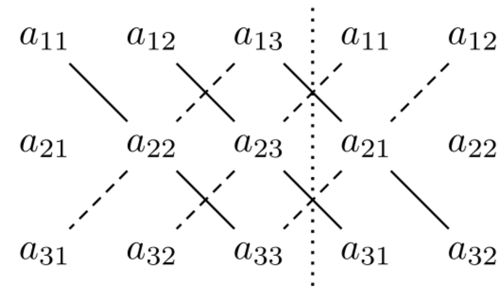
Assignment 1: Sarrus' Method



Sarrus' rule or Sarrus' scheme is a method and a memorization scheme to compute the determinant of a 3×3 matrix.

Consider a 3×3 matrix

$$M = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix},$$

then its determinant can be computed by the following scheme:

Write out the first 2 columns of the matrix to the right of the 3rd column, so that you have 5 columns in a row. Then add the products of the diagonals going from top to bottom (solid) and subtract the products of the diagonals going from bottom to top (dashed). These yields:

Let det(A) be the determinant of the diagonal element from top to bottom(solid).
Let det(B) be the determinant of the diagonals going from bottom to top (dashed).

det (A) = $a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32}$
det (B) = $a_{31}a_{22}a_{13} + a_{32}a_{23}a_{11} + a_{33}a_{21}a_{12}$

Therefore according to the rule of Sarrus Method,
det (M) = det(A) – det(B)

$$\det(M) = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}.$$

But for 2 X 2 matrix or n X n matrix (except 3X3 matrix) Sarrus method is not applicable.
Let us take an example for 2 X2 matrix:

$$M= \begin{bmatrix} 2 & \mathbf{3} \\ \mathbf{1} & \mathbf{4} \end{bmatrix}$$

det (A) = (2*4) + (3*1)
= 8+3
= 11

det (B) = (3*1) + (4*2)
=11

Therefore,
det (M) = det(A) – det(B)
= 11 – 11
= 0

But the determinant of this matrix is:
det (M) = (2*4) – (1*3)
= 8 – 3
= 5

So we can conclude that Sarrus method is only applicable for 3 X 3 matrix not for any other ordered matrix.

Pseudo Code:

```
// a[i][j]: Value of coefficient matrix row wise

float detcalc(int n,int arr[n][n])
{
    float a,b,d,t;
    int i,j;

    a=0.0;
    b=0.0;

    for(i=0 to n)
    {
        t=1.0;

        for(j=0 to n)
            t=t*arr[j][(j+i+n)%n];

        a=a+t;

        t=1.0;

        for(j=0 to n)
            t=t*arr[n-j-1][(j+i+n)%n];

        b=b+t;
    }
    d=a-b;
    // Return the values.
}
```

Example:

Enter the value of n : 3

Enter the value of the matrix:

Row 1. 1 2 3
Row 2. 4 5 6
Row 3. 7 8 9

Determinant of the matrix : 0

Assignment 2: Addition, Subtraction, Multiplication and Division

Pseudo Code:

```
void add()
{
	// Variables for loops and data
	for(i=0 to n-1)
		for(j=0 to n-1)
			c[i][j]=a[i][j]+b[i][j];
}
void sub(int n, int a[n][n], int b[n][n], int c[n][n])
{
	//Variables for loops and data
	for(i=0 to n-1)
		for(j=0 to n-1)
			c[i][j]=b[i][j]-a[i][j];
}
void mult(int n, int a1[n][n], int a2[n][n], int a3[n][n])
{
	// Variables for loop and data.
	for(i=0 to n-1)
	{
		for(j=0 to n-1)
		{
			m=0;
			for(k=0 to n-1)
				m=m+(a1[i][k]*a2[j][k]);
			a3[i][j]=m;
		}
	}
}
void trans(int n, int a1[n][n], int a2[n][n])
{
	int i,j;
	for(i=0 to n-1)
	{
		for(j=0 to n-1)
			a2[i][j]=a1[j][i];
	}
}
```

Example:

Enter the order of the matrix: 3

Choice: 3

Enter the value for matrix 1:
Row 1. 2 3 5
Row 2. 4 1 2
Row 3. 6 3 1

Enter the value for matrix 2:
Row 1. 1 3 2
Row 2. 4 3 1
Row 3. 2 4 3

The product is:
24 35 22
12 23 15
20 31 18

Assignment 3: Upper Triangular Method

In the mathematical discipline of linear algebra, a triangular matrix is a special kind of square matrix. A square matrix is called lower triangular if all the entries *above* the main diagonal are zero. Similarly, a square matrix is called upper triangular if all the entries *below* the main diagonal are zero.
A triangular matrix **U** of the form

$$U_{i\,j} = \begin{cases} a_{i\,j} & \text{for } i \le j \\ 0 & \text{for } i > j. \end{cases}$$

