1 Solving with AWGN

So far, we have focused on an algorithm that will solve the blind decoding problem for the BPSK case in the absence of noise. In this document we discuss an approach to adapt our algorithm to handle noisy inputs. I initially set out to make an algorithm that just works rather than worrying about optimality. The solution proposed here indeed works fairly well; it also seems tractable to analyze the proposed method theoretically which seems promising.

The changes to handle noise are all in the dynamic step of finding a BFS. Once we have at least n 'good' columns of $\mathbf{UY}$ (meaning linearly independent and all in $\{-1,+1\}^n$), then the rest of the algorithm can proceed as normal. The additional entries of $\mathbf{UY}$ will not necessarily be ± 1-valued but our constraints will ensure that we do not exceed ± 1. Out of the box, the odds that the dynamic step will find enough 'good' columns is low, especially when k is large. We illustrate why this is the case below and then propose a simple fix, a 'centering step', which seems to work reasonably well.

I still need to take a step back and consider this approach bit more theoretically. Currently I really like this approach. For one, it gives a good sense of where the errors in the final solution come from, namely errors will occur when we incorrectly choose an entry to be centered. A bit more on theory later.

From a performance perspective it works well but imposes an error floor. The error floor is still pretty low ($\sim 10^{-4}$); it is worth exploring whether or not we can improve on this. It does get a little slow for low SNR and when parameters are chosen incorrectly but I believe there's a lot of room to improve the performance of this algorithm.

1.1 What Goes Wrong: the Noiseless Case

First consider the noiseless case. Suppose we have the following value of $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & -1 & -1 \end{bmatrix}. \tag{1}$$

Most of the time, for $n = 4$, the dynamic step of the algorithm will return something that has all ± 1 entries. It turns out that (1) ensures that the only global optima are solutions to the blind decoding problem, so this means that almost all the time we are immediately done after the dynamic step. It is possible, though unlikely, that we end up a matrix that has all ± 1 entires, but is not an optima, in which case we have to do a single simplex hop to be done.

Sometimes, we end up with a $\mathbf{UY}$ that has some zeros in it, i.e. it's entries are in $\{-1, 0, +1\}$:

$$\mathbf{UY} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & 0 & 1 \\ 1 & -1 & -1 & 1 & 0 & -1 & -1 \end{bmatrix}. \tag{2}$$

The dynamic solver is only allowed to move without affecting any active constraints. In this case, the solver has to go that is in the nullspace of the active constraints but we are clearly not yet at a solution. A similar situation can happen (even more rare) where the entries are in $[-1, 1]$ rather than $\{-1, 0, +1\}$:

$$\mathbf{UY} = \begin{bmatrix} 0.265 & 1 & -1 & -1 & 1 & -1 & -1 \\ -0.208 & -1 & -1 & -1 & -0.604 & 1 & -0.604 \\ -1 & -1 & -1 & 1 & -1 & -1 & -1 \\ 1 & 1 & -1 & 1 & 0 & -1 & 0 \end{bmatrix}. \tag{3}$$

In both cases we take a similar approach, assuming we have at least four 'good' columns. We simply take the columns of $\mathbf{UY}$ that are ± 1-valued and form a simplex tableau out of these. If we don't actually have

four columns that are ± 1 and linearly independent, then we just run the dynamic algorithm again with a new starting point. We have come up other approaches besides restarting but restarting works very well; we may choose to come back to other approaches later.

Once we have formed an initial tableau, we then hop vertices, ensuring that each 'bad' column remains feasible. For $n = 4$ we generally only have to hop at most a couple times. This approach works well but takes up most of the solver's time. There are currently two ways this process will stop:

- All entires become ± 1. In this case we switch over to full simplex because it is a faster way of ensuring constraints remain satisfied.
- We exhaust the entire state space or exceed a maximum iteration count. This is what we refer to as a "trap" case; when this happens we just restart the solver.

We should add a third way: stop when we see an 'is_done' criteria.

1.2 Centering Step: First Attempt

We now consider the behavior of the solver given the above value of $\mathbf{X}$ when we add some noise. Suppose we have an SNR of 30 dB and run the dynamic step as is. We might end up with something like this:

$$\mathbf{UY} = \begin{bmatrix} -1 & -0.961 & 1 & 1 & -0.979 & -1 & 0.957 \\ 1 & -1 & 0.995 & -1 & 0.997 & 1 & 0.997 \\ -1 & -0.991 & -1 & -1 & -0.996 & 1 & 0.992 \\ 1 & -1 & -0.991 & 1 & -0.996 & 1 & 0.999 \end{bmatrix}. \tag{4}$$

The dynamic step forces at least n^2 values of $\mathbf{UY}$ to be exactly ± 1. Without noise, because not all columns can be linearly independent, we often end up with most, if not all, entires in $\mathbf{UY}$ being exactly ± 1, as we saw above. With noise this will no longer be the case; we will almost certainly only have n^2 ± 1-valued columns. As k grows, this means that the odds we end up with n 'good' columns vanishes rapidly.

