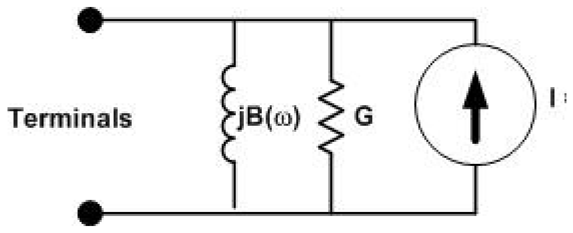
Key Variables and Functions for Writing PC Elements Models in OpenDSS

23 July 2015

Power Conversion (PC) Elements

PC Elements are the nonlinear elements in OpenDSS circuit models. They convert power one form to another while Power Delivery (PD) elements basically deliver the power from one PC Element to others. PD Elements are linear.

PC elements are the ones that change during the power flow solution process. They are generally represented by a Norton equivalent:



The Norton admittance branch, G + jB, is used to construct the primitive Y matrix, **Yprim**, for the element and then is incorporated in the **System Y** matrix. So the main task of the PC element model code is to compute the **injection current**, I, given the voltages appearing at the terminals. In many of the existing PC element models the desired terminal current, **Iterminal**, is the most straightforward to compute. Then the Yprim contribution is subtracted from Iterminal, leaving the injection current. The injection current, called **InjCurrent**, is then added into the system **Currents** array. Solving the system Y matrix equations populates the system voltage array, **NodeV**. NodeV is defined with a 0-based index in which the 0-th element is never used in the solution process. NodeV^[0] always equals 0 + j0.

Inherited from CktElement Class

Yorder – long integer

The number of terminals * number of conductors per terminal.

ComplexBuffer (protected) pComplexArray

This is a pointer to a buffer (array) of complex numbers of sufficient size to handle one complex number for each conductor in the device, such as the currents in each terminal. It is allocated with the element is created and re-allocated if the size changes to hold ***Yorder*** elements.

It is a protected variable, so only objects derived from the element can access this variable. Other arrays are public and could be referenced by other objects in the active circuit.

Vterminal – pComplexArray

This is a pointer to a complex array is intended to hold the Voltages at each terminal for various local calculations. It is populated by a call to the **ComputeVterminal** procedure. (In some model implementations, it is populated by custom code.) Its dimension is **Yorder**.

Iterminal – pComplexArray

This is a pointer to a complex array is intended to hold the Currents at each terminal for various local calculations. It is populated by a call to the **ComputeIterminal** procedure. (In some model implementations, it is populated by custom code.) Its dimension is **Yorder**.

ComputeVterminal – (Procedure)

This procedure populates the local **Vterminal** array from the present system voltage array. This procedure is not a virtual procedure and does the same for all circuit element classes.

ComputeIterminal – (Virtual Procedure)

This is the base class procedure for populating the local **Iterminal** array with the present values of the . This procedure is a virtual procedure, overridden in the PCElement class, although both currently do the same thing – they call the **GetCurrents** function for the circuit element and stick the result in the Iterminal array.

InjCurrents – (Virtual Function)

This is the base class function that is called from the Solution module for populating the system **Currents** array. The base class function should never be called and will give an error message. It is overridden by any element that uses it.

Inherited from PCElement Class

InjCurrent – pComplexArray

This is a pointer to a complex array is intended to hold the injection currents at each terminal for various local calculations. It is populated by a call to the **GetInjCurrents** procedure in the implementation of the object. Its dimension is **Yorder**. The values from this array are pumped directly into the system Currents array by the **InjCurrents** function (below), which is called from the **GetPCInjCurr** procedure in the Solution object.

InjCurrents – (Virtual Function Override)

This function is called from the Solution object **GetPCInjCurr** procedure for populating the system injection currents array. It uses the **NodeRef** index array for the specific element to put the PC Element's injection currents directly into the proper location in the system **Currents** array.

CalcYPrimContribution – (Procedure)

This procedure is called from the PC element to compute the current in the Norton equivalent admittance that will automatically be included in the solution of the System Y equation. It first populates **Vterminal** and then multiplies the **Yprim** matrix times the Vterminal vector. The next step in the process for a PC element is to compute the difference between this current vector and the desired terminal current vector, resulting in the values for the **InjCurrent** array.

```
PROCEDURE TPCElement.CalcYPrimContribution(Curr: pComplexArray);

begin
```

```
        ComputeVTerminal;
        // Apply these voltages to Yprim
        YPrim.MVMult(Curr, Vterminal);
    end;
```

Implemented in the PC Element Instance

GetInjCurrents – (Virtual Procedure Override)

This Procedure is implemented and called only from the PCElement object itself. It is actually defined in the CktElement class but only the implementation in the object instance should be called. A typical usage is as in the Vsource object:

```
Function TVsourceObj.InjCurrents:Integer;

Begin

   GetInjCurrents(InjCurrent);

{This is source injection}

   Result := Inherited InjCurrents; // Add into system array

End;
```

Here it is called in the InjCurrents override for the Vsource object to compute the local **InjCurrent** array. Then the inherited InjCurrents function (from PCElement class) is invoked to put the injection currents directly into the system Currents array.

The injection currents are the difference between the desired terminal current and the currents represented by the Norton equivalent admittances in the System Y matrix, which is derived from the Yprim matrix for each instance of the element. A typical sequence is

```
    CalcYPrimContribution(InjCurrent);  // Init InjCurrent Array

    (compute iTerminal)

    InjCurrent^[i] := Csub( Iterminal^[i],InjCurrent^[i]);  // diff
```

The signs in the above code may vary depending on how the programmer defines the direction of the currents for a specific object. For an example see the **StickCurrInTerminalArray** procedure in either the Generator, Load, or PVSystem model.