

Load Characteristic INDMOT1P

Load Characteristic INDMOT1P Dynamic Phasor Model of a single-phase induction motor load model

This model is taken from the paper Phasor Modeling Approach for Single Phase A/C Motors by B. Lesieutre, M. Kosterev, and J. Undrill published in the 2008 IEEE Power and Energy Society General Meeting – Conversion and Delivery of Electrical Energy in the 21st Century. The following is the resulting model from that paper which has been rewritten to remove complex numbers. See a later section for the derivation between the models.

As a user of PowerWorld Simulator you only need the first two pages of this documentation, but the remainder of the pages make it explicit how this model is implemented in software in PowerWorld Simulator. Funding for the complete public documentation of this model was provided by Lawrence Berkely National Laboratory under Subcontract 7578510 in March 2021. This model assumes that there are identical single-phase induction motors on each phase of a three-phase balanced power system model. It is meant as an aggregate model representing many single-phase induction motors applied equally across all three phases.

Parameter	Parameter Description
LF	Load Factor. All values are on an $MVABaseUsed = \frac{MW_{init}}{LF}$
$MBase$	If $MBase > 0$ then machine parameters are on this $MVABaseUsed$ is equation to this, otherwise the $MVABaseUsed$ is MW_{init}/LF
T_{nom}	Nominal load torque used for motor starting studies. For online motors this is automatically calculated from initial conditions.
R_{ds}	stator d-axis resistance in per unit
X'_{ds}	stator d-axis reactance in per unit
R_{qs}	stator q-axis resistance in per unit
X'_{qs}	stator q-axis reactance in per unit
X_c	capacitor reactance in per unit (will be negative)
X_m	magnetizing reactance in per unit
n	ratio of stator auxiliary winding turns to stator main winding turns
H	inertia constant for the motor
R_r	rotor resistance in per unit that is to calculate $T'_o = \frac{X_r}{\omega_p R_r}$
X_r	rotor reactance in per unit
E_1	These 4 parameters are used for defining the saturation function instead of enter A_{sat} and B_{sat} directly. This is consistent with all other models that use saturation
E_2	
S_{E1}	
S_{E2}	
	$Sat = \begin{cases} 1 & \text{for } \psi \leq B_{sat} \\ 1 + A_{sat}(\psi - B_{sat})^2 & \text{for } \psi > B_{sat} \end{cases}$
A	Coefficient for the quadratic term of mechanical torque equation
B	Coefficient for the linear term of mechanical torque equation
C_0	Coefficient for the constant term of mechanical torque equation
D	Coefficient for the exponential term of mechanical torque equation
E	Exponent of the exponential term of the mechanical torque equation
B_{pgas}	Coefficient for the linear delayed speed term of mechanical torque equation
T_{pgas}	Time constant for delayed speed calculation. Delayed speed represents the gas pressure in a compressor motor. [B_{pgas} and T_{pgas} added in June 15, 2023 patch of PowerWorld Simulator 23]
	$T_{mech} = T_{nom} \left[A\omega_r^2 + B\omega_r + C_0 + D\omega_r^E + B_{pgas} \left(\frac{\omega_r}{1 + sT_{pgas}} \right) \right]$

1 Stator and Rotor Equations

The network boundary equations for this model are as follows in a reference frame with a forward and backward rotating fields.

- $|V_s| = R_{ds}(I_{fR} + I_{bR}) - X'_{ds}(I_{fI} + I_{bI}) - (1 + \omega_s) \frac{X_m}{X_r} (\psi_{fI} + \psi_{bI})$
- $0 = R_{ds}(I_{fI} + I_{bI}) + X'_{ds}(I_{fR} + I_{bR}) + (1 + \omega_s) \frac{X_m}{X_r} (\psi_{fR} + \psi_{bR})$
- $|V_s| = R_{qs} \frac{1}{n} (-I_{fI} + I_{bI}) - (X'_{qs} + X_c) \frac{1}{n} (I_{fR} - I_{bR}) - (1 + \omega_s) \frac{nX_m}{X_r} (\psi_{fR} - \psi_{bR})$
- $0 = R_{qs} \frac{1}{n} (I_{fR} - I_{bR}) + (X'_{qs} + X_c) \frac{1}{n} (-I_{fI} + I_{bI}) + (1 + \omega_s) \frac{nX_m}{X_r} (-\psi_{fI} + \psi_{bI})$

The dynamic equations are as follows. The 3rd and 4th dynamic equations are equations for the flux dynamics of the backward rotating fields that are fast and thus approximated as algebraic relationships.

- $T'_o \frac{d(\psi_{fR})}{dt} = X_m I_{fR} - \psi_{fR} Sat + T'_o \omega_b (\omega_s - \omega_r) \psi_{fI}$
- $T'_o \frac{d(\psi_{fI})}{dt} = X_m I_{fI} - \psi_{fI} Sat - T'_o \omega_b (\omega_s - \omega_r) \psi_{fR}$
- $T'_o \frac{d(\psi_{bR})}{dt} = 0 = X_m I_{bR} - \psi_{bR} Sat + T'_o \omega_b (\omega_s + \omega_r) \psi_{bI}$
- $T'_o \frac{d(\psi_{bI})}{dt} = 0 = X_m I_{bI} - \psi_{bI} Sat - T'_o \omega_b (\omega_s + \omega_r) \psi_{bR}$
- $2H \frac{d\omega_r}{dt} = 2 \frac{X_m}{X_r} (I_{fI} \psi_{fR} - I_{fR} \psi_{fI} - I_{bI} \psi_{bR} + I_{bR} \psi_{bI}) - T_{mech}$
- $\frac{d\omega_{rdelayed}}{dt} = \frac{\omega_r - \omega_{rdelayed}}{T_{pgas}}$ // added on June 15, 2023

Where the saturation function and T'_o are defined as

- $Sat = Sat(\psi_f, \psi_b) = \begin{cases} 1 & \text{for } |\psi| \leq B_{sat} \\ 1 + A_{sat} (|\psi| - B_{sat})^2 & \text{for } |\psi| > B_{sat} \end{cases}$
- $|\psi| = \sqrt{\psi_{fR}^2 + \psi_{fI}^2 + \psi_{bR}^2 + \psi_{bI}^2}$
- $T'_o = \frac{1}{\omega_b} \frac{X_r}{R_r}$
- $T_{elec} = 2 \frac{X_m}{X_r} (I_{fI} \psi_{fR} - I_{fR} \psi_{fI} - I_{bI} \psi_{bR} + I_{bR} \psi_{bI})$

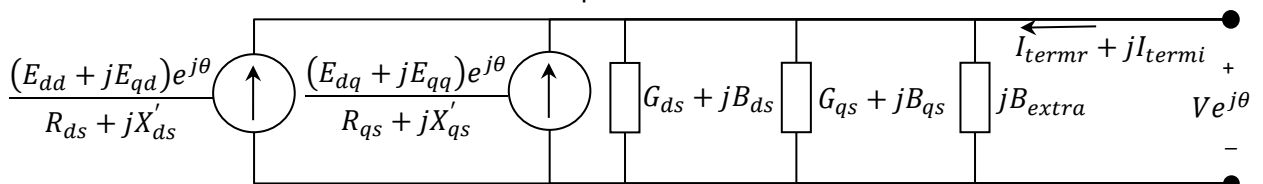
The network interface equations on the dq reference frame are as follows.

- $|V_s| = [R_{ds} + jX'_{ds}](I_{dsR} + jI_{dsI}) + j(\omega_s) \frac{X_m}{X_r} (\psi_{drR} + j\psi_{drI})$
- $|V_s| = [R_{qs} + j(X'_{qs} + X_c)](I_{qsR} + jI_{qsI}) + j(\omega_s) \frac{nX_m}{X_r} (\psi_{qrR} + j\psi_{qrI})$

These result in the following network boundary equations on the forward and backward rotating reference frames which are similar to 2 parallel REGC_B type network interfaces.

$$\begin{aligned} E_{dd} &= -(\psi_{fI} + \psi_{bI}) (\omega_s) \frac{X_m}{X_r} & E_{dq} &= -(\psi_{fR} - \psi_{bR}) (\omega_s) \frac{nX_m}{X_r} & G_{ds} &= \frac{+R_{ds}}{R_{ds}^2 + X_{ds}'^2} & G_{qs} &= \frac{+R_{qs}}{R_{qs}^2 + X_{qs}'^2} \\ E_{qd} &= +(\psi_{fR} + \psi_{bR}) (\omega_s) \frac{X_m}{X_r} & E_{qq} &= +(-\psi_{fI} + \psi_{bI}) (\omega_s) \frac{nX_m}{X_r} & B_{ds} &= \frac{-X_{ds}'}{R_{ds}^2 + (X_{ds}' + X_c)^2} & B_{qs} &= \frac{-(X_{qs}' + X_c)}{R_{qs}^2 + (X_{qs}' + X_c)^2} \end{aligned}$$

This then leads to the following network interface which will be described later in this document, where B_{extra} represents the extra Mvars due to the mismatch between the initial condition Mvars and the Mvars calculated from the induction machine equations.



2 Calculation of Saturation Function from E1, E2, SE1, and SE2

Consider the quadratic function which means that $S_{E1} = A_{sat}(E_1 - B_{sat})^2$ and $S_{E2} = A_{sat}(E_2 - B_{sat})^2$. From the first equation $A_{sat} = \frac{S_{E1}}{(E_1 - B_{sat})^2}$ which can be substituted into the second equation to give

$S_{E2} = \frac{S_{E1}}{(E_1 - B_{sat})^2} (E_2 - B_{sat})^2$. Now group terms to get $S_{E2}(E_1 - B_{sat})^2 = S_{E1}(E_2 - B_{sat})^2$. Finally, write in the terms of a quadratic equation and solve for B.

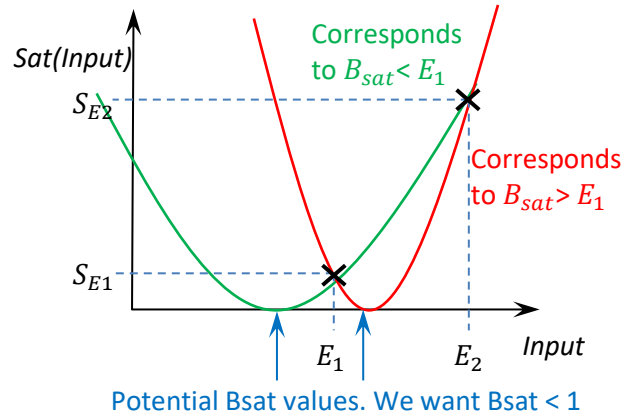
$$(S_{E2}E_1^2 - S_{E1}E_2^2) + (-2S_{E2}E_1 + 2S_{E1}E_2)B_{sat} + (S_{E2}^2 - S_{E1}^2)B_{sat}^2 = 0$$

Define the variables for the quadratic equation $c = (S_{E2}E_1^2 - S_{E1}E_2^2)$, $b = (-2S_{E2}E_1 + 2S_{E1}E_2)$, and $a = (S_{E2}^2 - S_{E1}^2)$

Then solve for $B_{sat} = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$. Note: we want the negative sign. To understand why, see the figure to the right which shows this value will give us the monotonically increasing saturation function. Once we have

B_{sat} then we calculate $A_{sat} = \frac{S_{E1}}{(E_1 - B_{sat})^2}$.

