Solving the Power Flow Problem via the

Newton-Raphson Method

By: Mike Mayhew and Michael Stickels

**Overview**

The power flowing around a system of interconnected buses, generators and loads can be determined by using the Power Flow equations. However, the complex power at each node within the system is dependent upon the values of power injected into the system, the system’s topography and the physical properties of the system. This yields a complicated series of equations where the inputs and outputs of each node are completely dependent upon those of their connected nodes. As such, we cannot solve the power flow equation parameters explicitly.

By reviewing the given characteristics of the system and understanding the desired (or physically required) performance of the system, we can assign identities to each node. These designations break down the power flow equations at each node into two groups; the explicit equations and the implicit equations. Explicit equations are, as implied, simpler to solve for. The implicit equations are more complicated and cannot be solved directly.

To solve the implicit equations, we use systems of equations methods from linear algebra. The overall process is known as the “Newton-Raphson Method.” This method works by building a matrix of the admittances between all nodes, building a matrix of all of the power flow equations at each node and another matrix with assumptions of values of the unknown variables. A Jacobian matrix of the power flow equations is built which includes four quadrants:

* H-quadrant (top left): Partial derivatives of P-equations with respect to angle θ.
* M-quadrant (top right): Partial derivatives of P-equations with respect to voltage.
* N-quadrant (bottom left): Partial derivatives of Q-equations with respect to angle θ.
* L-quadrant (bottom right): Partial derivatives of Q-equations with respect to voltage.

This Jacobian matrix is then inverted and it is multiplied by the negative matrix of the power flow equations. The assumed values of the unknown variables are plugged into this new “delta” matrix. It is named such because it is indicative of the changes of the unknown variables that have occurred. These suggested changes are added to the assumed values and the resulting matrix now takes the place of the previous version of the assumed value matrix.

These new suggested values for the unknown variables at each node are plugged into the power flow equation matrix which yields new values of the power flow equations at each node. The new values of P and Q are compared to the previous values. If the difference of these two values of P and Q at all nodes are within a designated tolerance, the power flow equations have been solved. If these values are outside of the tolerance, the associated unknown variable suggestions are returned to the top and the process is completed again with the new suggested values. This process continues until either the values of all unknowns yield P and Q values at all nodes are within the tolerance specified or, in rare circumstances, the values of P ad Q at all nodes diverge.

Convergence of the P and Q values indicates that correct voltage and phase angle at all nodes have been correctly determined. At this point, other system parameters such as power flows between nodes can be determined and checked against the physical limitations of the system.

**Visual Overview of Program to Solve Power Flow Equations**

Legend

Inputs / Outputs

System Quantities

System Mathematics

System Requirements

System Structure

Output Data File

Check line flows

Check bus voltages

Determine line flows

**Newton-Raphson Method**

Build convergence record

Log greatest P and Q mismatches and location

Log all values at each bus

Reference Voltages at PV busses

R, X, B and F line data

P and Q data at busses

Inside Mismatch Target

Outside Mismatch Target

Update Bus Voltage and Phase Angle Matrix

Calculate PQ Mismatches

Bus Voltage and Phase Angle Matrix

Inverse Jacobian Matrix

Jacobian Matrix

Admittance Matrix

Power Flow Equations

System Topography

System Components

Number of Generators

Number of Busses

Input Data File

System Values

Generator power data

**Details of Program to Solve Power Flow Equations**

Python was used to develop a program to solve the power flow equations in a hypothetical power system. The power system’s topography was provided along with data regarding all of the generators and busses. Some values of the variables at the busses were permanently established by the given data; others were to be solved for within the Python program.

Program front matter, imports and inputs:

As Numpy and Pandas were very helpful in the creation of this program, they were imported. Data about the system was imported from the provided .xlsx file. From this file, the system’s bus parameters and system topography could be directly determined.

Building the Admittance Matrix:

Two square matrices were initialized, one for each portion of the Y-matrix, both with the number of columns/rows equal to the number of busses. These two matrices, Real and Imaginary, were filled on the non-diagonal entries by having Python iterate over the input file, pull the R and X values associated with each bus and calculate the respective real or imaginary admittance value. For the the diagonal entries of these matrices, the axis entries were summed and then negated. Finally, the shunt admittances were added to the diagonal entries.

Building the P-Q Mismatch Matrix and V-θ Matrix:

This matrix was determined to be single dimension as would be later required for matrix multiplication. It was determined that this matrix would first need to include the P-value at all busses with the exception of the slack bus. Python iterated over the input file to place the given P values into this matrix. The program then iterated over the Q-values of the non-generating busses. It appended these Q-values to the bottom of the mismatch matrix.

