Linear Matrix Equations – Part 2

Mark A. Austin

University of Maryland

austin@umd.edu
ENCE 201, Fall Semester 2023

October 27, 2023

Overview

Numerical Solution

of

Linear Matrix Equations

Mathematical Viewpoint

Mathematical Viewpoint

Solutions to linear equations is given by theoretical results from linear algebra, namely:

- A unique solution $\{X\} = [A^{-1}] \cdot \{B\}$ exists when $[A^{-1}]$ exists (i.e., $\det [A] \neq 0$).
- The equations are inconsistent when $[A]$ is singular and $\text{rank}\,[A|B] \neq \text{rank}\,[A]$.
- If $\text{rank}\,[A|B]$ equals $\text{rank}\,[A]$, then there are an infinite number of solutions.

We can use these results to guide the design of numerical algorithms for computing solutions to linear equations.

Engineering Viewpoint

Direct Methods:

- Transform the original equations into equivalent equations that can be solved more easily.
- Exact answer results if there are no round-off errors. Finite work. Use on dense matrices containing few zeros.
- Gausss Elimination, LU Decomposition.

Iterative Methods:

- Start with a guess of the the solution and repeatedly refine the solution until convergence criteria are achieved.
- Needs infinite work to get an exact answer. No round-off errors. Use on sparse, large order systems.
- Gauss-Seidel Iteration, Jacobi's Iteration Method.

Engineering Viewpoint

Evaluation Criteria:

- Robustness (i.e., Do the solvers work? When will they work?)
- Accuracy and Efficiency (i.e., How does computatioal effort vary as a function of problem size?).
- Ease of Implementation (i.e., implementation speed and cost).

Common Problems:

- Is A singular (or not)?
- Matrix A might be nearly singular. This is important for finite-precision arithmetic.

The pathway from mathematical analysis to a software implementation requires consideration of ill-conditioned equations and finite precision arithmetic.

Ill-Conditioned Equations

Definition. A system of equations is said to be ill-conditioned if small changes to one or more coefficients of $[A]\,[X] = [B]$ causes large deviations in the solution.

Example: The following pairs of almost parallel equations:

$$\begin{bmatrix} 1.00 & 2.00 \\ 0.48 & 0.99 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3.00 \\ 1.47 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \qquad (1)$$

Let's increment coefficient a_{21} by 2%:

$$\begin{bmatrix} 1.00 & 2.00 \\ 0.49 & 0.99 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3.00 \\ 1.47 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}. \qquad (2)$$

Finite-Precision Arithmetic

Example 1. Compute a numerical solution to:

$$\begin{bmatrix} 1.0 \times 10^{-3} & 1.0 \\ 1.0 & 1.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 2.0 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \approx \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}. \quad (3)$$

with a computer having 2 digits of precision (mantissa + floating point + rounded).

Method 1: Transform the matrix equations into echelon form.

Step 1: Compute row multiplier m_{21}:

$$m_{21} = \left[\frac{a_{21}}{a_{11}}\right] = \frac{1}{1.0 \times 10^{-3}} = 1.0 \times 10^{3}. \quad (4)$$

Finite-Precision Arithmetic

Step 2: Compute row operation: $r_2 \rightarrow r_2 - m_{21} r_1$.

$$
\begin{aligned}
(1 - 1000 \times 1.0) x_2 &= (2.0 - 1.0 \times 1000) \\
-999 x_2 &= -998 \\
-1.0 \times 10^3 x_2 &= -1.0 \times 10^3 \\
x_2 &= 1.0 \\
x_1 &= 0.0
\end{aligned}
$$

This is completely the wrong answer! The source of the problem is the large multiplier m_{12}.

Finite-Precision Arithmetic

Method 2: Idea: Swap rows 1 and 2. Then transform the matrix equations into echelon form.

Step 1: Compute row multiplier m_{21}:

$$m_{21} = \left[\frac{a_{21}}{a_{11}} \right] = \frac{1.0 \times 10^{-3}}{1.0} = 10^{-3}. \tag{5}$$

Step 2: Compute row operation: $r_2 \rightarrow r_2 - m_{21} r_1$.

$$\begin{aligned} (1 - 0.001 \times 1.0)x_2 &= (1.0 - 0.0001 \times 2.0) \\ 0.999x_2 &= 0.998 \\ 1.0x_2 &= 1.0 \\ x_2 &= 1.0 \end{aligned}$$

Finite-Precision Arithmetic

Step 3: Backsubstitution:

$$x_1 = 1.0 \tag{6}$$

To two decimal places of accuracy $[x_1, x_2] = [1.0, 1.0]$ is correct.

The absolutely correct answer is $[x_1, x_2] = [1.001, 0.999]$.

Recommended Strategy

- Rearrange rows so that the largest coefficient in absolute value in the column is in the pivotal position.
- This process is called partial pivoting.
- Can also row equilibrate – scale individual rows so maximum element value is 1.

Gauss Elimination

(No Pivoting)

Gauss Elimination (No Pivoting)

Elementary Row Operations

- Interchange any two rows.
- Multiply any row by a non-zero number.
- Add to one equation a non-zero multiple of another equation.

Step-by-Step Procedure

- Write $[A]\,[x] = [B]$ in augmented matrix form.
- Apply a sequence of row operations to transform the augmented matrix into echelon form.
- Use back substitution to compute the solution vector.
- Validate answer.

