Section 1-4, Implementation Discussion
Discuss each problem and describe what your code demonstrates.

Solution: All values presented in this discussion come from output.txt. Should more information or digits be desired, please reference this text file. To ensure readability I've truncated the approximations. This output.txt contains all text generated by my executable Linal.exe and contains all requested answers for sections 1-5. Now to expand on each section's woes.

1. Untarring the provided file was pretty straight forward, not much worthy to report.

2. Though section 2 had "basic" in the title, it was not a cakewalk. This is primarily due to the numerous syntax mistakes I made - being a little rusty in Fortran had me fumbling a bit. For example, with my euclidNorm function I fell into the classic pitfall of exponentiating by 1/2, which fell to integer division and evaluated to 0 every time. This understandably created many ones in my outputs. Other than that, the module's implementation went relatively smoothly.

   The driver however was a bit of a headache. This is because it was the first time we needed to actually read, store, and work with Amat and Bmat. While much of the hard reading portions were provided (thank you), I was unclear on how access this information. Eventually this was overcame, but this foreshadows the implementation woes to come.

3. Moving into the good stuff, we examine Gaussian elimination. Initially I thought this was the most straight forward implementation. After struggling with a handful of index errors producing garbage values, I thought I had the answer. As it turns out, I was permuting rows and columns in the wrong way. This still produced an uncomfortably accurate solution, with my error matrix being impressively small. Ian and I determined this was due to the well conditioning of the system, coupled with the invertibility of Amat. Once I ensured I was finding the pivot correctly, that is locating the max in our first column, the next sneaky error was making sure I didn't loop over rows that had been already cleared. In doing this, zeros that had been created were being replaced by values. With Gaussian elimination completed by finding a max in a column, pivoting it to the respective diagonal, then clearing the lower entries we finally have a row reduced upper triangular matrix A. This means we've taken the original form of A,

$$A = \begin{bmatrix} 2.00 & 1.00 & 1.00 & 0.00 \\ 4.00 & 3.00 & 3.00 & 1.00 \\ 8.00 & 7.00 & 9.00 & 5.00 \\ 6.00 & 7.00 & 9.00 & 8.00 \end{bmatrix}$$

   and reduced it to the following form after Gaussian Elimination:

$$\begin{bmatrix} 8.00 & 7.00 & 9.00 & 5.00 \\ 0.00 & 1.75 & 2.25 & 4.25 \\ 0.00 & 0.00 & -0.857 & -0.286 \\ 0.00 & 0.00 & 0.00 & 0.667 \end{bmatrix}$$

   Then comes the back substitution, which was surprisingly painless. If $A \in \mathbb{R}^{m \times m}$, we do the usual $A_{m,m}x_m = b_m$ for column vectors x, b in our respective $AX = B$ system. This is slick since after Gaussian elimination the upper triangular matrix obviously has only one entry in the mth row, and as such the above is solvable. With one equation in one unknown solved, we can move up to the next row. Substituting in our new found solution, we turn one equation in two unknowns into a solvable equation in one unknown. Continuing this process, iterating from the bottom of our reduced matrix to the top we back solve for each unknown variable in column vector x. This was implemented by doing the above looped over each column in B. Calling this in the driver went fine, but I realised that my A being overwritten meant I would have to either call the read function again for my next section or save A as another variable to be worked with later. I went with the latter option. With all that done,

we have our solution matrix, which solves $AX = B$:

$$\begin{bmatrix} 4.440E-016 & 3.500 & 0.2499 & -2.005 & 360.2699 & 10.630 \\ 1.000 & -6.000 & -0.500 & 33.400 & -1234.359 & -22.327 \\ 2.000 & -1.000 & 1.000 & -28.490 & 515.919 & 4.207 \\ -3.000 & 5.000 & -1.500 & 3.689 & 223.039 & 7.124 \end{bmatrix}$$

To show how good of an approximation our numerical methods are, we compute the residual matrix $E = AX - B$ and find

$$E = \begin{bmatrix} 0.000 & 3.552E-015 & 0.000 & -1.7765E-014 & -1.091E-013 & -3.552E-015 \\ 8.881E-016 & -1.776E-015 & 0.000 & 3.552E-015 & 3.552E-013 & 1.776E-015 \\ 0.000 & 0.000 & 0.000 & 0.000 & 5.684E-014 & 0.00 \\ 0.000 & 4.440E-016 & 0.000 & 0.000 & 0.000 & 0.000 \end{bmatrix}$$

Which are all noticeably either zero or very close to machine epsilon.

