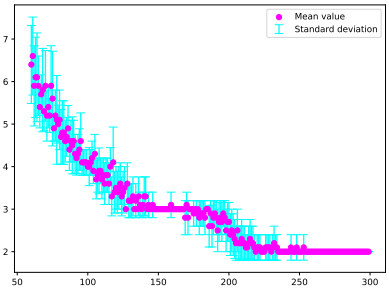
(a) Random. (b) Random cyclic. (c) Greedy. (d) Cyclic.

Figure 3: Plot of $\text{avg}(\text{time}/(\min(d, 1/\varepsilon)\log(\kappa)/\varepsilon))$ as a function of m, using various variations with $\varepsilon = 10^{-2}$ and $n = 60$. Matrices are generated according to the framework presented in Section 2.5.1. Forall $m \in [\![60, 300]\!]$, we generate 10 matrices of size n and sparsity m.

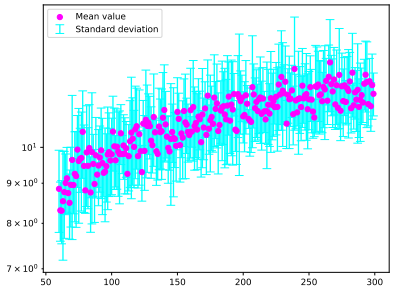
Interpreting the plots in Figure 3 using the theorems outlined in Section 2.4 poses challenges due to intrinsic connections between certain variables, such as d and m in the results. The expectation, based on Theorem 2.1, is that the plot using the greedy Osborn method should remain flat, indicating no dependence on m. However, contrary to this theoretical expectation, the observed plot does not exhibit this characteristic. A more nuanced understanding is provided by Figure 4, where the relationship between Figure 3c and Figure 4a suggests non-monotonic behavior.

Specifically, as Figure 4a decreases, Figure 3c increases, and vice versa, elucidating the non-monotonous trend observed. Additionally, insights from Figure 4b contribute to an understanding of why the scaled time is decreasing overall for the greedy variant. Similar principles apply to the interpretation of the other plots. Notably, the random variant appears to be less affected by overall trends, suggesting a higher dependency on m.

4 Note that here κ is not fixed.



(a) Evolution of d.



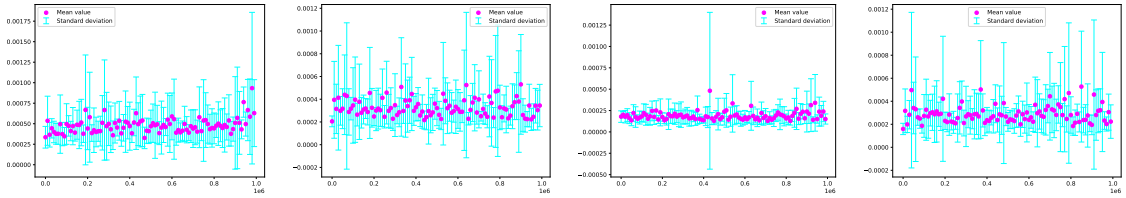
(b) Evolution of $\log \kappa$.

Figure 4: Plots of $\text{avg}(d)$ and $\text{avg}(\log \kappa)$ as a function of m, using $n = 60$ and the framework presented in Section 2.5.1. Forall $m \in [\![60, 300]\!]$, we generate 10 matrices of size n and sparsity m.

Unfortunately, the observed plots do not align with our expectations. They fail to exhibit a clear linear dependency for the three variants, contrary to the anticipated linear relationship.

2.5.4 Complexity as a function of κ

Finally, same experiments were ran to showcase the linear dependency in $\log \kappa$ in theorems outlined in Section 2.4. The result, provided in Figure 5 clearly fails to showcase any dependency in κ.



(a) Random. (b) Random cyclic. (c) Greedy. (d) Cyclic.

Figure 5: Plot of $\text{avg}(\text{time}/(\min(d, 1/\varepsilon)/\varepsilon))$ as a function of κ, using various variations with $\varepsilon = 10^{-2}$, $n = 10$ and $m = 30$. Matrices are generated according to the framework presented in Section 2.5.1. Forall $\kappa \in [\![40, 10^6]\!]$, we generate 10 matrices of size n and sparsity m.