Gauss Elimination (No Pivoting)

Example 1. Consider the family of matrix equations:

$$\begin{bmatrix} 3 & -6 & 7 \\ 9 & 0 & -5 \\ 5 & -8 & 6 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ -4 \end{bmatrix} \tag{7}$$

Step 1. Augmented Matrix Form

$$\left[\begin{array}{ccc|c} 3 & -6 & 7 & 3 \\ 9 & 0 & -5 & 3 \\ 5 & -8 & 6 & -4 \end{array}\right] \tag{8}$$

Gauss Elimination (No Pivoting)

Step 2. Apply Row Operations

Step 2.1: Divide Row 1 by 3 ($r_1 \to r_1/3$)

$$\left[\begin{array}{ccc|c} 3 & -6 & 7 & 3 \\ 9 & 0 & -5 & 3 \\ 5 & -8 & 6 & -4 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & -2 & 7/3 & 1 \\ 9 & 0 & -5 & 3 \\ 5 & -8 & 6 & -4 \end{array}\right] \tag{9}$$

Step 2.2: Subtract 9 times row 1 from row 2 ($r_2 \to r_2 - 9r_1$) Step 2.3: Subtract 5 times row 1 from row 3 ($r_3 \to r_3 - 5r_1$)

$$\left[\begin{array}{ccc|c} 1 & -2 & 7/3 & 1 \\ 9 & 0 & -5 & 3 \\ 5 & -8 & 6 & -4 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & -2 & 7/3 & 1 \\ 0 & 18 & -26 & -6 \\ 0 & 2 & -5.667 & -9 \end{array}\right] \tag{10}$$

Gauss Elimination (No Pivoting)

Step 2.4: Divide row 2 by 18 ($r_2 \to r_2/18$)

Step 2.5: Subtract 2 times row 2 from row 3 ($r_3 \to r_3 - 2r_2$)

Step 2.6: Divide row 3 by 2.777 ($r_3 \to r_3/2.777$)

$$\left[\begin{array}{ccr|r} 1 & -2 & 7/3 & 1 \\ 0 & 18 & -26 & -6 \\ 0 & 2 & -5.667 & -9 \end{array}\right] \to \left[\begin{array}{ccr|r} 1 & -2 & 7/3 & 1 \\ 0 & 1 & -26/18 & -1/3 \\ 0 & 0 & 1 & 3 \end{array}\right] \quad (11)$$

Equation 11 is now in echelon form.

Gauss Elimination (No Pivoting)

Step 3. Compute Solution via Back Substitution

Step 3.1: Row 3: $x_3 = 3.0 \rightarrow x_3 = 3.0$.

Step 3.2: Row 2: $x_2 - 26/18x_3 = -1/3 \rightarrow x_2 = 4.0$.

Step 3.3: Row 1: $x_1 - 2x_2 + 7/3x_3 = 1.0 \rightarrow x_1 = 2.0$.

Step 4. Validate the Answer, $X = [2, 4, 3]$.

$$\begin{bmatrix} 3 & -6 & 7 \\ 9 & 0 & -5 \\ 5 & -8 & 6 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 4 \\ 3 \end{bmatrix} \rightarrow \begin{bmatrix} 3 \\ 3 \\ -4 \end{bmatrix}. \tag{12}$$

Gauss Elimination (No Pivoting)

Python Source Code: Step-by-Step Row Reduction

```
import math
import numpy as np
import LinearMatrixEquations as lme

def main():
    print("--- Step 1: Create (3x3) matrix A, (3x1) matrix B ... ");

    A = np.array([ [ 3, -6,  7],
                   [ 9,  0, -5],
                   [ 5, -8,  6] ]);
    B = np.array([ [ 3],
                   [ 3],
                   [-4] ]);

    lme.printmatrix("A", A);
    lme.printmatrix("B", B);

    print("--- Step 2: Create (3x4) augmented matrix [ A | B ] (join on vertical axis) .

    AB = np.concatenate((A, B), axis=1)
    lme.printmatrix("Augmented Matrix [A|B] ...", AB );

    print("--- Step 3.1: Scale row 1 by 1/3 ... ");

    gauss01 = lme.rowscale(AB, 0, 1.0/3.0)
    lme.printmatrix("Matrix gauss01 ...", gauss01 );
```

Gauss Elimination (No Pivoting)

Python Source Code: Step-by-Step Row Reduction

```
print("--- Step 3.2: Subtract 9 times row 1 from row 2 ... ");

gauss02 = lme.rowadd( gauss01, 0, 1, -9.0)
lme.printmatrix("Matrix gauss02 ...", gauss02 );

print("--- Step 3.3: Subtract 5 times row 1 from row 3 ... ");

gauss03 = lme.rowadd( gauss02, 0, 2, -5.0)
lme.printmatrix("Matrix gauss03 ...", gauss03 );

print("--- Step 3.4: Scale row 2 by 1/18 ... ");

gauss04 = lme.rowscale(gauss03, 1, 1.0/18.0)
lme.printmatrix("Matrix gauss04 ...", gauss04 );

print("--- Step 3.5: Subtract 2 times row 2 from row 3 ... ");

gauss05 = lme.rowadd( gauss04, 1, 2, -2.0)
lme.printmatrix("Matrix gauss05 ...", gauss05 );

print("--- Step 3.6: Scale row 3 by -1/2.77777778 ... ");

echelon01 = lme.rowscale(gauss05, 2, -1.0/2.77777778 )

print("--- Step 4.0: Print matrix [A|B] in Echelon Form ... ");

lme.printmatrix("Matrix [A|B] in Echelon Form ...", echelon01 );
```

Gauss Elimination (No Pivoting)

Abbreviated Output: See TestLinearMatrixEquations01.py

```
--- Step 1: Create (3x3) matrix A, (3x1) matrix B ...

Matrix: A
   3.0000000e+00   -6.0000000e+00    7.0000000e+00
   9.0000000e+00    0.0000000e+00   -5.0000000e+00
   5.0000000e+00   -8.0000000e+00    6.0000000e+00

Matrix: B
   3.0000000e+00
   3.0000000e+00
  -4.0000000e+00

--- Step 2: Create (3x4) augmented matrix [ A | B ] (join on vertical axis) ...

Matrix: Augmented Matrix [A|B] ...
   3.0000000e+00   -6.0000000e+00    7.0000000e+00    3.0000000e+00
   9.0000000e+00    0.0000000e+00   -5.0000000e+00    3.0000000e+00
   5.0000000e+00   -8.0000000e+00    6.0000000e+00   -4.0000000e+00
```

Gauss Elimination (No Pivoting)

