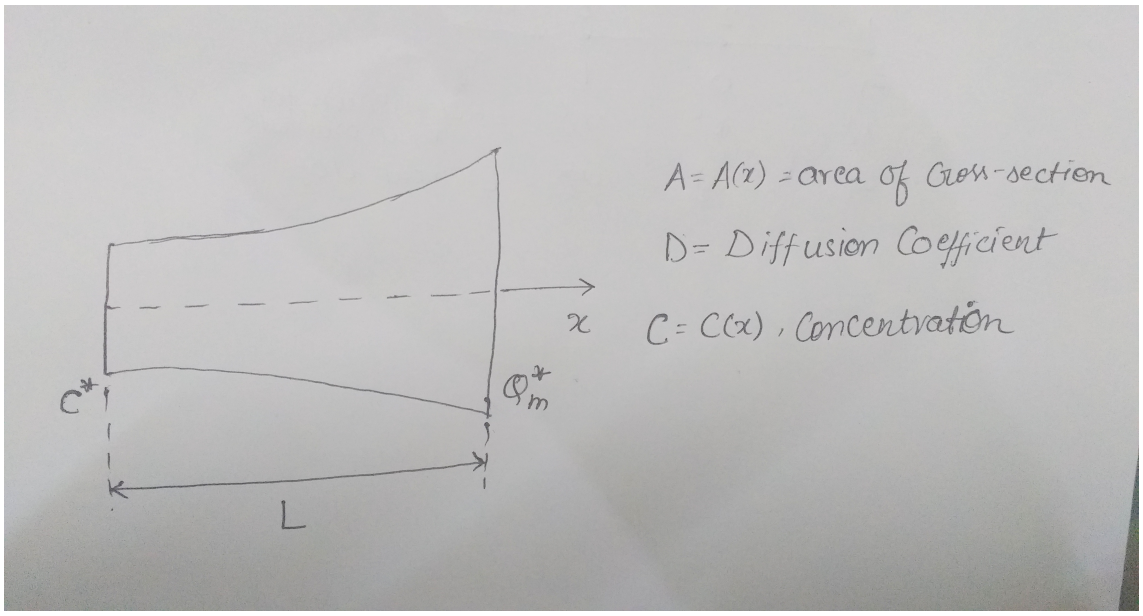
Diffusion in a Rod Problem



1 Variational Functional of the problem

$$\left[\int_0^l -\frac{1}{2}AD\left(\frac{dc}{dx}\right)^2 dx\right] + \left(-Q_m^* c\right)|_{x=l} \tag{1}$$

2 Information regarding shape function, derivative of shape function and element

1. Shape function

$$N_1^e = \frac{(\xi - \xi_2^e)(\xi - \xi_3^e)}{(\xi_1^e - \xi_2^e)(\xi_1^e - \xi_3^e)}$$

$$N_2^e = \frac{(\xi - \xi_1^e)(\xi - \xi_3^e)}{(\xi_2^e - \xi_1^e)(\xi_2^e - \xi_3^e)} \tag{2}$$

$$N_3^e = \frac{(\xi - \xi_1^e)(\xi - \xi_2^e)}{(\xi_3^e - \xi_1^e)(\xi_3^e - \xi_2^e)}$$

2. Differentiation of shape function

$$\frac{dN_1^e}{d\xi} = \frac{l^e}{2}\frac{(\xi - \xi_3^e) + (\xi - \xi_2^e)}{(\xi_1^e - \xi_2^e)(\xi_1^e - \xi_3^e)}$$

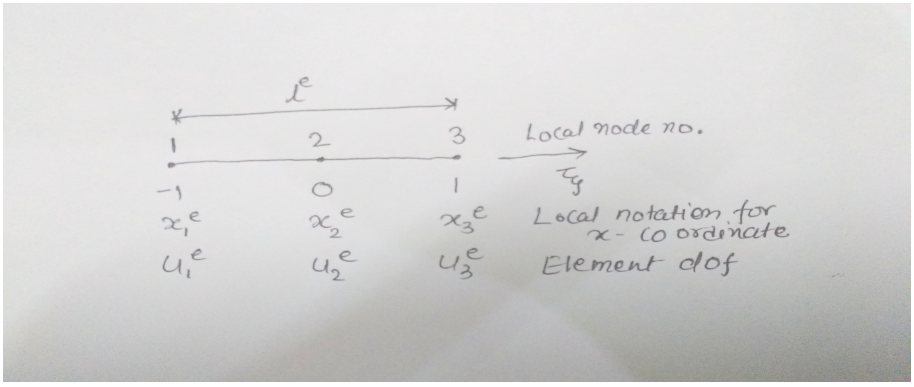
$$\frac{dN_2^e}{d\xi} = \frac{l^e}{2}\frac{(\xi - \xi_3^e) + (\xi - \xi_1^e)}{(\xi_2^e - \xi_1^e)(\xi_2^e - \xi_3^e)} \tag{3}$$