As required for the Newton-Rapshon method, another matrix of the same shape was initialized to house the V and θ values. While the θ values were all initialized at 0, the V values were set to those within the input file.

Building the P-Q Equation Matrix:

The P and Q equation matrices were built as functions within Python to allow the V-θ matrix to be passed in as parameter. The values of the V-θ matrix were inserted into the power flow equations at each node to find current values of P and Q. Both of the P and Q functions determined a temporary copy of their respective matrix that will be later passed into other methods within the Newton-Raphson method.

Building the Jacobian Matrix

The Jacobian matrix’s shape was determined by finding the system topography via the input file. The four quadrants of the Jacobian matrix that were previously discussed in the Overview were built as individual matrices with shapes:

* H-quadrant: [# of busses – 1 (slack bus)] x [# of busses – 1 (slack bus)]
* M-quadrant: [# of busses – 1 (slack bus)] x [# of busses – # of generator busses]
* N-quadrant: [# of busses – # of generator busses] x [# of busses – 1 (slack bus)]
* L-quadrant: [# of busses – # of generator busses] x [# of busses – # of generator busses]

These four individual matrices had parameters passed in for indices that were to be used to place the values of the V-θ matrix. This converts all of the values within the four quadrant matrices from functions to floats. These four individual matrices are later assembled into a single full-size Jacobian matrix.

Functions for Updating the V-θ Matrix:

Later during the Newton-Raphson method calculations, an ability to update the V-θ matrix will be needed to carry out the mismatch calculations. **Caution must be exercised while updating the V-θ matrix**. This stems from the fact that while all θ values will be updated, only busses without generators will have the V updated. A temporary matrix was constructed with two dimensions to prevent the Vset values from changing away from their declared values. The first dimension was used to house the index within the full V-θ matrix where the updates would occur. The second dimension contains the updates. This temporary matrix contains the same number of axes as the Jacobian and PQ matrices. This is critical as we are unable to multiply matrices with dimensional mismatches.

After the updates to the V-θ matrix were made to the correct corresponding index, the values of this matrix were passed into functions that updated the power flow equations at all busses with the new values for V and θ.

Setting Initial Values for the V-θ Matrix and Preparation for Newton-Raphson:

The V-θ matrix was initialized with V values provided by the input file and all θ values were set to the “flat start condition” of 0. In this same section, mismatch parameters were declared and a maximum number of iterations was added as a method to prevent the program from running eternally if the outputs of the program should result in divergence.

Newton-Raphson Algorithm

The Newton-Raphson method operates as a “while loop” as long as the mismatch of P and Q at any bus is larger than the acceptable mismatch of 0.1. The full-size Jacobian matrix from above is inverted, multiplied by the P and Q values at all busses and negated. The output of this operation is the change that needs to be made for each corresponding V and θ value in the system.

These changes are summed into the V-θ matrix and the new quantities in the V-θ matrix are sent into the P and Q equation functions to find the new values of P and Q at every bus. The new values of P and Q are compared to the assigned quantities of P and Q from the input file. If any of the values of P or Q vary by more than 0.1 from the assigned value at that bus, the new values of the V-θ matrix are sent back into the Newton-Raphson method.

If programmed correctly, the V-θ matrix should continue to be updated until the mismatch requirement is met. At that point, all variables have been solved for at all busses.

Output Information

Once all system parameters have been determined by the Newton-Raphson method, line flows can be calculated. All of this information is housed within a new matrix that defines all system characteristics. These values are compared to the ratings and requirements of the system, which are housed in another matrix. If system requirements are violated, the violation is flagged and printed with the output data.

**System Testing**

Throughout the creation of the program, print statements were used to ensure completed portions of the program were acting as expected or provided correct data. These statements include spot checks throughout the program or full displays of large matrix values. Indices were also checked via print statements. Programming indices was found to be one of the most demanding parts of writing the program. The large number of matrices used in the program have vastly different shapes and individual matrices have differing values within them that are used by multiple other matrices.

Additional system testing would be used to ensure the outputs of the program meet the system’s requirements and to verify that ratings within the system had not be violated.

Results obtained

(V and theta at each bus)

(Active and Reactive power produced by each gen)

(Violation of normal operating limits)

Discussion of these results

Convergence record

(largest active power mismatch)

(largest reactive power mismatch and bus location)

Contributions of each team member

Documented code