Abbreviated Output: Continued ...

```
--- Step 3.1: Scale row 1 by 1/3 ...

Matrix: Matrix gauss01 ...
   1.0000000e+00  -2.0000000e+00   2.3333333e+00   1.0000000e+00
   9.0000000e+00   0.0000000e+00  -5.0000000e+00   3.0000000e+00
   5.0000000e+00  -8.0000000e+00   6.0000000e+00  -4.0000000e+00

--- Step 3.2: Subtract 9 times row 1 from row 2 ...
--- Step 3.3: Subtract 5 times row 1 from row 3 ...
--- Step 3.4: Scale row 2 by 1/18 ...
--- Step 3.5: Subtract 2 times row 2 from row 3 ...
--- Step 3.6: Scale row 3 by -1/2.77777778 ...
--- Step 4.0: Print matrix [A|B] in Echelon Form ...

Matrix: Matrix [A|B] in Echelon Form ...
   1.0000000e+00  -2.0000000e+00   2.3333333e+00   1.0000000e+00
   0.0000000e+00   1.0000000e+00  -1.4444444e+00  -3.3333333e-01
  -0.0000000e+00  -0.0000000e+00   1.0000000e+00   3.0000000e+00
```

Gauss Elimination

(with Partial Pivoting)

Gauss Elimination (with Partial Pivoting)

Step-by-Step Procedure:

- Find largest element in absolute value in 1st column. Interchange rows so that it is the pivotal equation.
- Compute multipliers: $m_{i1} = [a_{i1}/a_{11}]$ for i = 2, 3, $\cdots$ n.
- Compute new coefficients in rows 2, 3, $\cdots$ n.

  For example, $a_{22}^{'} = a_{22} - m_{21}a_{12}$, $a_{23}^{'} = a_{23} - m_{21}a_{13}$, $a_{32}^{'} = a_{32} - m_{31}a_{12}$, etc.

  The partially transform matrix is:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{'} & a_{23}^{'} \\ 0 & a_{32}^{'} & a_{33}^{'} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2^{'} \\ b_3^{'} \end{bmatrix}. \qquad (13)$$

Gauss Elimination (with Partial Pivoting)

Step-by-Step Procedure:

- Repeat process for 2nd column.
- After (n-1) column calculations the A matrix will be in upper-triangular (echelon) form.
- Solve for unknowns through backsubstitution.

Key Points:

- If the pivotal coeffient is small, then the row multiplier will be very large, and errors magnified due to finite arithmetic.
- Row reduction and back substitution require $O(n^3/3)$ and $O(n^2)$ operations, respectively.

Gauss Elimination (with Partial Pivoting)

Example 2. Consider the following set of equations:

$$\begin{aligned} 2x_1 + 6x_2 - x_3 &= -12 \\ 5x_1 - x_2 + 2x_3 &= 29 \\ -3x_1 - 4x_2 + x_3 &= 5 \end{aligned}$$

Steps 1 and 2: Augmented matrix form, then row equilibration.

$$\left[\begin{array}{rrr|r} 2 & 6 & -1 & -12 \\ 5 & -1 & 2 & 29 \\ -3 & -4 & 1 & 5 \end{array}\right] \rightarrow \left[\begin{array}{rrr|r} 2/6 & 1 & -1/6 & -2.0 \\ 1 & -1/5 & 2/5 & 29/5 \\ -3/4 & -1 & 1/4 & 5/4 \end{array}\right]. \tag{14}$$

Gauss Elimination (with Partial Pivoting)

Step 3: Partial pivoting (swap rows 1 and 2):

$$\left[\begin{array}{rrr|r} 1 & -1/5 & 2/5 & 29/5 \\ 2/6 & 1 & -1/6 & -2.0 \\ -3/4 & -1 & 1/4 & 5/4 \end{array}\right] \tag{15}$$

Step 4: Compute row multipliers:

$$m_{21} = [a_{21}/a_{11}] = 2/6, \quad m_{31} = [a_{31}/a_{11}] = -3/4. \tag{16}$$

Step 5: Compute new coefficients for rows 2 and 3:

$$\left[\begin{array}{lrr|r} 1 & -1/5 & 2/5 & 29/5 \\ 0 & 32/30 & -9/30 & -118/30 \\ 0 & -23/20 & 11/20 & 118/20 \end{array}\right] \tag{17}$$

Gauss Elimination (with Partial Pivoting)

Step 6: Compute multiplier for row 3, columm 2.

$$m_{32} = [a_{32}/a_{22}] = -64/69. \tag{18}$$

Step 7: Eliminate elements below diagonal in column 2.

$$\left[\begin{array}{ccc|c} 1 & -1/5 & 2/5 & 29/5 \\ 0 & 32/30 & -9/30 & -118/30 \\ 0 & 0 & 145/690 & 870/690 \end{array}\right] \tag{19}$$

Step 8: Solve for x_1 through x_3 via back substitution.

$$x_3 = [870/690] \,/\, [145/690] = 6.0, \;\; x_2 = -2.0, \;\; x_1 = 3.0. \tag{20}$$

Step 9: Check solution by substitution.

