Teaching Power Flow Calculation Using MATLAB

Ningqiang Jiang and Can Huang
Department of Electrical Engineering, Sch. Automation, Nanjing University of Science and Technology, Nanjing, China
Email: jiangningqiang@hotmail.com, huangcan12@foxmail.com

***Abstract*—Power flow calculation is a challenge for students in power system studying. Many of them are confused by the Jacobian matrix used in the Newton-Raphson method. The main difficulty lies in the analysis of the power flow model and the programming of the Jacobian matrix. In this paper, a thorough explanation on the construction of the Jacobian matrix is presented. Based on the matrix operation and symbol manipulation capacity of MATLAB, three approaches of implementation are discussed in detail. In the program, loop operation is eliminated in the iteration by matrix and symbol operation. Programs via these approaches are concise since no element by element construction of the Jacobian matrix is required. Therefore, the opportunity of successful programing is provided for more students. The program via approach 1 is the most efficient in sense of running time, while the load flow equations are more intuitively expressed via approach 2 or 3. The program via approach 3 is the most compact in length, but the details of the Jacobian matrix is concealed. It is preferable to resorting to approach 3, if the load flow result is expected only. And approach 1 is recommanded if we intend to observe the details of iteration, including the Jacobian matrix.**

***Index Terms*—power system analysis, power flow, Matlab**

I. Introduction

Power flow calculation is one of the main points in the course Power System Analysis and is the basis for the successive courses on dynamic analysis and relay protection. The task of power flow calculation is to determine the voltage distribution and the power transmission of the network. The Newton–Raphson method is the most common method introduced in the textbooks [1].

Despite that many simulation tools are equipped with the function of power flow calculation, it is mandatory for undergraduate students majored in Electrical Engineering to write a power flow calculation program independently. After the principle of the method is explained in the classroom, many students find it difficult to write the program. According to the teaching experience in the last ten years, less than one third of them can carry out fluently within two weeks. The main difficulty lies in the construction of the notorious Jacobian matrix, i.e., determination of the elements of the four block matrices contained in this matrix [2]. How to overcome this difficulty is a challenge both for the students and the teacher.

Mathematically, power flow calculation is to solve a set of nonlinear algebraic equations [3]. Therefore we can resort to mathematical softwares such as Maple, Mathematica, etc. MATLAB is a widely used scientific analysis tool. It features in powerful matirx and symbol manipulation capacity. In fact, it takes Maple as the calculation engine. In this paper, we will discuss how to make use of the potential of MATLAB in writing a power flow calculation program.

Three approaches will be discussed in detail. The first approach uses matrix operation to replace the traditional loop operation. In C or C++ environment, manipulation of an array or a matrix depends on the usage of pointers and unavoidable loop operation which incur many misusages. On the contrary, all data processed by MATLAB are considered as matrices. It is convienient to take a row or a column of a matix as an object in operation. Besides, it is also convienient to obtain the inverse of a matrix. The second approach uses the symbol toolbox to define the equations. That means the expression of an equation is considered as symbolic. Then the sentences in the program can have the same appearance as the equations. And the Jacobian matrix can be derived by MATLAB which resorts to MAPLE automatically. The third approach uses FSOLVE in the Symbolic Math Toolbox to solve the equations. By means of FSLOVE, only the symbolic description of the equations is required, while the solution of the equations is relegated to MATLAB, and the details of Jacobian matrix construction are concealed.

We hope that these three approaches will give some aids to the students in successfully writing a program. We suggest that the students use the third approach to get a solution rapidly, then one of the first two approaches is employed to look into the details of calculation, including the power mismatch, elements in the Jacobian matrix, etc.

II. Power Flow Model

A. Mathematical Model

There are three types of buses in a power network, i.e., PQ bus, PV bus and slack bus. For each bus, four variables are considered, i.e., the voltage V, the phase θ,

Manuscript received May 3, 2017; revised September 27, 2017.

the net injection real power *P* and the reactive power *Q*. And for each bus, only two variables are given, and the other two have to be determined by calculation.

Suppose the number of buses is n and the number of PQ bus is m, then the number of PV bus is n–m–1, since the number of the slack bus is 1 for any network. The total number of given *P*, *Q* variables is n +m–1 and the same number of V, θ variables are unknown.