Once you have the A_{sat} and B_{sat} values simply use them in the following saturation function which assumes inputs greater than B_{sat} have a saturation function result of zero.



```
Function SaturationQuadratic(Input : double) : double;
{
  If Input < Bsat then result = 0
  Else result = Asat*sqr(Input-Bsat)
}

procedure SaturationCalculateValues(E1, SE1, E2, SE2 : Double);
Var A, B, C : Double
    Arg : double;
{
  if ( (SE1 <= 0) and (SE2 <= 0) ) // both zero or negative, then ignore
    or
    (abs(SE1-SE2) < 1E-6) // saturation values that are equal is a problem
  then begin
    SatB := 0;
    SatA := 0;
  end
  else begin
    if (SE1 <= 0) then begin
      SatB := E1;
    end
    else begin
      A := SE2 - SE1;
      B := - 2*SE2*E1 + 2*SE1*E2;
      C := + SE2*sqr(E1) - SE1*sqr(E2);
      Arg := sqr(B) - 3*A*C;
      SatB := (- B - sqrt(sqr(B) - 3*A*C)) / (2*A);
    end;
    SatA := SE1/Sqr(E1-SatB);
  end;
}
```

3 Initialization

Initialization of this model involves solving a set of 10 equations. Equations 1 – 4 are the network boundary equations given the initial voltage magnitude. Equations 5 – 8 are the dynamic equations including the flux fields and equation 9 is the power balance at the network boundary equation. Finally, Equation 10 represents the equation for saturation. One caveat when coding is if flux value $|\psi| < B_{sat}$ then meaning that the saturation is function $Sat = 1$, and all derivatives of saturation become 0.0. At steady state we know that $\omega_s = 1$, which leaves 10 unknown variables which are shown in red text to the right equations below. At initialization, $\omega_{rdelayed} = \omega_r$, so this is omitted in the discussion below.

1. $0 = R_{ds}(I_{fI} + I_{bI}) + X'_{ds}(I_{fR} + I_{bR}) + \frac{X_m}{X_r}(\psi_{fR} + \psi_{bR})$ ψ_{fR}
2. $0 = -|V_s| + R_{ds}(I_{fR} + I_{bR}) - X'_{ds}(I_{fI} + I_{bI}) - \frac{X_m}{X_r}(\psi_{fI} + \psi_{bI})$ ψ_{fI}
3. $0 = -|V_s| + R_{qs}\frac{1}{n}(-I_{fI} + I_{bI}) - (X'_{qs} + X_c)\frac{1}{n}(I_{fR} - I_{bR}) - \frac{nX_m}{X_r}(\psi_{fR} - \psi_{bR})$ ψ_{bR}
4. $0 = R_{qs}\frac{1}{n}(I_{fR} - I_{bR}) + (X'_{qs} + X_c)\frac{1}{n}(-I_{fI} + I_{bI}) + \frac{nX_m}{X_r}(-\psi_{fI} + \psi_{bI})$ ψ_{bI}
5. $0 = X_m I_{fR} - \psi_{fR} Sat + T'_o \omega_b (\omega_s - \omega_r) \psi_{fI}$ I_{fR}
6. $0 = X_m I_{fI} - \psi_{fI} Sat - T'_o \omega_b (\omega_s - \omega_r) \psi_{fR}$ I_{fI}
7. $0 = X_m I_{bR} - \psi_{bR} Sat + T'_o \omega_b (\omega_s + \omega_r) \psi_{bI}$ I_{bR}
8. $0 = X_m I_{bI} - \psi_{bI} Sat - T'_o \omega_b (\omega_s + \omega_r) \psi_{bR}$ I_{bI}
9. $0 = I_{fR} + I_{bR} + \frac{1}{n}(-I_{fI} + I_{bI}) - \frac{P_{init}}{|V|}$ ω_r
10. $0 = -Sat + 1 + A_{sat} \left(\sqrt{\psi_{fR}^2 + \psi_{fI}^2 + \psi_{bR}^2 + \psi_{bI}^2} - B_{sat} \right)^2$ Sat

This is a non-linear set of equations which will need to be solved using Newton's method. To aid in their calculation, the following is the Jacobian matrix which will be used in this solution. (For saturation function derivatives which all multiplied by $\left(1 - \frac{B_{sat}}{\psi}\right)$. If $|\psi| < B_{sat}$, then replace $\left(1 - \frac{B_{sat}}{\psi}\right) = 0$.

	ψ_{fR}	ψ_{fI}	ψ_{bR}	ψ_{bI}	I_{fR}	I_{fI}	I_{bR}	I_{bI}	ω_r	Sat
1	$+\frac{X_m}{X_r}$	0	$+\frac{X_m}{X_r}$	0	$+X'_{ds}$	$+R_{ds}$	$+X'_{ds}$	$+R_{ds}$	0	0
2	0	$-\frac{X_m}{X_r}$	0	$-\frac{X_m}{X_r}$	$+R_{ds}$	$-X'_{ds}$	$+R_{ds}$	$-X'_{ds}$	0	0
3	$-\frac{X_m}{X_r}$	0	$+\frac{X_m}{X_r}$	0	$-\frac{X'_{qs} + X_c}{n}$	$-\frac{R_{qs}}{n}$	$+\frac{X'_{qs} + X_c}{n}$	$+\frac{R_{qs}}{n}$	0	0
4	0	$-\frac{X_m}{X_r}$	0	$+\frac{X_m}{X_r}$	$+\frac{R_{qs}}{n}$	$-\frac{X'_{qs} + X_c}{n}$	$-\frac{R_{qs}}{n}$	$+\frac{X'_{qs} + X_c}{n}$	0	0
5	$-Sat$	$+T'_o \omega_b * (\omega_s - \omega_r)$	0	0	$+X_m$	0	0	0	$-T'_o \omega_b \psi_{fI}$	$-\psi_{fR}$
6	$-T'_o \omega_b * (\omega_s - \omega_r)$	$-Sat$	0	0	0	$+X_m$	0	0	$+T'_o \omega_b \psi_{fR}$	$-\psi_{fI}$
7	0	0	$-Sat$	$+T'_o \omega_b * (\omega_s + \omega_r)$	0	0	$+X_m$	0	$+T'_o \omega_b \psi_{bI}$	$-\psi_{bR}$
8	0	0	$-T'_o \omega_b * (\omega_s + \omega_r)$	$-Sat$	0	0	0	$+X_m$	$-T'_o \omega_b \psi_{bR}$	$-\psi_{bI}$
9	0	0	0	0	+1	$-\frac{1}{n}$	+1	$+\frac{1}{n}$	0	0
10	$\frac{2A_{sat}\psi_{fR}}{\left(1 - \frac{B_{sat}}{\psi}\right)}$	$\frac{2A_{sat}\psi_{fI}}{\left(1 - \frac{B_{sat}}{\psi}\right)}$	$\frac{2A_{sat}\psi_{bR}}{\left(1 - \frac{B_{sat}}{\psi}\right)}$	$\frac{2A_{sat}\psi_{bI}}{\left(1 - \frac{B_{sat}}{\psi}\right)}$	0	0	0	0	0	-1

3.1.1 Initial Guess for Initialization Newton's Method

In order to get an initial guess for Newton's method used in calculating an initial condition, we will make the following assumptions: $Sat = 1$, $\omega_s = 1.000$, $\omega_r = 0.99$. We are just assuming no saturation to start and a very small slip of 0.01. With these values assumed, the remainder of the equations become completely linear and an initial guess can be solved directly.

- $0 = R_{ds}(I_{fI} + I_{bI}) + X'_{ds}(I_{fR} + I_{bR}) + \frac{X_m}{X_r}(\psi_{fR} + \psi_{bR})$ ψ_{fR}
- $0 = -|V_s| + R_{ds}(I_{fR} + I_{bR}) - X'_{ds}(I_{fI} + I_{bI}) - \frac{X_m}{X_r}(\psi_{fI} + \psi_{bI})$ ψ_{fI}
- $0 = -|V_s| + R_{qs}\frac{1}{n}(-I_{fI} + I_{bI}) - (X'_{qs} + X_c)\frac{1}{n}(I_{fR} - I_{bR}) - n\frac{X_m}{X_r}(\psi_{fR} - \psi_{bR})$ ψ_{bR}
- $0 = R_{qs}\frac{1}{n}(I_{fR} - I_{bR}) + (X'_{qs} + X_c)\frac{1}{n}(-I_{fI} + I_{bI}) + n\frac{X_m}{X_r}(-\psi_{fI} + \psi_{bI})$ ψ_{bI}
- $0 = X_m I_{fR} - \psi_{fR} Sat + T'_o \omega_b (+\omega_s - \omega_r) \psi_{fI}$ I_{fR}
- $0 = X_m I_{fI} - \psi_{fI} Sat - T'_o \omega_b (+\omega_s - \omega_r) \psi_{fR}$ I_{fI}
- $0 = X_m I_{bR} - \psi_{bR} Sat + T'_o \omega_b (+\omega_s + \omega_r) \psi_{bI}$ I_{bR}
- $0 = I_{fR} + I_{bR} + \frac{1}{n}(-I_{fI} + I_{bI}) - \frac{P_{init}}{|V|}$ I_{bI}

Write this as a set of linear equations as follows.

$+\frac{X_m}{X_r}$	0	$+\frac{X_m}{X_r}$	0	$+X'_{ds}$	$+R_{ds}$	$+X'_{ds}$	R_{ds}	ψ_{fR}	0
0	$-\frac{X_m}{X_r}$	0	$-\frac{X_m}{X_r}$	$+R_{ds}$	$-X'_{ds}$	$+R_{ds}$	$-X'_{ds}$	ψ_{fI}	$ V_s $
$-n\frac{X_m}{X_r}$	0	$+n\frac{X_m}{X_r}$	0	$-\frac{X'_{qs} + X_c}{n}$	$-\frac{R_{qs}}{n}$	$+\frac{X'_{qs} + X_c}{n}$	$+\frac{R_{qs}}{n}$	ψ_{bR}	$ V_s $
0	$-n\frac{X_m}{X_r}$	0	$+n\frac{X_m}{X_r}$	$+\frac{R_{qs}}{n}$	$-\frac{X'_{qs} + X_c}{n}$	$-\frac{R_{qs}}{n}$	$+\frac{X'_{qs} + X_c}{n}$	ψ_{bI}	0
$-Sat$	$+T'_o \omega_b * (\omega_s - \omega_r)$	0	0	$+X_m$	0	0	0	I_{fR}	0
$-T'_o \omega_b * (\omega_s - \omega_r)$	$-Sat$	0	0	0	$+X_m$	0	0	I_{fI}	0
0	0	$-Sat$	$+T'_o \omega_b * (\omega_s + \omega_r)$	0	0	$+X_m$	0	I_{bR}	0
0	0	0	0	+1	$-\frac{1}{n}$	+1	$\frac{1}{n}$	I_{bI}	$\frac{P_{init}}{ V }$

Thus assuming $Sat = 1$, $\omega_s = 1.000$, $\omega_r = 0.99$, we can directly solve for initial guesses for all the flux and current variables in the forward and backward reference frames.

3.1.2 Handling of the Mvar Mismatch for the Induction Motor

At initial condition from the power flow solution, we have a terminal voltage ($|V_s|e^{j\theta}$) and current (I_s). The magnitude of voltage ($|V_s|$), can be used directly and the terminal current gives us a relationship for the d and q axis currents as given in the model: $I_s = (I_{dsR} + I_{qsR}) + j(I_{dsI} + I_{qsI})$.

From the initial condition we have $P_{motor} + jQ_{motor} = VI^* = V_s[(I_{dsR} + I_{qsR}) - j(I_{dsI} + I_{qsI})]$.

This means that $P_{motor} = +|V|(I_{dsR} + I_{qsR})$ and $Q_{motor} = -|V|(I_{dsI} + I_{qsI})$. Writing this in terms of forward and backward reference frame currents we get the following.

