Programming for solving plane rigid frame based on MATLAB

Xiaokun Chen[1,*]

[1]Yangzhou University, Department of Civil Engineering, 225009 Yangzhou, China

Abstract. Based on the idea of the matrix displacement method, this paper designs a program which can be used to solve the internal force of the continuous beam and rigid frame with MATLAB. It mainly demonstrates how to design a program to realize the matrix displacement method with MATLAB. In addition, some techniques are included in order to realize the correspondence between the manual calculation and the computer calculation, such as "Using lambda to locate", "Crossing out rows and columns" and visual design. Therefore, based on the structural mechanics, combined with the principle of matrix displacement method, this paper shows the whole process from inputting the information of the rigid frame to solving the internal force of the rigid frame to outputting the bending moment diagram using MATLAB as the programming tool.

Keywords. MATLAB; Matrix displacement method; Structural numerical simulation; Plane bar structure

1 Introduction

Since the publication of Mathematical Principles of Nature Philosophy, Newton's law of motion and law of universal gravitation have been systematically established and elaborated. After that, structural mechanics has gradually developed under the framework system of classical mechanics. In the development history of structural mechanics, to solve the statically indeterminate structure, the force method was established by Maxwell in 1864. Though with its easy rationale, the choice of basic system requires the engineers' extensive experience and human intervention. At the beginning of the 20th century, the displacement method [1-4], which is more advantageous than the force method, has been developed to solve high-order statically indeterminate structures. However, with the increase of basic unknowns, the manual method of the displacement method is still unbearable. In order to solve this problem, in 1930 Hardy Cross invented the progressive analysis dethronement–Moment Distribution Method to simplify the calculation, which brought great convenience to the engineering calculation at that time. Then, with the development of the computer, the progressive calculation was replaced by the matrix displacement method, which is more suitable for programming. The matrix displacement method greatly enhances the efficiency of solving the high-order statically indeterminate structure and liberates the designer from the tedious calculation to a great extent. At the same time, in 1943, the mathematician Courant put forward the prototype of the finite element method. Subsequently, with the development of Aerospace Engineering, the finite element method gradually formed a complete system. Besides, because the matrix displacement method is the elementary form of the finite element method [5-8]. The matrix displacement method is also called the member finite element method.

Using MATLAB to program a matrix displacement method program can give full play to its advantages of matrix operation and can be applied to the force solution of a bridge or a building structure. Although there is complete finite element calculation software on the market, the internal finite element calculation steps are "black boxes" for users between the input and output. Therefore, for some personalized problems which are not in the scope of the general program, it still needs to be studied [9].

Therefore, based on the structural mechanics, combined with the principle of matrix displacement method [10], this paper shows the whole process from inputting the information of the rigid frame to solving the internal force of a rigid frame to outputting the bending moment diagram using MATLAB as the programming tool.

2 Program Design

2.1 Information input

First of all, some basic variables need to be involved in the operation of the program. In the input information function – "TypeInInformation" function, it contains the basic information.

* Corresponding author: cxksubmittingpaper@163.com

Among them, input the number of members that make up the rigid frame, and then input the information of each member in turn, including "*length*"–length of current member, "*EI*"–bending rigidity of current member, "*EA*"–tensile rigidity of current member, "alpha"–rotation angle of current member (define alpha in radian system; horizontal right as 0; clockwise as positive), "*loadtype*"–type of load of current member (load type "0"–No load; "1"–Uniform load (full load); "2"–Concentrated load (the load is considered to be applied on the 1/2 of the member); "*loadvalue*"–value of load (if uniform load–Unit: M/N; if concentrated load–Unit: N).

After inputting the basic information, the function calculates the "*EAlength*" (*EA/length*), "*Fp0*"($\overline{F_p}^e$) – equivalent node load vector. (Since one member has two nodes and each node has three force components (N, V, M), "*Fp0*" is a vector with six rows and one column). "*Fp0*" is calculated according to the "*loadtype*" and "*loadvalue*". "1" represents uniform load, with left node $\mathrm{N}=0,\mathrm{V}=-\frac{\mathrm{Fl}}{2},\mathrm{M}=-\frac{\mathrm{Fl}^2}{12}$; right node$\mathrm{N}=0,\mathrm{V}=-\frac{\mathrm{Fl}}{2},\mathrm{M}=\frac{\mathrm{Fl}^2}{12}$. "2" represents concentrated load, with left node $\mathrm{N}=0,\mathrm{V}=-\frac{\mathrm{F}}{2},\mathrm{M}=-\frac{\mathrm{Fl}}{8}$; right node $\mathrm{N}=0,\mathrm{V}=\frac{\mathrm{F}}{2},\mathrm{M}=\frac{\mathrm{Fl}}{8}$. In addition, introduce "*P0*"($\overline{P}^e$) – a new variable that inverse sign of "*Fp0*"($\overline{F_p}^e$) for clear expression purpose.

