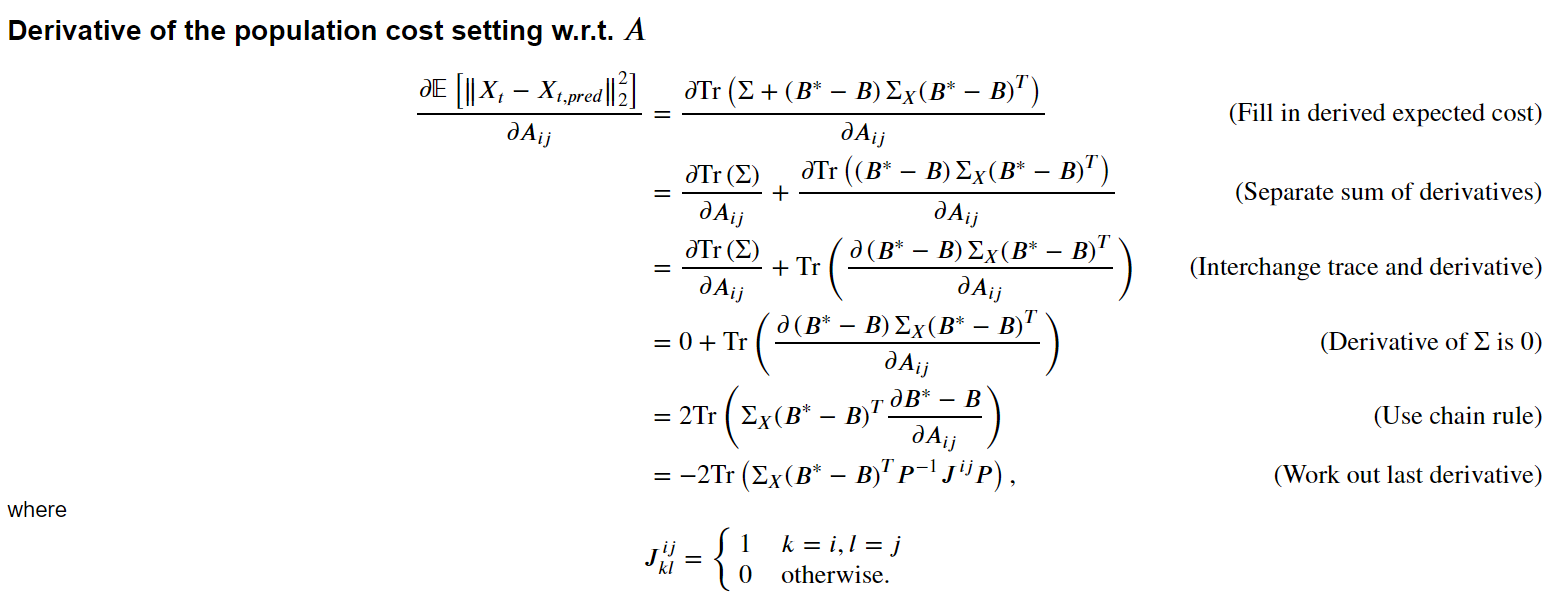
