6.3 Solving trusses on a computer

The method of joints is routine and is easily implemented on a computer one way or another.

- First, some software packages will accept a collection of algebraic equations, say the joint equilibrium equations, and solve them for the unknowns.
- Second, one can take the set of algebraic equations as written by hand, and organize them into matrix form and solve that form on a computer as described in Section 1.5 (see page 119).
- Finally, one can treat the whole truss problem as one for which you want to do all the algebra and solution on the computer.

The first two approaches are general purpose, using the linearity of the equations and nothing special about trusses. They are as useful for trusses as for any other situation in which you have several simultaneous equations to solve.

Here we take the third approach. We will set up and solve the equations for a truss using no hand-calculations whatsoever. We describe a method which you can program in whatever is your preferred computer package. The advantages of having a general-purpose computer program available include:

- you can quickly solve any truss
- you are less likely to make an error
- if you find an error in data entry, you can quickly correct it without having to redo all other data entry and calculation
- you can change the truss geometry easily to see the effect on the bar tensions and reactions
- you can just as easily solve non-simple trusses in which neither the method of joints nor the method of sections gives you equations that you can solve one at a time.

The act of understanding and writing such a program should also give you a better sense of the overall meaning of the method of joints.

The rest of this section is a description of the recipe, a presentation of the final program (on page 9), and some samples using that program. This program is just a systematic application of the method of joints.

The data that defines a truss problem

We first show how to define the truss, how it is supported, and the loads on it, with an organized collection of numbers rather than a picture. For definiteness, refer to the picture in fig. 6.41 which we want to communi-

cate to a computer.

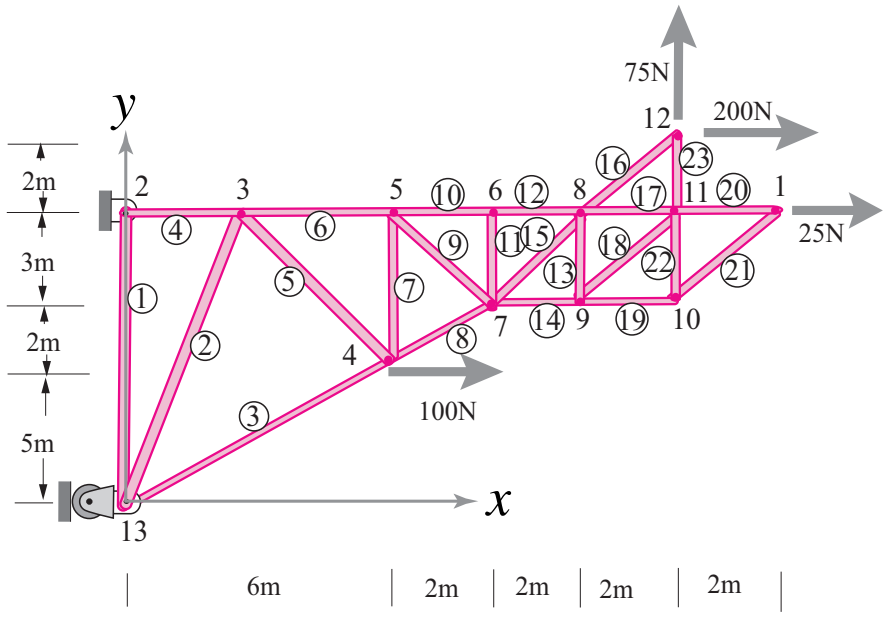

Figure 6.41: A truss with 23 bars (numbered with circles around them), 13 joints, 4 loads and 3 reaction components.

First pick an origin, coordinate directions, units to use for length and units to use for force. First a few numbers that say how many other numbers are needed.

n_{joints} is the number of joints, often called j. In the example $n_{joints} = 13$.

n_{bars} is the number of bars (rods), often called b. In the example $n_{bars} = 23$.

n_{bcs} is the number of reaction components (or boundary conditions), often called r. n_{bcs} is commonly 3: an x and y component at one joint and just an x or y component at another, as in the example.