2.2 The formation of stiffness matrix *K*

2.2.1 Element stiffness matrix in local coordinate system $\overline{k}^e$

As mentioned above, since a member has two nodes and each node has three force components (N, V, M), "$\overline{P}^e$" is a vector with six rows and one column. These six force components correspond to six displacement components, that is, the displacement at the nodes of a member is also a vector of six rows and one column. According to the displacement method, it can be deduced that there is a relationship between displacement and force: $\overline{P}^e=\overline{k}^e\overline{\Delta}^e$. $\overline{k}^e$ is the element stiffness matrix in the local coordinate system. Obviously, it's a matrix of six rows and six columns.

While in the program, the cell matrix "k0=cell(n,1)" (corresponding $\overline{k}^e$, n rows, and 1 column, each row is a matrix, n is the member number), "P0=cell(n,1)" (corresponding$\overline{P}^e$) is defined to represent the element stiffness matrix and equivalent node load of each member in the local coordinate system. Similarly, all variables (including "*length*", "*EI*", "*EA*", "*alpha*", "*loadtype*", "*loadvalue*", "*EAlength*") that need to be inputed in the "TypeInInformation" function are defined as "zeros (n, 1)" (n rows and 1 column vector, i.e. n members, each member stores one information above).

After the definition is completed, loop the "TypeInInformation" function n times, which means, input the above basic information of n members.

According to structural mechanics, $\overline{k}^e$ is determined by the information above. So after the input is completed, loop n times, make each $\overline{k}^e$ *(k0)* =

$$\begin{matrix}
\frac{EI}{l} & 0 & 0 & -\frac{EI}{l} & 0 & 0 \\
0 & \frac{12EI}{l^2} & \frac{6EI}{l^2} & 0 & -\frac{12EI}{l^2} & \frac{6EI}{l^2} \\
0 & \frac{6EI}{l^2} & \frac{4EI}{l} & 0 & -\frac{6EI}{l^2} & \frac{2EI}{l} \\
-\frac{EI}{l} & 0 & 0 & \frac{EI}{l} & 0 & 0 \\
0 & -\frac{12EI}{l^2} & -\frac{6EI}{l^2} & 0 & \frac{12EI}{l^2} & -\frac{6EI}{l^2} \\
0 & \frac{6EI}{l^2} & \frac{2EI}{l} & 0 & -\frac{6EI}{l^2} & \frac{4EI}{l}
\end{matrix}$$

2.2.2 Element stiffness matrix in global coordinate system k^e

However, as a rigid frame, each member needs to be integrated together, which requires a global coordinate system, so these stiffness matrix need to be transformed into the global coordinate system.

In order to distinguish, the element stiffness matrix in the local coordinate system is represented by $\overline{k}^e$ (*k0* in the program) and the element stiffness matrix in the global coordinate system is represented by k^e(*ke* in the program). The rules applied to the definition of $\overline{P}^e$(*P0*) and P^e(*Pe*) are the same.

According to structural mechanics, element coordinate transformation matrix T (T is already known by mathematical derivation) is used to transform stiffness matrix and equivalent node load from local($\overline{k}^e/\overline{P}^e$) to global($k^e/P^e$), so that

$$k^e=T^T\overline{k}^eT \tag{1}$$

$$F^e=-P^e=T^T\overline{k}^eT\overline{\Delta}^e \tag{2}$$

the element stiffness matrix in global coordinate system is formed.

While in the program, the cell matrix "ke=cell (n, 1)"(corresponding to k^e, n rows and 1 column, each row is a matrix, n is the number of members), "P0 = cell (n, 1)" (corresponding to -F^e, n rows and 1 column, each row is a vector), "T = cell (n, 1)" (element coordinate transformation matrix of each member, T is related to the "*alpha*") are defined. The angle is defined as "alpha=zeros (n, 1)". After these preparation, the element stiffness matrix in the global coordinate system can be obtained according to the above formula.

