ALGORITHMIC CODE DIFFERENTIATION APPROACH TO SOLVE JACOBIAN ELEMENTS IN POWER FLOW EQUATIONS

A. PRASAD RAJU[1], J.AMARNATH[2], D.SUBBARAYUDU[2], M. RAMESH REDDY[1]
[1]Sree Chaitanya College of Engineering, Karimnagar-02, A P, India.
[2]J N T U H College of Engineering, Hyderabad-85, A P, India.
Email:Prasadraj55@gmail.com

Abstract: The Power flow problem is formulated in its basic analytical form and power, voltage constraints with the network makes the problem to be represented as system of simultaneous nonlinear equations and the numerical solution with Newton-Raphson method are of iterative in nature, includes computation of first order derivatives. Necessary features required for computation of these derivatives are simplicity, reusability, accuracy, errorless ness and high computational speed. Algorithmic or Automatic, differentiation is concerned with the accurate and efficient evaluation of derivatives for equations of Power flow problem defined by the functions in computer programs, and the resulting derivative values can be used efficiently and easily by the remaining program that contain code only for the computation of the numeric mismatch values of all equations. The power flow problem is solved with a Newton-Raphson method and necessary derivatives for the power flow equations are computed based on an Automatic Differentiation tool, namely ADMAT coded in MATLAB. In this paper Automatic Differentiation method is performed on a five-bus test system and results of this work explains the benefits of the Automatic Differentiation comparing to all other approaches to solve the Jacobian Elements in Power Flow Equations.

1 INTRODUCTION

Power flow analysis requires calculation of derivatives of expressions by applying many numerical techniques. In power flow study expressions for real and reactive power made by applying Newton-Raphson method and determined in every iteration. The power flow solution is a powerful tool to carry out different goals like the economical load dispatch, the optimized voltage profile or the preparation of the required reactive power compensation plan. Recent power flow solutions widely use a powerful solving method rests on a non-linear programming approach: Newton-Raphson method. This method copes with non-linear objectives and constraints and demands for accurate evaluations of first order derivatives of power system network equations. In general, sparse Jacobian and Hessian matrices have been calculated by hand calculations and finite-differentiation method because of runtime performance. With the help of Algorithmic Code Differentiation or Automatic Differentiation (AD) tools like ADMAT, the tedious and error-prone work of computation of first order derivatives becomes very simple. The load flow problem dealt here is to evaluate an power system network at state variables and to maintain control variables so that operating constraints and physical constraints are being satisfactory.

2 MODELLING TECHNIQUE OF POWER FLOW PROBLEM

Assuming a N-Bus power system network and power transfer between k_{th} bus and m_{th} bus of a given system, the power mismatch equations ΔP and ΔQ at k_{th} bus are the power balance equations and given as

$$\Delta P_k = P_{Gk} - P_{Lk} - P_{kCal} = 0 \quad (1)$$

$$\Delta Q_k = Q_{Gk} - Q_{Lk} - Q_{kCal} = 0 \quad (2)$$

Where P_k, Q_k are the mismatch active, reactive powers at bus k, respectively. P_{Gk} and Q_{Gk} refers the active, reactive power injections of the generator at bus k. It is assumed that these variables can be regulated by the plant operator. P_{Lk}, Q_{Lk} represent the active and reactive powers drawn at the load bus k, respectively. At steady state conditions the operator has control of these variables, and in power flow problem formulation these are treated as known variables [1, 2].The transmitted active, reactive powers, P_{kCal} and Q_{kCal} are based on nodal voltages, phase angles and line impedances and calculated by the power flow equations. The functions for P_{kCal}, Q_{kcal} can be represented as

$$P_{kCal} = V_k^2 G_{kk} + V_k V_m [G_{km} Cos(\theta_k - \theta_m) + B_{km} Sin(\theta_k - \theta_m)] \quad (3)$$

$$Q_{kCal} = -V_k^2 B_{kk} + V_k V_m [G_{km} Sin(\theta_k - \theta_m) - B_{km} Cos(\theta_k - \theta_m)] \quad (4)$$