Gauss Elimination (with Partial Pivoting)

Python Source Code: Row Reduction + Back Substitution

```
# ===============================================================================
# TestLinearMatrixEquations02.py: Gauss Elimination with row reduction, partial
# pivoting, and back substitution.
# ===============================================================================

import math
import numpy as np
import LinearMatrixEquations as lme

def main():
    n = 3  # <--- size of the matrix equations ...

    print("--- Problem 1: Create (3x3) matrix A, (3x1) matrix B ... ");

    A = np.array([ [ 3, -6,  7],
                   [ 9,  0, -5],
                   [ 5, -8,  6] ]);

    B = np.array([ [ 3], [ 3], [-4] ]);

    lme.printmatrix("A", A);
    lme.printmatrix("B", B);

    print("--- Step 2: Compute augmented matrix equations ... ");

    AB = np.concatenate((A, B), axis=1)
    lme.printmatrix("Augmented Matrix [A|B] ...", AB );
```

Gauss Elimination (with Partial Pivoting)

Python Source Code: Continued ...

```python

    print("--- Step 3: Compute row reduction operations to echelon form ... ");

    echelon01 = lme.rowreduction( AB );
    lme.printmatrix("Matrix echelon01 ...", echelon01 );

    print("--- Step 4: Split echelon01 back to (nxn) piece and (nx1) pieces ... ");

    B_reduced = echelon01[:,n:n+1]
    A_reduced = echelon01[:,0:n]

    lme.printmatrix("Matrix A (reduced) ...", A_reduced );
    lme.printmatrix("Matrix B (reduced) ...", B_reduced );

    print("--- Step 5: Compute back substitution ... ");

    soln01 = lme.backsubstitution( A_reduced, B_reduced)
    lme.printmatrix("Matrix X (solution) ...", soln01 );

    print("--- Problem 2: Create (3x3) matrix A, (3x1) matrix B ... ");

    A = np.array([ [  2,  6, -1],
                   [  5, -1,  2],
                   [ -3, -4,  1] ]);

    B = np.array([ [-12], [ 29], [  5] ]);

    lme.printmatrix("A", A);
    lme.printmatrix("B", B);
```

Gauss Elimination (with Partial Pivoting)

Python Source Code: Continued ...

```
    print("--- Step 2: Compute augmented matrix equations ... ");

    AB = np.concatenate((A, B), axis=1)
    lme.printmatrix("Augmented Matrix [A|B] ...", AB );

    print("--- Step 3: Compute row reduction operations to echelon form ... ");

    echelon01 = lme.rowreduction( AB );
    lme.printmatrix("Matrix echelon01 ...", echelon01 );

    print("--- Step 4: Split echelon01 back to (nxn) piece and (nx1) pieces ... ");

    B_reduced = echelon01[:,n:n+1]
    A_reduced = echelon01[:,0:n]

    lme.printmatrix("Matrix A (reduced) ...", A_reduced );
    lme.printmatrix("Matrix B (reduced) ...", B_reduced );

    print("--- Step 5: Compute back substitution ... ");

    soln01 = lme.backsubstitution( A_reduced, B_reduced)
    lme.printmatrix("Matrix X (solution) ...", soln01 );

# call the main method ...

main()
```

Gauss Elimination (with Partial Pivoting)

Abbreviated Output: Problem 1 ...

```
Matrix: A                                                    Matrix: B
   3.0000000e+00   -6.0000000e+00    7.0000000e+00             3.0000000e+00
   9.0000000e+00    0.0000000e+00   -5.0000000e+00             3.0000000e+00
   5.0000000e+00   -8.0000000e+00    6.0000000e+00            -4.0000000e+00

--- Step 2: Compute augmented matrix equations ...

Matrix: Augmented Matrix [A|B] ...
   3.0000000e+00   -6.0000000e+00    7.0000000e+00    3.0000000e+00
   9.0000000e+00    0.0000000e+00   -5.0000000e+00    3.0000000e+00
   5.0000000e+00   -8.0000000e+00    6.0000000e+00   -4.0000000e+00

--- Step 3: Compute row reduction operations to echelon form ...

Matrix: Matrix echelon01 ...
   1.0000000e+00   -2.0000000e+00    2.3333333e+00    1.0000000e+00
   0.0000000e+00    1.0000000e+00   -1.4444444e+00   -3.3333333e-01
  -0.0000000e+00   -0.0000000e+00    1.0000000e+00    3.0000000e+00
```

Gauss Elimination (with Partial Pivoting)

Abbreviated Output: Problem 1 continued ...

```
--- Step 4: Split echelon01 back to (nxn) piece and (nx1) pieces ...

Matrix: Matrix A (reduced) ...

   1.0000000e+00   -2.0000000e+00    2.3333333e+00
   0.0000000e+00    1.0000000e+00   -1.4444444e+00
  -0.0000000e+00   -0.0000000e+00    1.0000000e+00

Matrix: Matrix B (reduced) ...
   1.0000000e+00
  -3.3333333e-01
   3.0000000e+00

--- Step 5: Compute back substitution ...

Matrix: Matrix X (solution) ...
   2.0000000e+00
   4.0000000e+00
   3.0000000e+00
```

Gauss Elimination (with Partial Pivoting)

Abbreviated Output: Problem 2 ...

```
Matrix: A                                                    Matrix: B
   2.0000000e+00    6.0000000e+00   -1.0000000e+00           -1.2000000e+01
   5.0000000e+00   -1.0000000e+00    2.0000000e+00            2.9000000e+01
  -3.0000000e+00   -4.0000000e+00    1.0000000e+00            5.0000000e+00

--- Step 2: Compute augmented matrix equations ...

Matrix: Augmented Matrix [A|B] ...
   2.0000000e+00    6.0000000e+00   -1.0000000e+00   -1.2000000e+01
   5.0000000e+00   -1.0000000e+00    2.0000000e+00    2.9000000e+01
  -3.0000000e+00   -4.0000000e+00    1.0000000e+00    5.0000000e+00

--- Step 3: Compute row reduction operations to echelon form ...

Matrix: Matrix echelon01 ...
   1.0000000e+00    3.0000000e+00   -5.0000000e-01   -6.0000000e+00
  -0.0000000e+00    1.0000000e+00   -2.8125000e-01   -3.6875000e+00
   0.0000000e+00    0.0000000e+00    1.0000000e+00    6.0000000e+00
```

Gauss Elimination (with Partial Pivoting)