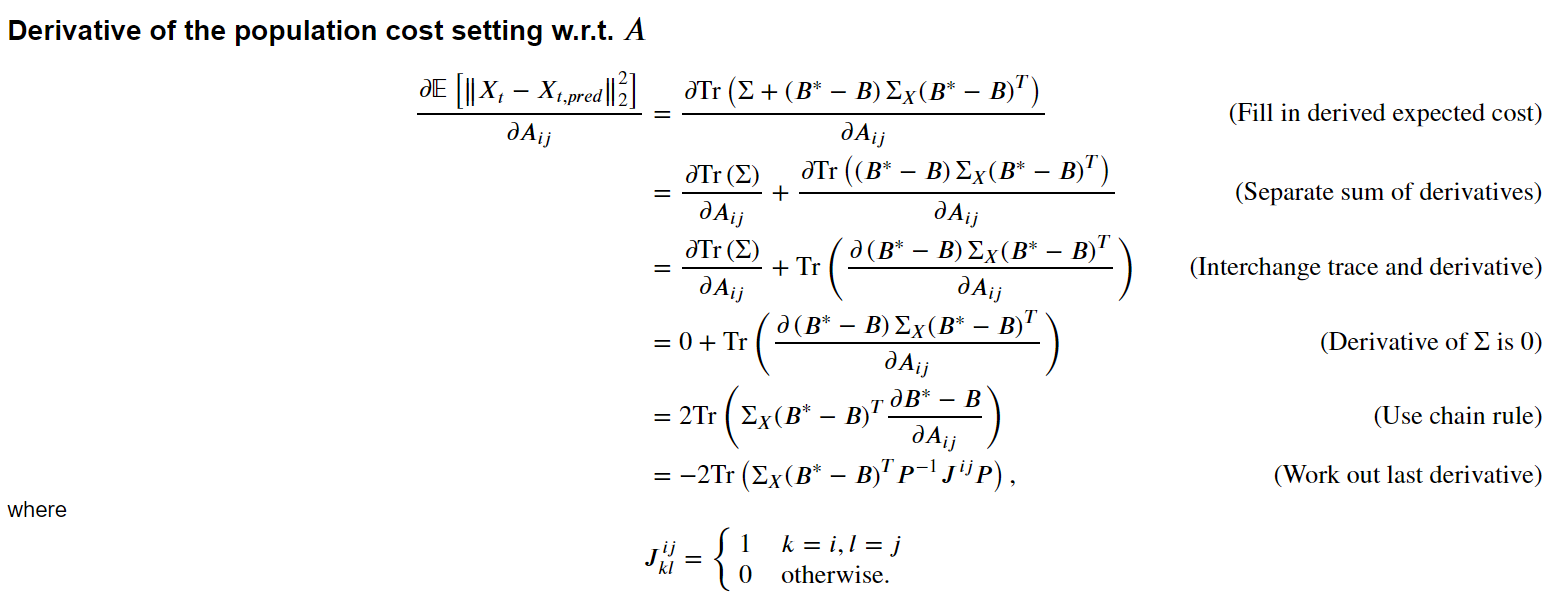
Prep Meeting 8

