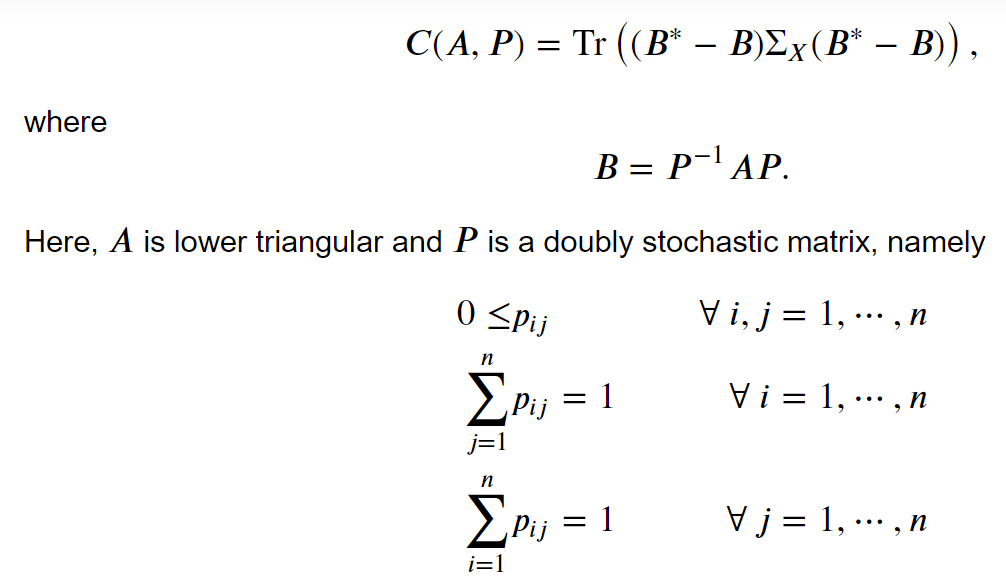
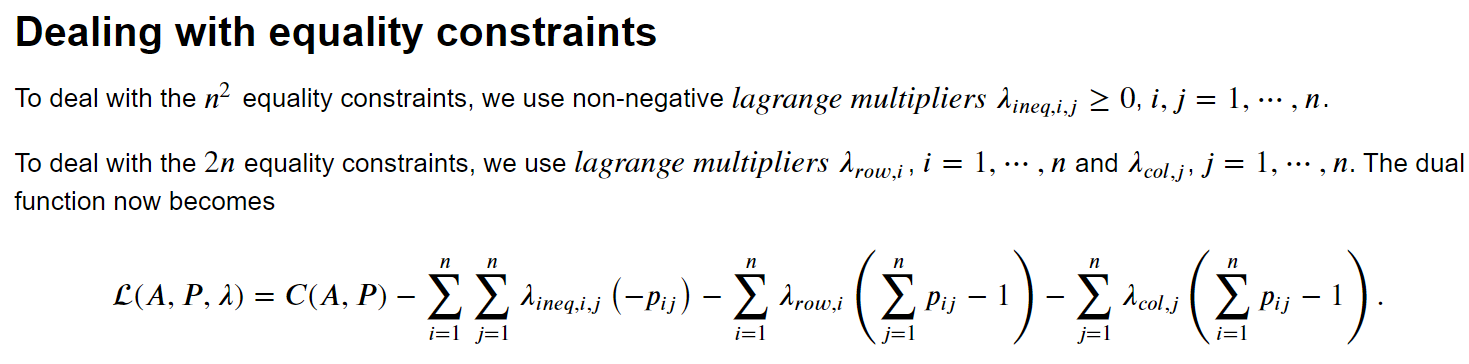