Denote the set of PQ, PV and slack bus number as C_{PQ}, C_{PV} and C_S respectively. Power flow calculation is to solve (1) to determine the unknown *V*, *θ* variables first and then the unknown *P*, *Q* variables by (2) [4].

$$\begin{cases} 0 = f_i = \Delta P_i \\ \quad = P_i - V_i \sum_{j \in i} V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}), \quad i \in C_{PV} \cup C_{PQ} \\ 0 = f_{n-1+i} = \Delta Q_i \\ \quad = Q_i - V_i \sum_{j \in i} V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}), \quad i \in C_{PQ} \end{cases} \tag{1}$$

$$\begin{cases} P_i = V_i \sum_{j \in i} V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}), & i \in C_S \\ Q_i = V_i \sum_{j \in i} V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}), & i \in C_{PV} \cup C_S \end{cases} \tag{2}$$

where G_{ij}, B_{ij} are the real part and the imaginary part of the (i-j)th element of the conductance matrix *Y*, i.e.,

$$Y = G + jB = \begin{bmatrix} G_{11} & G_{12} & \cdots & G_{1n} \\ G21 & G_{22} & \cdots & G_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ G_{n1} & G_{n2} & \cdots & G_{nn} \end{bmatrix} + j \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1n} \\ B_{21} & B_{22} & \cdots & B_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ B_{n1} & B_{n2} & \cdots & B_{nn} \end{bmatrix}.$$

Denote $x=[\theta^T,V^T]^T$, $\theta=[\cdots;\theta_p,\cdots]^T$, $V=[\cdots;V_q,\cdots]^T$, $p \in C_{PV} \cup C_{PQ}$, $q \in C_{PQ}$ and $f=[\Delta P,\Delta Q]^T$, $\Delta P=[f_1,\cdots;f_{n-1}]^T$, $\Delta Q=[f_n,\cdots;f_{n+m-1}]^T$, the superscipt T is used to represent the transpose of the matrix. The vector form of (1) is

$$0 = f(x), \quad x \in R^{n+m+1} \tag{3}$$

B. Computational Model

The Newton-Raphson method solves equation (1) by iterations [5-11]. In the *k*-th iteration, the correction of unkown variables in *V* and *θ* are collected in a vector Δx^k,

$$\Delta x^k = [\Delta\theta^k, \Delta V^k / V^k]^T \tag{4}$$

It is approximated by

$$\Delta x^k \approx -\mathrm{J}^{-1}\big|_{x^k} f(x^k) \tag{5}$$

where J is the Jacobian matrix. And the variables are corrected by

$$\theta^{k+1} = \theta^k + \Delta\theta^k, \; V^{k+1} = V^k + \Delta V^k. \tag{6}$$

The Jacobian matrix J has four block matrices.

$$J_{(n+m\text{-}1)\times(n+m\text{-}1)} = \begin{bmatrix} \overline{H}_{(n\text{-}1)\times(n\text{-}1)} & \overline{N}_{(n\text{-}1)\times m} \\ \overline{K}_{m\times(n\text{-}1)} & \overline{L}_{m\times m} \end{bmatrix} \tag{7}$$

where $\overline{H} = \frac{\partial\Delta P}{\partial\theta}$, $\overline{N} = \frac{\partial\Delta P}{\partial V}$, $\overline{K} = \frac{\partial\Delta Q}{\partial\theta}$, $\overline{L} = \frac{\partial\Delta Q}{\partial V}$, and the elements in them are

$$\begin{cases} \begin{array}{ll} \overline{H}_{ii} = \frac{\partial\Delta P_i}{\partial\theta_i} & \overline{H}_{ij} = \frac{\partial\Delta P_i}{\partial\theta_j} \\ = V_i \sum_{j \in i, j \neq i} V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}), & = -V_i V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}) \\ \overline{N}_{ii} = V_i \frac{\partial\Delta P_i}{\partial V_i} & \overline{N}_{ij} = V_j \frac{\partial\Delta P_i}{\partial V_j} \\ = -V_i \sum_{j \in i, j \neq i} V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}), & = -V_i V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}) \\ \overline{K}_{ii} = \frac{\partial\Delta Q_i}{\partial\theta_i} & \overline{K}_{ij} = \frac{\partial\Delta Q_i}{\partial\theta_j} \\ = -V_i \sum_{j \in i, j \neq i} V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}), & = V_i V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}) \\ \overline{L}_{ii} = V_i \frac{\partial\Delta Q_i}{\partial V_i} & \overline{L}_{ij} = V_j \frac{\partial\Delta P_i}{\partial\theta_j} \\ = -V_i \sum_{j \in i, j \neq i} V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}), & = -V_i V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}) \end{array} \end{cases} \tag{8}$$

Many students are confused by the subscript of the elements in (8). $\overline{H}_{ii}$, $\overline{N}_{ii}$, $\overline{K}_{ii}$ and $\overline{L}_{ii}$ are often named as diagonal elements and other elements are named as non-diagonal elements. But the subscript does not definitely refer to as the location of the element in the block matrix. For example, suppose bus 1 is the slack bus, then *H*(1,1), the element in the first-row and first column of *H*, is $\partial\Delta P_2 / \partial\theta_2$, i.e., $H(1,1)=H_{22}$. Therefore we should carefully determine each element of the Jacobian matrix.

