CS 3430: S19: SciComp with Py
Lecture 23

Linear Programming in 2 Variables
Part 2

Vladimir Kulyukin
Department of Computer Science
Utah State University

Typo on Slide 9, Lecture 20

As I was writing assignment 11, I saw that I made a typo on Slide 9, Lecture 20.

In both equations on the slide, $+$ should be replaced with -. In other words,

$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$;

$x_n = x_{n-1} - \frac{f(x_n)}{f'(x_{n-1})}$.

In lecture 21, when we reviewed Newton-Raphson, the equation on slide 4 is correct.

Review

Boundaries and Half Planes

Each line of the form $Ax + By \leq C$ or $Ax + By \geq C$ divides the plane into 2 *half planes*.

The line $Ax + By = C$ is the *boundary*.

We can use any point off the specified line $Ax + By = C$ to determine which half plane satisfies the inequality $Ax + By \leq C$ or $Ax + By \geq C$.

Example

The set of points (x, y) that satisfy $3x + 2y \leq 6$ lie below this line. We can use any point not on the line to check if that point satisfies $3x + 2y \leq 6$. For example, since (0, 0) satisfies $3 \cdot 0 + 2 \cdot 0 \leq 6$.

Problem

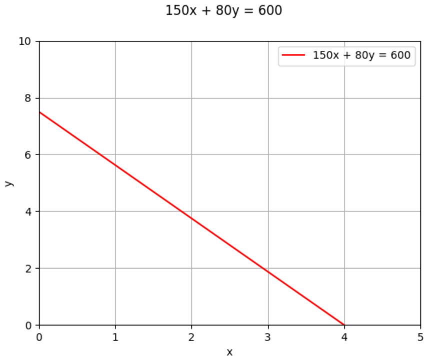
A hiker thinks that on a long hike she needs snacks with at least 600 calories. She plans to take chocolate and raisins. The chocolate has 150 calories per ounce. The raisins have 80 calories per ounce. Find and graph the system of inequalities for this problem and the set that satisfies them.

Solution

Let x be the number of ounces of chocolate and y the number of ounces of raisins. We have the following inequalities to constraint this problem.

1. $x \geq 0$;
2. $y \geq 0$;
3. $150x + 80y \geq 600$.

Solution



The set that satisfies this problem is $x \geq 0 \cap y \geq 0 \cap 150x + 80y \geq 600$.

Constraints, Feasible Sets, and Objective Functions

The constraints we will work with will always be inclusive, i.e., the ineqalities expressed with $\geq$ and $\leq$.

The set of points satisfying the constraints of the problem is known as the **feasible set** of the problem.

The function to be maximized or minimized is known as the **objective function** for the problem.

Basic Matrix Algebra

Matrix: Definition and Notation

A **matrix** is a rectangular array of numbers. Matrices are typically represented by uppercase boldface type letters (e.g., **A**, **B**, **E**). The **order** of the matrix refers to the number of rows and columns of the matrix. An $m \times n$ matrix may be written as

$$\mathbf{A} = [a_{i,j}] = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ . & . & \dots & . \\ . & . & \dots & . \\ . & . & \dots & . \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{bmatrix}$$

Matrix: Definition and Notation

Sometimes the commas are omitted in the subscripts of individual elements.

$$\mathbf{A} = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ . & . & \dots & . \\ . & . & \dots & . \\ . & . & \dots & . \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Matrix Addition

Two matrices of the same order can be added by adding the elements in each corresponding postion. If the matrices are not of the same order, the addition is undefined. Let

$$\mathbf{A} = \begin{bmatrix} 7 & 1 & -2 \\ 3 & 3 & 0 \end{bmatrix} \ \mathbf{B} = \begin{bmatrix} 2 & -3 & 4 \\ 1 & 5 & 9 \end{bmatrix} \ \mathbf{C} = \begin{bmatrix} 2 & 1 \\ 7 & 3 \\ 9 & 2 \end{bmatrix}$$

Then $\mathbf{A} + \mathbf{B} = \begin{bmatrix} 7 & 1 & -2 \\ 3 & 3 & 0 \end{bmatrix} + \begin{bmatrix} 2 & -3 & 4 \\ 1 & 5 & 9 \end{bmatrix} = \begin{bmatrix} 9 & -2 & 2 \\ 4 & 8 & 9 \end{bmatrix} =$ $\mathbf{B} + \mathbf{A}$.