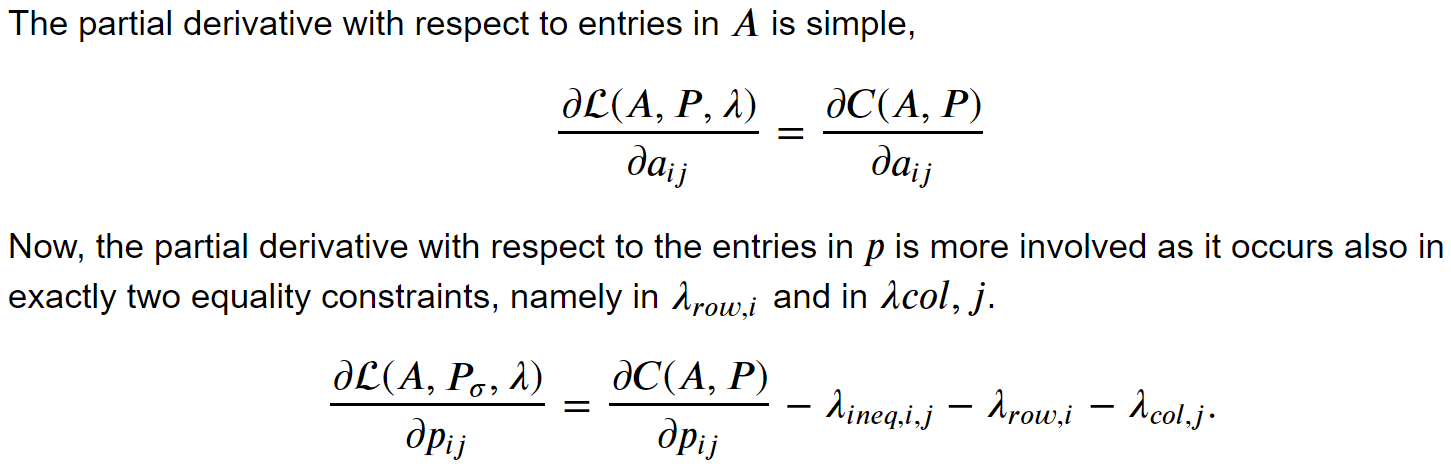
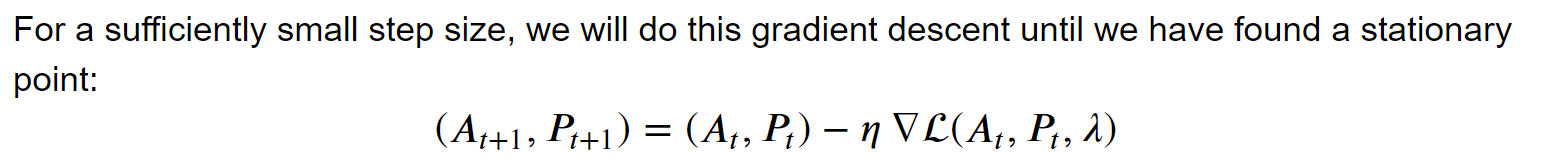
Meeting 10 Prep