III. Approaches in Power Flow Calculation Programming

Since the expression for the diagonal and the non-diagonal elements of each block matrix are different, traditionally, we use loop operation to cope with each element separately. The program is tedious and often results in error. Now we present three efficient approaches in programming in the MATLAB environment [12-14].

A. Approach 1: Matrix Operation

We first observe the components of the elements in the Jacobian matrix and obtain the following results.

- H_{ii} is the total reactive power transmitted from bus *i* to other buses.
- H_{ij} is opposite to the reactive power transmitted from bus *i* to bus *j*.
- N_{ii} is opposite to the total real power transmitted from bus *i* to other buses.
- N_{ij} is opposite to the real power transmitted from bus *i* to bus *j*.
- K_{ii} is opposite to the total real power transmitted from bus *i* to other buses.
- K_{ij} is the real power transmitted from bus *i* to bus *j*.
- L_{ii} is opposite to the the total reactive power transmitted from bus *i* to other buses.
- L_{ij} is opposite to the reactive power transmitted from bus *i* to bus *j*.

If the power transmission via each branch is obtained, it can be used to construct the block matrices. Notice that components $V_i \times V_j$ and θ_{ij} appear in each element in (8), element-element product of matices, denoted by $\otimes$, can be used to form these elements.

The phase difference matrix $[\theta_{ij}]$ and the voltage product matrix $[V_iV_j]$ are obtained as following.

$$[\theta_{ij}] = \left\{[\theta_1,\theta_2,\cdots,\theta_n]^T \times [\overbrace{1,1,\cdots,1}^{n}]\right\} - \left\{[\theta_1,\theta_2,\cdots,\theta_n]^T \times [\overbrace{1,1,\cdots,1}^{n}]\right\}^T$$

$$= \begin{bmatrix} 0 & \theta_{12} & \cdots & \theta_{1n} \\ \theta_{21} & 0 & \cdots & \theta_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \theta_{n1} & \theta_{n2} & \cdots & 0 \end{bmatrix}$$

$$[V_iV_j] = \left\{[V_1,V_2,\cdots,V_n]^T \times [\overbrace{1,1,\cdots,1}^{n}]\right\} \otimes \left\{[V_1,V_2,\cdots,V_n]^T \times [\overbrace{1,1,\cdots,1}^{n}]\right\}^T$$

$$= \begin{bmatrix} V_1^2 & V_1V_2 & \cdots & V_1V_n \\ V_2V_1 & V_2^2 & \cdots & V_2V_n \\ \vdots & \vdots & \vdots & \vdots \\ V_nV_1 & V_nV_2 & \cdots & V_n^2 \end{bmatrix}$$

The branch power transmission matrix can be constructed by (9-1) and (9-2).

$$\begin{cases} S_p = [V_iV_j] \otimes \left\{\overline{G} \otimes \cos([\theta_{ij}]) + \overline{B} \otimes \sin([\theta_{ij}])\right\} \\ S_q = [V_iV_j] \otimes \left\{\overline{G} \otimes \sin([\theta_{ij}]) - \overline{B} \otimes \cos([\theta_{ij}])\right\} \end{cases} \quad (9)$$

Both the matrices S_p and S_q are n-dimensional squrare matrices. The non-diagnal element S_p (i,j) is the real power transmitted from bus i to bus j. The diagnal element S_p (i,i) is the real power consumed by the shunt impedance connected to bus i. Similarly, $S_q(i,j)$ is the reactive power transmitted from bus i to bus j, while $S_q(i,i)$ is the reactive power consumed by the shunt impedance connected to bus i.

Combined with the above mentioned observation results, the four block matrices $\overline{H}$, $\overline{N}$, $\overline{K}$ and $\overline{L}$ can be constructed with the elements in S_p and S_q. From the node table N, the vector SEL_p can be defined to collecte the bus number of PV and PQ buses. The net real power injection of these buses are given, while the same number of phases are unknown. We can also define the vector SEL_q to collect the bus number of PQ buses, for which the net reactive power injections are given, while the same number of voltage amplitudes are unknown. The block matrices can be constructed as following.

