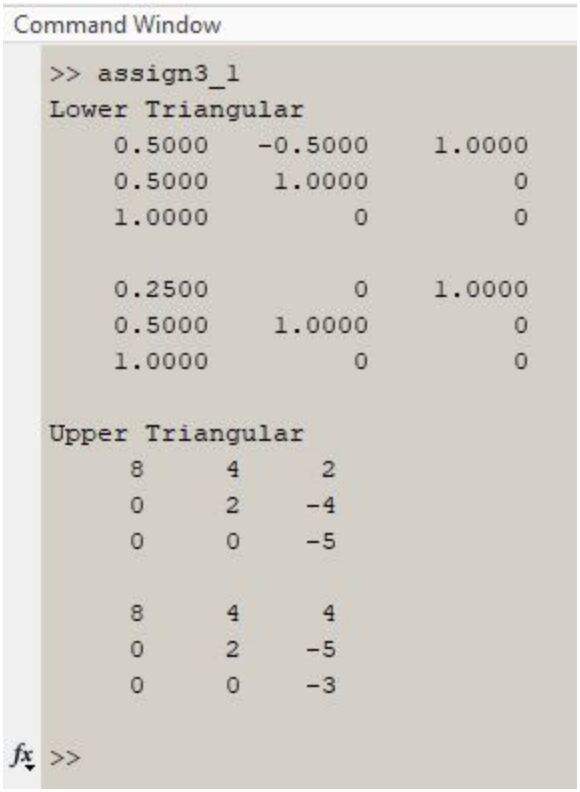
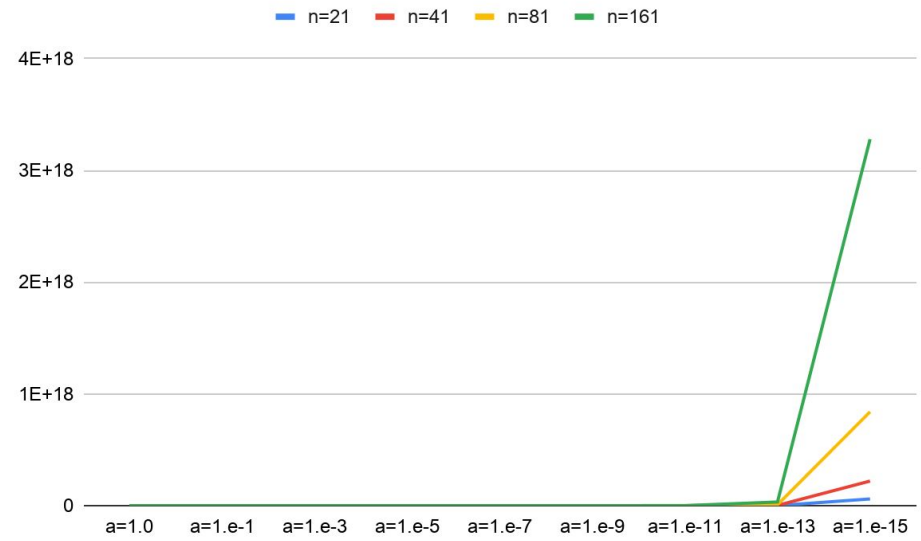
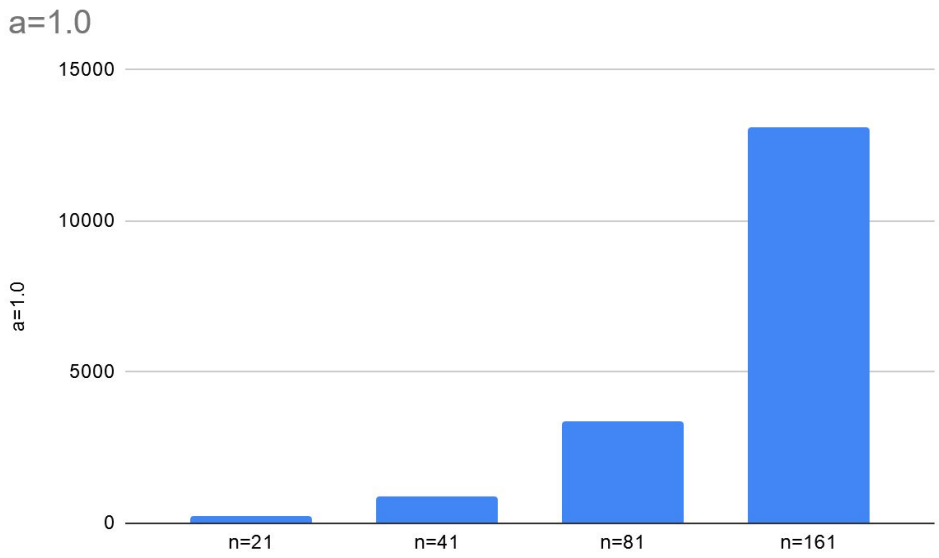
Jakob Horvath
u1092049