$$\frac{dN_3^e}{d\xi} = \frac{l^e}{2}\frac{(\xi - \xi_2^e) + (\xi - \xi_1^e)}{(\xi_3^e - \xi_1^e)(\xi_3^e - \xi_2^e)}$$

3. Mapping Function

$$x = \frac{x_m^e + x_1^e}{2} + \frac{l^e}{2}\xi \tag{4}$$

4. Element showing no. and location of nodes



3 Expression for element coefficient matrix $[k]^e$ and right side vector $[f]^e$

$$[k]^e = \int_{x_1^e}^{x_3^e} AD[B]^e[B]^{eT}dx \tag{5}$$

$$[f]^e = 0 \tag{6}$$

$$[B]^e = \frac{dN_i^e}{dx} \tag{7}$$

$$[N]^e = \begin{bmatrix} N_1^e \\ N_2^e \\ N_3^e \end{bmatrix} \tag{8}$$

4 Simplified Assembly relations

1. Global Stiffness Matrix

$$[K] = \sum_{e=1}^{n_e} [K^e] \tag{9}$$

$$K_{rs}^e = k_{pq}^e \text{ when } r = c_{ep} \text{ and } s = c_{eq} \tag{10}$$
$$= 0 \text{ otherwise}$$

2. Global right side Vector

$$[F] = Q^* \tag{11}$$

n_e = number of elements

$[c]$ = connectivity matrix

5 The numerical integration scheme (with Gauss point co-ordinates / weights)

$$[k]^e = \sum_{k=1}^{n_G} w_k \left(\frac{2AD}{l^e} [B]^{e`} [B]^{e`T} \right) |_{\xi=\xi^k} \tag{12}$$

$$[B]^{e`} = \frac{dN^e}{d\xi} \tag{13}$$

5.1 Choice of number of Gauss points

$$[N]^e = \text{ polynomial of degree 2}$$

$$[B]^e = \text{ polynomial of degree 1}$$

$$A = \sum_{i=1}^{m=3} N_i^e A_i^e \text{ which is polynomial of degree 2}$$

$$\text{Degree of integrand } [k]^e : n = 2 + 1 + 1 = 5 => n_G = \frac{n+1}{2} = \frac{5}{2} \approx 3$$

as we have $[f]^e = 0$ n_G will be same as calculated above.