$$\begin{cases} H = \text{-}S_q\text{+diag(sum(}S_q^T\text{))} \\ N = \text{-}S_p\text{-diag(sum(}S_p^T\text{))} \\ K = \ S_p\text{-diag(sum(}S_p^T\text{))} \\ L = \ \text{-}S_q\text{-diag(sum(}S_q^T\text{))} \end{cases} \quad (10\text{-}1)$$

$$\begin{cases} \overline{H} = H(\text{SEL}_\text{p},\text{SEL}_\text{p}) \\ \overline{N} = N(\text{SEL}_\text{p},\text{SEL}_\text{q}) \\ \overline{K} = K(\text{SEL}_\text{q},\text{SEL}_\text{p}) \\ \overline{L} = L(\text{SEL}_\text{q},\text{SEL}_\text{q}) \end{cases} \quad (10\text{-}2)$$

Based on these analyses, the following program are presented. N and gen, the node table and the generator parameter table, are defined in Appendix A.

```
n=length(N(:,1));                                          %line1
selp=find(N(:,2)<3);                                       %line2
selq=find(N(:,2)==0);                                      %line3
v=N(:,3).*exp(N(:,4)*pi/180*1i);                           %line4
s=v.*conj(Y*v);                                            %line5
dp=(N(selp,7)-N(selp,5))/100-real(s(selp));                %line6
dq=(N(selq,8)-N(selq,6))/100-imag(s(selq));                %line7
while  max(abs([dp;dq]))>1e-10                             %line8
      thetaij=(N(:,4)  *ones(1,n)-ones(n,1)*N(:,4)')*pi/180; %line9
      ViVj=(N(:,3)*ones(1,n)).*(N(:,3)*ones(1,n))';        %line10
      sp=ViVj.*(real(Y).*cos(thetaij)+imag(Y).*sin(thetaij));%line11
      sq=ViVj.*(real(Y).*sin(thetaij)-imag(Y).*cos(thetaij)); %line12
      J=[-sq+diag(sum(sq')),-sp-diag(sum(sp'));            %line13
          sp-diag(sum(sp')),-sq-diag(sum(sq'))];           %line14
      J=J([selp;n+selq],[selp;n+selq]);                    %line15
      du=-J\[dp;dq];                                       %line16
      N(selp,4)=N(selp,4)+du(1:n-1)*180/pi;                %line17
      N(selq,3)=N(selq,3)+du(n:end)./N(selq,3);            %line18
      v=N(:,3).*exp(N(:,4)*pi/180*1i);                     %line19
      s=v.*conj(Y*v);                                      %line20
      dp=(N(selp,7)-N(selp,5))/100-real(s(selp));          %line21
      dq=(N(selq,8)-N(selq,6))/100-imag(s(selq));          %line22
end                                                        %line23
slack=find(N(:,2)==3);                                     %line24
N(slack,7)=real(s(slack))*100+N(slack,5);                  %line25
N(gen(:,4),8)=imag(s(gen(:,4)))*100+N(gen(:,4),6);         %line26
```

Comments on the sentences in the program are given below.

Line 1: number of buses.
Line 2: PV, PQ bus with given P and unknown θ.
Line 3: PQ bus with given Q and unknown V.
Line 4: voltage vector.
Line 5: complex power.
Line 6: real power mismatch defined in equation (1).
Line 7: reactive power mismatch defined in (1).
Line 8-23: determine unknown V and Theta in (1).
Line 9: construct the phase difference matrix.
Line 10: construct the voltage product matrix.
Line 11: real power transmitted via each branch.
Line 12: reactive power transmitted via each branch.
Lines 13-15:construct the Jacobian matrix by (9) and (10).
Line 16: correction.
Line 17: correct θ.
Line 18: correct V.
Line 19: voltage vector.
Line 20: complex power.
Line 21: real power mismatch.
Line 22: reactive power mismatch.
Line 24-26: determine the output of generators by (2).

In each iteration, lines 9-22 use matrix operation to construct the Jacobian matrix. The loop operation is eliminated, which makes the program concise and easy to debug.

B. Approach 2: Symbol Operation

Approach 1 is efficient in programming, but it does not keep the appearance of the mathematical model (1) in the program. Now we discuss an alternate approach, which will make the program more intuitive by using the symbol operation in MATLAB.

The symbol operation in MATLAB is carried out by Symbolic Math Toolbox (SMT) [9]. SMT calls Maple to perform symbol operation on a symbol expression and obtain a non-numerical result. Compared with numerical operations, no round-off error is introduced, therefore the accuracy of the result is guaranteed.