To get around this issue, we propose forcing values of $\mathbf{UY}$ that are close to be ± 1 to be exactly ± 1. To do this, we choose a threshold, δ, and compute:

$$\Delta_{ij} = \begin{cases} (\mathbf{UY})_{ij} + 1 & |(\mathbf{UY})_{ij} + 1| < \delta \\ (\mathbf{UY})_{ij} - 1 & |(\mathbf{UY})_{ij} - 1| < \delta \\ 0 \end{cases} \tag{5}$$

$$\tilde{\mathbf{Y}} = \mathbf{Y} - \mathbf{U}^{-1}\mathbf{\Delta}. \tag{6}$$

We then form an initial tableau by taking $\mathbf{U}\tilde{\mathbf{Y}}$. As an example, if $\delta = 0.1$ then after centering (4) we have

$$\mathbf{\Delta} = \begin{bmatrix} 0 & 0.039 & 0 & 0 & 0.021 & 0 & -0.043 \\ 0 & 0 & -0.005 & 0 & -0.003 & 0 & -0.003 \\ 0 & 0.009 & 0 & 0 & 0.004 & 0 & -0.008 \\ 0 & 0 & 0.009 & 0 & 0.004 & 0 & -0.001 \end{bmatrix}. \tag{7}$$

In this case it happened to work out that we have solved the blind decoding problem after applying this one centering step. If the center step returns at least four good vertices, then we proceed with simplex as before. If centering does not return four good columns, then we throw out $\tilde{\mathbf{Y}}$, pick a new initial $\mathbf{U}$ and start over again. This is probably not optimal but a reasonable start.

Setting $\delta = 0.1$ may seem high, but is justified as follows: odds are low that we get a case where entires are in $[-1, 1]$ to begin with. Values seem to be somewhat uniformly distributed between $[-1, 1]$ or may actually be biased towards zero but I need to look into this more. So there is 10% of a small chance that we shouldn't be flipping. Even if we do flip, then there is still a 50% chance that this flip doesn't hurt us and it's also not clear that it will always mess up more than one bit if it is bad.

Also, note that the variance from ± 1 that we will see in (4) will potentially be substantially more than the raw noise variance. What we are seeing is linear combinations of the noise, which could grow based on the norm of $\mathbf{U}$ which is the same as the norm of $\mathbf{A}^{-1}$. At high dimensions, I expect the values of $\mathbf{UY}$ to be distributed something like

$$\mathcal{N}\left(1 - \frac{1}{n}\left\|\mathbf{A}^{-1}\right\|_*, \frac{1}{n}\sigma_N \left\|\mathbf{A}^{-1}\right\|_*\right), \tag{8}$$

where σ_N is the noise variance and $\|\cdot\|_*$ is the nuclear norm, but I'm not sure this will be a particularly meaningful expression to work with at low dimensions.

1.3 A False Trap

The centering step given above is not sufficient on it's own, even though it often works. Here is an example for $n = 4, k = 5$ where it doesn't work well. In this case, the solver eventually finds the solution but only after spinning through a bunch of false 'trap' cases.

Suppose we run the BFS solver followed by centering and end up with the following state:

$$\mathbf{UY} = \begin{bmatrix} 1 & -1 & -1 & -1 & -0.0006 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & 1 & -1 & 1 \\ -1 & -1 & 1 & -1 & -0.0081 \end{bmatrix}. \tag{9}$$

Here, we should be able to flip, for example, $(uy)_{1,2}$ followed by $(uy)_{4,1}$ and get to a solution without passing through a singular matrix. However, when we attempt to flip $(uy)_{1,2}$ we get the following result:

$$\mathbf{UY} = \begin{bmatrix} 1 & 1 & -1 & -1 & 1.0076 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & 1 & -1 & 1 \\ -1 & -1 & 1 & -1 & -0.0081 \end{bmatrix}, \tag{10}$$

which the solver sees as infeasible. This case is handled simply by centering entries that are close to 0. In other words, we should compute Δ as:

$$\Delta_{ij} = \begin{cases} (\mathbf{UY})_{ij} + 1 & |(\mathbf{UY})_{ij} + 1| < \delta \\ (\mathbf{UY})_{ij} - 1 & |(\mathbf{UY})_{ij} - 1| < \delta \\ (\mathbf{UY}) & |(\mathbf{UY})_{ij}| < \delta \\ 0 & \end{cases} \tag{11}$$

This gives us success nearly 100% of the time for $n = 4, k = 5$.

1.4 A third issue

By moving up to $k = 6$, we see that centering once is still not sufficient. Let $n = 4, k = 6$ and consider the following value of $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 & -1 & 1 \end{bmatrix}. \tag{12}$$

We can see that columns 4 and 6 are identical. Suppose we get the following output after running the BFS finder and then centering:

$$\mathbf{UY} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0.499 & 1 \\ -1 & 1 & -1 & -1 & 0 & -1 \\ -1 & 1 & -0.144 & 1 & -0.577 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}. \tag{13}$$

This value of $\mathbf{UY}$ does not have enough linearly independent columns to turn into a tableau. In the case, the solver restart with a new value of $\mathbf{U}_0$ and tries again. The issue is that this approach is that the dynamic step will almost never find enough linearly independent columns.

