f Potencia

Santiago Peñate Vera

```
Initial CX Solution:
S       V
(0,0)   (1,0)->1^0
(-0.4,-0.2)     (1,-0.0312951)->1.00049^-0.0312849
(-0.25,-0.15)   (1,-0.0243302)->1.0003^-0.0243254
(0.3,0) (1,-0.0104352)->1.00005^-0.0104348
(-0.5,-0.2)     (1,-0.0277923)->1.00039^-0.0277852
Run  0
iterations:2
Sbase = 100
Line 1-2
         r:0.05, x:0.11, c: 0.02
        Power
         bus 1 to 2: (27.9588,13.0714)
         bus 2 to 1: (-27.469,-13.9362)
         Losses: (0.489849,-0.864846)
        Current
         bus 1 to 2: (2.79588,-1.30714)
         bus 2 to 1: (-2.79351,1.50419)
Line 1-3
         r:0.05, x:0.11, c: 0.02
        Power
         bus 1 to 2: (21.6984,11.6645)
         bus 2 to 1: (-21.3828,-12.9215)
         Losses: (0.315605,-1.25703)
        Current
         bus 1 to 2: (2.16984,-1.16645)
         bus 2 to 1: (-2.16809,1.36397)
Line 1-5
         r:0.03, x:0.08, c: 0.02
        Power
         bus 1 to 2: (33.9157,18.4483)
         bus 2 to 1: (-33.4571,-19.1751)
         Losses: (0.458553,-0.726841)
        Current
         bus 1 to 2: (3.39157,-1.84483)
         bus 2 to 1: (-3.38944,2.04226)
Line 2-3
         r:0.04, x:0.09, c: 0.02
        Power
         bus 1 to 2: (-7.48242,-2.49848)
         bus 2 to 1: (7.50721,0.660377)
         Losses: (0.024788,-1.8381)
        Current
         bus 1 to 2: (-0.764205,0.276108)
         bus 2 to 1: (0.768331,-0.0815315)
Line 2-5
         r:0.04, x:0.09, c: 0.02
        Power
         bus 1 to 2: (-3.72902,-3.23789)
         bus 2 to 1: (3.73716,1.36403)
         Losses: (0.00813752,-1.87385)
        Current
         bus 1 to 2: (-0.375843,0.342802)
         bus 2 to 1: (0.380344,-0.148321)
Line 3-4
         r:0.06, x:0.13, c: 0.03
        Power
         bus 1 to 2: (-15.3067,-12.7688)
         bus 2 to 1: (15.5356,10.3377)
         Losses: (0.228891,-2.43111)
        Current
         bus 1 to 2: (-1.54552,1.33712)
         bus 2 to 1: (1.54883,-1.04084)
Line 4-5
         r:0.04, x:0.09, c: 0.02
        Power
         bus 1 to 2: (26.2191,15.8431)
         bus 2 to 1: (-25.8306,-16.9187)
         Losses: (0.388452,-1.07563)
        Current
         bus 1 to 2: (2.61467,-1.59624)
         bus 2 to 1: (-2.61208,1.79366)
bus1 -> VD
        Power: (83.5729,43.1841)
        Voltage: (10,0) Voltage p.u.: (1,0)
bus2 -> PQ
        Power: (-40,-20)
        Voltage: (9.70542,-0.23719)     Voltage p.u.: (0.970542,-0.023719)
bus3 -> PQ
        Power: (-25,-15)
        Voltage: (9.7522,-0.17536)      Voltage p.u.: (0.97522,-0.017536)
bus4 -> PV
        Power: (30,-15)
        Voltage: (9.9999,-0.0455512)    Voltage p.u.: (0.99999,-0.00455512)
bus5 -> PQ
        Power: (-50,-20)
        Voltage: (9.74267,-0.212981)    Voltage p.u.: (0.974267,-0.0212981)

circuit 1 solution mismatch: 0.141586
```

First publishing, December 2014

About the me and my motivation for producing this work.

Dear reader, my name is Santiago, I studied Industrial engineering {Electrical speciality} at the Escuela Técnica Superior de Ingenieros Industriales of Las Palmas of Gran Canaria, Spain.

I worked in a little start-up company, 3iDS programming optimization software to reverse engineer the generation dispatch, to be able to find the real generation cost of electricity in the island of Gran Canaria where I am from. We were so successful in finding this, and proposing a more efficient, stable and cheaper model, that the project was not pursued further.

After this episode I emigrated to the The Netherlands, and started as intern at DNV-KEMA. At the same time I was also finishing a master in RAMS (Reliability, Availability Maintainability and Safety) When I finished the practice period I joined DNV GL in the research unit of Power and Electrification.

At this unit I have been about a year and a half working in power system optimization, storage and other topics related to the smart grid development. During this period I purchased many books on the power flow subject, read many Ph.D. thesis and learned as much as I could. Now I can say that the power flow solution, that once was presented to me as something very complicated and distant, became familiar and more transparent, therefore my motivation for the production of this work; the power flow library and this manual, is to gain proficiency on the subject, as well as to produce a usable research product for you.

I must say that I decided to choose the electricity field because it is very much related to mathematics, and because it has a very large potential of being disruptive in how the things are ordered today (linking with the story I told at the beginning). So, let's begin.

SANTIAGO PEÑATE VERA, INDUSTRIAL ENGINEER
December 2014

Contents

1. *f*Potencia

*f*Potencia stands for *flujo de potencia*, the spanish words for power flow.

1.1 What is *f*Potencia?

*f*Potencia is a library for positive sequence power flow simulations.

*f*Potencia is specifically designed to be used as a library, it is made to allow you to run it in parallel taking the most of your simulation hardware. It is implemented in C++ because at the moment C++ seems to be the most popular high performance language.

1.2 Getting *f*Potencia

*f*Potencia is hosted at:

- `https://sourceforge.net/projects/fpotencia/`

You may get the latest development version using GIT:

```
git clone git://git.code.sf.net/p/fpotencia/code fpotencia-code
```

On Linux machines you can obtain GIT from the package manager, on Windows machines you can use the Official GIT installation `http://git-scm.com/downloads`.

1.3 Programming environments

I have tried to keep *f*Potencia as cross-platform as possible. Standard C++ is itself cross platform, that means that with an adequate compiler you can have the library working in the operative system of your choice. The code is written using the C++11 standard.

Since I work with Windows and Linux machines and I do not own an Apple computer, I will only be giving examples and instructions for Windows and Linux. The examples and instructions might be taken as some sort of guidance for the rest of platforms.

Specifically, in Linux I develop using Netbeans (v8.0.x) with its C/C++ plug-in, and in Windows I develop using visual studio 2013 or Netbeans as well. You will find both IDE projects in the *f*Potencia folder along with the code.

The code { `src` } and libraries { `libs` } are kept separated from the diverse projects, this means that the libraries and the code are common to all the provided IDE projects.

1.4 Installing the dependencies

*f*Potencia depends on some external libraries to handle the linear algebra operations with complex numbers.

Eigen: is a powerful linear algebra library.