The descriptions of the joints, the bars, the reactions and the loads are held in 4 matrices①.

$[J]$ is a matrix defining the joints. Each joint is identified by a number (1 or 2 or ...) with each number from 1 to n_{joints} associated with one joint. It doesn't matter which joint has which number. Each row of $[J]$ is the information for one joint. The first entry of a row is the joint number, and the next two numbers are the coordinates of the joint. If joint 6 is at $x = 8$ m, $y = 10$ m then the row 6 of $[J]$ would be [6 8 10]. $[J]$ has n_{joints} rows and 3 columns (fig. 6.42).

$[B]$ is a matrix defining the bars. It has one row for each bar (n_{bars} of them) and three columns. The bars are identified by numbers 1, 2,

① If you use and are comfortable with object-oriented programming some of the data structures below can be written in a more transparent form using suggestive naming rather than array locations for the various bits of data.

$$J = \begin{bmatrix} 1 & 14 & 10 \\ 2 & 0 & 10 \\ 3 & 2 & 10 \\ 4 & 6 & 5 \\ \vdots & \vdots & \vdots \\ 6 & 8 & 10 \\ \vdots & \vdots & \vdots \\ 13 & 0 & 0 \end{bmatrix}$$

joint # m

X & Y coordinates

Figure 6.42: The matrix $[J]$ gives the location of the joints.

$$B = \begin{bmatrix} 1 & 13 & 2 \\ 2 & 13 & 3 \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ 11 & 7 & 6 \\ \dots & \dots & \dots \\ 23 & 11 & 12 \end{bmatrix}$$

bar no.

base and tip joint no. S

Figure 6.43: The matrix $[B]$ gives the base and tip joint # of each bar.

...(sometimes circled, to distinguish them from the joint numbering). It doesn't matter which bar has which number so long as every integer from 1 to n_{bars} is associated with a bar (see fig. 6.43).

The first row of $[B]$ describes bar 1, the second describes bar 2, etc. The first element of each row is the bar number. This is also the number of the row, but it makes your data easier to read. The second two numbers are the numbers of the joints at the two ends of the bar. So if bar 11 connects base joint 7 with tip joint 6 the 11th row of $[B]$ is [11 7 6]. It is equivalent and ok to have instead the 11th row of $[B]$ be [11 6 7]; neither end of a bar is more special than the other. But once you have set the base and tip they are used to define angles in the calculations below.

$[R]$ is a matrix of reactions. It has as many rows as there are unknown reaction components, typically 3. $[R]$ has 4 columns. For easier reading, the first element of each row is the number of the row. The second element is the node at which the reaction applies. The next two numbers indicate the direction of the force acting on the truss (x and y components of a unit vector in the direction of the reaction):

- for a roller at a joint the last two numbers in the row are in the direction normal to the rollers. For normal support rollers they would be [0 1], for rollers against a vertical wall to the right of the structure they would be [−1 0]. For a roller on a 45° slope the two components could be [0.707 0.707]
- for a pin joint there are two rows in $[R]$: one for the x direction and one for the y.

Often $[R]$ will have exactly 3 rows. For the example matrix $[R]$ would be

$$[R] = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 2 & 2 & 0 & 1 \\ 3 & 13 & 1 & 0 \end{bmatrix}$$

$[F]$ is a matrix of applied loads. It has a row for each joint at which there is a non-zero load. It has three columns. The first entry of each row is the joint to which the load is applied. The next two numbers are the x and y components of the load applied to that joint. Any units can be used, they just have to be the same units for all loads. And the numerical answer for the tensions will be in these same units. If there is a rightwards load of 100 N at joint 4 one line of $[F]$ will read [4 100 0] (see fig. 6.44).

