

Lecture 11

Solving Linear Systems of Equations

L. Olson

Department of Computer Science
University of Illinois at Urbana-Champaign

Slides based on slides from Numerical Methods with Matlab by Gerald Recktenwald

February 22, 2006

Matrix Rank

- The **rank** of a matrix, A , is the number of linearly independent columns in A .
- $\text{rank}(A)$ is the dimension of the column space of A .
- Numerical computation of $\text{rank}(A)$ is tricky due to roundoff.

Consider

$$u = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad v = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad w = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Do these vectors span \mathbf{R}^3 ?

Matrix Rank

- The **rank** of a matrix, A , is the number of linearly independent columns in A .
- $\text{rank}(A)$ is the dimension of the column space of A .
- Numerical computation of $\text{rank}(A)$ is tricky due to roundoff.

Consider

$$u = \begin{bmatrix} 1 \\ 0 \\ 0.00001 \end{bmatrix} \quad v = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad w = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Do these vectors span \mathbf{R}^3 ?

Matrix Rank

- The **rank** of a matrix, A , is the number of linearly independent columns in A .
- $\text{rank}(A)$ is the dimension of the column space of A .
- Numerical computation of $\text{rank}(A)$ is tricky due to roundoff.

Consider

$$u = \begin{bmatrix} 1 \\ 0 \\ \varepsilon_m \end{bmatrix} \quad v = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad w = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Do these vectors span \mathbf{R}^3 ?

Matrix Rank (2)

We can use MATLAB's built-in **rank** function for exploratory calculations on (relatively) small matrices

Example:

```
1 >> A = [1 0 0; 0 1 0; 0 0 1e-5]      % A(3,3) is small
2 A =
3     1.0000         0         0
4         0     1.0000         0
5         0         0     0.0000
6
7 >> rank(A)
8 ans =
9     3
```

Matrix Rank (2)

Repeat numerical calculation of rank with smaller diagonal entry

```
1 >> A(3,3) = eps/2      % A(3,3) is even smaller
2 A =
3     1.0000           0           0
4           0     1.0000           0
5           0           0     0.0000
6
7 >> rank(A)
8 ans =
9      2
```

Even though $A(3,3)$ is not identically zero, it is small enough that the matrix is *numerically* rank-deficient

Matrix Determinant (1)

- Only square matrices have determinants.
- The determinant of a (square) matrix is a scalar.
- If $\det(A) = 0$, then A is singular, and A^{-1} does not exist.
- $\det(I) = 1$ for any identity matrix I .
- $\det(AB) = \det(A) \det(B)$.
- $\det(A^T) = \det(A)$.
- Cramer's rule uses (many!) determinants to express the the solution to $Ax = b$.

Matrix Determinant (2)

- $\det(A)$ is not useful for numerical computation
 - ▶ Computation of $\det(A)$ is expensive
 - ▶ Computation of $\det(A)$ can cause overflow

$$\det(A) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

$$\begin{aligned} \det(A) &= \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} \\ &= 1 \cdot \begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} - 4 \cdot \begin{vmatrix} 1 & 3 \\ 7 & 9 \end{vmatrix} + 9 \cdot \begin{vmatrix} 1 & 2 \\ 4 & 5 \end{vmatrix} \end{aligned}$$

Matrix Determinant (3)

- For diagonal and triangular matrices, $\det(A)$ is the product of diagonal elements
- The built in **det** computes the determinant of a matrix by first factoring it into $A = LU$, and then computing

$$\begin{aligned}\det(A) &= \det(L) \det(U) \\ &= (\ell_{11}\ell_{22} \dots \ell_{nn})(u_{11}u_{22} \dots u_{nn})\end{aligned}$$

Special Matrices

- Diagonal Matrices
- Tridiagonal Matrices
- The Identity Matrix
- The Matrix Inverse
- Symmetric Matrices
- Positive Definite Matrices
- Orthogonal Matrices
- Permutation Matrices

Diagonal Matrices (1)

Diagonal matrices have non-zero elements only on the main diagonal.

$$C = \text{diag}(c_1, c_2, \dots, c_n) = \begin{bmatrix} c_1 & 0 & \cdots & 0 \\ 0 & c_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & c_n \end{bmatrix}$$

The **diag** function is used to either create a diagonal matrix from a vector, or and extract the diagonal entries of a matrix.

```
1 >> x = [1 -5 2 6];
2 >> A = diag(x)
3 A =
4     1     0     0     0
5     0    -5     0     0
6     0     0     2     0
7     0     0     0     6
```