$$\xi^k = \begin{bmatrix} -0.77460 & 0 & 0.77460 \end{bmatrix} \tag{14}$$

corresponding weights for the problem

$$w_k = \begin{bmatrix} 0.55556 & 0.88889 & 0.55556 \end{bmatrix} \tag{15}$$

6 Computer Code

```
#include <bits/stdc++.h>

using namespace std;


        // defining function to calculate shape function at specified gauss point

        float shapefunction(float psi[],int k,int m,int i)
        {
                float N; // value of shape function at specified gauss point
                int j;
                float n=1.0,d=1.0; // numerator and denominator in shape function
                                    respectively

                for(j=0;j<m;j++)
                {
                        if(j!=i)
                        {
                                n=n*(psi[k]-psi[j]);

                        }
                }

                for(j=0;j<m;j++)
                {
                        if(j!=i)
                        {
                                d=d*(psi[i]-psi[j]);

                        }
                }

                N=n/d;

                return N;
        }

        // defining function to calculate differentiation of shape function at
    specified gauss point

        float dshapefunction(float psi[],int k,int m,int i)
        {
                float dN; // value of differentiation of shape function at specified
                     gauss point
                int j;
                float n=0.0,d=1.0; // numerator and denominator in differentiation of
                                    shape function respectively

                for(j=0;j<m;j++)
                {
                        if(j!=i)
                        {
                                n=n+(psi[k]-psi[j]);
```

```
                }
        }

        for(j=0;j<m;j++)
        {
                if(j!=i)
                {
                        d=d*(psi[i]-psi[j]);

                }
        }

        dN=n/d;

        return dN;
}

int main()

{
        int n_e=20; //number of elements

        int m=3; // number of nodes per element

        int n; // total number of nodes in the FEM analysis
        n=(n_e*(m-1))+1;


        int c[n_e][m]; //connectivity matrix

        int i,j,h=1,k,p,q,r,s; // dummy variables

        float c1=30; //specified concentration at left end

        float D=6*pow(10,-12); // diffusion coefficient

        float Q_m=1.2*pow(10,-9); // specified mass flow rate at right end

        // initializing connectivity matrix

        for (i=0;i<n_e;i++)
        {
                for(j=0;j<m;j++)
                {
                        c[i][j]=j+h;
                }
                h=c[i][m-1];
        }

        float L=1;// length of the rod in m

        // defining global co-ordinate vector

        float X[n];
```

```
for(i=0;i<n;i++)
{
        X[i]=(L/(n-1))*i;
}

float K[n][n]; // global stiffness matrix

for(i=0;i<n;i++)
{
        for(j=0;j<n;j++)
        {
                K[i][j]=0.0;
        }
}

float k_e[m][m]; // elemental stiffness matrix

for(i=0;i<m;i++)
{
        for(j=0;j<m;j++)
        {
                k_e[i][j]=0.0;
        }
}

float F[n]; //global right side vactor

for(i=0;i<n;i++)
{
        F[i]=0;
}

float f[m]; // elemental or local right side vector

for(i=0;i<m;i++)
{
        f[i]=0.0;
}

float C[n]; // global notation for primary variable or
            concentration in this case

for(i=0;i<n;i++)
{
        C[i]=0;
}

float x[m]; // local or elemental co-ordinate vector

int e; // dummy variable to run through all elements

float l; // length of an element

int n_G=3; // number of gauss points for numerical integration

float psi[n_G]={-0.77460,0.0,0.77460}; // natural co-ordinates
```

```
float w[n_G]={0.55556,0.88889,0.55556}; //weights corresponding to
                                        natural co-ordinates

float A[n]; // area at the defined nodes

// initializing area at all nodes
for(i=0;i<n;i++)
{
        A[i]=3+(4*X[i]);
}

float a[m]; // area defined for m-noded element

l=X[m-1]-X[0]; // length of the element

float A_e=0.0; // area at specified gauss point for numerical integration

float B[m]; // matrix containing value of differentiation of shape function

float G[m][m]; // matrix as a result of multiplying matrix B and transpose(B)

// computing [k] and [f] over all elements
for(e=0;e<n_e;e++)  // addition changing element to 1
{
    for(i=0;i<m;i++)
    {
        j=c[e][i];
        x[i]=X[j-1];

        }
        for(i=0;i<m;i++)
        {
              j=c[e][i];
              a[i]=A[j-1];

        }

        // computing numerical integration for calculating [k] and [f]
     according to weights
        for(k=0;k<n_G;k++)
        {
            A_e=0.0;
                for(i=0;i<m;i++)
                {
                      //value of shape function at psi[k]
                         A_e=A_e+((shapefunction(psi,k,m,i))*a[i]);
                }

                for(i=0;i<m;i++)
                {
                        // value of differentiation of shape function at psi[k]
                        B[i]= dshapefunction(psi,k,m,i);

                }
```

```
                // carrying out matrix multiplication
                for(i=0;i<m;i++)
                {
                        for(j=0;j<m;j++)
                        {
                                G[i][j]=B[i]*B[j];
                        }
                }


                for(i=0;i<m;i++)
                {
                        for(j=0;j<m;j++)
                        {
                                k_e[i][j]=k_e[i][j]+((w[k]*A_e*D*2*G[i][j])/l);
                        }
                }

                // [f] for all elements is 0 in our case as we dont have any
            c term in our variational functional
        }

        // Assembly of global stiffness matrix

        for(p=0;p<m;p++)
        {
            r=c[e][p];

            F[r-1]=F[r-1]+f[p];

                for(q=0;q<m;q++)
                {
                    s=c[e][q];

                   K[r-1][s-1]=K[r-1][s-1]+k_e[p][q];
                }
        }

}

//Application of essential boundary conditions
F[n-1]=-Q_m;
C[0]=c1;

// removing the meaningless row and modifying the matrices

float K_new[n-1][n-1];    // new stiffness matrix for computation

for(i=0;i<n-1;i++)
{
        for(j=0;j<n-1;j++)
        {
                K_new[i][j]=K[i+1][j+1];
        }
}
```

```
    float F_new[n-1]; // modified right hand side vector

    for(i=0;i<n-1;i++)
    {
            F_new[i]=F[i+1]-(K[i+1][0]*C[0]);
    }

    // modified Concentration matrix
    float C_new[n-1];

    for(i=0;i<n-1;i++)
    {
            C_new[i]=0;
    }


    //Calculating the concentration at each nodes using gauss seidel method

    float t=0.001,E[n]; // t is tolerance and E is error matrix

int z=1,v=0; // dummy variables

    float y[n-1]; // storing values for previous iteration for comparison

    float sum=0.0; // dummy variable for summation

while(z!=0)
    {     v=0;
        for(i=0;i<n-1;i++)
        {
            y[i]=C_new[i];
            }

    for(i=0;i<n-1;i++)
    {     sum=0;

            for(j=0;j<n-1;j++)
            {
                        if(j!=i)
                        {
          sum=sum+(K_new[i][j]*C_new[j]);

                        }
                }

                    C_new[i]=(F_new[i]-sum)/K_new[i][i];

            }


            for(i=0;i<n-1;i++)
            {
                    E[i]=abs(C_new[i]-y[i]);
                    if(t<E[i])
                    {
                            v=1;
```

```
                        }
                }

                if(v==0)
                {
                        z=0;
                }

        }

        // substituting value of C_new to original C matrix
        for(i=0;i<n-1;i++)
        {
             C[i+1]=C_new[i];
             }


        cout<<endl<<"Using the Gauss Siedel the calculated solution for concentration is:";
        cout<<endl<<endl;

        for(i=0;i<n;i++)
        {
             cout<<C[i]<<endl;
             }

}
```

