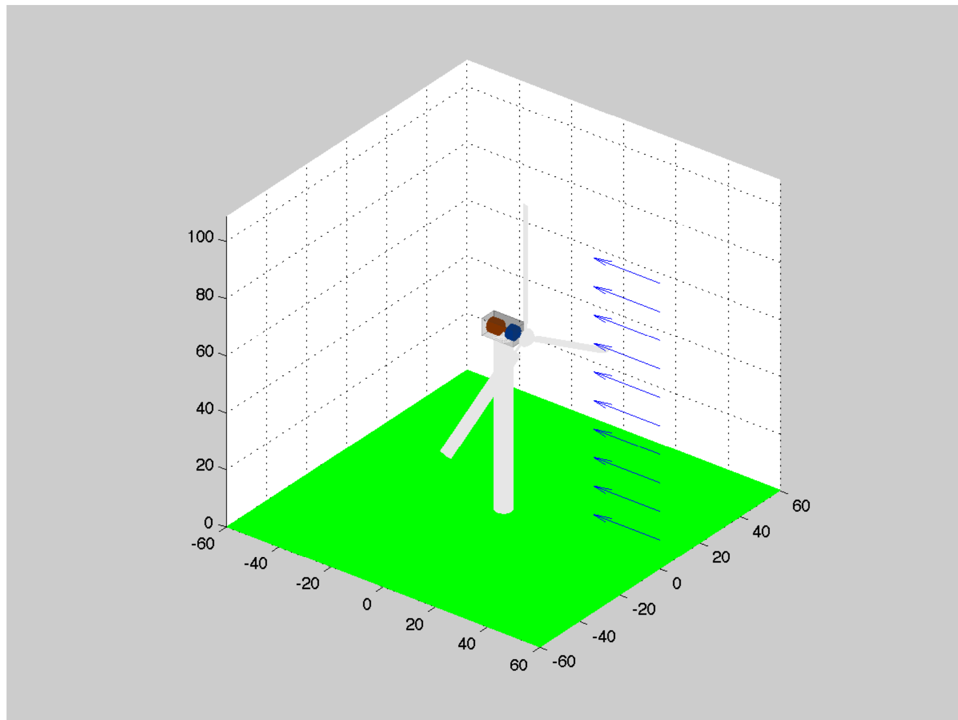
Princeton SATELLITE SYSTEMS

Wind Turbine Control Toolbox v2.0



This software described in this document is furnished under a license agreement. The software may be used, copied or translated into other languages only under the terms of the license agreement.

Wind Turbine Control Toolbox

Copyright © 2009-2010 by Princeton Satellite Systems, Inc. All rights reserved.

MATLAB is a trademark of the MathWorks.

All other brand or product names are trademarks or registered trademarks of their respective companies or organizations.

Printing History:

January 28, 2009 First Printing v1.0

Princeton Satellite Systems, Inc.
33 Witherspoon Street
Princeton, New Jersey 08542

Technical Support/Sales/Info: http://www.psatellite.com

CONTENTS

Contents iii

1 Introduction **1**

1.1 Organization . . . 1
1.2 Requirements . . . 2
1.3 Installation . . . 2
1.4 Getting Started . . . 2

2 Getting Help **5**

2.1 MATLAB Help . . . 5
2.2 FileHelp . . . 6
2.2.1 Introduction . . . 6
2.2.2 The List Pane . . . 7
2.2.3 Edit Button . . . 7
2.2.4 The Example Pane . . . 7
2.2.5 Run Example Button . . . 7
2.2.6 Save Example Button . . . 7
2.2.7 Help Button . . . 7
2.2.8 Quit . . . 7
2.3 Searching in File Help . . . 7
2.3.1 Search File Names Button . . . 8
2.3.2 Find All Button . . . 8
2.3.3 Search Headers Button . . . 8
2.3.4 Search String Edit Box . . . 8

2.4 Technical Support 8

3 Fundamentals **9**

3.1 Classes 9

3.2 Code Conventions 11

4 Simulation **13**

4.1 Introduction to `WTSim` 13

4.2 Getting Started 14

4.2.1 Examine a Block 14

4.2.2 View / Change the Values of States and Parameters 15

4.2.3 Load / Save a Setup File 15

4.2.4 Run a Simulation 16

4.2.5 Analyze the Results 16

4.3 Model Functions 17

4.3.1 Usage Formats 17

4.3.2 Defining the Model I/O 17

4.3.3 Model Initialization 19

4.3.4 Model Update 20

5 Airfoil Models **23**

5.1 Airfoil Functions 23

5.1.1 Introduction 23

5.1.2 Loading Data 23

5.1.3 Using the Data 24

5.2 An Example 24

5.3 Coefficients from Shapes 26

6 Blade Models **27**

6.1 Introduction 27

6.2 Vertical Axis Wind Turbine Blade Models 27

6.2.1 Torque Model 27

6.3 Horizontal Axis Wind Turbine Blade Models . . . 27
6.3.1 Torque Model . . . 27
6.4 Double Streamtube Models . . . 28

7 Control Design **33**
7.1 Introduction . . . 33
7.2 Maximum Power Tracking . . . 33
7.3 Generator Control . . . 36
7.4 Control Design Functions . . . 36
7.5 HAWT Demo . . . 37
7.6 DFIG Control . . . 37
7.7 VAWT Demo . . . 40
7.7.1 Pitch Control Algorithms . . . 42

8 Estimation **43**
8.1 Overview . . . 43
8.2 Fixed Gain Estimators . . . 44
8.3 Variable Gain Estimators . . . 44
8.4 Extended Kalman Filter . . . 45
8.5 Continuous Discrete Extended Kalman Filter . . . 45
8.5.1 Introduction . . . 45
8.6 Unscented Kalman Filter . . . 46

9 Electrical Models **47**
9.1 Introduction . . . 47
9.2 Circuit Element Models . . . 47
9.2.1 Introduction . . . 47
9.2.2 Diode . . . 47
9.2.3 Bridge Rectifier . . . 48
9.2.4 Capacitor . . . 48
9.2.5 Inductor . . . 48

9.2.6 Grid Model . . . 48
9.2.7 Transformer . . . 49
9.2.8 Matrix Converter . . . 49
9.2.9 Three Phase Rectifier . . . 50
9.3 Utility . . . 50
9.3.1 Phases . . . 50

10 Generator Models 51
10.1 Introduction . . . 51
10.2 Electrical and Mechanical Degrees . . . 51
10.3 Direct Quadrature Model . . . 52
10.4 Per Unit Normalization . . . 52
10.5 Permanent Magnet Generator Model . . . 53
10.6 Doubly Fed Induction Generator . . . 55

11 Mechanical 57
11.1 Overview . . . 57
11.2 Mechanism Models . . . 57
11.2.1 Introduction . . . 57
11.2.2 Bearing . . . 57
11.2.3 Coupling . . . 58
11.2.4 Rectangular Beam . . . 58
11.2.5 Gear Box . . . 58
11.2.6 Gears . . . 58
11.3 Joint Calculators . . . 58
11.3.1 Introduction . . . 58
11.3.2 Welds . . . 58
11.3.3 Interference Fits . . . 59
11.4 Blade Static Approximations . . . 59
11.4.1 Introduction . . . 59
11.4.2 Drag Force . . . 60

11.4.3 Hinge Moment 60
11.4.4 Moment of Inertia 60
11.4.5 Bending Stress 60
11.4.6 Bearing Distances 60

12 Multibody Models **61**
12.1 Introduction 61
12.2 Background 61
12.2.1 Tree 61
12.2.2 Hinges 61
12.2.3 Computations 61
12.3 Example 62

13 Utilities **65**
13.1 Introduction 65
13.2 Reynold's Number 65
13.3 DrawHAWT 66
13.4 DrawVAWT 66
13.5 LiftAndDragCoeff 67
13.6 PowerFromActuatorDisk 67

14 WindData **69**
14.1 Introduction 69
14.2 Wind Data 69

15 Wind Models **73**
15.1 Introduction 73
15.2 Dynamical Models 73
15.2.1 WindspeedHours 73
15.2.2 WindDeterministic 74
15.2.3 WindStochastic 74
15.2.4 WindAdmittance 74

15.2.5 Wind 76

Bibliography **77**

CHAPTER 1

INTRODUCTION

This chapter shows you how to install the Wind Turbine Control Toolbox (WCT) and how it is organized.

1.1 Organization

The Wind Turbine Control Toolbox (WCT) provides a suite of MATLAB functions designed to assist engineers with the design, simulation and performance analysis of wind turbines.

The toolbox code is organized into several different folders, described in the following table.

Table 1-1. Wind Turbine Control Toolbox

Folder	Functionality
AirfoilData	Data for airfoils and functions to read in airfoil data
Blade	Functions to model individual blade aerodynamics.
Common	Engineering constants database, control design and analysis tools, general coordinate transformation routines, graphics and plot utilities, vector math operations, trigonometric operations, Newton-Raphson method, Runge-Kutta integration, Simplex, and probability analysis tools.
ControlDesign	Example scripts that illustrate some specific control design techniques.
Demos	Demonstration scripts.
DeviceDesign	Functions for designing hardware.
DynamicModels	Models of dynamical components.
Electrical	Models of circuit components and electrical devices.
Estimation	Estimator functions.
Generator	Models of generators.
Mechanical	Mechanical design functions.
Multibody	A topological tree multi-rigid-body model
Sim	The wind turbine simulation. Includes a GUI for setting up and running simulations, and a directory structure for organizing models.
Utility	Miscellaneous functions for general analysis, design and modeling tasks (e.g. torque and power relationships.
WindData	Several different shape files with a function to load and plot the data.
WindModels	Includes a deterministic and a stochastic wind model.

The “Common” folder contains a large code base that provides the core functionality for other PSS software products, including the Aircraft Control Toolbox and the Spacecraft Control Toolbox.

1.2 Requirements

MATLAB 7.0 at a minimum is required to run all of the functions. Your monitor should have a resolution of at least 1024 by 768 pixels to see all of the GUIs.

1.3 Installation

The product can be downloaded in the form of a zip-file archive from the PSS website: http://www.psatellite.com.

It is recommended that you store a copy of the zip-file for future reference.

Installation involves 3 steps:

1. Unzip the archive
2. Copy the "WCT" folder to a location of your choice
3. Add the WCT folders to your MATLAB path.

You can keep the PDF documentation and the software anywhere you wish. There is no "installer" application to do the copying for you.

Once you have the software copied to your hard drive, the final step is to add the WCT folders to your MATLAB path. We recommend using the supplied function `PSSSetPaths.m` instead ofMATLAB's path utility. From the MATLAB prompt, cd to your WCT folder and then run PSSSetPaths. For example:

```
1 >> cd /Users/me/WCT
2 >> PSSSetPaths
```

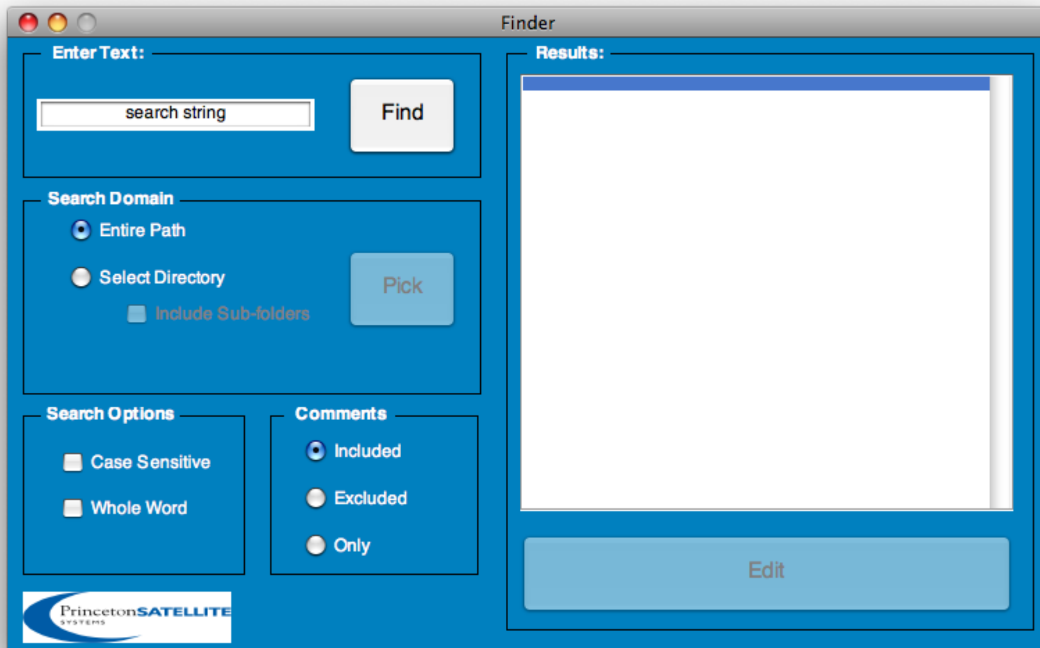
This will set all of the paths for the duration of the session, with the option of saving the new path for future sessions.

1.4 Getting Started

The `FileHelp` function, discussed in more detail in the next chapter, provides a graphical interface to the MATLAB function headers. You can peruse the functions by folder to get a quick sense of your new product's capabilities and search the function names and headers for keywords. `FileHelp` provides the best way to get an overview of the Wind Turbine Control Toolbox.

The `Finder` GUI, shown on the next page, is another handy function for searching for information in the toolbox. You can search for instances of keywords in the entire body of functions and demos, not just the help comments. You can use this function with any toolboxes, not just your PSS toolboxes, since this actively searches the files every time instead of using a parsed version of the headers the way `FileHelp` does. Consequently, it is a little slower to use, but you can use it with your own function libraries too.

The `Finder` function has options for searching the entire path or a selected directory. The subfolders of a higher-level directory can be included or not. The Pick button brings up a file selection dialog where you can navigate to your desired directory. The search can be case sensitive and you can select whole word matching. You can search on just file help comments, or include or exclude them. For example, you can find all functions and demos that actually use the function `PIDMIMO` by searching with comments excluded. Once your search results are displayed in the Results window, you can open any file by clicking the Edit button.


Finder
Enter Text:
search string
Find
Search Domain
Entire Path
Select Directory
Pick
Include Sub-folders
Search Options
Case Sensitive
Whole Word
Comments
Included
Excluded
Only
Results:
Edit
PrincetonSATELLITE SYSTEMS

CHAPTER 2

GETTING HELP

This chapter shows you how to use the help systems built into PSS Toolboxes. There are several sources of help. First, there is the MATLAB help system which prints help comments for inidividual files and lists the contents of folders. Then, there are special help utilities built into the PSS toolboxes: one is the file help function, the second is the demo functions and the third is the graphical user interface help system. Additionally, you can submit technical support questions directly to our engineers via email.

2.1 MATLAB Help

You can get help for any function by typing

```
>> help functionName
```

For example, if you type

```
>> help C2DZOH
```

you will see the following displayed in your MATLAB command window:

```
-----------------------------------------------------------------
    Create a discrete time system from a continuous system
    assuming a zero-order-hold at the input.
    Given
    .
    x = ax + bu

    Find f and g where
     x(k+1) = fx(k) + gu(k)

-----------------------------------------------------------------
    Form:
    [f, g] = C2DZOH( a, b, T )
-----------------------------------------------------------------
    ------
    Inputs
    ------
    a                   Plant matrix
    b                   Input matrix
    T                   Time step
    -------
    Outputs
    -------
```

```
24      f                       Discrete plant matrix
25      g                       Discrete input matrix
26
27  ------------------------------------------------------------------
```

All PSS functions have the standard header format shown above. Keep in mind that you can find out which folder a function resides in using the MATLAB command `which`, i.e.

```
>> which C2DZOH
/Software/Toolboxes/Aerospace/Common/Control/C2DZOH.m
```

When you want more information about a folder of interest, remember that you can get a list of the contents in any directory by using the `help` command with a folder name. The returned list of files is organized alphabetically. For example,

```
>> help GeneratorModels
```

2.2 FileHelp

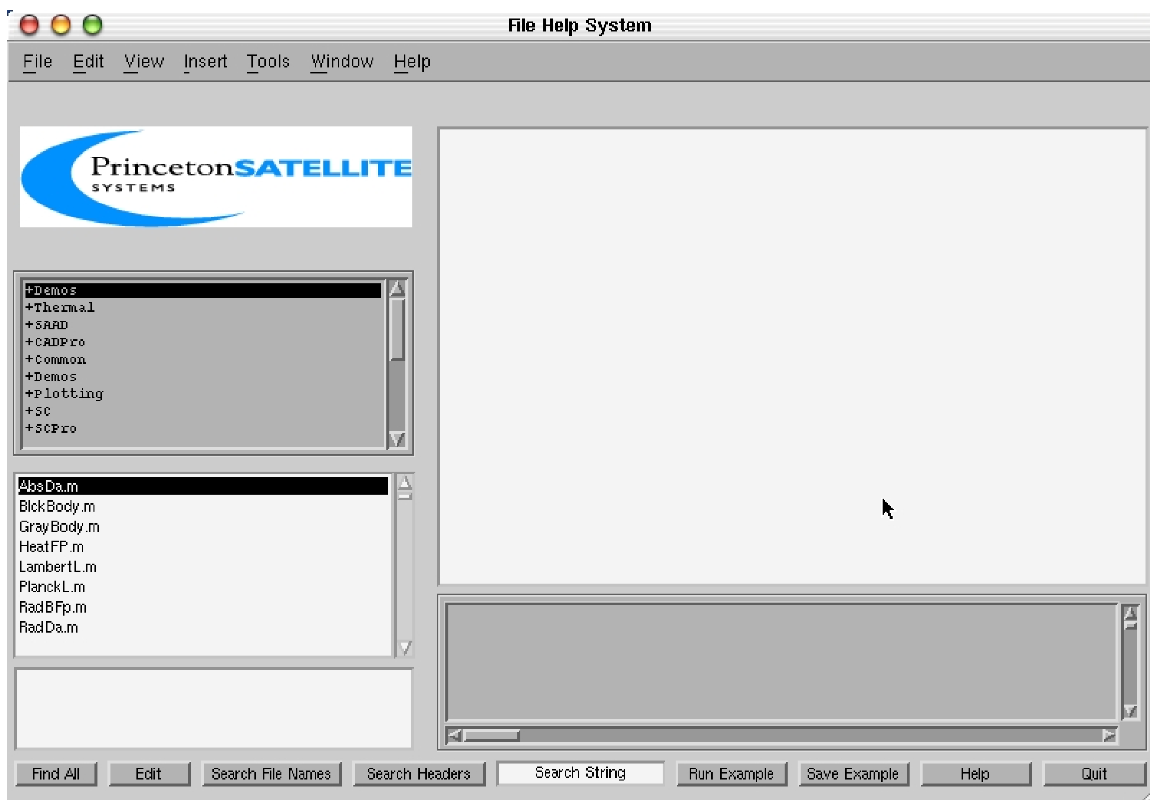
2.2.1 Introduction

When you type

```
FileHelp
```

the FileHelp GUI appears.

Figure 2-1. The file help GUI



There are five main panes in the window. On the left hand side is a display of all functions in the toolbox arranged in the same hierarchy as the PSSToolboxes folder. Scripts, including most of the demos, are not included. Below the hierarchical list is a list in alphabetical order by product. On the right-hand-side is the header display pane. Immediately below the header display is the editable example pane. To its left is a template for the function. You can cut and paste the template into your own functions.

2.2.2 The List Pane

If you click a file in the alphabetical or hierarchical lists, the header will appear in the header pane. This is the same header that is in the file. The headers are extracted from a .mat file so changes you make will not be reflected in the file. In the hierarchical list, any name with a + or - sign is a folder. Click on the folders until you reach the file you would like. When you click a file, the header and template will appear.

2.2.3 Edit Button

This opens the MATLAB edit window for the function selected in the list.

2.2.4 The Example Pane

This pane gives an example for the function displayed. Not all functions have examples. The edit display has scroll bars. You can edit the example, create new examples and save them using the buttons below the display. To run an example, push the Run Example button. You can include comments in the example by using the percent symbol.

2.2.5 Run Example Button

Run the example in the display. Some of the examples are just the name of the function. These are functions with built-in demos. Results will appear either in separate figure windows or in the Matlab Command Window.

2.2.6 Save Example Button

Save the example in the edit window. Pushing this button only saves it in the temporary memory used by the GUI. You can save the example permanently when you Quit.

2.2.7 Help Button

Opens the on-line help system.

2.2.8 Quit

Quit the GUI. If you have edited an example, it will ask you whether you want to save the example before you quit.

2.3 Searching in File Help

2.3.1 Search File Names Button

Type in a function name in the edit box and push Search File Names.

2.3.2 Find All Button

Find All returns to the original list of the functions. This is used after one of the search options has been used.

2.3.3 Search Headers Button

Search headers for a string. This function looks for exact, but not case sensitive, matches. The file display displays all matches. A progress bar gives you an indication of time remaining in the search.

2.3.4 Search String Edit Box

This is the search string. Spaces will be matched so if you type attitude control it will not match attitude control (with two spaces.)

2.4 Technical Support

Contact wctsupport@psatellite.com for email technical support. You can also call 01-609-279-9606.

CHAPTER 3

FUNDAMENTALS

This chapter gives you some basic information about the toolbox, including the classes and code conventions.

3.1 Classes

The Wind Turbine Control Toolbox defines and makes use of the `statespace` class.

The `statespace` class defines a linear statespace dynamic system. The system can be either continuous or discrete. A continuous system is of the form:

$$\dot{x} = Ax + Bu \qquad (3\text{-}1)$$
$$y = Cx + Du \qquad (3\text{-}2)$$

where x is the state vector, A is the state transition matrix, and B is the control effect matrix. Each system type is denoted with a unique string identifier. Continuous systems are denoted with "s".