Diagonal Matrices (2)

The **diag** function can also be used to create a matrix with elements only on a specified *super*-diagonal or *sub*-diagonal. Doing so requires using the two-parameter form of **diag**:

```
1 >> diag([1 2 3], 1)
2 ans =
3     0     1     0     0
4     0     0     2     0
5     0     0     0     3
6     0     0     0     0
7 >> diag([4 5 6], -1)
8 ans =
9     0     0     0     0
10    4     0     0     0
11    0     5     0     0
12    0     0     6     0
```

Identity Matrices (1)

An identity matrix is a square matrix with ones on the main diagonal.

Example: *The 3×3 identity matrix*

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

An identity matrix is special because

$$AI = A \quad \text{and} \quad IA = A$$

for *any* compatible matrix A . This is like multiplying by one in scalar arithmetic.

Identity Matrices (2)

Identity matrices can be created with the built-in **eye** function.

```
1 >> I = eye(4)
2 I =
3     1     0     0     0
4     0     1     0     0
5     0     0     1     0
6     0     0     0     1
```

Sometimes I_n is used to designate an identity matrix with n rows and n columns. For example,

$$I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Identity Matrices (3)

A non-square, *identity-like* matrix can be created with the two-parameter form of the eye function:

```
1 >> J = eye(3,5)
2 J =
3     1     0     0     0     0
4     0     1     0     0     0
5     0     0     1     0     0
6
7 >> K = eye(4,2)
8 K =
9     1     0
10    0     1
11    0     0
12    0     0
```

J and K are *not* identity matrices!

Matrix Inverse (1)

Let A be a square (i.e. $n \times n$) with real elements. The *inverse* of A is designated A^{-1} , and has the property that

$$A^{-1}A = I \quad \text{and} \quad AA^{-1} = I$$

The **formal solution** to $Ax = b$ is $x = A^{-1}b$.

$$Ax = b$$

$$A^{-1}Ax = A^{-1}b$$

$$Ix = A^{-1}b$$

$$x = A^{-1}b$$

Matrix Inverse (2)

- formal solution to $Ax = b$ is $x = A^{-1}b$

Matrix Inverse (3)

- formal solution to $Ax = b$ is $x = A^{-1}b$
- BUT it is *bad* evaluate x this way

Matrix Inverse (4)

- formal solution to $Ax = b$ is $x = A^{-1}b$
- BUT it is *bad* evaluate x this way
- why?

Matrix Inverse (5)

- formal solution to $Ax = b$ is $x = A^{-1}b$
- BUT it is *bad* evaluate x this way
- why?
- we will not form A^{-1} , but solve for x directly using Gaussian elimination.

Functions to Create Special Matrices

Matrix	MATLAB function
Diagonal	<code>diag</code>
Tridiagonal	<code>tridiags</code> (NMM Toolbox)
Identity	<code>eye</code>
Inverse	<code>inv</code>

Symmetric Matrices

If $A = A^T$, then A is called a *symmetric* matrix.

Example:

$$\begin{bmatrix} 5 & -2 & -1 \\ -2 & 6 & -1 \\ -1 & -1 & 3 \end{bmatrix}$$

Note

$B = A^T A$ is symmetric for any (real) matrix A .

Why do we care as Numerical Analysts?

Open questions:

- How *expensive* is it to solve $Ax = b$?
- What problems (errors) will we encounter solving $Ax = b$?
- Some matrices are easy/cheap to use: diagonal, tridiagonal, etc.
 - ▶ are there others? what makes something a "good" matrix numerically?
 - ▶ are there bad ones? how do we identify them numerically?
- what do actual numerical analysts, engineers, developers, etc use?!?!

Primary Topics

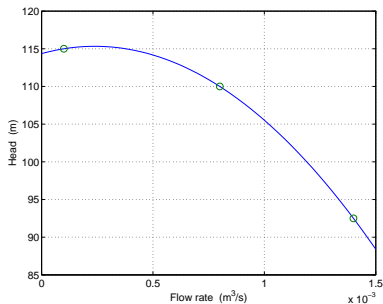
- Basic Concepts
- Gaussian Elimination
- Limitations on Numerical Solutions to $Ax = b$
- Factorization Methods

Basic Concepts

- Matrix Formulation
- Requirements for a Solution
 - Consistency
 - The Role of $\text{rank}(A)$
 - Formal Solution when A is $n \times n$

Pump Curve Model (1)

Objective: Find the coefficients of the quadratic equation that approximates the pump curve data.