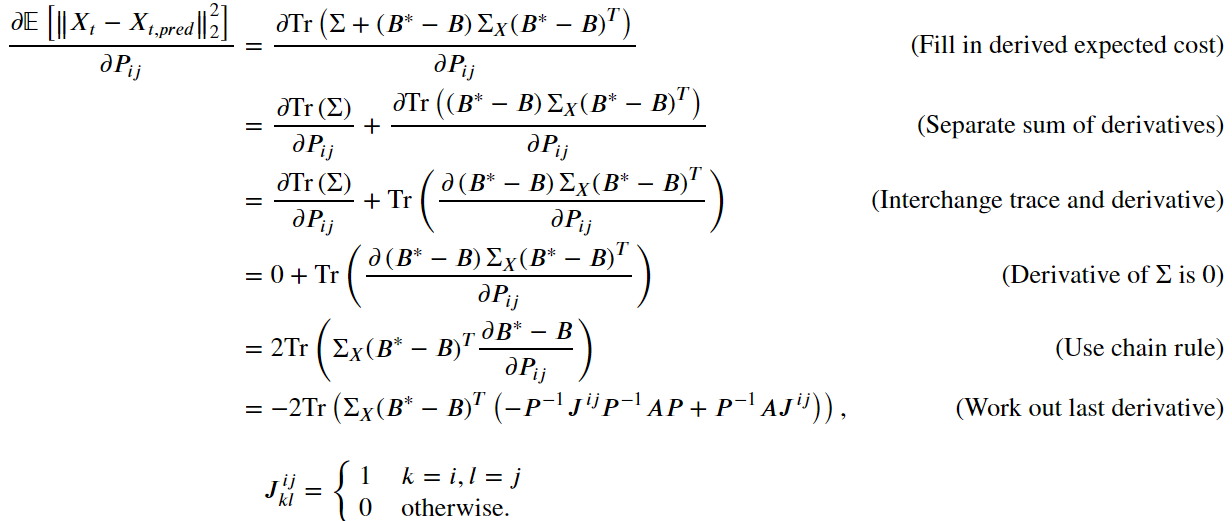
Recall: Cost function is



# Derivation of gradient in population setting

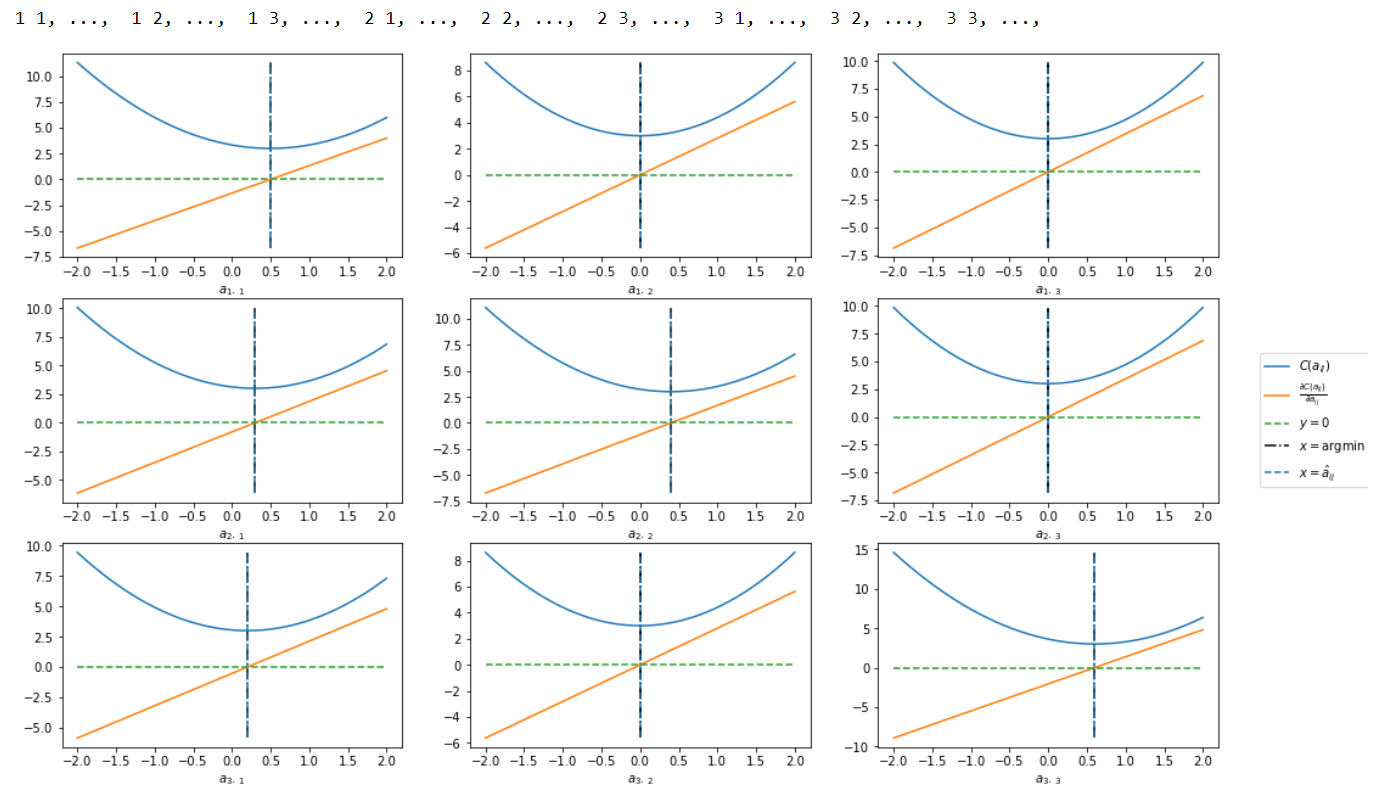