- Windows: If you use the Visual Studio project or the Netbeans project, then you don't need to do anything. Eigen will be automatically included. You will find Eigen in the folder blow.

  ```
  <fPotencia folder>/libs/eigen3
  ```

- Linux(Ubuntu): You can install Eigen through the package manager:

  ```
  sudo apt-get install libeigen3-dev.
  ```

  If you cannot find a package for your distribution, then just add the Eigen folder to you C++ project. By adding the *libs* folder to your C++ project in the *Includes* field of the project configuration.

1.5 Compiling *f*Potencia

*f*Potencia is written in standard C++ 2011, this means that you are suposed to be able to compile it under any architecture if using the correct compiler.

For the production of it I have used indiscrminatelly the gcc compiler under Linux and the gcc compiler provided by the MinGW project under Windows (They are suposed to be the same). As development environment I use Netbeans 8.0+ and I provide both Linux and Windows Netbens projects to help you start. I you preffer another IDE, it should be straight forward to include the code since you just need to include all the files in the */src* folder. There are no special configurations, all the steps are explained in this document.

1.6 Using *f*Potencia in your project

*f*Potencia itself is compiled as a static library. `fpotencia.a` in Linux, and `fpotencia.lib` in Windows. I was investigating about the difference of dynamic and static libraries and I couldn't find a good reason to not having it static, but of course you are free to compile it as you want.

1.6.1 Visual Studio

Visual Studio is only available for Windows.

In the project menu page:

```
project configuration {menu} > configuration properties > VC++ Directories
>
```

- `Include directories` : Directories of the header files used in your project (yes, the directory where the *f*Potencia headers are).
- `Library directories` : Directory where `fpotencia.lib` is located.

*Note: My experience with the Microsoft compiler is quite negative. In 32-bit mode, and for no good reason the produced executable does not converge for the ieee30 Bus test case. Compiling in 64-bit mode with visual studio leads to variables types mismatches and the results are incorrect (when converging). However under Linux, compiling under 32-bit or 64-bit the program produces the expected results.

1.6.2 Netbeans

Netbeans is available for both Windows and Linux.

In the project menu page:
`project configuration {menu} > Build >`

- `C++ Compiler > Include directories` : Directories of the header files used in your project (yes, the directory where the *f*potencia headers are). For example `../../../src; ../../../libs` for *f*Potencia as you find it in the GIT repository.
- `Linker > Libraries` : Selet the file `fpotencia.lib` if you are under Windows or `fpotencia.a` if you are under Linux.

x64

The link below explains how to configure MinGW (GCC for Windows) to work with Netbeans.
`http://wiki.netbeans.org/HowToCreate64-BitC/C++WindowsProgram`

When using MinGW (64 bits) under windows, you must set the compiler additional options to `-static -static-libgcc -static-libstdc++` otherwise the compiled program will not work as expected.

Object oriented programming
Why should a power flow solver have a modular structure?

2. Structure

In *f*Potencia, I wanted that the circuit and the solvers were completely separated. This allows to re use the circuit object in several different solvers, allowing easy exchange of solutions between solvers, and parallel simulation of different circuit states by initializing many solvers with the same circuit in a parallel loop.

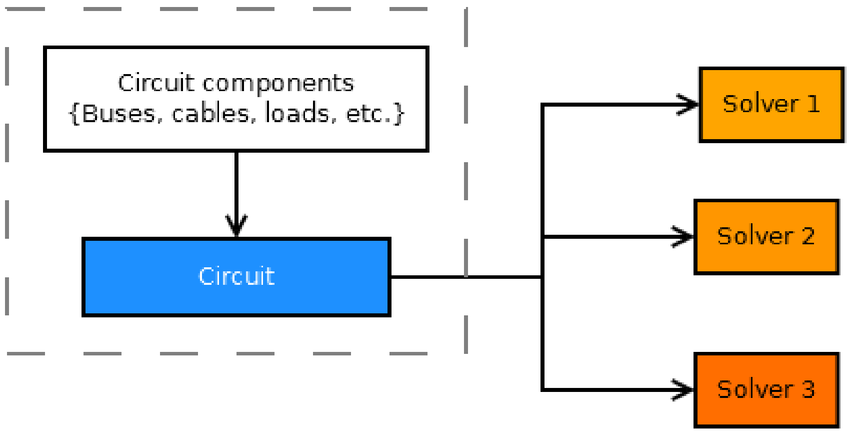

Figure 2.1: General structure

An example of this is:

```
Circuit model = ieee_30_bus();
model.compile(false);

Solver_ZBusGS zbs(model);
zbs.solve();
zbs.Model.print();

Solver_NRpolar nrs(model);
nrs.solve();
nrs.Model.print();
```

In this example we declare a circuit and we load the hard coded example *IEEE*_30_*Bus* grid. Then the circuit is compiled, this calculates the admittance and impedance matrices. After this, two solvers are initialized with the same grid and both solvers are executed. This modularity is very advantageous, but perhaps it is not obvious at this stage of the project.

2.1 Object oriented programming

A real life object has some properties and some functions, for example a teeth brush can have the following properties:

1. Color
2. Length
3. Category
4. Age

And the following functions:

1. Brush peoples teeth
2. Clean the toilet

In an object oriented programming language, one can create objects and those objects can contain properties that define them and functions that they can provide.

I am convinced that there are better explanations, but this definition is pretty simple to approach the structure of *f*Potencia. If you want to know more I encourage you to run a search on internet for tutorials on OOP in C++ or other simpler languages like python.

To illustrate our case, let's take the example of a line. It is an object with several properties, like for example the impedance, and with some functions like for example, to calculate its admittance matrix.

2.2 Why should a power flow solver have a modular structure?

The solution of a power system is a linear process, were there are certain unknown values that we want to solve. Given a circuit definition we calculate the admittance matrix, the initial solution, and by a numerical method we solve the power flow equations. If this is such a 'linear' process, why should we split everything in to objects and then combine them and all that?

The answer is; The power flow itself might be a linear process, but many processes that use power flows can be executed in parallel. A process executed in parallel can for example create a circuit, calculate its admittance matrix only once, and change the loads stochastically, solving the power flows with different instances of the object solver all in different parallel processing channels, joining all the results in a single final results array.

In short, without a modular structure we cannot use a power flow solver efficiently to do parallel programming and therefore to take the most of our multi core hardware.