Model equation:

$$h = c_1q^2 + c_2q + c_3$$

Write the model equation for three points on the curve. This gives three equations for the three unknowns c_1 , c_2 , and c_3 .

Pump Curve Model (2)

Points from the pump curve:

q (m ³ /s)	1×10^{-4}	8×10^{-4}	1.4×10^{-3}
h (m)	115	110	92.5

Substitute each pair of data points into the model equation

$$\begin{aligned}115 &= 1 \times 10^{-8} c_1 + 1 \times 10^{-4} c_2 + c_3, \\110 &= 64 \times 10^{-8} c_1 + 8 \times 10^{-4} c_2 + c_3, \\92.5 &= 196 \times 10^{-8} c_1 + 14 \times 10^{-4} c_2 + c_3,\end{aligned}$$

Rewrite in matrix form as

$$\begin{bmatrix} 1 \times 10^{-8} & 1 \times 10^{-4} & 1 \\ 64 \times 10^{-8} & 8 \times 10^{-4} & 1 \\ 196 \times 10^{-8} & 14 \times 10^{-4} & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 115 \\ 110 \\ 92.5 \end{bmatrix}.$$

Pump Curve Model (3)

Using more compact symbolic notation

$$Ax = b$$

where

$$A = \begin{bmatrix} 1 \times 10^{-8} & 1 \times 10^{-4} & 1 \\ 64 \times 10^{-8} & 8 \times 10^{-4} & 1 \\ 196 \times 10^{-8} & 14 \times 10^{-4} & 1 \end{bmatrix},$$

$$x = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}, \quad b = \begin{bmatrix} 115 \\ 110 \\ 92.5 \end{bmatrix}.$$

Pump Curve Model (4)

In general, for any three (q, h) pairs the system is still $Ax = b$ with

$$A = \begin{bmatrix} q_1^2 & q_1 & 1 \\ q_2^2 & q_2 & 1 \\ q_3^2 & q_3 & 1 \end{bmatrix}, \quad x = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}, \quad b = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}.$$

Matrix Formulation

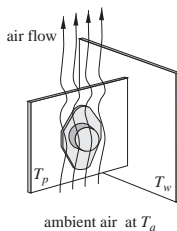
Recommended Procedure

- 1 Write the equations in natural form.
- 2 Identify unknowns, and order them.
- 3 Isolate the unknowns.
- 4 Write equations in matrix form.

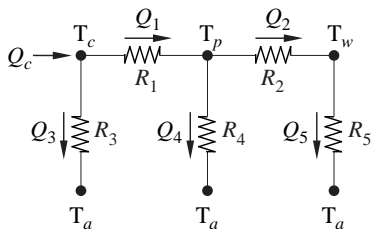
Thermal Model of an IC Package (1)

Objective: Find the temperature of an integrated circuit (IC) package mounted on a heat spreader. The system of equations is obtained from a thermal resistive network model.

Physical Model:



Mathematical Model:



Thermal Model of an IC Package (2)

1. Write the equations in natural form:

Use resistive model of heat flow between nodes to get

$$Q_1 = \frac{1}{R_1}(T_c - T_p)$$

$$Q_2 = \frac{1}{R_2}(T_p - T_w)$$

$$Q_3 = \frac{1}{R_3}(T_c - T_a)$$

$$Q_4 = \frac{1}{R_4}(T_p - T_a)$$

$$Q_2 = \frac{1}{R_5}(T_w - T_a)$$

$$Q_c = Q_1 + Q_3$$

$$Q_1 = Q_2 + Q_4$$

Thermal Model of an IC Package (3)

2. Identify unknowns, and order them:

Unknowns are Q_1 , Q_2 , Q_3 , Q_4 , T_c , T_p , and T_w . Thus,

$$x = \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ T_c \\ T_p \\ T_w \end{bmatrix}.$$

3. Isolate the Unknowns

$$R_1 Q_1 - T_c + T_p = 0,$$

$$R_2 Q_2 - T_p + T_w = 0,$$

$$R_3 Q_3 - T_c = -T_a,$$

$$R_4 Q_4 - T_p = -T_a,$$

$$R_5 Q_2 - T_w = -T_a,$$

$$Q_1 + Q_3 = Q_c,$$

$$Q_1 - Q_2 - Q_4 = 0.$$

Thermal Model of an IC Package (4)

4. Write in Matrix Form

$$\begin{bmatrix} R_1 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & R_2 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & R_3 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & R_4 & 0 & -1 & 0 \\ 0 & R_5 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ T_c \\ T_p \\ T_w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -T_a \\ -T_a \\ -T_a \\ Q_c \\ 0 \end{bmatrix}$$

The coefficient matrix has many more zeros than non-zeros. This is an example of a *sparse* matrix.