2.2.3 Global stiffness matrix in global coordinate system K

In the traditional displacement method, the influence of each node displacement(Δ) on all node force(*F*) is considered respectively, and then the node force of each node are stacked, and then the relationship($F=K\Delta$) is formed. However, the "Element Integration Method"

is used to consider the individual contribution of one element (or one member) to "F" at one time, and then stack them. That is to say, each element(or member) only controls six nodal force at its two nodes, which corresponds, the element stiffness matrix in global coordinate system $k^e(6 \times 6$ matrix). And ultimately we need a global stiffness matrix $K(3(n+1) \times 3(n+1)$ matrix). So we need to put k^e into K each time (attention, we need to follow certain rules, that is, use "*lambda*" to locate), and record it as K^e(also a $3(n+1) \times 3(n+1)$ matrix, and except for the place for k^e, which is a 6×6 matrix, the rest of $K^e$is 0). Then K=$\sum K^e$, and that is called the "element integration method" to form K.

Besides, the purpose of location vector "*lambda*" is that because the previous "loop n times" refers to "n" members, but there are a total of "n+1" nodes, that correspond, "n+1" equivalent node force vectors. Therefore, we need to use "*lambda*" to map the contribution of each member to K^e.

While in the program, the cell matrix "lambda=cell (n, 1)" is defined, which means it contains n cells, and in each cell is a 6×1 vector

```
lambda{n,1}(1,1)= 3*n-3+1;
lambda{n,1}(2,1)= 3*n-3+2;
lambda{n,1}(3,1)= 3*n-3+3;
lambda{n,1}(4,1)= 3*n-3+4;
lambda{n,1}(5,1)= 3*n-3+5;
lambda{n,1}(6,1)= 3*n-3+6;
```

Also, loop it n times, then the location vector "*lambda*" is obtained. It should be noted that the idea of "*lambda*" here can also be used for subsequent "Crossing out rows and columns". Then, define "K=zeros(3∗n+3,3×n+3)", "P=zeros(3*n+3,1)" (equivalent node load vector of all nodes). Then, use the above principle to obtain the global stiffness matrix K, and the implementation method is as follows

```
for j=1:n
  for m=1:6
    for o= 1:6
K(lambda{j,1}(m,1),lambda{j,1}(o,1))=K(lambda{j,1}(
m,1),lambda{j,1}(o,1))+ke{j,1}(m,o);
    end
P(lambda{j,1}(m,1),1)=P(lambda{j,1}(m,1),1)+Pe{j,1}(
m,1);
  end
end
```

As is shown above, during the formation of $\overline{k}^e$, k^e and K, $\overline{P}^e$, P^e and P are formed in the same method.

2.3 The formation of Δ

Based on the displacement method of structural mechanics, in order to solve statically indeterminate structures, the basic equations of displacement method are needed

$$K\Delta + F_P = 0 \tag{3}$$

then

$$K\Delta = -F_P = P \tag{4}$$

Here, K and P are the global stiffness matrix and the equivalent node load. Theoretically, we only need to solve Δ according to the following formula

$$\Delta = K^{-1}P \tag{5}$$

However, since some bearings may constrain one or more directions of displacement, the displacement of these constrained directions must be "0". While in the manual calculation, because it is simple and can be judged directly by experience that some displacements are "0", the displacements which are "0" will not be encoded (this can also be shown in Chapter 3: Example) in the first place, so that "a problem" can be avoided.

"A problem" is: in the computer program, which node displacement is "0" will not be judged by experience in advance. In fact, it does not conform to the characteristics of the program: mechanical, which is a major difference between the human brain and the computer. Then, all nodes should be encoded directly from beginning to end in programming, which brings a problem: the number of equations ($K\Delta = P$) is likely to be greater than the number of unknowns (Δ), then the equation has no solution. So here we need to add an additional program process: "Crossing out rows and columns".

2.3.1 A solution to too many equations –"Crossing out rows and columns"

The rule of "Crossing out rows and columns" is to delete the rows and columns in K matrix when this row or column's displacement is "0", and delete the row in P vector when this row's displacement is "0" (P has only 1 column). In this way, "the number of equations" = "the number of unknowns".

To implement it in the program, define the constraint condition "ConstraintCondition= zeros(n+1,1)" (note that all of the following discussed are node conditions, n members have n + 1 nodes), and the counter "counter=zeros(n+1,1)" (the function of the counter is similar to "*lambda*", which records how many rows and columns are crossed out in the last loop, locates them in K and P, and cross them out).