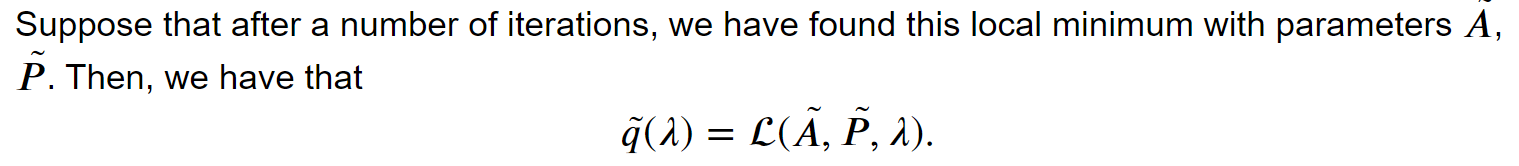
# Write out regular minimization with Lagrange multipliers

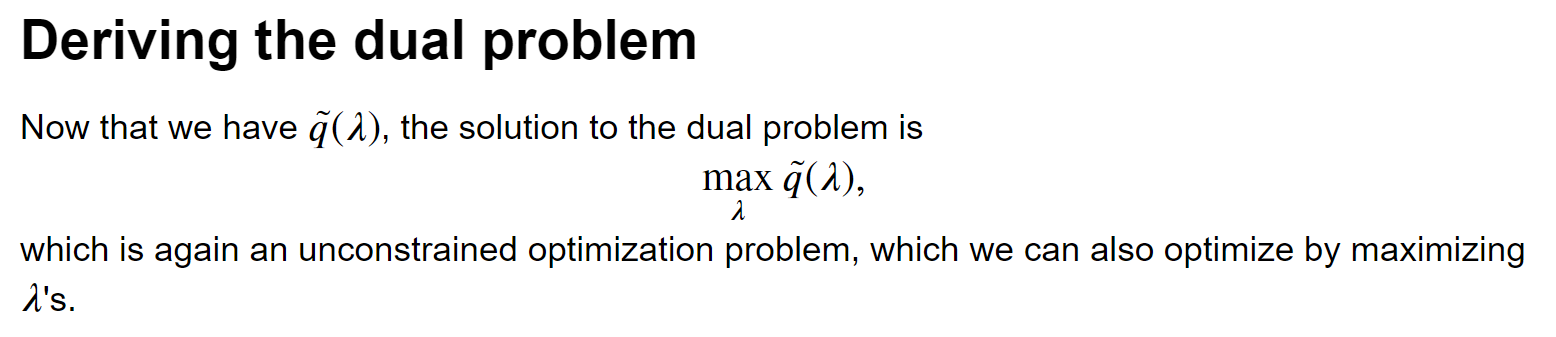








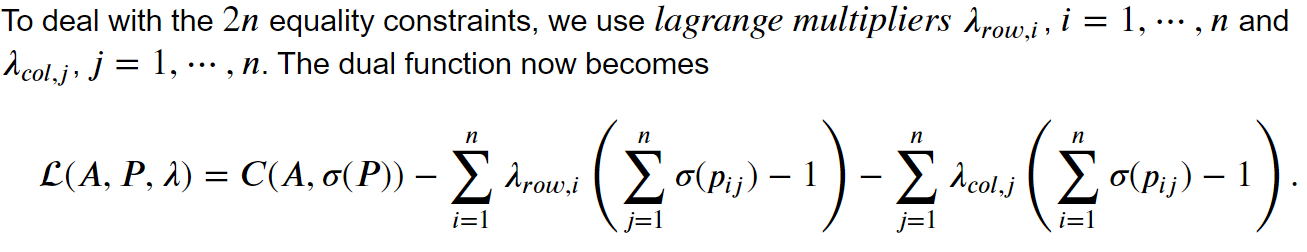


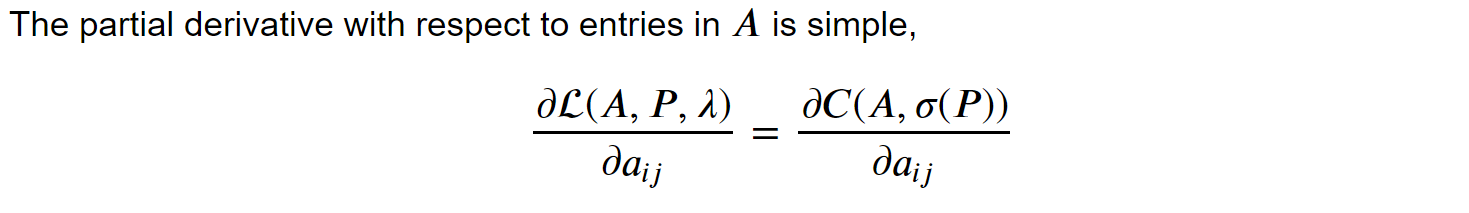


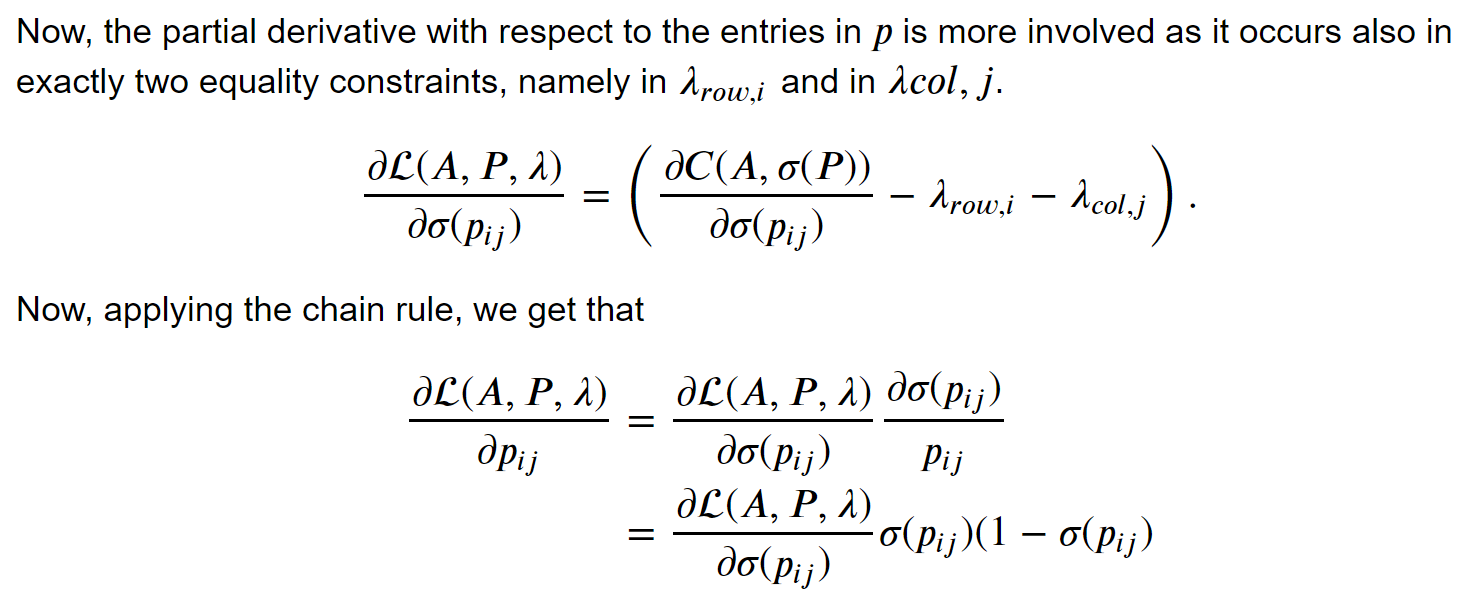
However, maximimizing this with respect to lambda yields issues. For example, if one p\_{ij} is negative, we can get +infinity by setting the corresponding lambda\_ineq\_ij to +infinity and all the others to zero. There