- $P_{motor} = +|V| \left[I_{fR} + I_{bR} + \frac{1}{n}(-I_{fI} + I_{bI}) \right]$
- $Q_{motor} = -|V| \left[I_{fI} + I_{bI} + \frac{1}{n}(I_{fR} - I_{bR}) \right]$

Much like the 3-phase induction motor, the power equation is enforced at the initial condition, but the second equation for $\frac{Q_{motor}}{|V|}$ is not. There will be a mismatch between the Q_{motor} and Q_{load} when initializing this induction motor. This mismatch is handled by adding in a B_{extra} shunt admittance with the motor equal to an amount so that the mismatch is balanced.

$$B_{extra} = \frac{Q_{load} - Q_{motor}}{|V|^2} = \frac{Q_{load} - |V| \left[I_{fI} + I_{bI} + \frac{1}{n}(I_{fR} - I_{bR}) \right]}{|V|^2}$$

3.1.3 Calculation of Nominal Torque from Initial Condition

We also must initialize a parameter of the mechanical torque equation because it must equal electric torque which can be written as follows.

$$T_{elec} = 2 \frac{X_m}{X_r} (I_{fI}\Psi_{fR} - I_{fR}\Psi_{fI} - I_{bI}\Psi_{bR} + I_{bR}\Psi_{bI})$$

The mechanical torque for the most generic model we will use may be written as follows.

The terms for B_{pgas} and T_{pgas} were added in June 15, 2023 patch of PowerWorld Simulator 23.

$$T_{mech} = T_{nom} \left[A\omega_r^2 + B\omega_r + C_0 + D\omega_r^E + B_{pgas} \left(\frac{\omega_r}{1 + sT_{pgas}} \right) \right]$$

Mechanical Torque must equal Electric Torque at steady state, so the ignore T_{pgas} delay and write:

$$2 \frac{X_m}{X_r} (I_{fI}\Psi_{fR} - I_{fR}\Psi_{fI} - I_{bI}\Psi_{bR} + I_{bR}\Psi_{bI}) = T_{nom} (A\omega_r^2 + B\omega_r + C_0 + D\omega_r^E + B_{pgas}\omega_r)$$

We can solve this for T_{nom} as

$$T_{nom} = \frac{2 \frac{X_m}{X_r} (I_{fI}\Psi_{fR} - I_{fR}\Psi_{fI} - I_{bI}\Psi_{bR} + I_{bR}\Psi_{bI})}{(A\omega_r^2 + B\omega_r + C_0 + D\omega_r^E + B_{pgas}\omega_r)}$$

For most of our dynamic models we will not use as complicated of a mechanical torque and may simply include on input parameter of E_{trq} . When that is the only input parameter then this mean that we are assuming $A = 0$, $B = 0$, $C_0 = 0$, $B_{pgas} = 0$, $D = 1$, and $E = E_{trq}$.

4 Derivative Calculations

The 4 derivatives equations are a function of 7 different algebraic variables: I_{fR} , I_{fI} , I_{bR} , I_{bI} , ψ_{bR} , ψ_{bI} , and Sat . These can be calculated using the following 7 algebraic equations.

1. $0 = R_{ds}(I_{fI} + I_{bI}) + X'_{ds}(I_{fR} + I_{bR}) + \frac{X_m}{X_r}(\psi_{fR} + \psi_{bR})$ I_{fR}
2. $0 = -|V_s| + R_{ds}(I_{fR} + I_{bR}) - X'_{ds}(I_{fI} + I_{bI}) - \frac{X_m}{X_r}(\psi_{fI} + \psi_{bI})$ I_{fI}
3. $0 = -|V_s| + R_{qs}\frac{1}{n}(-I_{fI} + I_{bI}) - (X'_{qs} + X_c)\frac{1}{n}(I_{fR} - I_{bR}) - \frac{nX_m}{X_r}(\psi_{fR} - \psi_{bR})$ ψ_{bR}
4. $0 = R_{qs}\frac{1}{n}(I_{fR} - I_{bR}) + (X'_{qs} + X_c)\frac{1}{n}(-I_{fI} + I_{bI}) + \frac{nX_m}{X_r}(-\psi_{fI} + \psi_{bI})$ ψ_{bI}
5. $0 = X_m I_{bR} - \psi_{bR} Sat + T'_o \omega_b (\omega_s + \omega_r) \psi_{bI}$ I_{bR}
6. $0 = X_m I_{bI} - \psi_{bI} Sat - T'_o \omega_b (\omega_s + \omega_r) \psi_{bR}$ I_{bI}
7. $0 = -Sat + 1 + A_{sat} \left(\sqrt{\psi_{fR}^2 + \psi_{fI}^2 + \psi_{bR}^2 + \psi_{bI}^2} - B_{sat} \right)^2$ Sat

If there is saturation, then this is a non-linear set of equations which will need to be solved using Newton's method (Alternatively, we could assume a fixed value of Sat based on the previous time-step, but this is not recommended). To aid in calculation, the following is the Jacobian matrix which will be used in this solution. If there is no saturation, then the final equation simply becomes $Sat = 1$ and the equations are completely linear so the solution would be direct with no numerical iteration required.

	I_{fR}	I_{fI}	ψ_{bR}	ψ_{bI}	I_{bR}	I_{bI}	Sat
1	$+X'_{ds}$	$+R_{ds}$	$+\frac{X_m}{X_r}$	0	$+X'_{ds}$	$+R_{ds}$	0
2	$+R_{ds}$	$-X'_{ds}$	0	$-\frac{X_m}{X_r}$	$+R_{ds}$	$-X'_{ds}$	0
3	$-\frac{X'_{qs} + X_c}{n}$	$-\frac{R_{qs}}{n}$	$+n\frac{X_m}{X_r}$	0	$+\frac{X'_{qs} + X_c}{n}$	$+\frac{R_{qs}}{n}$	0
4	$+\frac{R_{qs}}{n}$	$+\frac{X'_{qs} + X_c}{n}$	0	$+n\frac{X_m}{X_r}$	$-\frac{R_{qs}}{n}$	$+\frac{X'_{qs} + X_c}{n}$	0
5	0	0	$-Sat$	$+T'_o \omega_b * (\omega_s + \omega_r)$	$+X_m$	0	$-\psi_{bR}$
6	0	0	$-T'_o \omega_b * (\omega_s + \omega_r)$	$-Sat$	0	$+X_m$	$-\psi_{bI}$
7	0	0	$2A_{sat}\psi_{bR} * \left(1 - \frac{B_{sat}}{\psi}\right)$	$2A_{sat}\psi_{bI} * \left(1 - \frac{B_{sat}}{\psi}\right)$	0	0	-1

After solving for the 7 algebraic variables, this leaves 4 state equations for ψ_{fR} , ψ_{fI} , ω_r , and $\omega_{rdelayed}$.

- $\frac{d(\psi_{fR})}{dt} = \frac{X_m I_{fR} - Sat(\psi_{fR}, \psi_{bR})\psi_{fR}}{T'_o} + \omega_b (\omega_s - \omega_r) \psi_{fI}$
- $\frac{d(\psi_{fI})}{dt} = \frac{X_m I_{fI} - Sat(\psi_{fR}, \psi_{bR})\psi_{fI}}{T'_o} - \omega_b (\omega_s - \omega_r) \psi_{fR}$
- $\frac{d\omega_r}{dt} = \frac{2\frac{X_m}{X_r}(I_{fI}\psi_{fR} - I_{fR}\psi_{fI} - I_{bI}\psi_{bR} + I_{bR}\psi_{bI}) - T_{mech}}{2H}$
- $\frac{d\omega_{rdelayed}}{dt} = \frac{\omega_r - \omega_{rdelayed}}{T_{pgas}}$ // added on June 15, 2023

Where the saturation function and T'_o are defined as

- $Sat(\psi_{fR}, \psi_{bR}) = \begin{cases} 1 & \text{for } |\psi| \leq B_{sat} \\ 1 + A_{sat}(|\psi| - B_{sat})^2 & \text{for } |\psi| > B_{sat} \end{cases}$
- $|\psi| = \sqrt{\psi_{fR}^2 + \psi_{fI}^2 + \psi_{bR}^2 + \psi_{bI}^2}$ and $T'_o = \frac{1}{\omega_b} \frac{X_r}{R_r}$
- $T_{elec} = 2\frac{X_m}{X_r}(I_{fI}\psi_{fR} - I_{fR}\psi_{fI} - I_{bI}\psi_{bR} + I_{bR}\psi_{bI})$

5 Network Interface Equations

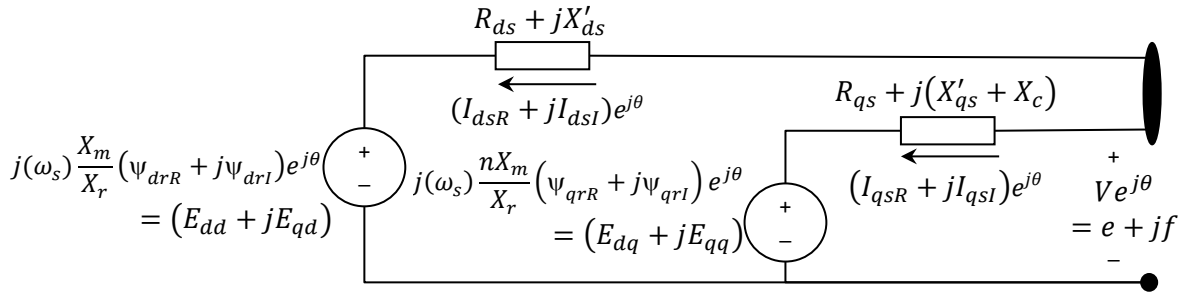
The network interface equations are as follows on the dq reference frame.

- $|V_s| = [R_{ds} + jX'_{ds}](I_{dsR} + jI_{dsl}) + j(\omega_s) \frac{X_m}{X_r} (\psi_{drR} + j\psi_{drI})$
- $|V_s| = [R_{qs} + j(X'_{qs} + X_c)](I_{qsR} + jI_{qsl}) + j(\omega_s) \frac{nX_m}{X_r} (\psi_{qrR} + j\psi_{qrI})$

We need equations with respect to the network voltage so must multiply by the phase angle at the terminal bus and thus have the following

- $|V_s|e^{j\theta} = [R_{ds} + jX'_{ds}](I_{dsR} + jI_{dsl})e^{j\theta} + j(\omega_s) \frac{X_m}{X_r} (\psi_{drR} + j\psi_{drI})e^{j\theta}$
- $|V_s|e^{j\theta} = [R_{qs} + j(X'_{qs} + X_c)](I_{qsR} + jI_{qsl})e^{j\theta} + j(\omega_s) \frac{nX_m}{X_r} (\psi_{qrR} + j\psi_{qrI})e^{j\theta}$

You can write this using the circuit equations as follows.



Note that this identical to 2 REGC_B type network interface in parallel with each other. Define

$$G_{ds} + jB_{ds} = \frac{1}{R_{ds} + jX'_{ds}} = \frac{R_{ds} - jX'_{ds}}{R_{ds}^2 + X_{ds}'^2}$$

$$G_{qs} + jB_{qs} = \frac{1}{R_{qs} + j(X'_{qs} + X_c)} = \frac{R_{qs} - j(X'_{qs} + X_c)}{R_{qs}^2 + (X'_{qs} + X_c)^2}$$

$$(E_{dd} + jE_{qd}) = j(\omega_s) \frac{X_m}{X_r} (\psi_{drR} + j\psi_{drI}) = -\psi_{drI}(\omega_s) \frac{X_m}{X_r} + j\psi_{drR}(\omega_s) \frac{X_m}{X_r}$$

$$(E_{dq} + jE_{qq}) = j(\omega_s) \frac{nX_m}{X_r} (\psi_{qrR} + j\psi_{qrI}) = -\psi_{qrI}(\omega_s) \frac{nX_m}{X_r} + j\psi_{qrR}(\omega_s) \frac{nX_m}{X_r}$$

Substitute in relationships to convert from dq to forward/backward reference frame.