After the definition, enter the loop "for j=1:n+1", loop n+1 times. Enter the "value" of constraint condition of each node, where "0" indicates no constraint; "1" indicates x - direction constraint; "2" indicates y - direction constraint; "3" indicates x&y direction constraint; "4" indicates three direction constraints. Take the case of "1 -> x - direction constraint" for example, execute in one loop

```
if j == 1              % if it is the first node
counter (j, 1) = 0;    % Because no row is
crossed out before, the counter remains 0.
end
if j > 1               % if it is not the first node
counter (j, 1) = counter (j-1, 1);  % The counter
maintains the last loop's value.
```

end
K(3*j-3+1-counter(j, 1),:) = []; % Cross out the corresponding rows in *K*. Here, "3*j-3" represents how many rows are in front of the j-th node, "+1" represents the first row of the j-th node, "-counter (j, 1)" represents to subtract the number of rows that have been crossed out according to previous counter records. In MATLAB syntax, "K(, :) = [] " means to delete the specified row. Therefore, the function of this line of code is to find out the row to be deleted (Step 1: due to the x–direction constraint, the displacement at "3*j-3+1" is "0", so the corresponding row and column in *K* need to be deleted. Step2: "-counter(j,1)" subtracts the number of previously deleted rows and columns to get the real ones to be deleted), and then delete them.
K (:, 3*j-3+1-counter(j, 1)) = []; % Same as deleting rows, this line of code is to locate and delete columns in *K*.
P (3*j-3+1-counter(j, 1),:) = []; % Similarly, this line of code is to locate and delete the corresponding rows in *P*.
if j==1
counter (j, 1) = 1; % The number of rows and columns that have been crossed out above is needed to record at the end of the loop, and because the case takes "1 -> x–direction constraint" as an example, the value of counter of the first node is 1.
end
if j >1
counter (j, 1) = counter (j-1, 1)+1; % If it is not the first node, then counter value is added by 1 based on the original value.
end
end

The above method is based on the case of "1 -> x–direction constraint". If the constraint is different, we only need to modify it slightly. For example, "2 -> y–direction constraint" only needs to change the line "K (3*j-3+1-counter(j,1),:) = [];" to "K(3*j-3+2-counter(j,1),:) = [];", which represents the corresponding row whose displacement in Y direction is also 0. The same rule is true for the column. For the "3 -> X&Y direction constraint", only another one row and one column need to be crossed out (that is, two rows and two columns in total), and the counter adds 2 more each time. It's also the same rule for "4 -> three direction constraints": three rows and three columns need to be crossed out, and the counter adds 3 more each time.

Judge all j nodes according to the above crossing out rows and columns rules successively. After execution, the remaining $K\Delta = P$ is not the previous $K\Delta = P$, but $K\Delta = P$ that some of the rows and columns are deleted. For the sake of distinction, write as $K_{incomplate}\Delta_{incomplete} = P_{incomplete}$.

2.3.2 Solve and Recover $\Delta_{incomplete}$

With the equation $K_{incomplate}\Delta_{incomplete} = P_{incomplete}$, the number of equations is equal to the number of unknowns, which can be solved mathematically. Utilize

$$\Delta_{incomplete} = K_{incomplete}^{-1} P_{incomplete} \quad (6)$$

Value of $\Delta_{incomplete}$ can be calculated directly. However, some rows with 0 items are deleted in $\Delta_{incomplete}$. To get the real node displacement Δ, we need to add the deleted rows with 0 items.

In order to do that, a loop is introduced, and loop j times (number of nodes), each time, judge the "ConstraintCondition" previously entered. In addition, introduce another counter "counter = 1" to serve as storing place, still take "1 -> x–direction constraint" as an example
if ConstraintCondition(j,1) == 1
Delta{j, 1}(1,1)=0; % Here, "*Delta*" represents Δ and "*Deltaincomplete*" represents $\Delta_{incomplete}$. This line of code means, if "1 -> x–direction constraint", the x direction displacement in Δ is supposed to be 0.
Delta{j,1} (2,1) =Deltaincomplete(counter, 1);
% The remaining value of "*Delta*" is directly taken from "*Deltaincomplete*".
Delta{j,1} (3,1) =Deltaincomplete(counter+1,1);
counter = counter + 2; % "+ 2" means that the next time "*Deltaincomplete*" is used, the previous value has already been taken out from *Delta*, we need to take value from the new point where new counter indicates, that is to ensure the code line "Delta{j,1} (2,1)=Deltaincomplete (counter,1)" can be successfully implemented.
end

Similar to the condition in chapter 1.3.1, if the "ConstraintCondition" is different, make some slight changes. For instance, if "ConstraintCondition(j, 1) == 3", just change to
Delta{j,1}(1,1) = 0;
Delta{j,1}(2,1) = 0;
Delta{j,1}(3,1) = Delta(counter,1);
counter = counter + 1;