Written explicitly,

$$\mathsf{U} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1\,n} \\ 0 & a_{22} & \cdots & a_{2\,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n\,n} \end{bmatrix}.$$

Pseudo Code:

```
// a[i][j]: Value of coefficient matrix row wise.
void uptri()
{
        // Variable declarations.

        for (i=0 to n)
        {
                for ( j=i+1 to n)
                {
                        t=a1[j][i]/a1[i][i];
                        for (k=0 to n)
                        {
                                a1[j][k]=a1[j][k]-(a1[i][k]*t);
                        }
                }
        }
}
```

Example:

```
Enter the value of n: 3
Enter the value of the matrix:
Row 1.    5    7    2
Row 2.    1    3    8
Row 3.    1    1    1
Determinant of the matrix : 20
```

Assignment 4: Cramer's Rule

- **Given a linear system**

$$
\begin{array}{c}
\text{x-column} \qquad \text{z-column} \\
\downarrow \qquad\qquad \downarrow \\
a_1x + b_1y + c_1z = d_1 \\
a_2x + b_2y + c_2z = d_2 \\
a_3x + b_3y + c_3z = d_3 \\
\uparrow \qquad\qquad \uparrow \\
\text{y-column} \qquad \text{constant-column}
\end{array}
$$

- **Labeling each of the four matrices**

coefficient matrix: $D = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}$

X – matrix: $D_x = \begin{bmatrix} d_1 & b_1 & c_1 \\ d_2 & b_2 & c_2 \\ d_3 & b_3 & c_3 \end{bmatrix}$

Y – matrix: $D_y = \begin{bmatrix} a_1 & d_1 & c_1 \\ a_2 & d_2 & c_2 \\ a_3 & d_3 & c_3 \end{bmatrix}$

Z – matrix: $D_z = \begin{bmatrix} a_1 & b_1 & d_1 \\ a_2 & b_2 & d_2 \\ a_3 & b_3 & d_3 \end{bmatrix}$

- **To solve for x:**

$$x = \frac{|D_x|}{|D|} = \frac{\begin{vmatrix} d_1 & b_1 & c_1 \\ d_2 & b_2 & c_2 \\ d_3 & b_3 & c_3 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}}$$

- **To solve for y:**

$$y = \frac{|D_y|}{|D|} = \frac{\begin{vmatrix} a_1 & d_1 & c_1 \\ a_2 & d_2 & c_2 \\ a_3 & d_3 & c_3 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}}$$

- **To solve for z:**

$$z = \frac{|D_z|}{|D|} = \frac{\begin{vmatrix} a_1 & b_1 & d_1 \\ a_2 & b_2 & d_2 \\ a_3 & b_3 & d_3 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}}$$

Pseudo Code:

```
// a[i][j]: Values of coefficient matrix.
// b[i]: Values of right hand side constants.

float cramer()
{
	// Variable declarations
	for(i=0 to n)
	{
		for(j=i+1 to n)
		{
			t=a1[j][i]/a1[i][i];
			for(k=0 to n)
			{
				a1[j][k]=a1[j][k]-(a1[i][k]*t);
			}
		}
	}

	t=1;

	for(i=0 to n)
		t=t*a1[i][i];

	return t;
}
```

Example:

```
Enter the value of n: 3
Enter the value of the matrix:
Row 1.    6    5    2
Row 2.    1    3    1
Row 3.    1    2    3

X[0] = 3.000000    X[1] =    2.000000 X[2] =    1.000000
```

Assignment 5: Inverse of a Matrix

In linear algebra, an *n*-by-*n* square matrix A is called invertible if there exists an *n*-by-*n* square matrix B such that

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I}_n$$

where, I_n denotes the *n*-by-*n* identity matrix and the multiplication used is ordinary matrix multiplication.

If this is the case, then the matrix B is uniquely determined by A and is called the *inverse* of A, denoted by A^{-1}.

