## ECE 6270, Spring 2021

## Homework #1 – SOLUTIONS

1. Prepare a one paragraph summary of what we talked about in class so far. I do not want just a bulleted list of topics, I want you to use complete sentences and establish context (Why is what we have learned relevant? How does it connect with other classes?). The more insight you give, the better.

   Solution: Make sure that what is written is in coherent, complete sentences.

2. Familiarize yourself with Python.

   (a) Install (if necessary) the Anaconda distribution of Python at https://www.anaconda.com/products/individual.

   (b) Review the Python tutorial at https://cs4540-f18.github.io/notes/python-basics.

   (c) Check out and consider bookmarking this site: https://numpy.org/doc/stable/user/numpy-for-matlab-users.html

   (d) If you feel it would be helpful, especially if you are new to Python, consider taking the time to go through one (or both) of these more basic Python tutorials:

   - https://developers.google.com/edu/python/
   - https://www.learnpython.org/

3. In the problems below and on subsequent homeworks you can/will make frequent use of the CVXPY package. You will probably need to install this package as it does not come standard with most python distributions. This should be easy – simply open up a Python prompt and type `pip install cvxpy`. If you are on Windows you may also need to install the Visual Studio build tools if you don't already have them. If you are having any trouble, see https://www.cvxpy.org/install/ for details. Install CVXPY and poke around on the CVXPY website to get a feeling for what it can do.

4. In our first class we mentioned a nonconvex optimization problem that nevertheless has a simple solution:

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{maximize}} \ \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} \quad \text{subject to} \quad \|\boldsymbol{x}\|_2 = 1.$$

   Here we consider a special case of this problem (that is not hard to extend to the general case). Suppose that $\boldsymbol{D}$ is a *diagonal* $N \times N$ matrix whose diagonal elements are positive. Show that one possible maximizer to

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{maximize}} \ \boldsymbol{x}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x} \quad \text{subject to} \quad \|\boldsymbol{x}\|_2 = 1$$

   is given by the $\widehat{\boldsymbol{x}}$ with a 1 in the entry corresponding to the largest diagonal element of $\boldsymbol{D}$, and is 0 elsewhere. You should approach this by rigorously arguing that any feasible $\boldsymbol{x}' \neq \widehat{\boldsymbol{x}}$ cannot achieve a higher value for the objective function.

Solution: First, note that in general, we can also write our objective function as

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x} = \sum_{n=1}^{N} x_n(d_n x_n) = \sum_{n=1}^{N} d_n x_n^2.$$

Now, suppose without loss of generality that the first entry $d_1$ on the diagonal is the largest, so that $d_1 \geq d_n$ for all $n \neq 1$. In this case the objective function for our proposed $\widehat{\boldsymbol{x}}$ takes a value of $\widehat{\boldsymbol{x}}^{\mathrm{T}}\boldsymbol{D}\widehat{\boldsymbol{x}} = d_1$. Now, consider any $\boldsymbol{x}' \neq \widehat{\boldsymbol{x}}$ satisfying $\|\boldsymbol{x}'\|_2 = 1$. Note that we can write the objective function evaluated at $\boldsymbol{x}'$ as

$$\begin{aligned}
\boldsymbol{x}'^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x}' &= \sum_{n=1}^{N} d_n (x_n')^2 \\
&\leq \sum_{n=1}^{N} d_1 (x_n')^2 \\
&= d_1 \sum_{n=1}^{N} d_1 (x_n')^2 \\
&= d_1,
\end{aligned}$$

where the inequality follows from the assumption that $d_1 \geq d_n$ and the final equality from the fact that $\|\boldsymbol{x}'\|_2 = 1$. But this shows that $\boldsymbol{x}'^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x}' \leq d_1$ for any possible $\boldsymbol{x}'$, and thus $\widehat{\boldsymbol{x}}$ is a maximizer.

5. A significant motivation in the early development of optimization algorithms involved applications in logistics and supply chain management. This is a somewhat contrived example. Suppose you are the manager of a factory that produces two products: toilet paper and paper towels. The profit for paper towels is \$10 per ton of pulp, but the profit for toilet paper is \$100 per ton of pulp. Your factory has a supply of 8 000 tons of pulp per month. In addition, you face the following conditions:

   - You have a government contract that requires you to produce 200 tons of paper towels each month.[1]
   - You must ship all of the products that you produce using a fleet of trucks that has a capacity of 100 000 ton-miles (i.e., you could transport 100 000 tons 1 mile, or 1 ton 100 000 miles, or somewhere in between).
   - Paper towels are delivered to a warehouse 10 miles from your factory, while toilet paper is delivered to a warehouse 35 miles from your factory.