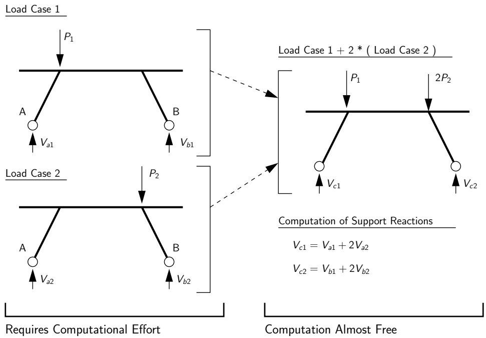
Abbreviated Output: Problem 2 continued ...

```
--- Step 4: Split echelon01 back to (nxn) piece and (nx1) pieces ...

Matrix: Matrix A (reduced) ...
   1.0000000e+00    3.0000000e+00   -5.0000000e-01
  -0.0000000e+00    1.0000000e+00   -2.8125000e-01
   0.0000000e+00    0.0000000e+00    1.0000000e+00

Matrix: Matrix B (reduced) ...
  -6.0000000e+00
  -3.6875000e+00
   6.0000000e+00

--- Step 5: Compute back substitution ...

Matrix: Matrix X (solution) ...
   3.0000000e+00
  -2.0000000e+00
   6.0000000e+00
```

LU Decomposition

LU Decomposition

Motivation. Suppose AX = B needs to be solved for a multiplicity of B vectors (e.g., A $X_1 = B_1$, A $X_2 = B_2$, etc).



Solving AX = B requires $O(n^3)$ computational work. Can we do better? ...

LU Decomposition

Objective. We want to solve $[A]\,[X] = [B]$, where A, X, and B are $(n \times n)$, $(n \times 1)$ and $(n \times n)$ matrices, respectively.

Idea. Factor $[A]$ into the product:

$$[A] = [L]\,[U] \to [L]\,[U]\,[X] = [B]\,. \tag{21}$$

and then solve in two steps.

Step 1: Foward Substitution to get $Z = [z_1, z_2, \cdots, z_n]$.

$$\begin{bmatrix} L_{11} & 0 & \cdots & 0 \\ L_{21} & L_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ L_{n1} & L_{n2} & \cdots & L_{nn} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}. \tag{22}$$

LU Decomposition

Step 2: Back Substitution to get $X = [x_1, x_2, \cdots, x_n]$.

$$\begin{bmatrix} U_{11} & U_{12} & \cdots & U_{1n} \\ 0 & U_{22} & \cdots & U_{2n} \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & U_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}. \qquad (23)$$

Observation: There are n^2 elements in matrix A, and $n(n+1)$ unknowns in equations 22 and 23.

We need to impose n additional constraints to make the method work.

LU Decomposition

Row Reduction Strategies:

- Assume upper diagonal elements are unity, i.e., $U_{ii} = 1$ (Crout Reduction).
- Assume $L_{ii} = I$ (Doolittle's Method).
- Assume $U_{ii} = L_{ii}$ (Cholesky Decomposition).

Step-by-Step Solution Procedure:

- Compute A = LU (this requires $O(n^3)$ work).
- Solve LU $X_1 = B_1$ (only requires $O(n^2)$ work).
- Solve LU $X_2 = B_2$ (only requires $O(n^2)$ work).
- Solve LU $X_3 = B_3$ (only requires $O(n^2)$ work), etc ...

LU Decomposition

General Equations. For a $(n \times n)$ set of equations $[L]\,[U] = [A]$,

$$a_{ij} = \sum_{k=min(i,j)}^{n} \left(L_{ik} U_{kj} \right). \tag{24}$$

Three cases:

$$i \le j \quad U_{ij} = \left[A_{ij} - \sum_{k=1}^{i-1} L_{ik} U_{kj} \right] / U_{ii}$$

$$i = j \quad U_{jj} = \left[A_{ij} - \sum_{k=1}^{j-1} L_{ik} U_{kj} \right] / L_{jj}$$

$$i \ge j \quad L_{ij} = \left[A_{ij} - \sum_{k=1}^{j-1} L_{ik} U_{kj} \right] / U_{jj}$$

LU Decomposition

Example 1. Consider the matrix equations:

$$\begin{bmatrix} 3 & -6 & 7 \\ 9 & 0 & -5 \\ 5 & -8 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ -4 \end{bmatrix}. \tag{25}$$

Assume $[L][U] = [A]$ with values of unity along the upper diagonal (Crout Reduction).

$$\begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} 1 & U_{12} & U_{13} \\ 0 & 1 & U_{23} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & -6 & 7 \\ 9 & 0 & -5 \\ 5 & -8 & 6 \end{bmatrix}. \tag{26}$$

LU Decomposition

Equating terms (1st row of L):

$$
\begin{aligned}
L_{11} \cdot 1 = 3 \quad &\rightarrow \quad L_{11} = 3. \\
L_{11} \cdot U_{12} = -6 \quad &\rightarrow \quad U_{12} = -2. \\
L_{11} \cdot U_{13} = -7 \quad &\rightarrow \quad U_{13} = 7/3.
\end{aligned}
$$

Equating terms (2nd row of L):

$$
\begin{aligned}
L_{21} \cdot U_{11} = 9 \quad &\rightarrow \quad L_{21} = 9. \\
L_{21} \cdot U_{12} + L_{22} \cdot 1 = 0 \quad &\rightarrow \quad L_{22} = 18. \\
L_{21} \cdot U_{13} + L_{22} \cdot U_{23} = 0 \quad &\rightarrow \quad U_{23} = -13/9.
\end{aligned}
$$

LU Decomposition

Equating terms (3rd row of L):

$$\begin{aligned} L_{31} \cdot U_{11} = 5 \quad &\rightarrow \quad L_{31} = 5. \\ L_{31} \cdot U_{12} + L_{32} = -8 \quad &\rightarrow \quad L_{32} = 2. \\ L_{31} \cdot U_{13} + L_{32} \cdot U_{23} + L_{33} = 6 \quad &\rightarrow \quad L_{33} = -25/9. \end{aligned}$$

Hence:

$$\begin{bmatrix} 3 & 0 & 0 \\ 9 & 18 & 0 \\ 5 & 2 & -25/9 \end{bmatrix} \begin{bmatrix} 1 & -2 & 7/3 \\ 0 & 1 & -13/9 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & -6 & 7 \\ 9 & 0 & -5 \\ 5 & -8 & 6 \end{bmatrix}. \tag{27}$$