Matrix B's LU decomposition appears as the first matrix for the Lower Triangular and Upper Triangular sections, while matrix C is what follows in both cases. Despite being superficially "similar", their decompositions differ slightly for their lower triangular matrices, and the third columns for their upper triangular matrices do not match at all. The two matrices are shown to be more different than what their original setup would imply.





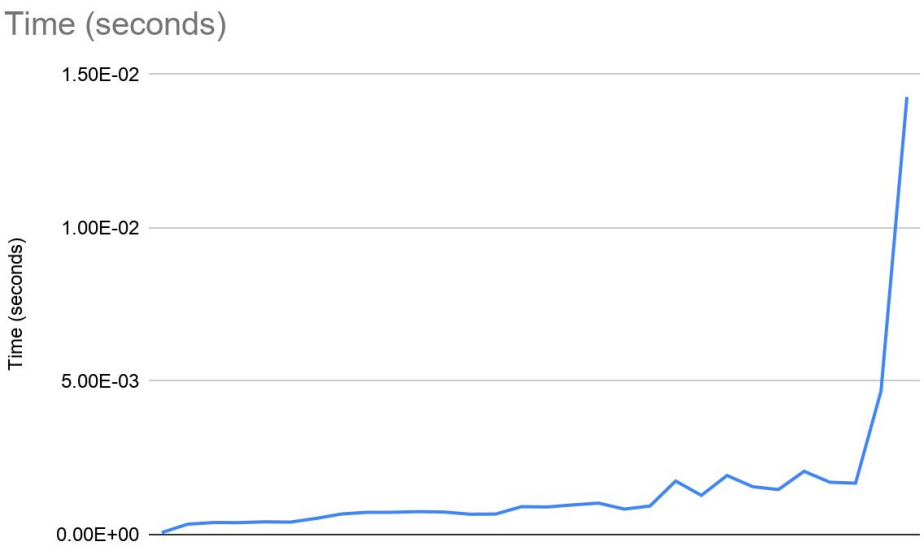
Using Matlab's *cond* function, we can see in the above graphs how ill-conditioned the matrix from question 2 truly is. No condition number is close to 1, and as the value of *a* gets larger the condition number appears to be trending in a quadratic fashion. When *a=1.e-15*, the matrix's condition number is so large that it's reasonable to assume that it is almost noninvertible.

	1	2	3
1	-8.0000	-8.0000	-8
2	0	-1.7764e-15	0
3	1.7764e-15	3.5527e-15	1.7764e-15
4	0	-1.7764e-15	-1.7764e-15
5	-1.7764e-15	0	8.8818e-16
6	0	0	-8.8818e-16
7	0	0	8.8818e-16
8	0	0	-8.8818e-16
9	0	0	0
10	1.7764e-15	1.7764e-15	0
11	-1.7764e-15	-1.7764e-15	0
12	0	0	0
13	0	0	0
14	0	0	0
15	0	0	0
16	0	0	0
17	0	0	0
18	0	0	0
19	1.7764e-15	1.7764e-15	1.7764e-15
20	0	0	0
21	0	0	0
22	3.5527e-15	1.7764e-15	8.8818e-16
23	0	0	2.6645e-15
24	0	0	-1.7764e-15
25	-1.7764e-15	0	0
26	1.7764e-15	0	0
27	-1.7764e-15	-1.7764e-15	0
28	1.7764e-15	1.7764e-15	0