   You would like to decide how much of each product to produce to maximize your profit. Write down a mathematical formulation of this as an optimization problem. What kind of optimization problem is this (linear, quadratic, semidefinite, ...)?

[1] Assume that 1 ton of paper towels or 1 ton of toilet paper each require 1 ton of raw pulp.

Solution: If $x_1$ denotes the tons of pulp devoted to paper towels and $x_2$ denotes the tons of pulp devoted to toilet paper, then this can be represented as

$$\begin{aligned} \underset{x_1,x_2}{\text{minimize}} \quad & 10x_1 + 100x_2 \\ \text{subject to} \quad & x_1 \geq 200, \quad x_2 \geq 0 \\ & x_1 + x_2 \leq 8\,000 \\ & 10x_1 + 35x_2 \leq 100\,000. \end{aligned}$$

6. Consider the optimization program

$$\min_{\boldsymbol{x} \in \mathbb{R}^N} \ \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_1,$$

where $\boldsymbol{A}$ is a given $M \times N$ matrix and $\boldsymbol{y} \in \mathbb{R}^M$. Show how this can be converted into an equivalent linear program.

Solution: Let $\boldsymbol{a}_m \in \mathbb{R}^N$ be the (transpose of the) $m^{\text{th}}$ row in $\boldsymbol{A}$. Then the solution $(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{u}})$ to the program

$$\min_{\boldsymbol{x},\boldsymbol{u}} \ \sum_{m=1}^{M} u_m, \quad \text{subject to} \quad -u_m \ \leq \ y_m - \boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x} \ \leq \ u_m, \quad m = 1, \ldots, M,$$

will obey $|y_m - \boldsymbol{a}_m^{\mathrm{T}}\widehat{\boldsymbol{x}}| = \widehat{u}_m$. This is clear, as if $|y_m - \boldsymbol{a}_m^{\mathrm{T}}\widehat{\boldsymbol{x}}| > \hat{u}_m$, the constraints above are violated, and if $|y_m - \boldsymbol{a}_m^{\mathrm{T}}\widehat{\boldsymbol{x}}| < \widehat{u}_m$, the function is not as small as it could be while still meeting the constraints (we could simply lower $\widehat{u}_m$).

We can rewrite the above as a linear program in $\mathbb{R}^{N+M}$:

$$\min_{\boldsymbol{z} \in \mathbb{R}^{N+M}} \ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{z} \quad \text{subject to} \quad \boldsymbol{C}\boldsymbol{z} \leq \boldsymbol{b},$$

with $\boldsymbol{c} = \begin{bmatrix} \boldsymbol{0}_N \\ \boldsymbol{1}_M \end{bmatrix}$, and

$$\boldsymbol{C} = \begin{bmatrix} -\boldsymbol{A} & -\mathbf{I} \\ \boldsymbol{A} & -\mathbf{I} \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} -\boldsymbol{y} \\ \boldsymbol{y} \end{bmatrix}.$$

7. One important optimization problem that arises in the context of machine learning has the form

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \ \sum_{m=1}^{M} \max(\boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x}, z_m) + \tau\|\boldsymbol{x}\|_2^2.$$

Show how to express this program as a quadratic program with linear constraints. (Hint: $\max(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}, z) \leq u$ is the same as saying $\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} \leq u$ and $z \leq u$.)

Solution: We introduce new optimization variables $\boldsymbol{u} \in \mathbb{R}^M$, replace $\sum_m \max(\boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x}, z_m)$ with $\sum_m u_m$ and add the constraints that $\boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x} \leq u_m$ and $z_m \leq u_m$. The final

