Corey Schulz
CS 3200 – Spring 2020
March 6, 2020

Homework 3 Report
Solving Equations

Question 1

I computed the LU factorizations of matrices B and C both on paper and in `question_1.m` included in the code directory of this project. When that script is run, it'll display the factorizations of both matrices. To save space in this document, only the matrices and LU factorizations are contained within this document, not the intermediate calculations of finding the LU factorizations.

$$B = \begin{bmatrix} 4 & 1 & -2 \\ 4 & 4 & -3 \\ 8 & 4 & 2 \end{bmatrix}$$

$$L_B = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.5 & -0.5 & 1 \end{bmatrix}$$

$$U_B = \begin{bmatrix} 8 & 4 & 2 \\ 0 & 2 & -4 \\ 0 & 0 & -5 \end{bmatrix}$$

Similarly,

$$C = \begin{bmatrix} 2 & 1 & -2 \\ 4 & 4 & -3 \\ 8 & 4 & 4 \end{bmatrix}$$

$$L_C = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.25 & 0 & 1 \end{bmatrix}$$

$$U_C = \begin{bmatrix} 8 & 4 & 4 \\ 0 & 2 & -5 \\ 0 & 0 & -3 \end{bmatrix}$$

…As you can see, the LU factorizations of matrices B and C are superficially similar just like their parents. The only difference between matrices B and C is that the number in position (1,1) and (3,3) are switched. This doesn't translate directly to the matrix factorizations – you can't just swap one number in the factorizations interchangeably like the parent matrices after all, but the LU factorizations, by looking at them, are indeed similar.

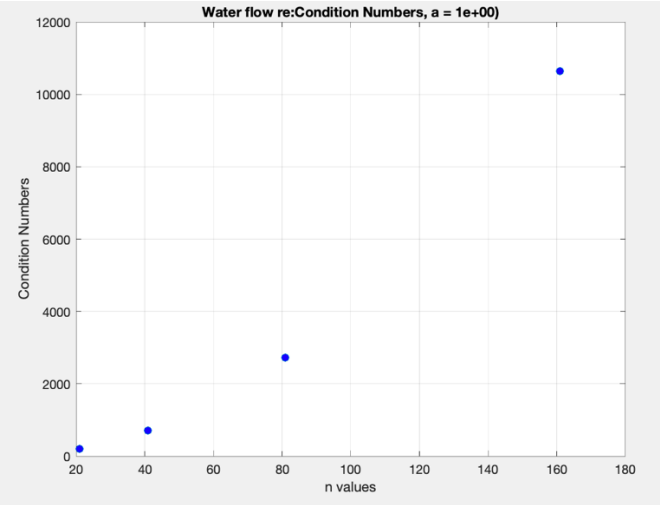
Question 2

The following linear system of equations describes the flow of two different liquids:

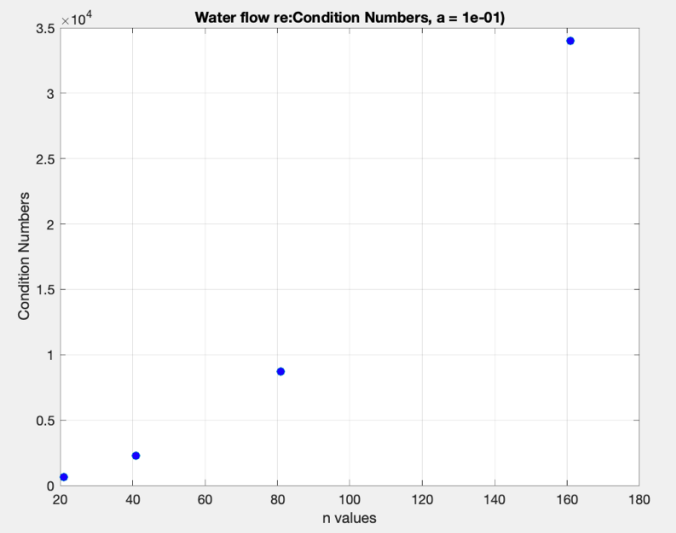
$$\begin{bmatrix} -H_1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \\ 0 \\ -aH_r \end{bmatrix} = \frac{1}{\Delta x^2} \begin{bmatrix} -2 & 1 & & & & & & & & & \\ 1 & -2 & 1 & & & & & & & & \\ & 1 & -2 & & 1 & & & & & & \\ & & \ddots & \ddots & & \ddots & & & & & \\ & & & 1 & -(1+a) & a & & & & & \\ & & & & \ddots & & \ddots & & \ddots & & \\ & & & & & a & & -2a & a & & \\ & & & & & & & a & -2a & a & \\ & & & & & & & & a & -2a & \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ \vdots \\ h_i \\ \vdots \\ h_{n-2} \\ h_{n-1} \\ h_n \end{bmatrix}$$

