Computer Routine for Structural Analysis

Suyog Garg

Semester Course Project
Advanced Mechanics of Mechanics

Department of Mechanical Engineering
IIITDM Kancheepuram
Chennai 600127
April 14, 2019

Contents

Chapter 1

Introduction

The method of discretizing complex geometries and finding piece-wise continuous functions for each of the subdomains, also known as, Finite Element Method (FEM), is highly successful in the structural analysis of many complicated mechanical components. A multi-system vehicle such as Rocket consisting of systems like propulsion, payload, recovery, electronics etc., present unique oppurtunities of applying the FEM. The varrying loading conditions on the Rocket can be roughly modelled assuming the rocket body to be a beam. The transverse forces acting the body such as the drag force, alongwith the trust of the rocket, can be used to find forces and stress in the components.

A rocket is made of different layers which each serve their own purpose. Figure 1.1 below shows a cut-section view of a typical rocket body.

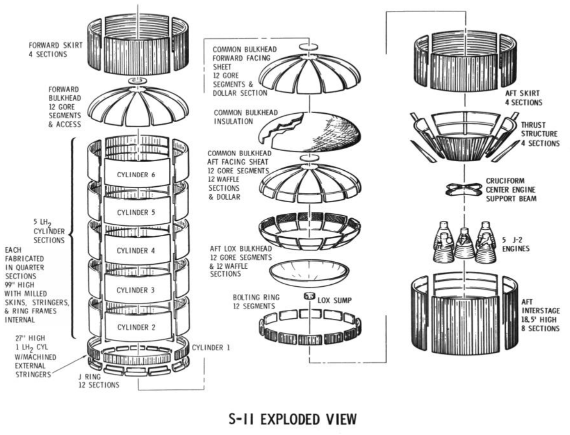
FORWARD SKIRT 4 SECTIONS
FORWARD BULKHEAD 12 GORE SEGMENTS & ACCESS
CYLINDER 6
CYLINDER 5
CYLINDER 4
CYLINDER 3
CYLINDER 2
CYLINDER 1
5 LH_2 CYLINDER SECTIONS
EACH FABRICATED IN QUARTER SECTIONS 99" HIGH WITH MILLED SKINS, STRINGERS, & RING FRAMES INTERNAL
27" HIGH 1 LH_2 CYL W/MACHINED EXTERNAL STRINGERS
J RING 12 SECTIONS
COMMON BULKHEAD FORWARD FACING SHEET 12 GORE SEGMENTS & DOLLAR SECTION
COMMON BULKHEAD INSULATION
COMMON BULKHEAD AFT FACING SHEAT 12 GORE SEGMENTS 12 WAFFLE SECTIONS & DOLLAR
AFT LOX BULKHEAD 12 GORE SEGMENTS & 12 WAFFLE SECTIONS
BOLTING RING 12 SEGMENTS
LOX SUMP
AFT SKIRT 4 SECTIONS
THRUST STRUCTURE 4 SECTIONS
CRUCIFORM CENTER ENGINE SUPPORT BEAM
5 J-2 ENGINES
AFT INTERSTAGE 18.5' HIGH 8 SECTIONS
S-II EXPLODED VIEW

Figure 1.1: Exploded view of Rocket

The rocket frame is very similar to the fuselage of a commercial aircraft. It consists of a space frame made of aluminium, a riveted compartment and a welded compartment. For the project, I have focused on developing a C program code to do structural analysis of a part of the welded compartment. A small portion of this compartment, is like a space truss with forces acting in two directions in each node. These forces arise from the drag and trust of the rocket. The nodal displacements calculated can give insight into the deformation that may take place. Accessing the strength of such components is very important for the safety of the whole rocket and the payload. The analysis of the forces and displacements developed when the rocket in taking off and in motion, help in deciding on the type of the material to be used in the rocket.

