SciComp with Py

Linear Algebra: Part 1

Vladimir Kulyukin
Department of Computer Science
Utah State University

Outline

Definition and Notation

A **matrix** is a rectangular array of numbers. Matrices are typically denoted 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

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

Definition and Notation

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

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

Examples

$$\mathbf{A}_1 = \begin{bmatrix} 3 & 4 \\ 101 & 542 \end{bmatrix}$$

$$\mathbf{A}_2 = \begin{bmatrix} 4 & 5 & 8 & 10 \\ 11 & 52 & 67 & 12 \\ 90 & 7 & 41 & 78 \end{bmatrix}$$

$$\mathbf{A}_3 = \begin{bmatrix} 2 & 3 & 4 & 10 \\ 120 & 54 & 78 & 34 \\ 123 & 65 & 81 & 594 \\ 9 & 13 & 74 & 15 \\ 11 & 33 & 54 & 103 \end{bmatrix}$$

Matrices in Numpy: Construction and Access

Consider this Py code (source in `linalg_01.py`):

```
import numpy as np
A = np.array([[7, -1, -2],
              [3, 3, 0]])
```

Here is a test:

```
>>> A
array([[ 7, -1, -2],
       [ 3,  3,  0]])
>>> A[1,1] == A[1][1] == 3
True
>>> A[0,1] == A[0][1] == -1
True
>>> A[1,2] == A[1][2] == 0
True
```

Outline

Matrix Addition

Two matrices of the same order (same number of rows and colums) 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 in Numpy

Consider this Py code (source in `linalg_01.py`):

```
A = np.array([[7, -1, -2], [3, 3, 0]])
B = np.array([[2, -3, 4], [1, 5, 9]])
print 'A+B='
print np.add(A, B)
```

Here is a test:

```
A+B=
[[ 9 -4  2]
 [ 4  8  9]]
```

Matrix Addition in Numpy

Consider this Py code (source in `linalg_01.py`):

```
A = np.array([[7, -1, -2], [3, 3, 0]])
B = np.array([[2, -3, 4], [1, 5, 9]])
print 'B+A='
print np.add(B, A)
```

Here is a test:

```
B+A=
[[ 9 -4  2]
 [ 4  8  9]]
```

Matrix Addition in Numpy: Question

What's the output of this Py code:

```
import numpy as np

A = np.array([[7, -1, -2],
              [3, 3, 0]])
C = np.array([[2, 1],
              [7, 3],
              [9, 2]])
try:
    print 'A+C='
    print np.add(A, C)
except Exception:
    print 'undefined'
```

Matrix Addition in Numpy: Question

What's the output of this Py code:

```
import numpy as np

B = np.array([[2, -3, 4],
              [1, 5, 9]])
C = np.array([[2, 1],
              [7, 3],
              [9, 2]])
try:
    print 'B+C='
    print np.add(B, C)
except Exception:
    print 'undefined'
```

Matrix Addition in Numpy: Question

What's the output of this Py code:

```
import numpy as np

A = np.array([[7, -1, -2],
              [3, 3, 0]])
D = np.array([[1, 2, 3],
              [0, 0, 0],
              [5, 6, 7],
              [0, 0, 0]])
try:
    print 'A+D='
    print np.add(A, D)
except Exception:
    print 'undefined'
```

Matrix Multiplication

Two matrices **A** and **B** may be multiplied 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**).

Example

Let **A** be a 3×4 matrix and **B** be a 4×5 matrix. Let $\mathbf{A} \times \mathbf{B} = \mathbf{C}$, where **C** is a 3×5 matrix. Let's compute a few entries in **C**.

1. $c_{11} = \sum_{k=1}^{4} a_{1k} b_{k1} = a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31} + a_{14} b_{41}$;
2. $c_{12} = \sum_{k=1}^{4} a_{1k} b_{k2} = a_{11} b_{12} + a_{12} b_{22} + a_{13} b_{32} + a_{14} b_{42}$;
3. $c_{13} = \sum_{k=1}^{4} a_{1k} b_{k3} = a_{11} b_{13} + a_{12} b_{23} + a_{13} b_{33} + a_{14} b_{43}$;
4. $c_{14} = \sum_{k=1}^{4} a_{1k} b_{k4} = a_{11} b_{14} + a_{12} b_{24} + a_{13} b_{34} + a_{14} b_{44}$;
5. $c_{15} = \sum_{k=1}^{4} a_{1k} b_{k5} = a_{11} b_{15} + a_{12} b_{25} + a_{13} b_{35} + a_{14} b_{45}$;
6. $c_{34} = \sum_{k=1}^{4} a_{3k} b_{k4} = a_{31} b_{14} + a_{32} b_{24} + a_{33} b_{34} + a_{34} b_{44}$;
7. $c_{23} = \sum_{k=1}^{4} a_{2k} b_{k3} = a_{21} b_{13} + a_{22} b_{23} + a_{23} b_{33} + a_{24} b_{43}$;

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

$\mathbf{AB} = \mathbf{C} = [c_{ij}] = \sum_{k=1}^{2} a_{ik} b_{kj}, i \in [1,3], j \in [1,2]$. 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 in Numpy

