CHAPTER 4

Programming a truss finite element solver in C

In this section, some C functions are developed that allow to compute displacements and internal efforts of a two-dimensional truss structure.

4.1 Linear system solver

As it was explained in Chapter 3, computing the equilibrium of a truss structure requires to solve a linear system of equations (see (3.14)). Here, we asssume the existence of a linear solver Application Program Interface (API). In such an API, the following functions should be availabe:

- `matrix_t * create_matrix (int n, int m);` This function allows to create a $n \times m$ matrix. Sparse matrices are preferred in finite elements because of the low bandwidth of finite element stifness matrices. Matrix that is returned is initialized with zeros.
- `void delete_matrix (matrix_t *K);` This one removes the memory allocated by `create_matrix`.
- `void add_to_matrix (matrix_t *K, int n, int m, double val);` This one allows to add value `val` to row n and column m of matrix `K`.
- `void solve_linear_system (matrix_t *K, double *f, double *x);` This one solves the linear system $[\mathbf{K}](\mathbf{x}) = (\mathbf{F})$.

4.2 Input and output

A finite element solver of a truss takes as input the following datas:

- An array `double *xy` of size $2N$ that contains the coordinates of the N nodes (or joints) of the truss. We assume that coordinates x_1^j and x_2^j of node j are `xy[2*j]` and `xy[2*j+1]`, $j = 0, \ldots, N-1$.

- An array `int *ind` of size $2B$ that contains the topology of the truss i.e. the index of starting and ending nodes of each of the B rods (or bars) of the truss. We assume that bar j's starting node b_1^j is `ind[2*j]` and that bar j's ending node b_2^j is `ind[2*j+1]`, $j = 0, \ldots, B-1$.
- An array `double *f` of size $2N$ that contains the nodal forces applied to each of the nodes of the truss. The two components f_1^j and f_2^j of the force that acts on node j are respectively `f[2*j]` and `f[2*j+1]`.
- The way fixations (or supports) are defined is a little more tricky. Here, we propose not to be 100% general. We could indeed ask the user to provide matrix $[\mathbf{c}]$ and vector $(\bar{\mathbf{u}})$ of (3.12) but this is not very intuitive. The truss solver should build $[\mathbf{c}]$ and $(\bar{\mathbf{u}})$ by itself using straightforward informations. We propose to use 2 arrays. Assume that M constraints have to be applied to the truss and that each of the constraints is applied to one node of the truss (this is the non general part). The first array `int *nc` of size M contains all nodes that are associated with constraints. The second array `double *vc` of size $3M$ assigns one vector per constrained node. Vector with components `vc[3*j]` and `vc[3*j+1]` is the direction of the prescribed displacement at that node `nc[j]` and `vc[3*j+2]` is the value of that displacement.
- An array `double *ea` of size B that contains products $E^j A^j$ for each bar.

The output (the results) of the solver are the displacements at all nodes as well as reaction forces corresponding to the constraints:

- An array `double *x` of size $2N$ that contains the nodal displacements of each node of the truss. Displacement u_1^j and u_2^j are respectively `x[2*j]` and `x[2*j+1]`.
- An array `double *r` of size M that contains normal reaction forces.

4.3 Stiffness matrix

The function that computes the stiffness matrix of a bar is given in 4.1. This code is a simple translation of formula (3.15).

4.4 The solver

The function computes the equilibrium of a truss is given in Listings 4.2. The stiffness matrix of the truss is assembled between lines 8 and 16. The force vector is set on line 17. The constraints are set between lines 18 and 27. Note that we consider that the third component `v[2]` is the absolute value of the displacement that is prescribed along direction `v[0], v[1]`. We then put a unit vector in the matrix (lines 22 to 25). If the system cannot be inverted (zero determinant), it actually means that the truss is unstable (hypostatic). This can be due to external hypostaticity (not enough constraints) or internal hypostaticity (frame modes).

```
1  void stiffnessMatrix (double x1, double y1,
2                        double x2, double y2, double ea , double k[4][4])
3  {
4    double t = atan2(y2-y1,x2-x1);
5    double c = cos(t);
6    double s = sin(t);
7    double l = sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
8    double f = ea/l;
9    k[0][0] = k[2][2] = c*c*f;
10   k[1][1] = k[3][3] = s*s*f;
11   k[0][2] = k[2][0] = -c*c*f;
12   k[3][1] = k[1][3] = -s*s*f;
13   k[0][1] = k[1][0] = k[3][2] = k[2][3] = s*c*f;
14   k[3][0] = k[0][3] = k[2][1] = k[1][2] = -s*c*f;
15 }
```