**Derivative of the population cost setting w.r.t. P**

****

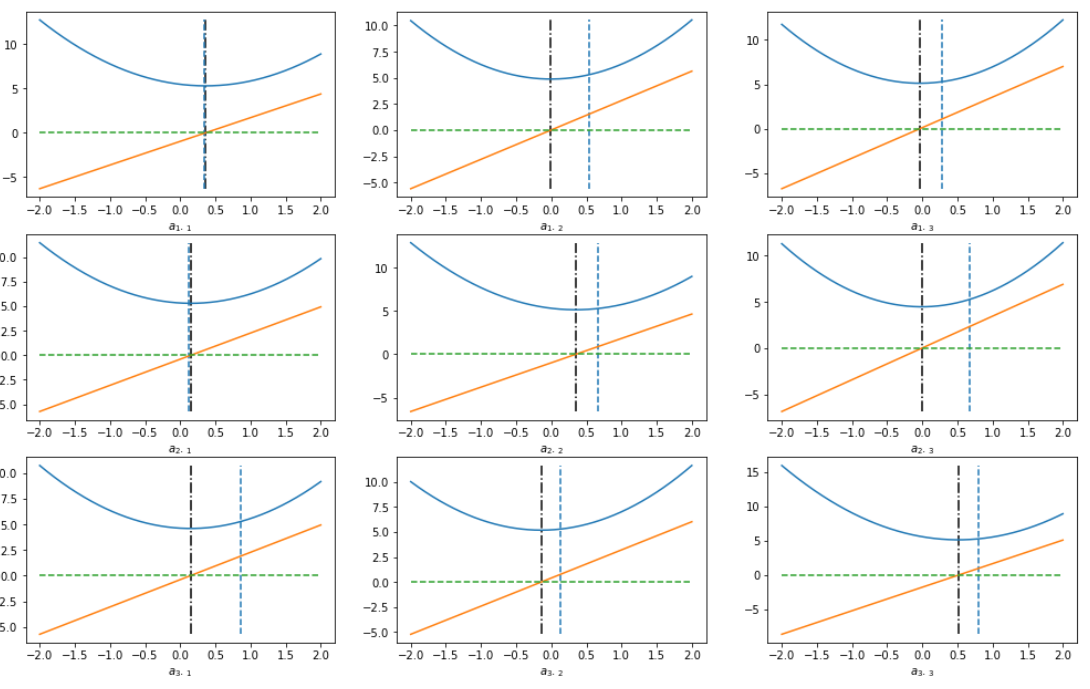
**Verification**

Verification of gradient w.r.t. A when A = A\*.