2.4 The formation of F^e&$\overline{F}^e$

After getting the full Δ, comparing its form with P^e and k^e, we can find that the definition of Δ is in terms of "nodes", so its form is a $3 \times (n+1)$ vector. However, the previous P^e and k^e are considered and defined in terms of "members". So it requires some changes that help converted Δ to Δ^e (will be described in detail in chapter 1.4.1). Then, according to the formula

$$F^e = k^e \Delta^e - P^e \quad (7)$$

$$\overline{F}^e = TF^e \quad (8)$$

The force of each member in the global coordinate system F^e and the force of each member in the local coordinate system $\overline{F}^e$ can be calculated.

2.4.1 Convert Δ to Δ^e

In the program, first of all, the cell matrix of "Deltae=cell (n,1)" needs to be defined. Similar as P^e, it means a cell matrix of n rows (which corresponds to n members) and 1 columns. Each row is composed of a vector of 6 rows and 1 column (which corresponds to 6 displacements in total at 2 ends of each member). Additionally, we can find that the definition of Δ is in terms of "nodes", so its form is a $3 \times (n+1)$vector (representing(n+1)nodes, each contains 3 displacement components, 3 (n+1) in total). However, the Δ^e needs to be defined in terms of "members". To convert Δ to Δ^e, we need to use these lines of code

```
for j = 1:n
    Deltae{j,1}(1,1) = Delta{j,1}(1,1);
    Deltae{j,1}(2,1) = Delta{j,1}(2,1);
    Deltae{j,1}(3,1) = Delta{j,1}(3,1);
    Deltae{j,1}(4,1) = Delta{j+1,1}(1,1);
    Deltae{j,1}(5,1) = Delta{j+1,1}(2,1);
    Deltae{j,1}(6,1) = Delta{j+1,1}(3,1);
end
```

It is not hard to see that this is the process of taking out the value in *Delta(* Δ) and loading them into *Deltae(* Δ^e).

2.4.2 Solve F^e & $\overline{F}^e$

Because the preparations have been done well before, this part of the program is very simple. Just define the cell matrix "Fe=cell(n,1)" (F^e) and "F0=cell(n,1)" ($\overline{F}^e$). And according to the formula

$$F^e = k^e\Delta^e - P^e \tag{9}$$

$$\overline{F}^e = TF^e \tag{10}$$

set loop n times, each time execute:

```
Fe{j,1} = ke{j,1}*Deltae{j,1} - Pe{j,1}
F0{j,1} = T{j,1}*Fe{j,1}
```

F^e & $\overline{F}^e$ could be solved.

3 Visual Design

At this point, the theoretical calculation is finished when $\overline{F}^e$ *(F0)* has been solved and the purpose of matrix displacement method is achieved.

The bending moment is an important factor affecting the safety performance of the structure. So after getting the $\overline{F}^e$ data, the bending moment diagram can be drawn automatically with programming so that it can be shown more intuitive to the reader. In the program, the last function, "DrawM" function, is designed to draw the bending moment diagram according to the bending moment information from $\overline{F}^e$. It is divided into two steps: Positioning and Connecting. First, locate the control point of each member "LocationOfMember" (it involves the length, angle and other information of the member), then locate the control point of the bending moment diagram "LocationOfM" (it involves the location, bending moment value, load type and other information of the member). Specifically, the number of control points need to be judged by the load type. "0" (No load) is the direct connection between the two control points. "1" (Uniform load) is three control points are connected with parabola, and "2" (Concentrated load) is three control points are connected with a polyline. After the position of control points of the member and the position of control points of the bending moment are pointed, the bending moment diagram can be drawn by using the "plot" command to connect those points.

4 Example

According to structural mechanics, the steps of calculating the plane rigid frame with the matrix displacement method are as follows:

(1) Sort out the original data and code the members and frames locally and globally

(2) Forming element stiffness matrix $\overline{k}^e$ in local coordinate system

(3) Forming element stiffness matrix k^e in global coordinate system

(4) Forming integral stiffness matrix *K* with element integration method

(5) Calculating the element equivalent node load vector $\overline{P}^e$ in the local coordinate system, convert it to the element equivalent node load vector P^e in the global coordinate system, and form the global equivalent node load vector *P* with the element integration method

(6) Solving the equation $K\Delta = P$, and get the displacement vector of nodes Δ

(7) Solving the internal force vector of the nodes $\overline{F}^e$

Here is an example which solved from the perspective of manual calculation and computer calculation.