It's not too hard to see why this is the case. Even though columns 4 and 6 are identical (without noise), when noise is added they are no longer identical and not necessarily linearly independent. The BFS finder will thus force both columns 4 and 6 to be all ± 1 valued. We will likely frequently see something like this occur whenever we have columns that are identical up to a sign — once one of the two columns becomes active, the second column will be nearby in terms of ℓ_2 norm and so it is likely the solver will find its way to activating the second column. Since centering doesn't occur until after the BFS solver has been run, the solver believes that the active constraints are full rank. After centering occurs, the set of active constraints will no longer be full rank.

The way to fix this would be to step back into the dynamic programming step with the current value of $\mathbf{U}$ after centering. We should perhaps loop doing this until $\mathbf{U}$ does not change. At most we'd have to do this n times.

1.5 Aside: Numerical Instability

In testing the noisy case, I also ran into the first example where numerical stability becomes problematic. I am actually very pleased at how the solver handles this case. This cases arises when the channel is extremely illconditioned. More than half the time, the dynamic solver returns an infeasible $\mathbf{UY}$ for the following input:

$$\mathbf{y} = \begin{bmatrix} -0.85105277 & -0.63492999 & 3.41237287 & 1.20811145 & 0.35892148 & 2.41836825 \\ 3.96976924 & -0.73519243 & -1.04010774 & 0.12971196 & 1.03251896 & -2.80775547 \\ 3.12355480 & 2.95773796 & 0.67356399 & 0.87056107 & 2.94202708 & 0.68919778 \\ -2.31709338 & 0.66587824 & 1.21174544 & 2.38172050 & -1.41046808 & 3.28820386 \end{bmatrix} \tag{14}$$

$$\mathbf{U} = \begin{bmatrix} -0.06779237 & -0.08776801 & 0.03347982 & -0.04695821 \\ -0.01830087 & -0.05418955 & -0.09233134 & 0.06187484 \\ 0.06205623 & -0.02276697 & -0.05832655 & -0.08862112 \\ 0.08272422 & -0.06683513 & 0.05076455 & 0.04168607 \end{bmatrix} \tag{15}$$

Given a fixed input, the BFS solver should be entirely deterministic so this is purely a matter of numerical instability. The fact that the algorithm returns a feasible $\mathbf{UY}$ some of the time implies that we can get away with just running it a few times and hoping that we end up with something usable. In reality, we may just want to not attempt to solve ill-conditioned inputs.

2 The Full Algorithm and Results

Centering on $\{-1, 0, 1\}$ and recentering after running the BFS solver works very well. Here is the full algorithm.

2.1 The Algorithm

1. (Initialize as usual)
2. Draw $\mathbf{U}_0$ at random as usual
3. Let $i = 0$; $\mathbf{Y}_0 = \mathbf{Y}$.
4. For $i = 0, i < n, i++$:
   (a) $\mathbf{U}_{i+1} = \texttt{find\_bfs}\,(\mathbf{U}_i, \mathbf{Y}_i)$
   (b) If $\|\mathbf{U}_{i+1} - \mathbf{U}_i\|_\infty < \delta$

i. break

(c) Center $\mathbf{Y}$:

i. Compute Δ:

$$\Delta_{ij} = \begin{cases} (\mathbf{UY})_{ij} + 1 & |(\mathbf{UY})_{ij} + 1| < \delta \\ (\mathbf{UY})_{ij} - 1 & |(\mathbf{UY})_{ij} - 1| < \delta \\ (\mathbf{UY}) & |(\mathbf{UY})_{ij}| < \delta \\ 0 \end{cases} \tag{16}$$

ii. $\mathbf{Y}_{i+1} = \mathbf{Y}_i - \mathbf{U}_{i+1}^{-1}\Delta$

5. (Continue as usual)

2.2 Low Noise Benchmark

As a first benchmark, we aimed for an algorithm that simply 'worked' about as often as our initial interior point implementation in MATLAB. We first explored a very low noise case with a 30 dB SNR.

2.2.1 MATLAB Results

k	Success Probability	BER
5	79.3%	<1e-6
10	98.0%	<1e-6
15	98.8%	<1e-6
20	99.1%	<1e-6
30	98.6%	<1e-6

2.2.2 New Algorithm Results

We did not compute the BER yet for this part, we simply looked at whether or not the algorithm finished and whether or not the block had zero bit errors ('Equals ATM'). Runout shows the number of times we exceeded 100 attempts at recovery. Cases that completed but did not return an ATM could be because of a single bit error or we actually recovered the wrong answer. We set $\delta = 0.1$ just to validate our approach.

k	Equals ATM	Completed	Runout
5	19827	19868	132
6	19996	19996	4
10	19794	19817	183
15	19784	19817	33
20	19832	19837	42
30	19551	19610	186

The majority of failures in the above results come from channels being illconditioned. Here are the same results but only for channels that had a smallest singular value of $\sigma_4 > 0.25$:

k	Equals ATM	Completed	Runout
5	12000	12000	0
6	11800	11800	0
10	11000	11000	0
15	10999	11000	0
20	10997	10997	3
30	11789	11791	9

2.3 Error Rate Performance: $n = 4, k = 30$

The performance of the algorithm is quite interesting empirically and I'm excited to dig into these results and try to explain them theoretically. First, we observed that the BER is almost entirely a function of the value of δ. This makes sense as errors only occur when we incorrectly attempt to center a value. If δ is fixed, then as the SNR decreases, the odds that the solver will complete decays. Getting reasonable performance out of the solver depends on picking the appropriate value of δ based on the operating conditions.