Listing 4.1: Computation of the stiffness matrix of a bar

Let us now try our truss solver on the same simple truss as the one of §3.5. The code given in 4.3 shows how to define input data for this simple truss. Note that this way of defining data is clearly not the one that should be use to solve large trusses. Computer Aided Design systems like AutoCAD, ArchiCAD, ProEngineer, CATIA or SolidWorks allow to define complex structures in a WYSIWYG fashion. We should in principle be able to connect such a system and get relevant informations that can be automatically translated into our input format.

The computed displacement at node 2 is $u_1^2 = -4.7619e-05$ which is in accordance to what we found in §3.5 i.e.

$$-\frac{FL}{EA} = \frac{1000 \times 1}{210 \times 10^{-5}} = -4.7619 \times 10^{-5}.$$

Similarly, $u_2^2 = -0.000182306$ which is also correct. The value of the 4 lagrangle multipliers is

$$(\boldsymbol{\lambda}) = \{-1000, 0, 1000, -1000\}$$

which is obviously the right value of reactions forces at nodes 0 and 1.

4.5 Post processing

It is indeed possible to compute normal efforts n^j, $j = 0, \ldots, N-1$ in each bar of the truss in a post processing stage. For that, let us give the following interresting interpretation of the local stiffness matrix $[\mathbf{k}^j]$ of bar j. Let us isolate bar j and let's apply to bar j the displacements

$$(\mathbf{u}^j) = \{u_1^{b_1^j}, u_2^{b_1^j}, u_1^{b_2^j}, u_2^{b_2^j}\}.$$

```
int trussSolver (int N, int B, int M, double *xy, int *ind,
                 double *f, int *nc, double *vc, double *ea,
                 double *x, double *r) {
  double stiff [4][4];
  matrix_t *K = create_matrix (2*N+M, 2*N+M);
  double *F = (double*) malloc ((2*N+M)*sizeof(double));
  double *X = (double*) malloc ((2*N+M)*sizeof(double));
  for (int i=0;i<B;i++){
    int n1 = ind[2*i];
    int n2 = ind[2*i+1];
    int indx[4] = {2*n1,2*n1+1,2*n2,2*n2+1};
    stiffnessMatrix ( xy[indx[0]], xy[indx[1]],
                      xy[indx[2]], xy[indx[3]], ea[i] , stiff );
    for (int j=0;j<4;j++) for (int k=0;k<4;k++)
      add_to_matrix(K,indx[j],indx[k],stiff[j][k]);
  }
  for (int i=0;i<2*N;i++) F[i] = f[i];
  for (int i=0;i<M;i++) {
    int indx[2] = {2*nc[i], 2*nc[i] +1};
    double v[3] = {vc[3*i], vc[3*i+1], vc[3*i+2]};
    double norm = sqrt(v[0]*v[0] + v[1]*v[1]);
    add_to_matrix(K,2*N+i,indx[0], v[0]/norm);
    add_to_matrix(K,2*N+i,indx[1], v[1]/norm);
    add_to_matrix(K,indx[0],2*N+i, v[0]/norm);
    add_to_matrix(K,indx[1],2*N+i, v[1]/norm);
    F[2*N+i] = v[2];
  }
  bool result = solve_linear_system (K, F, X);
  if (result == false)printf("ERROR : the truss is not stable\n");
  for (int i=0;i<2*N;i++) x[i] = X[i];
  for (int i=0;i<M;i++) r[i] = X[2*N+i];
  delete_matrix(K);
  free(F);
  free(X);
  return result;
}
```

Listing 4.2: A truss solver

```
int main(void) {
  const double L = 1.0;
  const double F = 1000;
  const double A = 1e-4;
  const double E = 210e9;
  const int N = 3;
  double xy[2*N] = {0,0,0,L,L,0};
  const int B = 2;
  int ind[2*B] = {1,2,0,2};
  double ea[B] = {E*A,E*A};
  double f [2*N] = {0,0,0,0,0,-F};
  const int M = 4;
  int nc[M] = {0,0,1,1};
  double vc[3*M] = {1,0,0,0,1,0,1,0,0,0,1,0};
  double x[2*N];
  double r[M];
  int result = trussSolver (N,B,M, xy,ind,f,nc,vc,ea,x,r);
  return result;
}
```