quadratic program with linear constraints is

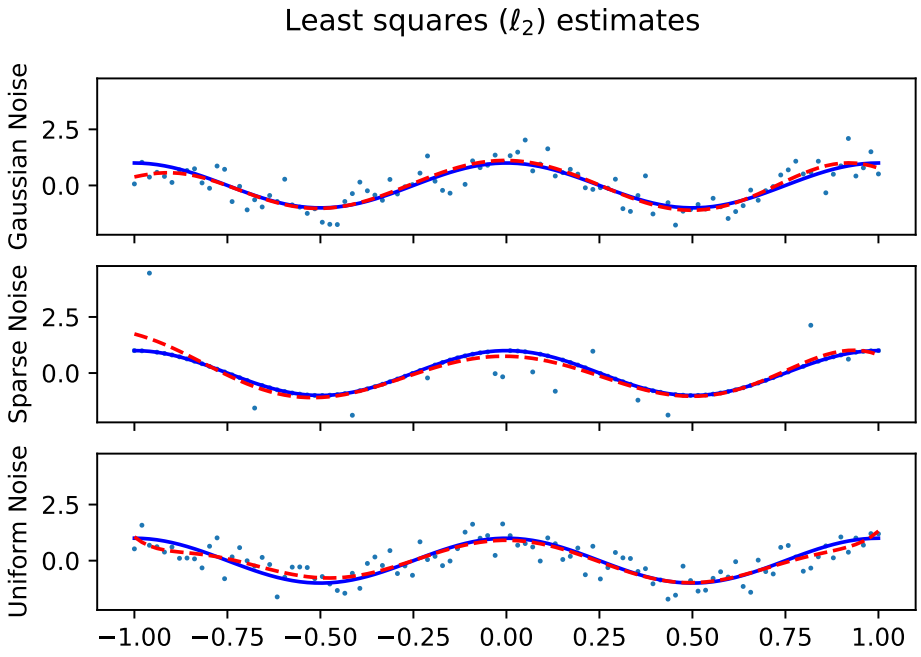
$$\underset{\boldsymbol{x}\in\mathbb{R}^N,\boldsymbol{u}\in\mathbb{R}^M}{\text{minimize}} \sum_{m=1}^{M} u_m + \tau\|\boldsymbol{x}\|_2^2 \quad \text{subject to} \quad \begin{aligned} &\boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x} - u_m \leq 0 \\ &z_m - u_m \leq 0 \\ &m = 1, \ldots, M. \end{aligned}$$

To convince yourself that these problems are equivalent, let $\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{u}}$ be minimizers of the program above. Then it is true that $\widehat{u}_m = \max(\boldsymbol{a}_m^{\mathrm{T}}, z_m)$ for all $m = 1, \ldots, M$, and so $\widehat{\boldsymbol{x}}$ is also a minimizer for the original program. This follows from two easy observations. First, if $\widehat{u}_m < \max(\boldsymbol{a}_m^{\mathrm{T}}\widehat{\boldsymbol{x}}, z_m)$, then $\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{u}}$ would not be feasible, since at least one of the constraints would be broken. Second, if $\widehat{u}_m > \max(\boldsymbol{a}_m^{\mathrm{T}}\widehat{\boldsymbol{x}}, z_m)$ for any $m$, then $\widehat{\boldsymbol{u}}$ is not optimal, as we could simply decrease $\widehat{u}_m$, while simultaneously maintaining feasibility and lowering the functional value.

8. You hopefully have some prior experience with least squares problems in which we aim to find an $\boldsymbol{x}$ that minimizes $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$. While minimizing the $\ell_2$ norm of the error is *often* a good idea, a big part of why it is so popular is just that it is so easy to solve the underlying optimization problem. However, with modern tools we can easily explore situations where other $\ell_p$ norms might be more appropriate. In particular, we will consider minimizing $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_1$ and $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_\infty$.

   (a) Download the file `hw01_prob8.py`. This file sets up a simple regression problem in which $\boldsymbol{y}$ consists of noisy observations of a smooth function $f(t)$ and considers three noise models: Gaussian noise, sparse Gaussian noise, and uniform noise. Compute and plot the least squares solution for each case.

   Solution: See the python code in Canvas.