Our first set of plots, shown in Figure 1, shows the probability that the algorithm terminates and returns a channel estimate as a function of SNR. We plot various values of δ in the range $0.025 < \delta < 0.5$. In practice, we can choose the value of δ to use based on the acceptable block erasure rate; following modern standards, we consider an erasure rate near 10% a good target.

We also plot the same results but only for channels that are well conditioned, that is have a smallest singular value greater than 0.1. Not surprisingly, the performance of our algorithm is strictly better in this case. In reality, we should probably be taking the condition number of the channel into account when choosing δ. It might be possible to pick δ based on the singular values of $\mathbf{Y}$, but we did not explore how to exploit this relationship.

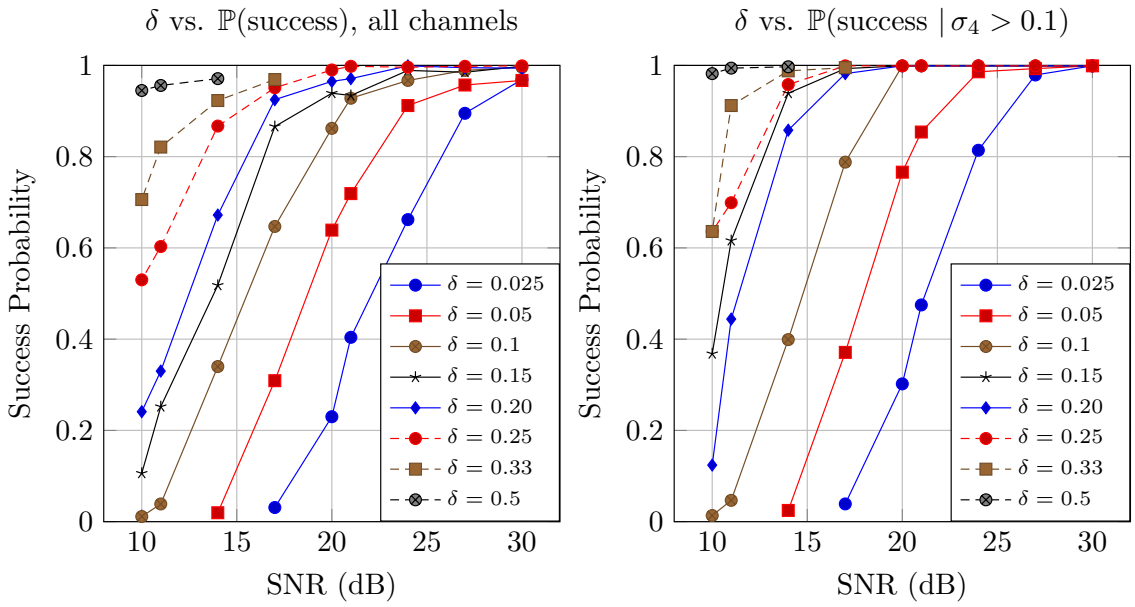

Figure 1: Odds of recovery for different values of δ as a function of SNR. The left has channels drawn i.i.d. $\mathcal{N}(0, 1)$. The right shows the same results with illconditioned channels rejected (we reject those with $\sigma_4 < 0.1$.

Our second set of plots, in Figure 2, shows the average BER as a function of δ. These results are averaged over all SNR values for which the solver had a success rate of at least 90%. I think we should be able to understand these results theoretically as the errors come from incorrectly choosing an entry of $\mathbf{UY}$ to center and seeing how this propagates through the solver. The difficulty in analyzing this performance is that we have randomness cause by the random channel gain matrix, random initial starting point and the channel noise. We might want to start by fixing a channel.

Our final plot Figure 3 shows the BER performance of our algorithm compared to the MATLAB interior point implementation. To get these results, for each SNR, we choose smallest the value of σ that ensures the solver succeeds 90% of the time. A few remarks about these results are in order:

- For low SNR our new algorithm does noticeably better. This is not entirely evident on the log-scale plot

below and is only based on the 10 dB data point: the old algorithm has an error rate of approximately 0.25, whereas our algorithm has a error rate below 0.0075. The interior point algorithm has an error rate near 0.5 for 7 dB. I need to see whether or not we completely fall apart here as well.

- From 11-20 dB, we match the performance of the old algorithm fairly well. The results from our new algorithm look a little strange around 20 dB, but I'm not sure whether or not this is an artifact of having selected δ incorrectly.
- At high SNR, the performance of our new algorithm seems a bit worse. The interior point algorithm has an error rate near 10^{-5} at 23dB and drops below 10^{-6} for 30 dB. However, our algorithm seems to level out near 10^{-4}. This appears to be an error floor that will be present as long as δ is set a non-negligible amount above 0. This error floor seems like a small price to pay given that we are already accepting a 10% erasure rate.