2.6 Connections between Osborn's and Sinkhorn's algorithms

A connection with Sinkhorn's Algorithm must be drawn. Those two algorithms compute a scaling of a matrix. The main difference is Osborn returns a scaling of the form DAD^{-1} whereas Sinkhorn returns a scaling of the form XAY with X and Y diagonals.

These two algorithms minimize a distance between two measures. For the first one, it is Kullback-Leibler and for the second one it is Hellinger.

Sinkhorn is an optimal transport algorithm, as we have seen in the course. It computes a solution to

$$\min_{P \in U(a,b)} \langle P, C \rangle + \varepsilon \text{KL}(P | \mathbb{1}_{n \times m})$$

using notations from the course. Unlike with Osborn's algorithm, we do not need to work with square matrices.

Osborn's algorithm does not compute an optimal transport. We tried to modify it to be able to use it to compute optimal transport solutions. However, we failed to do so. It still has some advantages : its scaling has a form that preserves eigenspaces.

We still wanted to compare those algorithms and the only way we found is to use that Sinkhorn's algorithm can be used as a balancing algorithm: we just have to use the same distribution as input and output for it. This comparaison is not fair because Sinkhorn's additional has more constrains as we impose the value of the sum of the rows and columns and also because it is more versatile in general.

Those two algorithms are really similar. Hence, we will use the number of iteration to compare them. We are using an implementation of Sinkhorn's algorithm from numerical tours of the course [Pey11] and use the stopping criteria of Osborn's algorithm:

$$\frac{||c(A) - r(A)||_k}{\sum_{i,j} a_{i,j}} \leq \varepsilon.$$

This criterion is justified for Sinkhorn's algorithm too because we are running it with the same distribution as input and output.

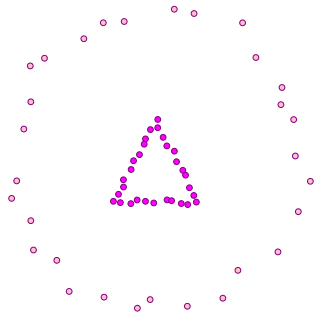
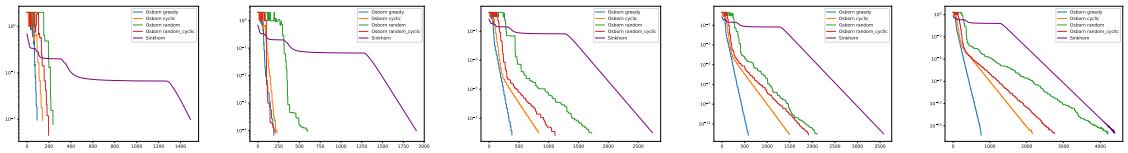


Figure 6: Data used for our experiments on entropic regularization. The outer set is generated using 30 regularly placed points on a circle of radius one centred at the origine. We added a Gaussian noise on each point with a scale of 6.10^{-2}. The inner set is generated by placing regularly 10 points on each edges of an equilateral triangle with edges of length $\sqrt{3}/3$. We added a Gaussian noise on each point with a scale of 10^{-2}.

In order to generate the matrices on which we will use the algorithms, we decided to use kernels from entropic regularization in optimal transport. We took inspiration from the numerical tours in the course [Pey11]. Given a ground cost matrix C, the kernel K is defined by

$$\forall i, j \quad K_{i,j} = \exp\left(-\frac{C_{i,j}}{\varepsilon_K}\right).$$

We decided to use such kernels because it allows us to test our algorithms on real use cases while being able change parameters such as C and ε_K in order to get kernels with various properties.