Symbol variable can be used to represent a number, a matrix or an expression, etc. For a pair of symbols *f* and *x*, standing for a function vector and a variable vector, respectively, the Jacobian matrix *J* can be obtained by using command *jacobian*(·), and the value of the Jacobian matrix can be obtained by command *eval*(·), i.e.,

$$J\text{=jacobian}(f,x),\;\; J\_eval\text{=eval}(J)$$

The task of power flow calculation is reduced to a symbolic description of (1). It is implemented by using *sym* to define the variables and the functions as symbols. The program is shown below.

```
n=length(N(:,1));                                         %line1
Yg=sym(real(Y),'d');                                      %line2
Ym=sym(imag(Y),'d');                                      %line3
U=[  ];  Theta=[  ];                                      %line4
for  i=1:n                                                %line5
      U=[U,  sym(['U'  num2str(i)])];                     %line6
      Theta=[Theta,sym(['Theta'  num2str(i)])];           %line7
end                                                       %line8
U(find(N(:,2)>0))=N(find(N(:,2)>0),3);                    %line9
Theta(find(N(:,2)==3))=N(find(N(:,2)==3),4)/180*pi;       %line10
Update='  ';                                              %line11
for  i=1:n                                                %line12
Update=strcat(Update,'U',num2str(i),'=u(',num2str(i),...  %line13
              ');Theta',num2str(i),'=theta(',num2str(i),');'); %line14
end                                                       %line15
selp=find(N(:,2)<3);                                      %line16
selq=find(N(:,2)==0);                                     %line17
S=[  ];                                                   %line18
for  i=1:n                                                %line19
  if  N(i,2)==0                                           %line20
    S=[S,N(i,7)/100-N(i,5)/100-U(i)*(Yg(i,:).*...         %line21
    cos(Theta(i)-Theta)+Ym(i,:).*sin(Theta(i)-Theta))*U.']; %line22
    S=[S,N(i,8)/100-N(i,6)/100-U(i)*(Yg(i,:).*...         %line23
    sin(Theta(i)-Theta)-Ym(i,:).*cos(Theta(i)-Theta))*U.']; %line24
  end                                                     %line25
  if  N(i,2)==2                                           %line26
    S=[S,N(i,7)/100-N(i,5)/100-U(i)*(Yg(i,:).*...         %line27
    cos(Theta(i)-Theta)+Ym(i,:).*sin(Theta(i)-Theta))*U.']; %line28
  end                                                     %line29
end                                                       %line30
JS=jacobian(S,[U(selq),Theta(selp)]);                     %line31
```

Comments on the sentences in the program:

Line1: number of buses.
Line 2: Define the real part of Y as symbols.
Line 3: Define the imaginary part of Y as symbols.
Line 4-8: Define voltage amplitude and phase as symbols.
Line 6: U=[U1,U2,…,Un].
Line 7: Theta=[Theta1,Thet2,…,Thetan].
Line 9: evaluate given voltages in U.
Line 10: evaluate given phases in Theta.
Lines 11-15: define a variable evaluation command
Line 16: buses with given *P*.
Line 17: buses with given *Q*.
Lines 18-30: define equation (1) as symbol *S*.
Lines 20-25: symbolic expression of (1) for PQ bus.
Lines 21-22: symbolic expression for function f_i.
Lines 23-24: symbolic expression for function f_{n-1+i}.
Lines 26-29: symbolic expression of (1) for PV bus.
Lines 27-28: symbolic expression for function f_{n-1+i}.
Line 31: define the symbolic expression of the Jacobian matrix.

Once the symbolic expression of the Jacobian matrix is defined, it can be repeatly evaluated, so that corrections can be obtained iteratively. In the following program, the Jacobian matrix and power mismatch are evaluated in lines 36-37 and the correction is obtained in line 38.

```
u=N(:,3).';                                               %line32
theta=N(:,4).'/180*pi;                                    %line33
eval(Update);                                             %line34
while  max(abs(eval(S)))>1e-10                            %line35
        JS_eval=eval(JS);                                 %line36
        S_eval=eval(S);                                   %line37
        dx=-inv(JS_eval)*S_eval';                         %line38
        u=u+dx(1:n)';                                     %line39
        theta=theta+dx(n+1:2*n)';                         %line40
        eval(Update);                                     %line41
end                                                       %line42
N(:,3:4)=[u',theta'*180/pi];                              %line43
v=N(:,3).*exp(N(:,4)*pi/180*1i);                          %line44
s=v.*conj(Y*v);                                           %line45
slack=find(N(:,2)==3);                                    %line46
N(slack,7)=real(s(slack))*100+N(slack,5);                 %line47
N(gen(:,4),8)=imag(s(gen(:,4)))*100+N(gen(:,4),6);        %line48
```

Comments of sentences in the program are the following.