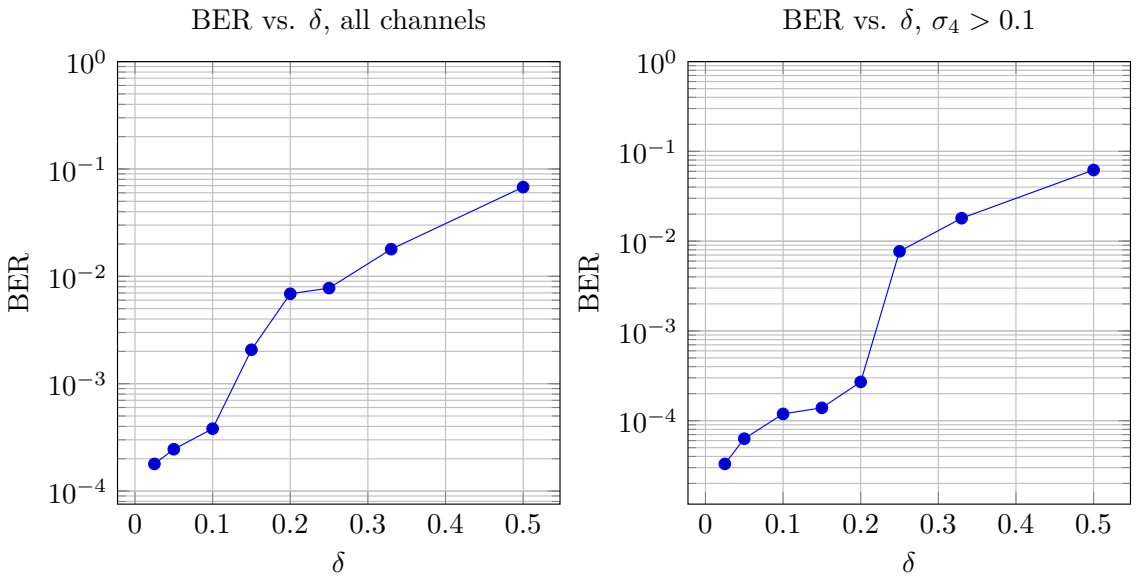

Figure 2: BER as a function of δ. This is averaged across all SNR's at which the solver succeeds at least with a success rate of at least 90%. The BER is essentially constant for a given value of δ.

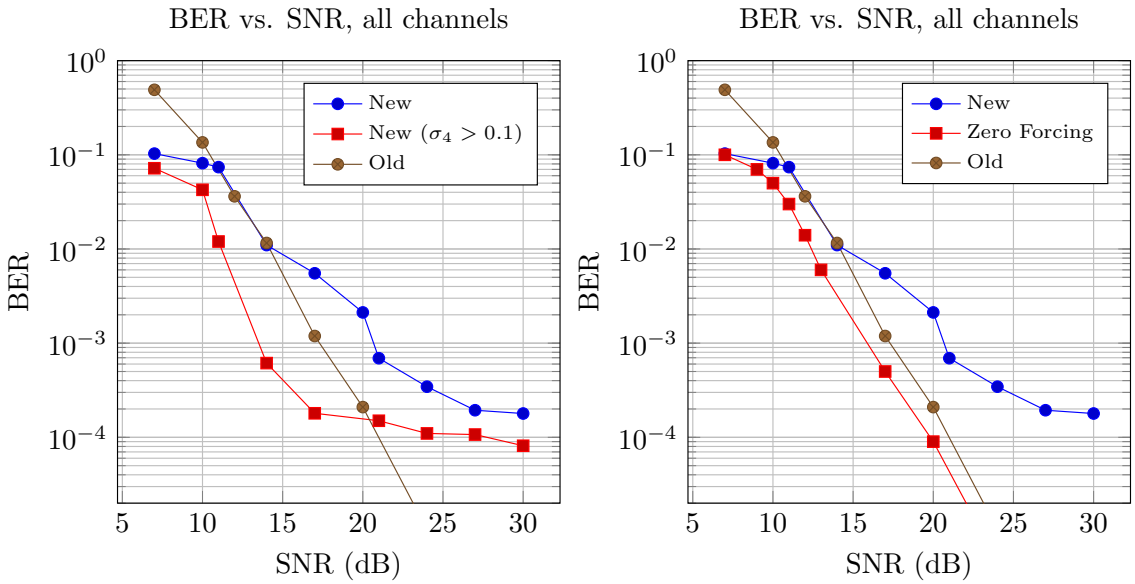

Figure 3: BER vs SNR for the value of δ that gives the best BER an guarantees at least 90% success rate. There is an error floor around 10^{-4}.