are many different ways to get a function value of +infinity by changing the lambdas, and all the outcomes do not seem to make sense.

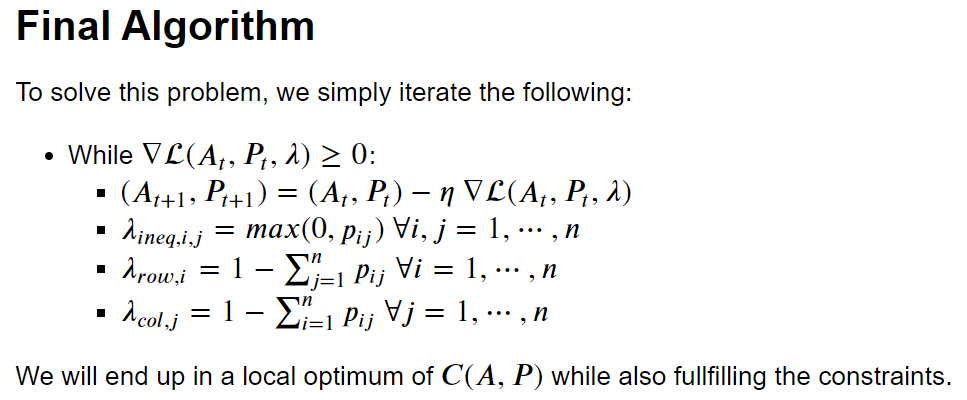
A solution is to do it iteratively. First, calculate the minimum with all lambdas zero. Then, set the values of lambda equal to the gradients. Then, minimize the cost function again with respect to A and P. In the end, we have reached a local optimum that adheres to the constraints. However, is this the correct way to go?