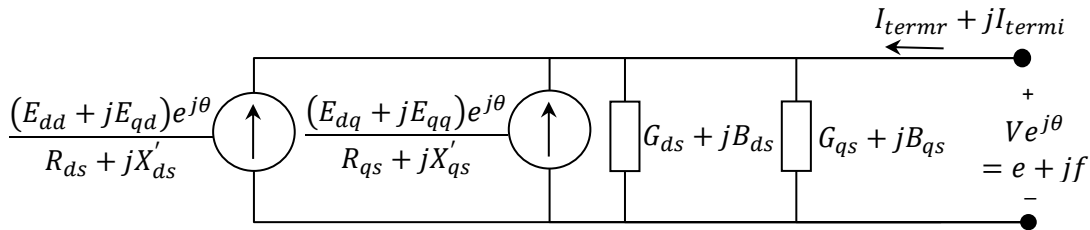
$$(E_{dd} + jE_{qd}) = -(\psi_{fI} + \psi_{bI})(\omega_s) \frac{X_m}{X_r} + j(\psi_{fR} + \psi_{bR})(\omega_s) \frac{X_m}{X_r}$$

$$(E_{dq} + jE_{qq}) = -(\psi_{fR} - \psi_{bR})(\omega_s) \frac{nX_m}{X_r} + j(-\psi_{fI} + \psi_{bI})(\omega_s) \frac{nX_m}{X_r}$$

Thus we can write the following

$$\begin{aligned} E_{dd} &= -(\psi_{fI} + \psi_{bI})(\omega_s) \frac{X_m}{X_r} & E_{dq} &= -(\psi_{fR} - \psi_{bR})(\omega_s) \frac{nX_m}{X_r} & G_{ds} &= \frac{+R_{ds}}{R_{ds}^2 + X_{ds}'^2} & G_{qs} &= \frac{+R_{qs}}{R_{qs}^2 + X_{qs}'^2} \\ E_{qd} &= +(\psi_{fR} + \psi_{bR})(\omega_s) \frac{X_m}{X_r} & E_{qq} &= +(-\psi_{fI} + \psi_{bI})(\omega_s) \frac{nX_m}{X_r} & B_{ds} &= \frac{-X'_{ds}}{R_{ds}^2 + (X'_{ds})^2} & B_{qs} &= \frac{-(X'_{qs} + X_c)}{R_{qs}^2 + (X'_{qs} + X_c)^2} \end{aligned}$$

This then leads to the following network interface.



Then apply the network interface equations, but use negative sign on the currents because we're using the motor convention of current instead of the generator convention.

$$\begin{aligned} V_{xd} &= \frac{eE_{dd} - fE_{qd}}{V} & V_{xq} &= \frac{eE_{dq} - fE_{qq}}{V} & I_r &= \begin{cases} -(V_{xd}G_{ds} - V_{yd}B_{ds}) \\ -(V_{xq}G_{qs} - V_{yq}B_{qs}) \end{cases} & \frac{dI_r}{de} &= +\frac{f}{V^2}I_i & \frac{dI_i}{de} &= -\frac{f}{V^2}I_r \\ V_{yd} &= \frac{fE_{dd} + eE_{qd}}{V} & V_{yq} &= \frac{fE_{dq} + eE_{qq}}{V} & I_i &= \begin{cases} -(V_{xd}B_{ds} + V_{yd}G_{ds}) \\ -(V_{xq}B_{qs} + V_{yq}G_{qs}) \end{cases} & \frac{dI_r}{df} &= -\frac{e}{V^2}I_i & \frac{dI_i}{df} &= +\frac{e}{V^2}I_r \end{aligned}$$

Finally the total terminal current and derivatives needed for the Jacobian are as follows

$$\begin{aligned} I_{rterm} &= I_r + e(G_{ds} + G_{qs}) - f(B_{ds} + B_{qs} + B_{extra}) & \frac{dI_r}{de} &= +\frac{f}{V^2}I_i + (G_{ds} + G_{qs}) \\ I_{iterm} &= I_r + e(B_{ds} + B_{qs} + B_{extra}) + f(G_{ds} + G_{qs}) & \frac{dI_r}{df} &= -\frac{e}{V^2}I_i - (B_{ds} + B_{qs} + B_{extra}) \\ & & \frac{dI_i}{de} &= -\frac{f}{V^2}I_r + (B_{ds} + B_{qs} + B_{extra}) \\ & & \frac{dI_i}{df} &= +\frac{e}{V^2}I_r + (G_{ds} + G_{qs}) \end{aligned}$$

Where B_{extra} represent the Extra Vars from the model initialization.

6 Calculation of Network Real and Reactive Power

Repeat the equations at the network interface we were using

$$|V_s| = [R_{ds} + jX'_{ds}](I_{dsR} + jI_{dsI}) + (E_{dd} + jE_{qd})$$

$$|V_s| = [R_{qs} + j(X'_{qs} + X_c)](I_{qsR} + jI_{qsI}) + (E_{dq} + jE_{qq})$$

Solve for the currents at the network

$$(I_{dsR} + jI_{dsI}) = \frac{|V_s| - (E_{dd} + jE_{qd})}{R_{ds} + jX'_{ds}} = [|V_s| - (E_{dd} + jE_{qd})](G_{ds} + jB_{ds})$$

$$(I_{qsR} + jI_{qsI}) = \frac{|V_s| - (E_{dq} + jE_{qq})}{R_{qs} + j(X'_{qs} + X_c)} = [|V_s| - (E_{dq} + jE_{qq})](G_{qs} + jB_{qs})$$

Now use the equation for complex power and get

$$S = VI^*, \text{ thus } S^* = V^*I$$

Apply these equations and go through the process of simplifying.

$$P_{ds} - jQ_{ds} = |V_s|(I_{dsR} + jI_{dsI}) = |V_s| [|V_s| - (E_{dd} + jE_{qd})](G_{ds} + jB_{ds})$$

$$P_{qs} - jQ_{qs} = |V_s|(I_{qsR} + jI_{qsI}) = |V_s| [|V_s| - (E_{dq} + jE_{qq})](G_{qs} + jB_{qs})$$

$$P_{ds} - jQ_{ds} = V_s [(V_s - E_{dd}) - jE_{qd}](G_{ds} + jB_{ds})$$

$$P_{qs} - jQ_{qs} = V_s [(V_s - E_{dq}) - jE_{qq}](G_{qs} + jB_{qs})$$

$$P_{ds} - jQ_{ds} = V_s \{ [(V_s - E_{dd})G_{ds} + E_{qd}B_{ds}] + j[(V_s - E_{dd})B_{ds} - E_{qd}G_{ds}] \}$$

$$P_{qs} - jQ_{qs} = V_s \{ [(V_s - E_{dq})G_{qs} + E_{qq}B_{qs}] + j[(V_s - E_{dq})B_{qs} - E_{qq}G_{qs}] \}$$

$$P_{ds} - jQ_{ds} = V_s \{ [(V_s - E_{dd})G_{ds} + E_{qd}B_{ds}] + j[(V_s - E_{dd})B_{ds} - E_{qd}G_{ds}] \}$$

$$P_{qs} - jQ_{qs} = V_s \{ [(V_s - E_{dq})G_{qs} + E_{qq}B_{qs}] + j[(V_s - E_{dq})B_{qs} - E_{qq}G_{qs}] \}$$

$$P_{ds} = V_s [(V_s - E_{dd})G_{ds} + E_{qd}B_{ds}]$$

$$Q_{ds} = V_s [-(V_s - E_{dd})B_{ds} + E_{qd}G_{ds}]$$

$$P_{qs} = V_s [(V_s - E_{dq})G_{qs} + E_{qq}B_{qs}]$$

$$Q_{qs} = V_s [-(V_s - E_{dq})B_{qs} + E_{qq}G_{qs}]$$

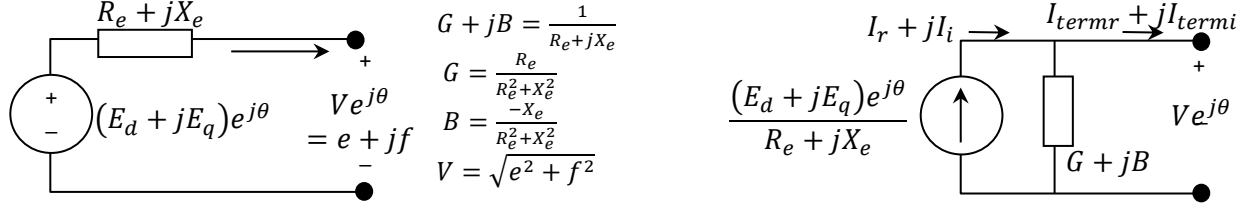
The total power then can be calculated from the network interface as follows.

$$P_{total} = P_{ds} + P_{qs} = V_s [(V_s - E_{dd})G_{ds} + E_{qd}B_{ds} + (V_s - E_{dq})G_{qs} + E_{qq}B_{qs}]$$

$$Q_{total} = Q_{ds} + Q_{qs} = V_s [-(V_s - E_{dd})B_{ds} + E_{qd}G_{ds} - (V_s - E_{dq})B_{qs} + E_{qq}G_{qs}] - V^2 B_{cap}$$

7 Network Interface Equations using Generator Convention

The network interface equations are what make this model different. E_d represents the part of the voltage that is in-phase with terminal voltage $V e^{j\theta}$ and E_q represents the voltage that is 90 degrees out-of-phase. The source impedance is $R_e + jX_e$. Define the terminal current as $I_r + jI_i$. This is depicted in the follow circuit diagram.



Now calculate the Norton current injection from this circuit as a function of the terminal voltage, internal voltage and Thevenin admittance.

$$\begin{aligned} I_r + jI_i &= [(E_d + jE_q)e^{j\theta}](G + jB) \\ I_r + jI_i &= \left[(E_d + jE_q) \frac{e + jf}{\sqrt{e^2 + f^2}} \right] (G + jB) \\ I_r + jI_i &= \left[\frac{(eE_d - fE_q) + j(fE_d + eE_q)}{\sqrt{e^2 + f^2}} \right] (G + jB) \\ I_r + jI_i &= \left[\frac{eE_d - fE_q}{\sqrt{e^2 + f^2}} + j \frac{fE_d + eE_q}{\sqrt{e^2 + f^2}} \right] (G + jB) \end{aligned}$$

To simplify the notation define $V_x = \frac{eE_d - fE_q}{\sqrt{e^2 + f^2}}$ and $V_y = \frac{fE_d + eE_q}{\sqrt{e^2 + f^2}}$. From this define the currents as

$$\begin{aligned} I_r + jI_i &= (V_x + jV_y)(G + jB) \\ I_r &= V_x G - V_y B \\ I_i &= V_x B + V_y G \end{aligned}$$

To solve the network boundary equations we will need to derivatives of the terminal currents which will require the derivatives of our defined variables $\frac{dx}{de}$, $\frac{dx}{df}$, $\frac{dy}{de}$ and $\frac{dy}{df}$. These are calculated as follows.