3 Thoughts on Future Directions

- If k is large enough, we may wish to throw out the first set of good columns that we find in the dynamic step, and possibly repeat throwing these out until we've discarded some percent of the columns. The idea here is that this is like ellipsoid peeling. If we throw away a few (less than half) we are likely getting rid of the entries where the second moment of the noise is large and 'away' from the origin (relative to the constellation point). Maybe get rid of the expected number that will move the points outside the parallelepiped?
  - My initial experiments for this at $n = 4$ did not go well, but these were done before I had centering working correctly. I attempted to discard columns when the solver returned 'LinDep'. I did this by picking the single column that has the smallest non-zero ℓ_∞ norm of Δ. In the high-runout cases, the solver removed columns until it eventually got caught in a 'trap' case or just returned the wrong answer. The results are strictly worse than the results above.
  - This idea might be worth revisiting now that I have a better implementation. I have only explored results for $k = 30$ and do not understand how performance changes as a function of k. Statistically, there is definitely something that can be exploited by large k and taking an ellipsoid peeling like approach. Whether or not this can be efficiently exploited is another question.
- Need to look into stopping criteria for BFS solver. Is there a way to return a 'best' state even when a runout occurs? It's possible that the subset of good columns cannot be forced into a maximal determinant case (say only columns 1,2,3,5 from (1) are good). In this case the reduced tableau would look like a 'trap' case. The solver may well arrive at the correct answer, but discard it because the remaining entries are not ± 1-valued.
  - This does not seem to be the case. At $n = 4$, we tend to either get k good columns of a 'LinDep' failure. There does not seem to be a simple change to the stopping criteria that would yield anything meaningful.
  - I do need to include checking the 'is_done' criteria in the initial BFS solver
  - It is still considering how we might either return something useful from the runout case or consider whether or not it is possible to tell we won't get a solution sooner and stop before we waste too much computational power.
- Verify distribution of entires in $[-1, 1]$ after the initial run of the dynamic solver. This seems critical to understanding the error performance from a theory perspective.
- Rather than centering entries close to 1, is it possible to instead change some of them into inequality constraints in the tableau?
- Need to consider implementing the corner case solver for the dynamic step of the solver.

4 A shot at some theory

The first theoretical question we want to ask is: What is the expected number of values that we will flip in the centering step? Mathematically, we can write this as:

$$\mathbf{E}\left[\sum_{i,j}\mathbb{1}\left(\Delta_{i,j}\neq 0\right)\right]=? \tag{17}$$

The answer to this will help us determine the appropriate threshold and also the success probability. This is not an easy question to answer but I think we can make progress towards an answer. We can start by breaking this down into some simplifying assumptions.

First assume that $\mathbf{A}\in O(n)$. This should make the entires of $\mathbf{A}^{-1}\mathbf{E}$ easy to analyze. Unfortunately, it is not that case that $\mathbf{U}$ will necessarily be orthogonal or close to being orthogonal. For example, at $n=4$, there will be vertices at $\det 8$, and on these vertices we will have $|\det \mathbf{UA}|=0.5$. For now, let's focus on the simplest case that $\mathbf{A}\in O(n)$ and $\mathbf{U}$ is close to a global optima.

For $n=4$, there will be two sets of columns of $\mathbf{X}$. Assume that $x_1,\ldots,x_4$ form a Hadamard matrix. The half of the remaining columns will be in the form $x_i=\pm x_j$ for some $i\in\{1,\ldots,4\}, j\in\{5,\ldots,k\}$. Let's focus on this half for now. The entry $x_{i,l}$ that will get forced to ± 1 will be the one with the largest value of $e_{i,l}$ for all i such that $x_i=\pm x_j$. So the expected number of $x_{i,k}$'s that will be centered equal to the expected number of $e_{i,l}$'s that are within δ of $\max_i e_{i,l}$. We also know that each row of $\mathbf{UY}$ (more precisely each row in the subset of columns of $\mathbf{UY}$ where the columns of $\mathbf{X}$ are equal is independent. So we have

$$\mathbf{E}\left[\sum_{i,j}\mathbb{1}\left(\Delta_{i,j}\neq 0\right)\right]\approx n\cdot\mathbf{E}\left[\sum_{j}\mathbb{1}\left(\Delta_{i,j}\neq 0\right)\right] \tag{18}$$

$$\approx n\cdot\mathbf{E}\left[\sum_{j}\mathbb{1}\left(\max_i|e_{i,j}|-|e_{j,j}|<\delta\right)\right]. \tag{19}$$

Next, we have that (should find a reference)

$$\mathbf{E}\left[\max_i|e_{i,j}|\right]=\sigma_N\sqrt{2\log k} \tag{20}$$

Now evaluating (19) is a simple matter of evaluating the Gaussian CDF, $\Phi(x)$:

$$\mathbf{E}\left[\sum_{i,j}\mathbb{1}\left(\Delta_{i,j}\neq 0\right)\right]=n\cdot\mathbf{E}\left[\sum_{j}\mathbb{1}\left(\max_i|e_{i,j}|-|e_{j,j}|<\delta\right)\right] \tag{21}$$

$$=nk\left[\Phi\left(\sigma_N\sqrt{2\log k}-\delta\right)-\Phi\left(\sigma_N\sqrt{2\log k}\right)\right] \tag{22}$$

k above should really be the number of columns that look like $x_i\neq\pm x_j$.

This is unfortunately a worthless bound as it predicts that, for cases simulated in the last section, we will center all entries of UY with almost certainty. This is clearly wrong based on the number of runouts!

4.1 Problems with the above analysis

- In each row, we are actually selecting n entires, so we don't care about $\max e_{i,j}$ as much as we care about the nth largest entry.
- The number of columns that look like $x_i=\pm x_j$ are pretty small. When this isn't the case, the interaction between $e_{l,i}$ and $e_{l,j}$ is non-trivial. I think we'd have to look at some sort of norm of $\mathbf{U}$ to tell what they'd look like.