Using different values of *a* and *n*, I wrote a Matlab script (in the file `question_2_part_1.m`) to find the condition number of the diagonal matrix as described above. To supplement this, I also wrote another Matlab function, in `make_diag_matrix.m` to generate the diagonal matrix.

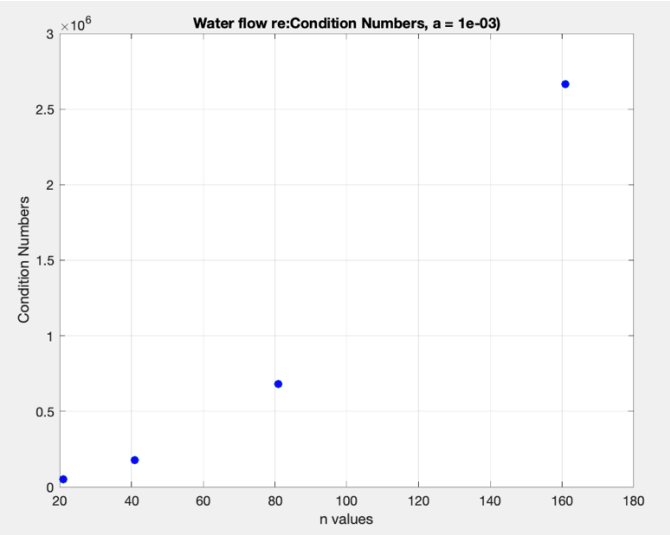
Below are the plots of the condition numbers as *a* changes with respect to n.