$$\begin{aligned} \frac{dV_x}{de} &= \frac{2e(eE_d - fE_q)}{-2(e^2 + f^2)^{3/2}} + \frac{E_d}{(e^2 + f^2)^{1/2}} = \frac{-e^2 E_d + efE_q}{(e^2 + f^2)^{3/2}} + \frac{+E_d(e^2 + f^2)}{(e^2 + f^2)^{3/2}} = \frac{+f^2 E_d + efE_q}{(e^2 + f^2)^{3/2}} = \frac{+f(fE_d + eE_q)}{(e^2 + f^2)^{3/2}} = \frac{+f}{(e^2 + f^2)} \left(\frac{fE_d + eE_q}{\sqrt{e^2 + f^2}} \right) = \frac{+fV_y}{V^2} \\ \frac{dV_x}{df} &= \frac{2f(eE_d - fE_q)}{-2(e^2 + f^2)^{3/2}} + \frac{-E_q}{(e^2 + f^2)^{1/2}} = \frac{-feE_d + f^2 E_q}{(e^2 + f^2)^{3/2}} + \frac{-E_q(e^2 + f^2)}{(e^2 + f^2)^{3/2}} = \frac{-feE_d - e^2 E_q}{(e^2 + f^2)^{3/2}} = \frac{-e(fE_d + eE_q)}{(e^2 + f^2)^{3/2}} = \frac{-e}{(e^2 + f^2)} \left(\frac{fE_d + eE_q}{\sqrt{e^2 + f^2}} \right) = \frac{-eV_y}{V^2} \\ \frac{dV_y}{de} &= \frac{2e(fE_d + eE_q)}{-2(e^2 + f^2)^{3/2}} + \frac{E_q}{(e^2 + f^2)^{1/2}} = \frac{-efE_d - e^2 E_q}{(e^2 + f^2)^{3/2}} + \frac{+E_q(e^2 + f^2)}{(e^2 + f^2)^{3/2}} = \frac{-efE_d + f^2 E_q}{(e^2 + f^2)^{3/2}} = \frac{-f(eE_d - fE_q)}{(e^2 + f^2)^{3/2}} = \frac{-f}{(e^2 + f^2)} \left(\frac{eE_d - fE_q}{\sqrt{e^2 + f^2}} \right) = \frac{-fV_x}{V^2} \\ \frac{dV_y}{df} &= \frac{2f(fE_d + eE_q)}{-2(e^2 + f^2)^{3/2}} + \frac{E_d}{(e^2 + f^2)^{1/2}} = \frac{-ffE_d - f^2 E_q}{(e^2 + f^2)^{3/2}} + \frac{+E_d(e^2 + f^2)}{(e^2 + f^2)^{3/2}} = \frac{-efE_q + e^2 E_d}{(e^2 + f^2)^{3/2}} = \frac{+e(eE_d - fE_q)}{(e^2 + f^2)^{3/2}} = \frac{+e}{(e^2 + f^2)} \left(\frac{eE_d - fE_q}{\sqrt{e^2 + f^2}} \right) = \frac{+eV_x}{V^2} \end{aligned}$$

From these calculate the current derivatives as follows

$$\begin{aligned} \frac{dI_r}{de} &= \frac{dV_x}{de} G - \frac{dV_y}{de} B = \frac{+fV_y}{V^2} G - \frac{-fV_x}{V^2} B = +\frac{f}{V^2} (V_y G + V_x B) = +\frac{f}{V^2} I_i \\ \frac{dI_r}{df} &= \frac{dV_x}{df} G - \frac{dV_y}{df} B = \frac{-eV_y}{V^2} G - \frac{+eV_x}{V^2} B = -\frac{e}{V^2} (V_y G + V_x B) = -\frac{e}{V^2} I_i \\ \frac{dI_i}{de} &= \frac{dV_x}{de} B + \frac{dV_y}{de} G = \frac{+fV_y}{V^2} B + \frac{-fV_x}{V^2} G = -\frac{f}{V^2} (V_x G - V_y B) = -\frac{f}{V^2} I_r \\ \frac{dI_i}{df} &= \frac{dV_x}{df} B + \frac{dV_y}{df} G = \frac{-eV_y}{V^2} B + \frac{+eV_x}{V^2} G = +\frac{e}{V^2} (V_x G - V_y B) = +\frac{e}{V^2} I_r \end{aligned}$$

From these definitions we have the following Norton currents. These are not just constants like for a synchronous machine though. These Norton currents are a function of the terminal voltage angle, so they effect the Jacobian. After complete simplification, we get the following.

$$\begin{aligned} V_x &= \frac{eE_d - fE_q}{V} & I_r &= V_x G - V_y B & \frac{dI_r}{de} &= +\frac{f}{V^2} I_i & \frac{dI_i}{de} &= -\frac{f}{V^2} I_r \\ V_y &= \frac{fE_d + eE_q}{V} & I_i &= V_x B + V_y G & \frac{dI_r}{df} &= -\frac{e}{V^2} I_i & \frac{dI_i}{df} &= +\frac{e}{V^2} I_r \end{aligned}$$

8 Derivation of the Model

This model is taken from the paper Phasor Modeling Approach for Single Phase A/C Motors by B. Lesieutre, M. Kosterev, and J. Undrill published in the 2008 IEEE Power and Energy Society General Meeting – Conversion and Delivery of Electrical Energy in the 21st Century. We make a few minor syntax changes to the derivation in that paper immediately though to be consistent with the nomenclature in this document. In the paper we see values of ω_s and ω_r which refer to the actual speed of the stator voltage signal and rotor speed. Our preference is for these values to represent the per unit of synchronous speed. Thus the translations from the paper to the nomenclature in the document are described as follows

$$\begin{aligned}\omega_s &= \frac{\omega_{spaper}}{\omega_b} & \omega_{spaper} - \omega_{rpaper} &= \omega_b(\omega_s - \omega_r) \\ \omega_r &= \frac{\omega_{rpaper}}{\omega_b} & \omega_{spaper} + \omega_{rpaper} &= \omega_b(\omega_s + \omega_r)\end{aligned}$$

Also the nomenclature in the paper includes some superscript R and I values to represent either the real or imaginary part of a complex number. In the nomenclature that follows these have all been moved to the end of subscripts instead.

$$\begin{aligned}I_{ds}^R(paper) &= I_{dsR}(this\ document) \\ \psi_{dr}^I(paper) &= \psi_{drI}(this\ document)\end{aligned}$$

8.1 Original Model from Lesieutre Paper

Network Interface Equations are as follows.

- $|V_s| = [R_{ds} + j(\omega_s)X'_{ds}](I_{dsR} + jI_{dsI}) + j(\omega_s)\frac{X_m}{X_r}(\psi_{drR} + j\psi_{drI})$
- $|V_s| = [R_{qs} + j(\omega_s)(X'_{qs} + X_c)](I_{qsR} + jI_{qsI}) + j(\omega_s)\frac{nX_m}{X_r}(\psi_{qrR} + j\psi_{qrI})$
- $I_s = [I_{dsR} + jI_{dsI} + I_{qsR} + jI_{qsI}]e^{j\phi} = [I_{dsR} + jI_{dsI}]e^{j\phi} + [I_{qsR} + jI_{qsI}]e^{j\phi}$

In the model, the network equations were derived using a dq reference frame. The differential equations for this model however are written in terms of flux variables that represent fields rotating in a forward direction and those rotating a backward direction. The translation relationships can be written.

$$\begin{aligned}\begin{bmatrix} \psi_{fR} + j\psi_{fI} \\ \psi_{bR} + j\psi_{bI} \end{bmatrix} &= \frac{1}{2} \begin{bmatrix} 1 & -j \\ 1 & +j \end{bmatrix} \begin{bmatrix} \psi_{drR} + j\psi_{drI} \\ \psi_{qrR} + j\psi_{qrI} \end{bmatrix} & \begin{bmatrix} \psi_{drR} + j\psi_{drI} \\ \psi_{qrR} + j\psi_{qrI} \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ +j & -j \end{bmatrix} \begin{bmatrix} \psi_{fR} + j\psi_{fI} \\ \psi_{bR} + j\psi_{bI} \end{bmatrix} \\ \begin{bmatrix} I_{fR} + jI_{fI} \\ I_{bR} + jI_{bI} \end{bmatrix} &= \frac{1}{2} \begin{bmatrix} 1 & -jn \\ 1 & +jn \end{bmatrix} \begin{bmatrix} I_{drR} + jI_{drI} \\ I_{qrR} + jI_{qrI} \end{bmatrix} & \begin{bmatrix} I_{drR} + jI_{drI} \\ I_{qrR} + jI_{qrI} \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ +\frac{j}{n} & -\frac{j}{n} \end{bmatrix} \begin{bmatrix} I_{fR} + jI_{fI} \\ I_{bR} + jI_{bI} \end{bmatrix}\end{aligned}$$

Finally the dynamic equations are derived as follows, although the second equation is just the algebraic result from neglecting backward rotating rotor dynamics.

- $T'_o \frac{d(\psi_{fR} + j\psi_{fI})}{dt} = X_m(I_{fR} + jI_{fI}) - [Sat(\psi_f, \psi_b) + jT'_o\omega_b(\omega_s - \omega_r)](\psi_{fR} + j\psi_{fI})$
- $0 = X_m(I_{bR} + jI_{bI}) - [Sat(\psi_f, \psi_b) + jT'_o\omega_b(\omega_s + \omega_r)](\psi_{bR} + j\psi_{bI})$
- $2H \frac{d\omega_r}{dt} = 2\frac{X_m}{X_r}(I_{fI}\psi_{fR} - I_{fR}\psi_{fI} - I_{bI}\psi_{bR} + I_{bR}\psi_{bI}) - T_{mech}$

Where the saturation function and T'_o are defined as

- $Sat(\psi_f, \psi_b) = \begin{cases} 1 & \text{for } |\psi| \leq B_{sat} \\ 1 + A_{sat}(|\psi| - B_{sat})^2 & \text{for } |\psi| > B_{sat} \end{cases}$
- $|\psi| = \sqrt{\psi_{fR}^2 + \psi_{fI}^2 + \psi_{bR}^2 + \psi_{bI}^2}$
- $T'_o = \frac{X_r}{\omega_b R_r}$
- $T_{elec} = 2\frac{X_m}{X_r}(I_{fI}\psi_{fR} - I_{fR}\psi_{fI} - I_{bI}\psi_{bR} + I_{bR}\psi_{bI})$

8.2 Translation to a Transient Stability Model

Another slight variation to the original model will be to remove the frequency-dependence from the reactances in the network boundary equations. This is to be consistent with all the other models we use such as GENROU, GENTPJ, and GENQEC, and the transmission line reactances which do not include this impact. This changes our network boundary equation interface to

- $|V_s| = [R_{ds} + j(\omega_s)X'_{ds}](I_{dsR} + jI_{dsI}) + j(\omega_s)\frac{X_m}{X_r}(\psi_{drR} + j\psi_{drI})$
- $|V_s| = [R_{qs} + j(\omega_s)(X'_{qs} + X_c)](I_{qsR} + jI_{qsI}) + j(\omega_s)\frac{nX_m}{X_r}(\psi_{qrR} + j\psi_{qrI})$

Next, rewrite these equations to remove the complex numbers and express them only in terms of real numbers instead. This is again to be consistent with other models and to aid in the implementation of these models in software tools. The network boundary equations will be instead written as

- $|V_s| = R_{ds}I_{dsR} - X'_{ds}I_{dsI} - (\omega_s)\frac{X_m}{X_r}\psi_{drI}$ (real part of first equation)
- $0 = R_{ds}I_{dsI} + X'_{ds}I_{dsR} + (\omega_s)\frac{X_m}{X_r}\psi_{drR}$ (imaginary part of first equation)
- $|V_s| = R_{qs}I_{qsR} - (X'_{qs} + X_c)I_{qsI} - (\omega_s)\frac{nX_m}{X_r}\psi_{qrI}$ (real part of second equation)
- $0 = R_{qs}I_{qsI} + (X'_{qs} + X_c)I_{qsR} + (\omega_s)\frac{nX_m}{X_r}\psi_{qrR}$ (imaginary part of second equation)

Also, rewrite the matrix conversion equations between the d and q axis and the forward and backward frames as follows.