For a discrete system, there are two different ways to write the state evolution. The first method is shown below, which we call the "z" method:

$$x_{k+1} = Ax_k + Bu_k \qquad (3\text{-}3)$$
$$y_k = Cx_k + Du_k \qquad (3\text{-}4)$$

The other discrete method uses the delta operator. This is termed the "delta" method:

$$x_{k+1} = x_k + Ax_k + Bu_k \qquad (3\text{-}5)$$
$$y_k = Cx_k + Du_k \qquad (3\text{-}6)$$

A continuous statespace system can be converted to discrete-time by using the `C2DZOH` or `C2DelZOH` methods, which use a zero-order-hold on the input over a specified sampling time. The conversion from continuous to discrete time changes the A and B matrices only. The same C and D matrices are valid for both continuous and discrete domains.

To define a `statespace` class, you must at least specify the A, B, C matrices. If the D matrix is not supplied it is set to all zeros. In addition, you may also supply a name for the system, individual names for the states, inputs, and outputs, the system type, and the time step (if the system is discrete). The "help" information on statespace.m explains how to create an `statespace` class object.

```
>> help statespace
 -----------------------------------------------------------------------------
    Create a state space object. Everything after c is optional.
 -----------------------------------------------------------------------------
    Form:
    g = statespace( a, b, c, d, name, states, inputs, outputs, sType, dT )
 -----------------------------------------------------------------------------


    ------
    Inputs
    ------
    a                          State transition matrix
    b                          State input matrix
    c                          State output matrix
    d                          State feedthrough matrix
    name     (1,:)             Name of the system
    states   (:,:)   or {:}    State  names
    inputs   (:,:)   or {:}    Input names
    outputs  (:,:)   or {:}    Outputs
    sType    (1,:)             's', 'z', 'delta'
    dT       (1,1)             Time step

    -------
    Outputs
    -------
    g                     (:)     Plant
                    g.a       State transition matrix
                    g.b       State input matrix
                    g.c       State output matrix
                    g.d       State feedthrough matrix
                    g.n       Number of states
                    g.nI      Number of inputs
                    g.nO      Number of outputs
                    g.states  Names of states
                    g.inputs  Names of inputs
                    g.outputs Names of outputs
                    g.sType   's', 'z', 'delta'
                    g.dT      Time step

 -----------------------------------------------------------------------------
```

You can view the methods associated with the `statespace` class by typing:

```
>> methods statespace

Methods for class statespace:

and           connect       get           getsub        mtimes        series
    statespace
close         eig           getabcd       isempty       plus          set
```

Assume you have a `statespace` class named `g`. You can extract the A, B, C, D matrices from the class by typing:

```
>> [a,b,c,d] = getabcd(g);
```

Similarly, you can extract individual matrices or other information using the `get` method.

```
>> b = get(g,'b');
>> stateNames = get(g,'states');
```

3.2 Code Conventions

It is important to follow consistent code conventions to make the code easy for other people to understand and use. The scripts and functions supplied with this toolbox are always supplied with a descriptive header that provides usage syntax and a list of inputs and outputs. You can type

```
>> help FUNCTION
```

for any function to view the header.

When naming variables, we strive to use meaningful names. We also follow the convention:

```
word1Word2Word3
```

where the beginning of each word after the first is capitalized. If a word is abbreviated the first letter is not capitalized. For example:

```
rPM
```

is revolutions per minute.

Almost all function names in WCT begin with a capital letter to distinguish them from variables. The only exceptions are class methods, such as `get` and `plus`, for example. These method names overload built-in MATLAB functions for other class methods, and therefore must be all lower case.

Many functions in the Wind Turbine Control Toolbox can be executed with no inputs, even when inputs are required. If an input is required but not provided, the function may use its own default value. You can see what the default values are by opening the function and examining the lines of code that immediately follow the help comments at the top of the file. For example, consider the `WindspeedHours.m` function. We see from the help header that it is called as follows:

```
%   Form:
%   [w, v] = WindspeedHours( kRef, hRef, cRef, h, v, dV, u )
```

This function takes 7 inputs. Examining the file, we see that if no inputs are provided, it uses its own set of default values:

```
% Demo
%-----
if( nargin < 1 )
    disp('Wind speed hours in Lubbock, TX, USA');
    kRef = 2.01;
    cRef = 12.48; % 5.5791
    hRef = 30;
    WindspeedHours( kRef, hRef, cRef, 60, linspace(4,36), 0.5, 'mph' );
    clear w;
    return
end
```

CHAPTER 4

SIMULATION

This chapter describes how to use the WTSim GUI to build and run your own wind turbine simulations.

4.1 Introduction to WTSim

Wind turbine simulations can be performed using the WTSim GUI. A screenshot is shown in Figure 4-1 on the following page. The purpose of this GUI is to provide a general, yet structured framework for conducting simulation and analysis of various wind turbine configurations.

The GUI is divided into 5 main panels:

- *Block Diagram*
- *Block Info*
- *Setup File*
- *Simulation*
- *Plot Time History*

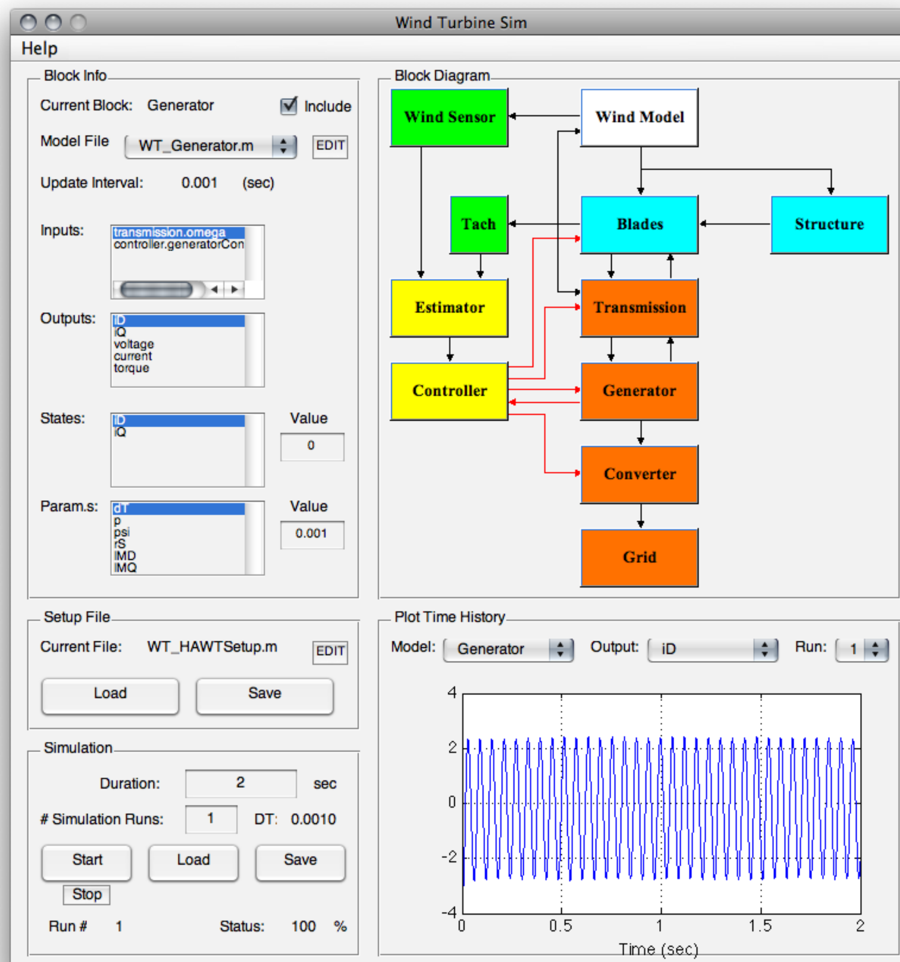
The *Block Diagram* panel shows a fixed set of blocks that illustrates the general data flow between common wind turbine components and models. Clicking on a block will select it. Information about the selected block is shown in the *Block Info* panel.

The *Setup File* panel allows you to load an existing setup file into the GUI, or save the current settings to a new setup file. The setup file includes the following data for each block: the name of the model function to be used, the values of all parameters, and the initial values of all states.

The *Simulation* panel enables you to select a time duration to simulate, and specify a number of runs. Multiple runs can be useful for conducting Monte Carlo simulations, where performance may be sensitive to random parameters, such as stochastic wind profiles. From this panel, you can start and stop the simulation, and view the current run number and status of each run while the simulation executes. You can also save the resulting data to a file, and load previously saved data.

The *Plot Time History* panel enables you to select a model and an output of that model for plotting. In addition, after a simulation run, all of the time history data is saved to the workspace in the variable name data, so that you can conduct further analysis.

Figure 4-1. The `WTSim` GUI



This chapter does not discuss the simulation models. Those are discussed in subsequent chapters.

4.2 Getting Started

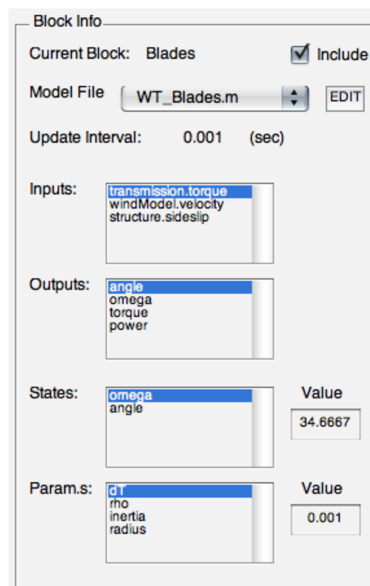
This section briefly describes how to use the `WTSim` GUI and run a simulation. The toolbox is provided with a built-in simulation for a horizontal axis wind turbine (HAWT). When the GUI is initialized, the setup file `WT_HAWTSetup.m` is automatically loaded.

Type `WTSim` in MATLAB to initialize the GUI.

4.2.1 Examine a Block

Click on a block in the *Block Diagram* panel to select it. This causes the *Block Info* panel to update, showing the data associated with that block. An example screenshot of the *Block Info* panel is shown in Figure 4-2 on the next page, here shown with the Blades block selected.

Figure 4-2. *Block Info* Panel



From within the *Block Info* panel, you will see that the display for the “Current Block” is updated to reflect the name of the block you just selected. You can also see which model function is associated with this block. A unique model function is supplied for each block for the built-in HAWT simulation. These blocks can serve as templates for generating your own blocks.

For the model function that you have selected, the corresponding names of the model’s inputs, outputs, states and parameters are displayed. All of these names are obtained automatically by the GUI, by calling the model function with no inputs.

4.2.2 View / Change the Values of States and Parameters

The values for states and parameters can be adjusted from the GUI. The original values that appear are specified inside the setup file. The values for the states represent the initial values of those states. The values of the parameters are assumed to be fixed inside the model for the duration of the simulation.

Every model must be supplied with a parameter named “`dT`”, to define its timestep. If it is missing, an error message will be displayed to warn the user.

4.2.3 Load / Save a Setup File

When the GUI is initialized, it executes the default setup file `WT_HAWTSetup.m`. This defines the model function to be used for each block, and sets the values for all parameters and initial states. You can reload the setup file at any time by pressing the “Load” button from the *Setup File* panel, and selecting `WT_HAWTSetup.m`. This will cause some output to print to the screen, displaying the dynamic properties of the system at the current operating point, and the eigenvalues of the closed-loop system with generator control.

To see how the initial state values and parameters were computed in the setup file, press the “EDIT” button inside the *Setup File* panel. You will notice that several parameters and some initial states are defined to have common values. This is important to remember when changing values manually from the GUI.

Any changes made to parameters or initial state values can be saved to a mat-file. These mat-files can also be loaded into the GUI in the future, in the same way that text-based setup files can be loaded.

4.2.4 Run a Simulation

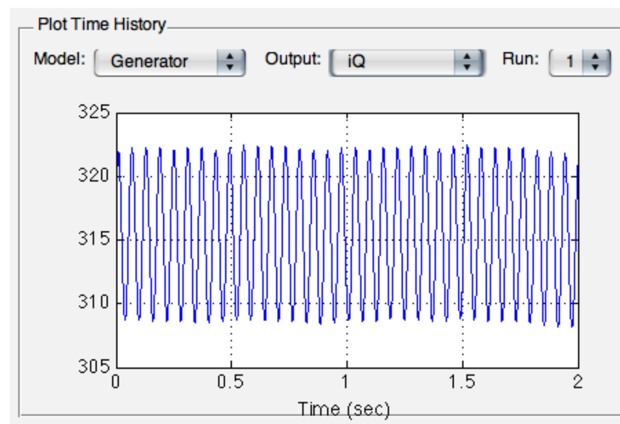
Now find the *Simulation Panel* in the bottom-left corner of the GUI. You will see the simulation timestep is 0.001 seconds (1 ms). This value is computed automatically as the greatest common divisor of all the individual model timestep values, with the number of significant digits limited to 4. In other words, the smallest possible timestep is 0.1 ms[1].

If you press the start button right away, you will first be prompted to enter a duration. Enter a small duration, on the order of a few seconds, to run a short simulation. Next press the start button and watch as the status progresses to 100%.

4.2.5 Analyze the Results

The results are displayed in the *Plot Time History* panel. An example is shown in Figure 4-3 on the following page. Here we have selected the Generator model and the output, `iQ`, which is the quadrature current.

Figure 4-3. Time History of the Generator Quadrature Current



The time histories of all outputs are also stored in the workspace under the names, `data` and `data2`. This enables you to conduct further analysis of the data outside of the GUI. The variable `data` has the following fields:

```
>> data
data =
          blades: [4x2001 double]
       windModel: [2x2001 double]
      windSensor: [2x2001 double]
       structure: [1x2001 double]
      controller: [8x2001 double]
       estimator: [3x2001 double]
            tach: [1x2001 double]
       generator: [5x2001 double]
    transmission: [2x2001 double]
       converter: [1x2001 double]
            grid: [2x2001 double]
```

As you can see, each field represents one of the simulation blocks. The contents of each field is a matrix of size $N_{out} \times N_{time}$, where N_{out} is the number of outputs for that block, and N_{time} is the number of time points for the entire simulation.

The variable `data2` contains the same simulation output data, but in a different form of data structure.

[1] If necessary, this can be changed by finding and changing the definition of `sim.sigDigits` in the `WTSim.m` file.

```
>> data2
data2 =
          Blades: [1x1 struct]
       WindModel: [1x1 struct]
      WindSensor: [1x1 struct]
       Structure: [1x1 struct]
      Controller: [1x1 struct]
       Estimator: [1x1 struct]
            Tach: [1x1 struct]
       Generator: [1x1 struct]
    Transmission: [1x1 struct]
       Converter: [1x1 struct]
            Grid: [1x1 struct]
               t: [1x2001 double]
```

Once again, each field contains the output for a specific model. In this case, however, the contents of each field is another data structure that contains that model's outputs organized by the output name. Here is an example:

```
>> data2.Blades
ans =
     angle: [1x2001 double]
     omega: [1x2001 double]
    torque: [1x2001 double]
     power: [1x2001 double]
```

4.3 Model Functions

In WTSim, each block represents a distinct, independent model of some component of the wind turbine. In order to simulate a block, you must assign a "model function" to it. A "model function" is simply a MATLAB function (m-file) that models the behavior of that block.

4.3.1 Usage Formats

Each model function follows a prescribed format. View the help header of any model function to see the different forms of usage. They are:

```
%   Forms:
%
%     ioData = WT_Blades;                       % Obtain i/o data
%     output = WT_Blades( input, time );        % Update with current sim time
%     output = WT_Blades( input, -1 );          % initialize
```

When called with no inputs, the function returns a data structure that details all of its expected inputs, and the outputs that it provides. When called with an `input` data structure and a non-zero time, the model updates. When a model updates, it takes in the `input` data structure, and returns an `output` data structure. If the model has dynamics, then an update also means that it integrates over its timestep from its current state to a new state.

4.3.2 Defining the Model I/O

The model I/O (input/output) information is returned whenever the function is called with no inputs. This is accomplished by using the MATLAB `nargin` command. This usage is needed by the GUI in order to properly display the inputs, outputs, states and parameters of the model. In addition, it can be used at the command line in order to quickly see the I/O data of any model. Here is an example of how to do this with the blades model:

```
>> [inputs, outputs, states, params] = ParseIOData( WT_Blades, 'Blades' )
inputs =
```

```
    'transmission.torque'
    'windModel.velocity'
    'structure.sideslip'
outputs =
    'angle'
    'omega'
    'torque'
    'power'
states =
    'omega'
    'angle'
params =
    'dT'
    'rho'
    'inertia'
    'radius'
```

Note that the user is free to specify whatever names she wishes here, independent of what the update portion of the code may be doing. It is therefore the responsibility of the user to ensure that these input and output names defined here do in fact match the names that are used in the update portion of the function. The advised coding practice is to

1. Identify the expected inputs and outputs,
2. write this I/O portion of the function accordingly,
3. write the update portion of the function,
4. check for consistency, and return to step 2. if necessary.

An example taken from the `WT_Blades.m` model function is shown below.

Listing 4.1. Example of Defining the Model I/O *WT_Blades.m*

```
function output = WT_Blades( input, time )

% HELP HEADER, OMITTED HERE

% return input / output structure (use "memory" field for internal states)
if( nargin==0 )
   in.blades.dT = [];              % time step
   in.blades.rho = [];             % atmospheric density
   in.blades.inertia = [];         % blade inertia
   in.blades.radius = [];          % blade radius
   in.memory.omega = [];           % radial velocity (state)
   in.memory.angle = [];           % blade angle (state)
   in.transmission.torque = [];    % back EMF torque
   in.windModel.velocity = [];     % wind velocity
   in.structure.sideslip = [];     % sideslip angle (HAWT only)

   out.angle = [];                 % blade reference angle
   out.omega = [];                 % radial velocity output
   out.torque = [];                % wind torque output
   out.power  = [];                % wind power output
   output.in = in;
   output.out = out;
   return
end
```

WT_Blades.m

The data structure returned in the variable named `output`. It has two fields: `in` and `out`, each of which is another data structure. The `out` structure contains all of the names of the outputs that this model will return when updated. The `in` structure contains all of the expected inputs.

Note that the input names have an additional layer of organization. In this case, we have the inputs dT, rho, inertia, and radius stored in the blades field. Because this is a model for the Blades block, these are the parameters of the model.

Similarly, torque is stored in transmission, velocity in windModel, and sideslip in structure. These fields represent external blocks. Therefore, these are considered actual inputs.

Finally, we have the omega and angle variables stored in the memory field. Anything stored in the memory field is a state. In order to function properly, all of the states of a model must also be supplied as outputs. This enables the simulation to supply them back to the model as inputs in the memory field for the next update.

4.3.3 Model Initialization

In order to run a simulation, the first step is to initialize all of the models. In general, initialization may be needed to:

- Compute the outputs necessary for another model to update
- Perform model-specific initialization steps, such as:
  - setting random seeds,
  - initializing additional state values,
  - clearing persistent memory, etc.

Initialization is done by calling the model with an input data structure, and with a negative time value. An example initialization code fragment from the Blades model is shown below:

Listing 4.2. Example of Model Initialization *WT_Blades.m*

```
% initialize
if( time<0 )
   % input.blades supplied
   output.angle = input.blades.angle;
   output.omega = input.blades.omega;
   output.torque = 0;
   output.power = 0;
   return;
end
```

WT_Blades.m

Here we set values of angle and omega. Recall that these are states for the model. When a model is initialized inside the GUI, the input data structure provided to it always includes the initial state values and parameter values defined for that model. Thus, in general, it is always necessary to have the following line(s) in your initialization case, for every state of the model:

```
output.STATE = input.BLOCK.STATE;
```

where STATE is a state of the model, and BLOCK represents which block the model is simulating (e.g. blades, structure, windModel, windSensor, etc.).

In this case we just return 0 for the torque and power outputs. For the HAWT simulation, this is okay because the model update order is such that the blades model will update in the simulation (providing true outputs for torque and power) before any other blocks need those outputs. It is therefore important to recognize the data dependency of all models in your simulation, and ensure that the required data is available when each model first updates.

4.3.4 Model Update

In general, there are 5 basic steps for updating a model:

1. Rename all required inputs to local variable names
2. Perform necessary computations prior to updating the dynamic states
3. Update the dynamic states
4. Perform necessary computations to compute the outputs
5. Return the outputs

If the the model has no dynamic states, then the update sequence is reduced to 3 steps:

1. Rename all required inputs to local variable names
2. Perform necessary computations to compute the outputs
3. Return the outputs

The first step is not required, but due to the length of the original input names, it is recommended for better code readability. An example of a model update from the Blades model is shown below.

Listing 4.3. Example of Model Update *WT_Blades.m*

```
% inputs
rho        = input.blades.rho;           % atmospheric density
inertia    = input.blades.inertia;       % blade inertia
radius     = input.blades.radius;        % blade radius
dT         = input.blades.dT;            % time step
omega      = input.memory.omega;         % radial velocity
angle      = input.memory.angle;         % blade angle
emfTorque  = input.transmission.torque;  % back EMF torque
windVel    = input.windModel.velocity;   % wind velocity
sideslip   = input.structure.sideslip;   % sideslip angle (HAWT only)
pitchAngle = input.controller.bladesControl; % pitch angle control input

% wind speed
vWind   = Mag(windVel) * cos( sideslip );

% integrate the angular rate with net torque
s.rTurbine = radius;
s.rho = rho;
s.wind = vWind;
s.beta = pitchAngle;
s.startup = .01;

[windTorque, windPower] = TorqueHAWT( omega, s );

d.tW  = windTorque;
d.tEM = emfTorque;
d.inr = inertia;
rhsHandle = @(x,d) WT_BladesRHS2(x, d);

% update state
x0 = [angle; omega];
x = RK4TI( rhsHandle, x0, dT, d );

% send output
output.angle  = x(1);
output.omega  = x(2);
```

```
output.torque = windTorque;
output.power = windPower;

% RHS FUNCTION
function xDot = WT_BladesRHS2( x, d )
xDot     = [x(2); (d.tW-d.tEM)/d.inr ];
```

WT_Blades.m

In this case, we embed the RHS function inside the model. This is not necessary. You can use any RHS function to integrate the dynamics.