The adjugate of a matrix can be used to find the inverse of as follows:

If is an invertible matrix, then

$$A^{-1} = \frac{1}{\det(A)} \operatorname{adj}(A)$$

Pseudo Code:

```
void invert()
{
	// Declarations.

		d = det(n,a);		// det() finds the determinant
	if(d==0)
		// Non-Invertible Matrix.
	// Calculating the co-factor matrix.
	for(i=0 to n)
	{
		for(j=0 to n)
		{
			t=a1[i][j];
			m=0;
			for(k=0 to n)
			{
				if(k==i)
					continue;
				o=0;
				for(l=0 to n)
				{
					if(l==j)
						continue;
					b[m][n]=a1[k][l];
					n++;
				}
				m++;
			}
			a1[i][j]=pow(-1,i+j)*det(n-1,b); // det() finds the determinant
		}
```

```
        }

        // Transposing the co-factor matrix to find adjoint matrix.
        for(i=0 to n)
        {
                for(j=0 to n)
                {
                        t=a1[i][j];
                        a1[i][j]=a1[j][i];
                        a1[j][i]=t;
                }
        }

        // Print the matrix while dividing with determinant.
}
```

Example:

Enter the value of n-> 3
Enter the value of the matrix:

Row 1.	5	7	2
Row 2.	1	3	8
Row 3.	1	1	1

The inverse is:

-0.2500	-0.2500	2.5000
0.3500	0.1500	-1.9000
-0.1000	0.1000	0.8000

Assignment 6: Gauss Elimination Method

This is the elementary elimination method and it reduces the system of equations to an equivalent upper triangular matrix, which can be solved by back substitution.

Pseudo Code:

```
void gauss()
{
   //c[i]=right hand side constants
   //a[i][j]=values row wise
      for(k=0 to n-1)
      {
       for(i=k+1 to n)
          {
                  for(j=k+1 to n)
                          a[i][j]=a[i][j]-(a[i][k]/a[k][k])*a[k][j];
                      c[i]=c[i]-(a[i][k]/a[k][k])*c[k];
      }
   }
   x[n-1]=c[n-1]/a[n-1][n-1];
   print(the solution is: n-1,x[n-1]   );

   for(k=0 to n-1)
   {

      i=n-k-2;
      for(j=i+1 to n)
         c[i]=c[i]-(a[i][j]*x[j]);
      x[i]=c[i]/a[i][i];
      print(solution : x[i]);
   }
}
```

Example:

Input:

```
Enter the matrix value: 3
6 5 2 30
1 3 1 10
1 2 3 10
```

Output:

```
Original matrix is:
6.000000     5.000000     2.000000     30.000000
1.000000     3.000000     1.000000     10.000000
1.000000     2.000000     3.000000     10.000000
Combined Upper Triangular Matrix is:
6.000000     5.000000     2.000000     30.000000
-0.000000    2.166667     0.666667     5.000000
-0.000000    -0.000000    2.307693     2.307693

X[0] = 3.000000 X[1] = 2.000000 X[2] = 1.000000
```

Assignment 7: Newton's Forward Interpolation Method

This is an N^{th} degree polynomial approximation formula to the function **f(x)**. If $y_0, y_1, y_2, \ldots\ldots, y_n$ are the values of $y = f(x)$ corresponding to equidistant values of $x = x_0, x_1, x_2, \ldots\ldots, x_n$, where $x_i - x_i\text{-}1 = h$, for i = 1, 2, 3, …..,n, then $y = y_0 + u/!1\Delta y_0 + u(u-1)/2!\Delta^2 y_0 + \ldots\ldots\ldots\ldots + u(u-1)\ldots\ldots(u- n-1)/n!\ \Delta^n y_0$, where $u = (x - x_0)/h$.

Pseudo Code:

```
Function NFI ()
 Read n, x
For I = 1 to n by 1 do
Read x[i], y[i]
End for
If ((x < x[i] or (x > x[n]))
   Print "Value lies out of boundary"

//Calculating p
p = (x – x [1]) / (x [2]-x [1])

// Forward diff table
For j = 1 to (n-1) by 1 do
   {
    For i =1 to (n - j) by 1 do
       {
          If (j=1) Then
              d[i][j] = y [i+1] – y[i]
          Else
              d[i][1j] = d[i+1][j-1] – d[i][j-1]
       }
   }

// Applying Formula
Sum =y [1]
For I = 1 to (n-1) by 1 do
{
    Prod = 1
    For j =0 to (i-1) by 1 do
       Prod = prod * (p-j)

    m = fact(i)
    Sum = sum + (d[1][i] * prod) / m
}
Print "Ans is", Sum
End Function
```

Example:

```
Enter the value of n (No. of data pairs - 1):
7
Enter the initial values of x:
0
Enter the step size:
1
Enter the values of y
 1  2  4  7  11  16  22  29
Enter the required no. of interpolated values of y:
6
Enter the 6 values of X for which values of y are required:
 -0.4  0.8  3.5  5  6.3  7.7

The values of X and Y are: -0.400000   0.880000
The values of X and Y are: 0.800000    1.720000
The values of X and Y are: 3.500000    8.875000
The values of X and Y are: 5.000000    16.000000
The values of X and Y are: 6.300000    23.995003
The values of X and Y are: 7.700000    34.494999
```