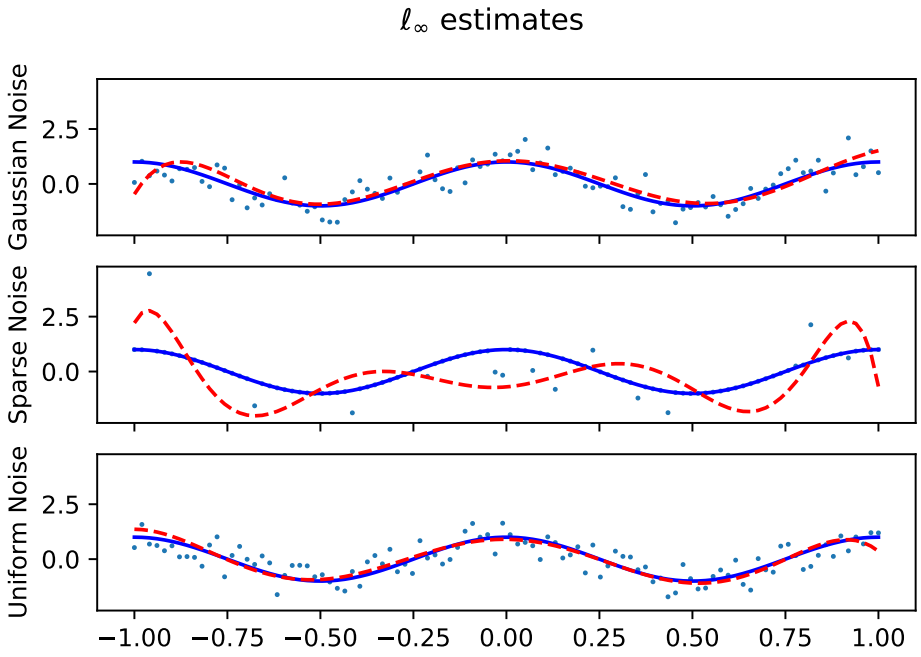


(b) Now estimate the solution for the Chebyshev approximation problem:

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}}\ \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_\infty.$$

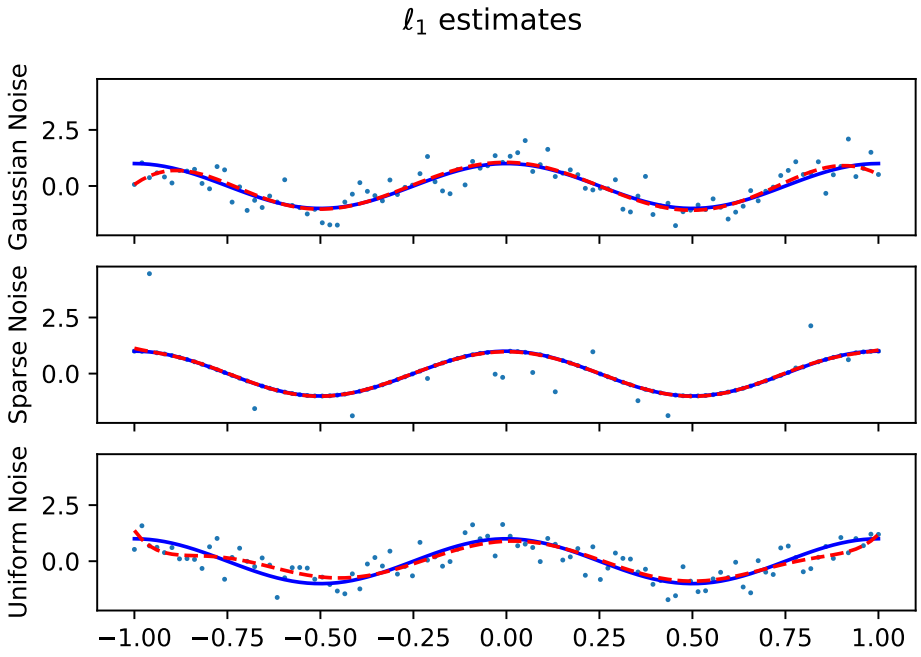
You do not have to convert this to a linear program first; CVXPY has built-in tools for handling the $\ell_\infty$ norm. Compute and plot the resulting estimate of $\boldsymbol{x}$ for each of the three noise cases.

Solution: See the python code in Canvas.




(c) Next repeat part (b) for $\|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_1$. Again, compute and plot the results for each of the three noise cases.

Solution: See the python code in Canvas.




(d) Compare the results that you obtained in the previous parts. Which method seems best suited for each case?

Solution: In the case of Gaussian noise, least squares clearly performs the best. In the case of sparse noise, minimizing the $\ell_1$ norm of the error does dramatically better than the others. In the case of uniform noise the performance is more mixed, although you might give the $\ell_\infty$ norm a slight edge over the others. In this case, due to some large errors near the boundary, none of the methods perform particularly well at the edge of the interval (a quite common phenomenon when fitting a polynomial.)