- $\psi_{fR} + j\psi_{fI} = \frac{1}{2}[(\psi_{drR} + j\psi_{drI}) - j(\psi_{qrR} + j\psi_{qrI})] = \frac{1}{2}[(\psi_{drR} + \psi_{qrI}) + j(\psi_{drI} - \psi_{qrR})]$
- $\psi_{bR} + j\psi_{bI} = \frac{1}{2}[(\psi_{drR} + j\psi_{drI}) + j(\psi_{qrR} + j\psi_{qrI})] = \frac{1}{2}[(\psi_{drR} - \psi_{qrI}) + j(\psi_{drI} + \psi_{qrR})]$
- $\psi_{drR} + j\psi_{drI} = (\psi_{fR} + j\psi_{fI}) + (\psi_{bR} + j\psi_{bI}) = (\psi_{fR} + \psi_{bR}) + j(\psi_{fI} + \psi_{bI})$
- $\psi_{qrR} + j\psi_{qrI} = j(\psi_{fR} + j\psi_{fI}) - j(\psi_{bR} + j\psi_{bI}) = (-\psi_{fI} + \psi_{bI}) + j(\psi_{fR} - \psi_{bR})$
- $I_{fR} + jI_{fI} = \frac{1}{2}[(I_{dsR} + jI_{dsI}) - jn(I_{qsR} + jI_{qsI})] = \frac{1}{2}[(I_{dsR} + nI_{qsI}) + j(I_{dsI} - nI_{qsR})]$
- $I_{bR} + jI_{bI} = \frac{1}{2}[(I_{dsR} + jI_{dsI}) + jn(I_{qsR} + jI_{qsI})] = \frac{1}{2}[(I_{dsR} - nI_{qsI}) + j(I_{dsI} + nI_{qsR})]$
- $I_{dsR} + jI_{dsI} = (I_{fR} + jI_{fI}) + (I_{bR} + jI_{bI}) = (I_{fR} + I_{bR}) + j(I_{fI} + I_{bI})$
- $I_{qsR} + jI_{qsI} = \frac{j}{n}(I_{fR} + jI_{fI}) - \frac{j}{n}(I_{bR} + jI_{bI}) = \frac{(-I_{fI} + I_{bI})}{n} + j\frac{(I_{fR} - I_{bR})}{n}$

This then gives the following conversions between the various current and flux quantities on different reference frames.

- $\psi_{fR} = \frac{1}{2}(\psi_{drR} + \psi_{qrI})$ and $\psi_{fI} = \frac{1}{2}(\psi_{drI} - \psi_{qrR})$
- $\psi_{bR} = \frac{1}{2}(\psi_{drR} - \psi_{qrI})$ and $\psi_{bI} = \frac{1}{2}(\psi_{drI} + \psi_{qrR})$
- $\psi_{drR} = (\psi_{fR} + \psi_{bR})$ and $\psi_{drI} = (\psi_{fI} + \psi_{bI})$
- $\psi_{qrR} = (-\psi_{fI} + \psi_{bI})$ and $\psi_{qrI} = (\psi_{fR} - \psi_{bR})$
- $I_{fR} = \frac{1}{2}(I_{dsR} + nI_{qsI})$ and $I_{fI} = \frac{1}{2}(I_{dsI} - nI_{qsR})$
- $I_{bR} = \frac{1}{2}(I_{dsR} - nI_{qsI})$ and $I_{bI} = \frac{1}{2}(I_{dsI} + nI_{qsR})$
- $I_{dsR} = (I_{fR} + I_{bR})$ and $I_{dsI} = (I_{fI} + I_{bI})$
- $I_{qsR} = \frac{1}{n}(-I_{fI} + I_{bI})$ and $I_{qsI} = \frac{1}{n}(I_{fR} - I_{bR})$

This means that the network boundary equations can be converted to the following.

- $0 = R_{ds}(I_{fI} + I_{bI}) + X'_{ds}(I_{fR} + I_{bR}) + (\omega_s) \frac{X_m}{X_r} (\psi_{fR} + \psi_{bR})$
- $|V_s| = R_{ds}(I_{fR} + I_{bR}) - X'_{ds}(I_{fI} + I_{bI}) - (\omega_s) \frac{X_m}{X_r} (\psi_{fI} + \psi_{bI})$
- $|V_s| = R_{qs} \frac{1}{n} (-I_{fI} + I_{bI}) - (X'_{qs} + X_c) \frac{1}{n} (I_{fR} - I_{bR}) - (\omega_s) \frac{nX_m}{X_r} (\psi_{fR} - \psi_{bR})$
- $0 = R_{qs} \frac{1}{n} (I_{fR} - I_{bR}) + (X'_{qs} + X_c) \frac{1}{n} (-I_{fI} + I_{bI}) + (\omega_s) \frac{nX_m}{X_r} (-\psi_{fI} + \psi_{bI})$

Also means that the dynamic equations can be converted to the following.

- $T'_o \frac{d(\psi_{fR})}{dt} = X_m I_{fR} - \psi_{fR} Sat + T'_o \omega_b (\omega_s - \omega_r) \psi_{fI}$
- $T'_o \frac{d(\psi_{fI})}{dt} = X_m I_{fI} - \psi_{fI} Sat - T'_o \omega_b (\omega_s - \omega_r) \psi_{fR}$
- $0 = X_m I_{bR} - \psi_{bR} Sat + T'_o \omega_b (\omega_s + \omega_r) \psi_{bI}$
- $0 = X_m I_{bI} - \psi_{bI} Sat - T'_o \omega_b (\omega_s + \omega_r) \psi_{bR}$
- $2H \frac{d\omega_r}{dt} = 2 \frac{X_m}{X_r} (I_{fI} \psi_{fR} - I_{fR} \psi_{fI} - I_{bI} \psi_{bR} + I_{bR} \psi_{bI}) - T_{mech}$

Where the saturation function and T'_o are defined as

- $Sat = Sat(\psi_f, \psi_b) = \begin{cases} 1 & \text{for } |\psi| \leq B_{sat} \\ 1 + A_{sat} (|\psi| - B_{sat})^2 & \text{for } |\psi| > B_{sat} \end{cases}$
- $|\psi| = \sqrt{\psi_{fR}^2 + \psi_{fI}^2 + \psi_{bR}^2 + \psi_{bI}^2}$
- $T'_o = \frac{1}{\omega_b} \frac{X_r}{R_r}$
- $T_{elec} = 2 \frac{X_m}{X_r} (I_{fI} \psi_{fR} - I_{fR} \psi_{fI} - I_{bI} \psi_{bR} + I_{bR} \psi_{bI})$

9 Psuedo Code

9.1 Input Parameters

```
fLoadFactor, fRds, fXds, fRqs, fXqs, fXc, fXm, fnRatio, fH, fXr, fRr, fA, fB, fC0, fD, fE, fTnomObject, fObjectMVABase, fSatA, fSatB
fPInitialObject, fQInitialObject : double;
// Dynamic States in this document (Note: we treat PsiBr and PsiBi as "states" but we implement them as ignored algebraic states
Speed, SpeedDelayed, PsiFr, PsiFi, PsiBr, PsiBi : double;
// Algebraic Variables
aSat, aIFr, aIFi, aIBr, aIBi : double;
// Network boundary equation. These get calculated in the Load and are used throughout
fGdObject, fBdObject, fGqObject, fBqObject, fBExtraObject : Double;
// Network boundary equation. These are updated each time derivatives are calculated
fCACHE_Edd, fCACHE_Eqd : Double;
fCACHE_Edq, fCACHE_Eqq : Double;
```

9.2 Part of Setting up the model

```
// Network boundary equation. These get calculated in the Load and are used throughout
fGdObject := + fRds / ( sqr(fRds) + sqr(fXds) );
fBdObject := - fXds / ( sqr(fRds) + sqr(fXds) );
fGqObject := + fRqs / ( sqr(fRqs) + sqr(fXqs+fXc) );
fBqObject := - (fXqs+fXc) / ( sqr(fRqs) + sqr(fXqs+fXc) );
fBExtraObject := 0;
```

9.3 Procedure to Calculate fSatA and fSatB from the input parameters E1, S1, E2, S2

```
procedure SaturationCalculateValues(tempE1,tempS1,tempE2,tempS2 : Double);
Var tempA,tempB,tempC,result1,result2 : Double; // quadratic equation coefficients and results
isImag : Boolean; // True if quadratic equation results are imaginary
Begin
  if ( (tempS1 <= 0) and (tempS2 <= 0) ) // both zero or negative, then ignore
  or
    (abs(tempS1-tempS2) < 1E-6) // saturation values that are equal are also a problem
  then begin
    fSatA := 0;
    fSatB := 0;
  end
  else begin
    if (tempS1 <= 0) then begin
      fSatB := tempE1;
    end
    else begin
      tempA := tempS2 - tempS1;
      tempB := 2*(-tempS2*tempE1+tempS1*tempE2);
      tempC := + tempS2*sqr(tempE1) - tempS1*sqr(tempE2);
      PWSolveQuadraticEq(tempA,tempB,tempC,result1,result2,isImag);
      fSatB := result2;
    end;
    fSatA := tempS2/Sqr(tempE2-fSatB);
  end;
End;
```

9.4 Electrical Torque Equation

```
function ElectricalTorqueObject(PsiFr, PsiFi, PsiBr, PsiBi : double) : double;
begin
    result := 2*fXm/fXr*(aIFi*PsiFr - aIFr*PsiFi - aIBi*PsiBr + aIBr*PsiBi)
end;
```

9.5 Mechanical Torque Equation

Note: It is important to handle situations where the speed becomes negative as when the speed hovers near 0.0 it can cause numerical instability if we do not do this.

```
Function MechanicalTorqueObject(Speed, SpeedDelayed : double) : Double;
Begin
    // SpeedDelayed add on June 15, 2023
    Result := (fA*Sqr(Speed) + fB*Speed + fBpgas*SpeedDelayed + fC0 + fD*Math_Y_to_X(abs(Speed), fE))*fTnomObject;
    if (Speed < 0) then result := -result;
end;
```

9.6 Real and Reactive Power Equations from Network Interface Parameters and Voltage

```
Function TotalP(local_VoltMag : Double; local_BusPUDeltaFreq : Single) : Double;
begin
    result := local_VoltMag * ( + (local_VoltMag - fCACHE_Edd)*fGdObject + fCACHE_Eqd*fBdObject
                                + (local_VoltMag - fCACHE_Edq)*fGqObject + fCACHE_Eqq*fBqobject
                                );
    result := result * FractionOfNominalStillOn * fObjectMVABase/ SystemMVABase;
end;

Function TotalQ(local_VoltMag : Double; local_BusPUDeltaFreq : Single) : Double;
begin
    result := local_VoltMag * ( - (local_VoltMag - fCACHE_Edd)*fBdObject + fCACHE_Eqd*fGdObject
                                - (local_VoltMag - fCACHE_Edq)*fBqObject + fCACHE_Eqq*fGqobject
                                );
    Result := Result - fBExtraObject*Sqr(local_VoltMag);
    result := result * FractionOfNominalStillOn * fObjectMVABase/ SystemMVABase;
end;
```

9.7 Fraction of Nominal StillOn

```
function FractionOfNominalStillOn : double;
begin
    // This is a function that will manage the fractional tripping
    result := 1;
end;
```