$\mathbf{A} + \mathbf{C}$ is undefined.
$\mathbf{B} + \mathbf{C}$ is undefined.

Matrix Addition

Two matrices of the same order can be added by adding the elements in each corresponding postion. If the matrices are not of the same order, the addition is undefined. Let

$$\mathbf{A} = \begin{bmatrix} 7 & 1 & -2 \\ 3 & 3 & 0 \end{bmatrix} \; \mathbf{B} = \begin{bmatrix} 2 & -3 & 4 \\ 1 & 5 & 9 \end{bmatrix} \; \mathbf{C} = \begin{bmatrix} 2 & 1 \\ 7 & 3 \\ 9 & 2 \end{bmatrix}$$

Then $\mathbf{A} + \mathbf{B} = \begin{bmatrix} 7 & 1 & -2 \\ 3 & 3 & 0 \end{bmatrix} + \begin{bmatrix} 2 & -3 & 4 \\ 1 & 5 & 9 \end{bmatrix} = \begin{bmatrix} 9 & -2 & 2 \\ 4 & 8 & 9 \end{bmatrix} = \mathbf{B} + \mathbf{A}$.

$\mathbf{A} + \mathbf{C}$ is undefined.
$\mathbf{B} + \mathbf{C}$ is undefined.

Matrix Addition with NUMPY

```
A = np.matrix(
       [[7, 1, -2],
        [3, 3, 0]])
B = np.matrix(
       [[2, -3, 4],
        [1, 5, 9]])
C = np.matrix(
        [[2, 1],
         [7, 3],
         [9, 2]])
print('A+B')
print(np.add(A, B))
print('B+A')
print(np.add(B, A))
try:
    print('A+C')
    print(np.add(A, C))
except Exception as e:
    print(e)
```

Matrix Addition with NUMPY: Output

```
A+B
[[ 9 -2  2]
 [ 4  8  9]]
B+A
[[ 9 -2  2]
 [ 4  8  9]]
A+C
operands could not be broadcast with shapes (2,3) (3,2)
```

Matrix Multiplication

Two matrices **A** and **B** may be multipled if they are **conformable**, i.e., if the number of columns of **A** is the same as the number of rows in **B**. If **A** is an $m \times n$ matrix and **B** is an $n \times q$ matrix, then $\mathbf{AB} = \mathbf{C}$ is an $m \times q$ matrix. Each element of **C** is given by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \text{ where}$$

n is the number of columns in **A** (or rows in **B**),
$i = 1, ..., m$ (m is the number of rows in **A**),
$j = 1, ..., q$ (q is the number of columns in **B**).

Matrix Multiplication Examples

Let

$$\mathbf{A} = \begin{bmatrix} 7 & 1 \\ 4 & -3 \\ 2 & 0 \end{bmatrix};\ \mathbf{B} = \begin{bmatrix} 2 & 1 & 7 \\ 0 & -1 & 4 \end{bmatrix};\ \mathbf{C} = \begin{bmatrix} 1 & -1 & 3 \\ 2 & 2 & 3 \\ -1 & 4 & 7 \end{bmatrix};.$$

Then $\mathbf{AB} = \begin{bmatrix} 7 & 1 \\ 4 & -3 \\ 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 & 7 \\ 0 & -1 & 4 \end{bmatrix} = \begin{bmatrix} 14 & 6 & 53 \\ 8 & 7 & 16 \\ 4 & 2 & 14 \end{bmatrix}$.

$\mathbf{AB} = \mathbf{C} = [c_{ij}] = \sum_{k=1}^{2} a_{ik} b_{kj}, i \in [1,3], j \in [1,2]$. Let us compute the first two columns of $\mathbf{C}$.