7 Input 1

$$n_e = \text{number of elements } = 20$$

$$m = \text{number of nodes per element } = 3$$

$$D = \text{Diffusion Coefficient } = 6 * 10^{-12} \frac{m^2}{s}$$

$$c1 = \text{ concentration at left end denoted by } c^* \text{ in the problem } = 30 \frac{kg}{m^3}$$

$$Q_m = \text{ mass flow rate at right end denoted by } Q_m^* \text{ in the problem } = 1.2 * 10^{-9} \frac{kg}{m^2 s}$$

$$L = \text{length of the rod } = 1m$$

$$A(x) = \text{ area of the rod } = 3 + 4x$$

$$t = \text{tolerance for Gauss Seidel } = 0.001$$

8 Input 2

$$n_e = \text{number of elements } = 40$$

$$m = \text{number of nodes per element } = 3$$

$$D = \text{Diffusion Coefficient } = 6 * 10^{-12} \frac{m^2}{s}$$

$$c1 = \text{ concentration at left end denoted by } c^* \text{ in the problem } = 30 \frac{kg}{m^3}$$

$$Q_m = \text{ mass flow rate at right end denoted by } Q_m^* \text{ in the problem } = 1.2 * 10^{-9} \frac{kg}{m^2 s}$$

$$L = \text{length of the rod } = 1m$$

$$A(x) = \text{ area of the rod } = 3 + 4x$$

$$t = \text{tolerance for Gauss Seidel } = 0.001$$

9 Output 1

Table 1: Diffusion in a Rod for Input 1

Distance x(m)	Concentration c(kg/m$\hat{3}$)
0	30
0.025	27.9396
0.05	25.945
0.075	24.9488
0.1	23.9837
0.125	23.3419
0.15	22.7197
0.175	22.2549
0.2	21.8042
0.225	21.4456
0.25	21.0979
0.275	20.8102
0.3	20.5312
0.325	20.2943
0.35	20.0645
0.375	19.8658
0.4	19.6732
0.425	19.5043
0.45	19.3406
0.475	19.1958
0.5	19.0555
0.525	18.9305
0.55	18.8094
0.575	18.701
0.6	18.5962
0.625	18.502
0.65	18.411
0.675	18.3291
0.7	18.2501
0.725	18.179
0.75	18.1105
0.775	18.049
0.8	17.9897
0.825	17.9368
0.85	17.8859
0.875	17.8406
0.9	17.7972
0.925	17.7589
0.95	17.7224
0.975	17.6905
1	17.6602