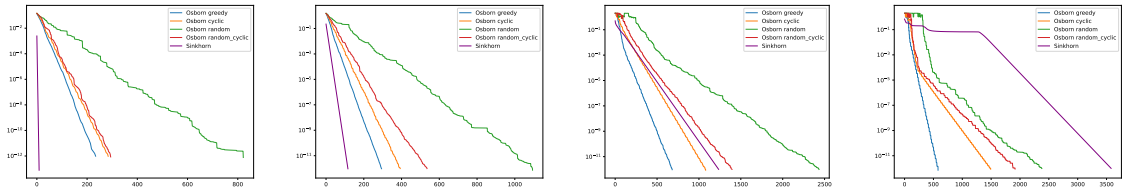
(a) $\varepsilon = 10^{-2}$ (b) $\varepsilon = 10^{-5}$ (c) $\varepsilon = 10^{-8}$ (d) $\varepsilon = 10^{-12}$ (e) $\varepsilon = 10^{-16}$

Figure 7: Convergence Analysis of Osborn's and Sinkhorn's Algorithms in the Regular Entropic Regularization Setting. The analysis utilizes data from Figure 6 with a regularization parameter ε_K set to 10^{-3}.

The Figure 7 shows a first application of the algorithms, including variants of Osborn's algorithm. We can see that all variants tends to have the same behaviors. This will be confirmed by the following experiments. However, Sinkohorn's algorithm tends to have a different comportment, it looks more continuous.

2.6.1 Using different regularization parameters

The regularization parameter ε_K has a high impact on the final result. However, this is not what we are looking for here. We are interested in how its impact on K impacts the convergence of the algorithms. A smaller value ok ε_K means that C/ε_K will have larger values. The matrix C has non-negative values, hence smaller values of the regularization parameter will reduce the gap between values in K.



(a) $\varepsilon_K = 10^{0}$ (b) $\varepsilon_K = 10^{-1}$ (c) $\varepsilon_K = 10^{-2}$ (d) $\varepsilon_K = 10^{-3}$

Figure 8: Convergence Analysis of Osborn's and Sinkhorn's Algorithms with Varied Regularization Parameters. The study explores the convergence behavior of both algorithms using different regularization parameters, leveraging data from Figure 6.

The illustration in Figure 8 provides insights into how the spacing of values in the kernel influences the effectiveness of the algorithms. Notably, when all values in the kernel are closely distributed, Sinkhorn's algorithm gains an advantage, while a larger gap between values tends to pose challenges for its performance. Variants of Osborn's algorithm exhibit consistent behavior across multiple runs. As anticipated, the greedy variant showcases superior per-iteration improvement. Interestingly, despite the distinct characteristics and varying performance of these algorithms, they demonstrate a similar asymptotic convergence rate, a finding that brings about a degree of surprise.

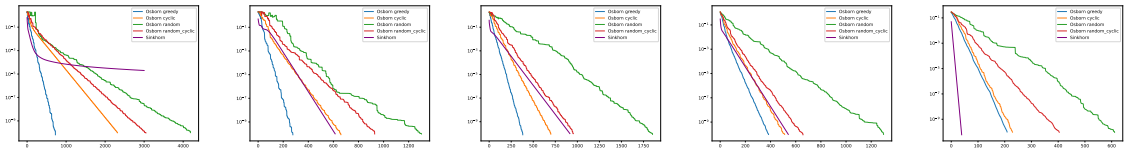
2.6.2 Using different Minkowski distances

The regular distance matrix is constructed based on the square of the Euclidean distance, equivalent to the square of the 2-Minkowski distance. To investigate the impact of altering this parameter, we decided to explore how the algorithms respond to variations

in this parameter. The cost matrix is then defined, for a parameter $p \geq 0$, as follows:

$$\forall i, j \quad C_{i,j} = \sum_k |x_{i,k} - y_{i,k}|^p .$$

Intuitively, when p is in the range $]0, 1]$, values close to zero become more pronounced, leading to a reduction in the overall scale. On the other hand, larger values of p cause values in $[0, 1[$ to approach zero, and those higher than 1 to become considerably larger.



(a) $p = 0.5$ (b) $p = 1$ (c) $p = 5$ (d) $p = 10$ (e) $p = 50$

Figure 9: Convergence Analysis of Osborn's and Sinkhorn's Algorithms with Varied Minkowski distances and $\varepsilon_K = 10^{-2}$. The investigation explores how different parameters of a Minkowski distance impact the convergence behavior of both algorithms, utilizing data from Figure 6.