$c_{11} = \sum_{k=1}^{2} a_{1k} b_{k1} = a_{11} b_{11} + a_{12} b_{21} = 7 \cdot 2 + 1 \cdot 0 = 14.$
$c_{21} = \sum_{k=1}^{2} a_{2k} b_{k1} = a_{21} b_{11} + a_{22} b_{21} = 4 \cdot 2 + (-3) \cdot 0 = 8.$
$c_{31} = \sum_{k=1}^{2} a_{3k} b_{k1} = a_{31} b_{11} + a_{32} b_{21} = 2 \cdot 2 + 0 \cdot 0 = 4.$
$c_{12} = \sum_{k=1}^{2} a_{1k} b_{k2} = a_{11} b_{12} + a_{12} b_{22} = 7 \cdot 1 + 1 \cdot (-1) = 6.$
$c_{22} = \sum_{k=1}^{2} a_{2k} b_{k2} = a_{21} b_{12} + a_{22} b_{22} = 4 \cdot 1 + (-3) \cdot (-1) = 7.$
$c_{32} = \sum_{k=1}^{2} a_{3k} b_{k2} = a_{31} b_{12} + a_{32} b_{22} = 2 \cdot 1 + 0 \cdot (-1) = 2.$

Matrix Multiplication with NUMPY

```
A = np.matrix(
        [[7, 1],
         [4, -3],
         [2, 0]])
B = np.matrix(
        [[2, 1, 7],
         [0, -1, 4]])
AB = np.dot(A, B)
print(AB)
```

Output:

```
[[14  6 53]
 [ 8  7 16]
 [ 4  2 14]]
```

Special Matrices

Identity Matrix

An **identity** matrix is a square matrix (i.e., the number of rows is equal to the number of columns) where all diagonal elements are equal to 1 and the other elements are equal to 0. It is typically denoted as $\mathbf{I}_m$.

$$\mathbf{I}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{I}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Identity Matrices with NUMPY

```
>>> import numpy as np
>>> np.identity(1) ## same as np.eye(1)
array([[ 1.]])
>>> np.identity(2) ## same as np.eye(2)
array([[ 1.,  0.],
       [ 0.,  1.]])
>>> np.identity(3) ## same as np.eye(3)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> np.identity(4) ## same as np.eye(4)
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

Inverse Matrix

The **inverse** of an *n* x *n* matrix $\mathbf{A}$, denoted as $\mathbf{A}^{-1}$, is a matrix such that $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{A}^{-1} \cdot \mathbf{A} = \mathbf{I}$.

If we can find the inverse of some matrix $\mathbf{A}$, then $\mathbf{A}$ is **invertible**; if $\mathbf{A}$ cannot be inverted, it is called **singular**.

Inverting Matrices with NUMPY

```
A = np.array(
        [[1, 4],
         [5, 6]])
invA = np.linalg.inv(A)
print(invA)
assert np.allclose(np.dot(A, invA),
                   np.eye(2))
assert np.allclose(np.dot(invA, A),
                   np.eye(2))
```

Output:

```
[[-0.42857143  0.28571429]
 [ 0.35714286 -0.07142857]]
```

Matrix Transpose

The **transpose** of a matrix $\mathbf{A}$, denoted as $\mathbf{A}^T$ is a reordering of $\mathbf{A}$ where the rows are interchanged with columns, in order. Row 1 of $\mathbf{A}$ becomes column 1 of $\mathbf{A}^T$, row 2 of $\mathbf{A}$ becomes column 2 of $\mathbf{A}^T$, etc. In other words,

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ . & . & \dots & . \\ . & . & \dots & . \\ . & . & \dots & . \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{bmatrix}$$

$$\mathbf{A}^T = \begin{bmatrix} a_{1,1} & a_{2,1} & \dots & a_{m,1} \\ a_{1,2} & a_{2,2} & \dots & a_{m,2} \\ . & . & \dots & . \\ . & . & \dots & . \\ . & . & \dots & . \\ a_{1,n} & a_{2,n} & \dots & a_{m,n} \end{bmatrix}$$

Matrix Transposes with NUMPY

```
A = np.matrix(
        [[7, 1],
         [4, -3],
         [2, 0]])
B = np.matrix(
        [[8, 11],
         [9, 4],
         [3, 3]])
print(A.T)
print(B.T)
```

Output:

```
[[ 7  4  2]
 [ 1 -3  0]]
[[ 8  9  3]
 [11  4  3]]
```

Matrix Transpose Properties

- $(\mathbf{A}^T)^T = A$ (the transpose of the transpose).
- $(\mathbf{A}+\mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$ (transpose of a sum).
- $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$ (transpose of a product).

We can easily check these properties with numpy:

```
assert np.array_equal(A, A.T.T)
assert np.array_equal(B, B.T.T)
assert np.array_equal(np.add(A, B).T,
                      np.add(A.T, B.T))
assert np.array_equal(np.dot(A, B).T,
                      np.dot(B.T, A.T))
```

Augmented Matrix

An augmented matrix is the matrix in which rows or columns of another matrix of the appropriate order are appended to the original matrix. If **A** is augmented on the right with **B**, the resultant matrix is denoted as $(\mathbf{A}|\mathbf{B})$. Let

$$\mathbf{A} = \begin{bmatrix} 1 & 4 \\ 5 & 6 \end{bmatrix};\ \mathbf{B} = \begin{bmatrix} 3 \\ 1 \end{bmatrix};\ \mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Then $(\mathbf{A}|\mathbf{B}) = \left[\begin{array}{cc|c} 1 & 4 & 3 \\ 5 & 6 & 1 \end{array}\right]$ and $(\mathbf{A}|\mathbf{I}) = \left[\begin{array}{cc|cc} 1 & 4 & 1 & 0 \\ 5 & 6 & 0 & 1 \end{array}\right]$.

Linear Systems

Introduction

- Solving systems of linear equations is a fundamental problem of linear algebra and linear programming.
- The solution set of any system of linear equations is the intersection of the solution sets of the individual equations.
- Any solution of a system must be a solution of each equation in the system.
- Any solution of every equation in the system is a solution of the system.

Generic Linear System

A generic linear systems with m equations in n unknowns is written as follows:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\
&. \\
&. \\
&. \\
a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m
\end{aligned}
$$

Since the system is determined by its $m \times n$ coefficient matrix $\mathbf{A} = [a_{ij}]$ and its column vector $\mathbf{b}$, it can be written as $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{x}$ is a column vector $(x_1, x_2, \ldots, x_n)$.

Generic Linear System as Augmented Matrix

A generic linear systems with m equations in n can be expressed with the following augmented matrix:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ . & . & \dots & . & . \\ . & . & \dots & . & . \\ a_{m1} & a_{m2} & \dots & a_{mn} & b_m \end{array}\right]$$

The above matrix is typically shorthanded as $[\mathbf{A}|\mathbf{b}]$.

Elementary Row Operations on Augmented Matrix

Computing all solutions to a linear system is done with the three elementary row operations on the corresponding augmented matrix.

- **R1** (Row interchange): Interchange any two rows in a matrix.
- **R2** (Row scaling): Multiply any row in the matrix by a nonzero scalar.
- **R3** (Row addition): Replace any row in the matrix with the sum of that row and a multiple of another row in the matrix.

Row Equivalence

- If a matrix $\mathbf{B}$ can be obtained from a matrix $\mathbf{A}$ by a sequence of elementary row operations, then $\mathbf{B}$ is **row equivalent** to $\mathbf{A}$.
- Since each elementary row operation can be undone (reversed), if $\mathbf{B}$ is row equivalent to $\mathbf{A}$, denoted as $\mathbf{B} \sim \mathbf{A}$, then $\mathbf{A}$ is row equivalent to $\mathbf{B}$, i.e., $\mathbf{A} \sim \mathbf{B}$.
- The elementary row operations do not change the solution set of an augmented matrix.

A Fundamental Theorem of Linear Algebra

If $[\mathbf{A}|\mathbf{b}] \sim [\mathbf{H}|\mathbf{c}]$, then the corresponding linear systems $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{Hx} = \mathbf{c}$ have the same solution set.

Row Echelon Form and Pivot

A matrix is in **row echelon form** if:

- All rows containing only zeros appear below the rows containing nonzero entries.
- The first nonzero entry in any row appears in a column to the right of the first nonzero entry in any preceding row.

The first nonzero entry in a row of a row echelon form matrix is called the **pivot**.

Row Echelon Form Quiz

Which matrices are in row echelon form?

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \mathbf{B} = \begin{bmatrix} 2 & 4 & 0 \\ 1 & 3 & 2 \\ 0 & 0 & 0 \end{bmatrix}; \mathbf{C} = \begin{bmatrix} 0 & -1 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix};$$

$$\mathbf{D} = \begin{bmatrix} 1 & 3 & 2 & 5 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Solutions of $\mathbf{Hx} = \mathbf{c}$

Let $\mathbf{H}$ be in row echelon form. Let's find all solutions of $\mathbf{Hx} = \mathbf{c}$, where

$$[\mathbf{H}|\mathbf{c}] = \left[\begin{array}{ccc|c} -5 & -1 & 3 & 3 \\ 0 & 3 & 5 & 8 \\ 0 & 0 & 2 & -4 \end{array}\right].$$

The equations corresponding to $[\mathbf{H}|\mathbf{c}]$ are

1. $-5x_1 - x_2 + 3x_3 = 3$;
2. $0x_1 + 3x_2 + 5x_3 = 8$;
3. $0x_1 + 0x_2 + 2x_3 = -4$.

We solve for $x_3 = -2$ in equation 3, substitute this value in equation 2 to solve for $x_2 = 6$, and then substitute the values of x_2 and x_3 into equation 1 to solve for $x_1 = -3$. This method is known as **back substitution**.

A Generic Algorithm Fundamental for Solving a Linear System

Given a linear system $\mathbf{Ax} = \mathbf{b}$, obtain $[\mathbf{A}|\mathbf{b}]$, row-reduce it to $[\mathbf{H}|\mathbf{c}]$, where $\mathbf{H}$ is in row echelon form and use back substitution to find a solution (or not in case of inconsistency).

Solving Linear Systems with NUMPY

Problem

Let's use `numpy` to solve the following linear system:

1. $-5x_1 - x_2 + 3x_3 = 3;$
2. $0x_1 + 3x_2 + 5x_3 = 8;$
3. $0x_1 + 0x_2 + 2x_3 = -4.$

Solution

