Steady State Analysis of Autonomous Wind Energy Conversion System Employing Induction Machines for Irrigation Purpose

Kanoj B, Arjun M
A B Raju and Satish Annigeri

Department of Electrical and Electronics Engineering,
B. V. B. College of Engineering and Technology,
HUBLI-580 031, KARNATAKA
abraju@bvb.edu

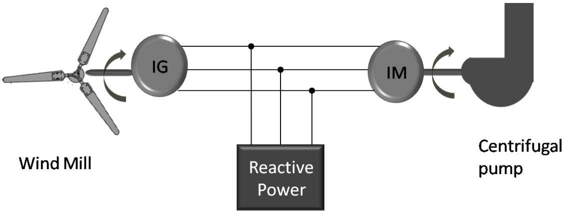
December 29, 2012

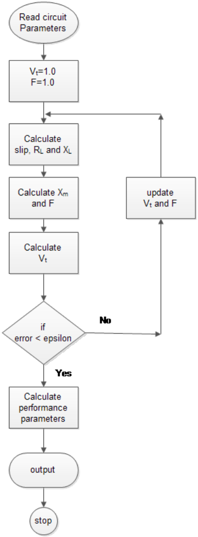
Overview

1. Objective of the Presentation
2. Block Diagram and Flowchart
3. Wind turbine and Induction Machine Specifications
4. Equivalent Circuit
5. Building Mathematical Model
6. Why Python and SciPy?
7. SciPy Code for solving nonlinear algebraic equations
8. Results
9. Conclusion
10. References

Objective of the presentation

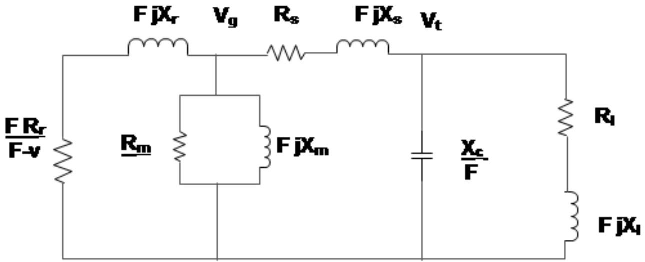
To use Scientific Python as a computational tool for steady state analysis of wind energy conversion system

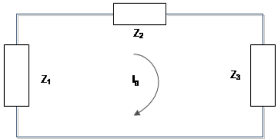

IG
IM
Wind Mill
Reactive Power
Centrifugal pump


Read circuit Parameters
V_t=1.0
F=1.0
Calculate slip, R_L and X_L
Calculate X_m and F
Calculate V_t
if error < epsilon
No
update V_t and F
Yes
Calculate performance parameters
output
stop

Wind Turbine and Induction Machine Specifications

Wind turbine specifications:
Swept area radius r = 1.27 m
cut-in wind speed v_c = 3 m/s
rated wind speed v_r = 15 m/s
furl-on wind speed v_f = 25 m/s
Induction Machine specifications
3 Phase, 50 Hz, 4 pole, 5 HP, 1450 rpm, 415 V
Stator resistance r_s = 5.2 Ω
Rotor Resistance r_r = 4.66 Ω
Stator and rotor reactances x_s and x_r = 2.46 Ω
Core resistance r_c = 746.40 Ω


F jXr
Vg
Rs
F jXs
Vt
F Rr
F-v
Rm
F jXm
Xc
F
Rl
F jXl


Z2
Z1
Z3

Building Mathematical Model

- Applying Kirchhoff's voltage law to the loop current I_g the resulting equation is

$$
\begin{aligned}
I_g Z_g &= 0 \\
Z_g &= Z_1 + Z_2 + Z_3 \\
Z_1 &= \frac{Z_r Z_m}{Z_r + Z_m} \\
Z_m &= \frac{F\, jX_m R_m}{R_m + F\, jX_m} \\
Z_2 &= R_s + F\, jX_s \\
Z_3 &= \frac{(R_l + F\, jX_l) jX_c}{R_l + F\, jX_l + jX_c}
\end{aligned}
$$

Substituting Z_1, Z_2 and Z_3, we get

$$\frac{-X_l X_c + jX_l \frac{R_l}{F}}{-FR_l + j(X_c - F^2 X_l)} + \frac{R_s}{F} + jX_s + \frac{-X_r X_m + jX_m \frac{R_r}{F-v}}{\frac{R_r}{F-v} + j(X_r + X_m)} = 0$$

- Its a complex non-linear algebraic equation and is solved for F and X_m by separating real and imaginary parts and equating them to zero separately.

Solving Non Linear Equations

- Non linear equations can be solved numerically either writing our own code in compiled languages like:
  1. C or C++
  2. Fortran

  or using the readily available routines in interpreter languages like:
  1. MATLAB
  2. Scilab and
  3. Octave
  4. Python

Why Python and Scipy?

- Free and open-source software, widely used, with a vibrant community.
- It has rich modules/libraries.
- Well thought out language, allowing to write very readable and well structured code.
- Capability of handling complex numbers and equations.
- Possibility of improving performance comparable to compiled languages
- Possibility of developing GUI application.