In order to put on the Newton–Raphson method to deal with the power flow solution, the appropriate equations should be expressed in terms of Equation (1), where X denotes the set of

unknown nodal voltage magnitudes (v's) and phase angles (θ's) [1,2]. The power mismatch equations ΔP for real power, ΔQ for reactive power are evaluated around an initial point (θ(0), V (0)) and therefore the power flow Newton–Raphson approach is represented by the following Jacobian relationship:

$$\underbrace{\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix}}_{\mathbf{F}(\mathbf{X}^{(i-1)})} = \underbrace{\begin{bmatrix}\frac{\partial P}{\partial \theta} & \frac{\partial P}{\partial V}V\\ \frac{\partial Q}{\partial \theta} & \frac{\partial Q}{\partial V}V\end{bmatrix}}_{\mathbf{J}(\mathbf{X}^{(i-1)})} \underbrace{\begin{bmatrix}\Delta\theta\\ \frac{\Delta V}{V}\end{bmatrix}}_{\Delta\mathbf{X}^{(i)}} \tag{5}$$

The elements in the Jacobian consists of (nb-1) $\times$ (nb-1) size and each element given as

$$\frac{\partial P_k}{\partial \theta_m}, \quad \frac{\partial P_k}{\partial V_m}V_m \tag{6}$$

$$\frac{\partial Q_k}{\partial \theta_m}, \quad \frac{\partial Q_k}{\partial V_m}V_m \tag{7}$$

Where k =1…nb, m = 1…nb and 'nb' is number of buses but exclude the slack bus entries. Additionally, if buses 'k' and 'm' are not linked by transmission element, corresponding k–m entry in Jacobian is null. Having to the low degree of connectivity between all buses of the practical power systems, Jacobian matrices of power flows are of highly sparse. The efficiency of the Newton–Raphson method to acquire proper iterative solutions is depends on the choice of feasible initial values for the state variables like bus voltages, angles associated in the power flow problem and changes in the values of state variables ΔV, $\Delta\theta$ estimated for every iteration making use of equation (5) and the iterative solution represented as the function of correction vector $\Delta X^{(i)}$:

$$\Delta X^{(i)} = -J^{-1}(X^{(i-1)})F(X^{(i-1)}) \tag{8}$$

And the state variables are updated as

$$X^{(i)} = X^{(i-1)} + \Delta X^{(i)} \tag{9}$$

The computing is repeated with upgraded values of $X^{(i)}$ of equation (9), such that the value attained by the ΔX is within the tolerance. In this practice for each iteration, exact errorless computing of complex first order derivatives (Jacobian elements) is desired [3]. Ample amount of the computing work load is to be committed to obtain the first order derivatives of Jacobian elements. The original functions are usually described in a higher level programming languages like MATLAB, C, and C++. There are several ways to obtain the first order derivatives of a function with a computer program: like (i) Hand coding (ii) Finite-difference approximation (iii) Symbolic differentiation .Hand coding is complex for large functions, memory & time requirements are large and sparse is tedious task. Finite-difference approximation results in truncation and round-off errors causing failure of accuracy. Work ratio increases as number of variable in expression increases. Whereas Symbolic differentiation run into resource limitation and cannot handle CPU intensive processes when the dimension of the matrices is large .Owing to above explained disadvantages ,in this paper, a new technique known as Algorithmic Code Differentiation or Automatic Differentiation (AD) is propounded to compute the Jacobian elements.

3 TECHNIQUE OF ALGORITHMIC CODE DIFFERENTIATION

As explained in the above sections, the most familiar methods of computational differentiation have considerable disadvantages, made them infeasible for various complex applications. Desirably, a computational differentiation technique should (i) Compute derivatives automatically, exactly and fastly. (ii) Be able to handle arbitrary high-level codes, rather than expressions (iii) Compute exact derivatives (free of truncation errors) (iv) Compute derivatives at the cost independent of the number of variables. Automatic differentiation (AD) has these four properties; it computes derivatives of functions represented by means of a program written in a high-level language such as MATLAB, C, and C++. The AD approach depends on the principle of chain rule for calculating derivatives [5] as shown in equation (10):

$$\frac{\partial f(g(t))}{\partial t}\Big|_{t=t0} = \left(\frac{\partial f(s)}{\partial s}\Big|_{s=g(t0)}\right)\left(\frac{\partial g(t)}{\partial t}\Big|_{t=t0}\right) \tag{10}$$

It is employed in a mechanical fashion to compute derivatives of a complex functions. The AD software packages generate code for the derivatives rather than full symbolic expressions with respect to the independent variables.

Assume the function $y = f(x), \quad f: R^n \longrightarrow R$ represented by the following subroutine:

function (y,x)
for $i = n+1 : n+p$
    $x_i = f_i(x_j)_{j \in J_i}$
    end
    $y = x_{n+p}$
Where
$J_i \subset \{1,2,\ldots i-1\}, \forall i = n+1, n+2, \ldots n+p$

Figure 1: Representation of x_i in terms of already computed xj

Subroutine that shown in Figure.1 represents the function f(x) as a composition of the elementary or library functions $\{f_i\}_{i=n+1}^{n+p}$. Where f_i, is a function of already computed quantities x_j, $j \in J_i$.