This report has been divided into 4 sections: Introduction, Geometric Model, Mathematical Model and Code. The first section introduced the problem statement, followed by discussion on the geometry in the second section. The Third section tells how a space truss can be mathematically modelled using the Finite Element Method and how the nodal displacements can be calculated. In the fourth section, the formulation for obtaining the different forces acting on the rocket body is given. Finally, the actual computer routine written in C is elaborated upon and an example is given.

Chapter 2

Geometric Model

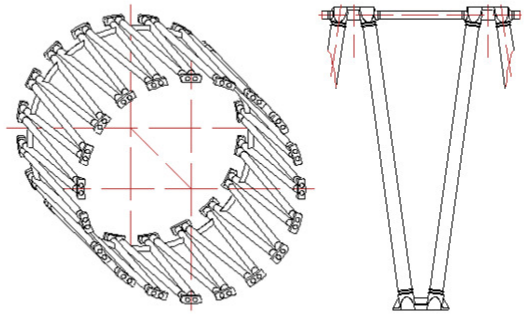

Figure 2.1: Cylindrical compartment of a rocket

Figure 2.1 shows the welded compartment of a rocket frame. This compartment is usually present in the aft bay of the rocket, and houses instrumentation, pipelines, cable harness, and protects the engine from the atmospheric forces acting on the body. It is a cylindrical waffle grid shell and frame comprising of triangular sections. Since, the frame is bounded above and below by other materials, a small portion of the compartment can only be displaced in the z and r directions. The motion along the θ direction and all rotations are contrained. Further, for this analysis a small portion can be assumed to be plane and not bend (as it is in reality), thus reducing the problem to that of a 2D Truss in 3D space. Such types of trusses are called 'Space Trusses' for which FEM, as described in the next section can be used.

For the Finite Element Method implimentation, the following geometric model (Figure 2.2) is used. The forces applied at nodes can be justified by doing a force analysis, as done sometime later. We only take one of the repeating triangular truss elements. Forces act at each of the nodes of this structure. A Spatial Truss has 3 degrees of freedom at each node, one each in x, y and z direction. A linear interpolation model is used to approximate the nodal displacements.

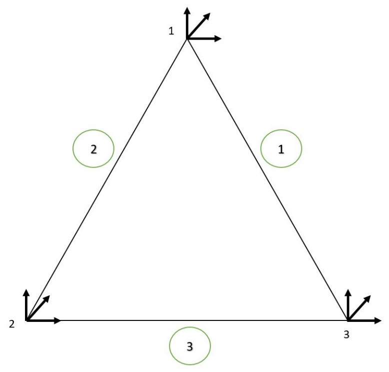

Figure 2.2: Finite Element Model for Space Truss

Chapter 3

Mathematical Model

A truss is one of the simplest and most widely used structural members. It is a straight bar that is designed to take only axial forces, therefore it deforms only in its axial direction. The cross-section of the bar can be arbitrary, but the dimensions of the cross-section should be much smaller than that in the axial direction. For space trusses there are three components in the x, y and z directions for both displacements and forces.

Consider a structure consisting of a number of trusses or bar members. Each of the members can be considered as a truss/bar element of uniform cross-section bounded by two nodes (n = 2). Consider a bar element with nodes 1 and 2 at each end of the element, as shown in Figure below.

Assume that the local nodes 1 and 2 of the element correspond to the global nodes i and j, respectively, as shown in Figure. The displacement at a global node in space should have three components in the x, y and z directions, and numbered sequentially. For example, these three components at the ith node are denoted by Q_{3i-2}, Q_{3i-1} and Q_{3i}. The coordinate transformation gives the relationship between the displacement vector q_e based on the local coordinate system and the displacement vector Q_e for the same element, but based on the global coordinate system XYZ:

$$q^e = \begin{bmatrix} l_{ij} & {}_{ij} & n_{ij} & 0 & 0 & 0 \\ 0 & 0 & 0 & l_{ij} & m_{ij} & n_{ij} \end{bmatrix}^e \begin{Bmatrix} Q_{3i-2} \\ Q_{3i-1} \\ Q_{3i} \\ Q_{3j-2} \\ Q_{3j-1} \\ Q_{3j} \end{Bmatrix}^e \tag{3.1}$$