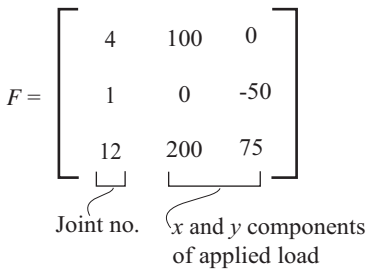

Figure 6.44: The matrix $[F]$ defines the applied loads, one row for each joint at which a load is applied.

All the information about a truss that we usually communicate with a sketch is in the 4 matrices $[J]$, $[B]$, $[R]$, and $[F]$.

These specify the locations of the joints, which joints the bars are connected to, the directions and locations of reaction forces and the applied

loads. Given these matrices and nothing else one could draw the truss, supports, and loading.

The unknowns

Solving the truss, finding the tensions in the bars and reaction components, is just a matter of manipulating the numbers in the four data matrices. We will hold that answer in the list $[T]$:

$[T]$ is a column vector holding the unknowns. It has as many elements as there are unknowns ($n_u \equiv n_{bars} + n_{bcs}$). The first n_{bars} elements are the unknown tensions, the last n_{bcs} elements are the unknown reaction components.

The problem

Our goal now is to use the data matrices $[J]$, $[B]$, $[R]$, and $[F]$ to find the unknowns $[T]$. We know it can be done by hand and, because the equations are linear, computer solutions should be straightforward.

Setting up the joint equations in matrix form

We now apply the method of joints.

For each joint we draw a free-body diagram (in our mind). And we apply force balance in the x and y directions. Thus we will have $2n_{joints}$ equations in terms of our $n_u \equiv n_{bars} + n_{bcs}$ unknowns. The strategy is to write all these equations long hand (in our mind) and then assemble those into matrix form.

If joint 1 has emanating from it bars 20 and 21 and also has a 25 N horizontal load to the right the first of these $2n_{joints}$ equations is (see fig. 6.45):

$$\cos\theta_{20}T_{20} + \cos\theta_{21}T_{21} + 25\,\text{N} = 0,$$

where θ_{20} and θ_{21} are the angles of bars 20& 21, measured CCW from the plus x direction. We can write this again as

$$0 * T_1 + 0 * T_2 + \cdots + A_{1,20} * T_{20} + A_{1,21} * T_{21} = -25\,\text{N}$$

where the cosines have been rewritten as elements of a matrix. If we assume that lots of these matrix elements are zero we can rewrite the first equation once again as

$$A_{1,1} * T_1 + A_{1,2} * T_2 + A_{1,3} * T_3 + \cdots + A_{1,n_u} * T_{n_u} = -F_{11}.$$

using $[A]$ as a matrix with lots of zeros, but sines and cosines of bar angles where appropriate. Recall that n_u is the number of unknown bar tensions and reaction components and $F_{11} = 25\,\text{N}$ is the x component of the load applied to joint 1.

For the second equation we similarly write the equation for force balance in the y direction for joint 1.

$$\sin\theta_{20}T_{20} + \sin\theta_{21}T_{21} = 0,$$

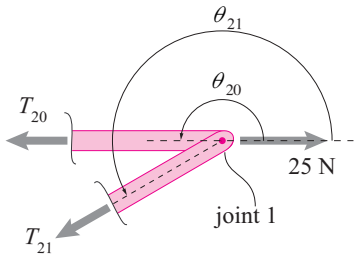

Figure 6.45: Free-body diagram of joint 1 on the truss of fig. 6.41 on page 2.

which can also be written out with the terms of $[A]$ (see fig. 6.46) as

$$A_{21} * T_1 + A_{22} * T_2 + A_{23} * T_3 + \cdots + A_{2n_u} * T_{n_u} = -F_{12}.$$

The next two equations describe joint 2, etc. Thus the assembly of $2n_{joints}$ equations looks like this

$$\begin{array}{cccc} A_{11} * T_1 & A_{12} * T_2 & \ldots & = & -F_{11} \\ A_{21} * T_1 & A_{22} * T_2 & \ldots & = & -F_{12} \\ \ldots & & & & \\ \ldots & & & & \\ A_{2n_{joints}\,2} T_1 & A_{2n_{joints}\,2} T_2 & \ldots & = & -F_{n_{joints}\,2} \end{array}$$

which we can write more compactly as

$$[A][T] = [L] \tag{6.19}$$

where $[A]$ is a matrix with cosines and sines of the bar angles and lots of zeros (because most bars don't touch a given joint) and $[L]$ is a list of negative of the loads applied in the x and y directions at the joints.