Consider this Py code (source in `linalg_02.py`).

```
import numpy as np

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

Here is the output:

```
A x B =
[[14  6 53]
 [ 8  7 16]
 [ 4  2 14]]
```

More 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 $\mathbf{BA} = \begin{bmatrix} 2 & 1 & 7 \\ 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} 7 & 1 \\ 4 & -3 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} 32 & -1 \\ 4 & 3 \end{bmatrix}.$

The entries of **BA** are computed as follows:

$32 = 2 \cdot 7 + 1 \cdot 4 + 7 \cdot 2 = 14 + 4 + 14 = 32.$
$4 = 0 \cdot 7 + (-1) \cdot 4 + 4 \cdot 2 = 0 + -4 + 8.$
$-1 = 2 \cdot 1 + 1 \cdot (-3) + 7 \cdot 0 = 2 + -3 + 0.$
$3 = 0 \cdot 1 + (-1) \cdot (-3) + 4 \cdot 0 = 0 + 3 + 0.$

AC and **CB** are undefined, because they are not conformable.

Matrix Multiplication in Numpy: Question

```
import numpy as np
A = np.array([[7, 1],
              [4, -3],
              [2, 0]])
C = np.array([[1, -1, 3],
              [2, 2, 3],
              [-1, 4, 7]])
```

What is the output?

```
print 'A x C = '
print np.dot(A, C)
```

Outline

Diagonal Matrix

A **diagonal** matrix is a square matrix (i.e., the number of rows is equal to the number of columns) whose off-diagonal elements (i.e., $a_{ij}, i \neq j$) are equal to 0.

$$\mathbf{A} = [a_{ij}] = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ . & . & \dots & . \\ . & . & \dots & . \\ . & . & \dots & . \\ 0 & 0 & \dots & a_{nn} \end{bmatrix}$$

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 Matrix in Numpy

This Py code shows how to construct identity matrices (source in `linalg_03.py`).

```
import numpy as np

I2 = np.identity(2) ## same as np.eye(2)
I3 = np.identity(3) ## same as np.eye(3)
A2x2 = np.array([[0, 2],
                 [3, 5]])
A3x3 = np.array([[1, 10, 3],
                 [7, 12, 5],
                 [9, 2, 10]])
print np.dot(I2, A2x2)
print np.dot(A2x2, I2)
print np.dot(A3x3, I3)
print np.dot(I3, A3x3)
```

Inverse Matrix

An $n \times n$ matrix A is invertible if there exists an $n \times n$ matrix A^{-1} such that $A^{-1} \times A = A \times A^{-1} = I$, where I is the $n \times n$ identity matrix. The matrix A^{-1} is called A's inverse.

Inverse Matrix in Numpy

This Py code shows how to construct identity matrices (source in `linalg_07.py`).

```
import numpy as np
A = np.matrix([
    [2, 1],
    [6, 5]
    ])
I2 = np.matrix(np.eye(2))
## compute inverse of A.
print A.I
assert np.array_equal(I2, np.dot(A, A.I))
assert np.array_equal(np.dot(A, A.I), I2)
print 'assertions passed'
```

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 Transpose in Numpy

This Py code shows how to do matrix transpose (source in `linalg_04.py`):

```
import numpy as np
A = np.array([
        [7, 1],
        [4, -3],
        [2, 0]
        ])
## transpose
print 'A^T = ', A.T
assert np.array_equal(A, np.transpose(np.transpose(A)))
assert np.array_equal(A, A.T.T)
```

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).

Symmetric Matrix

A **symmetric** matrix $\mathbf{A}$ is a square matrix equal to its transpose, i.e., $\mathbf{A} = \mathbf{A}^T$. For example, these two matrices are symmetric.

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 6 & 4 \\ 3 & 4 & 9 \end{bmatrix}$$
$$\mathbf{B} = \begin{bmatrix} 1 & 7 \\ 7 & 10 \end{bmatrix}$$

Here is how we can tell if a numpy matrix is symmetric.

```
def is_symmetric_matrix(mat):
    return np.array_equal(mat, mat.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 $\mathbf{A}$ is augmented on the right with $\mathbf{B}$, 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 $(\mathbf{A}|\mathbf{I}) = \left[\begin{array}{cc|cc} 1 & 4 & 1 & 0 \\ 5 & 6 & 0 & 1 \end{array}\right]$.

Multiplying Matrices by Scalars in Numpy

Here is how we can multiply matrices by scalars.

```
import numpy as np
A = np.array([
        [7, 1],
        [4, -3],
        [2, 0]
        ])
print 3.0*A
print 4.5*A
```

Outline

Generic Linear System

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

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

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 $(x_1, x_2, ..., x_n)$.

Example

Here is a concrete linear system:

$$\begin{aligned} 3x + 2y + z &= 39 \\ 2x + 3y + z &= 34 \\ x + 2y + 3z &= 26 \end{aligned}$$

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 $[\mathbf{A}|\mathbf{b}]$.