****

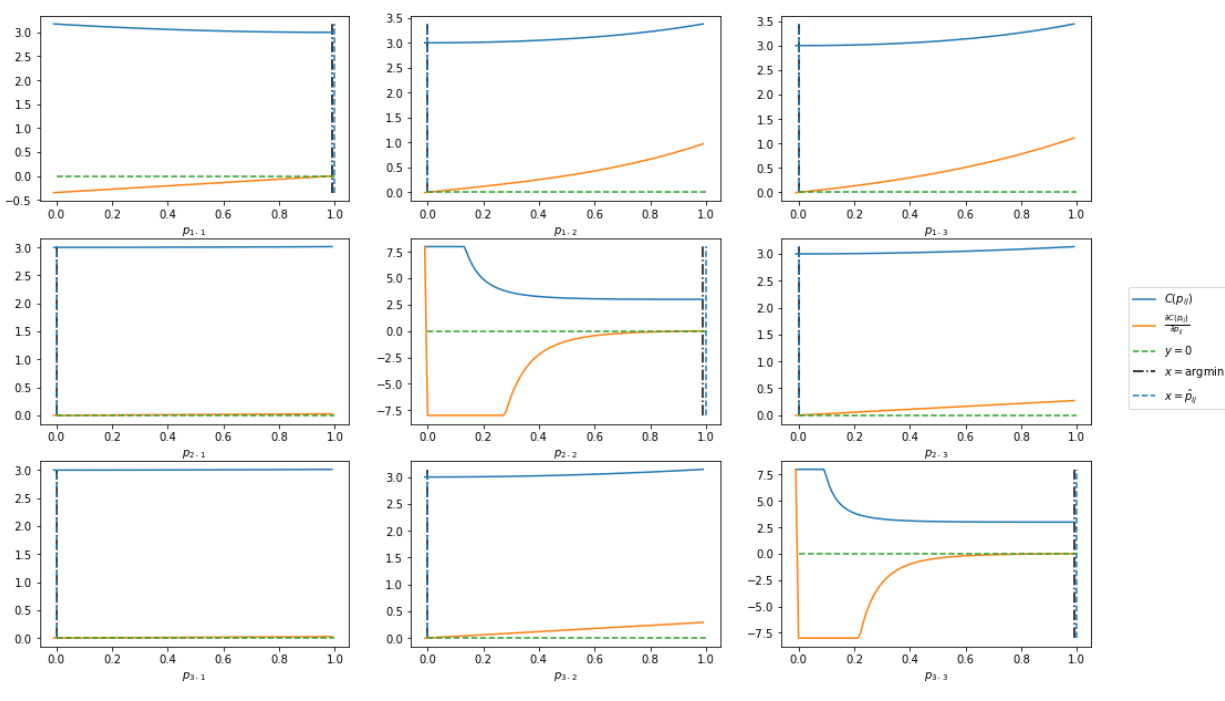
Conclusion: Gradient correctly finds stationary point. Furthermore, we see that we indeed attain a global optimum w.r.t. A for this given P.

**Verification for gradient w.r.t. A when A is randomly initialized**

****

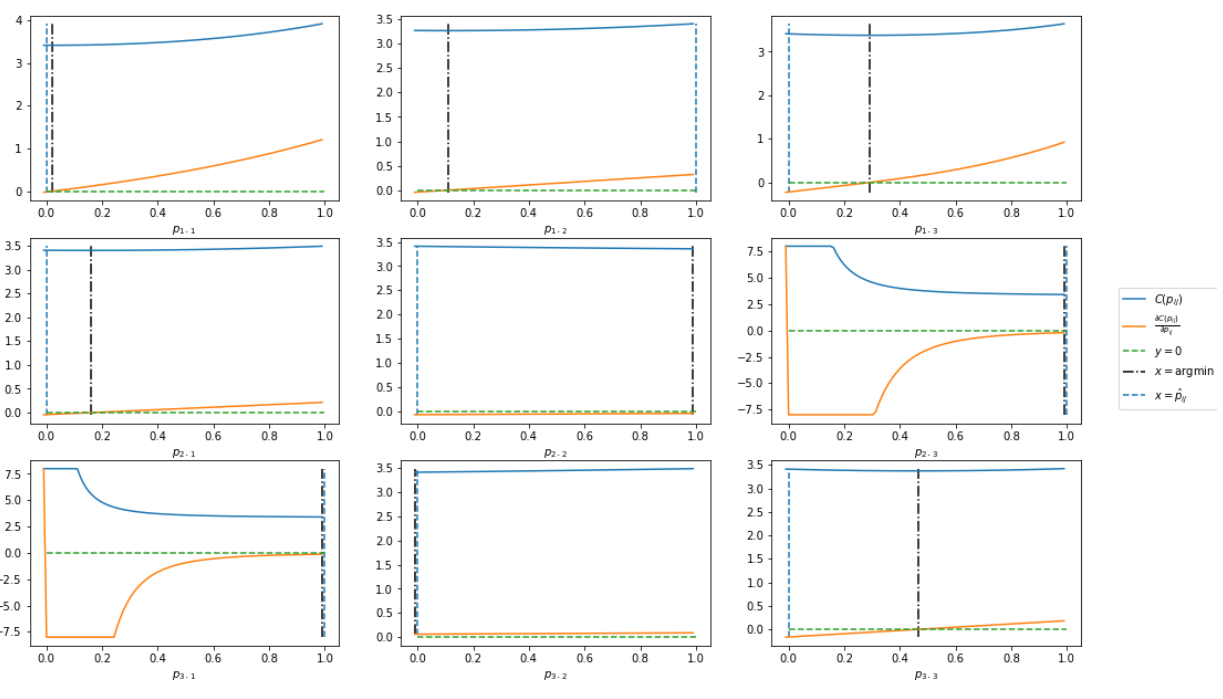
Conclusion: even for a random A, we see that the local optima for each aij seem to coincide wuite nicely with the actual A\*, e.g. the upper triangular part has a minimum around zero. However, one “iteration” does not find A\* immediately, but after say two or three, we have found the global optimum.