CHAPTER 5

AIRFOIL MODELS

5.1 Airfoil Functions

5.1.1 Introduction

The airfoil functions allow you to load in airfoil data and compute lift, drag and moment coefficients.

5.1.2 Loading Data

Data is loaded from a text file of the form

```
CL
Reynold’s Numbers
alpha1 CL(alpha1,Reynold’s numbers)
CD
Reynold’s Numbers
alpha1 CD(alpha1,Reynold’s numbers)
CM
Reynold’s Numbers
alpha1 CM(alpha1,Reynold’s numbers)
```

The text “CL” must be in capital letters. There cannot be a line between the CL and the Reynold’s number data. The Reynold’s numbers are a list of the Reynold’s numbers, one per column of the coefficient. “CM”, “CL” and “CD” need not have the same number of Reynold’s numbers. Each coefficient row has the corresponding angle of attack (alpha) which is in degrees.

An example of a file is shown below.

```
CL
   160000     360000     700000    1000000    2000000    5000000
    0     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000

    1     0.1100     0.1100     0.1100     0.1100     0.1100     0.1100
CD
160000     360000     700000    1000000    2000000    5000000
    0     0.0103     0.0079     0.0067     0.0065     0.0064     0.0064

    1     0.0104     0.0080     0.0068     0.0066     0.0064     0.0064

    2     0.0108     0.0084     0.0070     0.0068     0.0066     0.0066

    3     0.0114     0.0089     0.0075     0.0071     0.0069     0.0068
CM
```

```
60000      500000      700000      860000      1360000      1760000

    0      0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
```

The file should have the name *.af. To load the data do the following:

```
airfoil = LoadAirfoilFile( 'MyAirfoil.af' )
```

The output is a data structure of the form:

```
%   airfoil            (1,1)  Airfoil data structure
%                          .reCL     (1,:) Reynold's number
%                          .alphaCL  (1,:) Angle of attack (deg)
%                          .cL       (:,:) Lift coefficient
%                          .reCD     (1,:) Reynold's number
%                          .alphaCD  (1,:) Angle of attack (deg)
%                          .cD       (:,:) Drag coefficient
%                          .reCM     (1,:) Reynold's number
%                          .alphaCM  (1,:) Angle of attack (deg)
%                          .cM       (:,:) Moment coefficient
```

5.1.3 Using the Data

You can use the data directly by means of

```
[cL, cD, cM] = ComputeAirfoilCoeff( airfoil, alpha, rE )
```

The inputs are the airfoil data structure, desired angles of attack and one Reynold's number. Alternatively you can compute the linear list curve slope

```
cLAlpha = CLAlpha( airfoil, rE )
```

which output the slope of the curve before it reaches stall. The lift coefficient is then

```
cL = cLAlpha*alpha;
```

where the angle of attack is in radians.

5.2 An Example

The toolbox includes four data files, NACA 0012, NACA 0015, NACA 0018 and NACA 0021.

```

alpha = LoadAirfoilFile

ans =

       reCL: [160000 360000 700000 1000000 2000000 5000000]
    alphaCL: [1x116 double]
         cL: [116x6 double]
       reCD: [160000 360000 700000 1000000 2000000 5000000]
    alphaCD: [1x116 double]
         cD: [116x6 double]
       reCM: [60000 500000 700000 860000 1360000 1760000]
    alphaCM: [1x110 double]
         cM: [110x6 double]

         ComputeAirfoilCoeff(airfoil,linspace(0,20))
```

Figure 5-1. Loaded NACA 0012 airfoil data

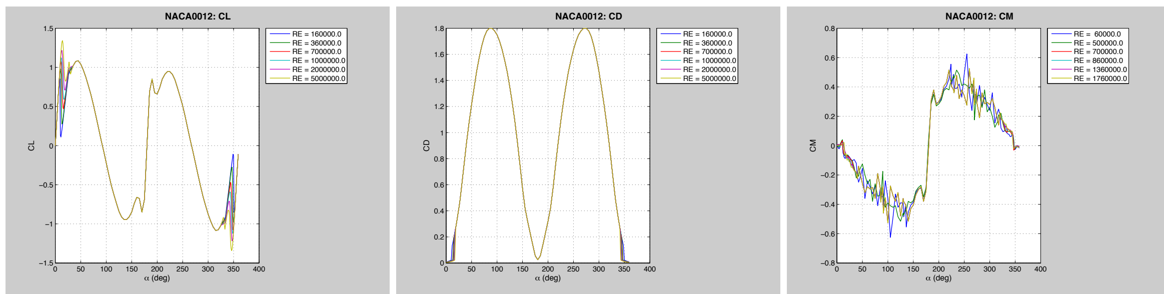
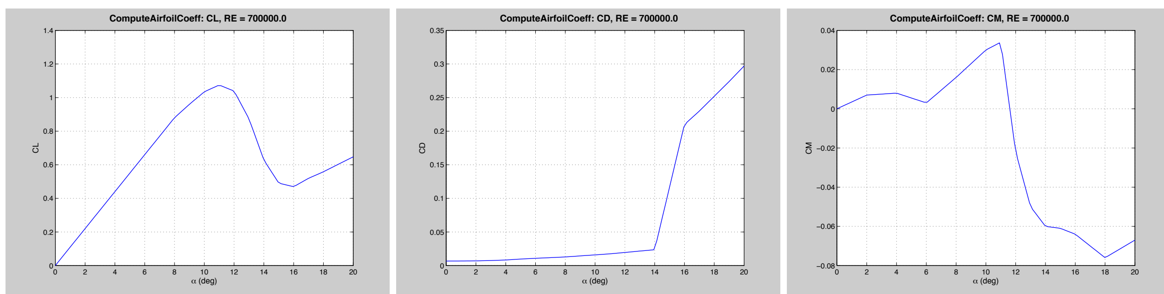


Figure 5-1 on page 25 shows the loaded data.

Figure 5-2 shows the computed coefficients for the range from 0 to 20 deg.

Figure 5-2. Compute NACA 0012 airfoil coefficients

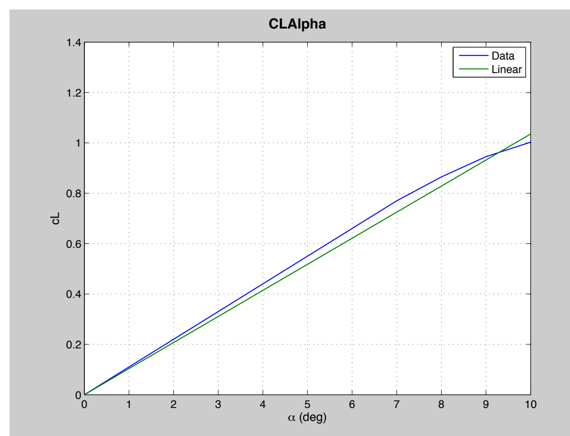


To compute the lift curve slope

```
clA = CLAlpha(airfoil)

clA =

        5.9331

CLAlpha(airfoil)
```

Compare the value with the theoretical lift curve slope of 2π.

Figure 5-3. NACA 0012 lift curve slope



5.3 Coefficients from Shapes

The function `ComputeCoeffFromShape` will generate lift, drag and moment coefficients for an airfoil. The airfoil may be entered as a file with x and y coordinates, an image file (such as a .jpg) or as a data file

Files of numbers are a text file with a .dat suffix and consist of a line with a name and then a list of x, y pairs.

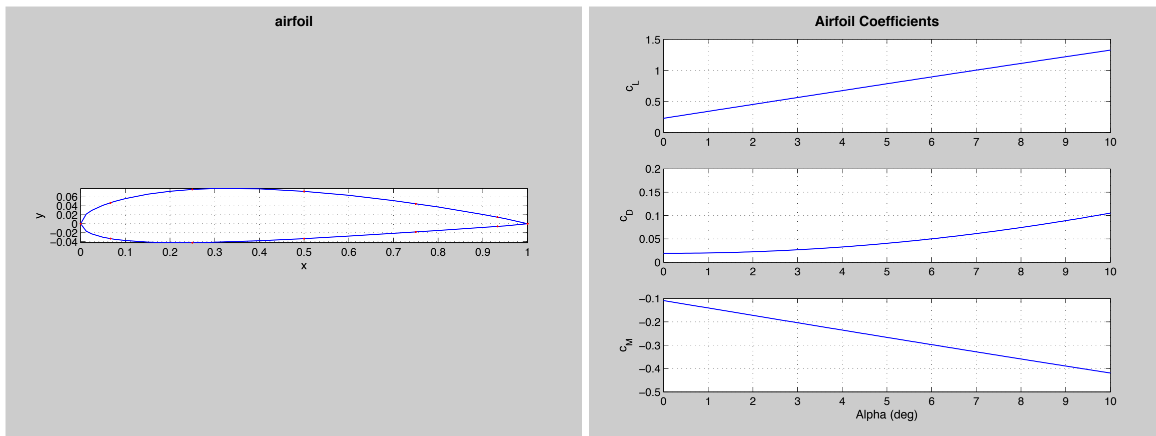
```
[cL, cD, cM] = ComputeCoeffFromShape( x, y, n, alpha )
```

Type

```
ComputeCoeffFromShape
```

for a demo. It will demonstrate an airfoil from a .jpg file. Figure 5-4 on the next page gives the results.

Figure 5-4. Airfoild shape and results



The computation uses the vortex panel method [8] and is for inviscid flow only. n can be a number of panels (top and bottom) or a list of x-coordinates for both top and bottom. at 1.0. It is suitable for quick analyses of the changes of shape of an airfoil (for example due to an airfoil.)

CHAPTER 6

Blade Models

6.1 Introduction

This section discusses the functions for modeling the wind turbine blades. Figure 6-1 on page 28 shows the vertical axis wind turbine blade coordinates. Figure 6-2 on page 29 shows the horizontal axis wind turbine blade coordinates.

6.2 Vertical Axis Wind Turbine Blade Models

6.2.1 Torque Model

The normal and tangential force are found from the following expression.

$$F_N = \frac{1}{2}\rho W^2 S \left(C_L(\alpha)\cos\alpha_0 - C_D(\alpha)\sin\alpha_0\right) \tag{6-1}$$

$$F_T = \frac{1}{2}\rho W^2 S \left(-C_L(\alpha)\sin\alpha_0 - C_D(\alpha)\cos\alpha_0\right) \tag{6-2}$$

The torque is just RF_T. S is the area of the blade.

Functions `LiftAnlytMdl.m` and `DragAnlytMdl.m` provide analytical approximations to common lift and drag profiles.

The VAWT torque model is available in `TorqueVAWT.m`.

6.3 Horizontal Axis Wind Turbine Blade Models

6.3.1 Torque Model

The aerodynamic torque and power captured by a horizontal axis wind turbine is a function of the tip speed ratio and blade pitch. Many coefficient based models are available in the literature. An example model found in [6] has been incorporated in the function`CP2D.m`. This model is called by the function `TorqueHAWT.m`, which returns the HAWT (rotor) torque and power.

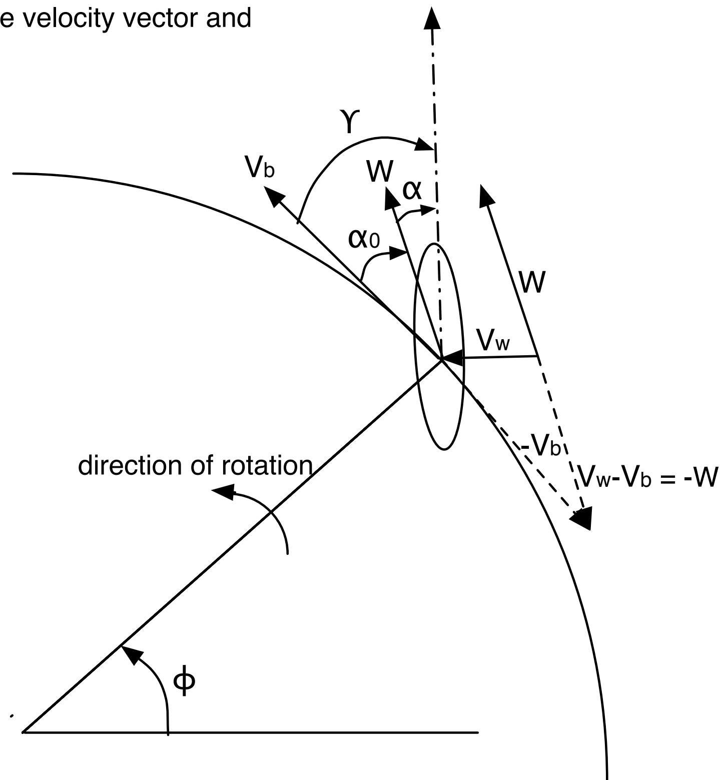
Figure 6-1. Vertical axis wind turbine blade coordinates

ϕ: Rotor angle

α0: Angle between the relative velocity vector and the tangent to the rotor circle

α: Angle of attack

ϒ: Blade pitch angle



Note: For all angles, the arrows indicate the direction in which they are defied to be positive

6.4 Double Streamtube Models

A simplified model for the double-multiple stream tube theory applicable to straight-bladed VAWT is used for modeling and analysis. This model is derived from references [16, 5, 14]. There are many variants of this model in the literature. The version presented in [16] (closest to our application) is not described in sufficient detail, and it may have some inconsistencies or errors. In order to obtain a model suitable for our application a derivation using the underlying concepts of the double-multiple stream tube theory was performed, and is summarized below. We note that any semi-analytical or empirical VAWT modeling approach needs to be validated for a specific application through computational fluid dynamics and/or wind tunnel testing.

Figure 6-3 on page 29(a) shows a schematic of the angular notation for an individual blade. In the figure $\mathrm{V_b}$ is the velocity of the blade in the inertial frame, and V is the effective wind velocity seen by the blade. We note that the double-multiple stream tube theory provides a method to calculate V, the magnitude of V as a function of the far-upstream wind velocity $\mathrm{V_{inf}}$. Vector $-\mathrm{W}$ represents the velocity of the blade relative to the effective wind. The angle θ represents the blade rotation angle, α_1 is the angle from W to $\mathrm{V_b}$, α is the total angle of attack (the angle from W to the chord line of the blade), and γ is the blade pitch angle. The aerodynamic drag force D acts in the direction of $-\mathrm{W}$, whereas the aerodynamic lift force L acts perpendicular to W. The resultant speed where $W = |\mathrm{W}|$, X is the

Figure 6-2. Horizontal axis wind turbine blade coordinates

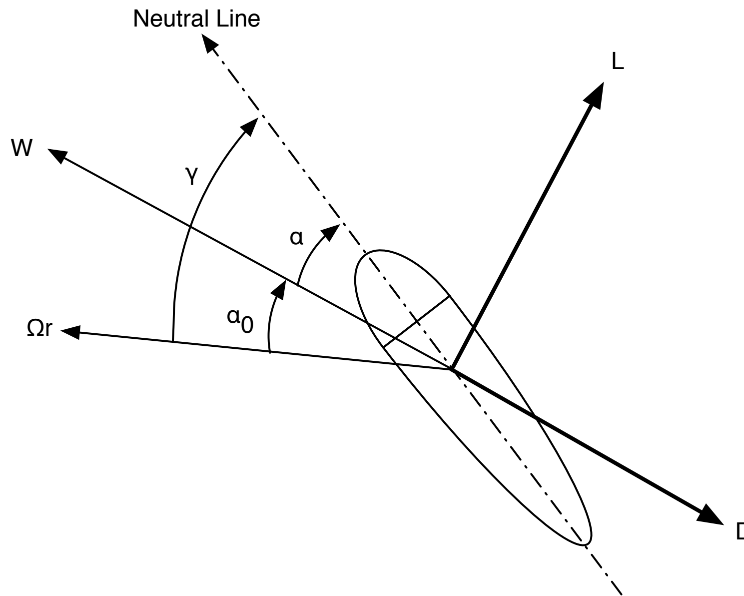
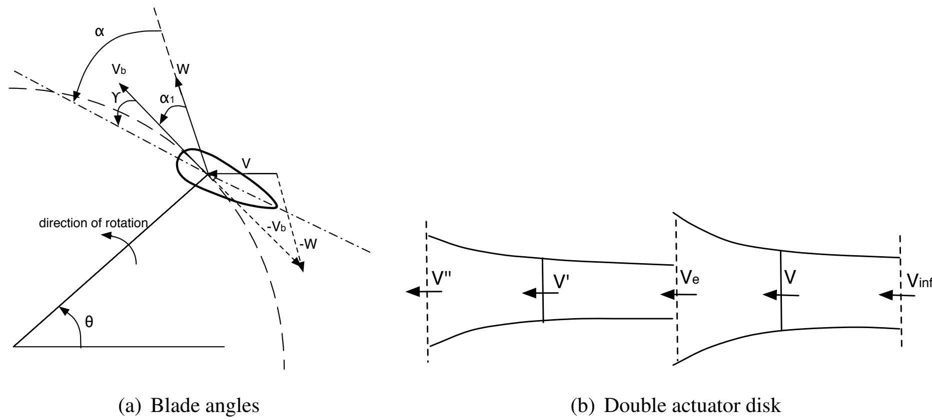


Figure 6-3. Blade angles and Schematic of two actuator disks and associated wind speeds



(a) Blade angles

(b) Double actuator disk

tip speed ratio, R is the radius of the turbine and $\dot{\theta} = \Omega$ is the angular rate of rotation is given by

$$W = V\sqrt{(X - \sin\theta)^2 + \cos^2\theta} \tag{6-3}$$

$$X = \frac{|\mathrm{V_b}|}{V} = \frac{R\Omega}{V} \tag{6-4}$$

The total angle of attack of a pitch-actuated blade is composed of two components α_1 and γ:

$$\alpha = \alpha_1 + \gamma. \tag{6-5}$$

Component α_1 is due to the geometry of flow conditions, and is given by

$$\alpha_1 = \arcsin\left[\frac{\cos\theta}{\sqrt{((X - \sin\theta)^2 + \cos^2\theta)}}\right]. \tag{6-6}$$

Component γ is due to pitch control, and it serves as a control input designed in Section 7.7.1. The double multiple stream tube model uses the double actuator disc theory, momentum theory and blade element theory to first compute the effective speed V experienced by a blade at any azimuthal angle θ, and eventually all the aerodynamic effects.

The main feature of the double actuator disc theory is the consideration of two actuator disks placed behind each other, connected at the center of the turbine as shown in Figure 6-3 on the next page(b). The speeds at the upwind V and downwind $V^{'}$ sides of the rotor are given by

$$V = uV_{\text{inf}} \tag{6-7}$$
$$V^{'} = uV(2u^{'} - 1)V_{\text{inf}} \tag{6-8}$$

is the speed at center of the rotor. Factors u and $u^{'}$ are upwind and downwind interference factors respectively, and they are computed using momentum theory and blade element theory. Using two actuator disks allows consideration of asymmetric airfoils, which will be used in Phase II. The speed at far downstream is given by

$$V^{''} = (2u - 1)(2u^{'} - 1)V_{\text{inf}} \tag{6-9}$$

The momentum theory is applied to two sets of stream tubes that cover the actuator disks. The first set of stream tubes cover the upwind actuator disk. The output of these stream tubes feeds the set that covers the downwind actuator disk. This arrangement allows computation of the total stream wise force acting on the upwind and downwind halves of the wind turbine separately.