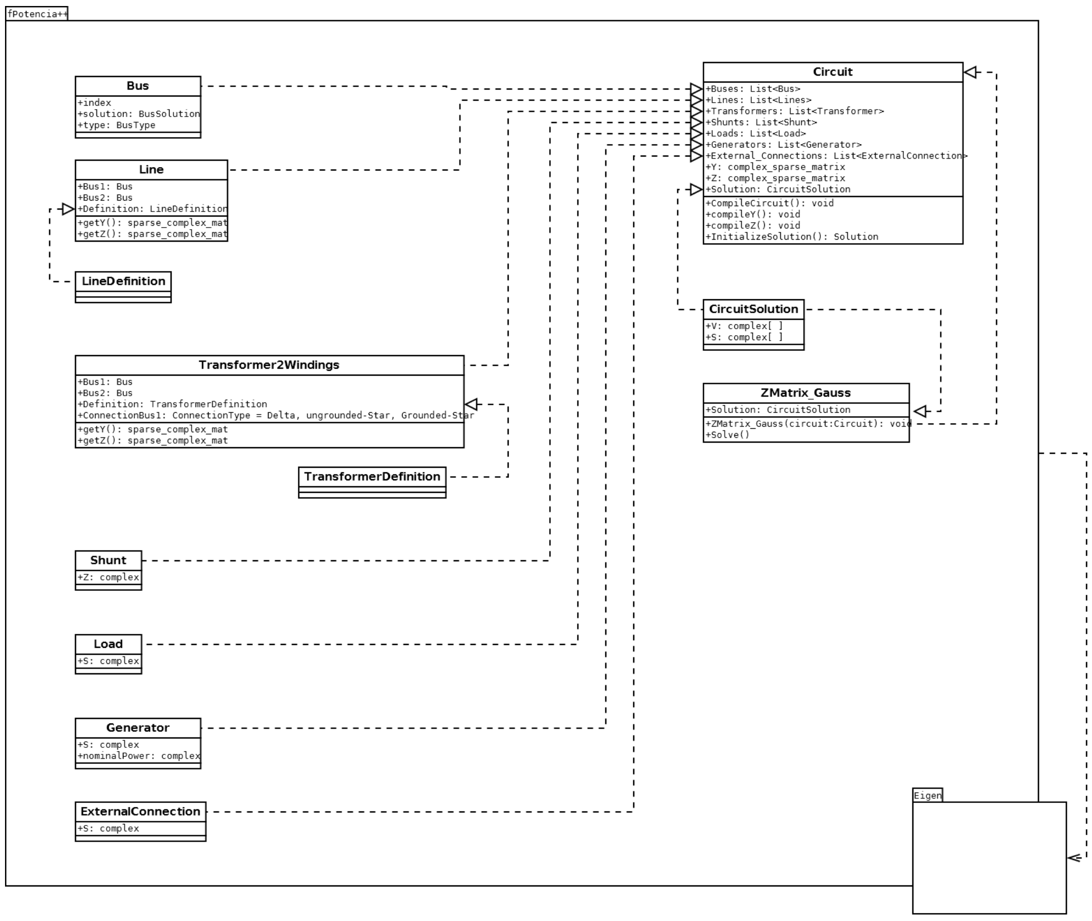
fPotencia++
Bus
+index
+solution: BusSolution
+type: BusType
Line
+Bus1: Bus
+Bus2: Bus
+Definition: LineDefinition
+getY(): sparse_complex_mat
+getZ(): sparse_complex_mat
LineDefinition
Transformer2Windings
+Bus1: Bus
+Bus2: Bus
+Definition: TransformerDefinition
+ConnectionBus1: ConnectionType = Delta, ungrounded-Star, Grounded-Star
+getY(): sparse_complex_mat
+getZ(): sparse_complex_mat
TransformerDefinition
Shunt
+Z: complex
Load
+S: complex
Generator
+S: complex
+nominalPower: complex
ExternalConnection
+S: complex
Circuit
+Buses: List<Bus>
+Lines: List<Lines>
+Transformers: List<Transformer>
+Shunts: List<Shunt>
+Loads: List<Load>
+Generators: List<Generator>
+External_Connections: List<ExternalConnection>
+Y: complex_sparse_matrix
+Z: complex_sparse_matrix
+Solution: CircuitSolution
+CompileCircuit(): void
+compileY(): void
+compileZ(): void
+InitializeSolution(): Solution
CircuitSolution
+V: complex[ ]
+S: complex[ ]
ZMatrix_Gauss
+Solution: CircuitSolution
+ZMatrix_Gauss(circuit:Circuit): void
+Solve()
Eigen

Figure 2.2: *f*Potencia class diagram

3. Using the library

In this chapter I will show how to create the different elements in the library and the preferred order to create them. I will also explain what happens in the library to produce each outcome.

3.1 Includes (C++)

To use the *f*Potencia library in your C++ project, first you must include it in the project as described in the section 1.6.

To use the code, just simply include the fpotencia.h file.

```
#include "../fpotencia.h"
```

Or

```
#include "fpotencia.h"
```

Depending of how do you include the library references.

3.2 Creating a circuit

In this section I explain how to create a circuit. The circuit is the element that contains the network devices (lines, transformers, bus bars, etc.) and it produces the essential mathematical objects for other parts of the program, for instance the circuit object creates the admittance matrix that is used in the Newton-Raphson solver.

3.2.1 Circuit

Object constructor

`Circuit(string name)`

- name: The name of the circuit

Example

```
Circuit ieee30("IEEE 30 bus circuit");
```

3.2.2 Bus

Object constructor

`Bus(string name, const BusType& type, double Bus_Nominal_Voltage)`

- name: The name of the bus bar.
- type: Type of bus bar:
  1. PQ
  2. PV
  3. VD (Slack bus)
  4. undefined_bus_type
- Bus_Nominal_Voltage: This is the rated voltage of the bus bar. I.e.: 10 kV.

Example

```
//create a bus bar object
Bus b1("bus1", undefined_bus_type, 10.0);

//Add the bus bar object to the already created circuit object ieee30
ieee30.add_Bus(b1)
```

By adding a bus bar object into the circuit, the object ($b1$ in this example) is assigned the index property and added to the circuit list of bus bars. This allows to use the bus bar index in the creation of new circuit objects. In this sense there is the necessity of creating a circuit first and then create the bus bars. The order of the creation of the rest of the network devices is not relevant.

3.2.3 Line type

Object constructor

`LineType(string name, double r, double x, double b, bool per_unit_values)`

- name: The name of the line type.
- r: Line resistance value per length unit.
- x: Line inductance value per length unit.
- b: Line shunt reactance per length unit.
- : per_unit_values: Boolean value, it is true if r, x, b are provided already as per unit values.

Example

```
LineType ltype1("line type 1", 0.05, 0.11, 0.02, true);
```

3.2.4 Line

Object constructor

```
Line(string name, int connection_bus1, int connection_bus2,
     LineType line_type, double line_lenght)
```

- name: The name of the line.
- connection_bus1: First connection bus index.
- connection_bus2: Second connection bus index.
- line_type: Type of line. Should be defined before.
- : line_lenght: Length of the line. Remember that it has to be consistent with the length units used in the line type.

Since the line object is symmetric, the order of the first and second bus does not really matters.

Example

```
Line l1("Line 1-2", b1.index, b2.index, ltype1, 1.0);
```

3.2.5 Transformer type

The transformer type has two constructors; the first provides the possibility to define a transformer type from its short circuit test, and the second allows to requires the values processed from the short circuit test, this is, the π equivalent values of the transformer model. To know more please see the section 5.2.2.