LU Decomposition

Forward Substitution:

$$\begin{bmatrix} 3 & 0 & 0 \\ 9 & 18 & 0 \\ 5 & 2 & -25/9 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ -4 \end{bmatrix} \to \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1.0 \\ -1/3 \\ 3.0 \end{bmatrix}. \tag{28}$$

Backward Substitution:

$$\begin{bmatrix} 1 & -2 & 7/3 \\ 0 & 1 & -13/9 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ -1/3 \\ 3.0 \end{bmatrix} \to \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 4.0 \\ 3.0 \end{bmatrix}. \tag{29}$$

LU Decomposition

Example 2. Consider $[L]\,[U] = [A]$ with values of unity along the lower diagonal (Doolittle's Method).

$$\begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix} = \begin{bmatrix} 3 & -6 & 7 \\ 9 & 0 & -5 \\ 5 & -8 & 6 \end{bmatrix}. \quad (30)$$

Can show:

$$\begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 5/3 & 1/9 & 1 \end{bmatrix} \begin{bmatrix} 3 & -6 & 7 \\ 0 & 18 & -26 \\ 0 & 0 & -25/9 \end{bmatrix} = \begin{bmatrix} 3 & -6 & 7 \\ 9 & 0 & -5 \\ 5 & -8 & 6 \end{bmatrix}. \quad (31)$$

Working Examples

Example 3: LU Decomposition with Symbolic Python

Problem Statement: Let's try to use SymPy to compute algebraic representations for LU decomposition for matrices

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = LU, \tag{32}$$

and

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = LU. \tag{33}$$

Then, we will check the results by assigning numerical values to the symbols to match equation 25. Note: det(A) = det(L).det(U) = -150.

Example 3: LU Decomposition with Symbolic Python

Python Source Code: Compute $A = LU$.

```
# ===============================================================
# TestMatrixLUDecomposition01.py: Compute symbolic descriptions
# of matrix LU decomposition ....
# ===============================================================

import sympy as sp
from sympy import Integral, Matrix, pi, pprint

def main():
    # Define symbolic representation of matrix ...

    print("--- Part 1: Compute LU decomposition of 2x2 matrix equations ...");

    a, b, c, d, e, f = sp.symbols('a,b,c,d,e,f')

    A = sp.Matrix(([a,b],[c,d]))
    B = sp.Matrix([e,f])

    pprint(A)

    L, U, P = A.LUdecomposition()

    print("--- Lower triangular matrix ...\n");

    pprint(L)

    print("--- Upper triangular matrix ...\n");
```

Example 3: LU Decomposition with Symbolic Python

Python Source Code: Continued ...

```

    pprint(U)

    print("--- Check matrix product L*U ...\n");

    pprint(L*U)

    print("--- Part 2: Symbolic LU decomposition of 3x3 matrix equations ...");

    g, h, i = sp.symbols('g,h,i')
    j, k, l = sp.symbols('j,k,l')

    A = sp.Matrix(([a,b,c],[d,e,f], [g,h,i] ))
    B = sp.Matrix([j,k,l])

    pprint(A)

    L, U, P = A.LUdecomposition()

    print("--- Lower triangular matrix ...\n");

    pprint(L)

    print("--- Upper triangular matrix ...\n");

    pprint(U)
```

Example 3: LU Decomposition with Symbolic Python

Python Source Code: Continued ...

```
    print("--- Check matrix product L*U ...\n");

    pprint(L*U)

    print("--- Part 3: LU decomposition for sample 3x3 matrix equations ...");

    print("--- Set values in L ...");
    print("---   L = L.subs( {a:3, b:-6, c:7, d:9, e:0, f:-5, g:5, h:-8, i:6 } ) \n")

    L = L.subs( {a:3, b:-6, c:7, d:9, e:0, f:-5, g:5, h:-8, i:6 } )

    pprint(L)

    print("--- L.det() = {:s} ...".format( str( sp.det(L) ) ))

    print("--- Set values in U ...");
    print("---   U = U.subs( {a:3, b:-6, c:7, d:9, e:0, f:-5, g:5, h:-8, i:6 } ) \n")

    U = U.subs( {a:3, b:-6, c:7, d:9, e:0, f:-5, g:5, h:-8, i:6 } )

    pprint(U)

    print("--- U.det() = {:s} ...".format( str( sp.det(U) ) ))

# call the main method ...

main()
```

Example 3: LU Decomposition with Symbolic Python

Abbreviated Output: Part 1: decomposition of 2x2 matrices

```
| a   b |
|       |
| c   d |

--- Lower triangular matrix    --- Upper triangular matrix

| 1   0 |                      | a      b     |
|       |                      |              |
| c     |                      |        b.c   |
| -   1 |                      | 0   d - ---  |
| a     |                      |        a     |

--- Check matrix product L*U ...

| a   b |
|       |
| c   d |
```

Example 3: LU Decomposition with Symbolic Python

Abbreviated Output: Part 2: decomposition of 3x3 matrices

```
| a  b  c |
|         |
| d  e  f |
|         |
| g  h  i |

--- Lower triangular matrix    --- Upper triangular matrix ...

| 1     0     0 |         | a     b                   c                  |
|               |         |                                            |
| d             |         |        b.d                 c.d             |
| -     1     0 |         | 0  e - ---             f - ---             |
| a             |         |         a                   a              |
|               |         |                                            |
|        b.g    |         |                 (    c.d) (    b.g)        |
|    h - ---    |         |                 (f - ---).(h - ---)        |
| g       a     |         |                 (     a ) (     a )   c.g  |
| -  -------  1 |         | 0     0     i - ------------------- - ---  |
| a       b.d   |         |                       b.d              a   |
|     e - ---   |         |                   e - ---                  |
|          a    |         |                        a                   |
```