Try to solve the internal force of the frame as shown in figure1. It is assumed that each member is of rectangular section, beam $b_2 \times h_2$ *=0.5m* $\times$ *1.26m* , column $b_1 \times h_1$*=0.5m*$\times$ *1m*.

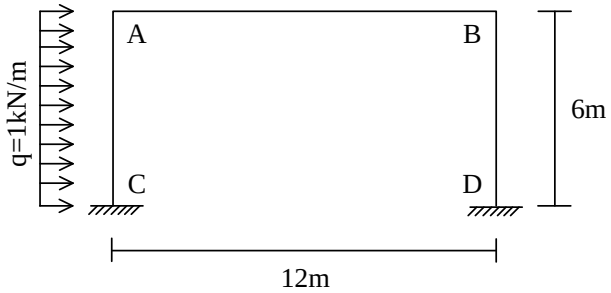

Fig. 1. Structure diagram

For the manual calculation, list the variables that need to be used: *A*, *I*, *EA*, *EI*, *L*, etc. Then, encode the rigid frame globally, as shown in figure 2. Members CA, AB, and BD are 1, 2 and 3 respectively. Note that since nodes C and D are fixed bearings, which displacement are "0", so they are not encoded in the first place in the manual calculation, which directly avoids the problem of "equation number > unknown number" when solving the

equation $K\Delta = P$. Therefore, node A is encoded as (1, 2, 3) and node B is encoded as (4, 5, 6).

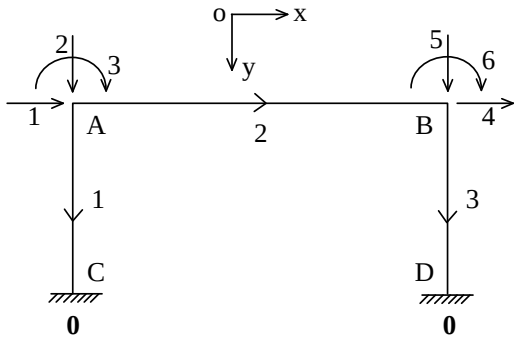

Fig. 2. Structure coding diagram

Follow the above steps in turn to solve $\overline{k}^{e}$, k^{e}, K.

Solve out $P = (3, 0, -3, 0, 0, 0)^{-1}$.

Solve the equation "$K\Delta = P$", $\Delta = (847, -5.13, 28.4, 827, 5.13, 96.5)^{-1}$.

Utilize $F^{e} = k^{e}\Delta^{e} - P^{e}$ & $\overline{F}^{e} = TF^{e}$, to solve out

$$\begin{cases} \overline{F}^{①} = (-0.43, -4.76, -8.49, 0.43, -1.24, -2.09)^{-1} \\ \overline{F}^{②} = (1.24, 0.43, 2.09, -1.24, -0.43, 3.04)^{-1} \\ \overline{F}^{③} = (0.43, -1.24, -3.04, -0.43, 1.24, -4.38)^{-1} \end{cases}$$

For the computer calculation, step (1), enter the input information function to input the basic information of members in the rigid frame, as shown in figure 3.

After inputting the basic information, enter the process previously described for processing. As is stated in chapter 1.3, it is impossible to judge whether there is a node displacement of 0 (or not to judge it conform the mechanical characteristic of computer calculation and is more convenient for a program) during the computer calculation, so the formed K is a 12×12 matrix (as is highlighted in figure 3), which needs to be crossed out some rows and columns. And step (5) is also completed simultaneously with step (2) (3) (4).

The resulting intermediate variables and final values are shown in the right part of figure 3, (the naming rules and the corresponding relationship can be checked in chapter 4). Among them

$$P = (3, 1.83697019872103e-16, -3, 0, 0, 0)^{-1}$$

$$\begin{cases} \overline{F}^{①} = (-0.427722975632854, -4.76361980720319, -8.48814137501562, \\ \quad 0.427722975632854, -1.23638019279681, -2.09357746820351)^{-1} \\ \overline{F}^{②} = (1.23638019279681, 0.427722975632854, 2.09357746820351, \\ \quad -1.23638019279681, -0.427722975632854, 3.03909823939074)^{-1} \\ \overline{F}^{③} = (0.427722975632854, -1.23638019279681, -3.03909823939075, \\ \quad -0.427722975632854, 1.23638019279681, -4.37918291739010)^{-1} \end{cases}$$

It can be seen that the error between the computer calculation result and the manual calculation result is within the allowable range, the program is accurate.

The moment diagram drawn by 'DrawM' is shown in figure4 below, which is consistent with the manually calculated moment diagram.