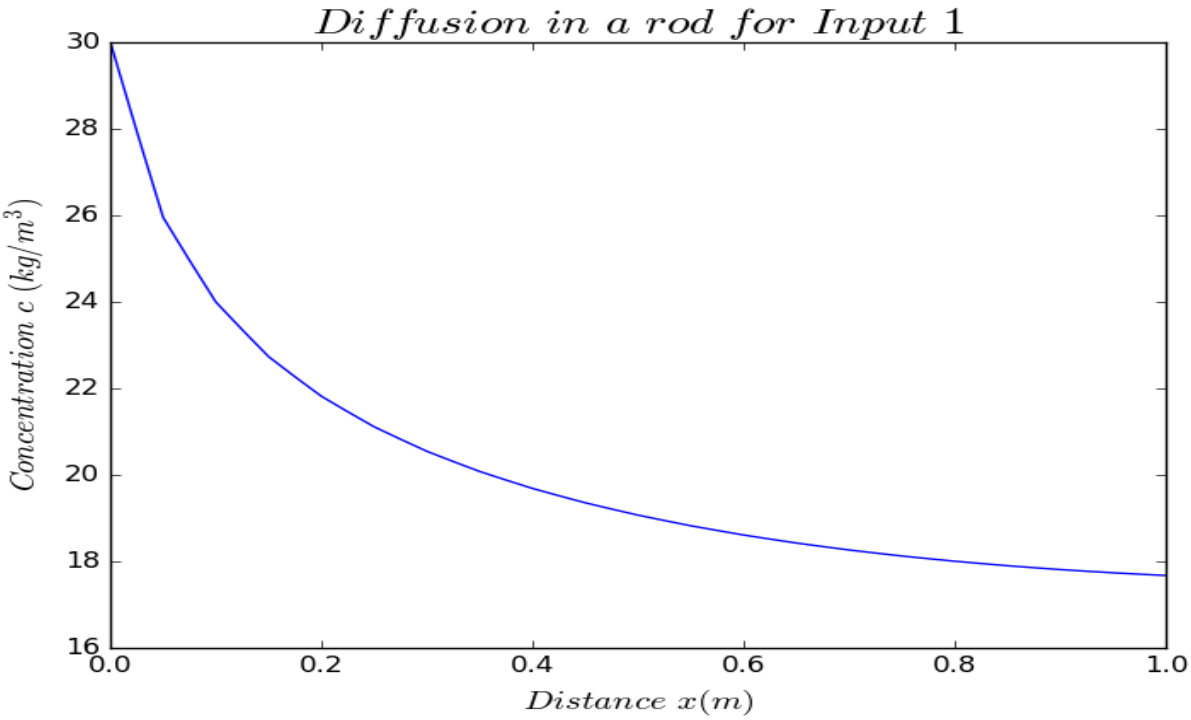
10 Output 2

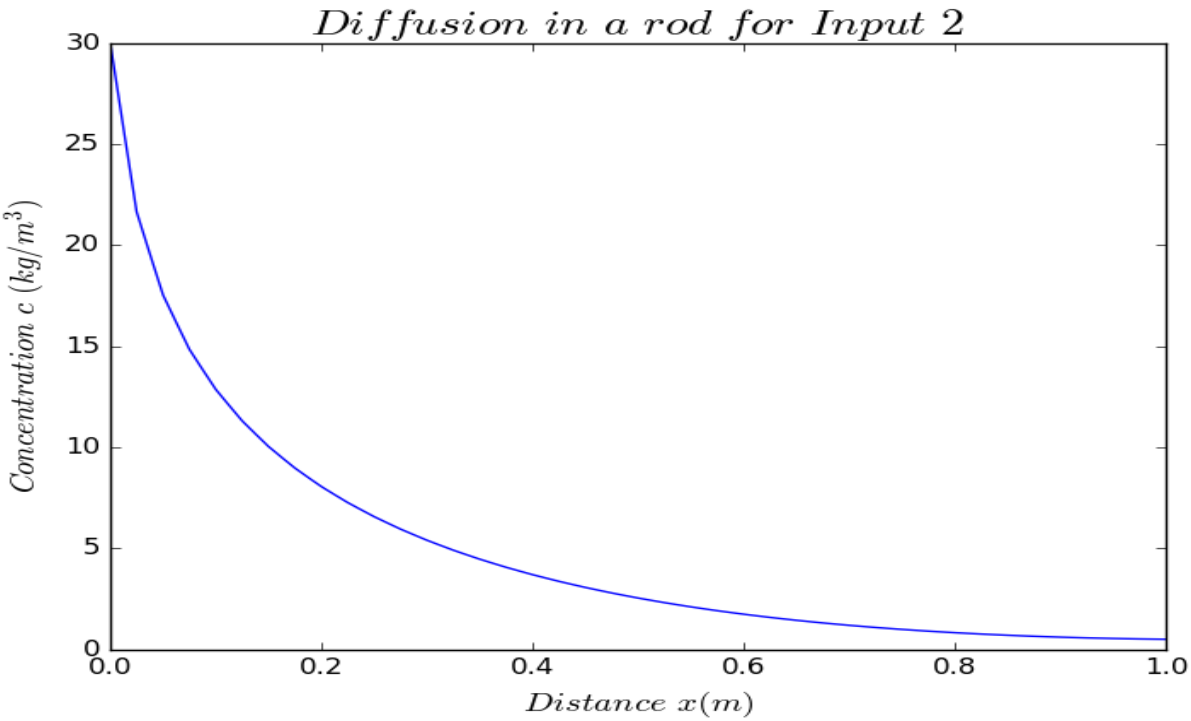
Table 2: Diffusion in a rod for Input 2

Distance x(m)	Concentration c(kg/m$\hat{3}$)
0	30
0.0125	25.7845
0.025	21.638
0.0375	19.5659
0.05	17.5275
0.0625	16.1702
0.075	14.8349
0.0875	13.8354
0.1	12.8522
0.1125	12.0677
0.125	11.2961
0.1375	10.6554
0.15	10.0252
0.1625	9.48751
0.175	8.95869
0.1875	8.49853
0.2	8.04603
0.2125	7.64645
0.225	7.25358
0.2375	6.90269
0.25	6.55775
0.2625	6.24692
0.275	5.94142
0.2875	5.66415
0.3	5.3917
0.3125	5.14299
0.325	4.89867
0.3375	4.67459
0.35	4.45452
0.3625	4.2519
0.375	4.05296
0.3875	3.86924
0.4	3.6889
0.4125	3.52192
0.425	3.35807
0.4375	3.20605
0.45	3.05694
0.4625	2.91837
0.475	2.7825
0.4875	2.65608
0.5	2.53217

Table 3: Diffusion in a rod for Input 2

Distance x(m)	Concentration c(kg/m$\hat{3}$)
0.5125	2.41678
0.525	2.30374
0.5375	2.19842
0.55	2.09528
0.5625	1.99917
0.575	1.9051
0.5875	1.81745
0.6	1.73171
0.6125	1.65185
0.625	1.57379
0.6375	1.50115
0.65	1.43018
0.6625	1.36421
0.675	1.29983
0.6875	1.24007
0.7	1.1818
0.7125	1.12783
0.725	1.07526
0.7375	1.02669
0.75	0.979446
0.7625	0.935938
0.775	0.893683
0.7875	0.854927
0.8	0.817358
0.8125	0.783078
0.825	0.749922
0.8375	0.719865
0.85	0.690876
0.8625	0.664818
0.875	0.639775
0.8875	0.617512
0.9	0.596216
0.9125	0.577566
0.925	0.559839
0.9375	0.54464
0.95	0.530323
0.9625	0.518427
0.975	0.507377
0.9875	0.498654
1	0.490745


Diffusion in a rod for Input 1
Concentration c (kg/m^3)
Distance x(m)
30
28
26
24
22
20
18
16
0.0
0.2
0.4
0.6
0.8
1.0


Diffusion in a rod for Input 2
Concentration c (kg/m^3)
Distance x(m)
30
25
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0