Lines 32-33: initial value of u and theta.
Line 34: evaluate the symbols U and Theta.
Lines 35-42: iteration by Newton-Raphson method.
Line 36: evaluate the Jacobian matrix.
Line 37: evaluate the power mismatch.
Line 38: correction.
Line 39: correct u.
Line 40: correct theta.
Line 41: evaluate the variables.
Line 42: end of iteration.
Lines 43: renew *V* and θ in N.
Line 44: voltage vector.
Line 45: complex power.
Lines 46-48: determine the output of generators by (2).

Despite the longer program, the power flow model (1) is expressed intuitively in lines 21-24 and 27-28. And by using Jacobian(·), we are free from the details of constructing each element of the Jacobian matrix.

C. Approach 3: Using Algebraic Equation Solver

A solver for algebraic equations, fsolve(·), is also included in the Symbolic Math Toolbox. The definition of the equation and initial values of the variables are required by fsolve. The usage is depicted in the following program.

```
n=length(N(:,1));                                      %line1
selp=find(N(:,2)~=3);                                  %line2
selq=find(N(:,2)==0);                                  %line3
options=optimset('TolFun',1e-12,'Display','off');      %line4
x0=[N(selp,4)*pi/180;N(selq,3)];                       %line5
[x,fval]=fsolve(@(x)fu(x,N,Y,selp,selq),x0,options);   %line6
N(selq,3)=x(n:end);                                    %line7
N(selp,4)=x(1:n-1)*180/pi;                             %line8
u=N(:,3).*exp(N(:,4)/180*pi*1i);                       %line9
s=u.*conj(Y*u);                                        %line10
slack=find(N(:,2)==3);                                 %line11
N(slack,7)=real(s(slack))*100+N(slack,5);              %line12
N(gen(:,4),8)=imag(s(gen(:,4)))*100+N(gen(:,4),6);     %line13
```

In the program,
Line 1: number of buses.
Line 2: PV bus and PQ bus with given *P* and unknown *θ*.
Line 3: PQ bus with given *Q* and unknown *V*.
Line 4: tolerence and display options.
Line 5: initial values of *V* and *θ*.
Line 6: solve equation (1) defined in fu.m by FSOLVE.
Line 7: renew *V* in N.
Line 8: renew *θ* in N.
Line 9: voltage vector.
Line 10: complex power.
Line 11-13: determine the output of generators by (2).

The equation is defined in a separate file fu.m. The program is shown as following.

```
function  y=fu(x,N,Y,selp,selq)                            %line1
N(selq,3)=x(length(selp)+1:length([selp;selq]));            %line2
N(selp,4)=x(1:length(selp))*180/pi;                         %line3
u=N(:,3).*exp(N(:,4)/180*pi*1i);                            %line4
s=(N(:,7)+N(:,8)*1i)/100-(N(:,5)+N(:,6)*1i)/100-u.*conj(Y*u); %line5
y=[real(s(selp));imag(s(selq))];                            %line6
```

In fu.m,
Line 1: equation description, x=[theta,V].
Line 2: unkown voltage amplitudes.
Line 3: unknown phase.
Line 4: voltage vector.
Line 5: power mismatch.
Line 6: description of equation (1).

By using the solver, we do not have to construct the Jacobian matrix, no loop operation is required in the program and only the description of the power flow model is required in a separate file, i.e. line 6 in fu.m. The concealed iteration within the solver can be displayed if the options in line 4 is replaced by the following sentence.

options=optimset('TolFun',1e-12,'Display','iter'); %line4

Now we have present a thorough explanation on the construction of the Jacobian matrix when approach 1 is addressed. And based on the matrix operation and symbol manipulation capacity of MATLAB, three approaches of implementation are discussed in detail. They are hoped to relieve the burden in programming.

IV. Example

A. Example System and Load Flow Calculation Result

Take the WSCC three-machine-nine-bus system as example [15]. The system data are depicted in Appendix B. The Y matrix is obtained by the following sentences.

```
Y=diag(N(:,9)+N(:,10)*1i);                                 %line1
Br(find(Br(:,3)==0),7)=1;                                  %line2
for  ii=1:length(Br(:,1))                                  %line3
      a=Br(ii,1:2);                                        %line4
      k=Br(ii,7);                                          %line5
     Y(a,a)=Y(a,a)+[1,-1/k;-1/k,1/k/k]/(Br(ii,4)+...       %line6
                   Br(ii,5)*1i)+[1,0;0,1/k/k]*Br(ii,6)/2*1i; %line7
end                                                        %line8
```

In line 1, the shunt elements in N are put into Y. Line 2 makes transmission lines equivalent to transformers with tap ratio k=1. Lines 3-8 read the branch table Br described in Appendix A and determine the corresponding elements in Y.