The similar transformations are done for the force vector.

$$F^e = \begin{Bmatrix} F_{3i-2} \\ F_{3i-1} \\ F_{3i} \\ F_{3j-2} \\ F_{3j-1} \\ F_{3j} \end{Bmatrix} \tag{3.2}$$

Here, with L^e been the length of the element, the direction cosines l, m and n are defined as:

$$\begin{aligned} L^e &= \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} \\ l_{ij} &= \cos\theta_x = \frac{x_j - x_i}{L^e} \\ m_{ij} &= \cos\theta_y = \frac{y_j - y_i}{L^e} \\ n_{ij} &= \cos\theta_z = \frac{z_j - z_i}{L^e} \end{aligned} \tag{3.3}$$

The Element Stiffness Matrix for a Truss Element is given by:

$$K^e = \frac{A^e E^e}{L^e} \begin{bmatrix} l_{ij}^2 & l_{ij}m_{ij} & l_{ij}n_{ij} & -l_{ij}^2 & -l_{ij}m_{ij} & -l_{ij}n_{ij} \\ l_{ij}m_{ij} & m_{ij}^2 & m_{ij}n_{ij} & -l_{ij}m_{ij} & -m_{ij}^2 & -m_{ij}n_{ij} \\ l_{ij}n_{ij} & m_{ij}n_{ij} & n_{ij}^2 & -l_{ij}n_{ij} & -m_{ij}n_{ij} & -n_{ij}^2 \\ -l_{ij}^2 & -l_{ij}m_{ij} & -l_{ij}n_{ij} & l_{ij}^2 & l_{ij}m_{ij} & l_{ij}n_{ij} \\ -l_{ij}m_{ij} & -m_{ij}^2 & -m_{ij}n_{ij} & l_{ij}m_{ij} & m_{ij}^2 & m_{ij}n_{ij} \\ -l_{ij}n_{ij} & -m_{ij}n_{ij} & -n_{ij}^2 & l_{ij}n_{ij} & m_{ij}n_{ij} & n_{ij}^2 \end{bmatrix}^e \tag{3.4}$$

The displacements can then be found by solving the algebraic set of equations defined by the equation below, after applying the boundary conditions.

$$[K]_{modified}\{Q\}_{modified} = \{F\}_{modified} \tag{3.5}$$

The Global Stiffness Matrix, Global Force Vector and the Global Displacement Vector for the structure chosen here, are given in the I/O of the code.

Chapter 4

Force Analysis

The forces acting on the rocket body can be divided into two components: Axial Force and the Normal Force. The Axial Forces are drag, thrust and inertial forces. The equation below describes how the axial force can be calculated at an arbitary point along the length of the rocket. The Thrust force is taken to be negative because it acts in the downward direction.

$$F_{axial} = -F_{thrust} + D_{nosecone} + D_{fuselage} + D_{fin} + a_{axial} \sum_{base}^{x} m \tag{4.1}$$

For finding the Normal Force acting on the rocket body, it is assumed to a beam in equilibrium. This assumption allows the normal forces to be treated as an applied load on a simply supported beam. The equation seen below can be used to sum the normal forces acting at crucial stations along the rocket. Notice that the normal forces generated by the nose cone and fins are taken to be a lift force L. Also, the equation takes into account inertial forces generated by a lateral acceleration. Lateral acceleration is simply an acceleration perpendicular to the rocket body. A crosswind most commonly causes this acceleration. Finally the last term in the equation takes into account roll characteristics of the rocket.

$$F_{normal} = L_{nose} + L_{fin} - a_{lateral} \sum_{base}^{x} m - R \sum_{base}^{x} m(x_{CG} - x_m) \tag{4.2}$$

The following table lists out an example of the forces at various crucial points in a model rocket:

Position	Normal Force (lbs)	Axial Force (lbs)
Payload location	-10.62	30.35
Thrust Bulkhead	-12.92	-318.38
Middle of the Fins	45.40	-201.94
Rocket base	-17.54	-47.18

Chapter 5

Code

A computer routine written in C is generated for solving the Space Truss problem. The complete code has been attached in the appendix. Explanatory notes are written alongside various lines in the code to make it more readable. The Terminal window's screenshot (Figure 5.1) is given below, which shows the input and output to the program. Gauss-Seidel Iteration is used to find the solution to the algebraic equations that result at the end of the analysis. Gauss-Seidel iteration is a better and easier way to arrive at the solution then, say, Matrix Inverse.

This computer program can be used for both Spatial and Planar Trusses as it is without any modifications. For using the program to analyze Bar elements (which have single degree of freedom), Beam element or Frame element (which have rotational degrees of freedom), a slight modification is required. Since the element stiffness matrix differs of these cases, it needs to be calculated using different formula, which can be incorporated in the program by providing if statement and asking the user. The rest of the program can be used as it is, for only the value of the stiffness matrix changes.

```
//*****************************************************************************************************************************//

Hey, Tell me the no. of nodes and no. of elements.
3 3

Give the global node I and J of element 0.
1 3
Give the coordinates of element node i of element 0.
50 100 0
Now, give the coordinates of element node j of element 0.
100 0 0
Give the Young's modulus and Area for element 0
200000 1
Length of element 0 is 111.803398.


Give the global node I and J of element 1.
1 2
Give the coordinates of element node i of element 1.
50 100 0
Now, give the coordinates of element node j of element 1.
0 0 0
Give the Young's modulus and Area for element 1
200000 1
Length of element 1 is 111.803398.


Give the global node I and J of element 2.
2 3
Give the coordinates of element node i of element 2.
0 0 0
Now, give the coordinates of element node j of element 2.
100 0 0
Give the Young's modulus and Area for element 2
200000 1
Length of element 2 is 100.000000.


The element connectivity table is:
 Element        Global node i   Global node j
 0                  1                 3
 1                  1                 2
 2                  2                 3


Now, give the force vector components (x,y,z of node i & j resp.)
For element 0
0 514.3923 -47.240113 0 514.3923 -47.240113
For element 1
0 514.3923 -47.240113 0 514.3923 -47.240113
For element 2
0 514.3923 -47.240113 0 514.3923 -47.240113
357.770874

Global stiffness matrix is:
-47.240112    0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000
0.000000      715.541748    0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000
0.000000      0.000000      2862.166992   0.000000      0.000000      0.000000      0.000000      0.000000      0.000000
0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000
0.000000      0.000000      0.000000      0.000000      2357.770996   0.000000      0.000000      0.000000      0.000000
0.000000      0.000000      0.000000      0.000000      0.000000      1431.083496   0.000000      0.000000      0.000000
0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000
0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      2357.770996   0.000000
0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      0.000000      1431.083496
0.000000


The Global Force Vector is -
0.000000
0.000000
1028.784546
-94.480225
0.000000
1028.784546
-94.480225
0.000000
1028.784546


The nodal displacements are:
Q1: -0.000000
Q2: 0.000000
```

```
Q3: 0.359443
Q4: 0.000000
Q5: 0.000000
Q6: 0.718885
Q7: 0.000000
Q8: 0.000000
Q9: 0.718885

Process returned 9 (0x9)   execution time : 86.163 s
Press any key to continue.
```

//**//

Chapter 6

Conclusion

My project for the Advanced Mechanics of Materials Course involved writing a computer program for structural analysis of mechanical component. A Finite Element Analysis of a Space Truss, modelled from a cylindrical compartment in a modern-day rocket, was done and a C code was written for the execution and numerical calculation. The results of the program are presented in this report alongwith the procedure, geometrical and mathematical models.