```
A = np.array(
        [[-5, -1, 3],
         [0, 3, 5],
         [0, 0, 2]])
b = np.array([3, 8, -4])
x = np.linalg.solve(A, b)
print(x)
assert np.allclose(np.dot(A, x), b)
```

Output:

```
[-3.  6. -2.]
```

Problem

Let's use `numpy` to solve the following linear system:

1. $4x + 3y = 480$;
2. $3x + 6y = 720$.

Solution

```
A = np.array(
        [[4, 3],
         [3, 6]])
b = np.array([480, 720])
x = np.linalg.solve(A, b)
print(x)
assert np.allclose(np.dot(A, x), b)
```

Output:

```
[48.  96.]
```

Problem

Use back substitution to find all solutions of $\mathbf{Hx} = \mathbf{c}$, where

$$[\mathbf{H}|\mathbf{c}] = \begin{bmatrix} 1 & -3 & 5 & | & 3 \\ 0 & 1 & 2 & | & 2 \\ 0 & 0 & 0 & | & -1 \end{bmatrix}.$$

The equations corresponding to $[\mathbf{H}|\mathbf{c}]$ are

1. $1x_1 - 3x_2 + 5x_3 = 3$;
2. $0x_1 + 1x_2 + 2x_3 = 2$;
3. $0x_1 + 0x_2 + 0x_3 = -1$.

We cannot solve for x_3 in equation 3, thus the system has no solution, i.e., **inconsistent**.

Solution

```
A = np.array(
        [[1, -3, 5],
         [0, 1, 2],
         [0, 0, 0]])
b = np.array([3, 2, -1])
try:
   x = np.linalg.solve(A, b)
   print(x)
   assert np.allclose(np.dot(A, x), b)
except Exception as e:
   print(e)
```

Output:

```
Singular matrix
```

Problem

Let's use back substitution to find all solutions of $\mathbf{Hx} = \mathbf{c}$, where

$$[\mathbf{H}|\mathbf{c}] = \left[\begin{array}{ccccc|c} 1 & -3 & 0 & 5 & 0 & 4 \\ 0 & 0 & 1 & 2 & 0 & -7 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right].$$

Notice that the 2nd and 4th columns have no pivots. The equations corresponding to $[\mathbf{H}|\mathbf{c}]$ are

1. $x_1 - 3x_2 + 5x_4 = 4$;
2. $x_3 + 2x_4 = -7$;
3. $x_5 = 1$.

We obtain the following solutions:

1. $x_1 = 3x_2 - 5x_4 + 4$;
2. $x_3 = -2x_4 - 7$;
3. $x_5 = 1$.

Solution

What does it mean to have a solution like this?

1. $x_1 = 3x_2 - 5x_4 + 4$;
2. $x_3 = -2x_4 - 7$;
3. $x_5 = 1$.

It simply means that we can take any real values for the values of x_2 and x_4, say r and s, and get a solution to the system. In other words, the solution vector looks as follows:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 3r - 5s + 4 \\ r \\ -2s - 7 \\ s \\ 1 \end{bmatrix}.$$

Solution