Example

Here is a concrete linear system:

$$\begin{aligned} 3x + 2y + z &= 39 \\ 2x + 3y + z &= 34 \\ x + 2y + 3z &= 26 \end{aligned}$$

Here is the corresponding linear system:

$$\begin{bmatrix} 3 & 2 & 1 & | & 39 \\ 2 & 3 & 1 & | & 34 \\ 1 & 2 & 3 & | & 26 \end{bmatrix}$$

Solving Linear Systems

- Solving systems of linear equations is a fundamental problem of linear algebra.
- 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.

Example

Here is a concrete linear system:

$$\begin{aligned} -5x_1 - x_2 + 3x_3 &= 3 \\ 3x_2 + 5x_3 &= 8 \\ 2x_3 &= -4 \end{aligned}$$

The solution to this linear system is $x_1 = -3, x_2 = 6, x_3 = -2$.

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.

Matrix Row Interchange in Numpy

Here is how you can interchange two rows in numpy (source in `linalg_06.py`).

```
import numpy as np
A = np.array([
        [7, 1],
        [4, -3],
        [2, 0]
        ])
## Interchanging two rows r1, r2 in a matrix M, do
## M[[r1, r2]] = M[[r2, r1]].
print A
A[[1, 2]] = A[[2, 1]]
print A
```

Accessing Rows and Columns in Numpy

Here is how you can access and iterate over rows and columns (source in `linalg_06.py`).

```
import numpy as np
A = np.array([
        [7, 1],
        [4, -3],
        [2, 0]
        ])

for r in xrange(A.shape[0]):
    print 'row', r, A[r,:]

for c in xrange(A.shape[1]):
    print 'col', c, A[:,c]
```

Matrix Row and Column Scaling in Numpy

Here is how you can scale a matrix row over in Numpy (source in `linalg_06.py`).

```
## To multiply a row r by a scalar s in a 2D matrix
## M, do s*M[r,:].
print 4.0*A[1,:]

## To multiply a colum c by a scalar s in a 2D matrix
## M, do s*M[:,c].
print 4.0*A[:,0]
```

Matrix Row Addition in Numpy

Here is how you can do row addition in Numpy (source in `linalg_06.py`).

```
## To multiply a row r by a scalar s in a 2D matrix
## M, do s*M[r,:].
print 4.0*A[1,:]

## To multiply a colum c by a scalar s in a 2D matrix
## M, do s*M[:,c].
print 4.0*A[:,0]
```

Matrix Row Addition in Numpy

Here is how you can do row addition in Numpy (source in `linalg_06.py`).

```
## To replace row r1 in 2D matrix M with the sum
## of r and a multiple of row r2,
## do M[r1,:] = M[r1,:] + s*M[r2,:]
A2 = A.copy()
print A2
A2[1,:] = A2[1,:] + 2*A2[2,:]
print A2
```

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?

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

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

Back Substitution: Solving $\mathbf{Hx} = \mathbf{c}$

Before outlining a generic algorithm for solving linear systems, let us go though several examples to see how easy it is to determine all solutions of the system if $\mathbf{H}$ is in row echelon form. Let us 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 x_3 in equation 2 to solve for $x_2 = 6$, and then substitute x_2 and x_3 into equation 1 to solve for $x_1 = -3$. This method is **back substitution**.

Back Substitution: No Solution

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

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

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**.

Back Substitution: Multiple Solutions

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. $x_5 = 1$.

Back Substitution: Multiple Solutions

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:

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

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).

Gauss Reduction of $\mathbf{Ax} = \mathbf{b}$ to $\mathbf{Hx} = \mathbf{c}$

- If the first column of $\mathbf{A}$ contains only 0's, cross it off. Keep crossing off zero columns until the left column has a non-zero entry or the matrix is exhausted.
- Use the row exchange if necessary to the obtain the pivot entry in the top row of the first non-zero column.
- Use the scalar multiplication to set the value of the pivot entry in the top row of the first non-zero column to 1.
- Use row addition and scalar multiplication to create 0's below the pivot.
- Cross off the left column and recurse to step 1 on the smaller matrix.

Gauss Reduction Problem

Reduce to row echolon form the following matrix and make all pivots equal to 1.

$$\mathbf{A} = \begin{bmatrix} 2 & -4 & 2 & -2 \\ 2 & -4 & 3 & -4 \\ 4 & -8 & 3 & -2 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

Answer to Gauss Reduction Problem

Reduce to row echolon form the following matrix and make all pivots equal to 1.

$$\mathbf{A} = \begin{bmatrix} 2 & -4 & 2 & -2 \\ 2 & -4 & 3 & -4 \\ 4 & -8 & 3 & -2 \\ 0 & 0 & -1 & 2 \end{bmatrix} \sim \begin{bmatrix} 1 & -2 & 1 & -1 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Linear System Problem

Solve the following linear system.

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

Answer to Linear System Problem

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

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

Computation of $\mathbf{A}^{-1}$

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.

Example of Using $\mathbf{A}^{-1}$ to Solve Linear Systems

Solve the linear system:

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

Answer to Example

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

Thus,

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

Conditions for $\mathbf{A}^{-1}$ to Exist

1. $\mathbf{A}$ is invertible.
2. $\mathbf{A}$ is row equivalent to the identify matrix $\mathbf{I}$.