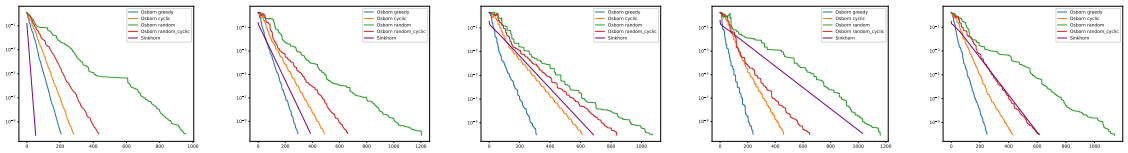
As p increases, both algorithms exhibit a trend where fewer iterations are needed. All variants of Osborn's algorithm respond uniformly to this trend and show resilience without apparent failures. However, Sinkhorn's algorithm encounters challenges, particularly when p decreases, and it faces convergence issues, being unable to converge for $p = 0.5$. This observation reinforces the notion that Sinkhorn tends to underperform in scenarios characterized by numerous small values.

2.6.3 Using different powers of a given distance

The experiment detailed in Section 2.6.2 presented intriguing findings. However, it's essential to acknowledge that when p changes, the matrix's structure itself may undergo substantial alterations due to its impact on the differences between every pair of coordinates. To mitigate this, we opted for a different approach. We selected a specific cost—namely, the squared Euclidean distance—and applied an element-wise power to it. Subsequently, for all parameters k, we defined the cost matrix as follows:

$$\forall i, j \quad C_{i,j} = \left(\sum_l (x_{i,l} - y_{j,l})^2 \right)^k .$$

The parameter k introduces an element-wise scaling, thereby influencing the previously introduced parameter κ. This adjustment allows us to investigate the impact of element-wise scaling on the algorithms' behavior, providing additional insights into their sensitivity to different cost structures.



(a) $k = 0.1$ (b) $k = 0.5$ (c) $k = 1$ (d) $k = 5$ (e) $k = 20$

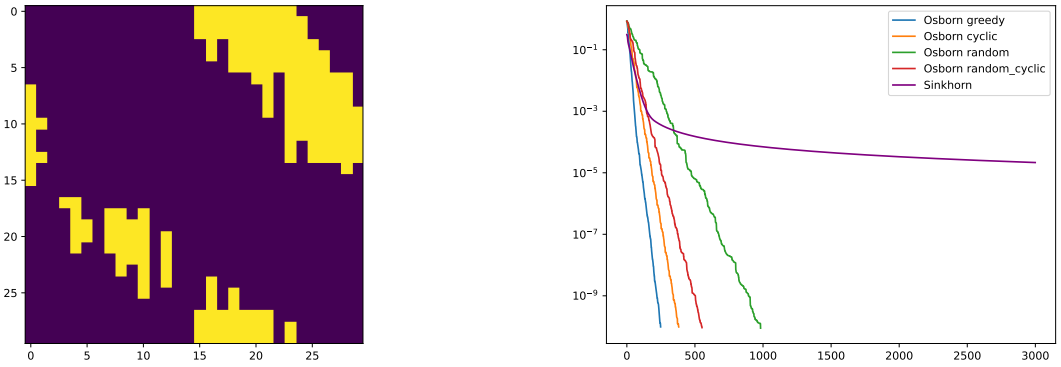
Figure 10: Convergence Analysis of Osborn's and Sinkhorn's Algorithms with Varied Distance Powers and $\varepsilon_K = 10^{-2}$. The investigation explores how different powers of a distance impact the convergence behavior of both algorithms, utilizing data from Figure 6.

The observed results in Figure 10 indicate a preference for smaller values of k by all algorithms, contributing to the compression of costs. This preference aligns with the characteristic that all Euclidean distances in our dataset are greater than one. Notably, Sinkhorn's algorithm is notably more affected by variations in k, highlighting its sensitivity to changes in the element-wise scaling parameter.

2.6.4 Using other distances

For our final experiment, we aimed to leverage the insights gathered from previous experiments to construct a cost matrix that would favor Osborn while challenging Sinkhorn, validating our intuitive understanding.