3.1 Algorithmic Code Differentiation Applied to Power Flow Studies

To explain the method of Algorithmic Code Differentiation by applying to the power flow equations for computing the Jacobian elements or first order derivatives, here consider a partial portion of network with two number of buses(k and m) linked by a transmission line as shown in Figure.2

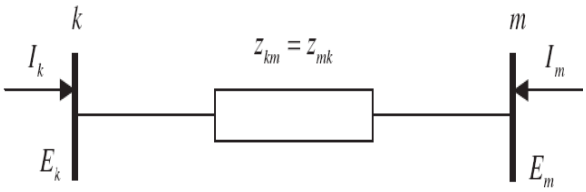

Figure 2: Representation of partial portion of power system network

Active power P_k is the function of nodal voltages V_m, V_k, power angles θ_m, θ_k and network impedance Z_{km}. P_k is computed using the equation given by equation (3).Rewriting it gives:

$$P_{kCal} = P_k^2 G_{kk} + V_k V_m [G_{km} Cos(\theta_k - \theta_m) + B_{km} Sin(\theta_k - \theta_m)] \quad (3)$$

Independent variables of equation (3) are given as:

$$[x_1 \quad x_2 \quad x_3 \quad x_4] = [V_m \quad V_k \quad \theta_m \quad \theta_k] \quad (11)$$

Substituting Eq. (11) in Eq. (3) gives:

$$y = P_k = x_1^2 G_{KK} + x_1 x_2 [G_{km} \cos(x_3 - x_4) + B_{km} \sin(x_3 - x_4)] \quad (12)$$

Also the gradient vector is given as:

$$\nabla P_k = \nabla y = \left[\frac{\partial y}{\partial x_1} \quad \frac{\partial y}{\partial x_2} \quad \frac{\partial y}{\partial x_3} \quad \frac{\partial y}{\partial x_4} \right]^T \quad (13)$$

To utilise the elementary rules of the calculus for differentiation, Eq. (12) can be decomposed into basic functions as shown in Figure 3. The variables { x_i }as i = 5 to 15 in Figure 3 are the intermediate variables in which the results of the elementary or library functions are stored. Fig. 4 shows the computational graph of the function shown in Eq. (12) with an acyclic direction. Each node in the graph indicates an intermediate or independent variable. And arrow runs from the node $\mathbf{x_j}$ to $\mathbf{x_i}$ representing dependency between variables. Direction xj to xi indicates that $\mathbf{x_i}$ depends on the already computed variable **xj**.

$$y = x_{16} = f_{16}(x_6, x_{15}) = x_6 + x_{15}$$
$$x_{15} = f_{15}(x_{13}, x_{14}) = x_{13} + x_{14}$$
$$x_{14} = f_{14}(x_7, x_{12}) = x_7 x_{12}$$
$$x_{13} = f_{13}(x_7, x_{11}) = x_7 x_{11}$$
$$x_{12} = f_{12}(x_{10}) = G_{km} x_{10}$$
$$x_{11} = f_{11}(x_9) = B_{km} x_9$$
$$x_{10} = f_{10}(x_8) = \cos(x_8)$$
$$x_9 = f_9(x_8) = \sin(x_8)$$
$$x_8 = f_8(x_3, x_4) = (x_3 - x_4)$$
$$x_7 = f_7(x_1, x_2) = x_1 x_2$$
$$x_6 = f_6(x_1) = x_1^2 G_{kk}$$
$$x_5 = f_5(x_1) = x_1^2$$

Figure 3: Decomposition of Eq. (12) and representing in terms of intermediate variables