The point is, that all the information needed to calculate all the terms in $[A]$ and $[L]$ is in our four truss-definition matrices $[J]$, $[B]$, $[R]$ and $[F]$. And eqn. (6.19) for the unknown $[T]$ is exactly of the type that computers are great at solving.

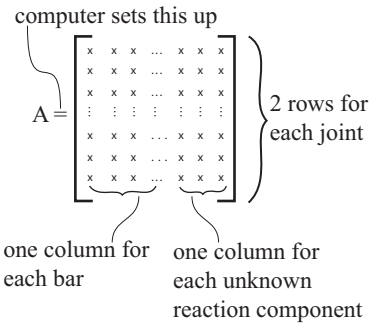

Figure 6.46: The matrix $[A]$ contains all the coefficients in the linear joint equations. It is made of sines and cosines of bar angles and the direction cosines of reaction-force directions.

Some preliminary geometry

The matrix $[A]$ is made up of sines and cosines of bar angles and we have specified the truss by the x and y positions of the ends of the bars. We first tell the computer to do some simple trig to find the sines and cosines.

$[X]$ is a list of x coordinates of each bar tip relative to its base. $[X]$ is a single column with n_{bars} entries. To find the entries of $[X]$ subtract the base-joint x coordinate from the tip-joint x coordinate. For bar 13 this would be

```
X(13) = J( B(13,3), 2 ) - J( B(13,2), 2 )
```

because `B(13,3)` is the joint at the tip of bar 13 and `B(13,2)` is the joint at the base. Thus `J(B(13,3),2)` and and `J(B(13,2),2)` are the x coordinates of the joints at the tip and base of bar 13. To find all of the elements of $[X]$ you may need to loop through all the bars or, depending on your package, you may be able to do the subtraction in one step.

$[Y]$ is a list of base-to-tip y coordinates for the bars defined analogously to $[X]$ above. Thus

```
Y(13) = J( B(13,3), 3 ) - J( B(13,2), 3 )
```

$[D]$ is a list of bar lengths (distances), so

```
D(13) =  ( X(13)^2 + y(13)^2 )^.5
```

$[C]$ is a list of n_{bars} cosines for the bars, one cosine for each bar. It is defined as the counter-clockwise angle of the base-to-tip bar relative to the positive x axis. Thus

```
C(13) =  X(13)/D(13)   % cosine
```

$[S]$ is a similar list of sines so

```
S(13) =  Y(13)/D(13)   % sine
```

All we need from the above are the $[C]$ and $[S]$ column vectors②.

②The calculation of $[X]$, $[Y]$ and $[D]$ are just intermediate steps to simplify the presentation. If you can tolerate dense coding and use a package that deals well with matrices, $[C]$ and $[S]$ can be generated with as few as 2 dense lines of code.

Building up $[A]$ from $[J]$, $[B]$ and $[R]$

The only difficult work in setting up a statically-determinate truss for computer solution is making up the matrix $[A]$. First let's set $[A]$ to be a matrix with $2n_{joints}$ rows and n_u columns and with every entry zero.

```
A = [0]
```

We now need to put a bunch of cosines and sines into the right places.

Cycle through the bars and put in cosines and sines of bar angles. If we look at the whole $[A]$ matrix we see that the information about bar 7, say, only occurs in column 7 of $[A]$; column 7 of $[A]$ consists of the terms that multiply T_7. Furthermore, information about bar 7 only shows up in the rows corresponding to the x and y force balance for the joints at its two ends; that's 4 places in total.