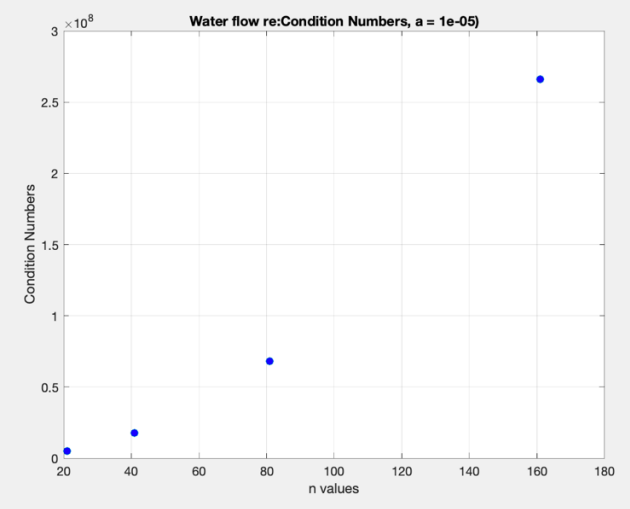
Condition Numbers, a = 1.0



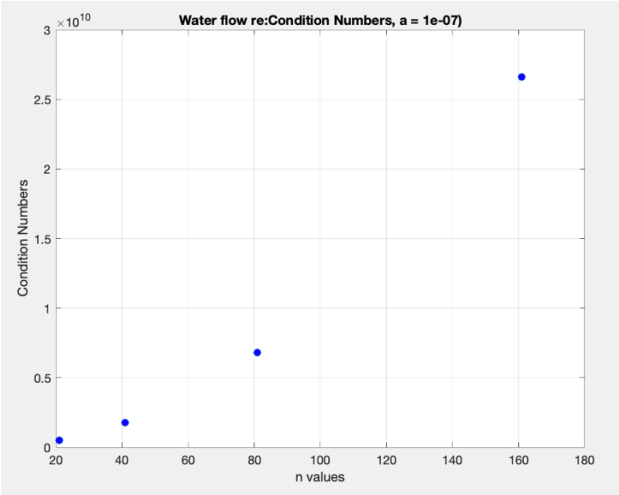
Condition Numbers, a = 1.0e-1



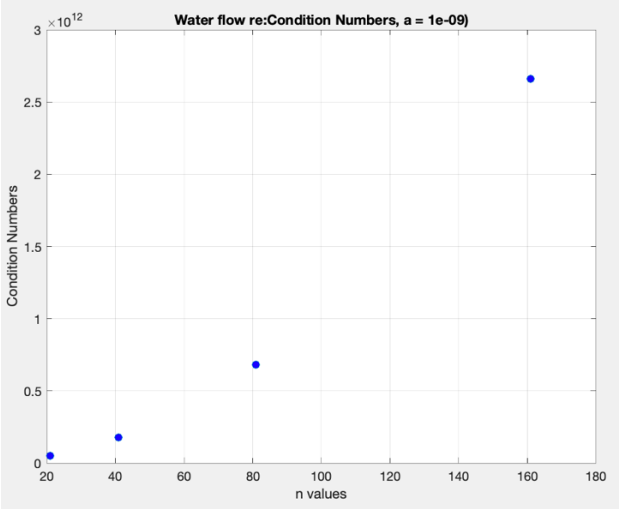
Condition Numbers, a = 1.0e-3



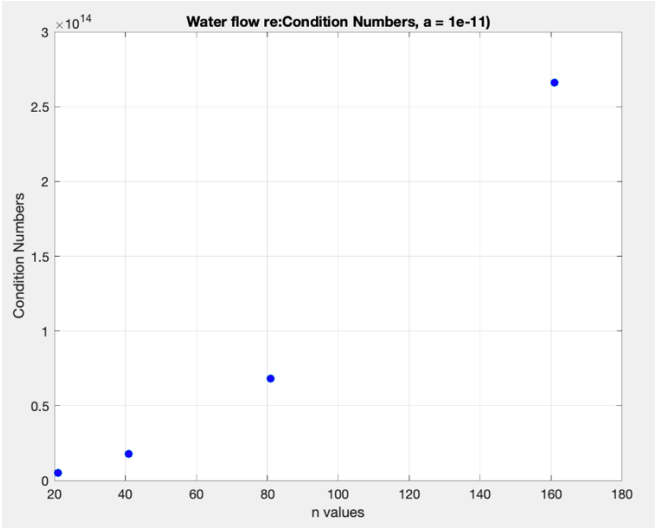
Condition Numbers, a = 1.0e-5



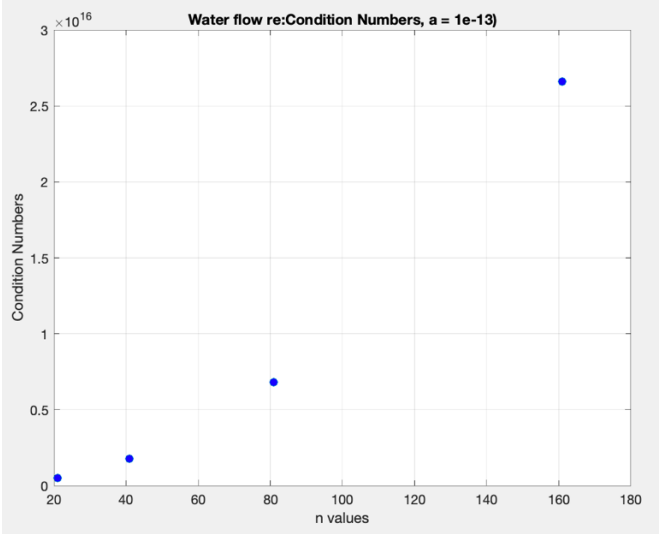
Condition Numbers, a = 1.0e-7



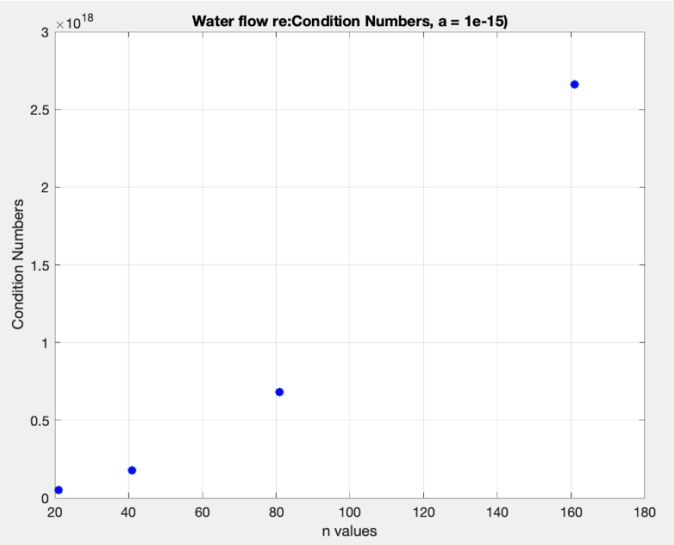
Condition Numbers, a = 1.0e-9



Condition Numbers, a = 1.0e-11



Condition Numbers, a = 1.0e-13



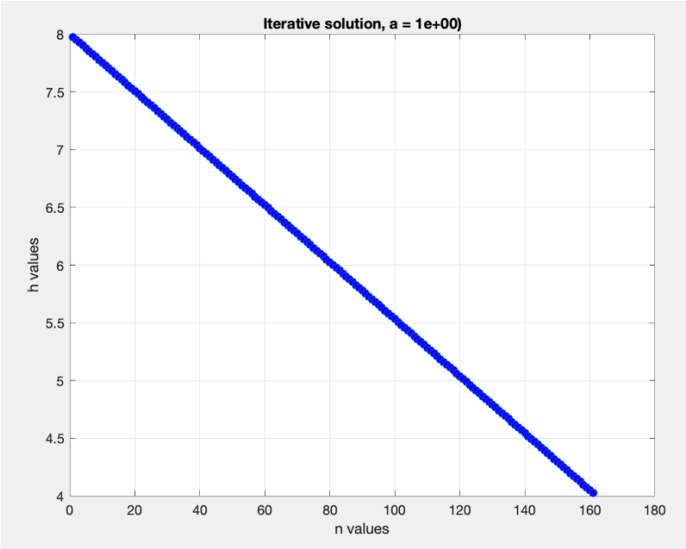
Condition Numbers, *a = 1.0e-15*

All of these graphs look *incredibly* similar. Each of the preceding 9 graphs have the exact same shape. They look so similar, in fact, I didn't think at first that changing the *a* value changed anything. But, each of these graphs is on a completely different order of magnitude.

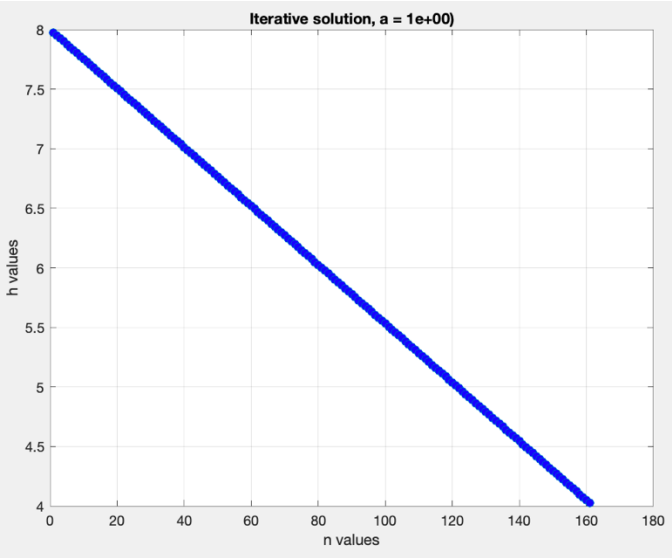
As the values of *a* decrease, the condition numbers go up wildly on the graph, starting from small values as *n* is small and *a* is large, but the numbers go up significantly, up to about 2.7 * 10 ^ 18 when *a* is small and *n* is large. Throughout all the graphs, the condition number ranges from 195.4915 in the smallest case (*a = 1, n = 21)* all the way up to 2.6584e+18 (*a = 1e-15, n = 161)*. The largest factor in determining the condition number seems to be the value of *a* from these tests.

For the second part of this problem, I use the same matrix generation code to iteratively solve the above system of equations described at the top of this problem with varying values of *a*. In my solution, for each value of *a* (1.0, 1.0e-5, 1.0e-15), I setup the problem using the designated values of *a,* and H_1 and H_r. Then, I used a second for loop to solve the system of equations over and over, continually improving the solution. I did these computations 100 times per value of *a* – it doesn't appear that increasing the iteration threshold changes the end values drastically.