Verification of gradient w.r.t. P when P = P\*.



Conclusion: Does not look too bad when A = A\*, P = P\*. Note that this does not take the DS’ness into account. We see tha the gradients are very small for the lower triangular part. Furthermore, the behavor of p22 and p33 is very strange at the beginning, diverging to infinity when it gets close to zero (it was cropped out in the figures).

Verification of gradient w.r.t. P when P is another permutation.



Conclusion: Although the gradient is indeed correct, we see that the outcomes do not make much sense. We inputted the permutation [2, 3, 1], when [1, 2, 3] is correct. We would find the global optimum when p11 = 1, p12 = 0, p13 = 0. However, it finds the local optima for p11 close to zero, and p12 and p13 also close to 0. For p22 we indeed get 1, but for p23 we also get 1 as optimum. For p33, we get p31 to be 1 as optimum, but this is not the global optimum with P\*.

# Solve the problem in the population setting

**New approach: Estimate non-restriced A, then top sort to get P.**

**A\* a DAG**

I have been trying to solve this problem in the population setting, and I discovered that iteratively improving P and A is not very feasible (I also found a very similar paper that argued the same). Hence, I tried something different. I just estimate A by iteratively computing the local optima of the entries, after which I retrieve a permuted version of A\*. Then, iteratively using topological sort yields the permutation, easy as that!

However, this assumes that A\* is the WAM of a DAG. However, in the experiments of e.g. NOTEARS, this was always the case. So e.g. using least squares to estimate A\*, thresholding, and then doing topological sort should also get some okayish results.

What do we do when the estimated A is not the WAM of a DAG? We iteratively increase the threshold, or equivalently, we set the most neglible not-yet-thresholded entry to zero, and retry, until we succeed.

This problem is **not** NP-hard when A\* is indeed a DAG. However, when A\* is **not** a DAG, then what is the best A that is the WAM of a DAG, where best here means it minimizes the cost function? I think that problem is NP-hard, as simply setting the smallest entry to zero until we have a WAM of a DAG will not necessarily yield in the smallest score increase.

**A\* not a DAG**

How do we do this top-sort? We remove diagonal entries (as we ignore self-loops). Then, we find a row with no non-zero entries, and set that row and that column all to zero (remove the dependencies on this node), and add this node to the return list. This is the root node; then, we iteratively do this until we have a full topological list. Note that when two permutations are possible, e.g. [0,1,2] and [0,2,1], we want to pick the one that is lexographically the smallest, to avoid identifiability issues.

This works for VAR(1)s, however, not for SEMS, as estimating that matrix seems to be more difficult.

**Open question for now: SEM estimating.**

**Really interesting papers, close to our approaches**

1. Learning Bayesian Networks through Birkhoff Polytope: A Relaxation Method.

Also relaxes the constraint! Solves PX – PAX. Comes to the exact same conclusion (always seems to converge to J/p. Uses an interesting technique which forces the solution to be more on Permutations. However, not convex, still very interesting!

1. Inferring large graphs using $$\ell \_1$$ ℓ 1 -penalized likelihood.

GADAG; competitive with NOTEARS. Solves X – PTAPX. Comes to the exact same conclusion: coordinate ascent w.r.t. A, P does not really make sense. Better is to use a whole different method to find P, then find A. Then, use a **genetic** algorithm to find P, then find A, etc.