Modern-day rockets are very complicated systems consisting of numerous integral components and sub-systems. Gauging the strength, against failure, of mechanical parts involved is very important for the success of rocket missions. With the advent of the new dawn to the space age, studies like this become more and more significant.

Appendix

```c
//Computer routine for Structural Analysis.

/* The layout of the program will be as follows -
    -> Ask the user the type of analysis req. Use the Banking program to have different functions for different types of analysis, for instance,
    simple linear, with multi-point constraint, temperature effects, torsion etc. But make the one for this analysis first and only extend it further.

    -> Ask the number of nodes, n. NO. of Elements in n-1. Then ask the coordinates of the nodes, and using this find the length of the elements. Name the
    elements appropriately. Make the element connectivity table. The global stiffness matrix will depend on this. For finding k, go element by element.

    -> Calculate the value of each element stiffness matrix.
    -> Then directly find the value of the Global Stiffness Matrix and Global Force Vector.

    -> Apply the boundary condition. Here, it is req. to know what type of boundary condition is given and using that the global matrices will be modified.
    Applying the boundary condition will render some of the displacements equal to nil. Don't change the other matrices, multiplication with zero will
    automatically make some rows and columns zero.

    -> Now using some method, preferably Gauss-Seidel Iteration, find the unknowns and give them as the answer.
    -> Have a structure for each element. The object type structure will have, element node i, element node j, length of element, E & A of the element.

    -> Have a separate function for getting the Glo+bal Force vector. This can be got directly by asking the user, atleast for the initial basic case. For
    the more complex case the element force vector has to be evaluated.

    -> The nodal displacements are solved for using Gauss-Siedel Iteration, for which 3 for loops are used. The first loop will iterate and the other two
    loops are required to transverse the 2D arrays. Just use the if condition j!=i to sum the product of K and Q, other than the req. one, and then just
    subtract this sum from F.
*/

#include<stdio.h>
#include<math.h>
struct element  //Structure type 1: element.
{
    int i[3]; //Coordinates for Element node i.
    int j[3]; //Coordinates for Element node j.
    int I; //Global node i.
    int J; //Global node j.
    float len;
    float E;
    float A;
    float stiff; //This is the common term multiplied at the start.
    float l;
    float m;
    float n; //The cosines.
};

struct element read (struct element e, int t)  //Function to read the inputs of each element from the user.
{
    printf ("\nGive the global node I and J of element %d.\n", t);
    scanf ("%d %d", &e.I, &e.J);

    printf ("Give the coordinates of element node i of element %d.\n", t);
    scanf ("%d %d %d", &e.i[0],&e.i[1],&e.i[2]); //Here, the ith x,y,z are for ith element.

    printf ("Now, give the coordinates of element node j of element %d.\n", t);