For the upwind half, at each stream tube the stream wise force according to the momentum theory is

$$dF_{x,u} = 2A_u\rho V(V - V_{inf}), \tag{6-10}$$

where ρ is the density of air, and

$$A_u = Rd\theta \,|\cos\theta|\, dz, \tag{6-11}$$

dz is the width of the stream tube in the plane perpendicular to the flow. Substituting equation (6-11) in equation (6-10) we get for the upstream and downstream

$$\begin{aligned} dF_{x,u} &= 2Rd\theta \,|\cos\theta|\, dz\, \rho V(V - V_{inf}) && (6\text{-}12)\\ &= 2\rho R\,|\cos\theta| V^2 \frac{u-1}{u} d\theta dz && (6\text{-}13)\\ dF_{x,d} &= 2A_d\rho V^{'}(V^{'} - V_e) && (6\text{-}14)\\ &= 2\rho R\,|\cos\theta|\, V^{'2} \frac{u^{'}-1}{u^{'}} d\theta dz, && (6\text{-}15) \end{aligned}$$

According to the blade element theory the stream wise force acting on the blade element (within a stream tube) is given by

$$dF_x = \frac{1}{2}\rho W^2 \tilde{c}\, dz\, (C_N\cos\theta + C_T\sin\theta)\,, \tag{6-16}$$

where C_N and C_T are the normal and tangential force coefficients of the airfoil, $\tilde{c}$ is a measure of the probability of a blade being within a stream tube (located at the azimuthal angle θ):

$$\tilde{c} = \frac{Nc\,\Omega dt}{2\pi} = \frac{Nc\,d\theta}{2\pi}. \tag{6-17}$$

We note that in terms of the airfoil lift coefficient C_L and drag coefficient C_D the normal and tangential coefficients are given by:

$$\begin{aligned} C_N &= C_L\cos\alpha + C_D\sin\alpha && (6\text{-}18)\\ C_T &= C_L\sin\alpha - C_D\cos\alpha && (6\text{-}19) \end{aligned}$$

Substituting equation (6-17) in equation (6-16) we get

$$dF_x = \frac{1}{2}\rho W^2 \frac{Ncd\theta}{2\pi} dz\, (C_N\cos\theta + C_T\sin\theta) \tag{6-20}$$

The stream wise force computations of the momentum theory and blade element theory given can be equated. For the upwind half this equation (obtained from equations (6-13 and 6-20)) is

$$2\rho R\,|\cos\theta| V^2 \frac{u-1}{u} d\theta dz = \frac{1}{2}\rho W^2 \frac{Ncd\theta}{2\pi} dz\,(C_N\cos\theta + C_T\sin\theta)\,. \tag{6-21}$$

After some algebra the above equation reduces to

$$(u-1)|\cos\theta| d\theta = \frac{Ncu}{8\pi R}(C_N\cos\theta + C_T\sin\theta)\frac{W^2}{V^2}d\theta \tag{6-22}$$

Integrating both sides from $\theta = -\pi/2$ to $\theta = \pi/2$ we get

$$2(u-1) = f_u u, \tag{6-23}$$

where f_u is the upwind and downwind functions are given by

$$f_u = \int_{-\pi/2}^{\pi/2} \frac{Nc}{8\pi R}(C_N\cos\theta + C_T\sin\theta)\frac{W^2}{V^2}d\theta \tag{6-24}$$

$$f_d = \int_{\pi/2}^{3\pi/2} \frac{Nc}{8\pi R}(C_N\cos\theta + C_T\sin\theta)\frac{W^2}{V_e^2}d\theta \tag{6-25}$$

A simple iteration procedure as described in [16] can be used to compute u and $u^{'}$. Initially a value is assumed for u. Now for a given Ω and V_{inf} it is possible to compute V and the tip speed ratio X. The lift and drag coefficients are obtained by interpolating known experimental data using the local α and Reynolds number Re_b

$$Re_b = \frac{Wc}{\nu_\infty}, \tag{6-26}$$

where ν_∞ is the kinematic viscosity. This allows computation of the upwind function f_u. Using equation (6-23) a new value of u is computed. The iteration is carried out until there is a convergence in u. After u has been computed, V_e can be computed using $V_e = (2u-1)V_{\text{inf}}$. Now a similar iteration procedure as above can be used to compute $u^{'}$.

The aerodynamic pitching moment acting on the blade at any time can be computed as

$$M_p = \frac{1}{2}\rho W^2 c C_M, \tag{6-27}$$

where the pitching moment coefficient C_M is obtained by interpolating known experimental data using the local α and Reynolds number Re_b. For a wind turbine with active pitch control the actuator torque depends on M_p, the moment of inertia of the blade about the pitching axis, and the desired pitch profile. Since the blades of a VAWT experience a constantly changing angle of attack and flow velocity, unsteady aerodynamic loads may have a significant effect on the turbine performance. In our simulations for studying aerodynamic performance we consider the dynamic stall effects predicted by the Gormont model modified by Cardona as described in [13, 7]. An alternative dynamic stall model that can be considered in Phase II is the MIT model modified by Noll and Ham as described in [13, 11]. The Gormont model provides a simple empirical relation for the effective angle of attack to be used for lift, drag and moment computations. This effective angle of attack is a function of the actual angle of attack α, its rate $\dot{\alpha}$, airfoil properties and the speed of the blade relative to the effective wind, W.

The aerodynamic torque used in system simulations can be computed using

$$T_r = \sum_{i=1}^{N} \frac{1}{2}\rho W_i^2 c\, C_{T,i}, \tag{6-28}$$

where C_{T_i} and W_i are the torque coefficient and effective relative blade speed computed at the ith blade.

Figure 6-4 on page 32 shows results for an example simulation using the above aerodynamic model. The parameters in the simulation are set to reflect the prototype constructed as part of the Phase I project, and are summarized in Table 6-1 on the next page. For this simulation no pitch control is used. The average power generated is 13 Watts. We note that

in this simulation the angular speed was held fixed at 150 rpm, and that the simulation does not include the variation in angular speed due to rotor dynamics and generator control. These aspects are described in Section **??**, and included in the system simulation described in Section **??**. In Section 7.7.1 we present another simulation incorporating a pitch control algorithm, and demonstrate the improvement in energy extraction.

Figure 6-4. Constant Pitch Simulation

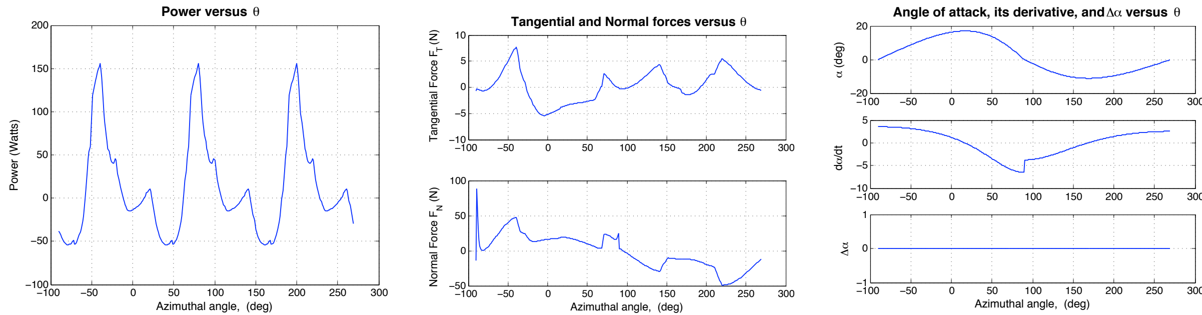


Table 6-1. Simulation Parameters

Parameter	Value	Parameter	Value
Airfoil	NACA 0012	Number of blades N	3
Wind speed V_{inf}	5 m/s	Angular speed Ω	150 rpm
Chord length	0.2032 m	Air density ρ	1.21 kg/m^3
Blade length	1.5240 m	Kinematic viscosity ν	1.48×10^{-5} m^2/s
Rotor radius R	0.9735 m	Dynamic Stall Model	Gormont Model [13]

The double multiple stream tube models are demonstrated in the demo `DMSDemo` for a 2 bladed VAWT and `DMSDemoPC3Bld` for a 3 bladed model.

CHAPTER 7

CONTROL DESIGN

7.1 Introduction

In this chapter we present control design examples for horizontal axis wind turbine (HAWT) and a vertical axis wind turbine (VAWT) systems. We also discuss analytical modeling functions that can used for control design purposes.

7.2 Maximum Power Tracking

The aerodynamic power P_a captured by the rotor of the wind turbine is usually given by the expression

$$P_a = \frac{1}{2}\rho\pi R^2 C_P(\lambda)V^3, \tag{7-1}$$

where ρ is the air density, R is the rotor radius, C_P is the power coefficient, v is the effective wind speed, and $\lambda = \frac{\Omega_r R}{V}$ is the tip-speed ratio, where Ω_r is the rotor angular speed. We note that the power coefficient is a function only of λ, since the pitch of the turbine is held constant in our example. Correspondingly the torque in the rotor T_r is expressed in terms of the torque coefficient $C_Q = C_P/\lambda$:

$$T_r = \frac{1}{2}\rho\pi R^2 C_Q(\lambda)V^2. \tag{7-2}$$

The torque coefficient takes a maximum value $C_{Q_{max}}$ at a certain optimum tip-speed ration λ_{max}. Typically just a discrete set of values for the torque coefficient is available. A good approximation of the torque coefficient that is commonly employed [4] is a second-order polynomial of the form:

$$C_Q(\lambda) = C_{Q_{max}} - K_Q(\lambda - \lambda_{Q_{max}})^2, \tag{7-3}$$

where $K_Q > 0$ is a constant.

The dynamic model of the wind turbine drive-train can be described by the following equations [4]:

$$\begin{bmatrix} \dot{\theta}_s \\ \dot{\Omega}_r \\ \dot{\Omega}_g \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 \\ -\frac{K_s}{J_r} & -\frac{B_s}{J_r} & \frac{B_s}{J_r} \\ \frac{K_s}{J_g} & \frac{B_s}{J_g} & -\frac{B_s}{J_g} \end{bmatrix} \cdot \begin{bmatrix} \theta_s \\ \Omega_r \\ \Omega_g \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{T_r}{J_r} \\ -\frac{T_g}{J_g} \end{bmatrix}, \tag{7-4}$$

where θ_s is the torsion angle depicting the difference in angular positions of the rotor and generator, Ω_g is the angular speed of the generator, T_g is the generator torque, K_s and B_s are the stiffness and damping of the transmission

respectively, J_r and J_g are the moments of inertia of the rotor and generator respectively. For a variable-speed fixed-pitch turbine, T_g regulated by the power generation unit can be considered to be a control input to the above drive-train dynamics.

The goal of the maximum power tracking control law is to follow a control strategy, specified as a locus of operating points. The operating points are equilibria of the system modeled by equation (7-4). The control strategy, usually chosen based on trade-offs between energy extraction and load alleviation, picks a desired steady rotor speed $\Omega_{r,e}$ for each steady wind speed V_e, thereby effectively determining the steady operating torque $T_{r,e}$. From the first component of equation (7-4), the steady generator speed $\Omega_{g,e} = \Omega_{r,e}$. The corresponding torsion angle $\theta_{s,e}$ and generator torque $T_{g,e}$ can be calculated by solving the equilibrium relations corresponding to the last two components of equation (7-4).

For the purpose of deriving the control law let us rewrite equation (7-4) in terms of deviations from the operating point:

$$\begin{bmatrix} \dot{\bar{\theta}}_s \\ \dot{\bar{\Omega}}_r \\ \dot{\bar{\Omega}}_g \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 \\ -\frac{K_s}{J_r} & -\frac{B_s}{J_r} & \frac{B_s}{J_r} \\ \frac{K_s}{J_g} & \frac{B_s}{J_g} & -\frac{B_s}{J_g} \end{bmatrix} \cdot \begin{bmatrix} \bar{\theta}_s \\ \bar{\Omega}_r \\ \bar{\Omega}_g \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\bar{T}_r}{J_r} \\ -\frac{\bar{T}_g}{J_g} \end{bmatrix}, \tag{7-5}$$

where $\bar{(.)}$ denotes the variable minus its value at the operating point. The equilibrium (operating point) in terms of the new variables is the origin $(0, 0, 0)$.

Using equations (7-2)-(7-3) and the definition of λ we can write

$$\bar{T}_r = -K_1\bar{\Omega}_r^2 + K_2\bar{\Omega}_r, \tag{7-6}$$

where K_1, K_2 are constants.

In order to compute the control law ($\bar{T}_g$) and prove the stability of the closed-loop system, consider the Lyapunov function candidate,

$$\Phi = \frac{1}{2}\left[\left(\frac{2J_r + J_g}{J_r}\right)K_s\bar{\theta}_s^2 + J_g\left(\bar{\Omega}_r - \bar{\Omega}_g\right)^2 + J_r\bar{\Omega}_r^2 + J_g\bar{\Omega}_g^2\right]. \tag{7-7}$$

It is straightforward to verify that Φ is a valid Lyapunov function candidate [7]. Now, we compute

$$\begin{aligned}\dot{\Phi} = -B_s\left[\left(\frac{J_r + J_g}{J_r}\right)\left(\bar{\Omega}_r - \bar{\Omega}_g\right)^2 + \bar{\Omega}_r^2 + \bar{\Omega}_g^2\right] + \frac{J_g + J_r}{J_r}\bar{\Omega}_r^2\left(K_2 - K_1\bar{\Omega}_r\right) \\ -\frac{J_g}{J_r}\bar{\Omega}_r\bar{\Omega}_g\left(K_2 - K_1\bar{\Omega}_r\right) - \bar{T}_g\left(\bar{\Omega}_r - 2\bar{\Omega}_g\right).\end{aligned} \tag{7-8}$$

We chose a control law of the form

$$\bar{T}_g = m\bar{\Omega}_r + n\bar{\Omega}_g, \tag{7-9}$$

where m and n are constants. Further, we set

$$n - 2m = \left(1 + \frac{J_g}{J_r}\right)K_2. \tag{7-10}$$

Substituting the inequalities $-\bar{\Omega}_r^3 \le \frac{1}{2}\left(\bar{\Omega}_r^4 + \bar{\Omega}_r^2\right)$, and $\bar{\Omega}_g\bar{\Omega}_r^2 \le \frac{1}{2}\left(\bar{\Omega}_r^4 + \bar{\Omega}_g^2\right)$ in equation (7-8) we get the following relation,

$$\dot{\Phi} \le -a\bar{\Omega}_r^2 - b\bar{\Omega}_g^2 - c\left(\bar{\Omega}_r - \bar{\Omega}_g\right)^2, \tag{7-11}$$

where,

$$a = m + B_s - \frac{J_g + J_r}{J_r}\left(\frac{K_1}{2} + K_2\right) - \frac{2J_g + J_r}{2J_r}K_1\bar{\Omega}_r^2 \tag{7-12}$$

$$b = 2m + B_s - \frac{J_g}{2J_r}\left(2K_2 + K_1\right) \tag{7-13}$$

$$c = B_s\left(\frac{J_r + J_g}{J_r}\right). \tag{7-14}$$

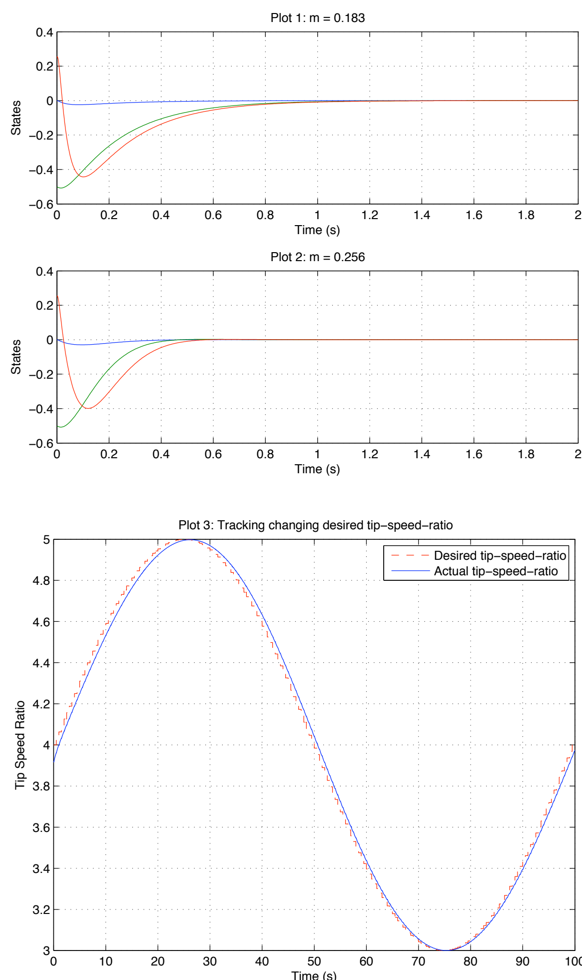


Figure 7-1. Closed-loop simulations for a variable-speed, fixed-pitch WECS

From the above equations we see $c > 0$, and if we pick m large enough we also have $a, b > 0$ for some $\left|\bar{\Omega}_r\right| \leq A$, where A is some constant. Thus,

$$\dot{\Phi} \leq 0 \tag{7-15}$$

for all $x := \left(\bar{\theta}_s, \bar{\Omega}_r, \bar{\Omega}_g\right) \in B_A$, where B_A is a ball of radius A in the space $S^1 \times R^2$. This proves local asymptotic stability of the closed-loop system with the region of attraction given by B_A. In fact, the control law also guarantees local exponential stability. Figure 7.2 demonstrates the application of the control law in a drive-train dynamics simulation. Plots 1 and 2 of the figure show the convergence of the states of the system to the desired operating point (the origin in both cases) for two different values of the control gain. The larger m provides faster convergence. Plot 3 shows the result of a simulation where the system is made to track a desired tip-speed ratio profile. Good performance is observed in all simulations.

We note that the size of the region of attraction (determined by A) is a function of the adjustable control gain m. Choosing a larger value of m will lead to a faster transient and larger region of attraction. However, a large m may not be desirable if the wind has dominant high frequency components, since this may lead to larger mechanical loads.

7.3 Generator Control

Generator models available in the toolbox are described in Chapter 10. Consider the model represented by equations (10-22)-(10-22). The control strategy prescribes a certain desired generator torque or a desired generator angular speed, which must be realized by applying control voltages u_d and u_q. A simple way to compute $u = (u_d, u_q)$ is to calculate the steady state values of the voltage to apply as a feedforward component u_{ff} and then use a linear control law to regulate the system about the desired steady state values. At steady state $i_d = 0$. The steady state i_q corresponding to a desired generator torque or generator speed can be computed using equations (10-19), (10-23) by setting $\frac{d\omega_m}{dt} = 0$.

7.4 Control Design Functions

There are several linear control options available in the toolbox. Here, we describe the `QCR` function for performing a linear quadratic control design. The `QCR` function creates a regulator of the form

$$u = -Kx \tag{7-16}$$

minimizing the cost functional

$$J = \int \left(\frac{1}{2} \left(u^T R u + x^T Q x \right) + u^T N x + x^T N u \right) dt \tag{7-17}$$

given the linear dynamics

$$\dot{x} = Ax + Bu. \tag{7-18}$$

The function call with sample input and output (taken from the linear quadratic regulator design used in the HAWTSim demo for designing the feedback component of generator control) is

```
>> a = 1.0e+03 *[-0.0178 0 -0.0016; 1.2616   -0.1593    0.1387;-0.0403   -0.1387
      -0.1593]

a =

   1.0e+03 *

   -0.0178         0   -0.0016
    1.2616   -0.1593    0.1387
   -0.0403   -0.1387   -0.1593

>> b = [0 0;37.0370 0;0 37.0370]

b =

         0         0
   37.0370         0
         0   37.0370

>> kR      = QCR( a, b, diag([1 0.1 0.1]), eye(2) )

kR =

    0.0865    0.0118   -0.0003
   -0.0029   -0.0003    0.0114
```

Several control design functions are included in the toolbox. Table 7-1 on page 37 provides a summary of a selection of those functions:

Table 7-1. Selected Control Design Functions

Function Name	Description
`Acker`	Computes the gain for desired pole locations using Ackermann's formula so that the closed-loop system has the desired poles
`C2DZOH`	Create a discrete time system from a continuous system assuming a zero-order-hold at the input
`CButter`	Build a continuous Butterworth Filter
`CGram`	Compute the controllability gramian for a continuous time system
`CToD`	Create a discrete time compensator from a continuous time compensator
`DBode`	Generates a Bode plot for a discrete time system
`Delay`	Create a model of a delay using Pade approximants
`DigitalFilter`	Implement a digital filter
`EVAssgnC`	Use eigenvector assignment to design a controller
`FResp`	Compute the frequency response of the system
`GPMargin`	Computes gain and phase margins
`LQC`	Linear quadratic controller
`Nyquist`	Generate a Nyquist plot from a statespace object or gain/phase data
`PDDesign`	Design a proporational-derivative controller
`PhasePlane`	Implement a phase-plane controller
`PID`	Design a proporational-integral-derivative controller
`Riccati`	Solves the matrix Riccati equation
`RootLocus`	Generate the root locus for a single-input-single-output system
`S2Z`	Transform an s-plane transfer function into the z-plane
`Step`	Generate a step response of the system
`SVPlot`	Compute the maximum and minimum singular values
`Windup`	Implements anti windup compensation
`ZFresp`	Generates the frequency response for a digital filter

7.5 HAWT Demo

The file `HAWTSim.m` contains a demonstration of HAWT control. The HAWT blade model is described by `TorqueHAWT.m`. The generator is modeled as a permanent magnet brushless machine in DQ coordinates. The HAWT system is linearized about the desired operating point. The generator control feedforwards the expected control voltage and then controls about that voltage with the linear quadratic regulator. We assume that we measure all states including turbine rotational speed and generator currents. Figure 7-2 on page 38 shows a plot from the HAWT control demo.

7.6 DFIG Control

This section discusses the control of a wind turbine with a DFIG generator. In this demonstration we control the speed of the generator to maximize the power output and control the reactive power to drive it to zero.