Listing 4.3: Example of use of the truss solver

that were computed by the truss solver. The product

$$[\mathbf{f}^j] = [\mathbf{k}^j](\mathbf{u}^j)$$

allows to compute two force vectors

$$(\mathbf{f}^j) = \{\underbrace{f_1^{b_1^i}, f_2^{b_1^i}}_{\mathbf{f}^{b_1^i}}, \underbrace{f_1^{b_2^i}, f_2^{b_2^i}}_{\mathbf{f}^{b_2^i}}\}$$

at both ends b_1^i and b_2^i of the bar. Matrix $[\mathbf{k}^j]$ of (3.15) is of rank 2: its rows 1 and 3 are the opposite of each other. This is also true for rows 2 and 4. This means actually that $f_1^{b_1^i} = -f_1^{b_2^i}$ and $f_2^{b_1^i} = -f_2^{b_2^i}$. Then, we have $\mathbf{f}^{b_1^i} = -\mathbf{f}^{b_2^i}$. This makes sense: the bar is in equilibrium. Then it is easy to see that $\mathbf{f}^{b_1^i}$ is aligned with the axis of the bar. Consider a vector $\mathbf{n} = \{\sin\theta, -\cos\theta\}$ that is orthogonal to bar j. It is easy to see that $\mathbf{f}^{b_1^i} \cdot \mathbf{n} = 0$. This result again makes a lot of sense: the bar can only handle axial forces along X_1. The normal effort at both ends can then be computed as

$$n^j = \mathbf{f}^{b_1^i} \cdot \mathbf{t} = 0$$

with $\mathbf{t} = \{\cos\theta, \sin\theta\}$. We choose to use $\mathbf{f}^{b_1^i}$ in order that a positive n^j correspond to a bar in traction. The code given in 4.4 allows to compute normal efforts in every bar of the truss. In our simple example, we find

$$(\mathbf{n}) = \{1414.21, -1000\}$$

which is obviously the right answer.

```
1  void postPro (int B, double *xy, int *ind,
2                double *ea, double *x, double *n) {
3    double stiff [4][4];
4    for (int i=0;i<B;i++){
5      int n1 = ind[2*i];
6      int n2 = ind[2*i+1];
7      int indx[4] = {2*n1,2*n1+1,2*n2,2*n2+1};
8      stiffnessMatrix ( xy[indx[0]], xy[indx[1]],
9                        xy[indx[2]], xy[indx[3]], ea[i] , stiff );
10     double f [4] = {0,0,0,0};
11     for (int j=0;j<4;j++) {
12       for (int k=0;k<4;k++) {
13           f[j] += stiff[j][k] * x[indx[k]];
14       }
15     }
16     double t = atan2( xy[indx[1]]- xy[indx[3]], xy[indx[0]]-
           xy[indx[2]]);
17     n[i] = f[0] * cos(t) + f[1] * sin(t);
18   }
19 }
```

Listing 4.4: Computing normal efforts in all bars

4.6 Example

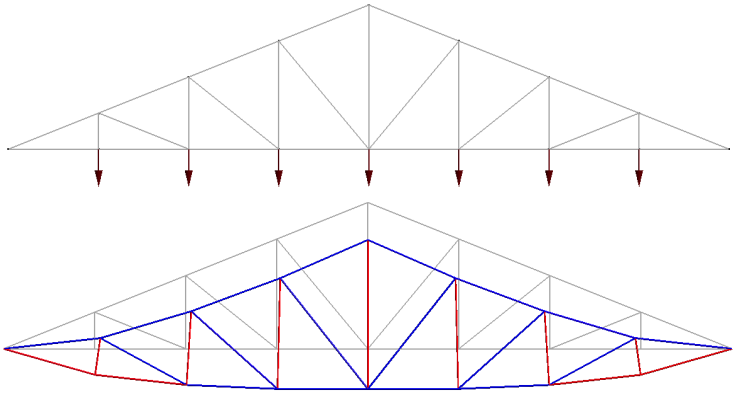

Figure 4.1: A Howe truss. The deformation of the truss is enhanced 100 times. The color correspond to traction (red) and compression (blue)

We consider here a truss with a well known topology that is called a Howe truss (see Figure 4.1. The Howe Truss was designed by William Howe in 1840. It used mostly wood in construction and was suitable for longer spans than the Pratt truss.

Therefore, it became very popular and was considered one of the best designs for railroad bridges back in the day.