Why Python and Scipy?

1. SciPy is an Open Source library of scientific tools for Python.
2. It gathers a variety of high level science and engineering modules together as a single package.
3. Using numpy, scipy and matplotlib modules, one can think of replacing a well known proprietary software.

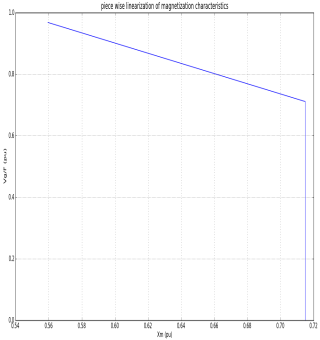
In this presented work, we have used the following routines from scipy.optimize

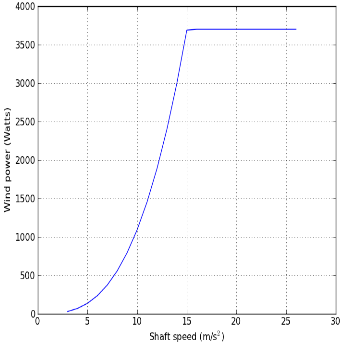
- fsolve()
- newton_krylov() and
- matplotlib.pyplot

SciPy Code For Solving Non-Linear Equations

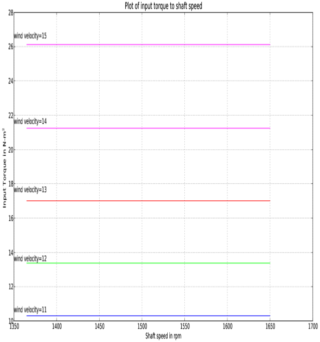
```
def funct(x):
    Xmg = x[1]
    F   = x[0]
    Zl  = (Rl/F) + (1j*Xl)
    Zc  = (-1j*Xc)/(F**2)
    Zcc = (Zl * Zc ) /(Zl + Zc)
    Zsg = (Rs/F) + (1j*Xs)
    Zrg = (Rr/(F-v)) + (1j*Xr)
    Zmg = ((F/Rc) + (1/(1j* Xmg)))**(-1)
    Zaa = (Zmg * Zrg)/(Zmg + Zrg)
    g   = Zaa + Zsg + Zcc
    out = [real(g)]
    out.append(imag(g))
    return array(out)
D = fsolve(funct,array([1.0, 1.0]))
```

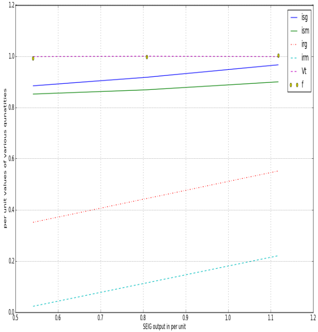
Results


piece wise linearization of magnetization characteristics
1.0
0.8
0.6
0.4
0.2
0.0
Vg/F (pu)
0.54
0.56
0.58
0.60
0.62
0.64
0.66
0.68
0.70
0.72
Xm (pu)

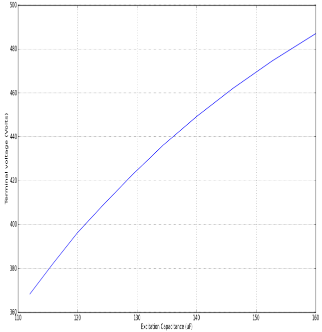

4000
3500
3000
2500
2000
1500
1000
500
0
Wind power (Watts)
0
5
10
15
20
25
30
Shaft speed (m/s^2)

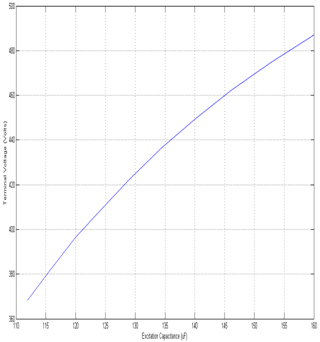
Results


Plot of input torque to shaft speed
wind velocity=15
wind velocity=14
wind velocity=13
wind velocity=12
wind velocity=11
Input Torque in N-m²
Shaft speed in rpm
28
26
24
22
20
18
16
14
12
10
1350
1400
1450
1500
1550
1600
1650
1700


isg
ism
irg
irm
Vt
f
per unit values of various qunatities
SEIG output in per unit
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2

Results of Python against Matlab





Conclusion

- Python code executes faster than the corresponding Matlab code by a factor of about 7.197474264.
- Ten iterations of Matlab code using fsolve() took 1.8346 s, whereas Python fsolve() took 0.254894971848 s, and newton_ krylov() took 1.17562890053 s.
- newton_ krylov() function of python executes faster than fsolve() by a factor of 4.612209068

L. Shridhar, Bhim Singh, C. S . Jha and B.P. Singh, "Analysis of Self Excited Induction Generator feeding Induction Motor", *Analysis of Self Excited Induction Generator feeding Induction Motor*,pp. p. 390 - 396, June, 1994.

THANK YOU