Object constructor from short circuit test

```
TransformerType(string name, double HV_nominal_voltage,
                double LV_nominal_voltage, double Nominal_power,
                double Copper_losses, double Iron_losses,
                double No_load_current, double Short_circuit_voltage,
                double Tap_position, double Phase_Shift, double GX_HV1 = 1,
                double GR_HV1 = 1)
```

- name: The name of the circuit
- HV_nominal_voltage: Primary voltage in kV.
- LV_nominal_voltage: Secondary voltage in kV.
- Nominal_power: Transformer nominal power in MVA.
- Copper_losses: Transformer copper losses in kW.
- Iron_losses: Transformer iron losses in kW.
- No_load_current: No load current in %.
- Short_circuit_voltage: Short circuit voltage in %.
- Tap_position,: The tap value in per units (i.e. 1.01 equals a 1% voltage increase in the secondary).
- Phase_Shift: Angle that the voltage is shifted in radians. (30 deg=$\pi/6$ is the delta-star value)
- GX_HV1: Reactance contribution to the HV side. value from 0 to 1.
- GR_HV1: Resistance contribution to the HV side. value from 0 to 1.

Example

```
TransformerType TType1("Transformer type1", 66.0,
                       10.0, 50.0, 15.0, 7.0, 18.0,
                       50.0, 1.0, 0.5236, 0.5, 0.5 );
```

Object constructor from per unit values

```
TransformerType(string name, cx_double leakage_z,
                cx_double magnetizing_z)
```

- name: The name of the circuit
- leakage_z: Leakage impedance as complex number in per unit values.
- magnetizing_z: Magnetizing impedance as complex number in per unit values.

The leakage and magnetizing impedances are obtained as described in the section 5.2.2.

Example

```
TransformerType TType1("Transformer type1",
                      cx_double(0.05, 0.02), cx_double(0.0, 0.0004));
```

3.2.6 Transformer

Object constructor

```
Transformer(string name, int connection_busHV,
            int connection_busLV, TransformerType transformer_type)
```

- name: The name of the transformer.
- connection_busHV: Index of the primary side bus bar.
- connection_busLV: Index of the secondary side bus bar.
- transformer_type: Transformer type object

As you may imagine, the primary and secondary connexions are very important and should not be confused.

Example

```
Transformer T1("Transformer 1", b1.Index, b2.Index, TType1);
```

3.2.7 Load

Object constructor

```
Load(string name, int connection_bus, double P, double Q)
```

- name: The name of the circuit
- connection_bus: Index of the connection bus bar.
- P: Active power of the load.
- Q: Reactive power of the load.

Example

```
Load L1("Load 1", b1.Index, 10.0, 8.0);
```

3.2.8 Generator

Object constructor as simple generator

```
Generator(string name, uint connection_bus, double P, double Q)
```

When this constructor is used the generator acts as a negative load.

- name: The name of the generator.
- connection_bus: Index of the connection bus bar.
- P: Generator active power.
- Q: Generator reactive power.

Example

```
Generator G1("Generator 1", b1.Index, 15.0, 8.0);
```

Object constructor as voltage controlled generator

```
Generator(string name, uint connection_bus, double P, double Vset,
          double Qmin, double Qmax, bool Vset_per_unit)
```

When this constructor is used the generator is voltage controlled, this means that it sets its connection bus voltage to the $Vset$ value and the generator reactive power is calculated after.

- name: The name of the generator.
- connection_bus: Index of the connection bus bar.
- P: Generator active power.
- Vset: Voltage module set at the generator output (in the connection bus bar voltage magnitude or in per unit)
- Qmin: Minimum reactive power allowed.
- Qmax: Maximum reactive power allowed.
- Vset_per_unit: Boolean value, if true the voltage set is in per unit.

Example

```
Generator G1v("Generator 1", b1.Index, 15.0, 1.0, -10.0, 10.0, true);
```

3.2.9 External connection

This object turns the bus where it is connected to a slack bus. If this object is used, then it is not necessary to specify the slack bus type when defining the buses.

Object constructor

```
ExternalGrid(string name, int connection_bus)
```

- name: The name of the external circuit connection.
- connection_bus: Index of the connection bus bar.

Example

```
ExternalGrid("External grid 1", b1.Index);
```

3.3 Example circuit definition

Find here an example of how to create a circuit in fPotencia using the constructors explained before. You can find more information about the structure in the chapter 2.

```cpp

Circuit circuit_Lynn_Powell() {

    Circuit model("circuit Lynn Powell 5-bus");

    //Buses definition, this is the first thing to do.
    Bus b1("bus1", undefined_bus_type, 10.0);
    Bus b2("bus2", undefined_bus_type, 10.0);
    Bus b3("bus3", undefined_bus_type, 10.0);
    Bus b4("bus4", undefined_bus_type, 10.0);
    Bus b5("bus5", undefined_bus_type, 10.0);
    model.add_Bus(b1);
    model.add_Bus(b2);
    model.add_Bus(b3);
    model.add_Bus(b4);
    model.add_Bus(b5);
    /* once each bus is added to the list, its Index property is modified
     * reflecting the index in which it was added to the circuit
     */

    //External grids
    ExternalGrid eg("External1", b1.index);
    model.external_grids.push_back(eg);

    //Lines types definition
    LineType ltype1("line type 1", 0.05, 0.11, 0.02, true);
    LineType ltype2("line type 2", 0.03, 0.08, 0.02, true);
    LineType ltype3("line type 3", 0.04, 0.09, 0.02, true);
    LineType ltype4("line type 4", 0.06, 0.13, 0.03, true);

    Line l1("Line 1-2", b1.index, b2.index, ltype1, 1.0);
    Line l2("Line 1-3", b1.index, b3.index, ltype1, 1.0);
    Line l3("Line 1-5", b1.index, b5.index, ltype2, 1.0);
    Line l4("Line 2-3", b2.index, b3.index, ltype3, 1.0);
    Line l5("Line 2-5", b2.index, b5.index, ltype3, 1.0);
    Line l6("Line 3-4", b3.index, b4.index, ltype4, 1.0);
    Line l7("Line 4-5", b4.index, b5.index, ltype3, 1.0);

    //here the line objects are added to the circuit
    model.lines.push_back(l1);
    model.lines.push_back(l2);
    model.lines.push_back(l3);
    model.lines.push_back(l4);
    model.lines.push_back(l5);
    model.lines.push_back(l6);
    model.lines.push_back(l7);

    //Generators
    Generator g1("Generator", b4.index, 70, 10, 20, -15);
    model.generators.push_back(g1);

    //Loads
    Load ld2("Load1", b2.index, 40, 20);
    Load ld3("Load2", b3.index, 25, 15);
    Load ld4("Load3", b4.index, 40, 20);
    Load ld5("Load4", b5.index, 50, 20);
    model.loads.push_back(ld2);
    model.loads.push_back(ld3);
    model.loads.push_back(ld4);
    model.loads.push_back(ld5);

    return model;
}
```

4. Power system problem definition

4.1 The power flow equations

$$\overline{\mathbf{S}} = \overline{\mathbf{V}}\, x\, \overline{\mathbf{I}}^* \tag{4.1}$$

This is the most fundamental equation to solve. The apparent power $\overline{\mathbf{S}}$ is equal to the product of the voltage $\overline{\mathbf{V}}$ by the conjugate of the current $\overline{\mathbf{I}}^*$. This equality is a linear system. Very simple to solve, the problems is that in a normal power system the unknowns are not all concentrated in the same vector. Some unknowns are in $\overline{\mathbf{S}}$, others are in $\overline{\mathbf{V}}$ and certainly all $\overline{\mathbf{I}}$'s are unknown. This will be explained further in when explaining about the node types.