9. In this problem we will explore the idea of *group testing* as a strategy for testing a large population for a rare disease by pooling samples together. Suppose that we have a population of $N$ people and we collect saliva samples from each of them. We are looking for genetic signatures of a particular virus in these samples. Let $x_n$ denote the concentration of this material in the sample for the $n^{\text{th}}$ person. We will assume that for healthy people $x_n = 0$, but for infected people, $x_n > 0$. Our testing procedure will be to form a series of $M$ mixtures of samples from different subsets of people, and then only run tests on these mixtures. The goal here is to set $M < N$, and the question is then whether we can identify the infected people from the results of these tests.

   We will consider the following approach:[2] we will form mixtures by constructing *random* combinations of samples, and we will attempt to recover the original $\boldsymbol{x}$ using a simple convex optimization problem.

   To mathematically represent the sampling/testing process, assume that we will ultimately run $M$ tests, each of which will tell us the concentration of viral material

[2] Just to be clear, this is not exactly what Georgia Tech is doing, but it is similar in spirit. The approach that Georgia Tech is somewhat simpler and builds on ideas that date back to World War II, when they were used to test soldiers for syphilis.

in the combined sample being tested. For each person, their sample will be divided into $K < M$ equal portions, which will be assigned at random to the $M$ tests. We will do this independently for each of the $N$ people. We can ultimately represent the concentration of viral material in each of the mixed samples that we will ultimately test as a vector $\boldsymbol{y} \in \mathbb{R}^M$. We can write $\boldsymbol{y}$ as

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x},$$

where $\boldsymbol{A}$ is a matrix that represents the assignment of people to mixed samples/tests. Specifically, $\boldsymbol{A}$ is a $M \times N$ matrix where each column is constructed independently by picking $K$ entries at random, setting them to 1, and setting the remaining entries to 0.

Suppose there is no noise in our tests, so that we can estimate $\boldsymbol{y}$ perfectly. Our inference problem is now to estimate $\boldsymbol{x}$ given knowledge of $\boldsymbol{y}$ and $\boldsymbol{A}$. In general, since $M < N$, recovering $\boldsymbol{x}$ is impossible. However, when $\boldsymbol{x}$ is sparse, meaning that it has only a few nonzeros (in this case meaning that most of the population is negative), then recovering $\boldsymbol{x}$ *is* possible, although this fact was only broadly appreciated within the last 15 years or so.

We will try to estimate $\boldsymbol{x}$ by solving the following optimization problem:

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \|\boldsymbol{x}\|_1 \qquad \text{subject to} \qquad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}, \quad \boldsymbol{x} \geq \boldsymbol{0}.$$

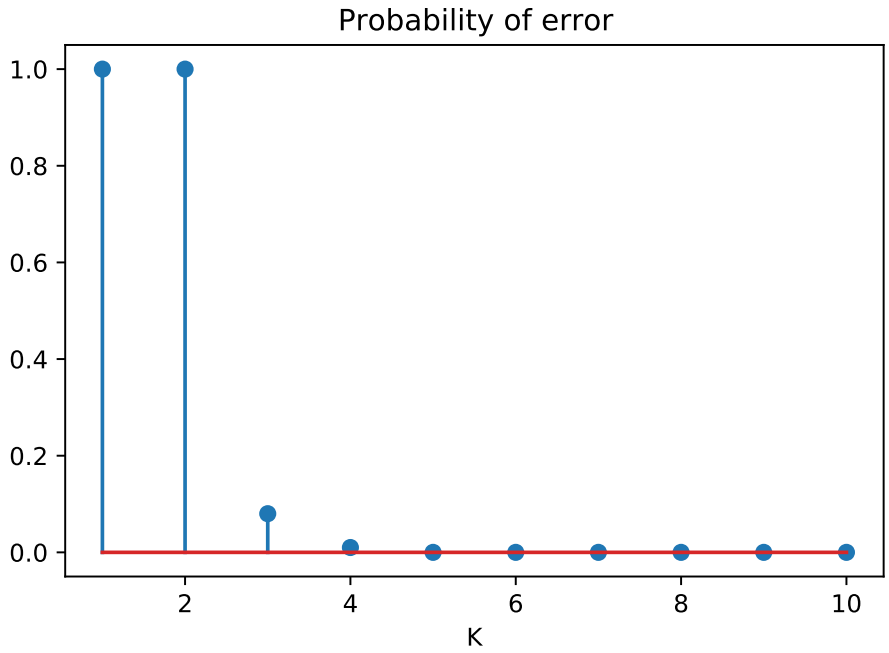
Below we will explore when and how well this works.