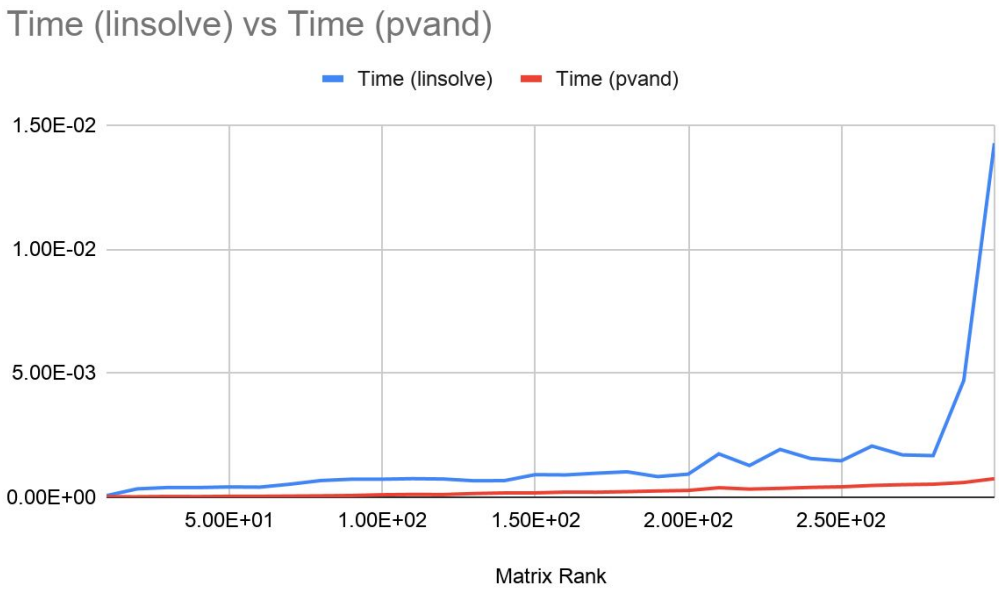
136	0	0	-7.8886e-31
137	-4.4409e-16	0	0
138	0	3.3881e-21	0
139	0	-6.7763e-21	-7.8886e-31
140	-4.4409e-16	0	0
141	0	-3.3881e-21	0
142	-4.4409e-16	-6.7763e-21	0
143	8.8818e-16	0	0
144	0	6.7763e-21	1.5777e-30
145	0	0	-7.8886e-31
146	4.4409e-16	6.7763e-21	7.8886e-31
147	0	0	0
148	0	0	0
149	0	0	0
150	0	0	0
151	-4.4409e-16	-6.7763e-21	-7.8886e-31
152	4.4409e-16	0	7.8886e-31
153	0	6.7763e-21	0
154	-4.4409e-16	0	-7.8886e-31
155	0	0	0
156	4.4409e-16	0	7.8886e-31
157	4.4409e-16	0	0
158	-4.4409e-16	6.7763e-21	0
159	0	-6.7763e-21	0
160	4.4409e-16	6.7763e-21	0
161	-4.0000	-4.0000e-05	-4.0000e-15

The second part of the problem had $n=161$, while $a=1.0$ (column 1), $1.e\text{-}5$ (column 2), and $1.e\text{-}15$ (column 3). After solving the system of equations, I multiplied my estimated vector x by the original matrix, M. The images above show the beginning and end of the result, and do a good job of representing the rest of the computed vector, b. As shown, the vector b is very nearly equal to what the exact solution should be in each case, with extremely small values in some rows where a zero should occupy.

	1
1	4.2412e+07
2	4.4331e+16
3	8.8668e+22
4	4.5627e+26
5	1.3088e+31
6	1.1645e+36
7	1.7480e+39
8	2.8540e+45
9	5.1812e+47
10	1.1972e+54
11	2.9691e+57
12	7.3529e+60
13	1.9352e+66
14	1.0652e+70
15	2.7816e+74
16	1.0967e+80
17	4.0553e+85
18	8.0651e+88
19	2.1472e+94
20	1.1051e+96
21	3.8468e+99
22	4.2474e+104
23	2.9358e+108
24	7.8323e+113
25	4.5876e+117
26	2.5813e+122
27	2.9020e+128
28	7.1510e+130
29	2.2278e+135
30	1.8082e+140



Using the Vandermonde equations to compute a polynomial approximation for e^x turns out to be a pretty bad idea. As seen on the left, the condition numbers only get worse as the rank of the matrix goes from 10 to 300 (incrementing by 10). Even when the rank is equal to 10, the condition number makes clear that the solution to a linear system involving this matrix is likely to contain large numerical errors. The *linsolve* function stays pretty consistent in it's timing data, until the last few entries appear to jump up in an exponential manner.



The *pvand* function courtesy of John Burkardt at https://people.sc.fsu.edu/%7Ejburkardt/f_src/vandermonde/vandermonde.html is clearly a faster function than *linsolve*. While *linsolve* approaches exponential complexity as the condition number gets worse, *pvand* stays relatively consistent and appears linear. However, just as with *linsolve*, *pvand* also produces wildly inaccurate results when dealing with such an ill-conditioned matrix and should therefore not be relied on in this situation.