By the law of Ohm, we know that $\overline{\mathbf{V}} = \overline{\mathbf{Z}}x\overline{\mathbf{I}}$, which provides $\overline{\mathbf{I}} = \overline{\mathbf{Y}}x\overline{\mathbf{V}}$, substituting in the main equation we obtain:

$$\overline{\mathbf{S}} = \overline{\mathbf{V}}_n x(\overline{\mathbf{Y}}x\overline{\mathbf{V}})^* \tag{4.2}$$

With this last equation we have reduced the number of unknowns, making simpler the solution of the problem. Now we only need to solve the power and the voltage at each bus bar, the current is no longer an explicit unknown. Therefore the only magnitudes to monitor are the power summation at each bus and the voltage at each bus.

But in our problem we must discretize the problem for the n nodes, then the equation $\overline{\mathbf{I}} = \overline{\mathbf{Y}}x\overline{\mathbf{V}}$ turns into:

$$\overline{I}_k = \sum_{j=0}^{n-1} \overline{V}_j \overline{Y}_{kj} \quad k = 0, 1, 2, 3...\text{n-1} \tag{4.3}$$

And the equation $\overline{\mathbf{S}} = \overline{\mathbf{V}}\, x\, \overline{\mathbf{I}}^*$ turns into:

$$\overline{S}_k = \overline{V}_k\, \overline{I}_k^* \tag{4.4}$$

Combining the last two expressions we obtain:

$$\overline{S}_k = \overline{V}_k \left(\sum_{j=0}^{n-1} \overline{V}_j \overline{Y}_{kj} \right)^* \quad k = 0, 1, 2, 3...\text{n-1} \tag{4.5}$$

The equation 4.5 provides the value of the power at any bus bar of the system. Now we need an expression for the voltage at the bus bars. We may rewrite the equation 4.5 as:

$$\frac{\overline{S}_k^*}{\overline{V}_k^*} = \sum_{j=0}^{n-1} \overline{V}_j \overline{Y}_{kj} \quad k = 0, 1, 2, 3...\text{n-1} \tag{4.6}$$

Taking the $j = k$ term outside the summation we get:

$$\frac{\overline{S}_k^*}{\overline{V}_k^*} = \overline{V}_k \overline{Y}_{kk} + \sum_{j=0, j\neq k}^{n-1} \overline{V}_j \overline{Y}_{kj} \quad k = 0, 1, 2, 3...\text{n-1} \tag{4.7}$$

Clearing the voltage $\overline{V}_k$:

$$\overline{V}_k = \frac{1}{\overline{Y}_{kk}} \left(\frac{\overline{S}_k^*}{\overline{V}_k^*} - \sum_{j=0, j\neq k}^{n-1} \overline{V}_j \overline{Y}_{kj} \right) \quad k = 0, 1, 2, 3...\text{n-1} \tag{4.8}$$

The equation 4.8 provides the formula to calculate the voltage at a bus bar of the system.

Magnitudes mentioned:

- $\overline{\mathbf{S}}$: Apparent power vector (complex for AC systems).
- $\overline{\mathbf{V}}$: Voltage vector (complex for AC systems).
- $\overline{\mathbf{V}}_n$: Identity matrix multiplied by the voltage vector (a square matrix with the voltage vector values in its diagonal).
- $\overline{\mathbf{I}}$: Current vector (complex for AC systems).
- $\overline{\mathbf{Y}}$: Admittance matrix (complex for AC systems).
- $\overline{\mathbf{Z}}$: Impedance matrix (complex for AC systems).

Common complex representations:

- $\overline{S} = P + jQ$
- $\overline{V} = E + jF = \|V\|^{\llcorner \theta}$
- $\overline{Y} = G + jB$
- $\overline{Z} = R + jX$

4.2 The node types

As we said the variables that come into play in the power flow problem are the bus bars power $\overline{S}$ and voltage $\overline{V}$. If those two variables are accounted as complex, which is the case of the alternate current systems, then the variables to account are P (the active power), Q (the reactive power), $\|V\|$ (the voltage module) and θ (the voltage angle, also called D)

The most extended node types are:

- **PQ**: where the power in complex form is given. This type of node is associated with power injections (generators) or extractions (loads) at the bus bar.
- **PV**: where the active power and the voltage magnitude are given. This type of node is associated with bus bars where there are synchronous generators that can keep a constant voltage output.
- **Vθ** or **VD** or **Slack**: where the voltage in complex form is given. This type of node is associated with external grid connections where there is power exchange and the voltage can be considered fix. In the case of transmission grids this necessary bus type introduces a malformation in the modelling, this is explained in the node types implications.

4.3 The node types implications

Here I will discus about what does it implicate the use of nodes where some quantities come fixed and others are to be solved.

4.3.1 The slack node

This is a node where the voltage comes fix (both module and angle). This is essential to the problem because it needs a voltage reference to propagate the rest of the voltage values in the non-slack bus bars given a reference. Without a reference the problem cannot be solved using fast iterative numerical techniques.
In transmission grids the problem is that a slack node is a place of infinite power, usually the slack node is a bus bar that hosts a very big generator. In this idealization the slack generator will supply the grid losses, which is a completely unrealistic situation since in the real life all the connected generators provide load power and the grid losses.
The slack node is necessary to solve the problem with fast solvers like Newton-Raphson or Gauss Seidel. The power flow problem can be solved using optimization techniques that are slower but can certainly produce a much more realistic outcome if programmed correctly.

Given that the only equation to fulfil is the equation 4.1, any set of variables that solves this equality is a mathematically correct solution. Of course that solution set must be within a feasible range to be usable.

4.3.2 The slack node and the island mode

The slack node in an electric model is not a burden if it models an external connection. This is the case of the low and medium voltage networks. But what happens if one wants to simulate a grid in island mode, this is a grid that has no connection with other grids (like many transmission cases)?

One suggestion to achieve that would be to optimize the generators power to provide the power initially provided by the slack bus, this is the same as optimizing the generation power to make the slack bus power equal to zero. This is true only if we connect to the slack bus a generation bus with a zero impedance connection, this is like modelling two generators connected to very close bus bars but we will try to make the slack generation null.