- Bar 7 pulls on its base joint `B(7, 2)` in the x direction. Because we write 2 equations for each joint this equation corresponds to row `2*B(7, 2)-1`. Thus we can make the assignment

  ```
  A(  (2*B(7, 2)-1), 7  )  =  C(7)
  ```

- Bar 7 pulls on its base joint in the y direction. This equation corresponds to the next row `2 * B(7, 2)` Thus we can make the assignment

  ```
  A( (2*B(7, 2) ), 7 )     =  S(7)
  ```

- Bar 7 pulls in the opposite direction on its tip joint `B(7, 3)` so

  ```
  A( 2*B(7, 3) -1,   7 )   =  -C(7)
  ```

- and

  ```
  A( 2*B(7, 3) , 7 )       =  -S(7)
  ```

One needs to cycle through all the bars③ and make these 4 assignments, 7 was just used as an example. In a package that deals well with matrices all four assignments associated with one bar could be in a single line of code.

③**Naive approaches**. One could imagine working one row at a time, corresponding to working one joint equation at a time rather than one bar at a time. For each joint we then would need to hunt through the list of bars and see which are connected to that joint. One could write a program to do this, it's just more complex than the approach we present. Alternately, you might imagine that in our original data set we would have associated each joint with the bars that connect to it (rather than the other way around as we did). This is also legitimate. But, because the number of connected bars varies from joint to joint the data structure would be more complex. Finally, because the key information is the location of the bar ends, we could have used those coordinates in our data array for the bars. But this would have required our entering the coordinates of each joint over and over, once for each bar-end connected to that joint.

Cycling through the reactions to fill in the right-most columns of $[A]$. The unknown reaction components have much the same role as do the bar tensions. But they act on only one joint. Thus each reaction component only affects 2 rows of $[A]$, the x and y components of that joint equation.

For reaction 3, say, the relevant joint is `R(3, 2)` and thus the relevant rows are `2*R(3, 2)-1` and `2*R(3, 2)` . The relevant column is $n_{bars}+3$.

- for the x component of reaction 3

  ```
  A( (2*R(3, 2)-1), (nbars+3) )  =  R(3,3)
  ```

- for the y component of reaction 3

  ```
  A((2*R(3, 2)   ), (nbars+3) )  =  R(3,4)
  ```

Most often, for trusses that are rigid even when floating, one only has three such reaction components to cycle through.

The load vector $[L]$ The load vector is just made up of the forces applied to the joints. For load 2, for example, applied at joint `F(2,1)` , the two relevant rows of $[L]$ are `2*F(2,1)` and `2*F(2,1)-1` at which act the x, and y components of the force `F(6,2)` and `F(6,3)`, respectively. Thus, for load 6, we have

```
L( 2*F(2,1) -1 )  =   -F(2,2)
L( 2*F(2,1)    )  =   -F(2,3)
```

Recall that the minus sign follows from moving the applied load to the right side of the equation. This pair of commands needs to be applied to each line of the $[F]$ matrix.

Solution

We have now constructed all the unknowns in eqn. (6.19)

$$[A][T] = [L] \tag{6.20}$$

and can thus hand the problem to the computer for solution

```
Solve {A T = L} for T
```

The resulting column vector $[T]$ is a list of bar tensions and reaction components.

The complete truss program

The complete truss program, in pseudo-code that you need to convert to your preferred computer language/package, is shown in fig. 6.41 on page 9. Some of the loops can be 'vectorized' if your package supports such. The output $[T]$ is a column with the tensions followed by the reaction components.

What can go wrong?

Besides the various careless errors you will discover the first 10 or so times you try to run your code, there are possible deeper problems.

Because we are not trying to write general purpose super-robust software we assume the simple check for determinacy (number of unknowns = number of equations):

$$n_{rods} + n_{bcs} = 2n_{joints} \quad \text{or} \quad b + r = 2j$$

has been satisfied. Thus $[A]$ will be square. If the truss is determinate the computer will give you a nice solution. If the truss is not determinate, with $[A]$ square or not, the result of the computer calculation will depend on the software package, ranging from an error message (e.g., "`Matrix singular!`" or "`Divide by zero!`") to the computer's making its best guess at what you want (even though the equations may have no solution, or there may be many solutions to select from). Some computer packages don't tell you when they are guessing.