We will assume a fixed β of 0 deg. Then the optimal λ is 8. This leads to the desired mechanical rotation speed of

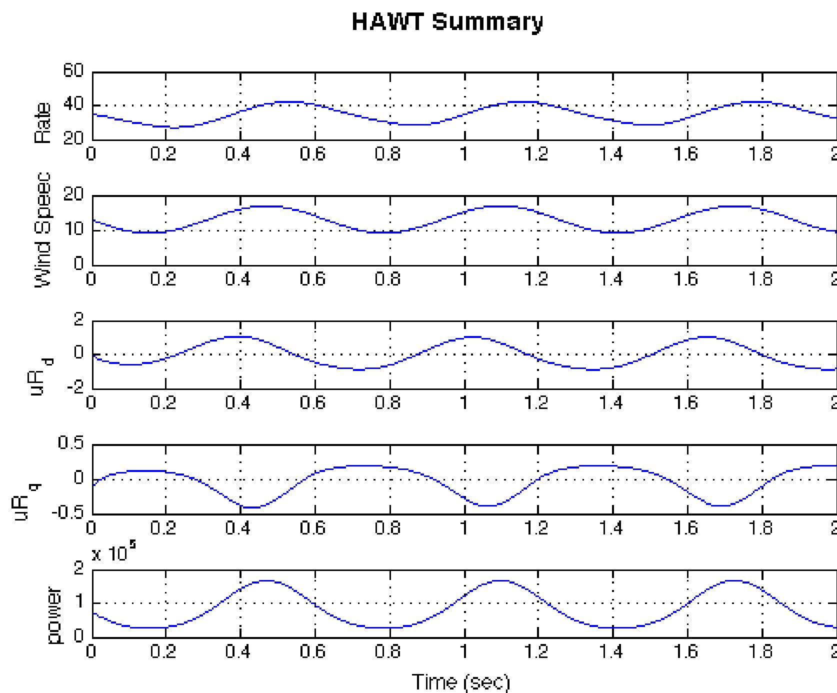
$$\omega_W = W\frac{8}{R} \tag{7-19}$$

The desired mechanical rotation rate is

$$\omega_m = \frac{\omega_W}{n} \tag{7-20}$$

where n is the gearbox ratio. The electrical angular rate is

Figure 7-2. HAWT Control Demo



$$\omega_e = p\omega_m \tag{7-21}$$

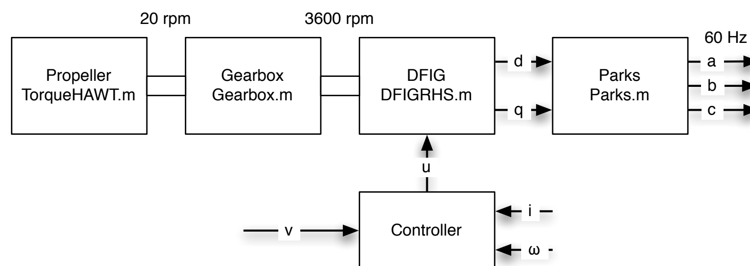
where p is the number of pole pairs. We will assume a 2 pole pair generator so that

$$\omega_e = p\frac{\omega_W}{n} \tag{7-22}$$

A GE 2.5 MW wind turbine is a model with a 50 m blade and nominal wind speed of 12.5 m/s. The shaft rate is 1800 rpm in this case (3600 rpm/2).

We have four control voltages, two for the rotor and two for the stator. The overall control/simulation flow is shown in Figure 7-3 on the following page. It shows the toolbox functions used in the simulation. Unlike the simulation above we will output 3 phase currents.

Figure 7-3. DFIG simulation



The first step is to find the steady-state values for the currents and voltages. Our inputs are

1. Q_{ref}

2. ω_m - mechanical speed
3. v_{ds} and v_{qs} - the stator direct and quadrature voltages
4. T_m - the mechanical torque
5. ω - the reference frequency

The flux equations are

$$\psi_{qs} = (L_s + L_m)i_{qs} + L_m i_{qr} \tag{7-23}$$
$$\psi_{ds} = (L_s + L_m)i_{ds} + L_m i_{dr} \tag{7-24}$$
$$\psi_{dr} = (L_r + L_m)i_{dr} + L_m i_{ds} \tag{7-25}$$
$$\psi_{qr} = (L_r + L_m)i_{qr} + L_m i_{qs} \tag{7-26}$$
$$\tag{7-27}$$

We need to solve the following nonlinear equations for currents.

$$0 = v_{ds} - R_s i_{ds} + \omega\psi_{qs} \tag{7-28}$$
$$0 = v_{qs} - R_s i_{qs} - \omega\psi_{ds} \tag{7-29}$$
$$T_e = \frac{3}{2}p\left(\psi_{ds}i_{qs} - \psi_{qs}i_{ds}\right) \tag{7-30}$$
$$Q_{\text{ref}} = 3\left(v_{ds}i_{qs} - v_{qs}i_{ds}\right) \tag{7-31}$$

The currents are computed numerically in the function `DFIGEquilibrium` which uses downhill simplex, i.e. `fminsearch`. The rotor voltages are computed in the function from

$$v_{dr} = R_r i_{dr} - \omega_s\psi_{qr} \tag{7-32}$$
$$v_{qr} = R_r i_{qr} + \omega_s\psi_{dr} \tag{7-33}$$

This provides the equilibrium solution for the current and the feedforward values for the rotor voltages. An example

```
1  d.u       = [120;120;0;0];
2  d.p       = 2;
3  d.lM      = 47.3e-3;   % Mutual inductance  (H)
4  d.lR      = 50e-3;     % Rotor inductance   (H)
5  d.lS      = 50e-3;     % Stator inductance  (H)
6  d.rR      = 0.38;      % Rotor resistance   (Ohm)
7  d.rS      = 0.05;      % Stator resistance  (Ohm)
8  d.omega   = 2*pi*60;   % Synchronous speed  (rad/s)
9  d.tM      = 1000;      % Mechanical torque  (Nm)
10 d.j       = 0.5;       % Inertia            (kg-m^2)
11 qRef      = 0;
12 omegaM    = 1800*pi/30;
13
14 [uR, x]   = DFIGEquilibrium( qRef, omegaM, d );
15 d.u(3:4)  = uR;
16 [xDot, tE, q] = DFIGRHS( x, 0, d )
17
18 xDot =
19
20   -8.2982e-06
21     1.283e-05
22    4.0339e-06
23   -6.2371e-06
24    1.1336e-06
25         188.5
26
27 tE =
28
29         -1000
30
31 q =
```

```
32
33  -1.2149e-07
34        33052
```

The first five elements of `xDot` are small showing that we are near equilibrium.

7.7 VAWT Demo

The file `VAWTSimDemo.m` contains a demonstration of variable pitch VAWT control. The VAWT blade model is described by `VAWTDemoBldMdlRHS.m`. The generator is modeled simply as providing a feedback (control) torque. Then, the dynamics can be described using the following equations:

$$J_r \frac{d\phi}{dt^2} = \tau - \tau_g \tag{7-34}$$

$$J_{b_i} \frac{d\gamma}{dt^2} = u_{p_i}, \tag{7-35}$$

where J_r is the total moment of inertia of the VAWT rotor, J_{b_i} is the moment of inertia of the ith blade and u_{p_i} is the corresponding pitch control torque. τ_g is the feedback control torque from the generator. The total aerodynamic torque is given by

$$\tau = \sum_{i=1}^{n} \tau_i = \sum_{i=1}^{n} C_{T,i} \frac{1}{2} \rho A W^2 R, \tag{7-36}$$

where τ_i is the aerodynamic torque on each individual blade, n is the number of blades, ρ is the density of air, A is the reference area, R is the radius of the rotor, and C_{T_i} is the tangential force coefficient, made up of contributions from the lift and drag coefficients:

$$C_{T,i} = -\left(C_{L,i} \sin \alpha_i + C_{D,i} \cos \alpha_i\right) \tag{7-37}$$

The total power captured by the rotor is given by

$$P_T = \sum_{i=1}^{n} \tau_i \frac{d\phi}{dt} \tag{7-38}$$

We choose individual blade pitch to maximize the lift to drag ratio over the entire cycle of rotation in such a way that the force on each blade contributes positively to power extracted for most of the rotation cycle. The following control strategy is employed for the pitch control torque, u_p:

$$u_p = -K_p \left(\alpha - \alpha_{ref}\right) - K_d \dot{\gamma}, \tag{7-39}$$

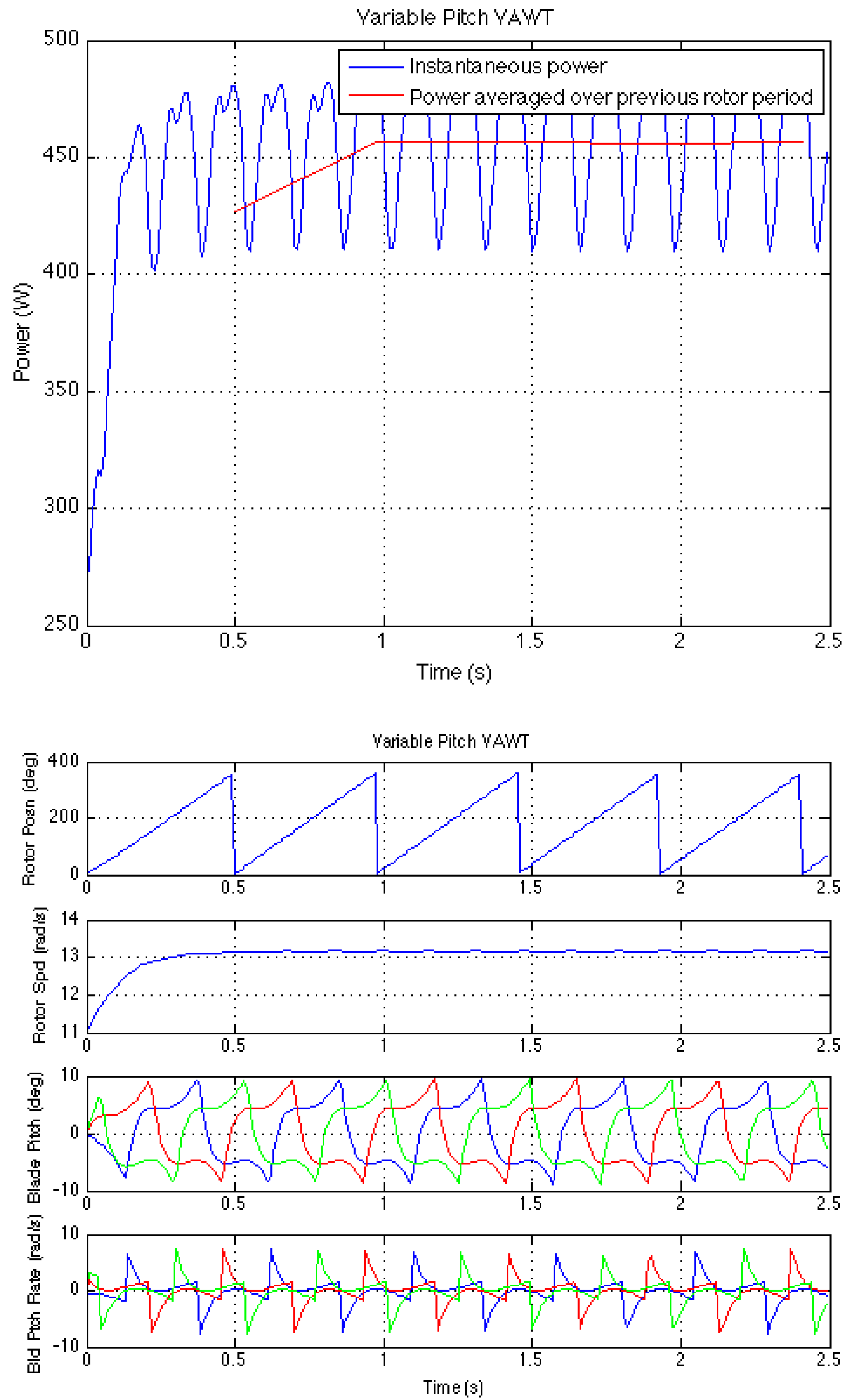
$$\alpha_{ref} = \begin{cases} -\alpha_m \text{ if } \alpha_0 > 0 \\ \alpha_m \text{ if } \alpha_0 < 0 \end{cases}, \tag{7-40}$$

where $\dot{\gamma}$ is the pitch rate, K_p, K_d are control gains, α is the angle of attack, and α_{ref} is a reference angle of attack computed as above. α_m is the reference angle of attack magnitude. Figure 6-1 on page 28 provides a definition of α_0. The value of α_m is chosen as high as possible, but sufficiently smaller than the magnitude of the stall angle of attack. The pitch angle control law ensures that the blade does not get into stall, and that the tangential force component on the blade contributing to positive power extraction is high throughout the entire rotation cycle. The electromagnetic torque control is chosen to stabilize the rotation speed of the blade to a desired value, Ω_{des}:

$$\tau_g = K_{\tau_g,1} \left(\Omega - \Omega_{des}\right) + K_{\tau_g,2} \int \left(\Omega - \Omega_{des}\right) .\mathrm{d}t, \tag{7-41}$$

where $K_{\tau_g,1}$, $K_{\tau_g,2} > 0$ are control gains. For an ambient effective wind speed, V_w, there is an associated optimal rotation speed that can be determined by experiments. Generator torque control is used so that the VAWT tracks this optimal rotor speed. Figure 7-4 on the facing page shows plots from the VAWT control demo.

Figure 7-4. VAWT Control Demo

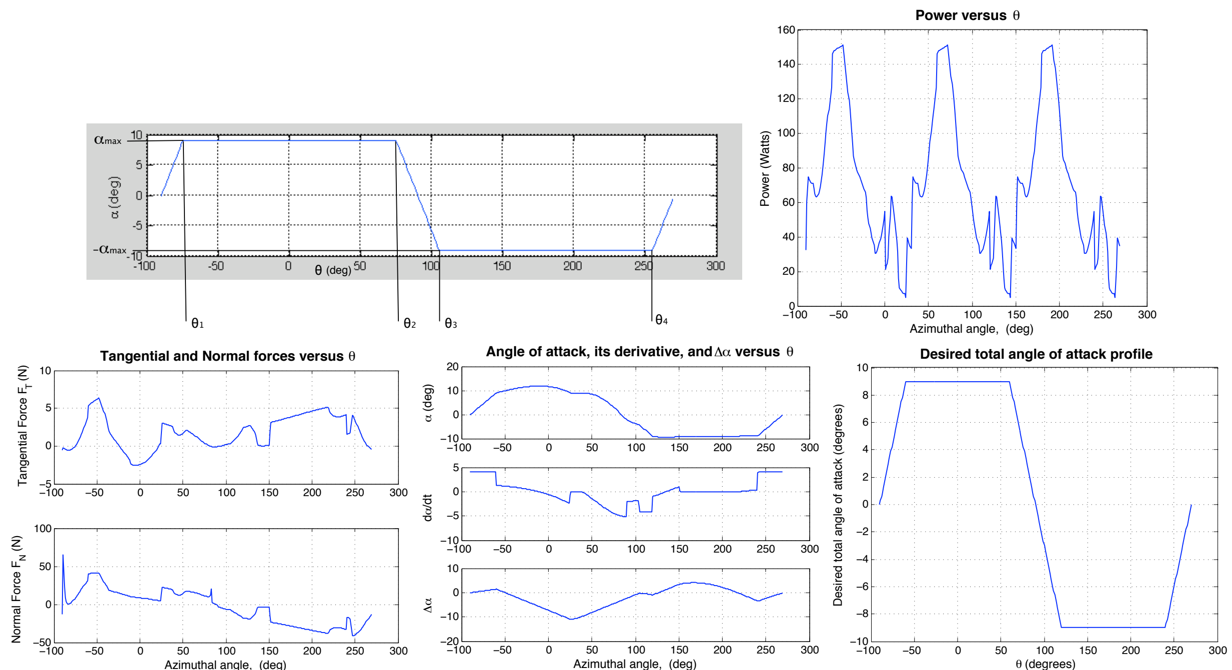


7.7.1 Pitch Control Algorithms

The pitch control algorithm proposed in Phase I attempts to keep the local total angle of attack just below stall for the entire rotation cycle. In order to make the control algorithm implementable, the demanded blade pitch must not have any discontinuities. A generic form of the total angle of attack profile of the form shown in Figure 7-5 on page 42 is considered. We note that the pitch control actuator will have torque and speed constraints that may limit the implementation of desired total angle of attack profile. The speed constraints are included explicitly in our system simulation studies. The stepper motor used in the Phase I prototype has been designed to provide the required torque with sufficient factor of safety. Control and system simulations incorporating the torque constraint can be further studied in Phase II using the test data gathered from the Phase I prototype.

Figure 7-5 shows results for an example simulation where the maximum pitching speed is restricted to 18.9 rpm, which is close to the limit which the motors on the Phase I prototype can deliver (although they were specified to have a higher speed limit). The simulation parameters are the same as that given in Table 6-1 on page 32, except for the steady rotor speed which is set to 130 rpm. The angles in the desired total angle of attack profile are set as follows: $\theta_1 = -60$ degrees, $\theta_2 = 60$ degrees, $\theta_3 = 120$ degrees, $\theta_4 = 240$ degrees and $\alpha_{max} = 9$ degrees. The average power generated in this case is 72 Watts, much higher compared to the power without pitch control at 150 rpm.

Figure 7-5. Desired Total Angle of Attack Profile and Variable Pitch Simulation



CHAPTER 8

ESTIMATION

8.1 Overview

The toolbox includes software for developing estimation algorithms. The algorithms can be used to solve any estimation problem. Particular technologies included are

1. Fixed Gain Kalman Filters
2. Variable Gain Kalman Filters
3. Extended Kalman Filters
4. Unscented Kalman Filters
5. Fault Detection

The toolbox focuses on recursive estimation. Recursive estimation is simply taking a measurement and using it to update an existing estimate of the state of the system. The measurement need not be a state, but for all states to be observed the measurement must be related to the states or derivatives of the states.

In this module, recursive estimators can be divided into six classes:

1. Fixed gain estimators
2. Variable gain estimators for linear systems
3. Variable gain estimators for nonlinear systems in which the plant (the state dynamics) are linearized about the current state and propagated as a linear system. This type of system is used for attitude estimation. This is also called the Extended Kalman Filter.
4. Variable gain estimators for nonlinear systems in which the plant is numerically integrated. This type of system is used for orbit estimation. This is also called the Continuous Discrete Extended Kalman Filter.
5. Unscented Kalman Filters. These filters compute multiple states and covariances for each in effect computing the statistics online.

This chapter will discuss estimation in general with examples from attitude and orbit estimation.

8.2 Fixed Gain Estimators

Fixed gain estimators are written in the form

$$x_{k+1} = ax_k + bu_k + k(y - cx_k) \tag{8-1}$$

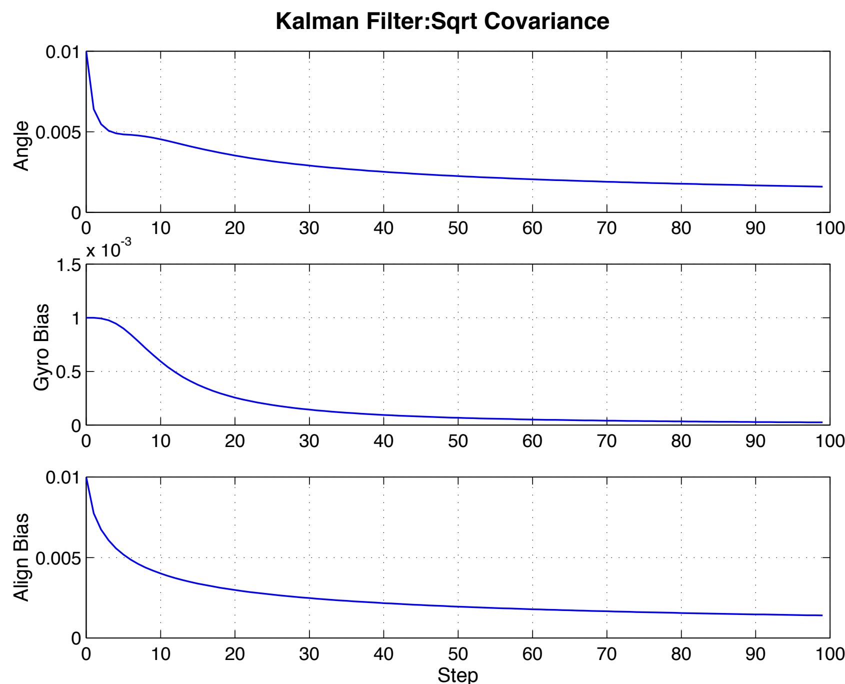
where x_k is the state vector at step k, a is the state transition matrix, b is the input matrix, u_k is the input vector, y is the measurement vector and c is the measurement matrix that relates the states to the measurements.

The gain matrix, k may be computed in many different ways. Note that this, in contrast to a conventional noise filter, requires an estimate of the inputs. However, the filter returns the vector of all observable states which is needed for linear quadratic controllers.

This kind of estimator can be designed using the toolbox function `QCE`, for continuous systems, and `DQCE` for discrete systems. If the former is used, the filter must be converted to discrete time in a second step. If the latter is used you must be careful that the filter state matrix is balanced and well-conditioned.

An interesting example is given in `KalmanFilterDemo` in which two angle measurements are available for single degree of freedom case. One of the states is the unknown misalignment between the two sensors. Figure 8-1 on page 44 shows the covariance for this problem.

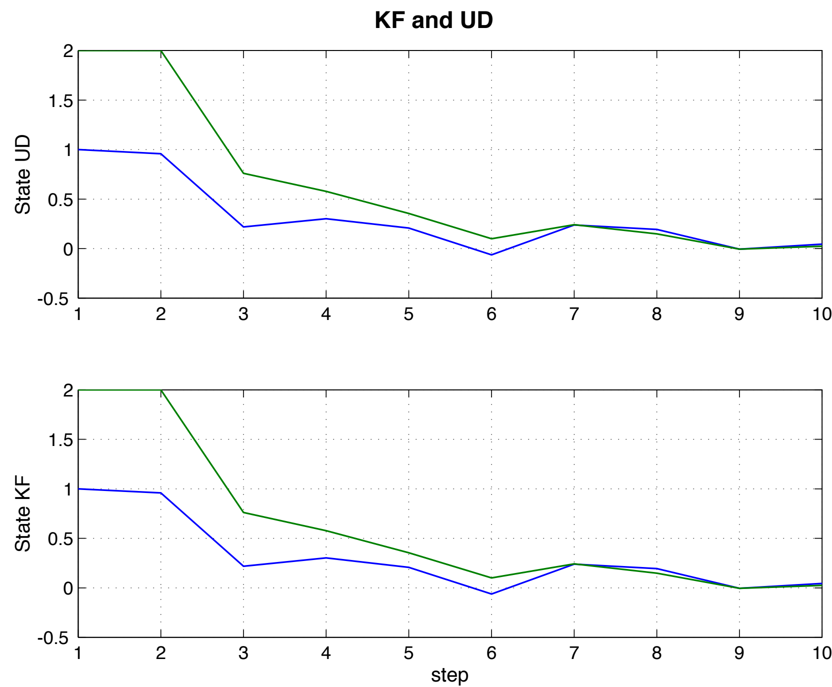
Figure 8-1. Fixed gain Kalman filter



8.3 Variable Gain Estimators

Two routines are available for variable gain estimators. These are `KFilter` for conventional Kalman filters and `UDKalmanFilter`, for filters in which the plant matrix is formulated in Upper Diagonal (UD) form. The latter has better numerical properties. Both filters are demonstrated for double integrator plant in the demo `UDKFDemo`. The results are compared in Figure 8-2 on the next page.