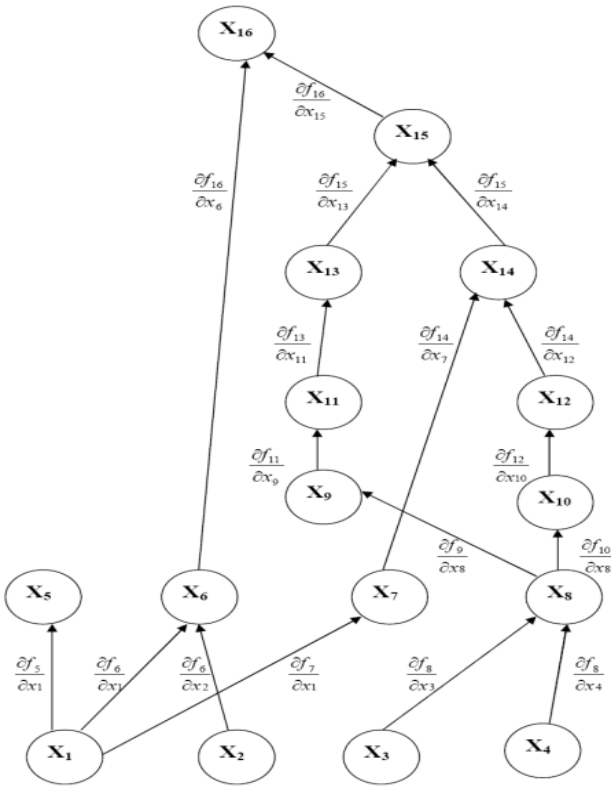

Figure 4: Directed acyclic computational graph of the function of Eq. (12)

Two approaches of Automatic Differentiation are developed namely Forward mode and Reverse mode. In this power flow problem Forward mode of Automatic Differentiation is implemented which is known as bottom to top approach, where the process starts from independent variables (x_1, x_2, x_3, and x_4) to dependent variables (x_5 to x_{15}) as shown in figure 4. Using $\mathbf{x_i}$ the gradient vector ∇x_i

is computed by the source transformation. Fig.5 shows the calculation of gradient vector of given function by using Fig.3 and Fig. 4 in the Forward mode approach. It is clear from the Fig. 5 that cost of computational work to calculate gradient vectors is directly proportional to the number of independent variables. But the effectiveness of the Automatic Differentiation is that, for a large system where numbers of independent variables are very large, in contrast to the above said, cost of computational work is decreases as number of independent variables increases. The relation between cost of computation and variables is given by:

$$q\{f,\nabla f\} = \sum_{i=n+1}^{p} q\{f_i,\nabla f_i\} + nn_i \qquad (14)$$

.

Where nn_i = (number of multiplications + number of additions). And 'q' is the cost of computation.

$$\begin{aligned}
\nabla x_5 &= 2x_1.\nabla x_1 \\
\nabla x_6 &= 2x_1.G_{kk}.\nabla x_1 \\
\nabla x_7 &= x_1.\nabla x_2 + x_2.\nabla x_1 \\
\nabla x_8 &= \nabla x_3 - \nabla x_4 \\
\nabla x_9 &= \cos(x_8).\nabla x_8 \\
\nabla x_{10} &= -\sin(x_8).\nabla x_8 \\
\nabla x_{11} &= B_{km}.\nabla x_9 \\
\nabla x_{12} &= G_{km}.\nabla_{10} \\
\nabla x_{13} &= x_7.\nabla x_{11} + x_{11}.\nabla x_7 \\
\nabla x_{14} &= x_7.\nabla x_{12} + x_{12}.\nabla x_7 \\
\nabla x_{15} &= \nabla x_{13} + \nabla x_{14} \\
\nabla y &= \nabla x_6 + \nabla x_{15}
\end{aligned}$$

Figure 5: Calculation of gradient vector for x_i of function shown by Eq.(12).

In case of differentiation of multiple functions of a system of $y = f(x)$ where $f : R^n \to R^m$,

$$\begin{bmatrix} \nabla y_1{}^T \\ . \\ . \\ \nabla y_m{}^T \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & . & . & \frac{\partial y_1}{\partial x_n} \\ . & & & . \\ . & & & . \\ \frac{\partial y_m}{\partial x_1} & . & . & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \begin{bmatrix} \nabla x_1{}^T \\ . \\ . \\ \nabla x_n{}^T \end{bmatrix} \qquad (15)$$

4 ALGORITHMIC DIFFERENTIATION BASED ON OBJECT ORIENTED PROGRAMMING

There are two ways of implementation of AD in the Forward Mode and Reverse Mode propagation of derivatives. Namely (i) Source Transformation Technique (ii) Object-Oriented Programming Operator Overloading Technique.

Source Transformation Technique uses a pre-processor (e.g. ADIFOR in FORTRAN) for the generation of derivative code. Source Transformation output is compliable and resulted code runs faster. Disadvantages of this technique are, it needs highly complex exceptional compiler type software to read in computer programs and justify the appropriate statements which requires the differentiation. In Source Transformation it is difficult to handle the reverse mode propagation in the presence of branches and data-dependent loops.