How the pros solve trusses

The algorithm here is one way to set up and solve statically-determinate mechanics problems on a computer. In detail, however, this recipe is simpler than that commonly used in the *finite-element* method. Finite-element programs can also solve statically-indeterminate problems. A statically-indeterminate truss has tensions which can't be found from statics alone, but which can be found if the bar stiffnesses are known. Finite-element programs don't assume the bars are rigid. Rather, they take account of the small deformations of the bars.

A simple finite-element program for statically-indeterminate trusses would not use the tensions in the bars as unknowns, but rather the displacements of the joints. Such a program would be a little longer than the one presented here, and also requires introduction of a 'stiffness matrix'④, a topic a shade too advanced to cover in detail here.

④**Stiffness matrix**. The stiffness matrix $[K]$ for a truss has $2n_{joints}$ rows and columns. It satisfies the equation $[\Delta] = [K][L]$ where $[L]$ is a list of x and y components of the loads applied to all the joints and $[\Delta]$ are the x and y displacements of the joints for those loads. The matrix $[K]$ can be *assembled* if the properties of all the bars are given.

```
%PSEUDO-CODE TO SOLVE ANY 2D STATICALLY DETERMINATE TRUSS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Assign values to the matrices which define the truss and loading
  J = [ 1 . . . ]   % specify the joint locations
  B = [ 1 . . . ]   % specify the joints that the bars connect
  R = [ 1 . .   ]   % specify which nodes connect to the ground and how
  F = [ . . . . ]   % specify which nodes have what applied loads
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Program TRUSS, input is (J,B,R,F)  output is (T)
% Set up
 A     =  a square matrix  of zeros with twice as many rows as J
 L     =     a column      of zeros with twice as many rows as J
 nbars =  the number of rows of B
% Fill in the columns of the matrix A associated with bar tensions
 Loop for every bar  (each row i of B)
   base = B(i,2)                  % joint at one end of a bar
   tip  = B(i,3)                  % joint at the other end
   X = J( tip, 2 ) - J( base, 2 )  % base to tip x shadow of bar
   Y = J( tip, 3 ) - J( base, 3 )  % base to tip y shadow of bar
   D = ( X^2 + y^2 )^.5           % length of bar
   C = X/D                        % cosine of bar angle
   S = Y/D                        % sine   of bar angle
   A( (2*base-1), i ) =  C        % x comp of pull direction on base
   A( (2*base  ), i ) =  S        % y comp of pull direction on base
   A( (2*tip -1), i ) = -C        % x comp of pull direction on tip
   A( (2*tip   ), i ) = -S        % y comp of pull direction on tip
 End Loop
% Fill in  rightmost columns of  A, associated with reaction forces
 Loop for every reaction component (each row j of R)
   joint = R(j,2)                           % joint at ground connection
   A( (2*joint-1),  (nbars+j) )  = R(j,3)    % x comp of reaction direction
   A( (2*joint  ),  (nbars+j) )  = R(j,4)    % y comp of reaction direction
 End Loop
 Loop for all joints with loads (each row k of F)
   joint  =  F(k,1)                 % joint at which load is applied
   L( 2*joint -1 )  = - F(k,2)      % x component of load
   L( 2*joint    )  = - F(k,3)      % y component of load
 End Loop


% Solve the truss (solve the  set of simultaneous joint-equilibrium equations)
 Solve  {AT = L} for T     % The whole calculation is done in this one line.
                           % T is a list of bar tensions
                           %   followed  by reaction components
End Program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 6.47: **Pseudo-code for solving any statically determinate truss.** This 'program' calculates the bar tensions and reactions in a statically determinate truss. The algorithm is described in detail starting on page 1. This program could be reduced to 10 lines of code in some common computer languages (with some loss of clarity).