Requirements for a Solution

- 1 Consistency
- 2 The Role of $\text{rank}(A)$
- 3 Formal Solution when A is $n \times n$
- 4 Summary

Consistency (1)

- If an exact solution to $Ax = b$ exists, b must lie in the column space of A
- If it does, then the system is said to be **consistent**.
- **If the system is consistent, an exact solution exists.**

Consistency (2)

- rank(A) gives the number of linearly independent columns in A
- $[A \ b]$ is the *augmented* matrix formed by combining the b vector with the columns of A :

$$[A \ b] = \left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1,n} & b_1 \\ a_{21} & a_{22} & & a_{2,n} & b_2 \\ \vdots & & \ddots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{n,n} & b_n \end{array} \right]$$

- If $\text{rank}([A \ b]) > \text{rank}(A)$ then b does not lie in the column space of A .

since $[A \ b]$ has a larger set of basis vectors than A , and since the difference in the size of the basis set is solely due to the b vector, the b vector cannot be constructed from the column vectors of A .

Role of rank(A)

- If A is $m \times n$, and z is an n -element column vector, then $Az = 0$ has a nontrivial solution only if the columns of A are linearly dependent¹.
- In other words, the *only* solution to $Az = 0$ is $z = 0$ when A is full rank.
- Given the $m \times n$ matrix A , the system $Ax = b$ has a unique solution if the system is consistent and if $\text{rank}(A) = n$.

¹"0" is the m -element column vector filled with zeros

Summary of Solution to $Ax = b$ where A is $m \times n$

For the general case where A is $m \times n$ and $m \geq n$,

- If $\text{rank}(A) = n$ and the system is consistent, the solution exists and it is unique.
- If $\text{rank}(A) = n$ and the system is inconsistent, no solution exists.
- If $\text{rank}(A) < n$ and the system is consistent, an infinite number of solutions exist.

If A is $n \times n$ and $\text{rank}(A) = n$, then the system is consistent and the solution is unique.

Formal Solution when A is $n \times n$

The *formal solution* to $Ax = b$ is

$$x = A^{-1}b$$

where A is $n \times n$.

If A^{-1} exists then A is said to be **nonsingular**.

If A^{-1} does not exist then A is said to be **singular**.

Formal Solution when A is $n \times n$

If A^{-1} exists then

$$Ax = b \quad \implies \quad x = A^{-1}b$$

but

Do not compute the solution to $Ax = b$ by finding A^{-1} , and then multiplying b by A^{-1} !

We see: $x = A^{-1}b$

We do: **Solve $Ax = b$ by Gaussian elimination or an equivalent algorithm**

Singularity of A

If an $n \times n$ matrix, A , is **singular** then

- the columns of A are linearly dependent
- the rows of A are linearly dependent
- $\text{rank}(A) < n$
- $\det(A) = 0$
- A^{-1} does not exist
- a solution to $Ax = b$ may not exist
- If a solution to $Ax = b$ exists, it is not unique

Summary of Requirements for Solution of $Ax = b$

Given the $n \times n$ matrix A and the $n \times 1$ vector, b

- the solution to $Ax = b$ exists and is unique for any b if and only if $\text{rank}(A) = n$.
- $\text{rank}(A) = n$ automatically guarantees that the system is consistent.

Gaussian Elimination

- Solving Diagonal Systems
- Solving Triangular Systems
- Gaussian Elimination Without Pivoting
 - ▶ Hand Calculations
 - ▶ Cartoon Version
 - ▶ The Algorithm
- Gaussian Elimination with Pivoting
 - ▶ Row or Column Interchanges, or Both
 - ▶ Implementation
- Solving Systems with the Backslash Operator

Solving Diagonal Systems

The system defined by

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix} \quad b = \begin{bmatrix} -1 \\ 6 \\ -15 \end{bmatrix}$$