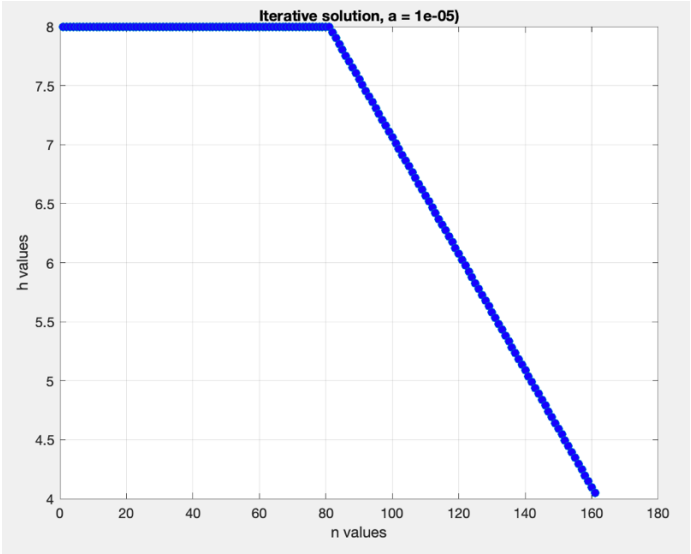
Pictured below are the solutions for each of the three *a* values, obtained after the iterative refinement is completed, as well as the first computed solution before the refinement takes place.