- The order that each $x_{i,j}$ becomes active is mostly a function of the initial choice of $\mathbf{U}$
- For columns that aren't like $x_i = \pm x_j$, then we don't necessarily pick the largest value first so it isn't yet clear to me how to apply order statistics.
- $\mathbf{A}$ is certainly not orthogonal nor is it necessarily well conditioned.

5 Raw Data

5.1 Results $\delta = 0.01$

Theses are for $n = 4, k = 30$ with 100 channels, 200 trials per channel, 100 attempts max per trial, $\delta = 0.01$.

SNR(dB)	Completed	Runout	Error	BER
30	0.766	0.234)	4.00e-4	5.48e-4
27	0.572	0.427	3.00e-4	6.77e-4
24	0.167	0.833	5.00e-5	3.41e-4
21	2.26e-2	0.977	0	0

Given $\sigma_4 > 0.1$:

SNR(dB)	Completed	Runout	Error	BER
30	0.980	1.97e-2	5.41e-4	2.32e-4
27	0.690	0.310	3.61e-4	6.52e-4
24	0.212	0.788	6.33e-5	3.41e-4
21	0.475	0.525	5.88e-5	4.40e-4
20	2.79e-2	0.972	0	0

5.2 Results $\delta = 0.025$

Theses are for $n = 4, k = 30$ with 100 channels, 200 trials per channel, 100 attempts max per trial, $\delta = 0.025$.

SNR(dB)	Completed	Runout	Error	BER
30	0.969	3.00e-2)	1.45e-3	1.785e-4
27	0.895	0.105	4.50e-4	7.23e-4
24	0.662	0.338	1.50e-4	2.19e-3
21	0.404	0.596	5.00e-5	4.39e-4
20	0.230	0.770	0	7.06e-6
17	3.1e-2	0.969	0	3.508e-3

Given $\sigma_4 > 0.1$:

SNR(dB)	Completed	Runout	Error	BER
30	0.999	0	1.10e-3	5.14e-5
27	0.979	2.02e-2	4.55e-4	1.87e-4
24	0.814	0.186	1.85e-4	1.69e-3
21	0.475	0.525	5.88e-5	4.40e-4
20	0.302	0.698	0	7.98e-5
17	3.89e-2	0.961	0	3.51e-3

5.3 Results, $\delta = 0.05$

Theses are for $n = 4, k = 30$ with 100 channels, 200 trials per channel, 100 attempts max per trial, $\delta = 0.05$.

SNR(dB)	Completed	Runout	Error	BER
30	0.967	3.15e-2)	1.35e-3	3.520e-4
27	0.957	4.20e-2	8.50e-4	2.01e-4
24	0.912	8.76e-2	5.00e-4	1.83e-4
21	0.669	0.331	1.00e-4	3.16e-4
20	0.639	0.371	1.5e-4	7.64e-4
17	0.309	0.691	0	5.85e-4
14	1.95e-2	0.981	0	0

Given $\sigma_4 > 0.1$:

SNR(dB)	Completed	Runout	Error	BER
30	0.999	0	7.93e-4	0
27	0.993	0	7.14e-4	0
24	0.986	1.37e-2	4.6e-4	1.01e-4
21	0.854	0.146	6.49e-5	2.70e-4
20	0.766	0.234	1.23e-4	1.93e-4
17	0.371	0.628	0	5.85e-4
14	2.46e-2	0.975	0	0

5.4 Results, $\delta = 0.10$

Theses are for $n = 4, k = 30$ with 100 channels, 200 trials per channel, 100 attempts max per trial, $\delta = 0.1$.

SNR(dB)	Completed	Runout	Error	BER
30	0.996	2.8e-3	1.45e-3	3.96e-4
27	0.989	0.03	5.5e-4	1.94e-4
24	0.967	3.2e-2	5.5e-3	3.45e-4
21	0.928	7.21e-2	2.5e-4	5.88e-4
20	0.862	0.138	1.5e-4	7.64e-4
17	0.647	0.354	0	1.85e-4
14	0.340	0.661	0	1.08e-3
11	3.88e-2	0.961	0	0
10	1.13e-2	0.989	0	0

Given $\sigma_4 > 0.1$:

SNR(dB)	Completed	Runout	Error	BER
30	0.999	0	9.5e-4	7.55e-5
27	0.999	0	3.2e-4	0
24	0.999	0	2.5e-4	1.71e-5
21	0.999	7.14e-4	2.5e-4	2.72e-4
20	0.997	2.27e-2	1.5e-4	2.29e-4
17	0.788	0.212	0	6.70e-5
14	0.399	0.601	0	1.08e-3
11	4.67e-2	0.989	0	0
10	1.34e-2	0.987	0	0

5.5 Results, $\delta = 0.15$

Theses are for $n = 4, k = 30$ with 100 channels, 200 trials per channel, 100 attempts max per trial, $\delta = 0.15$.