    scanf ("%d %d %d", &e.j[0], &e.j[1], &e.j[2]); //Thus, the x,y,z coordinates of element node j are stored.

    printf("Give the Young's modulus and Area for element %d\n", t);
    scanf("%f %f", &e.E, &e.A);

    //printf ("%d %d %d %d\n", e.i[0], e.i[1], e.i[2], e.J);

    float L=sqrt( pow(e.j[0]-e.i[0],2) + pow(e.j[1]-e.i[1],2) + pow(e.j[2]-e.i[2],2) ); //0 means x, 1 means y and 2 means z.
    e.len = L; //Length of the element.

    e.stiff = e.E*e.A/e.len; //Magnitude of the stiffness.

    e.l = (e.j[0]-e.i[0])/e.len; //cos x.
    e.m = (e.j[1]-e.i[1])/e.len; //cos y.
    e.n = (e.j[2]-e.i[2])/e.len; //cos z.

    printf ("Length of element %d is %f.\n\n", t, e.len);
    return (e);
}

int n,m; //No. of nodes and no. of elements.

int main ()
{
    int t; //The counting variable.
    printf ("Hey, Tell me the no. of nodes and no. of elements.\n");
    scanf ("%d %d", &n, &m); //No. of Nodes and no. of elements.

    struct element a[m];  //Creates m objects of class element.
    for (t=0;t<=m-1;++t)
    {
        a[t] = read(a[t],t); //Function takes the inputs from the user and stores it in each object. This is element t.
        //printf ("Element %d: %f %f %f\n", a[t].len, a[t].E, a[t].A); //Printing the structure.
    }

    //Getting the element connectivity table.
    printf ("\nThe element connectivity table is:\n Element \t Global node i \t Global node j\n");
    for (t=0;t<=m-1;++t)
    {
        printf (" %d \t\t %d \t\t %d \t\t %f\n", t, a[t].I, a[t].J, a[t].stiff);
    }

    struct matrix //structure type 2: 3nx3n matrix.
    {
        float ke[3*n][3*n]; //The element stiffness matrix.
        float f[3*n]; //The element force vector.
    };

    struct matrix b[m]; //m objects of class matrix.

    printf ("\nHey!!!\n");
    //Element stiffness matrix and force vector.
    int r,s;
    printf ("\n%f\n",a[0].stiff);
    for (t=0;t<=m-1;++t)
    {
        printf ("\n%f\n",a[t].stiff);
        // Stiffness Matrix.
        for (r=0;r<3*n;++r)
        {
            //printf ("%f\n\n",b[t].ke[0][0]);
            for (s=0;s<3*n;++s)
            { b[t].ke[r][s]=0;
            printf ("%f\t",b[t].ke[r][s]); } //Initialize all the values to zero.
            printf ("\n");
            b[t].f[r]=0; //similarly for the element force vector.
        }

        //printf ("\n%f %f *********\n",a[t].stiff,pow(a[t].l,2));
        b[t].ke[(3*a[t].I)-2][(3*a[t].I)-2]=a[t].stiff*pow(a[t].l,2); //Giving some positions values.
        b[t].ke[(3*a[t].I)-2][(3*a[t].I)-1]=a[t].stiff*a[t].l*a[t].m;
        b[t].ke[(3*a[t].I)-2][(3*a[t].I)]=a[t].stiff*a[t].l*a[t].n;
        b[t].ke[(3*a[t].I)-2][(3*a[t].J)-2]=a[t].stiff*pow(a[t].l,2)*(-1);
        b[t].ke[(3*a[t].I)-2][(3*a[t].J)-1]=a[t].stiff*a[t].l*a[t].m*(-1);
        b[t].ke[(3*a[t].I)-2][(3*a[t].J)]=a[t].stiff*a[t].l*a[t].n*(-1);

        b[t].ke[(3*a[t].I)-1][(3*a[t].I)-1]=a[t].stiff*pow(a[t].m,2);
        b[t].ke[(3*a[t].I)-1][(3*a[t].I)]=a[t].stiff*a[t].m*a[t].n;
        b[t].ke[(3*a[t].I)-1][(3*a[t].J)-2]=a[t].stiff*a[t].l*a[t].m*(-1);
        b[t].ke[(3*a[t].I)-1][(3*a[t].J)-1]=a[t].stiff*pow(a[t].m,2)*(-1);
        b[t].ke[(3*a[t].I)-1][(3*a[t].J)]=a[t].stiff*a[t].m*a[t].n*(-1);