Programs via the three approaches give the same result with the tolerance 1e-10, the renewed node table is:

```
 N =

 1   3   1.0400        0     0    0    71.6410    27.0459   0   0
 2   2   1.0250   9.2800     0    0   163.0000     6.6537   0   0
 3   2   1.0250   4.6648     0    0    85.0000   -10.8597   0   0
 4   0   1.0258  -2.2168     0    0          0          0   0   0
 5   0   0.9956  -3.9888   125   50          0          0   0   0
 6   0   1.0127  -3.6874     0    0          0          0   0   0
 7   0   1.0258   3.7197    90   30          0          0   0   0
 8   0   1.0159   0.7275   100   35          0          0   0   0
 9   0   1.0324   1.9667     0    0          0          0   0   0
```

B. Discussion on the Prgrams

Comparison on the programs is summarized in Table I.

TABLE I. Comparison of the Programs

Approach	1	2	3
Iteration programming required	Yes	Yes	No
Jacobian matrix programming required	Yes	No	No
Equ. (1) programming required	Yes	Yes	Yes
Equ. (2) programming required	Yes	Yes	Yes
Matrix operation used	Yes	Yes	Yes
SMT used	No	Yes	Yes
Program length in lines	26	48	19
Number of iterations	9	4	4
Running Time	<1e-6	0.750	0.015

The first main difference is the running time. The program via approach 1 is the fastest. The running time is less than 1e-6 second, almost imperceptible. Although iteration is required and each element of the Jacobian matrix should be explicitely determined by the program, matrix operation based on the analysis of the elements results in the most time-efficient program. The program via approach 2 is the most time-consuming due to frequent symbol processing resort to Maple. The time consumption of the program via approach 3 depends on the performance of FSOLVE.

The second main difference is the number of iterations. 9 iterations are required for the program via approach 1, which is more than the 4 iterations required by the programs via other approaches. The difference lies in the definition of the variables. In approach 1, the variable is defined by (4), while for the other approaches, it is defined by

$$\Delta x^k = [\Delta\theta^k, \Delta V^k]^T . \tag{11}$$

If we adopt this definition, the program via approach 1 can be modified by changing lines 15 and 18 into the following two sentences.

```
J=J([selp;n+selq],[selp;n+selq]) *diag([ones(size(selp));1./N(selq,3)]);
                                                              %line 15
N(selq,3)=N(selq,3)+du(n:end);                                %line 18
```

Then the number of iterations is reduced to 4 and the running time is also less than 1e-6 second.

The inner iterations of the program via approach 3 is listed below.

TABLE II. INNER ITERATIONS OF PROGRAM VIA APPROACH 3

Number of iter.	Func-count	f(x)	Norm of step	First-order optimality	Trust-region radius
0	15	8.4901	-	40	1
1	30	6.6197e-2	2.4639e-1	6.62	1
2	45	1.0617e-5	3.1280e-2	0.0733	1
3	60	4.1577e-13	4.6340e-4	1.52e-5	1
4	75	3.3140e-27	1.0347e-7	9.37e-13	1

V. CONCLUSION

An explanation on the construction of the Jacobian matrix is presented. And three approaches for load flow calculation using MATLAB are discussed. In the program, loop operation can be eliminated in the iteration by matrix and symbol operation. Programs via the three approaches are concise since no element by element construction of the Jacobian matrix is required. Therefore, the opportunity of successful programing is provided for more students. The program via approach 1 is the most efficient in sense of running time, while the load flow equations are more intuitively expressed via approach 2 or 3. The program via approach 3 is the most compact in length, but the details of the Jacobian matrix is concealed. It is preferable to resorting to approach 3, if the load flow result is expected only. And approach 1 is recommanded if we intend to observe the details of iteration, including the Jacobian matrix. The approaches can be employed in teaching practice and in following the developments of power flow calculation methods.

APPENDIX A DATA FORMAT

TABLE A1. NODE DATA (N)

Item No.	Description	Unit
1	Bus number	-
2	Node type(3:slack bus, 2:PV bus, 0: PQ bus)	-
3	Voltage amplitude	pu
4	Phase	degree
5	Real power load	MW
6	Reactive power load	MVA
7	Real power injection	MW
8	Reactive power injection	MVA
9	Shunt conductance	pu
10	Shunt susceptance	pu

TABLE A2. BRANCH DATA (BR)

Item No.	Description	Unit
1	From Bus number	-
2	To bus number	-
3	Branch type(0: transmission line, 1: transformer)	-
4	Resistence	Pu
5	Reactance	Pu
6	Capacitive suceptance	Pu
7	Tap ratio	-