5. Modelled components

5.1 Lines

5.1.1 π model

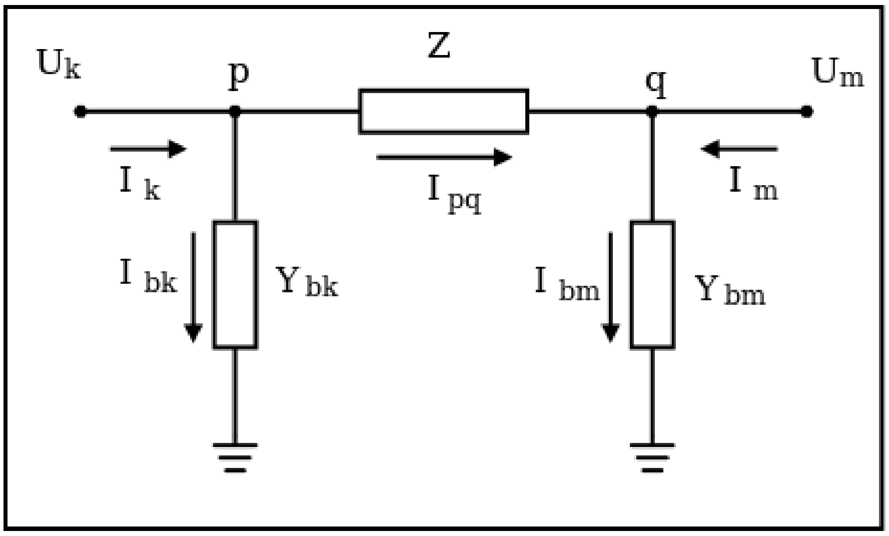

Figure 5.1: π model of a transmission line

admittance matrix

This is the line circuit admittance matrix. If represented in sparse mode it can be directly added to the general circuit admittance matrix.

$$\mathbf{Y}_{line} = \begin{bmatrix} Y_{pq} + Y_{bk} & -Y_{pq} \\ -Y_{pq} & Y_{pq} + Y_{bm} \end{bmatrix} \tag{5.1}$$

$$Y_{pq} = 1/Z, \quad Y_{bm} = Y_{bk} = Y_{shunt}/2 \tag{5.2}$$

Currents calculation

After calculating the circuit voltages at the bus bars with any of the solvers, the lines individual currents are calculated with this representation of the Ohm law.

$$\mathbf{I}_{line} = \begin{bmatrix} Y_{kk} & Y_{km} \\ Y_{mk} & Y_{mm} \end{bmatrix} \begin{bmatrix} V_k \\ V_m \end{bmatrix} \tag{5.3}$$

Remember that the elements admittance matrix were calculated for the composition of the circuit admittance matrix, therefore they remain for the currents calculation once the circuit is solved.

5.2 Transformers

5.2.1 π model

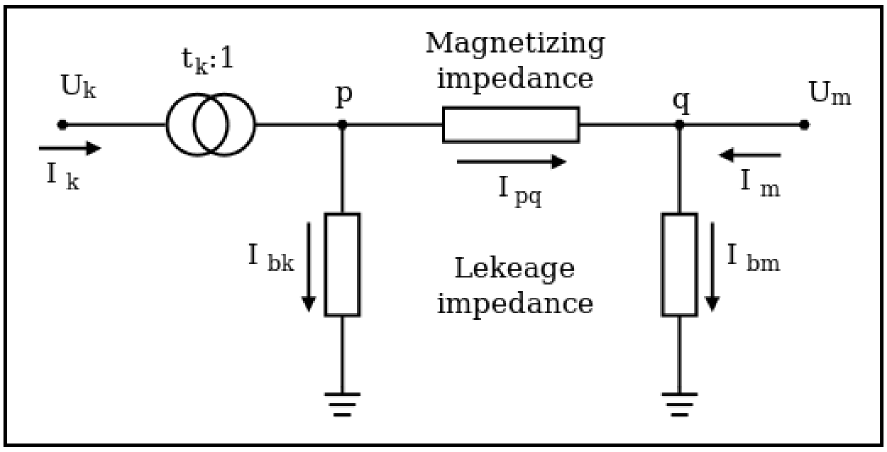

Figure 5.2: π model of a transformer

5.2.2 Model parameters calculation from short circuit test values

Nominal impedance HV (Ohm)

$$Zn_{hv} = U_{hv}^2 / S_n \tag{5.4}$$

Nominal impedance LV (Ohm)

$$Zn_{lv} = U_{lv}^2 / S_n \tag{5.5}$$

Short circuit impedance (p.u.)

$$z_{sc} = U_{sc} / 100 \tag{5.6}$$

Short circuit resistance (p.u.)

$$r_{sc} = \frac{P_{cu}/1000}{S_n} \tag{5.7}$$

Short circuit reactance (p.u.)

$$x_{sc} = \sqrt{z_{sc}^2 - r_{sc}^2} \tag{5.8}$$

HV resistance (p.u.)

$$r_{cu,hv} = r_{sc} \cdot GR_{hv1} \tag{5.9}$$

LV resistance (p.u.)

$$r_{cu,lv} = r_{sc} \cdot (1 - GR_{hv1}) \tag{5.10}$$

HV shunt reactance (p.u.)

$$xs_{hv} = x_{sc} \cdot GX_{hv1} \tag{5.11}$$

LV shunt reactance (p.u.)

$$xs_{lv} = x_{sc} \cdot (1 - GX_{hv1}) \tag{5.12}$$

Shunt resistance (p.u.)

$$r_{fe} = \frac{Sn}{P_{fe}/1000} \tag{5.13}$$

Magnetization impedance (p.u.)

$$z_m = \frac{1}{I_0/100} \tag{5.14}$$

Magnetization reactance (p.u.)

$$x_m = \frac{1}{\sqrt{\frac{1}{z_m^2} - \frac{1}{r_{fe}^2}}} \tag{5.15}$$

If the content of the square root is negative, set the magnetization impedance to zero.

The final complex calculated parameters in per unit are:

Leakage impedance

$$Z_l = r_{sc} + j \cdot x_{sc} \tag{5.16}$$

Magnetizing impedance

$$Z_m = r_{fe} + j \cdot x_m \tag{5.17}$$

admittance matrix

This is the transformer circuit admittance matrix. This matrix is the one that if represented as an sparse matrix can be intermediately summed to the complete circuit admittance matrix.

$$\mathbf{Y}_{transformer} = \begin{bmatrix} \frac{Y_l + Y_m}{tap_c^2} & -\frac{Y_l + Y_m}{tap_c^*} \\ \frac{-Y_l}{tap_c} & Y_l \end{bmatrix} \tag{5.18}$$

$$Y_l = 1/Z_l \quad Y_m = 1/Z_m \tag{5.19}$$

$$tap_c = tap * cos(phase\_shift) + j * tap * sin(phase\_shift) \tag{5.20}$$

The $phase\_shift$ is the difference in the voltage and current angle between the primary ans the secondary. This value depends on the transformer connection type.

The tap is the transformer voltage regulator tap value. Is is a value around one, indicating the per unit voltage regulation in the secondary.

Currents calculation

After the circuit voltages are calculated with any of the solvers provided, the individual branch elements currents are calculated with the following equation (Actually, the Ohm law)

$$\mathbf{I}_{line} = \begin{bmatrix} Y_{kk} & Y_{km} \\ Y_{mk} & Y_{mm} \end{bmatrix} \begin{bmatrix} V_k \\ V_m \end{bmatrix} \tag{5.21}$$

6. Algorithms

In this chapter I describe the logic behind the code. The algorithms are in some cased modified by myself, but in any case they come from the cited sources.