```
A = np.array(
      [[1, -3, 0, 5, 0],
       [0, 0, 1, 2, 0],
       [0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0]])
b = np.array([4, -7, 1, 0])
try:
   x = np.linalg.solve(A, b)
   print(x)
except Exception as e:
   print(e)
```

Output:

```
Last 2 dimensions of the array must be square
```

Problem

Solve the following linear system.

1. $x_2 - 3x_3 = -5$.
2. $2x_1 + 3x_2 - x_3 = 7$.
3. $4x_1 + 5x_2 - 2x_3 = 10$.

Solution

If we row-reduce **A** (it will take a while to do it by hand), we get

$$\mathbf{A} = \left[\begin{array}{ccc|c} 0 & 1 & -3 & -5 \\ 2 & 3 & -1 & 7 \\ 4 & 5 & -2 & 10 \end{array}\right] \sim \left[\begin{array}{ccc|c} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & 3 \end{array}\right]$$

So the solutions are $x_1 = -1$, $x_2 = 4$, $x_3 = 3$.

Solution

```
A = np.array(
        [[0, 1, -3],
         [2, 3, -1],
         [4, 5, -2]])
b = np.array([-5, 7, 10])
try:
   x = np.linalg.solve(A, b)
   print(x)
   assert np.allclose(np.dot(A, x), b)
except Exception as e:
   print(e)
```

Output:

```
[-1.  4.  3.]
```

Computation of the Inverse Matrix $\mathbf{A}^{-1}$

The inverse of a matrix $\mathbf{A}$ is denoted as $\mathbf{A}^{-1}$. To find $\mathbf{A}^{-1}$, if it exists, proceed as follows:

1. From the augmented matrix $[\mathbf{A}|\mathbf{I}]$.
2. Apply the Gauss method to attempt to reduce $[\mathbf{A}|\mathbf{I}]$ to $[\mathbf{I}|\mathbf{C}]$. If the reduction can be carried out, then $\mathbf{A}^{-1} = \mathbf{C}$. Otherwise, $\mathbf{A}^{-1}$ does not exist.

Using $\mathbf{A}^{-1}$ to Solve $\mathbf{Ax} = \mathbf{b}$

If $\mathbf{A}^{-1}$ exists, then

1. $\mathbf{A}^{-1}\mathbf{Ax} = \mathbf{A}^{-1}\mathbf{b}$;
2. $\mathbf{Ix} = \mathbf{A}^{-1}\mathbf{b}$;
3. $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$.

Problem

Solve the linear system:

1. $2x + 9y = -5$.
2. $x + 4y = 7$.

Solution

$$[\mathbf{A}|\mathbf{b}] = \left[\begin{array}{cc|cc} 2 & 9 & 1 & 0 \\ 1 & 4 & 0 & 1 \end{array}\right] \sim \left[\begin{array}{cc|cc} 1 & 0 & -4 & 9 \\ 0 & 1 & 1 & -2 \end{array}\right].$$

Thus,

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -4 & 9 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} -5 \\ 7 \end{bmatrix} = \begin{bmatrix} 83 \\ -19 \end{bmatrix}$$

Solution

```
A = np.array(
        [[2, 9],
         [1, 4]])
b = np.array([-5, 7])
invA = np.linalg.inv(A)
x = np.dot(invA, b)
print(x)
assert np.allclose(np.dot(A, x), b)
```

Output:

```
[ 83. -19.]
```

References

1. J.B. Fraleigh, R. A. Beauregard. *Linear Algebra*, Addison-Wesley.
2. `www.python.org`.
3. `docs.scipy.org`.