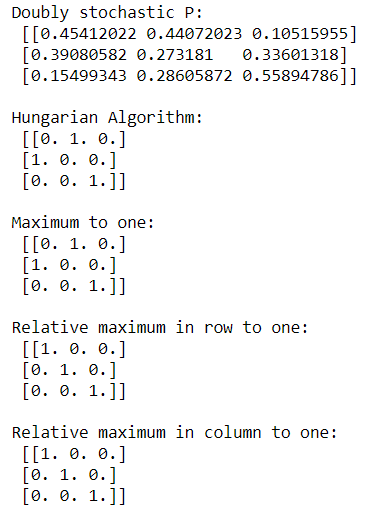
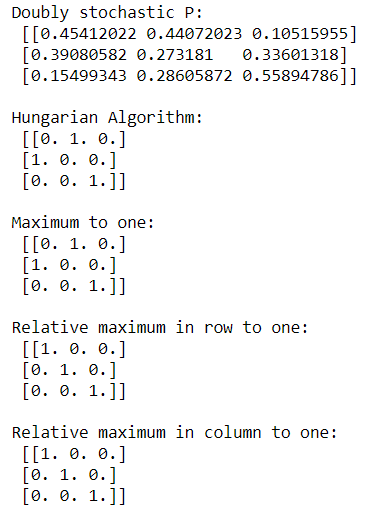
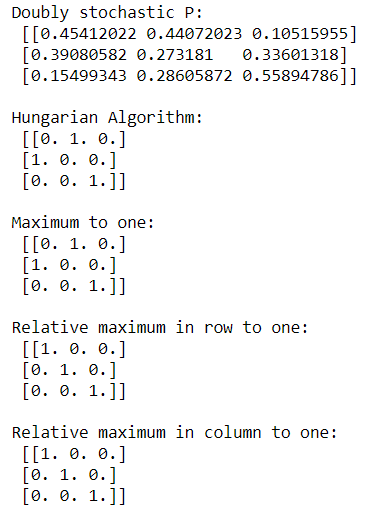
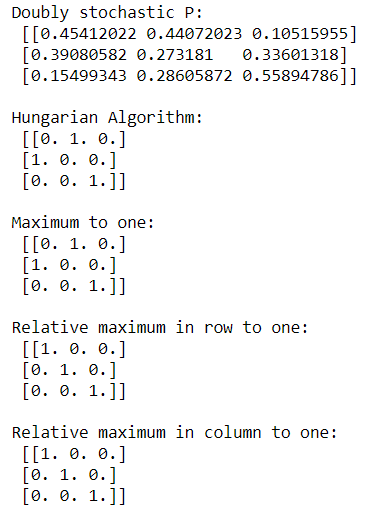
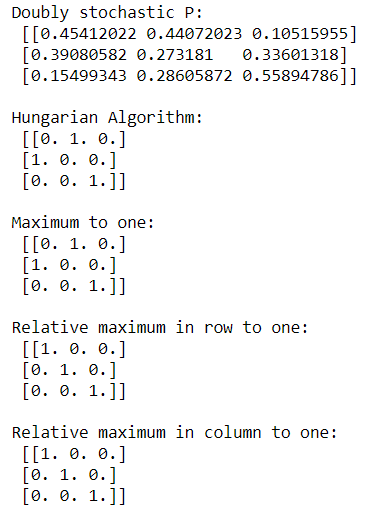
Matrix Completion, Gradient Descent, Regularization, University of Chicago.

# Prep Meeting 6

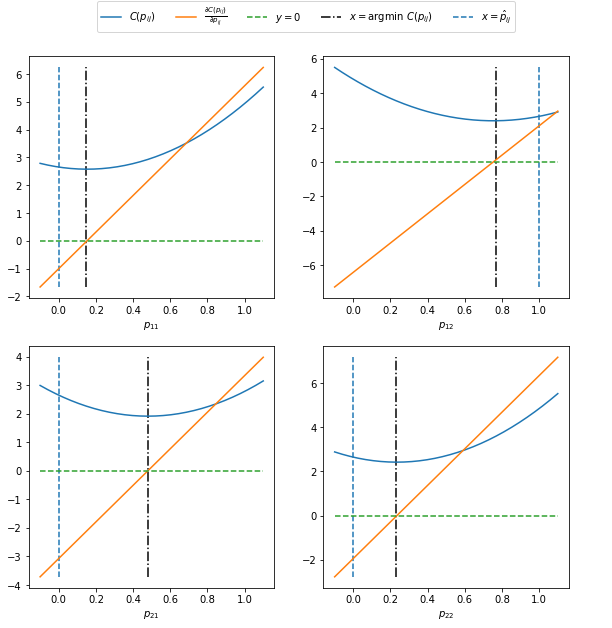
**Investigate projecting P\_DS onto “best” P\_perm**

Solution found: Hungarian Algorithm. Also used in the [literature](http://bioinfo.ict.ac.cn/~dbu/AlgorithmCourses/Lectures/Lec10-Matching-HungarianMethod-Variants.pdf). The question is finding the permutation j\_1, …, j\_n of the integers i = 1, ..., n that maximizes the sum a\_1j\_1 + a\_2j2 + … + a\_njn. In other words, you need to find one entry per row and column such that the sum of all entries is maximized. The entries that are chosen to represent the ones that will be a *1* in the permutation matrix. It is the permutation matrix closest to the doubly stochastic matrix.