6.1 Formation of the circuit admittance matrix

I'll describe how to compose the circuit admittance matrix from the individual element admittance matrices.

I assume that the circuit elements can provide its own admittance matrix. For more information see the posts of the line model or the transformer model.

The admittance matrix of the circuit includes the admittance of all those elements that are connected to at least one bus and at the most to "N" buses. For instance; a line is connected to two buses, a shunt reactance is connected to one bus and a tree-winding transformer is connected to three buses.

So, imagine a 4-bus circuit with two lines, one transformer and one shunt element like the one in the figure6.1.

The circuit elements that are not buses will have their own admittance matrices of dimension equal to the number of buses that they are connected to. This is, if an element is connected to 2 buses, then its admittance matrix will be of 2 by 2. Again, look at the individual element posts to see the elements own admittance matrices formulation.

I'll continue with this example since it is way easier to understand if I explain this with an example. I begin with an empty circuit admittance matrix of N by N buses dimension, in this case $N = 4$ so the circuit admittance matrix is of 4 x 4

$$\mathbf{Y}_{circuit} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The individual circuit elements have their own smaller admittance matrices. Expanding those small matrices to match the circuit admittance matrix, we can directly sum the expanded matrices to compose the circuit admittance matrix.

The way to expand the individual matrices is to match the circuit admittance matrix is:

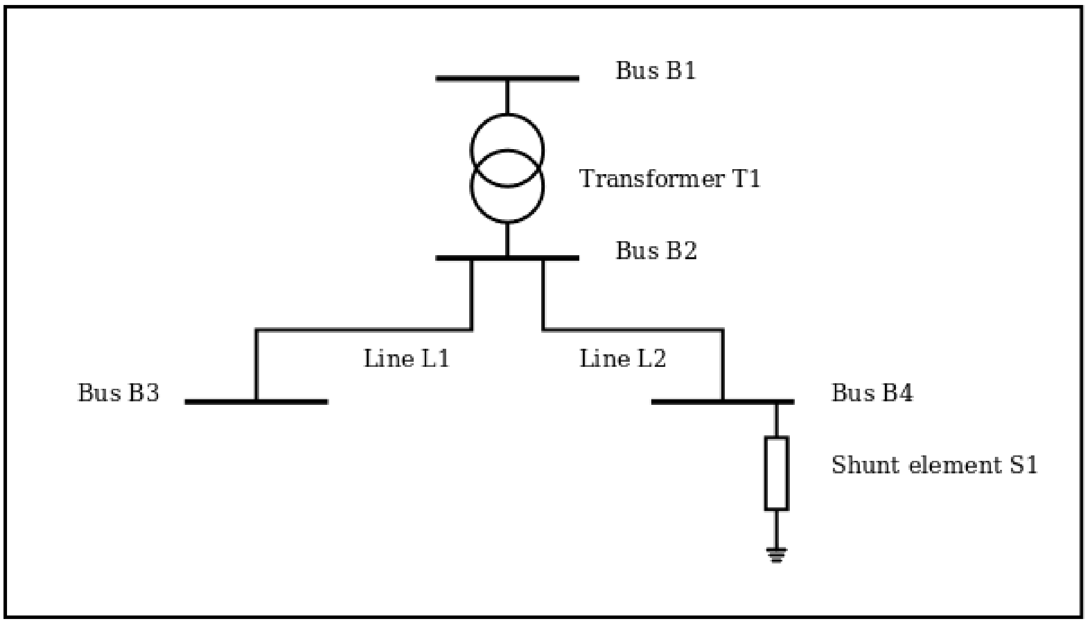

Figure 6.1: Example 4 bus circuit

Number the circuit bus bars like I did in the picture. Bus1, Bus2, Bus3, etc. Each element must know the indices of the connection bus bars in the circuit, for example the connection indices of the line L2 are 2 and 4. Then assimilate the element terminals to the circuit terminals, for example the terminal 1 of the line L2 is the terminal 2 in the circuit and the terminal 2 of the line L2 is the terminal 4 in the circuit. Knowing this, the expansion of the matrices is pretty straight forward.

The admittance matrix of the transformer T1, expanded to match the circuit dimension is:

$$\mathbf{Y}_{T1,expanded} = \begin{pmatrix} Y_{T1,11} & Y_{T1,12} & 0 & 0 \\ Y_{T1,21} & Y_{T1,22} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The admittance matrix of the line L1, expanded to match the circuit dimension is:

$$\mathbf{Y}_{L1,expanded} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & Y_{L1,11} & Y_{L1,12} & 0 \\ 0 & Y_{L1,12} & Y_{L1,22} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The admittance matrix of the line L2, expanded to match the circuit dimension is:

$$\mathbf{Y}_{L2,expanded} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & Y_{L2,11} & 0 & Y_{L2,12} \\ 0 & 0 & 0 & 0 \\ 0 & Y_{L2,12} & 0 & Y_{L2,22} \end{pmatrix}$$

The admittance matrix of the shunt element S1, expanded to match the circuit dimension is:

$$\mathbf{Y}_{S1,expanded} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & Y_{S1,11} \end{pmatrix}$$

Finally the circuit admittance matrix is the sum of all the individual matrices:

$$\mathbf{Y}_{circuit} = \mathbf{Y}_{T1,expanded} + \mathbf{Y}_{L1,expanded} + \mathbf{Y}_{L2,expanded} + \mathbf{Y}_{S1,expanded} = \begin{pmatrix} Y_{T1,11} & Y_{T1,12} & 0 & 0 \\ Y_{T1,21} & Y_{T1,22} + Y_{L1,11} + Y_{L2,11} & Y_{L1,12} & Y_{L2,12} \\ 0 & Y_{L1,12} & Y_{L1,22} & 0 \\ 0 & Y_{L2,12} & 0 & Y_{L2,22} + Y_{S1,11} \end{pmatrix}$$

Putting the method in a more general form, we can say that the circuit admittance matrix is the sum of the expanded admittance matrices of the branch elements that compose the circuit. Being the branch elements those elements in the circuit connected to at least one bus bar. The general formula can be:

$$\mathbf{Y}_{circuit} = \sum_{branch \in Branchs} \mathbf{Y}_{branch} \tag{6.1}$$

In a real life program, it is highly inefficient to use full matrices to do this since most of the memory will be occupied by zeros. Therefore it is best to use sparse matrices to represent the expanded element admittance matrices and of course the circuit admittance matrix. (Do the described process a couple of times manually with small circuits and you will get the idea...)

In the past I have implemented this method by direct composition of the admittance matrix element by element, knowing the position of the individual elements of the smaller matrices in the bigger circuit matrix. Certainly it is more efficient than doing the described procedure using full matrices. But I find this matrix composition of the admittance matrix quite tidy and easy to implement. It can be also very efficient if sparse matrices are used.

6.2 Formation of the reduced impedance matrix

Given the circuit admittance matrix $\mathbf{Y}_{circuit}$, the obtaining of the reduced system impedance matrix is fairly simple.