Solving Diagonal Systems

The system defined by

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix} \quad b = \begin{bmatrix} -1 \\ 6 \\ -15 \end{bmatrix}$$

is equivalent to

$$\begin{aligned} x_1 &= -1 \\ 3x_2 &= 6 \\ 5x_3 &= -15 \end{aligned}$$

Solving Diagonal Systems

The system defined by

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix} \quad b = \begin{bmatrix} -1 \\ 6 \\ -15 \end{bmatrix}$$

is equivalent to

$$\begin{aligned} x_1 &= -1 \\ 3x_2 &= 6 \\ 5x_3 &= -15 \end{aligned}$$

The solution is

$$x_1 = -1 \quad x_2 = \frac{6}{3} = 2 \quad x_3 = \frac{-15}{5} = -3$$

Solving Diagonal Systems (3)

Listing 1: Diagonal System Solution

```
1 given A, b
2 for i = 1...n
3      $x_i = b_i/a_{i,i}$ 
4 end
```

In MATLAB:

```
1 >> A = ...           % A is a diagonal matrix
2 >> b = ...
3 >> x = b./diag(A)
```

This is the *only* place where element-by-element division (`./`) has anything to do with solving linear systems of equations.

Triangular Systems (1)

The generic lower and upper triangular matrices are

$$L = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & & 0 \\ \vdots & & \ddots & \vdots \\ l_{n1} & & \cdots & l_{nn} \end{bmatrix}$$

and

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & & u_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & u_{nn} \end{bmatrix}$$

The triangular systems

$$Ly = b \quad Ux = c$$

are easily solved by **forward substitution** and **backward substitution**, respectively

Solving Triangular Systems (2)

$$A = \begin{bmatrix} -2 & 1 & 2 \\ 0 & 3 & -2 \\ 0 & 0 & 4 \end{bmatrix} \quad b = \begin{bmatrix} 9 \\ -1 \\ 8 \end{bmatrix}$$

Solving Triangular Systems (3)

$$A = \begin{bmatrix} -2 & 1 & 2 \\ 0 & 3 & -2 \\ 0 & 0 & 4 \end{bmatrix} \quad b = \begin{bmatrix} 9 \\ -1 \\ 8 \end{bmatrix}$$

is equivalent to

$$\begin{array}{rclcl} -2x_1 & + & x_2 & + & 2x_3 & = & 9 \\ & & 3x_2 & + & -2x_3 & = & -1 \\ & & & & 4x_3 & = & 8 \end{array}$$

Solving Triangular Systems (4)

$$A = \begin{bmatrix} -2 & 1 & 2 \\ 0 & 3 & -2 \\ 0 & 0 & 4 \end{bmatrix} \quad b = \begin{bmatrix} 9 \\ -1 \\ 8 \end{bmatrix}$$

is equivalent to

$$\begin{aligned} -2x_1 + x_2 + 2x_3 &= 9 \\ 3x_2 + -2x_3 &= -1 \\ 4x_3 &= 8 \end{aligned}$$

Solve in backward order (last equation is solved first)

$$x_3 = \frac{8}{4} = 2$$

$$x_2 = \frac{1}{3}(-1 + 2x_3) = \frac{3}{3} = 1$$

$$x_1 = \frac{1}{-2}(9 - x_2 - 2x_3) = \frac{4}{-2} = -2$$

Solving Triangular Systems (5)

Solving for x_1, x_2, \dots, x_n for a lower triangular system is called **forward substitution**.

```
1  given  $L, b$ 
2   $x_1 = b_1/\ell_{11}$ 
3  for  $i = 2 \dots n$ 
4       $s = b_i$ 
5      for  $j = 1 \dots i - 1$ 
6           $s = s - \ell_{i,j}x_j$ 
7      end
8       $x_i = s/\ell_{i,i}$ 
9  end
```

Solving Triangular Systems (6)

Solving for x_1, x_2, \dots, x_n for a lower triangular system is called **forward substitution**.

```
1 given  $L, b$ 
2  $x_1 = b_1/l_{11}$ 
3 for  $i = 2 \dots n$ 
4    $s = b_i$ 
5   for  $j = 1 \dots i - 1$ 
6      $s = s - l_{i,j}x_j$ 
7   end
8    $x_i = s/l_{i,i}$ 
9 end
```

Using forward or backward substitution is sometimes referred to as performing a **triangular solve**.

What's Next?

- 1 Gaussian Elimination on full systems
 - ▶ expense?
 - ▶ coding pitfalls (a.k.a. what can go wrong?)
- 2 Factorizations
- 3 What's done in practice?
 - ▶ sparse matrices
 - ▶ BLAS
 - ▶ LAPACK
 - ▶ Iterative Methods