# Sigmoid Projection





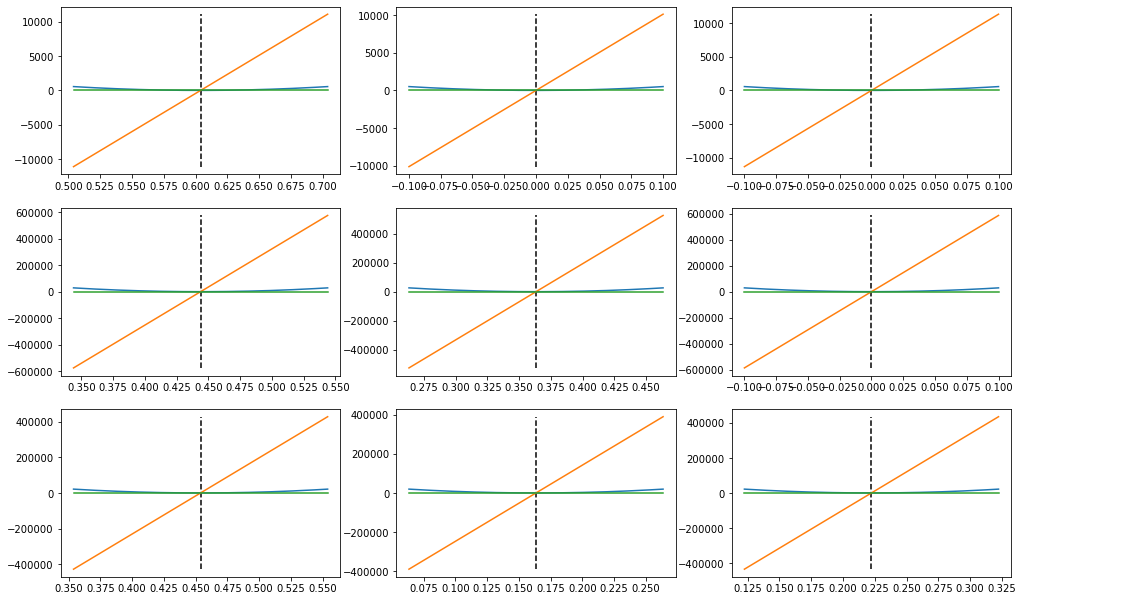


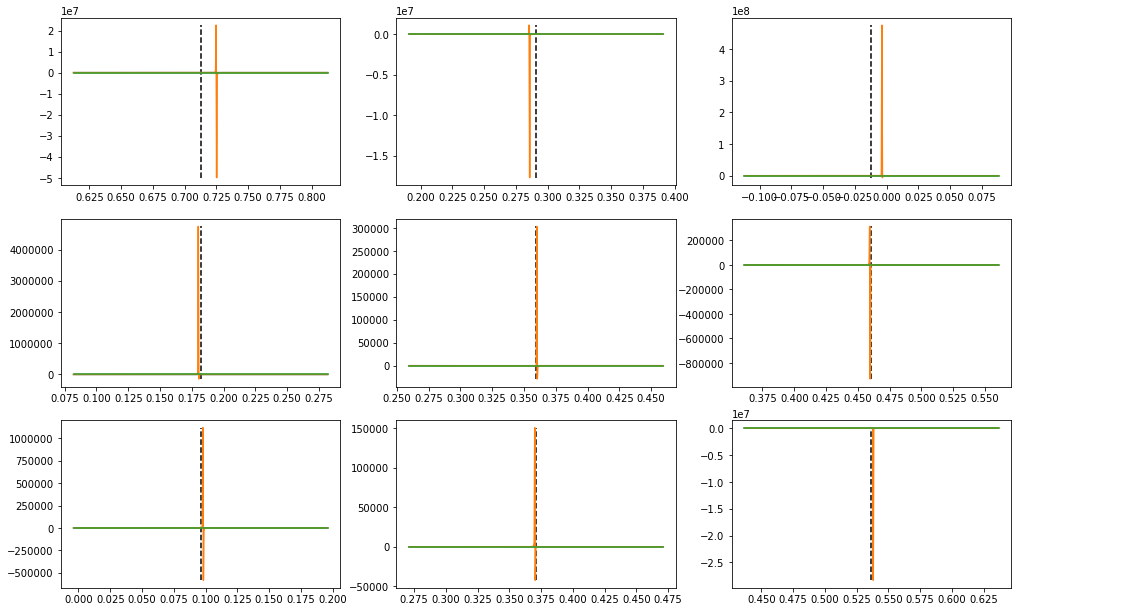


# Outcome

## Small step size required

Indeed, the outcome results in a local optimum while also adhering to the constraints, which is good! However, the step size needs to (sometimes) be extremely small to not over extend with one parameters which would result in a larger cost value rather than a smaller cost value. Hence, convergence takes extremely long, but the proof of concept works. However, an open question is still whether it will converge *exactly* to the global minimum.





## Plots of gradients

In the figures we see the issues. The plots for A are smooth, but we see that the gradient quickly becomes very large even for a very small change in the entries in A. We get in the order of thousands when we even change one 0.01, so we need incredibly small steps.

If we also looka t P, we see similar issues. Here, the gradient is much less nice, and we see that in a lot of scenarios, we are very close to the irregularity. These hills are impossible to climb over, and may cause large issues.

## Inspection of stationary point B\*

However, the algorithm often converges to a solution that is close to the global optimum, but not in the way we want it. The outcome B is extremely close to B\*. But the matrices A and P are far from A\* and P\*. Apparently, there are different ways, such as a doubly stochastic matrix P and a lower triangular matrix A such that B = B\*. So, the minimum is not achieved by a unique decomposition of B\* into a permutation matrix P\* and a lower triangular matrix A\*.