9.8 Initialization of Model from Terminal Power and Voltage

```
function InitializeFluxCurrentSpeedSat(thePUTol, thePmotor, theVnow : double;  
                                     var theQmotorCalc, Speed, PsiFr, PsiFi, PsiBr, PsiBi : double) : boolean;  
                                     // aSat, aIFr, aIFi, aIBr, aIBi are algebraic variables  
var theMatrix : Array of doubles;  
    theB      : Vector of doubles;  
    theX      : Vector of doubles;  
    _Topwb    : double;  
    _ws       : double;  
    _Psi      : double;  
  
procedure LOCAL_PopulateTopLeft8X8;  
begin  
    theMatrix[1,1] := +fXm/fXr;           theMatrix[2,1] := 0;  
    theMatrix[1,2] := 0;                 theMatrix[2,2] := -fXm/fXr;  
    theMatrix[1,3] := +fXm/fXr;           theMatrix[2,3] := 0;  
    theMatrix[1,4] := 0;                 theMatrix[2,4] := -fXm/fXr;  
    theMatrix[1,5] := +fXds;             theMatrix[2,5] := +fRds;  
    theMatrix[1,6] := +fRds;             theMatrix[2,6] := -fXds;  
    theMatrix[1,7] := +fXds;             theMatrix[2,7] := +fRds;  
    theMatrix[1,8] := +fRds;             theMatrix[2,8] := -fXds;  
  
    theMatrix[3,1] := -fnRatio*fXm/fXr;   theMatrix[4,1] := 0;  
    theMatrix[3,2] := 0;                 theMatrix[4,2] := -fnRatio*fXm/fXr;  
    theMatrix[3,3] := +fnRatio*fXm/fXr;   theMatrix[4,3] := 0;  
    theMatrix[3,4] := 0;                 theMatrix[4,4] := +fnRatio*fXm/fXr;  
    theMatrix[3,5] := -(fXqs+fXc)/fnRatio; theMatrix[4,5] := +fRqs/fnRatio;  
    theMatrix[3,6] := -fRqs/fnRatio;       theMatrix[4,6] := -(fXqs+fXc)/fnRatio;  
    theMatrix[3,7] := +(fXqs+fXc)/fnRatio; theMatrix[4,7] := -fRqs/fnRatio;  
    theMatrix[3,8] := +fRqs/fnRatio;       theMatrix[4,8] := +(fXqs+fXc)/fnRatio;  
  
    theMatrix[5,1] := -aSat;             theMatrix[6,1] := -_Topwb*(_ws - Speed);  
    theMatrix[5,2] := +_Topwb*(_ws - Speed); theMatrix[6,2] := -aSat;  
    theMatrix[5,3] := 0;                 theMatrix[6,3] := 0;  
    theMatrix[5,4] := 0;                 theMatrix[6,4] := 0;  
    theMatrix[5,5] := +fXm;             theMatrix[6,5] := 0;  
    theMatrix[5,6] := 0;                 theMatrix[6,6] := +fXm;  
    theMatrix[5,7] := 0;                 theMatrix[6,7] := 0;  
    theMatrix[5,8] := 0;                 theMatrix[6,8] := 0;  
  
    theMatrix[7,1] := 0;                 theMatrix[8,1] := 0;  
    theMatrix[7,2] := 0;                 theMatrix[8,2] := 0;  
    theMatrix[7,3] := -aSat;             theMatrix[8,3] := -_Topwb*(_ws + Speed);  
    theMatrix[7,4] := +_Topwb*(_ws + Speed); theMatrix[8,4] := -aSat;  
    theMatrix[7,5] := 0;                 theMatrix[8,5] := 0;  
    theMatrix[7,6] := 0;                 theMatrix[8,6] := 0;  
    theMatrix[7,7] := +fXm;             theMatrix[8,7] := 0;  
    theMatrix[7,8] := 0;                 theMatrix[8,8] := +fXm;  
end;
```

```

procedure LOCAL_PopulatePowerEquationAtRowFirst8(theRow : integer);
begin
    theMatrix[theRow,1] := 0;
    theMatrix[theRow,2] := 0;
    theMatrix[theRow,3] := 0;
    theMatrix[theRow,4] := 0;
    theMatrix[theRow,5] := +1;
    theMatrix[theRow,6] := -1/fnRatio;
    theMatrix[theRow,7] := +1;
    theMatrix[theRow,8] := +1/fnRatio;
end;

procedure LOCAL_PopulateAlgebraicAndStatesAndCalculatePsi;
begin
    PsiFr      := theX[ 1];
    PsiFi      := theX[ 2];
    PsiBr      := theX[ 3];
    PsiBi      := theX[ 4];
    aIFr       := theX[ 5];
    aIFi       := theX[ 6];
    aIBr       := theX[ 7];
    aIBi       := theX[ 8];
    Speed      := theX[ 9];
    aSat       := theX[10];
    Psi := sqrt( sqr(PsiFr) + sqr(PsiFi) + sqr(PsiBr) + sqr(PsiBi) );
end;

function LOCAL_PopulateMismatchBAndNorm : double;
var i : integer;
begin
    theB[ 1] :=          + fRds          *(+ aIFi + aIBi) + fXds          *(+ aIFr + aIBr) + fXm/fXr          *(+ PsiFr + PsiBr);
    theB[ 2] := -theVnow + fRds          *(+ aIFr + aIBr) - fXds          *(+ aIFi + aIBi) - fXm/fXr          *(+ PsiFi + PsiBi);
    theB[ 3] := -theVnow + fRqs/fnRatio*(- aIFi + aIBi) - (fXqs+fXc)/fnRatio*(+ aIFr - aIBr) - fXm/fXr*fnRatio*(+ PsiFr - PsiBr);
    theB[ 4] :=          + fRqs/fnRatio*(+ aIFr - aIBr) + (fXqs+fXc)/fnRatio*(- aIFi + aIBi) + fXm/fXr*fnRatio*(- PsiFi + PsiBi);
    theB[ 5] := +fXm*aIFr - PsiFr*aSat + _Topwb*(_ws - Speed)*PsiFi;
    theB[ 6] := +fXm*aIFi - PsiFi*aSat - _Topwb*(_ws - Speed)*PsiFr;
    theB[ 7] := +fXm*aIBr - PsiBr*aSat + _Topwb*(_ws + Speed)*PsiBi;
    theB[ 8] := +fXm*aIBi - PsiBi*aSat - _Topwb*(_ws + Speed)*PsiBr;
    theB[ 9] := aIFr + aIBr + 1/fnRatio*(-aIFi + aIBi) - thePmotor/theVnow;
    if _Psi < fSatB
    then theB[10] := -aSat + 1
    else theB[10] := -aSat + 1 + fSatA*sqr(_Psi - fSatB) ;

    result := 0;
    for i := 1 to 10 do result := result + sqr(theB[i]);
    result := sqrt(result);
end;

```

```

var i : integer;
    tempSatTerm : double;
    IterationCount : integer;
begin
    result := false;
    theMatrix := Matrix of 8x8 doubles;
    theB      := Vector of 8 doubles;
    theX      := Vector of 8 doubles;

    _ws := 1.0; // initial frequency
    _Topwb := fXr/fRr;

    // Get the initial Gues
    aSat := 1.0;
    Speed := 0.99; // initial guess is slightly lowere speed
    LOCAL_PopulateTopLeft8X8;
    LOCAL_PopulatePowerEquationAtRowFirst8(8); // replace with this.
    theB[1] := 0;
    theB[2] := theVnow;
    theB[3] := theVnow;
    theB[4] := 0;
    theB[5] := 0;
    theB[6] := 0;
    theB[7] := 0;
    theB[8] := thePmotor/theVnow;
    theMatrix.Factor(false,0, TRUE);
    theMatrix.ForwardBackward(theB);
    FreeAndNil(theMatrix);

    theMatrix := Matrix of 10x10 doubles;
    For i := 1 to 8 Do theX[i] := theB[i];
    // increase size of vector to 10
    theX.Add(Speed); // 9
    theX.Add(aSat); // 10
    theB.Add(0); // 9
    theB.Add(0); // 10
    LOCAL_PopulateAlgebraicAndStatesAndCalculatePsi;

    IterationCount := 0;
    While (LOCAL_PopulateMismatchBAndNorm > thePUTol) and (IterationCount < 20) do begin
        Inc(IterationCount);
        // Same method as for initial guess for the upper left 8x8 matrix
        LOCAL_PopulateTopLeft8X8;
        // Add the entries in the 9th and 10 column for the forward and backward flux equation s
        theMatrix[5, 9] := -_Topwb*PsiFi;    theMatrix[5,10] := -PsiFr;
        theMatrix[6, 9] := +_Topwb*PsiFr;    theMatrix[6,10] := -PsiFi;
        theMatrix[7, 9] := +_Topwb*PsiBi;    theMatrix[7,10] := -PsiBr;
        theMatrix[8, 9] := -_Topwb*PsiBr;    theMatrix[8,10] := -PsiBi;
        // Build the 9th equation which enforces that the Pinit/V matches
        LOCAL_PopulatePowerEquationAtRowFirst8(9);
        theMatrix[9, 9] := 0; theMatrix[9,10] := 0;

```

```

// Build the 10th equation which calculated the saturation function
if _Psi < fSatB then begin
  theMatrix[10, 1] := 0;
  theMatrix[10, 2] := 0;
  theMatrix[10, 3] := 0;
  theMatrix[10, 4] := 0;
end
else begin
  tempSatTerm := 2*fSatA*(1-fSatB/_Psi);
  theMatrix[10, 1] := +tempSatTerm*PsiFr;
  theMatrix[10, 2] := +tempSatTerm*PsiFi;
  theMatrix[10, 3] := +tempSatTerm*PsiBr;
  theMatrix[10, 4] := +tempSatTerm*PsiBi;
end;
theMatrix[10, 5] := 0;
theMatrix[10, 6] := 0;
theMatrix[10, 7] := 0;
theMatrix[10, 8] := 0;
theMatrix[10, 9] := 0;
theMatrix[10,10] := -1;

// Solve Ax=b for DeltaX = Inv(A)*B
theMatrix.Factor;
theMatrix.ForwardBackward(theB);
For i := 1 to 10 Do theX[i] := theX[i] - theB[i];
LOCAL_PopulateAlgebraicAndStatesAndCalculatePsi;
end;

FreeAndNil(theMatrix);
FreeAndNil(theB);
FreeAndNil(theX);

result := (IterationCount < 20);

// The initial reactive power is a function of the algebraic states
theQmotorCalc := -theVnow*(aIFi + aIBi + 1/fnRatio*(+aIFr - aIBr));

CalculateNetworkInterfaceParameters(_ws, PsiFr, PsiFi, PsiBr, PsiBi);
end;