4. Finally, LU decomposition. Since this used partial pivoting, a substantial amount of the Gaussian elimination code was able to be reused. One key difference is the implementation of the permutation vector, something that would be needed in the back sub process. I think the permutation vector is a very clever idea, simply initialize it to be integers 1 to m, then permute it in the same way that A is permuted so that the back substitution is possible. Namely, we'll have to set our solution matrix's jth row to be the row in B corresponding to the jth entry in s. Though a mouth full, this trick captures all needed info permutation wise.

   We compute the $L_{j,j}$ with a simple relation of entries, specifically $L = M^{-1}$ where M is as defined in equation (2.54) of the lecture notes. This L is a lower triangular matrix with ones along the main diagonal, though we actually store it in A since the Gaussian elimination portion has cleared that part and thus is known to be zero. All this together results in an equivalent system, $LU = PA$, with A identical to that one would receive from Gaussian elimination. Thus we solve $Ax = b$ by actually solving $Ux = y$, where $y = L^{-1}Pb$. This back substitution is a bit more involved than that of Gaussian elimination, this time we must initialize with the permutation vector, do forward substitution ($y = L^{-1}Pb$) then back substitution ($Ux = y$). Again, I did this for all column vectors at once. With this process, we find

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.75 & 1 & 0 & 0 \\ 0.50 & -0.296 & 1 & 0 \\ 0.25 & -0.429 & 0.333 & 1 \end{bmatrix}$$

and

$$U = \begin{bmatrix} 8.00 & 7.00 & 9.00 & 5.00 \\ 0.00 & 1.75 & 2.25 & 4.25 \\ 0.00 & 0.00 & -0.857 & -0.286 \\ 0.00 & 0.00 & 0.00 & 0.667 \end{bmatrix}$$

We can eyeball that this is correct as our U is precisely A after Gaussian elimination. Further, it is indeed true that $LU = PA$ for permutation matrix P. After back substitution we find an almost identical solution matrix, but with some minor differences. Each entry differs from the solution matrix from Gaussian elimination by no more than $1E-14$. This is a little alarming, as the methods should be equivalent, but I imagine since the exact same computations are not performed in both some extraneous rounding error has come into play. Regardless, the norms of the columns of the error matrix after LU back substitution are at most $8.19E-13$, but as small as $4.57E-16$. Both are pretty close to machine epsilon.

Section 5
Consider the following three points in 3D space, with corresponding (x, y, z) coordinates

$$A(1,2,3), B(-3,2,5), C(\pi, e, -\sqrt{2})$$

Explain what equations you need to solve to find the equation of the plane passing through all three points, and do so numerically, using either the Gaussian elimination routine, or the LU decomposition routine (as you prefer). Use Python or Matlab to produce a 3D figure with a graph of this plane, as well as the three points clearly marked.

Solution: Much like a line is determined by two points, a plane is determined by three. Ultimately we are looking for an equation of the form

$$ax + by + cz = d$$

We have three points, which gives us three equations in four unknowns. Namely,

$$\begin{aligned} a + 2b + 3c &= d \\ -3a + 2b + 5c &= d \\ \pi a + eb - \sqrt{2}c &= d \end{aligned} \tag{1}$$

Where a, b, c represent unknown coefficients to solve for. Note that with three unknowns and four equations, we will have an infinite number of solutions. That is, d is a free variable. If I were to solve analytically, I would leave as d a variable, substitute the varying equations into one another until I had an equation only in d, then pick a value for d since it is a free variable. The resultant equation would be one representation of our plane, though all representations are ultimately equal - just scaled by constants.

On the other hand, if we wanted to solve this numerically clearly we cannot leave d as a variable. So let's just pick a value at the start. After all, we only need find one representation of our plane. To that end, take $d = 1$.

Then our system has become

$$\begin{aligned} a + 2b + 3c &= 1 \\ -3a + 2b + 5c &= 1 \\ \pi a + eb - \sqrt{2}c &= 1 \end{aligned} \tag{2}$$

Which may be represented in matrix form as

$$\begin{bmatrix} 1 & 2 & 3 \\ -3 & 2 & 5 \\ \pi & e & -\sqrt{2} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{3}$$