SNR(dB)	Completed	Runout	Error	BER
30	0.998	5e-5	1.5e-3	1.28e-3
27	0.985	1.48e-2	7.0e-4	2.56e-3
24	0.989	1.05e-2	3.5e-3	2.76e-3
21	0.934	6.54e-2	2.0e-4	1.45e-3
20	0.939	6.09e-2	2.5e-4	2.32e-3
17	0.866	0.134	5e-5	2.06e-3
14	0.518	0.482	0	2.70e-3
11	0.252	0.748	0	1.34e-4
10	0.106	0.894	0	5.02e-3

Given $\sigma_4 > 0.1$:

SNR(dB)	Completed	Runout	Error	BER
30	0.999	0	8.93e-4	1.16e-4
27	0.999	0	4.38e-4	1.83e-4
24	0.999	0	2.5e-4	1.71e-5
21	0.999	0	3.09e-4	2.57e-5
20	0.999	0	1.88e-4	0
17	0.993	4.22e-4	2.41e-4	1.81e-5
14	0.939	6.06e-2	5.68e-5	6.14e-4
11	0.616	0.384	0	9.06e-4
10	0.124	0.876	0	1.28e-4

5.6 Results, $\delta = 0.20$

Theses are for $n = 4, k = 30$ with 100 channels, 200 trials per channel, 100 attempts max per trial, $\delta = 0.20$.

SNR(dB)	Completed	Runout	Error	BER
30	0.994	1.45e-2	1.35e-3	5.35e-3
27	0.995	4.15e-3	1.00e-3	7.95e-3
24	0.999	1.1e-3	4.00e-4	3.63e-3
21	0.971	2.15e-2	8.00e-3	1.06e-2
20	0.965	3.49e-2	2.50e-4	7.75e-3
17	0.925	7.51e-2	1.5e-4	6.02e-3
14	0.672	0.328	0	5.92e-3
11	0.330	0.670	0	6.17e-3
10	0.271	0.729	0	2.82e-3

Given $\sigma_4 > 0.1$:

SNR(dB)	Completed	Runout	Error	BER
30	0.999	0	1.34e-3	1.35e-4
27	0.999	0	3.85e-4	2.11e-4
24	0.999	0	2.60e-4	1.16e-4
21	0.999	0	1.23e-4	2.10e-4
20	0.999	0	2.47e-4	4.03e-4
17	0.982	1.57e-2	5.75e-5	3.92e-4
14	0.858	0.142	0	8.59e-4
11	0.444	0.556	0	2.43e-3
10	0.368	0.632	0	3.15e-4

5.7 Results, $\delta = 0.25$

Theses are for $n = 4, k = 30$ with 100 channels, 200 trials per channel, 100 attempts max per trial, $\delta = 0.25$.

SNR(dB)	Completed	Runout	Error	BER
30	0.999	0	1.20e-3	1.98e-3
27	0.998	1.10e-3	4.00e-4	4.75e-3
24	0.996	4.05e-3	4.00e-4	3.63e-3
21	0.998	1.7e-3	3.00e-4	7.99e-3
20	0.990	8.80e-3	8.50e-4	1.13e-2
17	0.951	4.91e-2	2.50e-4	1.13e-2
14	0.867	0.133	0	1.30e-2
11	0.603	0.397	5.00e-5	1.42e-2
10	0.570	0.430	1.00e-4	6.18e-3

Given $\sigma_4 > 0.1$:

SNR(dB)	Completed	Runout	Error	BER
30	0.999	0	1.2e-3	1.9e-3
27	0.999	0	3.95e-4	7.67e-4
24	0.998	0	2.41e-4	5.88e-4
21	0.999	0	5.62e-4	2.69e-4
20	0.999	0	6.33e-5	5.57e-4
17	0.999	1.13e-4	1.14e-4	1.13e-3
14	0.958	4.25e-2	0	1.58e-3
11	0.699	0.301	0	5.548e-3
10	0.636	0.364	0	8.90e-4

5.8 Results, $\delta = 0.33$

Theses are for $n = 4, k = 30$ with 100 channels, 200 trials per channel, 100 attempts max per trial, $\delta = 0.33$.

SNR(dB)	Completed	Runout	Error	BER
17	0.969	2.70e-2	4.35e-3	2.48e-2
14	0.923	7.55e-2	1.1e-3	1.1e-2
11	0.821	0.178	3.00e-4	2.40e-2
10	0.706	0.294	0	3.66e-2

Given $\sigma_4 > 0.1$:

SNR(dB)	Completed	Runout	Error	BER
17	0.995	0	4.81e-3	3.31e-3
14	0.988	1.15e-2	1.01e-3	4.81e-3
11	0.912	8.10e-2	2.38e-4	1.02e-2
10	0.636	0.364	0	8.90e-4

5.9 Results, $\delta = 0.5$

Theses are for $n = 4, k = 30$ with 100 channels, 200 trials per channel, 100 attempts max per trial, $\delta = 0.5$.

SNR(dB)	Completed	Runout	Error	BER
14	0.971	2.34e-2	5.80e-3	5.28e-2
11	0.956	4.04e-2	4.00e-3	7.40e-2
10	0.945	4.92e-2	6.30e-3	7.65e-2

Given $\sigma_4 > 0.1$:

SNR(dB)	Completed	Runout	Error	BER
14	0.997	6.10e-4	2.50e-3	2.59e-2
11	0.994	3.14e-4	2.69e-3	3.32e-2
10	0.982	1.50e-2	2.59e-3	4.35e-2