Three other methods have been developed as well which yield similar results.

*Question:* I could not find literature on using sampling and SGD for finding the permutation matrix closest to the DS one.

**Investigate using the projected P.**

Solution found: Issues with gradient descent, as the cost function is not continuous and not differentiable everywhere.

Actual P:

[[0. 1. 0.]

[0. 0. 1.]

[1. 0. 0.]]

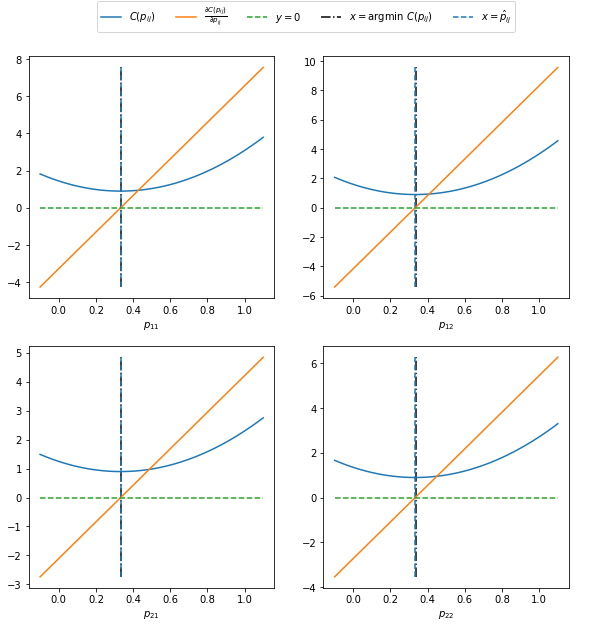
First iteration:

We deviate from original.

Even when we start with

Optimal P, we steer

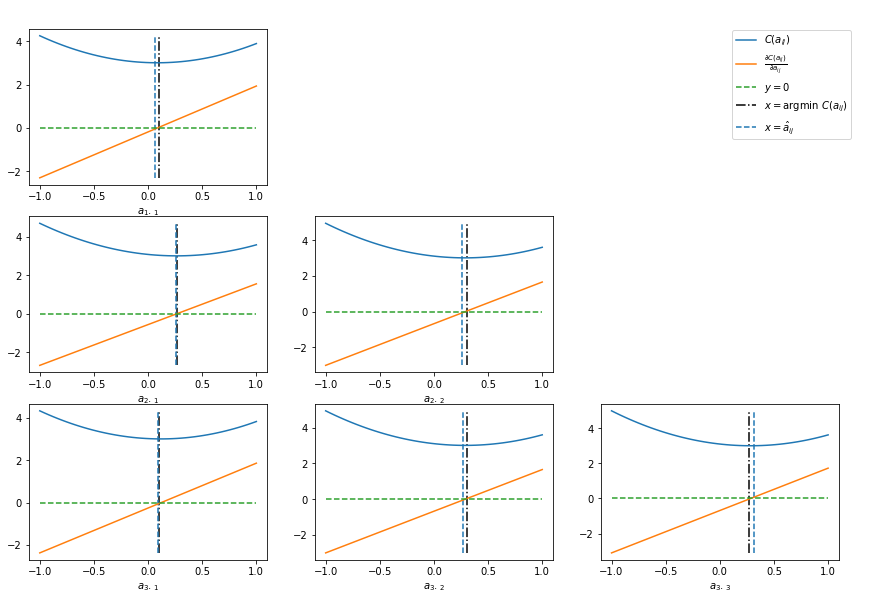
Towards 1/n.

“Convergence”.

We reach a cost of ~0.9, which is much smaller than the “optimal” of n \* sigma. This indicates that something fishy is going on. Nevertheless, the gradient descent algorithm performs very well.

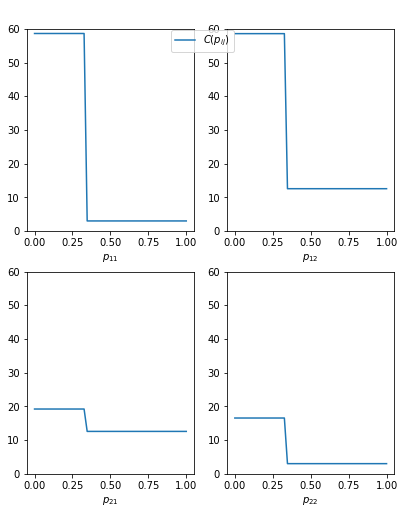
*Using projection of P*.

Derivatives with respect to A do not change.



Cost with respect to p does change.