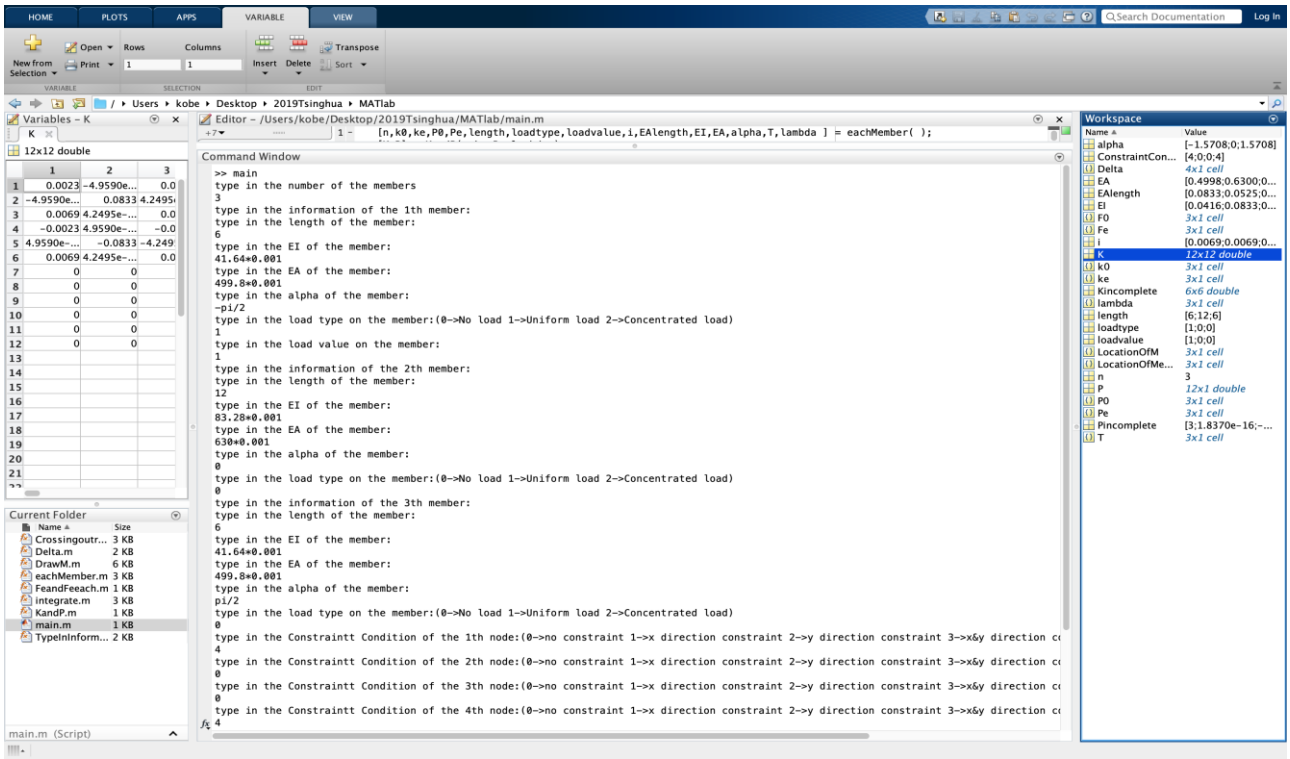

Fig. 3. The basic information of members and results

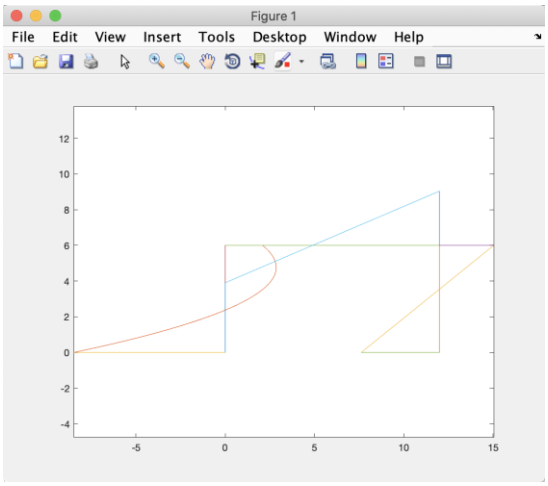

Fig. 4. Bending moment diagram

5 Conclusion

At this point, the introduction of the program is completed. This program is suitable for continuous beam and rigid frame. Its main function is to demonstrate how to correspond to the manual calculation of the matrix displacement method with the programmed computer calculation. In order to achieve that, in addition to the basic programming, some skills are included, such as "Using lambda to locate", "Crossing out rows and columns" and visual design.