Firstly, we constructed a distance matrix using the square Euclidean norm and normalized it by its maximum value. This yielded a matrix with values in the range $]0, 1]$. Subsequently, values below 0.3 were set to zero. The thresholded values at 0 were then transformed: those set to 0 were mapped to 1 upon exponentiation, while other non-zero values became extremely small in comparison ($\leq 1^{-20}$). This deliberate manipulation resulted in an extremely ill-balanced matrix.



(a) The kernel K. (b) Convergence of the criterion.

Figure 11: Matrix Balancing with Osborn and Sinkhorn Algorithms. The kernel is derived from data in Figure 6. We construct a distance matrix D by computing the squared Euclidean norm and normalizing it by its maximum value. Subsequently, values below 0.3 are thresholded to 0. The kernel K is then defined as $K = \exp(-C/\varepsilon_K)$ with $\varepsilon_K = 10^{-2}$. It is important to note that K excludes any inputs equal to 0.

This experiment yielded successful outcomes, as evident in Figure 11. Osborn demonstrated convergence with a relatively small number of iterations, whereas Sinkhorn not

only struggled but failed to converge entirely. These results provide further support for our earlier intuitions and observations regarding the algorithms' sensitivity to specific cost matrix structures.

2.6.5 Conclusion

Exploring various regularization parameters, MINKOWSKI distances, powers, and other distances, we observed distinct behaviors. Notably, SINKHORN's sensitivity to parameter variations, especially in ill-balanced matrices, raises questions about its robustness in certain scenarios.

3 Conclusion and perspective

We presented matrix balancing and its practical applications, with a specific emphasis on introducing OSBORN's algorithm and its variants as outlined in this article. We provided a comprehensive explanation of the steps required for an efficient implementation, and the clean code is made publicly available on GitHub. However, it's worth noting that the original article lacked a detailed discussion on achieving efficiency, particularly in terms of temporal and spatial complexity. The absence of explicit data structures or supporting resources for their claims was a notable limitation.

The theoretical results, pivotal to the article, were presented and complemented with numerical experiments, yielding relative success in showcasing their convergence rate.

Moreover, we established a connection between OSBORN's and SINKHORN's algorithms, conducting a comparative analysis across various types of kernels for matrix balancing. Our findings illustrated instances where SINKHORN encounters challenges. Notably, we demonstrated that OSBORN's convergence remains consistent across variants, with a consistent ranking of the number of iterations required in every case. Leveraging these results, we successfully constructed a kernel where OSBORN excels while SINKHORN struggles and fails to converge.

4 Connexion with the course

In the context of this paper, several notions, results, and algorithms presented in the course are either used or related, although the connections may sometimes be nuanced due to the unique characteristics of OSBORN's algorithm. The exploration follows a chronological order:

- Kantorovich Relaxation:

  The concept of KANTOROVICH relaxation is foundational. Admissible couplings are required only to satisfy mass conservation. For two discrete mass distributions a and b, the set $U(a, b)$ is defined as:

  $$U(a,b) := \left\{ P \in \mathcal{M}_{n,m}(\mathbb{R}_+) \mid P\mathbf{1}_m = a \text{ and } P^\top \mathbf{1}_n = b \right\}.$$

  OSBORN's algorithm provides assignments in $U(a, a)$, although a is an output of the algorithm, and it will not sum to 1.

- Optimal Transport as a Distance:

  There is a link to the notion that optimal transport can define a distance. While Sinkhorn's algorithm is practically employed to define a distance [Cut13; Qia+16], and given its close relation to Osborn's algorithm, attempts were made to find analogous definitions. However, these attempts were unsuccessful.

- Sinkhorn's Algorithm:

  The connection to Sinkhorn's algorithm is evident. Both algorithms perform matrix scaling but with different objectives. The report extensively evaluates the connections between these two algorithms.

- Discrete Barycenters:

  The concept of discrete barycenters is linked to Osborn through a modified problem. Considering the optimization problem involving discrete barycenters, one could run Osborn on a modified problem inspired by the use of Sinkhorn for barycenters. This connection suggests potential applications of Osborn in barycenter-related scenarios.