Figure 8-2. UD filter



As expected with double precision arithmetic both filters produce the same result. With single precision arithmetic the UD form should be better results. Note that `KFilter` eliminates the problem of instabilities due to asymmetric covariance matrices by forcing it to be symmetric each step.

8.4 Extended Kalman Filter

The Extended Kalman Filter is used for attitude determination and many other applications. In an Extended Kalman filter the state transition matrix and measurement matrix is linearized about the current estimated state. This is in contrast to the linearized Kalman Filter in which the measurement and state transition matrices are linearized about a predetermined state or state trajectory.

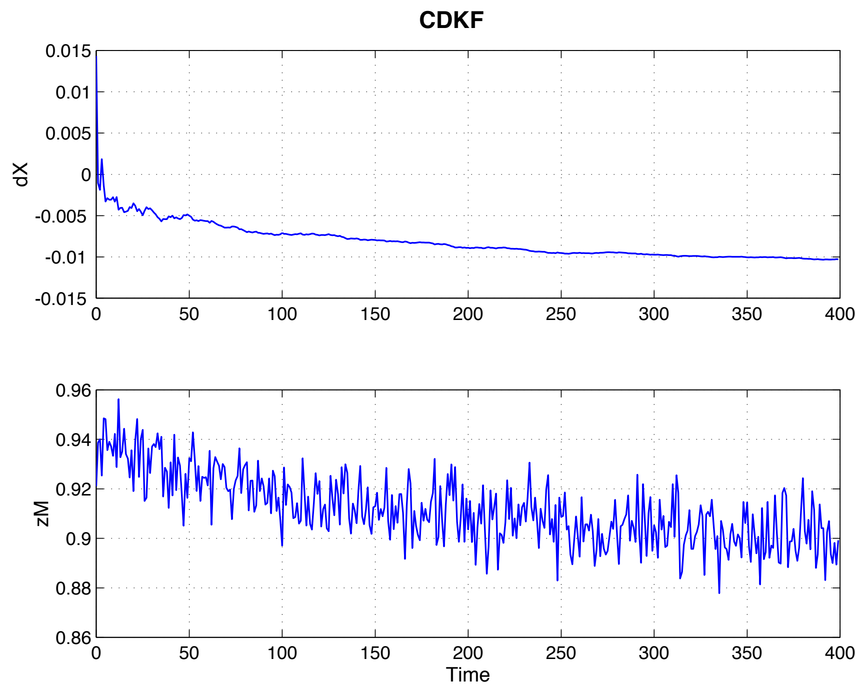
8.5 Continuous Discrete Extended Kalman Filter

8.5.1 Introduction

The script `CDKFDemo` demonstrates the use of a Continuous Discrete Kalman Filter with a nonlinear spring model, and a nonlinear position measurement.

The first plot in Figure 8-3 on page 46 shows that the estimate converges to the true value of the spring position less a bias. The second shows the noisy measurement.

Figure 8-3. Continuous discrete Kalman filter



8.6 Unscented Kalman Filter

The Unscented Kalman Filter (UKF) is able to achieve greater estimation performance than the Extended Kalman Filter (EKF) through the use of the unscented transformation (UT). The UT allows the UKF to capture first and second order terms of the nonlinear system. Instead of just propagating the state, the filter propagates the state and additional sigma points which are the states plus the square roots of rows or columns of the covariance matrix. Thus the state and the state plus a standard deviation are propagated. This captures the uncertainty in the state. It is not necessary to numerically integrate the covariance matrix.

CHAPTER 9

ELECTRICAL MODELS

9.1 Introduction

Electrical models are included for building electrical circuits used for wind turbines. These models can be incorporated into WTSim models or used independently. This chapter shows how to use them independently.

9.2 Circuit Element Models

9.2.1 Introduction

The circuit elements included are

1. diode
2. bridge rectifier
3. capacitor
4. inductor
5. transformer
6. matrix converter
7. three phase rectifier/inverter

9.2.2 Diode

A diode passes current if the voltage across the diode exceeds the threshold. The function

```
vO = Diode( g, v)
```

models a simple diode.`v` is compared against `g.vDiode`. If `v(k)` is greater than `g.vDiode` then `iO(k)` is `i(k)`. Otherwise it is zero. Since this is a nonlinear device the state space model is only valid when the diode is conducting.

9.2.3 Bridge Rectifier

A bridge or full wave rectifier passes current if the magnitude of the voltage across the diode exceeds the threshold. Negative voltages are flipped to positive. This is the basis for AC to DC conversion

```
vO = RectifierFullWave g, v)
```

9.2.4 Capacitor

The capacitor is modeled as

$$i = C\frac{dv}{dt} \tag{9-1}$$

where i is the current passing through the capacitor, C is the capacitance and v is the voltage across the capacitor. The model says that once the voltage stops changing the current goes to zero. The impedance of a capacitor is

$$Z = \frac{1}{sC} \tag{9-2}$$

where $s = j\omega$.

9.2.5 Inductor

The inductor is modeled as

$$v = L\frac{di}{dt} \tag{9-3}$$

where L is the inductance. The model says that when the current stops changing the voltage across the inductor is zero. All motors are composed of coils. The impedance of an inductor is

$$Z = sL \tag{9-4}$$

where $s = j\omega$.

9.2.6 Grid Model

The grid is modeled as a voltage source and a series impedance

$$Z_g = R_g + jX_g \tag{9-5}$$

where j denotes an imaginary number. The voltage drop across the element is

$$v = iZ_g \tag{9-6}$$

The impedance can be written as

$$Z_g = R_g + sL_g \tag{9-7}$$

where $s = j\omega$. ω is the frequency of oscillation in the circuit which is 60 Hz for the grid connection. The equivalent differential equation for this element is

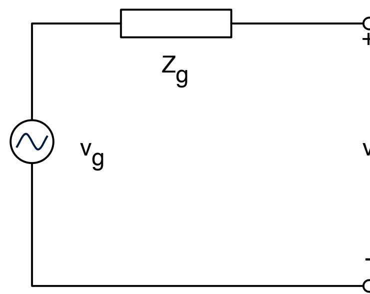
$$v = iR_g + L_g\frac{di}{dt} \tag{9-8}$$

where v is the voltage drop across the element. The voltage across the terminals is therefore

$$v = v_g - iR_g - L_g\frac{di}{dt} \tag{9-9}$$

Typical values for R_g is 1 Ohm and for L_g is 0.1 mH (milli henries).

Figure 9-1. Grid



9.2.7 Transformer

The transformer model just performs a voltage and current scaling based on the ratio of turns.

```
[v2, i2] = Transformer( v1, i1, a )
```

The current ratio, a can be n or $1/n$ where n is the ratio of turns in the two inductors.

A dynamical model right-hand-side in the DQ frame is

```
iDot = TransformerRHS( i, t, d )
```

This function can be called by RK4. The data structure defining the transformer parameters is

```
%   d             (1,1) Data structure
%                       .L                (1,1) Inductance (H)
%                       .R                (1,1) Resistance (ohms)
%                       .omega    (1,1) Reference frequency (rad/s)
%                       .u1               (2,1) [uD;uQ]
%                       .u2               (2,1) [uD;uQ]
```

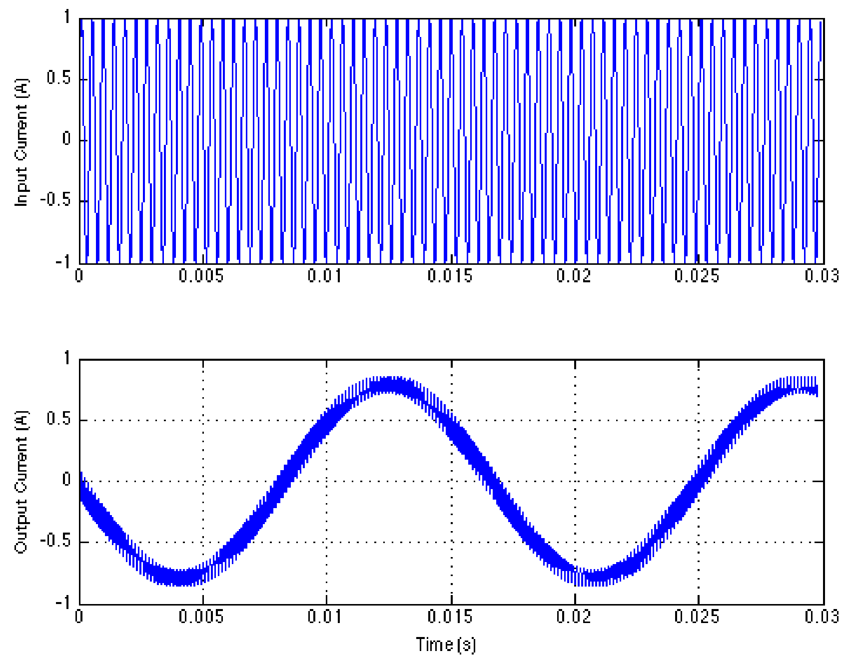
9.2.8 Matrix Converter

A matrix converter can be used for efficiently processing the three-phase electrical output from the VAWT. Matrix converters use an array of controlled, bidirectional semiconductor switches to convert AC power from one frequency to another. They generate a variable output voltage with unrestricted frequency. Matrix converters do not have a dc-link circuit and do not use large energy storage elements.

MOSFETs (for low power) and IGBTs (for high power) enable implementation of bidirectional switches make the matrix converter technology very attractive for AC power handling. Figure 5 shows a schematic of the matrix converter set up [Wheeler et al, 2002], showing the power stage containing nine bidirectional switches, the input filter block and the clamp circuit. The input filter minimizes the high frequency components in the input currents and reduces the impact of perturbations of input power. The input filter can be realized using inductor - capacitor combinations, with parallel damping resistors. The clamp circuit provides overcurrent/overvoltage protection, and is implemented using fast recovery diodes.

In MatConSSDemo.m a switching control strategy based on [10] is implemented. Figure 9-2 on page 50 shows results of the steady state simulation. The generator input at 300 Hz is converted to a 60 Hz output for interfacing with the grid. A switching frequency of 7200 Hz is employed. There is a duty cycle factor that can be adjusted to regulate the ratio of output to input voltage, up to a maximum value. The output is passed through a low-pass filter to filter out the high frequency switching harmonics.

Figure 9-2. Matrix Converter Steady State Simulation



9.2.9 Three Phase Rectifier

This model takes a 3 phase input and given switch states `s` produces a rectified output. The states are +1 or 0. The even switches invert the sign of the output.

```
vO = RectifierThreePhase( g, v, s )
```

9.3 Utility

9.3.1 Phases

```
v = NPhase( w0, t, n )
```

Generates an `n` by `length(t)` array.

CHAPTER 10

GENERATOR MODELS

10.1 Introduction

Their are numerous generators used in wind turbines. Some are

1. Permanent Magnet Synchronous [2]
2. Induction [1]
3. Switched Reluctance [20, 17, 19, 12, 9, 15]

The toolbox provides induction and permanent magnet generator models for use in WTSim. Permanent Magnet and Switched Reluctance are both types of synchronous generators. Induction generators are asynchronous. Permanent Magnet and Switched Reluctance generators rely on power electronics control to work with wind turbines. There are many other types of generators which can be added to the toolbox by using the included functions as models.

All generators have a moving and stationary part. In a rotating machine the moving part is the rotor and the fixed part is the stator. One part generates a field the other has coils in which currents are produced by the motion of the moving part driven by wind produced torque. The field can be generated by

1. Permanent magnets
2. A coil

The coil can be energized by induction or by an electrical source.

10.2 Electrical and Mechanical Degrees

In a rotating machine the number of mechanical degrees is 360 per rotation. Electrical degrees refers to the number of degrees the electrical wave moves while the rotor rotates. The number of electrical degrees is

$$\theta_e = p\theta_m \tag{10-1}$$

where p is the number of pole pairs. A machine with a positive and negative pole (two magnets, for example) has one pole pair. The relationship between rates is the same

$$\omega_e = p\omega_m \tag{10-2}$$

10.3 Direct Quadrature Model

The direct quadrature model transforms from a n-phase rotor to a model which consists of a direct axis winding which is the equivalent of one of the windings but aligned directly with the field. The quadrature winding leads the field winding by 90 electrical degrees. The general transformation is

$$\begin{bmatrix} u_d \\ u_q \\ u_0 \end{bmatrix} = Tu_{ph} \tag{10-3}$$

where for an n-phase machine T is a 3-by-n matrix and u_{ph} is an n-by-1 vector of phase voltages. This transformation maps balanced sets of phase currents into constant currents in the d-q frame. This transformation is known as the Park's transformation. For a 3 phase machine it is

$$T = \begin{bmatrix} \cos(\theta) & \cos(\theta - \frac{2\pi}{3}) & \cos(\theta + \frac{2\pi}{3}) \\ \sin\theta) & \sin(\theta - \frac{2\pi}{3}) & \sin(\theta + \frac{2\pi}{3}) \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \tag{10-4}$$

The function `ABToDQ` will do an 3 phase to dq transformation.

10.4 Per Unit Normalization

We do not use per unit normalization in the toolbox models but it is available using the function `PerUnitNormalization`. The normalization quantities are

$$P_B = Q_B = S_B = \frac{V_B I_B}{n_p} \tag{10-5}$$

$$Z_B = \frac{V_B}{I_B} \tag{10-6}$$

$$Y_B = \frac{I_B}{V_B} \tag{10-7}$$

$$\lambda_B = \frac{V_B}{\omega_{e0}} \tag{10-8}$$

$$T_B = \frac{p}{\omega_{e0}} P_B H = \frac{\frac{1}{2} J\omega_{e0}^2}{n_p P_B} \tag{10-9}$$

λ_B is the base flux and p is the number of pole pairs. n_p is the number of phases. ω_{e0} is the base electrical frequency. Neither time nor frequency are normalized. The base voltage is the line-to-line voltage divided by the square root of the number of phases. Table 10-1 on page 53 shows the normalizations

For example, the equation for rotor spin rate change is

$$\frac{d\omega_r}{dt} = \frac{T_e - T_m}{J} \tag{10-10}$$

where J is inertia, T_e is the electrical torque and T_m is the mechanical torque.

```
1     lM  = 69.31;
2     j   = 0.089;
3     p   = 3*746;
4     vL  = 220;
5     nPh = 3;
6     nPP = 2;
7     f   = 60;
```

Table 10-1. Per unit normalizations

Parameter	Units	Normalization
Resistance	Ohm	Z_B
Capacitance	Farad	$\frac{Z_B}{\omega_{e0}}$
Inductance	Henry	$\frac{Z_B}{\omega_{e0}}$
Inertia	kg-m^2	$\frac{n_p P_B}{\frac{1}{2}\omega_{e0}^2}$
Voltage	Volt	$\frac{1}{V_B}$
Current	Amp	$\frac{1}{I_B}$
Torque	Nm	
Flux		$\frac{1}{\lambda_B}$

```
8     d   = PerUnitNormalization( p, vL, nPh, nPP, f )
9     puLM   = PerUnitNormalization( p, vL, nPh, nPP, f,'inductance', lM );
10    disp(sprintf('Lm␣=␣%8.1f␣mH␣%8.1f␣pu',lM, puLM ));
11    puJ    = PerUnitNormalization( p, vL, nPh, nPP, f,'inertia', j );
12    disp(sprintf('J␣=␣%8.3f␣kg-m^2␣H␣=␣%8.4f␣s',j, puJ ));
13
14    d =
15
16         pB: 746
17         vB: 127.02
18         iB: 5.8732
19         rB: 21.626
20         lB: 0.057366
21         cB: 0.057366
22     omegaB: 188.5
23         tB: 11.873
24    lambdaB: 0.67385
25       pNom: 2238
26
27 Lm =      69.3 mH   1208.2 pu
28 J =     0.089 kg-m^2 H =   0.7065 s
```

10.5 Permanent Magnet Generator Model

Brushless permanent magnet machines are of types

1. Surface-mounted magnets, conventional stator
2. Surface-mounted magnetics, air-gap stator winding
3. Internal magnets (flux-concentrating)

The rotor may be inside or outside.

The equations for a permanent magnet machine in direct-quadrature axes are

$$u_q = R_s i_q + \omega_r(L_d i_d + \psi) + \frac{dL_q i_q}{dt} \tag{10-11}$$

$$u_d = R_s i_d - \omega_r L_q i_q + \frac{d(L_d i_d + \psi)}{dt} \tag{10-12}$$

ψ is the flux due to the permanent magnets. The electrical torque is

$$T_e = \frac{3}{2}p((L_d i_d + \psi)i_q - L_q i_q i_d) \tag{10-13}$$

where p is pole pairs.

The mechanical torque equation is

$$T_e = T_w + b\omega_m + J\frac{d\omega_m}{dt} \tag{10-14}$$

where b is the mechanical damping coefficient, T_w is the wind torque, J is the inertia and the relationship between mechanical and electrical angular rate is

$$\omega_e = p\omega_m \tag{10-15}$$

In a magnet surface mount machine with coils in slots $L_d = L_q = L$ and ψ and the inductances are not functions of time. The equations simplify to

$$u_q = R_s i_q + \omega_e L i_d + \omega_e \psi + L\frac{di_q}{dt} \tag{10-16}$$

$$u_d = R_s i_d - \omega_e L i_q + L\frac{di_d}{dt} \tag{10-17}$$

$$T_e = \frac{3}{2}p\psi i_q \tag{10-18}$$

Thus the torque is a function of the quadrature current only. The rotor torque due to the wind is

$$T_w = \frac{1}{2}\rho\pi R^2\frac{C_p(\lambda,\beta)W^3}{\omega_m} \tag{10-19}$$

where ρ is the atmospheric density, W is the wind speed and R is the rotor radius. The rotor angular velocity is the same as the generator mechanical angular rate because this is a direct drive system. β is the pitch angle. The tip speed ratio λ is

$$\lambda = \frac{\omega_m R}{W} \tag{10-20}$$

The power extracted from the wind by the rotor is $T_w\omega_m$ or

$$P_w = \frac{1}{2}\rho\pi R^2 C_p(\lambda,\beta)W^3 \tag{10-21}$$

Thus for a given wind speed we want to select ω_m and β to maximize C_p.

The final set of dynamical equations are

$$\omega_m = \frac{d\theta}{dt} \tag{10-22}$$

$$0 = \frac{1}{2}\rho\pi R^2\frac{C_p(\lambda,\beta)W^3}{\omega_m} - \frac{3}{2}p\psi i_q + b\omega_m + J\frac{d\omega_m}{dt} \tag{10-23}$$

$$u_d = R_s i_d - p\omega_m L i_q + L\frac{di_d}{dt} \tag{10-24}$$

$$u_q = R_s i_q + p\omega_m L i_d + p\omega_m\psi + L\frac{di_q}{dt} \tag{10-25}$$

The equations are nonlinear and bilinear in ω_m and i_q and i_d.

The control vector is

$$u = \begin{bmatrix} u_d \\ u_q \end{bmatrix} \tag{10-26}$$

and the state vector is

$$x = \begin{bmatrix} \theta \\ \omega_m \\ i_d \\ i_q \end{bmatrix} \tag{10-27}$$

10.6 Doubly Fed Induction Generator

The Doubly Fed Induction Generator (DFIG) is a generator in which both the rotor and stator are connected to external sources. The rotor is typically a wound rotor. The torque equation is [3]

$$\frac{d\omega_m}{dt} = \frac{T_m + T_e}{J} \tag{10-28}$$

J is the inertia.