In addition, there are some small details in the program that need to be paid attention to, such as symbol naming: variables with 0, such as "*P0*", represent variables without a "T" change in the local coordinate system. Variables with "e", such as "*Pe*", represent variables without a "T" change in the global coordinate system. Besides, variables used in the program contain a set of information, which is usually a cell matrix, representing the information set of all members(n) or all nodes(n+1). Accordingly, it is necessary to loop the variables n times (number of members) or n+1 times (number of nodes) to get the full information. Detailed variable name corresponding relationship is shown on table 1.

Table 1. Some variable naming rules

Variable's original name or description	Variable's symbols in Structural Mechanics	Variable's name in MATLAB Program
Element equivalent node load vector in the local coordinate system	$\overline{F_p}^e$	*Fp0*
Element stiffness matrix in local coordinate system	$\overline{k}^e$	*k0*
Element stiffness matrix in global coordinate system	k^e	*ke*
Global stiffness matrix in global coordinate system	K	K
Element equivalent node load vector in the local coordinate system	$\overline{P}^e$ $(=-\overline{F_p}^e)$	*P0*
Element equivalent node load vector in the global coordinate system	P^e	*Pe*
Global equivalent node load vector	P	P
Force of each member in the global coordinate system	F^e	*Fe*
Force of each member in the local coordinate system	$\overline{F}^e$	*F0*
Displacement defined in terms of "nodes"	Δ	*Delta*
Displacement defined in terms of "nodes"	Δ^e	*Deltae*
Displacement of deleted rows and deleted columns	$\Delta_{incomplete}$	*Deltaincomplete*

The purpose of the program is to consider the initial correspondence between the manual calculation and the computer calculation, to solve the internal force in the program. But further modification and expansion can be made based on that. First, the influence of bearing displacement and temperature change, elastic bearing and shear deformation can be considered to get more accurate results. Secondly, the program of deformation under load can be designed according to the formula "$\theta = -\frac{M}{EI}$" of material mechanics. Finally, although the program has a "DrawM" function to draw the moment diagram, it can be more precise and well developed based on the MATLAB GUI. The intelligent development of the visual graphical user interface and the automatic drawing function of the internal force diagram can further increase its universality and intelligence.

References

1. M. Bakhtiari, A. Tarkashvand, K. Daneshjou, Plane-strain wave propagation of an impulse-excited fluid-filled functionally graded cylinder containing an internally clamped shell[J]. Thin-Walled Structures, 2020, 149.
2. P.Sun, Y.Li, Z.Wang, K.Chen, B.Chen, X.Zeng, J.Zhao, Y.Yue, Inverse displacement analysis of a novel hybrid humanoid robotic arm[J]. Mechanism and Machine Theory, 2020, 147.
3. Y.Cai, T.Verdel, O.Deck, Using plane frame structural models to assess building damage at a large scale in a mining subsidence area[J]. European Journal of Environmental and Civil Engineering, 2020, 24(3).

4. X.Li, S.Wei, Q.Liao, Y.Zhang, A novel analytical method for four-bar path generation synthesis based on Fourier series[J]. Mechanism and Machine Theory, 2020, 144.
5. Y.Xia, X.Zhang, Z.Ye, M.Qiao, Research on Grid Side Power Factor of Unity Compensation Method for Matrix Converters[J]. JOURNAL OF POWER ELECTRONICS, 2019, 19(6).
6. L.Chen, J. Huang, M.X.Yi, Y.F.Xing, Physical interpretation of asymptotic expansion homogenization method for the thermomechanical problem[J]. Composite Structures, 2019, 227.
7. F.Pang, R.Huo, H.Li, C.Gao, X.Miao, Y.Ren, S.Georgantzinos, Wave-Based Method for Free Vibration Analysis of Orthotropic Cylindrical Shells with Arbitrary Boundary Conditions[J]. Mathematical Problems in Engineering, 2019, 2019.
8. S.Biswas, B.Mukhopadhyay, Three-dimensional vibration analysis in transversely isotropic cylinder with matrix Frobenius method[J]. Journal of Thermal Stresses, 2019, 42(10).
9. X.Ma, G.Yu, Application of MATLAB in structural mechanics [J]. Journal of Baicheng Normal University, 2006 (04): 99-102. (in Chinese)
10. Y.Long, S.Bao and S.Yuan, Structural Mechanics, Beijing: Higher Education Press, 2012: 303–333. (in Chinese)