- Unbalanced Optimal Transport:

  Unbalanced optimal transport is another area of connection. The notion of optimal transport defined without exact mass conservation serves as inspiration for the use of Osborn in the context of optimal transport problems. The application of Osborn in scenarios where mass conservation is not strictly enforced aligns with unbalanced optimal transport formulations.

References

[Alt22] Jason M Altschuler. "Transport and Beyond: Efficient Optimization over Probability Distributions". PhD thesis. Massachusetts Institute of Technology, 2022.

[AP22] Jason M Altschuler and Pablo A Parrilo. "Approximating Min-Mean-Cycle for low-diameter graphs in near-optimal time and memory". In: *SIAM Journal on Optimization* 32.3 (2022), pp. 1791–1816.

[AP23] Jason M Altschuler and Pablo A Parrilo. "Near-linear convergence of the Random Osborne algorithm for Matrix Balancing". In: *Mathematical Programming* 198.1 (2023), pp. 363–397.

[CD00] Tzu-Yi Chen and James W Demmel. "Balancing sparse matrices for computing eigenvalues". In: *Linear algebra and its applications* 309.1-3 (2000), pp. 261–287.

[Che01] Tzu-Yi Chen. *Preconditioning sparse matrices for computing eigenvalues and solving linear systems of equations*. University of California, Berkeley, 2001.

[Cut13] Marco Cuturi. "Sinkhorn Distances: Lightspeed Computation of Optimal Transport". In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper_files/paper/2013/file/af21d0c97db2e27e13572cbf59eb343d-Paper.pdf.

[Eav+85] B Curtis Eaves et al. "Line-sum-symmetric scalings of square nonnegative matrices". In: *Mathematical Programming Essays in Honor of George B. Dantzig Part II* (1985), pp. 124–141.

[JLL14] Rodney James, Julien Langou and Bradley R Lowery. "On matrix balancing and eigenvector computation". In: *arXiv preprint arXiv:1401.5766* (2014).

[KK96] Bahman Kalantari and Leonid Khachiyan. "On the complexity of nonnegative-matrix scaling". In: *Linear Algebra and its applications* 240 (1996), pp. 87–103.

[KKS97] Bahman Kalantari, Leonid Khachiyan and Ali Shokoufandeh. "On the complexity of matrix balancing". In: *SIAM Journal on Matrix Analysis and Applications* 18.2 (1997), pp. 450–463.

[LS81] B Lamond and Neil F Stewart. "Bregman's balancing method". In: *Transportation Research Part B: Methodological* 15.4 (1981), pp. 239–248.

[Osb60] EE Osborne. "On pre-conditioning of matrices". In: *Journal of the ACM (JACM)* 7.4 (1960), pp. 338–345.

[Pey11] Gabriel Peyré. "The numerical tours of signal processing-advanced computational signal and image processing". In: *IEEE Computing in Science and Engineering* 13.4 (2011), pp. 94–97.

[PR71] Beresford N Parlett and Christian Reinsch. "Balancing a matrix for calculation of eigenvalues and eigenvectors". In: *Handbook for Automatic Computation* 2 (1971), pp. 315–326.

[Qia+16] Wei Qian et al. "Non-Negative Matrix Factorization with Sinkhorn Distance." In: *IJCAI*. 2016, pp. 1960–1966.

[SK67] Richard Sinkhorn and Paul Knopp. "Concerning nonnegative matrices and doubly stochastic matrices". In: *Pacific Journal of Mathematics* 21.2 (1967), pp. 343–348.

[SNT17] Mahito Sugiyama, Hiroyuki Nakahara and Koji Tsuda. "Tensor balancing on statistical manifold". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3270–3279.

[Sou91] George W Soules. "The rate of convergence of Sinkhorn balancing". In: *Linear algebra and its applications* 150 (1991), pp. 3–40.

[SZ90] Michael H Schneider and Stavros A Zenios. "A comparative study of algorithms for matrix balancing". In: *Operations research* 38.3 (1990), pp. 439–455.

[Wat06] David S Watkins. "A case where balancing is harmful". In: *Electron. Trans. Numer. Anal* 23 (2006), pp. 1–4.