        b[t].ke[(3*a[t].I)][(3*a[t].I)]=a[t].stiff*pow(a[t].n,2);
        b[t].ke[(3*a[t].I)][(3*a[t].J)-2]=a[t].stiff*a[t].l*a[t].n*(-1);
        b[t].ke[(3*a[t].I)][(3*a[t].J)-1]=a[t].stiff*a[t].m*a[t].n*(-1);
        b[t].ke[(3*a[t].I)][(3*a[t].J)]=a[t].stiff*pow(a[t].n,2)*(-1);

        b[t].ke[(3*a[t].J)-2][(3*a[t].J)-2]=a[t].stiff*pow(a[t].l,2);
        b[t].ke[(3*a[t].J)-2][(3*a[t].J)-1]=a[t].stiff*a[t].l*a[t].m;
        b[t].ke[(3*a[t].J)-2][(3*a[t].J)]=a[t].stiff*a[t].l*a[t].n;

        b[t].ke[(3*a[t].J)-1][(3*a[t].J)-1]=a[t].stiff*pow(a[t].m,2);
        b[t].ke[(3*a[t].J)-1][(3*a[t].J)]=a[t].stiff*a[t].m*a[t].n;

        b[t].ke[(3*a[t].J)][(3*a[t].J)]=a[t].stiff*pow(a[t].n,2);

        printf ("\n\n");
        for (r=0;r<3*n;++r)
        {
            for (s=0;s<3*n;++s)
            { b[t].ke[r][s]=b[t].ke[s][r];
              printf ("%f\t",b[t].ke[r][s]);
            } //using symmetry.
            printf ("\n");
        }
    }

    //Element Force Vector.
    printf ("\nNow, give the force vector components (x,y,z of node i & j resp.)\n");
    float fxi,fyi,fzi,fxj,fyj,fzj;
    for (t=0;t<=m-1;++t)
    {
        printf ("For element %d\n",t);
        scanf ("%f %f %f %f %f %f",&fxi, &fyi, &fzi, &fxj, &fyj, &fzj);

        b[t].f[(3*a[t].I)-2]=fxi;
        b[t].f[(3*a[t].I)-1]=fyi;
        b[t].f[(3*a[t].I)]=fzi;
        b[t].f[(3*a[t].J)-2]=fxj;
        b[t].f[(3*a[t].J)-1]=fyj;
        b[t].f[(3*a[t].J)]=fzj;
    }

    printf("%f\n\nGlobal stiffness matrix is:\n",b[1].ke[1][1]);
    //Global Stiffness Matrix.
    float K[3*n][3*n];
    for (r=0;r<3*n;++r)
    {
        for (s=0;s<3*n;++s)
        { K[r][s]=0; } //Initialize to zero.
    }
```

```c

    for (r=0;r<3*n;++r)
    {
        for (s=0;s<3*n;++s)
        {
            for (t=0;t<=m-1;++t) //There are m elements.
            { K[r][s]=K[r][s]+b[t].ke[r][s]; } //Sum the i,j element and store in Global Stiffness.
            printf ("%f\t",K[r][s]);
        }
        printf ("\n");
    }

    //Global Force Vector.
    float F[3*n];
    printf ("%f\n\n",F[0]);
    for (r=0;r<3*n;++r)
    { F[r]=0; } //Initialize to zero.

    printf ("\nThe Global Force Vector is - \n");
    for (r=0;r<3*n;++r)
    {
        for (t=0;t<=m-1;++t)
        { F[r]=F[r]+b[t].f[r]; } //Sum the r element of all matrices and save in r of F.
        printf ("%f\n",F[r]);
    }

    //Global Displacement vector.
    float Q[3*n];
    for (r=0;r<3*n;++r)
    { Q[r]=1; } //Initialize to one.


    //Gauss-Siedel Iteration.
    float sum;
    int N=5; //No. of iterations.
    int c;

    printf ("\n");
    for (c=0;c<N;++c)
    {
        printf ("\nIteration number %d: ",c);
        for (r=0;r<3*n;++r)
        {
            sum=0;
            for (s=0;s<3*n;++s)
            {   if (s!=r)
                { sum = sum + Q[s]*K[r][s]; }
            }

            if (K[r][r]!=0)
            { Q[r]=(F[r]-sum)/K[r][r]; }
            else
            { Q[r]=0; }
            printf ("%f\t",Q[r]);
        }
    }

    //Printing the output.
    printf ("\n\nThe nodal displacements are: \n");
    for (r=0;r<3*n;++r)
    {
        printf("Q%d: %f\n",r+1,Q[r]);
    }
}
//****************************************************************//
```