Example 3: LU Decomposition with Symbolic Python

Abbreviated Output: Part 2: decomposition of 3x3 matrices

```
| a   b   c |   # <--- Check matrix product L*U ...
|           |
| d   e   f |
|           |
| g   h   i |

--- Check results with sample (3x3) matrix (see equation 25) ...

--- Set values in L ...                     --- Set values in U ...
--- L = L.subs( { a:3, b:-6,   c:7,         --- U = U.subs( { a:3, b:-6,   c:7,
                  d:9,   e:0,  f:-5,                          d:9,   e:0,  f:-5,
                  g:5, h:-8,   i:6 } )                        g:5, h:-8,   i:6 } )

|  1    0    0 |                            |  3   -6     7   |
|              |                            |                 |
|  3    1    0 |                            |  0   18    -26  |
|              |                            |                 |
| 5/3  1/9   1 |                            |  0    0   -25/9 |

--- L.det() = 1,                            --- U.det() = -150
```

Example 3: LU Decomposition with Symbolic Python

Points to note:

- Theoretical analysis indicates:

$$det(A) = det(L)det(U) = 1 * -150 = -150. \tag{34}$$

  The symbolic analysis works!

- The analysis only works when the parameter "a" is non-zero.
- To avoid division-by-zero, we need a way to shuffle the matrix rows.

Example 4: LU Decomposition with Row Permutations

Problem Statement: Compute LU decomposition with row permutation matrix P, i.e.,

$$A = PLU, \tag{35}$$

then solve matrix equations 25:

Note: Here, P is an orthogonal matrix (i.e., $P^T = P^{-1}$). Thus, solving AX = B translates to LUX = P^T B.

Solution Procedure:

- Create matrices A and B.
- Use scipy.linalg to compute A = PLU.
- Solve LY = P^T B (forward substitution).
- Solve UX = Y (backward substitution).

Example 4: LU Decomposition with Row Permutations

Python Source Code:

```
# ================================================================================
# TestLUDecomposition01.py: Use scipy.linalg to compute LU decomposition with
# row permutation matrix P.
# ================================================================================

import math
import numpy as np
import scipy.linalg as la
import LinearMatrixEquations as lme

def main():
    print("--- Problem 1: Create (3x3) matrix A, (3x1) matrix B ... ");

    print("--- Step 1: Define A and B matrices ... ");

    A = np.array([ [ 3, -6,  7], [ 9,  0, -5], [ 5, -8,  6] ]);
    B = np.array([ [3], [3], [-4] ]);

    lme.printmatrix("A", A);
    lme.printmatrix("B", B);

    print("--- Step 2: Compute A = PLU Decomposition ... ");

    P, L, U = la.lu(A)

    lme.printmatrix("P", P);
    lme.printmatrix("L", L);
```

Example 4: LU Decomposition with Row Permutations

Python Source Code: Continued ...

```
    lme.printmatrix("U", U);

    print("--- Step 3: Check decomposition A = PLU ... ");

    lme.printmatrix("np.dot(P,np.dot(L,U))", np.dot(P,np.dot(L,U)) );

    print("--- Step 4: Compute Z = P^T.B (note: matrix P is orthogonal) ... ");

    Z = np.dot(P.T,B);
    lme.printmatrix("Z", Z);

    print("--- Step 5: Compute forward substitution ... ");

    Y = la.solve_triangular(L, Z, lower=True)

    print("--- Step 6: Compute backward substitution ... ");

    X = la.solve_triangular(U, Y, lower=False)

    lme.printmatrix("Solution: X", X);

# call the main method ...

main()
```

Example 4: LU Decomposition with Row Permutations

Abbreviated Output: ...

```
--- Problem 1: Create (3x3) matrix A, (3x1) matrix B ...
--- Step 1: Define A and B matrices ...

Matrix: A                                    Matrix: B
   3.0e+00 -6.0e+00  7.0e+00                    3.0e+00
   9.0e+00  0.0e+00 -5.0e+00                    3.0e+00
   5.0e+00 -8.0e+00  6.0e+00                   -4.0e+00

--- Step 2: Compute A = PLU Decomposition ...

Matrix: P
   0.0e+00  0.0e+00  1.0e+00
   1.0e+00  0.0e+00  0.0e+00
   0.0e+00  1.0e+00  0.0e+00

Matrix: L                                    Matrix: U
   1.0e+00  0.0e+00  0.0e+00                    9.0e+00  0.0e+00 -5.0e+00
   5.5e-01  1.0e+00  0.0e+00                    0.0e+00 -8.0e+00  8.7e+00
   3.3e-01  7.5e-01  1.0e+00                    0.0e+00  0.0e+00  2.0e+00
```

Example 4: LU Decomposition with Row Permutations

Abbreviated Output: Continued ...

```
--- Step 3: Check decomposition A = PLU ...

Matrix: np.dot(P,np.dot(L,U))
   3.0000000e+00   -6.0000000e+00    7.0000000e+00    <---  It works !!!
   9.0000000e+00    0.0000000e+00   -5.0000000e+00
   5.0000000e+00   -8.0000000e+00    6.0000000e+00

--- Step 4: Compute Z = P^T.B (note: matrix P is orthogonal) ...

Matrix: Z
   3.0000000e+00
  -4.0000000e+00
   3.0000000e+00

--- Steps 5-6: Compute forward/backward substitution ...

Matrix: Solution: X   <---  It works !!!
   2.0000000e+00
   4.0000000e+00
   3.0000000e+00
```

Python Code Listings

Code 1: Matrix Row Operations for Gauss Elimination

```
# ======================================================================
# LinearMatrixEquations.py: Functions to compute operations on linear matrix
#                           equations.
# ======================================================================

import math
import numpy as np

# ==================================================================
# LinearMatrixEquations.printmatrix(): Print two-dimensional matrices.
#
# Args: name: string description of matrix.
#       A (nxn) matrix.
#
# Returns: void.
# ==================================================================

def printmatrix(name, a):
    print("");
    print("Matrix: {:s} ".format(name) );
    for row in a:
        for col in row:
            print("{:16.7e}".format(col), end=" ")
        print("")

# ===============================================================
# LinearMatrixEquations.rowswap(): Create duplicate of matrix, then
#                                  swap rows.
```