So, this is the major hurdle. The optimization works (albeit slow), but the outcome of the optimization is not what we want. It indeed gives A, P such that B = B\* (which is what we wanted initially), but the A and P are not of the constraint form that we wanted them to be (lower triangular yes, but permutation matrix no).

3D Example using Sigmoid:

Original: A = [[0.668 0 0], [0.507 0.397 0], [0.777 0.939 0.718]], P\* = I

Estimate: A = [[0.766 0 0], [0.615 0.596 0], [0.713 0.895 0.415]]

P = [[0.849 0.105 0.043], [0.098 0.688 0.216], [0.055 0.206 0.740]]

B = [[0.667 -0.001 -0.001], [0.506 0.396 0.000], [0.777 0.939 0.718]]

Diff. B, B\* = [[0.001, 0.001, 0.001], [0.000, 0.000, 0.000], [0.000, 0.000, 0.000]]

So, I think we can get arbitrarily close to B also when P is not a permutation matrix.

### Workarounds

**Penalty on Frobenius norm of P** So, we need a way to circumvent this. I was originally thinking of a penalty that favours permutation matrices over doubly stochastic matrices, such as the square of the frobenius norm (this was also used in the paper which relaxes to the Birkhoff polytope as well. However, the derivative of this with respect to the entries in P is equal to mu \* p\_{ij}, and this interplays with the lagrange inequality constraint lambda p\_{ij} as well.

# EM Algorithm

## Outcome

Have it implemented, and it seems to work well, albeit a little bit slow. The good thing is that it seems to be very certain, and we do not keep track of any permutation matrix P. However, we need to loop over a combinatorial search space, which is a major hurdle.

For example, for three dimensions, only 25 samples were required.

Also works for five dimensions, but it takes quite a long time (but code is also not optimized). However, it takes so much longer than simply iterating over all permutations possible and picking the permutation that achieves the smallest cost value.