TABLE A3. GENERATOR PARAMETER (GEN)

Item No.	Description	Unit
1	H*100	MWs/MVA
2	Power ratings	-
3	Xd'*10000	pu
4	Generator bus No.	-

APPENDIX B EXAMPLE SYSTEM DATA, POWER BASE=100MVA

```
N=[1 3 1.040 0    0   0 71.6 27.0   0  0;  2 2 1.025 0    0   0 163 6.7  0 0;
   3 2 1.025 0    0   0  85  -10.90  0 0;  4 0 1.000 0    0   0     0    0  0 0;
   5 0 1.000 0 125 50  0       0       0 0;  6 0 1.000 0   90 30     0    0  0 0;
   7 0 1.000 0    0   0  0       0       0 0;  8 0 1.000 0 100 35     0    0  0 0;
   9 0 1.000 0    0   0  0       0       0 0];
Br=[1 4 1 0.0000  0.0576  0.000  1;  2 7 1 0.0000  0.0625  0.000  1;
   3 9 1 0.0000  0.0586  0.000  1;  4 5 0 0.0100  0.0850  0.176  0;
   4 6 0 0.0170  0.0920  0.158  0;  5 7 0 0.0320  0.1610  0.306  0;
   6 9 0 0.0390  0.1700  0.358  0;  7 8 0 0.0085  0.0720  0.149  0;
   8 9 0 0.0119  0.1008  0.209  0];
gen=[ 2364 247.5  608 1 0; 640 192.0 1198 2 0; 301 128.0 1813 3  0];
```

ACKNOWLEDGMENT

This work was supported by National Science Foundation of China, grant no. 51577092.

REFERENCES

[1] J. Arrillaga, C. P. Arnold, and B. J. Harker, *Computer Modelling of Electrical Power Systems*, New York: John Willey & sons, 1983.

[2] A. R. Bergen and V. Vittal. *Power System Analysis*, 2nd ed., New York: Prentice Hall, 2000.
[3] P. Kundur, *Power System Stability and Control*, New York: Mc Graw-Hill, 1994.
[4] H. Saadat, *Power System Analysis*, New York: McGraw-Hill, 1999.
[5] W. F. Tinney and C. E. Hart, "Power flow solution by Newton's method," *IEEE Transactions on Power System and Apparatus*, vol. 86, pp. 1449-1460, 1967.
[6] V. H. Quintana and N. Muller, "Studies of load flow method in polar and rectangular coordinates," *Electric Power System Research*, vol. 20, pp. 225-235, 1991.
[7] V. M. da Costa, N. Martins, and J. L. Pereira, "Developments in the Newton Raphson power flow formulation based on current injections," *IEEE Transactions on Power Systems*, vol. 14, pp. 1320-1326, 1999.
[8] M. E. El-Hawary, *Introduction to Electrical Power Systems*, New York: John Wiley & Sons, 2008.
[9] J. F. Gutierrez, M. F. Bedrinana, and C. A. Castro, "Critical comparison of robust load flow methods for ill-conditioned systems," in *Proc. IEEE Int. Conf. on Power System Technology*, Trondheim, Norway, 2011, pp. 19-23.
[10] D. P. Kothari and I. J. Nagrath, *Modern Power System Analysis*, New York: McGraw Hill, 2008.
[11] O. L. Elgerd, *Electric Energy Systems Theory: An Introduction*, 2nd ed., Mc-Graw-Hill, 2012.
[12] *Simulink User's Guide*, The Mathworks, Natick, MA, 1999.
[13] *Power System Blockset User's Guide*, The Mathworks, Natick, MA, 1998.
[14] B. Aroop, B. Satyajit, and H. Sanjib, "Power flow analysis on IEEE 57 bus system using Mathlab," *International Journal of Engineering Research & Technology*, p. 3, 2014.
[15] P. M. Anderson and A. A. Fouad, *Power System Control and Stability*, Ames, IA: Iowa State University Press, 1977.

Ningqiang Jiang received the B.E. degree from Zhejiang University, Hangzhou, China, in 1992, and the Ph.D. degree from Southeast University, Nanjing China, in 2005.He is associate professor in the Department of Electrical Engineering, at the School of Automation, Nanjing University of Science and Technology, Nanjing, China. His research interest is in power system stability and control and the application of power electronics in power systems. Dr. Jiang is an IEEE member as well as an IEEE PES member.

Can Huang received the B.E. degree from Nanjing Normal University, Nanjing, China, in 2015. He is working toward a Master's degree in the Department of Electrical Engineering, at the School of Automation, Nanjing University of Science and Technology, Nanjing, China. His research direction is in power system modeling and stability analysis.