The electrical torque is

$$T_e = \frac{3}{2}p\left(\psi_{ds}i_{qs} - \psi_{qs}i_{ds}\right) \tag{10-29}$$

where r is rotor, s is stator, q is quadrature and d is direct. The synchronous speed is

$$\omega_s = \omega - p\omega_m \tag{10-30}$$

where p is pole pairs. ω is the reference frame rotation speed which need not be constant. The flux linkage equations are

$$\begin{aligned}
\psi_{ds} &= (L_s + L_m)i_{ds} + L_m i_{dr} && (10\text{-}31)\\
\psi_{qs} &= (L_s + L_m)i_{qs} + L_m i_{qr} && (10\text{-}32)\\
\psi_{dr} &= (L_r + L_m)i_{dr} + L_m i_{ds} && (10\text{-}33)\\
\psi_{qr} &= (L_r + L_m)i_{qr} + L_m i_{qs} && (10\text{-}34)\\
& && (10\text{-}35)
\end{aligned}$$

The dynamical flux equations are

$$\begin{aligned}
\frac{d\psi_{ds}}{dt} &= v_{ds} - R_s i_{ds} + \omega\psi_{qs} && (10\text{-}36)\\
\frac{d\psi_{qs}}{dt} &= v_{qs} - R_s i_{qs} - \omega\psi_{ds} && (10\text{-}37)\\
\frac{d\psi_{dr}}{dt} &= v_{dr} - R_r i_{dr} + \omega_s\psi_{qr} && (10\text{-}38)\\
\frac{d\psi_{qr}}{dt} &= v_{qr} - R_r i_{qr} - \omega_s\psi_{dr} && (10\text{-}39)\\
& && (10\text{-}40)
\end{aligned}$$

Assuming that the inductances are constant we can get the dynamical equations in terms of phase currents

$$\begin{aligned}
(L_s + L_m)\frac{di_{ds}}{dt} + L_m\frac{di_{dr}}{dt} &= v_{ds} - R_s i_{ds} + \omega\psi_{qs} && (10\text{-}41)\\
(L_s + L_m)\frac{di_{qs}}{dt} + L_m\frac{di_{qr}}{dt} &= v_{qs} - R_s i_{qs} - \omega\psi_{ds} && (10\text{-}42)\\
(L_r + L_m)\frac{di_{dr}}{dt} + L_m\frac{di_{ds}}{dt} &= v_{dr} - R_r i_{dr} + \omega_s\psi_{qr} && (10\text{-}43)\\
(L_r + L_m)\frac{di_{qr}}{dt} + L_m\frac{di_{qs}}{dt} &= v_{qr} - R_r i_{qr} - \omega_s\psi_{dr} && (10\text{-}44)\\
& && (10\text{-}45)
\end{aligned}$$

We get an inductance matrix of the form

$$L = \begin{bmatrix} L_s + L_m & 0 & L_m & 0 \\ 0 & L_s + L_m & 0 & L_m \\ L_m & 0 & L_r + L_m & 0 \\ 0 & L_m & 0 & L_r + L_m \end{bmatrix} \tag{10-46}$$

and

$$L\frac{d}{dt}\begin{bmatrix} i_{ds} \\ i_{qs} \\ i_{dr} \\ i_{qr} \end{bmatrix} = \begin{bmatrix} v_{ds} + R_s i_{ds} + \omega_s \psi_{qs} \\ v_{qs} + R_s i_{qs} - \omega_s \psi_{ds} \\ v_{dr} + R_r i_{dr} + \sigma\omega_s \psi_{qr} \\ v_{qr} + R_r i_{qr} - \sigma\omega_s \psi_{dr} \end{bmatrix} \tag{10-47}$$

The reactive power from the stator is

$$Q = 3\left(v_{ds} i_{qs} - v_{qs} i_{ds}\right) \tag{10-48}$$

The active power from the stator is

$$P = 3\left(v_{ds} i_{ds} + v_{qs} i_{qs}\right) \tag{10-49}$$

There are similar equations for the rotor. The state vector is

$$x = \begin{bmatrix} \theta \\ \omega_r \\ i_{ds} \\ i_{qs} \\ i_{dr} \\ i_{qr} \end{bmatrix} \tag{10-50}$$

CHAPTER 11

MECHANICAL

11.1 Overview

The `Mechanical` folder contains design tools for the mechanical subsystem. These include models for mechanisms, joint design calculation functions, and calculations specific to wind turbine design.

11.2 Mechanism Models

11.2.1 Introduction

Functions modeling common mechanical parts are included for mechanical design analysis. These mechanism models input user-supplied part specifications and output performance parameters that can be used in system design. The mechanisms included are

1. bearing
2. coupling
3. gear box
4. helical gear
5. spur gear

11.2.2 Bearing

The file `BearingLife.m` models a bearing under operating conditions given by the user. The function returns expected bearing life based on the following equation:

$$L = \frac{10^6}{60} n(C/P)^t \tag{11-1}$$

where L is the expected lifetime of a bearing, n is the rotational speed of operation, C is the bearing load rating, P is the equivalent load acting on the bearing, and t is an exponent dependent on the type of bearing used. For ball bearings, this value is 3.

11.2.3 Coupling

CouplingStress.m calculates the axial and shear stresses acting on a coupling based on load conditions and the rotational speed of operation.

11.2.4 Rectangular Beam

Likewise, RectangularBeam.m calculates the axial and shear stresses acting on a rectangular beam based on its load conditions.

11.2.5 Gear Box

An ideal gear box is included. The ratio n greater than or less than zero depending on the direction of

```
[omegaOut, torqueOut] = Gearbox( n, omegaIn, torqueIn )
```

The model is

$$\omega_o = \frac{1}{n}\omega_i \tag{11-2}$$

$$T_0 = nT_i \tag{11-3}$$

where T is torque and ω is angular rate.

11.2.6 Gears

The function SpurGear.m determines the maximum tooth load, output torque and power for a common spur gear based on its dimensions and tooth form factor.

```
[ Load, Torque, Power ] = SpurGear( d )
```

Similarly, HelicalGear.m computes these values for a helical gear, in addition to the maximum axial thrust load.

```
[ Load, Torque, Power, Thrust ] = HelicalGear( d )
```

11.3 Joint Calculators

11.3.1 Introduction

Joint calculation functions contain computations based on design guidelines for various joining methods, including

1. welds
2. interference fits

11.3.2 Welds

WeldThroatArea.m uses ISO standards to calculate the minimum throat area for joining with welds, with the input being the load supported by the welded parts.

```
a = WeldThroatArea( l )
```

11.3.3 Interference Fits

The file `InterferenceFit.m` gives the amount of torque that an interference fit can transfer. Typing

```
>> help FileName
```

in your MATLAB window gives input and output information on that function. The help information for `InterferenceFit.m` gives the following:

```
>> help InterferenceFit
 ---------------------------------------------------------------------------
   Calculates torque transferred in an interference fit, and gives the
   factor of safety for the stress distribution in the fit.
 ---------------------------------------------------------------------------
   Form:
   [T, FOS] = InterferenceFit ( d )
 ---------------------------------------------------------------------------

   ------
   Inputs
   ------
   d           (1,1)  Data structure
                      .delta      (1,1)  Total diametral interference (m)
                      .ri         (1,1)  Inner radius of shaft (m)
                      .ro         (1,1)  Outer radius of hub (m)
                      .r          (1,1)  Radius of interference (m)
                      .Eo         (1,1)  Young's modulus (Pa)
                      .Ei         (1,1)  Young's modulus (Pa)
                      .mu         (1,1)  Viscosity (Pa*s)
                      .nuo        (1,1)  Poisson's ratio of Hub
                      .nui        (1,1)  Poisson's ratio of shaft
                      .l          (1,1)  Length of engagement (m)

   -------
   Outputs
   -------
   T           (1,1)  Torque (N*m)
   FOS         (1,1)  Factor of Safety

 ---------------------------------------------------------------------------
   Reference: Robert L. Norton. Machine Design: An Integrated Approach.
              Upper Saddle River, NJ: Prentice Hall, 2000
 ---------------------------------------------------------------------------
```

11.4 Blade Static Approximations

11.4.1 Introduction

Blade statics approximation functions account for various design calculations for including

1. drag force
2. hinge moment
3. moment of inertia
4. bending stress
5. bearing distances

11.4.2 Drag Force

The maximum drag force acting on a plate-shaped blade as it travels through a fluid can be calculated using `DragForce.m`. The user must input the blade height, chord, speed, and the fluid density.

```
[F] = DragForce( d )
```

11.4.3 Hinge Moment

`HingeMoment.m` calculates the moment at the hinge joint of a blade on a vertical axis wind turbine.

```
M2 = HingeMoment( d )
```

11.4.4 Moment of Inertia

The file `NACABladeMOI.m` gives an approximation for the moment about an axis on a symmetrical NACA blade.

```
MOI = NACABladeMOI(d)
```

11.4.5 Bending Stress

`RotBeamBending` models a beam rotating about an axis that is perpendicular to it. It gives the bending stress in the beam based on input parameters. This calculation is a simple model for the bending stress acting upon the vanes of a VAWT.

```
[bS, FOS] = RotBeamBending( d , r , omega)
```

11.4.6 Bearing Distances

For cases where two bearings are needed on a rotorshaft, `VAWTBearingDistance` calculates the minimum distance between the bearings based on the characteristics of the shaft.

```
dist = VAWTBearingDistance( d, yS, F, FoS )
```

CHAPTER 12

Multibody Models

12.1 Introduction

This chapter discusses the multibody model included in the toolbox. This model is of a generic wind turbine with a flexible mast and blades and with additional articulated degrees of freedom between the two. It can be configured to be a vertical axis or horizontal axis wind turbine. The model is based on Tsai [18]. The multi-body model runs within the WCTSim framework.

12.2 Background

12.2.1 Tree

The multibody model is a topological tree that can have any number of branches but no closed loops. The base node must be attached to the ground.

12.2.2 Hinges

Two types of hinges are used in this assembly. The first is a revolute joint. This is a joint with one rotational-degree-of-freedom and no translational degrees of- freedom. The second is a prismatic joint. This is a joint with one translational degree-of-freedom and no rotational degrees-of-freedom. The two types of joints are illustrated in Figure 12-1 on page 62. The figure also shows a link which is two joints connected by a solid bar We define a link as a solid piece of material with hinges on each end. This link has a revolute joints at both ends. Note that the axis $zi - 1$ is associated with joint i and the axis z_i is associated with $i + 1$. For a revolute joint the axis z is along the rotation axis. For a prismatic joint, the z axis is along the translational axis.

12.2.3 Computations

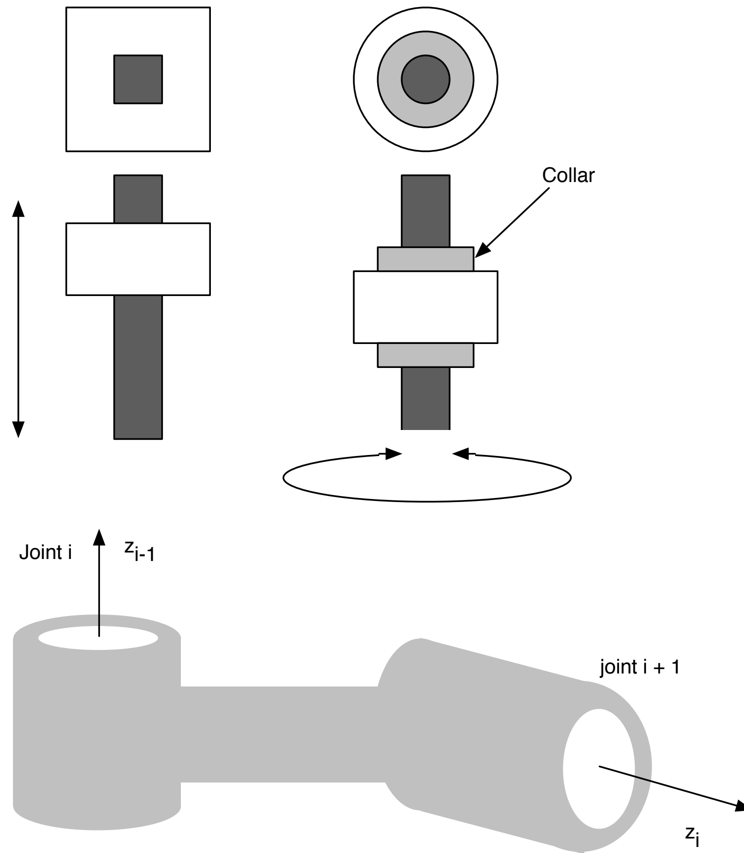
In the forward computation the angular velocities, linear velocities, angular accelerations, linear accelerations and gravity vector are propagated forward from the base to each joint.

The forward rates for revolute joints are

$$v_k = v_{k-1} + \omega_k \times r_k \tag{12-1}$$

$$\omega_k = \omega_{k-1} + z_{k-1}\dot{\theta}_k \tag{12-2}$$

Figure 12-1. Joints



It is necessary to keep track of the elements in the path to each hinge since there may be branches.

The forward accelerations for revolute joints are

$$v_k = \dot{v}_{k-1} + \dot{\omega}_k \times r_k + \omega_k \times (\omega_k \times r_k) \quad (12\text{-}3)$$
$$\dot{\omega}_k = \dot{\omega}_{k-1} + z_{k-1}\ddot{\theta}_k + \omega_{k-1} \times z_{k-1}\dot{\theta}_k \quad (12\text{-}4)$$

Once the velocities and accelerations have been propagated to the end of the chain, the joint forces and moments can be computed started at the end of each chain.

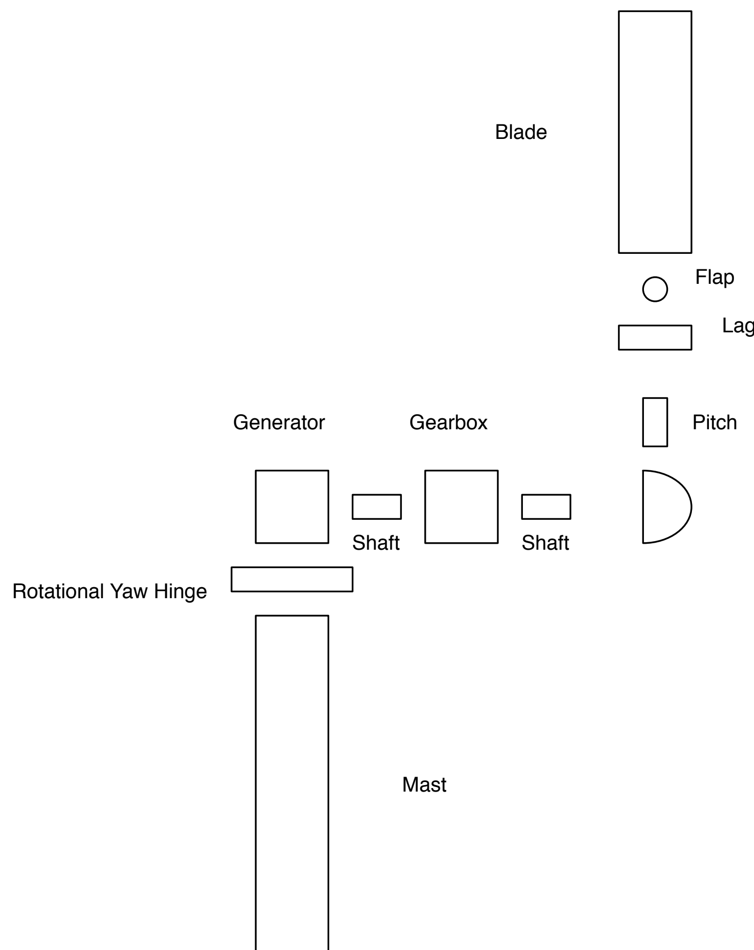
$${}^k f_{k,k-1} =^k f_{k+1,k} - m_k^k g -^k f_k^* \quad (12\text{-}5)$$
$${}^k n_{k,k-1} =^k n_{k+1,k} + ({}^k r_k +^k r_{ck}) \times^k f_{k,k-1} -^k r_{ck} \times^k f_{k+1,k} -^k n_k^* \quad (12\text{-}6)$$

where$f_{k,k-1}$ is the resulting force exerted on link k by link $k-1$ at point O_{k-1}, f_k^* is the inertia force exerted at the center of mass of link k, $n_{k,k-1}$ is the resulting moment exerted on link k by link $k-1$ at point O_{k-1}, n_k^* is the inertia moment exerted at the center of mass of link k, r_k is the position vector of the origin of the kth link frame with respect to the $(k-1)^{th}$ link frame and r_{ck} is the position vector of the center of mass of link k with respect to the k^{th}link frame.

12.3 Example

As an example we will model a horizontal axis wind turbine with 3 blades. The model is shown in Figure 12-2 on page 63. Starting from the base, the model has the following degrees of freedom

Figure 12-2. Multibody model



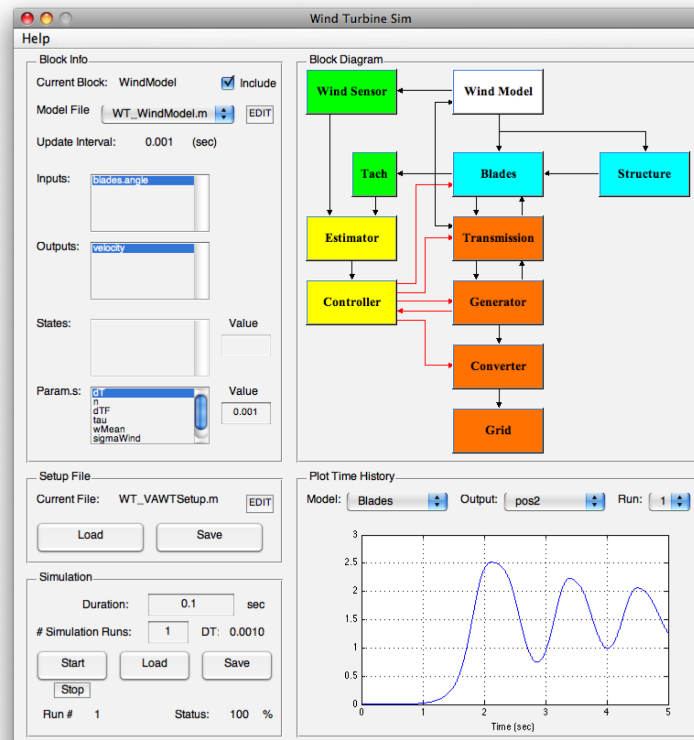
1. mast cantilevered to the ground
2. rotational yaw hinge
3. torsional spring shaft
4. gearbox
5. torsional spring shaft
6. pitch
7. lag hinge
8. flap hinge
9. blade

The gearbox model allows any function to be included that relates torque and rotational rate at the two sides of the gearbox. Thus backlash and other nonlinear effects can be included in the model. The generator model is any desired generator function in the toolbox.

Type WTSim to start the simulation. Under Setup File click Load and select `WT_VAWTSetup.m`. The setup file specifies which models (m-files) will be used for each block in the wind turbine simulation. In this setup file, the “Blades” block is modeled with `WT_MultiBody.m`. To begin the simulation, set the Duration to 0.1 and hit Start. When it is done look under model “Blades” for the states. The multi-body model used in this simulation has a single blade assembly that rotates around a vertical shaft. Note that the purpose of the simulation is to illustrate the general

use of multi-body dynamics, rather than to provide a detailed engineering model for a VAWT. Figure 12-3 on page 64 shows the GUI.

Figure 12-3. WTSim with multibody model



To create a complete wind turbine simulation with multi-body dynamics, one would need to:

1. set up an appropriate multi-body model for a wind turbine
2. if necessary, create an alternate controller for wind turbine sims that sends the appropriate voltage commands,
3. compute the power and torque generated to supply as outputs

CHAPTER 13

UTILITIES

13.1 Introduction

This chapter discusses utility functions. These functions are found in the Utility folder of the toolbox. This chapter only covers a few of the utility functions.

13.2 Reynold's Number

The function `ReynoldsNumber` computes the Reynold's number for a fluid.

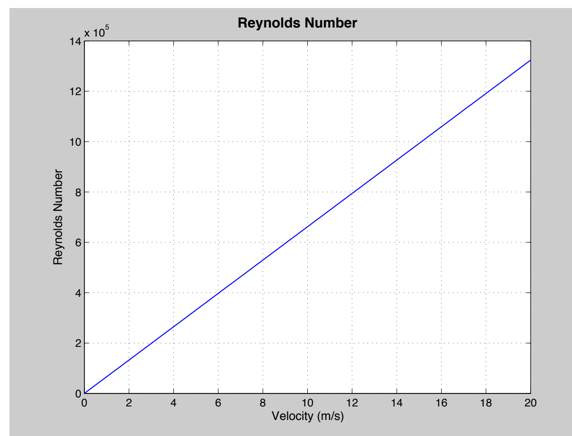
```
rE = ReynoldsNumber( u, l, nu )
```

The inputs are the wind speed, the length and the kinematic viscosity. The wind speed may be a vector. If you type

```
ReynoldsNumber
```

You will get Figure 13-1 on the facing page shows the loaded data.

Figure 13-1. Reynolds number



The units in the header are metric but any set of consistent units will work.

13.3 DrawHAWT

This function animates a horizontal axis wind turbine. This is meant to show the movement of the blades and turbine in a cartoon form. The function allows for variable yaw angle, generator angle, propeller angle and blade angles. The length, diameter of the tower and the length and chord of the blades can be specified. All other parameters are generated automatically.

You first need to initialize the function

```
DrawHAWT( 'initialize', name, lBlade, chordBlade, lTower, dTower, nBlades )
```

Then update each time step

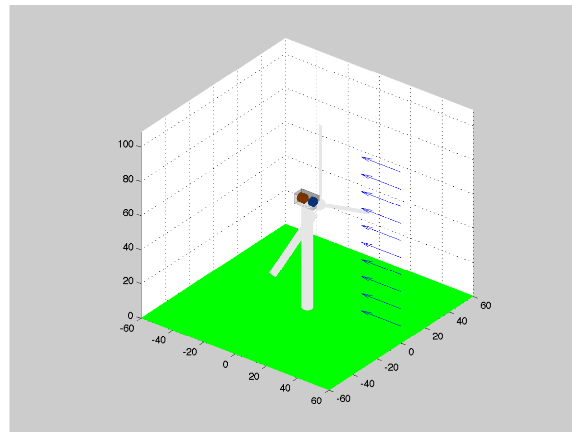
```
DrawHAWT( 'update', yaw, angleGen, angleProp, angleBlade, windVector )
```

When you are done close the window

```
DrawHAWT( 'close' )
```

For a demo type `DrawHAWT`. At the end of the animation you will see Figure 13-2 on the next page.

Figure 13-2. HAWT at the end of the demo animation



13.4 DrawVAWT

This function animates a vertical axis wind turbine. This is meant to show the movement of the blades and turbine in a cartoon form. The function is invoked with the following three calls.