(a) Suppose that you are testing a population of size $N = 1000$, but you can only process $M = 100$ tests. Assume that only 1% of the population is positive (meaning that there will be 10 infected individuals). Each person's sample will be split and added to $K = 10$ different batches. The file `hw01_prob09.py` contains code that sets up this problem. Use CVXPY to solve the optimization problem above and verify that this approach correctly identifies the 10 infected individuals.

Solution: See the python code in Canvas.

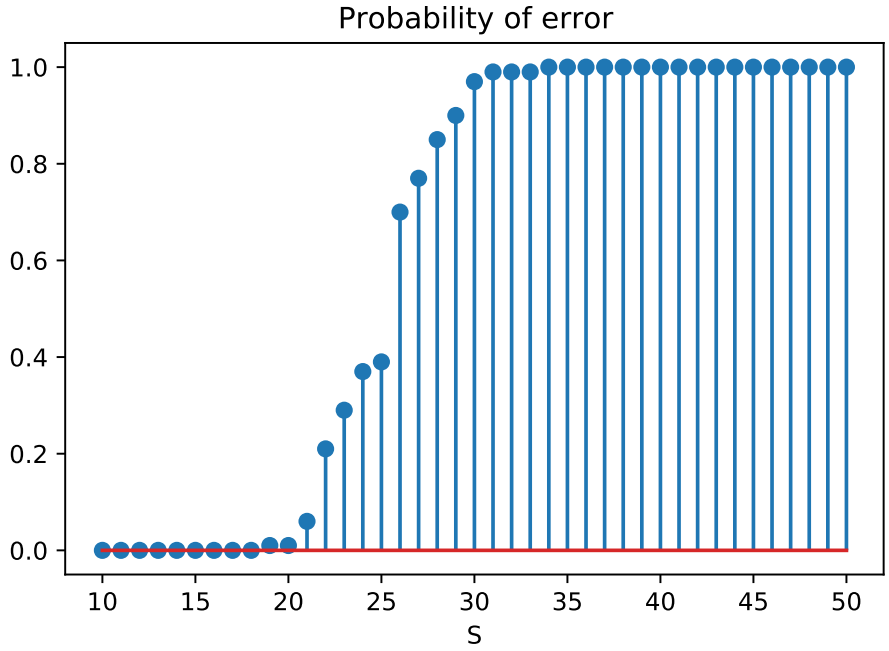
(b) Experiment with $K$. In practice, you might not want to divide a person's sample into too many tests. How low can you set $K$ before things begin to fail?

Solution: See the python code in Canvas. In this part I did 100 random trials with different random assignment to pools. We can take $K$ as low as 5 pools without there being any impact on performance. However, for $K = 4$ there becomes a small (1%) chance of error. When $K = 1$ or $K = 2$, the algorithm failed on every trial. This should make some intuitive sense – when $K = 1$ you need to be *extremely* lucky to be able to unambiguously identify the locations of the nonzeros. The only way this would work (for $K = 1$) is if each nonzero happened to coincide with a column of $\boldsymbol{A}$ where the one nonzero in that column also happened to be the only nonzero in that row. The situation is not too different for $K = 2$, but there is a pretty sharp improvement as $K$ grows.




(c) Suppose that the prevalence of the disease begins to grow beyond 1%. How widespread can the disease become before the approach begins to fail (holding $M$ and $N$ fixed). As the disease becomes more widespread, you may need to adjust $K$. What value of $K$ seems to work best when the spread of the disease is just below the threshold where the approach begins to fail?

Solution: See the python code in Canvas. Using a similar approach as in the previous part, we can look at the probability of failure as $S$ (the number of infected individuals) grows. We see that we still get reliable recovery for $S \leq 18$, but after that if $S$ continues to grow the performance degrades fairly rapidly:




For $S = 18$ if we examine the impact of $K$ we find that as we decrease $K$ the algorithm fails sooner (when $S = 10$ the algorithm still had OK performance for $K = 3, 4$, but as $S$ increases the performance degrades for these particularly small values of $K$.