Object-Oriented Programming Operator Overloading Technique takes the benefit of ability to create new classes using powerful programming languages such as C, C++ and MATAB. Objects of the new A D class will stores the values and derivatives of a given expression [4]. Library functions and operators of the programming language are extended based on Operator Overloading. Given function that to be differentiated by the AD tool is coded by the program in the combination of these operators and from very fundamental derivative principles of calculus such as addition, subtraction, multiplication, division and chain rule. AD object of a derived class contains methods in which, equation or function to be differentiated will be coded, and the variables of the class object are known as the Active variables, these variables hold the value of the variable as well as the derivative information of the variable. Actual computational statements of the user provided code need not to be altered for the purpose of the Automatic Differentiation. All operators are overloaded and their operands are the active variables. An example function of AD class object is given below in Fig.6 as shown below.

$$\begin{aligned}
&function\ y = getfun(x) \\
&z = x * x; \\
&z = x + z; \\
&y = z * z; \\
&end
\end{aligned}$$

Figure 6: Function of AD class object containing active variables

4.1 Computation of Sparse Jacobian Using ADMAT: An A D Tool Developed Using Object Oriented Features of MATLAB

ADMAT is a powerful Algorithmic code Differentiation tool box designed in MATLAB to relief MATLAB users from computing first and second order derivatives precisely, expertly and automatically. To use ADMAT, the users of

MATLAB need to supply only an M-File to ADMAT, of equations or functions (which are to be require differentiation) which are coded using the MATLAB. Name of this M-File (string variable) and required input data passed as arguments to the methods of ADMAT classes those returns correct value and derivatives of that function.ADMAT contains rich set of functions derived from by overloading elementary operators and functions of MATLAB language. These functions handle the Jacobian, Hessian elements and also the sparsity.

Two functions of ADMAT plays crucial role in computing the Sparse Jacobian matrix [6]. Description of these functions as follows:

Function 'getjpi ( )' computes sparsity information for efficient computation of Jacobian matrix. It is invoked as:

[JPI, SPJ] = getjpi (fun, n, m, Extra, method, SPJ);

Function 'evalj ( )' computes the given functions values and its jacobian elements at point x (Where the vector X represents the state variable vector). It is invoked as:

[F, J] = evalj (fun, x, Extra, m, JPI, verb, stepsize);

5 CASE STUDY

Algorithmic Code Differentiation using ADMAT in the MATLAB environment is applied to a test system to solve the power flow solution with Newton-Raphson method. The test system shown in Fig. 7, has five buses, two generators, seven transmission lines and four loads with sufficient data.

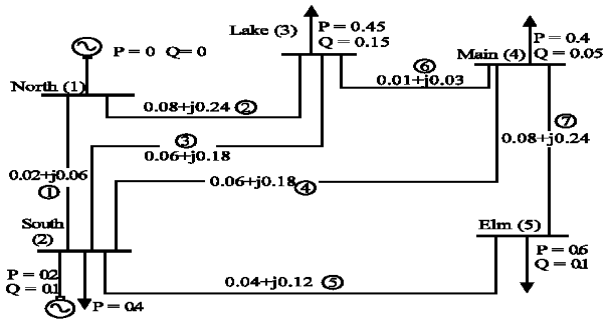

Figure 7: Five bus test power system with data

The given system data ,bus data ,generator data, branch data is coded in a main program written in MATLAB, to form Y_{Bus}, to calculate Power mismatch, to control Generator limits, sufficient number of functions are developed in the main program.

The main program relieves and calls ADMAT tool whenever required to compute Jacobian elements and the well designed functions of ADMAT will do this task and returns the accurate values of derivatives of Jacobian elements to the main program.

The procedure to be followed by the ADMAT to compute derivatives of a given function or equation is as shown:

1) Code the function or equation to be differentiated, in MATLAB and name this M-File with a string variable. Here in our power flow problem it is required to differentiate power flow equations (Eq.(3) and Eq.(4)) w.r.t the state variables (voltages and angles) to compute Jacobian elements, therefore code these equations in MATLAB.Let name it as 'powercal.m'

2) Set the problem size. Suppose for a five bus(n=5) system we set size as (n-1)×2 as first bus generally omitted as it is treated as slack bus:

›› x=9

3) Initialize the state variables vector **x.** For five bus system, four nodal voltage values and angle values, which are state variables, initialized in vector **x,** all five voltage values set to 1.0 and angles are set to 0.0 as shown:

```
›› i=1: m
›› x=ones (m, 1)
›› i=m+1: n
›› x=zeros (n,1)
```

4).Compute the sparsity pattern information of the function "powercal.m" by calling 'getjpi ()'function of the ADMAT as shown:

JPI = getjpi (powercal, n);

5).Compute the function values and Jacobian values of the given function (based on the sparsity information obtained by getjpi ( )) by calling "evalj ()"function of the ADMAT as follows:

[F, J] = evalj (powercal, x, [], n, JPI);

Where 'F' returns the function value and 'J' returns the Jacobian values.