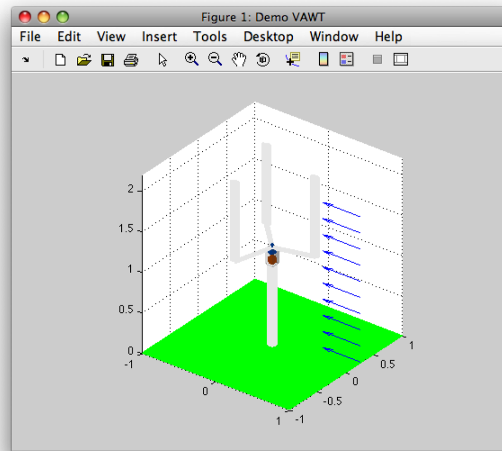
```
DrawVAWT( 'initialize', name, lBlade, lArm, chordBlade, lTower, dTower, nBlades )
DrawVAWT( 'update', angleGen, angleProp, angleBlade, windVector )
DrawVAWT( 'close' )
```

The following code runs a demo. It is the same demo that you will get if you type `DrawVAWT`.

```
 n           = 100;
DrawVAWT( 'initialize', 'Demo␣VAWT', 1, 0.5, 0.1, 1, 0.1, 3 )
angleGen    = linspace(0,100*pi);
angleProp   = linspace(0, 10*pi);
angleBlade  = zeros(3,n);
wind        = [-ones(1,n);zeros(2,n)];
pause(3)
for k = 1:n
```

```
        DrawVAWT( 'update', angleGen(k), angleProp(k), angleBlade(:,k), wind );
        pause(0.1)
end
pause(3)
DrawVAWT( 'close' );
```

Figure 13-3. VAWT at the end of the demo animation



13.5 LiftAndDragCoeff

This function generates lift and drag coefficients using analytical relationships between the physical parameters of an airfoil.

```
[cL, cD] = LiftAndDragCoeff( d, alpha )
```

Figure 13-4 on page 68 shows lift and drag coefficients for an airfoil found by typing

```
LiftAndDragCoeff
```

Lift is linear with angle of attack but because the drag coefficient goes as the square of the lift coefficient we see a quadratic relationship. This function does not model stall or other effects. You can generate coefficients from blade models using the Airfoil functions.

13.6 PowerFromActuatorDisk

This function computes power from actuator disk theory.

```
[p, cP] = PowerFromActuatorDisk( rho, A, v0, a )
```

This provides a limit on how much power a wind turbine can produce. a is defined from the relationship $v_3 = v_0(1 - 2a)$ where v_0 is the wind speed in front of the machine and v_3 is the wind speed at the end of the wake. Note that $v_3 = 0$ is NOT the optimum value. The optimum value is with $a = 1/3$ so $c_P = 16/27$. This can never be achieved in practice due to aerodynamic losses.

Figure 13-5 on the next page shows power as a function of wind velocity

```
PowerFromAD
```

Figure 13-4. Lift and drag coefficients

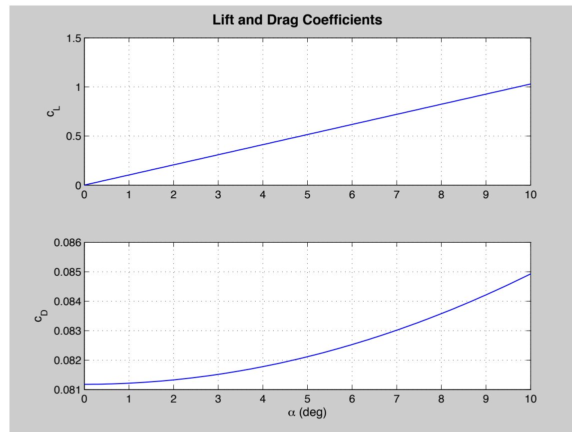
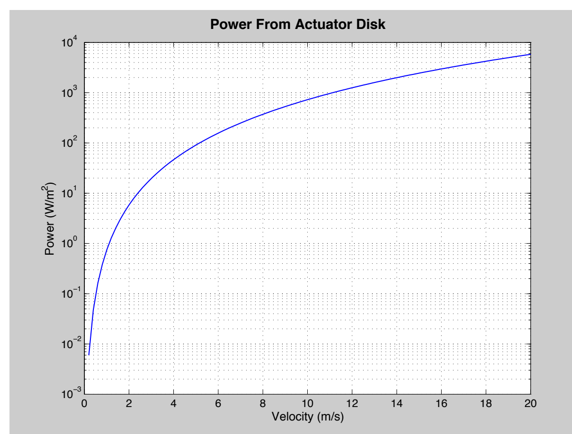


Figure 13-5. Power from actuator disk theory



CHAPTER 14

WINDDATA

14.1 Introduction

This chapter discusses the wind data models available in the toolbox.

14.2 Wind Data

Wind data is available for the continental United States (lower 48 states) Hawaii and Alaska. The data is read in from the binary files using

```
LoadShapeFile
```

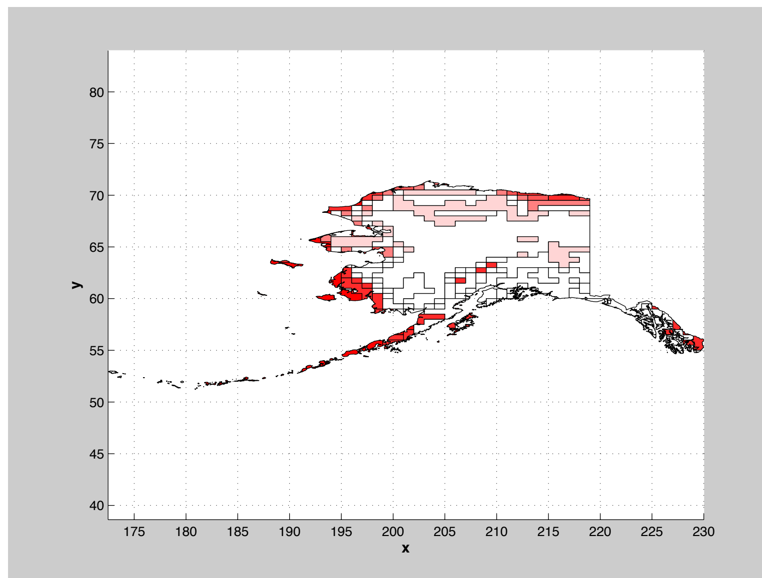
For example to load in the Alaska data type

```
LoadShapeFile('akwindclp.shp')

Shape file: akwindclp
Shape type is Polygon

Boundaries:
 xMin     0.00
 yMin    -2.00
 xMax     2.00
 yMax     2.00
 zMin     2.00
 zMax     0.00
 mMin     0.00
 mMax     0.00

nPolygons =

   728

ff =

AREA

ff =

PERIMETER

ff =
```

```
29
30 ANNUALDD05
31
32 ff =
33
34 ANNUALDD05
35
36 ff =
37
38 GRID_CODE
```

Figure 14-1 on page 70 shows the wind intensity. Dark red is the highest wind and white is very little wind.

Figure 14-1. Alaska



When using high resolution models the map may not appear when running on Mac OS X due to Java memory limitations.

The other models are "I48wndatlas.shp" and "hiwindpolyclp.shp". The function returns a data structure

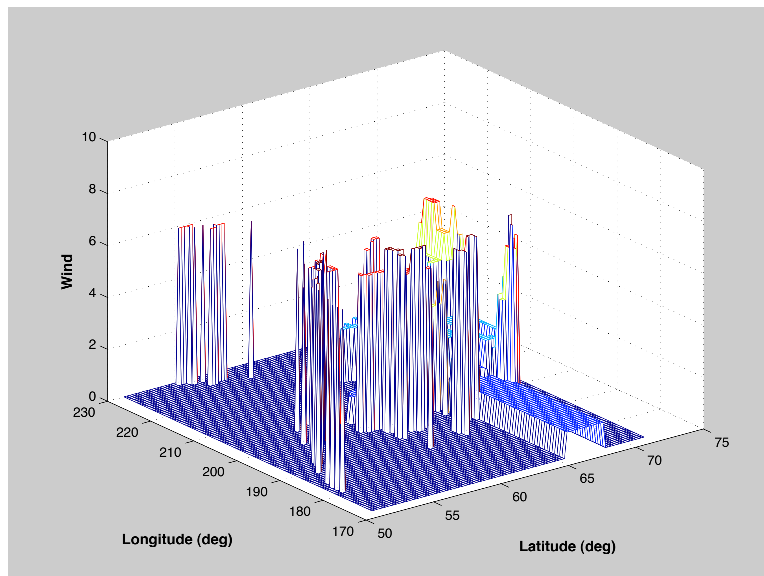
```
1 %   w          (1,1)  Data structure
2 %                      .lon       {n} (1,:) Longitude of polygon vertices (deg)
3 %                      .lat       {n} (1,:) Latitude of polygon vertices (deg)
4 %                     .wind(n)              Wind values
5 %                      .lonRange (1,2)      Longitude range [min max]  (deg)
6 %                      .latRange (1,2)      Latitude range [min max]  (deg)
7 %                      .name     (1,:)      File name
```

This data structure can be converted to wind as a function of latitude and longitude using

```
1 w = LoadShapeFile( filename );
2 wind = WindFromLatLon( w )
```

This function searches all the polygons generated by "LoadShapeFile" and computes the wind at each latitude and longitude. As a consequence it will take a long time if you ask for a lot of points at once. If you are going to generate data for a lot of latitudes and longitudes it is best to run this function and save the resulting data in a .mat file. Figure 14-2 on the next page shows a plot generated by the function.

Figure 14-2. Wind for latitudes and longitudes



CHAPTER 15

WIND MODELS

15.1 Introduction

Wind models include stochastic and dynamical models.

15.2 Dynamical Models

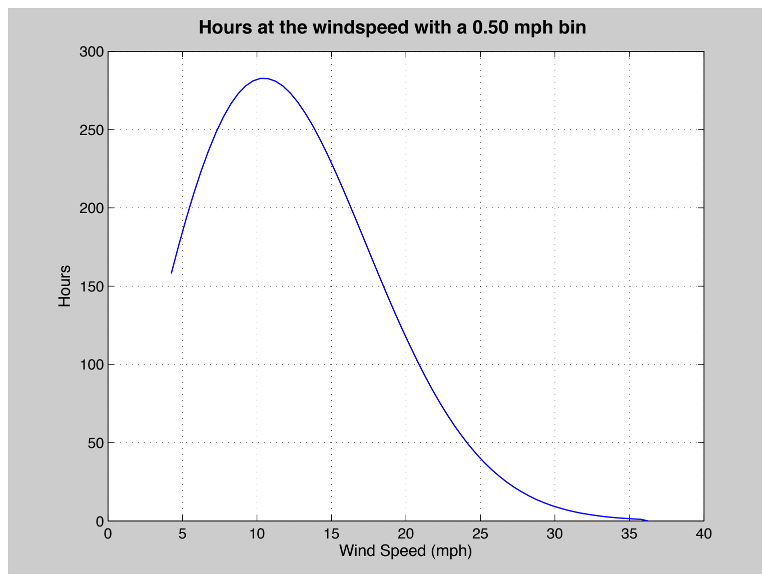
15.2.1 WindspeedHours

```
1 WindspeedHours( kRef, hRef, cRef, h, v, dV, u )
```

This function computes the hours that the wind is between v(k) +/- dV/2.

Figure 15-1 on page 73 shows the demo results

Figure 15-1. Wind hours versus wind speed



WindspeedHours

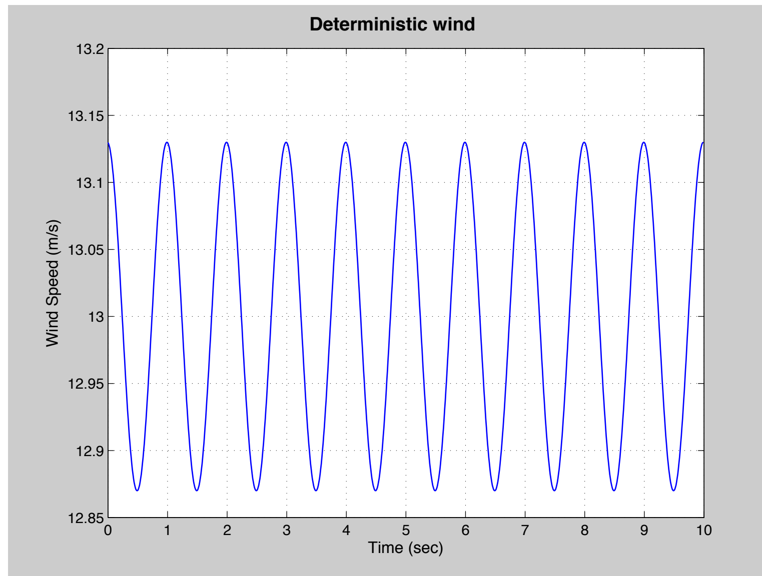
15.2.2 WindDeterministic

```
WindDeterministic( d )
```

This is a deterministic wind model. It models the tower effect.

Figure 15-2 on the next page shows the deterministic wind demo.

Figure 15-2. Deterministic wind



15.2.3 WindStochastic

```
WindStochastic( mode, d )
```

Generates a discrete time model using a zero order hold for the wind noise filters. First call with

```
WindStochastic( 'init', d )
```

to initialize the model. Then call

```
w = WindStochastic( 'run', d )
```

to get the stochastic wind model. Figure 15-3 on the facing page shows the deterministic wind demo.

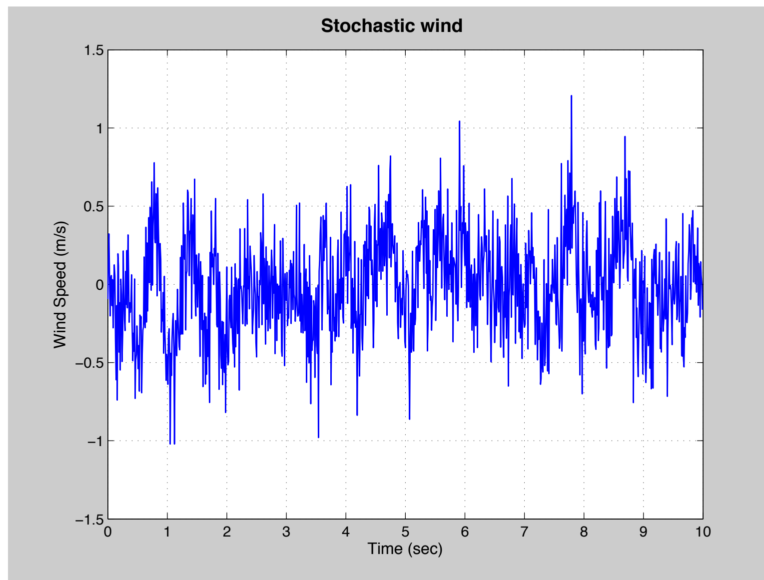
15.2.4 WindAdmittance

```
[a,b,c,d] = WindAdmittance( harm, dTF, dT )
```

Creates a wind admittance filter of the form

$$x_{k+1} = ax_k + bu_k \quad (15\text{-}1)$$
$$y_k = cx_k + du_k \quad (15\text{-}2)$$

Figure 15-3. Stochastic wind



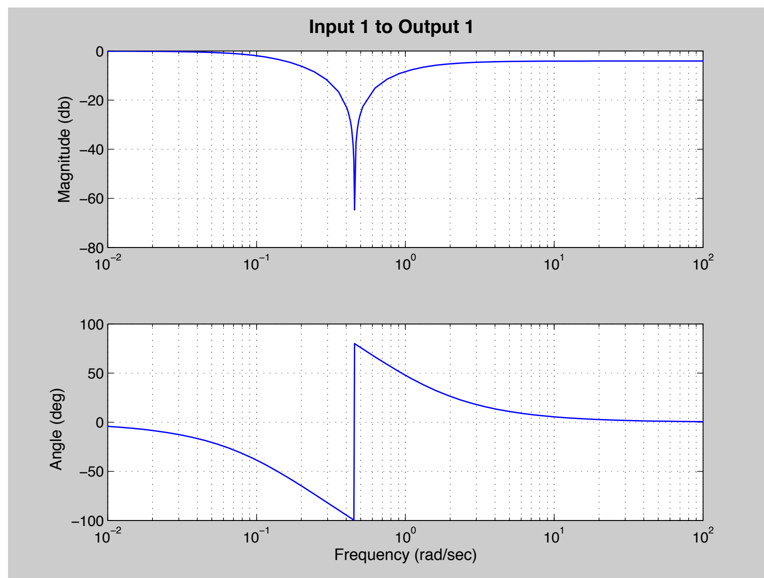
if a time step is entered otherwise it generates

$$\dot{x} = ax + bu \tag{15-3}$$

$$y = cx + du \tag{15-4}$$

Figure 15-4 on the next page shows the admittance filter Bode plot

Figure 15-4. Admittance wind



15.2.5 Wind

"Wind" combines all the effects. It generates a discrete time model using a zero order hold for the wind noise filters. First call with

```
Wind( 'init', d )
```

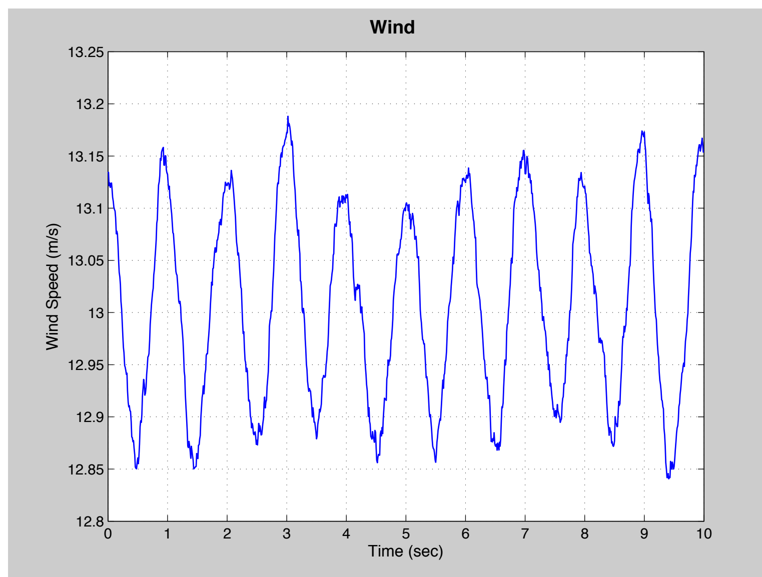
to initialize the model. Then call

```
  w = Wind( 'run', d ) to
```

to get the wind model. This model is designed for 3 blade horizontal axis wind turbines as the harmonic filters are order 3.

Figure 15-5 on the following page shows the demo.

Figure 15-5. Wind model



BIBLIOGRAPHY

[1] Danish Wind Energy Association. Asynchronous (induction) generators. http://www.windpower.org/en/tour/wtrb/async.htm.

[2] Danish Wind Energy Association. Synchronous Generators. http://www.windpower.org/en/tour/wtrb/syncgen.htm.

[3] H. Wayne Beaty and Jr. James L. Kirtley. *Electric Motor Handbook*. McGraw-Hill, 1998.

[4] F. D. Bianchi, H. De Battista, and R. J. Mantz. *Wind Turbine Control Systems: Principles, Modelling and Gain Scheduling Design*. Springer, 2006.

[5] M.C. Claessens. *The design and testing of airfoils for application in small vertical axis wind turbines*. 2006. Delft University of Technology Master of Science Thesis.

[6] P. G. Estevez. *Modeling of a variable speed wind turbine*. Buenos Aires, Argentina, 2007. Universidad de Buenos Aires Senior Thesis.

[7] J. Cardona. Flow curvature and dynamic stall simulated with an aerodynamic free-vortex model for VAWT . *Wind Engineering*, 18(3):135–143, 1984. http://www.awea.org/pubs.

[8] A.M. Kuethe and C. Chow. *Foundations of Aerodynamics, Fifth Edition*. John Wiley and Sons, 1998.

[9] P Lobato, A. Cruz, J. Silva, and J. Pires, A. The switched reluctance generator for wind power conversion.

[10] H. Nikkhajoei, A. Tabesh, and R. Iravani. A Matrix Converter Based Micro-Turbine Distributed Generation System. *IEEE Transactions on Power Delivery*, 20(3):2182–2192, 2005.

[11] R. Noll and N. Ham. Effects of dynamic stall on swecs. *Journal of solar energy engineering*, 104:96–101, 1982.

[12] Z. Pan, J. Ying, and Z. Hui. Study on switched reluctance generator*. *Journal of Zhejiang University SCIENCE*, 5(5):594–602, 2004.

[13] N.C.K. Pawsey. *Development and evaluation of passive variable-pitch vertical axis wind turbines*. 2002. University of New South Wales Doctor of Philosophy Thesis.

[14] D.J. Sharpe. Wind turbine aerodynamics. In *Wind energy conversion systems*, 1990. edited by L.L. Freris.

[15] Y. Sozer and D. A. Torrey. Closed loop control of excitation parameters for high speed switched-reluctance generators. 2003.

[16] Y. Staelens, F. Saeed, and I. Paraschivoiu. A straight-bladed variable pitch VAWT concept for improved power generation. In *Proc. 41st Aerospace Sciences Meeting and Exhibit*, Reno, NV, 2003.

[17] D. A. Torrey. Switched Reluctance Generators and Their Control. *IEEE Transactions on Industrial Electronics*, 49(1):3–14, 2002.

[18] L. Tsai. *Robot Analysis*. Wiley Interscience, 1999.

[19] P. Zai-ping, J. Ying, and Z. Hui. Switched Reluctance Generators for Wind Energy Applications. In *Proc. Power Electronics Specialists Conference*, volume 1, pages 559–564, 1995.

[20] P. Zai-ping, J. Ying, and Z. Hui. Study on Switched Reluctance Generator. *Journal of Zhejiang University SCIENCE*, 5(5):594–602, 2004.