A system very capable of being solved numerically. Doing this, we'll find a, b, c such that

$$ax + by + cz = 1$$

is satisfied by all three of the points A, B, C. For graphing, we'll rewrite the above as

$$z = \frac{1}{c} - \frac{a}{c}x - \frac{b}{c}y$$

Running the FORTRAN Gaussian elimination subroutine with partial pivoting, then back substituting, yields

$$a = 0.0390336, \;\; b = 0.363382, \;\; c = 0.0780671,$$

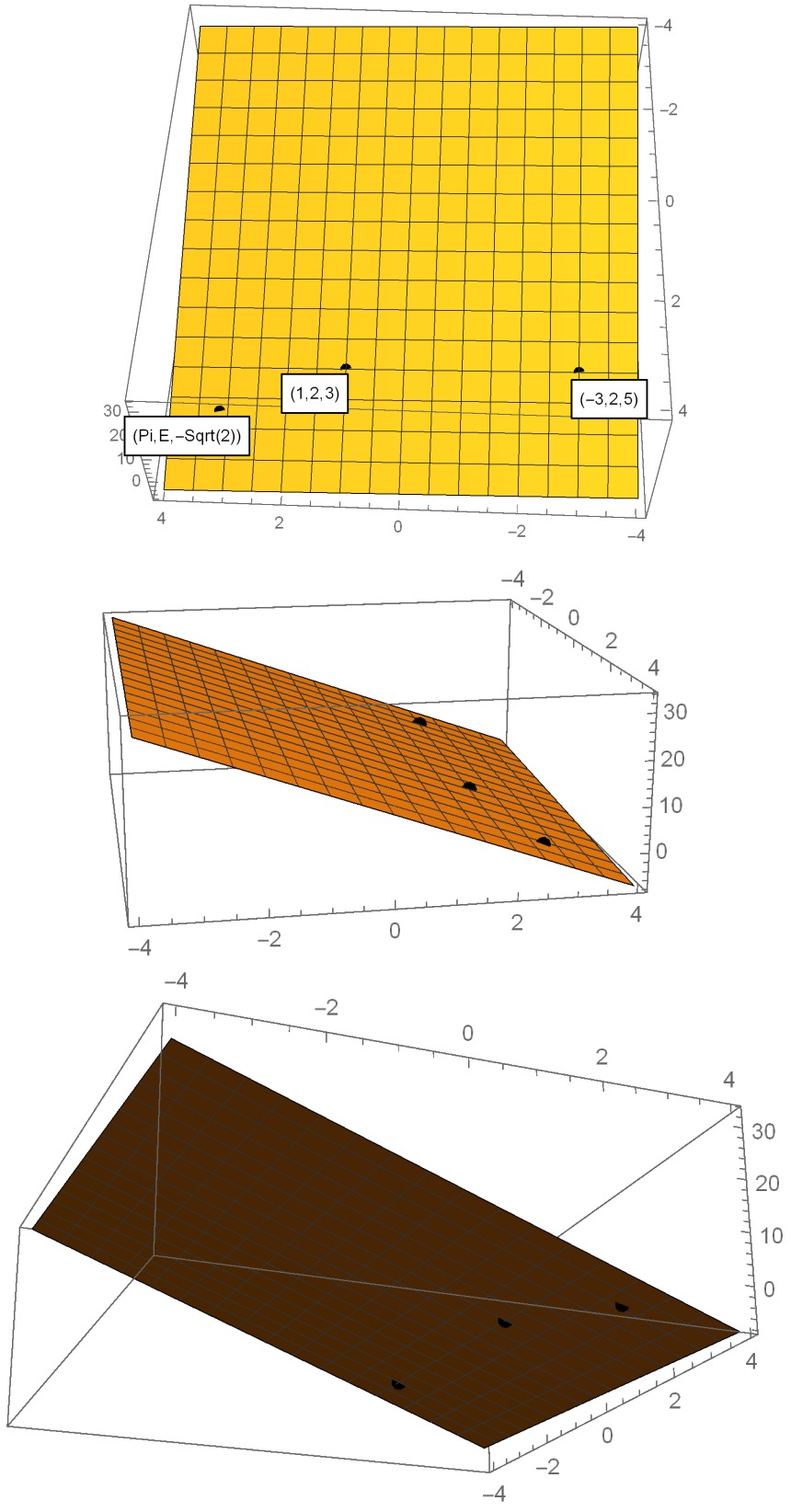
and thus one representation of the equation of the plane in question is

$$0.0390336x + 0.363382y + 0.0780671z = 1$$

plugging in the points $A(1,2,3), B(-3,2,5), C(\pi, e, -\sqrt{2})$ all yield 1, as desired.

[please see figures found on the next page...]

3D plots of plane, contains 3 given points



figures generated with Mathematica, code provided in submission as planePlot.nb