As the main program calls the above two function of the ADMAT, they compute accurate values of the Jacobian elements and return these values to the main program for the further use in main program such as to compute changes in state variable vector to complete the power flow solution.

6 RESULTS AND DICUSSIONS

The five bus test system shown in Fig.9 is simulated to solve the power flow problem in the environment of MATLAB using both type of differentiation techniques, namely Finite Differentiation and Automatic Differentiation.

The results of both presented as shown below.

TABLE 1: Nodal Voltages and Angles computed using Finite Differentiation

NODAL VOLTAGE	BUS 1	BUS 2	BUS 3	BUS 4	BUS 5
MAGNITUDE(P.U)	1.06000000	1.00078543	0.98723258	0.98416512	0.97170595
PHASE ANGLE (DEG)	0.00000000	-2.06120059	-4.63679114	-4.95709914	-5.76492831

TABLE 2: Nodal Voltages and Angles with Algorithmic Differentiation

NODAL VOLTAGE	BUS 1	BUS 2	BUS 3	BUS 4	BUS 5
MAGNITUDE(P.U)	1.06000000	1.00076212	0.96211246	0.97112101	0.96020412
PHASE ANGLE (DEG)	0.00000000	-2.06943860	-4.68989556	-4.96890086	-5.78506945

TABLE 3: Error values (of Voltages and Angles) corrected by Algorithmic Differentiation, that due to Finite-Differentiation Approximation

NODAL VOLTAGE	BUS 1	BUS 2	BUS 3	BUS 4	BUS 5
MAGNITUDE(P.U)	0.00000000	0.00002331	0.02512012	0.01304411	0.01150183
PHASE ANGLE (DEG)	0.00000000	0.00823801	0.05310442	0.01180172	0.02014114

From the above results it is observed that computation of the Jacobians as well as the nodal voltages, phase angles, real powers and reactive powers of the load flow solution using the Finite Differentiation is tedious and error prone and the output values obtained are approximate values. Alternatively computing the Jacobians, nodal voltages, phase angles, real powers and reactive powers of the load flow solution using the Algorithmic Differentiation is very simple, efficient and gives the very accurate, errorless solution up to the precision of 1e-08.

Table 1 and Table 2 gives the computation of nodal voltages and phase angles of test system using Finite and Algorithmic Differentiation respectively. And Table 3 gives the error caused while computation of nodal voltages and phase angles by the Finite Differentiation approximation and that is corrected by the Algorithmic Differentiation.

7 CONCLUSIONS

In this paper it has been presented that computation of the first order derivatives (or Jacobian elements) of power flow problem using Automatic Differentiation technique is simpler, flexible and accurate over the traditional Finite Differentiation technique. It has been observed that computational cost and time is very less due to the decomposition of complex expressions into simple terms.

A model has been formulated for a standard five bus power system network then the technique of Automatic Differentiation implemented and tested using that system and the effectiveness of this technique is demonstrated.

8 ACKNOWLEDGMENTS

The authors are thankful to the managements of Sree Chaitanya College of Engineering and JNT University, Hyderabad, for providing facilities to publish this work.

9 REFERENCES

[1] P.Kundur, Power System Stability and Control, (McGraw-Hill, 1994).

[2] Stott, Review of Load-flow Calculation Methods, IEEE Proceedings vol 62 pp 916–929, July 1974.

[3] Tinney, W.F., Hart, Power Flow Solution by Newton's Method, IEEE Trans. Power Apparatus and Systems PAS-86(11) 1449–1460, 1967.

[4] Richard D. Neidinger , Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming, SIAM REVIEW, Vol. 52, No. 3, pp. 000–000 , 2010.

[5] G.Corliss, C.Faure, A.Griewank, and U.Naumann, Automatic Differentiation of Algorithms: From Simulation to Optimization, (Springer, 2002).

[6] T. F. Coleman and A. Verma. ADMAT: An Automatic Differentiation Toolbox for MATLAB. (Technical report, Computer Science Department, Cornell University, 1998).