Code 1: Matrix Row Operations for Gauss Elimination

```
#
# Args: A (nxn) numpy array.
#       k: row to be swapped.
#       l: row to be swapped.
#
# Returns: B (nxn) copy of A with rows swapped.
# =================================================================

def rowswap(A,k,l):
    m = A.shape[0]   # m is number of rows in A
    n = A.shape[1]   # n is number of columns in A

    B = np.copy(A).astype('float64')

    for j in range(n):
        temp = B[k][j]
        B[k][j] = B[l][j]
        B[l][j] = temp

    return B

# ==================================================================
# LinearMatrixEquations.rowscale(): Create duplicate of matrix, then
#                                   scale specified row.
#
# Args: A (nxn) numpy array.
#     : k (int: is matrix row number to be scaled.
#     : scale (double): scale factor.
```

Code 1: Matrix Row Operations for Gauss Elimination

```
#
# Returns: B (nxn) copy of A with row k multiplied by scale ...
# ==================================================================

def rowscale(A,k,scale):

    m = A.shape[0]   # m is number of rows in A
    n = A.shape[1]   # n is number of columns in A

    B = np.copy(A).astype('float64')

    for j in range(n):
        B[k][j] *= scale

    return B

# ======================================================================
# LinearMatrixEquations.rowadd(): Add multiple of a row to another row.
#
# Args: A (nxn) numpy array.
#     : k (int: is matrix row number to be added to.
#     : l (int: is matrix row number to be scaled.
#     : scale (double): scale factor.
#
# Returns: B (nxn) copy of A with row k multiplied by scale ...
# ======================================================================

def rowadd(A,k,l,scale):
    m = A.shape[0]   # m is number of rows in A
    n = A.shape[1]   # n is number of columns in A
```

Code 1: Matrix Row Operations for Gauss Elimination

```
    B = np.copy(A).astype('float64')

    for j in range(n):
        B[l][j] += B[k][j]*scale

    return B
```

Code 2: Gauss Elimination with Partial Pivoting

```
# ========================================================================
# LinearMatrixEquations.py: Functions to compute operations on linear matrix
#                           equations.
# ========================================================================

import math
import numpy as np

# ===========================================================================
# LinearMatrixEquations.rowreduction(): Computes row reduction to echelon form.
#
# Args: A: an augmented matrix of dimension n x (n+1) associated with a
#          linear system.
#
# Returns: B, a numpy array that represents the row echelon form of A.
#
# Note: RowReduction may not return correct results if the the matrix A does
# not have a pivot in each column (i.e., the matrix is rank deficient).
# ===========================================================================

def rowreduction(A):

    m = A.shape[0]   # A has m rows
    n = A.shape[1]   # It is assumed that A has m+1 columns

    B = np.copy(A).astype('float64')

    # For each step of elimination, we find a suitable pivot, move it into
```

Code 2: Gauss Elimination with Partial Pivoting

```
    # position and create zeros for all entries below.

    for k in range(m):
        # Set pivot as (k,k) entry
        pivot = B[k][k]
        pivot_row = k

        # Find a suitable pivot if the (k,k) entry is zero

        while(pivot == 0 and pivot_row < m-1):
            pivot_row += 1
            pivot = B[pivot_row][k]

        # Swap row if needed

        if (pivot_row != k):
            B = rowswap(B,k,pivot_row)

        # If pivot is nonzero, carry on with elimination in column k

        if (pivot != 0):
            B = rowscale(B,k,1./B[k][k])
            for i in range(k+1,m):
                B = rowadd(B,k,i,-B[i][k])
        else:
            print("Pivot could not be found in column",k,".")

    return B
```

Code 2: Gauss Elimination with Partial Pivoting

```

# ==============================================================================
# LinearMatrixEquations.backsubstitution(): Returns an (nx1) vector that is the
# solution to the matrix system UX = B.
#
# Args: U: A numpy array that represents an upper triangular square mxm matrix.
#       B: A numpy array that represents an nx1 vector
#
# Returns: X, A (nx1) numpy array representing the solution to UX = B.
# ==============================================================================

def backsubstitution(U,B):

    # m is number of rows and columns in U

    m = U.shape[0]
    X = np.zeros((m,1))

    # Calculate entries of X backward from m-1 to 0

    for i in range(m-1,-1,-1):
        X[i] = B[i]
        for j in range(i+1,m):
            X[i] -= U[i][j]*X[j]
        if (U[i][i] != 0):
            X[i] /= U[i][i]
        else:
            print("Zero entry found in U pivot position",i,".")
    return X
```

Code 2: Gauss Elimination with Partial Pivoting

```
# =============================================================================
# LinearMatrixEquations.backsubstitution(): Returns an (nx1) vector that is the
# solution to the matrix system AX = B.
#
# Args: A: A numpy array that represents a matrix of dimension n x n.
#       B: A numpy array that represents a matrix of dimension n x 1.
#
# Returns: X, A (nx1) numpy array representing the solution to AX = B.
#
# Note: Computational procedure will fail if AX = B does not have a
#       unique solution.
# =============================================================================

def solvesystem(A,B):

    # Step 1: Check shape of A

    if (A.shape[0] != A.shape[1]):
        print("--- ERROR: solvesystem accepts only square arrays ...")
        return
    n = A.shape[0]  # n is number of rows and columns in A

    # Step 2: Join A and B to make the augmented matrix

    A_augmented = np.hstack((A,B))

    # Step 3: Carry out elimination
```

Code 2: Gauss Elimination with Partial Pivoting

```

    R = rowreduction(A_augmented)

    # Step 4: Split R back to nxn piece and nx1 piece

    B_reduced = R[:,n:n+1]
    A_reduced = R[:,0:n]

    # Step 5: Compute back substitution

    X = backsubstitution( A_reduced, B_reduced)

    return X
```