```

9.9 Algebraic Equations Solved before Differential Equations

```
procedure AlgebraicUpdate(theVnow, FreqPU, Speed, PsiFr, PsiFi : double; var PsiBr, PsiBi : double);
var theMatrix : Array of doubles;
    theB       : Vector of doubles;
    theX       : Vector of doubles;
    _wb : double;
    _Top : double;
    _Topwb : double;
    _Psi : double;

procedure LOCAL_PopulateJacobian;
var tempSatTerm : double;
begin
    theMatrix[1,1] := +fXds;
    theMatrix[1,2] := +fRds;
    theMatrix[1,3] := +fXm/fXr;
    theMatrix[1,4] := 0;
    theMatrix[1,5] := +fXds;
    theMatrix[1,6] := +fRds;
    theMatrix[1,7] := 0;
    theMatrix[3,1] := -(fXqs+fXc)/fnRatio;
    theMatrix[3,2] := -fRqs/fnRatio;
    theMatrix[3,3] := +fnRatio*fXm/fXr;
    theMatrix[3,4] := 0;
    theMatrix[3,5] := +(fXqs+fXc)/fnRatio;
    theMatrix[3,6] := +fRqs/fnRatio;
    theMatrix[3,7] := 0;
    theMatrix[5,1] := 0;
    theMatrix[5,2] := 0;
    theMatrix[5,3] := -aSat;
    theMatrix[5,4] := +_Topwb*(FreqPU + Speed);
    theMatrix[5,5] := +fXm;
    theMatrix[5,6] := 0;
    theMatrix[5,7] := -PsiBr;
    // Build the 10th equation which calculated the saturation function
    theMatrix[7,1] := 0;
    theMatrix[7,2] := 0;
    if _Psi < fSatB then begin
        theMatrix[7,3] := 0;
        theMatrix[7,4] := 0;
    end
    else begin
        tempSatTerm := 2*fSatA*(1-fSatB/_Psi);
        theMatrix[7,3] := +tempSatTerm*PsiBr;
        theMatrix[7,4] := +tempSatTerm*PsiBi;
    end;
    theMatrix[7,5] := 0;
    theMatrix[7,6] := 0;
    theMatrix[7,7] := -1;
end;

    theMatrix[2,1] := +fRds;
    theMatrix[2,2] := -fXds;
    theMatrix[2,3] := 0;
    theMatrix[2,4] := -fXm/fXr;
    theMatrix[2,5] := +fRds;
    theMatrix[2,6] := -fXds;
    theMatrix[2,7] := 0;
    theMatrix[4,1] := +fRqs/fnRatio;
    theMatrix[4,2] := -(fXqs+fXc)/fnRatio;
    theMatrix[4,3] := 0;
    theMatrix[4,4] := +fnRatio*fXm/fXr;
    theMatrix[4,5] := -fRqs/fnRatio;
    theMatrix[4,6] := +(fXqs+fXc)/fnRatio;
    theMatrix[4,7] := 0;
    theMatrix[6,1] := 0;
    theMatrix[6,2] := 0;
    theMatrix[6,3] := -_Topwb*(FreqPU + Speed);
    theMatrix[6,4] := -aSat;
    theMatrix[6,5] := 0;
    theMatrix[6,6] := +fXm;
    theMatrix[6,7] := -PsiBi;
```

```

procedure LOCAL_PopulateAlgebraicAndStatesAndCalculatePsi;
begin
    aIFr      := theX[ 1];
    aIFi      := theX[ 2];
    PsiBr     := theX[ 3];
    PsiBi     := theX[ 4];
    aIBr      := theX[ 5];
    aIBi      := theX[ 6];
    aSat      := theX[ 7];
    _Psi := sqrt( sqr(PsiFr) + sqr(PsiFi) + sqr(PsiBr) + sqr(PsiBi) );
end;

function LOCAL_PopulateMismatchBAndNorm : double;
var i : integer;
begin
    theB[1] :=          + fRds          *(+ aIFi + aIBi) + fXds          *(+ aIFr + aIBr) + fXm/fXr          *(+ PsiFr + PsiBr);
    theB[2] := -theVnow + fRds          *(+ aIFr + aIBr) - fXds          *(+ aIFi + aIBi) - fXm/fXr          *(+ PsiFi + PsiBi);
    theB[3] := -theVnow + fRqs/fnRatio*(- aIFi + aIBi) - (fXqs+fXc)/fnRatio*(+ aIFr - aIBr) - fXm/fXr*fnRatio*(+ PsiFr - PsiBr);
    theB[4] :=          + fRqs/fnRatio*(+ aIFr - aIBr) + (fXqs+fXc)/fnRatio*(- aIFi + aIBi) + fXm/fXr*fnRatio*(- PsiFi + PsiBi);
    theB[5] := +fXm*aIBr - PsiBr*aSat + _Topwb*(FreqPU + Speed)*PsiBi;
    theB[6] := +fXm*aIBi - PsiBi*aSat - _Topwb*(FreqPU + Speed)*PsiBr;
    if _Psi < fSatB
    then theB[7] := -aSat + 1
    else theB[7] := -aSat + 1 + fSatA*sqr(_Psi - fSatB) ;

    result := 0;
    for i := 1 to 7 do result := result + sqr(theB[i]);
    result := sqrt(result);
end;

```

```

var i : integer;
tempSatTerm : double;
IterationCount : integer;
local_Top : double;
begin
theMatrix := Matrix of 7x7 doubles
theB      := Vector of 7 doubles;
theX      := Vector of 7 doubles;
_wb       := 2*Pi*NomFreqHz;
_Topwb    := fXr/fRr;
_Top      := fXr/fRr/_wb;
_Psi := 0;
theX[ 1] := aIFr ;
theX[ 2] := aIFi ;
theX[ 3] := PsiBr;
theX[ 4] := PsiBi;
theX[ 5] := aIBr ;
theX[ 6] := aIBi ;
theX[ 7] := aSat ;
LOCAL_PopulateAlgebraicAndStatesAndCalculatePsi;

IterationCount := 0;
While (LOCAL_PopulateMismatchBAndNorm > 1E-10) and (IterationCount < 20) do begin
Inc(IterationCount);
// Calculate Jacobian
LOCAL_PopulateJacobian;
// Solve Ax=b
theMatrix.Factor;
theMatrix.ForwardBackward(theB);
For i := 1 to 7 Do begin
theX[i] := theX[i] - theB[i];
end;
LOCAL_PopulateAlgebraicAndStatesAndCalculatePsi;
end;
// Do algebraic update of the states PsiBr and PsiBi
end;

```

9.10 Differential Equations

```

procedure CalculateDerivates;
Begin
    _wb      := 2*Pi*NomFreqHz;
    _Top     := fXr/fRr/_wb;
    dotPsiFr := (fXm*aIFr - aSat*PsiFr)/_Top + _wb*(FreqPU - Speed)*PsiFi;
    dotPsiFi := (fXm*aIFI - aSat*PsiFi)/_Top - _wb*(FreqPU - Speed)*PsiFr;
    dotSpeed := 1/(2*fH)*(ElectricalTorqueObject(PsiFr, PsiFi, PsiBr, PsiBi) - MechanicalTorqueObject(Speed, SpeedDelayed));
    dotSpeedDelayed := 1/Tpgas*(Speed - SpeedDelayed); // Added in June 15, 2023 patch of PowerWorld Simulator 23
end;

```

9.11 Initialization

```

procedure SetInitialLoads(local_PSystem, local_QSystem, local_V : Double;
                           out local_UsedPSystem, local_UsedQSystem : Double);
Var local_W0, local_TElectObject : Double;
    local_Speed, local_PsiFr, local_PsiFi, local_PsiBr, local_PsiBi : double;
    local_SpeedMult : double;
    local_Telec : double;
Begin
    local_UsedPSystem := 0;
    local_UsedQSystem := 0;
    // Set the MVABase of the load
    If fObjectMVABase <= 0 then begin
        if fLoadFactor > 0
        then fObjectMVABase := local_PSystem*SystemMVABase/fLoadFactor
        else fObjectMVABase := local_PSystem*SystemMVABase/0.8;
    end;
    If (Inservice) and (local_V > 0) Then Begin
        // convert to initial MVABase
        fPInitialObject := local_PSystem*SystemMVABase/fObjectMVABase;
        InitializeFluxCurrentSpeedSat(fPInitialObject, local_V,
                                       fQInitialObject, local_Speed, local_PsiFr, local_PsiFi, local_PsiBr, local_PsiBi);
        local_Telec := ElectricalTorqueObject(local_PsiFr, local_PsiFi, local_PsiBr, local_PsiBi);
        // MUST set fTNomObject := 1 initially!
        fTNomObject := 1; // set to 1 so that MechanicalTorqueObject(Speed) returns the multiplier on TNom
        local_SpeedMult := MechanicalTorqueObject(local_Speed, local_Speed); // SpeedDelayed = Speed at initialization
        fTNomObject := local_Telec / local_SpeedMult;

        local_UsedPSystem := fPInitialObject*fObjectMVABase/SystemMVABase;
        local_UsedQSystem := fQInitialObject*fObjectMVABase/SystemMVABase;
        // If it's NOT a component model, then explicitly include the CapBShunt as part of the load model
        if (not ModelIsAComponent) then Begin
            fBExtraObject := (local_UsedQSystem - local_QSystem)/Sqr(local_V) * SystemMVABase/fObjectMVABase;
            local_UsedQSystem := local_QSystem;
        End
        Else Begin
            fBExtraObject := 0;
        End;
    end;
end;

```

9.12 Calculation of Network Current Injections and Network Current Jacobian Entries

```

procedure CalculateNortonCurrentsObjectBase(MyVoltageE, MyVoltageF : double;
                                         out local_Irdq, local_Iidq : double);

var local_e, local_f, local_v : double;
    local_Vxd, local_Vyd, local_Vxq, local_Vyq : double;
begin
    local_e := MyVoltageE;
    local_f := MyVoltageF;
    local_V := sqrt( sqr(MyVoltageE) + sqr(MyVoltageF) );
    local_Vxd := (local_e * fCACHE_Edd - local_f*fCACHE_Eqd)/local_v;
    local_Vyd := (local_f * fCACHE_Edd + local_e*fCACHE_Eqd)/local_v;

    local_Vxq := (local_e * fCACHE_Edq - local_f*fCACHE_Eqq)/local_v;
    local_Vyq := (local_f * fCACHE_Edq + local_e*fCACHE_Eqq)/local_v;

    // Negative sign as compared to REGC_B type model because we're a load and not a generator
    //
    local_Irdq := - (local_Vxd*fGdObject - local_Vyd*fBdObject)
                  - (local_Vxq*fGqObject - local_Vyq*fBqObject);

    local_Iidq := - (local_Vxd*fBdObject + local_Vyd*fGdObject)
                  - (local_Vxq*fBqObject + local_Vyq*fGqObject);
end;

procedure CurrentInjection(local_BusVoltMag : double; local_BusPUDeltaFreq : Single;
                          MyVoltageE, MyVoltageF : double;
                          var local_Ir, local_Ii : double);
Var local_BaseConversionAndScalar, local_e, local_f, local_Irdq, local_Iidq : double;
    local_G, local_B : double;
begin
    local_BaseConversionAndScalar := FractionOfNominalStillOn * fObjectMVABaseOrg/ TTxPowerSystem(LoadObject.Owner).SBase;
    local_e := MyVoltageE;
    local_f := MyVoltageF;
    CalculateNortonCurrentsObjectBase(local_Irdq, local_Iidq);
    local_Ir := local_Ir + local_BaseConversionAndScalar*( local_Irdq );
    local_Ii := local_Ii + local_BaseConversionAndScalar*( local_Iidq );

    // Let's include the impedance directly in this instead of including it in the Y-Bus because it's just easier
    local_G := fGdObject + fGqObject;
    local_B := fBdObject + fBqObject + fBExtraObject;
    local_Ir := local_Ir + local_BaseConversionAndScalar*( (local_e*local_G - local_f*local_B) );
    local_Ii := local_Ii + local_BaseConversionAndScalar*( (local_e*local_B + local_f*local_G) );
end;

```

```

procedure CurrentInjectionDerivatives(local_BusPUDeltaFreq : Single; MyVoltageE, MyVoltageF : double;
                                     out dIr_de, dIr_df, dIi_de, dIi_df : double);
Var local_BaseConversionAndScalar : Double;
    local_e, local_f, local_v2, local_Irdq, local_Iidq : double;
    local_G, local_B : double;
begin
    local_BaseConversionAndScalar := FractionOfNominalStillOn * fObjectMVABaseOrg/ TTxPowerSystem(LoadObject.Owner).SBase;
    local_e := MyVoltageE;
    local_f := MyVoltageF;
    local_v2 := sqr(MyVoltageE) + sqr(MyVoltageF);
    CalculateNortonCurrentsObjectBase(local_Irdq, local_Iidq);
    dIr_de := + local_f/local_v2*( local_Iidq );
    dIr_df := - local_e/local_v2*( local_Iidq );
    dIi_de := - local_f/local_v2*( local_Irdq );
    dIi_df := + local_e/local_v2*( local_Irdq );

    // Now handle the derivatives with respect to the impedances
    local_G := fGdObject + fGqObject;
    local_B := fBdObject + fBqObject + fBExtraObject;
    dIr_de := + local_BaseConversionAndScalar*( + local_G );
    dIr_df := + local_BaseConversionAndScalar*( - local_B );
    dIi_de := + local_BaseConversionAndScalar*( + local_B );
    dIi_df := + local_BaseConversionAndScalar*( + local_G );
end;

```

9.12.1 Algebraic Update of Network Interface Variables

```

procedure UpdateNetworkInterfaceVariables(FreqPU : double);
begin
    // Network boundary equation. These are updated each timestep
    fCACHE_Edd := - ( + PsiFi + PsiBi)*FreqPU*fXm/fXr;
    fCACHE_Eqd := + ( + PsiFr + PsiBr)*FreqPU*fXm/fXr;
    fCACHE_Edq := - ( + PsiFr - PsiBr)*FreqPU*fnRatio*fXm/fXr;
    fCACHE_Eqq := + ( - PsiFi + PsiBi)*FreqPU*fnRatio*fXm/fXr;
end;

```