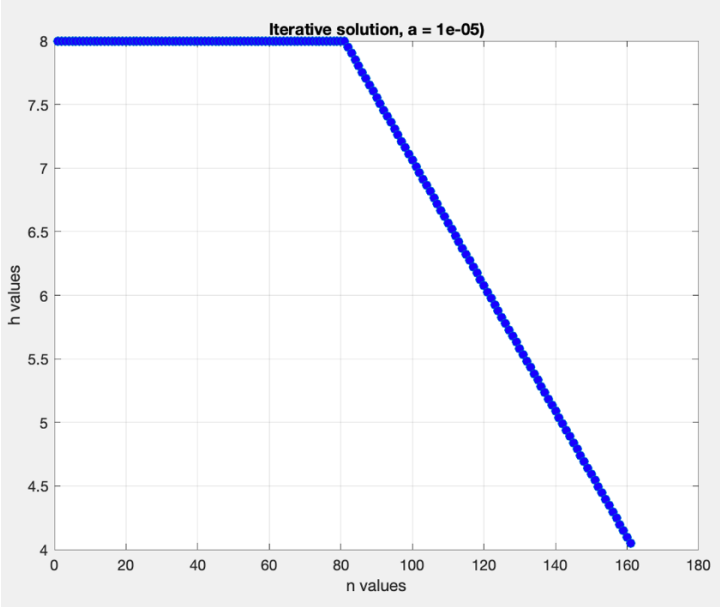
a = 1, first



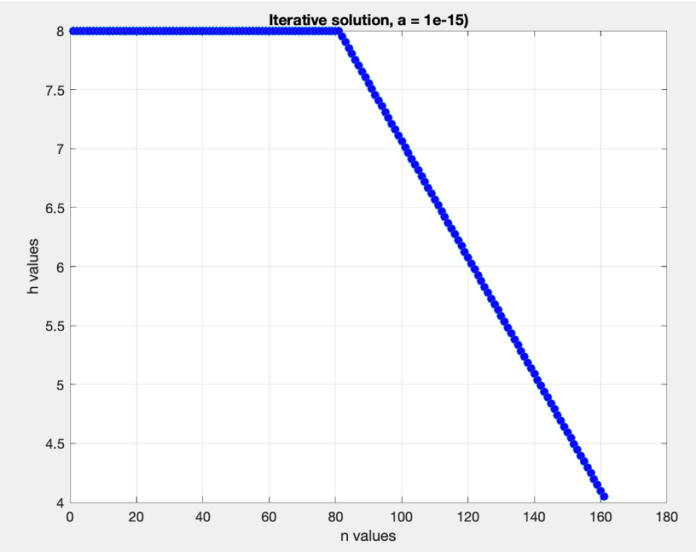
a = 1, final



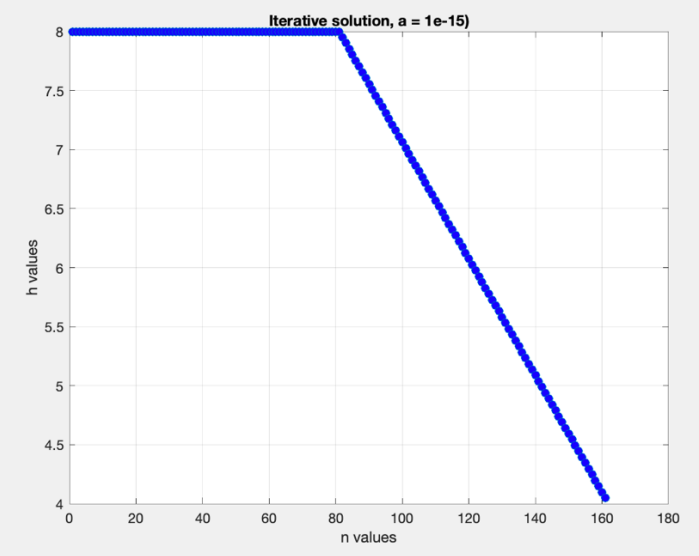
a = 1e-5, first



a = 1e-5, final



a = 1e-15, first



a 1e-15, final

(I wasn't sure of a better way to present the final data for this problem, given the solution to the matrices each have 161 items, so thanks for bearing with the graphs!)

…As you can see, the first solutions before the iterative refinement appear identical to the final solutions, but there *are* minute differences, on the order of magnitude ranging from 1e-13 when *a* is large to 1e-15 when *a* is small. So, over time with refinement, the graphs *do* in fact get more accurate with more computation time! However, there *are* diminishing returns when considering computation time solving these linear systems.

Question 3

For this question, in `question_3.m`, I solved a Vandermonde system of equations in two different ways. Using `linsolve`, Matlab's builtin solver for linear systems, as well as using a third party Vandermonde library.

Starting off, I generated variable SIZE equally spaced points to the function:

$$y = e^x$$

Then, I generated the Vandermonde matrix for a variable SIZE using the `vander` method built in to Matlab. Then, simply, I timed the results for different values of the SIZE variable.
You can see those here:

SIZE	Linsolve time (s)	Vandermonde Library time
100	0.000555	0.002306
200	0.000823	0.002809
300	0.001026	0.003361
400	0.001942	0.004559
500	0.002430	0.005931
600	0.003582	0.007868
700	0.004506	0.009398
800	0.006215	0.011882
900	0.008678	0.015546
1000	0.009505	0.017890

As you can see, Matlab's `linsolve` function tends to be finished in about half the time as the Vandermonde library from the Internet. *However,* they don't tend to be on different orders of complexity. Solving a Vandermonde equation is O(n^3).

O(n^3) == O(2n^3)

So, the two methods of solving a Vandermonde system of equations have the same complexity, but using Matlab's linsolve appears to always be a bit faster. I'm unsure why this is, but it could be because linsolve has efficiencies builtin to the language, given it's an important method within Matlab.

However, in terms of efficiency, I'd argue that one should go with the Vandermonde solver. When SIZE is low, I noticed that the builtin linsolve would produce inaccurate results at times, and the Vandermonde library never seemed to do that. Then, I'd say it's best to use the Vandermonde solving library as opposed to simply naively using linsolve to finish the calculations.

Finally, the condition numbers of the matrices are as follows:

SIZE (N)	A cond	x cond	b cond
100	2.9735e+58	1	1
200	5.4435e+101	1	1
300	8.3378e+144	1	1
400	3.6454e+188	1	1
500	3.6415e+231	1	1
600	4.4160e+274	1	1
700	Inf (too big to store)	1	1
800	Inf (too big to store)	1	1
900	Inf (too big to store)	1	1
1000	Inf (too big to store)	1	1

Talk about ill-conditioned! Here, the condition number of the Vandermonde matrix grows incredibly fast! It almost doesn't tell us anything, really, other than that the Vandermonde matrix is ill-conditioned.

Overall, using the Vandermonde library is the recommended route for solving these. It generates a Vandermonde matrix equivalent to the `vander` function in Matlab, but is more accurate in generating the results than its `linsolve` competitor. They operate on the same order of complexity, too, so the difference in computation time is negligible between the two.