Double projection



Although all four show the exact same behavior (it is better to have p\_ij = 1 rather than p\_in). We see that for p22, it has the lowest cost value, indicating that p22 should “definitely be 1”. Furthermore, we see that p11 has a lower cost for high p than p12, indicating that p11 should be 1 before p12 should be 1.

**New approach: X\_t = P-1 A P X\_t-1**

Works the best as of now. Linear constraints: all 0 < p\_ij < 1. Rows must sum to one. Columns must sum to one.

Does find, however, a local minimum sometime. If it finds the correct solution, then the F norm between the P\_DS and the P\_perm is very small.

Approach: Run the algorithm *n* times on a random input, and pick the output with the smallest difference between P\_DS and P\_perm, where “smallest difference” means the smallest F\_norm. Alternative, pick the P\_perm such that the average F\_norm is the smallest.

Some small results:

Actual A:

[[0.417022 0. 0. ]

[0.30233257 0.14675589 0. ]

[0.18626021 0.34556073 0.39676747]]

Actual P:

[[0. 0. 1.]

[0. 1. 0.]

[1. 0. 0.]]

Outcome: 7 / 10 times, the correct matrix was retrieved, with an average F-norm of ~0.7 (min: 0.56, max: 1.24). Three times 1 and 2 were switched (which sorta makes sense, as the 0.15 is not *that* high, which achieved an average score of 1.25 (min: 1.24, max: 1.27). This was for *500* samples.

When we increased *n* to *5000* from *500*, with the same A and P (and the same initializations), we get correct 6 / 10, average F-norm 0.6 (min: 0.23, max: 1.02). Three times 2 and 3 were switched, which is stranger but also since 0.18 is not *that* big, but increasing n does not seem to be that much better, but smaller F-score.

Example with different, clearer A:

Actual A:

[[0.4359949 0. 0. ]

[0.43532239 0.4203678 0. ]

[0.20464863 0.61927097 0.29965467]]

Actual P:

[[0. 0. 1.]

[0. 1. 0.]

[1. 0. 0.]]

N = 5000, Correct 6 / 10, with F score of 0.04, min: 0.0, max: 0.2. Incorrect 4 / 10 with 1 and 2 swapped, yields F-score of 1 (min: 0.93,max:1.07).

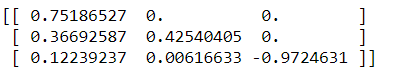
Other examples yielded similar results, ~60-70% of the time correct, but when correct, it indeed was a much lower F-score.

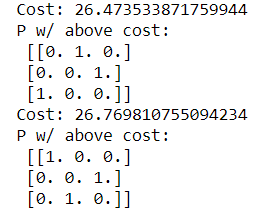
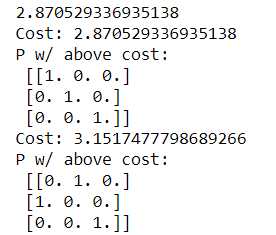
So, all in all, the method seems to work, but requires trying multiple starting points as there are local minima. I am also not that big of a fan of the linear constraints. Furthermore, requiring recomputing the inverse at every iteration of the optimization process can be slow, and indeed can give some numerical problems. Also, even for n = 3 it took some time, so I somehow doubt it will scale well for larger time series.

All yielded scores of around 3.0x, so in terms of cost, they are all equally optimal.

**Is the cost function of a different non fitting permutation matrix P significantly lower?**

Solution found: Often yes, provided we do not optimize A. There are different P with a similar cost function, but this is generally higher, and also often involves simply switching two rows or two colums, where also the corresponding weight in A was close to zero. Especially when A can also have negative values, the differences were clearly noticeable, but this was less when the matrix A had only nonnegative entries, of which some were close to zero.





Discrete stuff:

**MIP:** Found some papers using [MIP](https://www.jmlr.org/papers/volume18/17-033/17-033.pdf), but they are all quite complex and different than what we are doing unfortunately, but they seem to be used for the same problem; estimating a DAG from high dimensional data.

**Greedy DAG Search**: Found code with Greedy DAG Search.

**Joris Mooij**: Has other interesting approaches, but mostly combinatorial and tailored towards causal inference, rather than predictive inference.

**Anchor Regression:**

Seems to be mostly used for when training and test data distributions are shifts of each other. But it seems an interesting concept, a mixture of OLS, PA, and IV. The parameter for anchor regression, gamma, represents “how close” the estimator is to the other estimators.

Approaches:

* Sample from P\_DS to get P\_PERM. -> *No sampling, but Hungarian Algorithm.*
* Try permutation matrices P and see if the results make sense. *Results are a bit strange, as per the figure*.
* Check whether it makes sense for higher dimensions (n = 3, …). *The notation in principle makes sense, also with the projection, and with P^{-1} to the other side. Finding the right relaxation and constraints is difficult*.
* Rather than ||P\_DS X – A P\_DS X||, try ||P\_perm X – A P \_DS X||. Tried, optimizing is difficult due to discontinuities.
* *Anchor Regression, read, but not investigated in detail.*
* *Use 1 norm, tried quickly, but no severe differences.*