- Make a clone of the $\mathbf{Y}_{clone} = \mathbf{Y}_{circuit}$ matrix.
- Remove the row and the column of $\mathbf{Y}_{clone}$ corresponding to the index of the slack bus bar. This results in the matrix $\mathbf{Y}_{circuit,reduced}$.
- Perform the inverse of $\mathbf{Y}_{circuit,reduced}$. An LU method is preferred to perform the inverse. The result is the circuit $Z_{reduced}$ matrix.

6.3 Z-Matrix Gauss-Seidel

This algorithm is implemented in the file `Solver_ZBusGS.cpp`.

The Zbus Gauss Seidel algorithm is similar to the Gauss Seidel algorithm, the main difference is the use of the $\mathbf{Z}_{reduced}$ matrix instead of the $\mathbf{Y}_{circuit}$ matrix. Since the $\mathbf{Z}_{reduced}$ is full and not very sparse like the $\mathbf{Y}_{circuit}$, the algorithm converges better.

Convergence

The convergence criteria in this algorithm is less accurate for the final solution than the one used in the Newton-Raphson Algorithm. The criteria consists in checking the buses voltage convergence, this is, comparing the voltage in the current iteration (vector `E` in the code) with the voltage in the previous iteration (vector `E0` in the code). So if the difference of the absolute values of the voltages `E[k] - E0[k]` are less of equal than a given threshold ε the voltage in the bus bar k has converged.

Algorithm

```
1. Prepare a vector that contains the indices of the PQ buses. The vector in the code is BUSES.
2. set the variable has_converged=false.
3. while has_converged=false AND Iterations < Max_Iter:
   (a) has_converged=true
   (b) for every bus index $k$ in BUSES:
       i. Calculate the bus $k$ new voltage.
       ii. Calculate the bus $k$ new current.
       iii. if the bus $k$ voltage did not converged, set has_converged=false.
   (c) update the vector E0 with the vector E.
4. Update the final solution voltages with the values of the vector E.
5. Calculate the slack bus power.
```

6.4 Newton-Raphson

This algorithm is implemented in the file `Solver_NRpolar.cpp`.

The Newton-Rapson algorithm consists in solving the linear system of equations 6.2 iteratively until convergence. The vector $\begin{bmatrix}\Delta\theta \\ \Delta V/V\end{bmatrix}$ constitutes the solution at each step, this is, the voltage module (divided by the voltage; this gives more stability to the process) for the PQ buses and the voltage angle for the PQ and PV buses.

$$\begin{bmatrix}J1 & J2 \\ J3 & J4\end{bmatrix}\begin{bmatrix}\Delta\theta \\ \Delta V/V\end{bmatrix} = \begin{bmatrix}\Delta P \\ \Delta Q\end{bmatrix} \tag{6.2}$$

$$\Delta P_i = P_i^{esp} - P_i \tag{6.3}$$

$$\Delta Q_i = Q_i^{esp} - Q_i \tag{6.4}$$

$$P[i] = V[i]\sum_{j=0}^{n-1} G[i,j]cos(\theta[i,j]) + B[i,j]sin(\theta[i,j]) \tag{6.5}$$

$$Q[i] = V[i]\sum_{j=0}^{n-1} G[i,j]sin(\theta[i,j]) - B[i,j]cos(\theta[i,j]) \tag{6.6}$$

Where:

- P_i^{esp} is the active power of the buses at the beginning of the process. (The one given by the users)
- Q_i^{esp} is the reactive power of the buses at the beginning of the process. (The one given by the users)
- $\theta[i,j]$ is the difference of the voltage angle of the buses i and j ($\theta[i,j] = \theta[i] - \theta[j]$)

Solution update

The solution of each linear system (Eq. 6.2) gives the amount to add to the current solution in order to achieve the final solution. The voltage modules and angles must be updated according to the following formulas for the given formulation in this chapter:

$$V_i^{(new} = V_i^{(old} * (1 + \Delta V_i) \tag{6.7}$$

$$\theta_i^{(new} = \theta_i^{(old} + \Delta\theta_i \tag{6.8}$$

Convergence

This convergence criteria used in the Newton-Raphson algorithm is equivalent to check the equality given by the fundamental equation 4.1, therefore it is the "purest" convergence criteria.

It boils down to check if all the elements in the vector $\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix}$ are smaller that a given threshold ε.

Algorithm

1. calculate $\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix}$
2. Check convergence.
3. If it did not converge:
   (a) Calculate the Jacobian.
   (b) Solve the system obtaining $\begin{bmatrix}\Delta\theta\\ \Delta V/V\end{bmatrix}$
   (c) Update the solution
   (d) Calculate Q_i for the PV buses.
   (e) calculate $\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix}$
   (f) Check convergence.
   (g) Go to 3.

6.4.1 Jacobian in polar form

The Jacobian matrix is the derivative of the circuit equations with respect to the voltage variables. It is needed to compute the Ïirectionöf the solution in each iteration.

$$J = \begin{bmatrix} J1 & J2\\ J3 & J4\end{bmatrix} = \begin{bmatrix} \frac{\delta P_i}{\delta\theta_i} & V_j\frac{\delta P_i}{\delta V_i}\\ \frac{\delta Q_i}{\delta\theta_i} & V_j\frac{\delta Q_i}{\delta V_i}\end{bmatrix} \tag{6.9}$$

The Jacobian formulation for Newton-Raphson in polar coordinates is:

Defining i as the index of the rows and j as the index of the columns, the formulas resulting from the derivatives for each subpart of the Jacobian are:

J1

- Number of rows = number of PQ nodes + number of PV nodes.
- Number of columns = number of PQ nodes + number of PV nodes.

$$J1[i,i] = -Q[i] - B[i,i]V[i]^2 \tag{6.10}$$

$$J1[i,j] = V[i]V[j](G[i,j]sin(\theta[i,j]) - B[i,j]cos(\theta[i,j])) \tag{6.11}$$

J2

- Number of rows = number of PQ nodes + number of PV nodes.
- Number of columns = number of PQ nodes.

$$J2[i,i] = P[i] + G[i,i]V[i]^2 \tag{6.12}$$

$$J2[i,j] = V[i]V[j](G[i,j]cos(\theta[i,j]) + B[i,j]sin(\theta[i,j])) \tag{6.13}$$

J3

- Number of rows = number of PQ nodes.
- Number of columns = number of PQ nodes + number of PV nodes.

$$J3[i,i] = P[i] - G[i,i]V[i]^2 \tag{6.14}$$

$$J3[i,j] = -V[i]V[j](G[i,j]cos(\theta[i,j]) + B[i,j]sin(\theta[i,j])) \tag{6.15}$$

J4

- Number of rows = number of PQ nodes.
- Number of columns = number of PQ nodes.

$$J4[i,i] = Q[i] - B[i,i]V[i]^2 \tag{6.16}$$

$$J4[i,j] = V[i]V[j](G[i,j]sin(\theta[i,j]) - B[i,j]cos(\theta[i,j])) \tag{6.17}$$

Where:

- B is the real part of the admittance matrix ($B_{ij} = real(Y_{ij})$)
- G is the imaginary part of the admittance matrix ($G_{ij} = imag(Y_{ij})$)

Bibliography