DSWE R Package User Help Document

Prepared by Dr. Yu Ding's Research Group at
Georgia Institute of Technology

Last updated: February 22, 2024

Contact: yu.ding@isye.gatech.edu

Table of Contents

1 Differences between DSWE's R package and Python package.

The functions in the two DSWE packages closely mirror each other, but with the following differences:

A. DSWE R package does not have the deep learning power curve function (DNNPowerCurve).
B. DSWE Python package does not have the spline-based power curve function (SplinePCFit).
C. DSWE Python package's BayesTreePowerCurve function uses BartPy, which is the Python implementation of BART. BartPy was not implemented by the original authors of BART and its results differ slightly from the BART function in R. So, the difference between the R and Python versions of the Bayes Tree power curve is larger than those between other power curve functions in the two packages.
D. For the time being, AMK in Python has to take the bandwidth parameters from the R package, as the optimal bandwidth selection algorithm, i.e., the direct plug-in (DPI) algorithm, is still being implemented. Once DPI is implemented, AMK in Python will be stand alone.
E. DSWE Python package does not have the Energy Decomposition function (deltaEnergy).

2 How do I install the package?

The package is available through CRAN, the official package repository of R, as well as on GitHub. This package is tested on and is compatible with **R of version 3.5.0 or above**. When using an earlier R version, problems may arise.

The package should be installed using the standard `install.packages()`command in R or RStudio:

```
install.packages("DSWE")
```

Note: The package contains C++ source codes, and thus can be installed using the pre-compiled binaries available on CRAN or compiling the source code itself. When installing, R might ask if one wants to install the binaries or compile from source. Please select the latter only if you have the necessary tools to compile C++ code. For those who wish to compile from source, we provide the details on how to get the necessary tools in an optional section at the end of this document.

How to update the package to a new version?

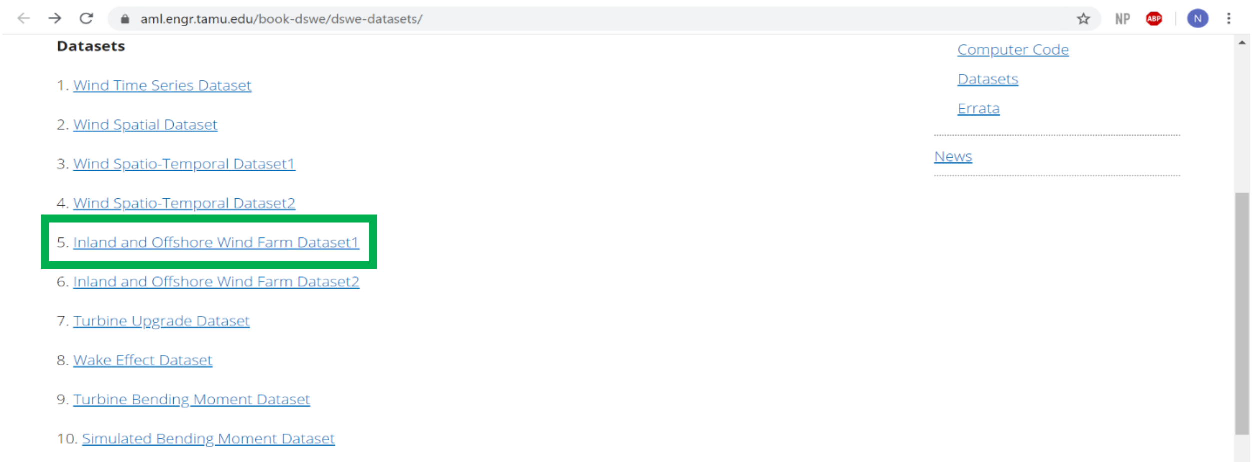
Use the following command in R or RStudio:

```
update.packages("DSWE")
```

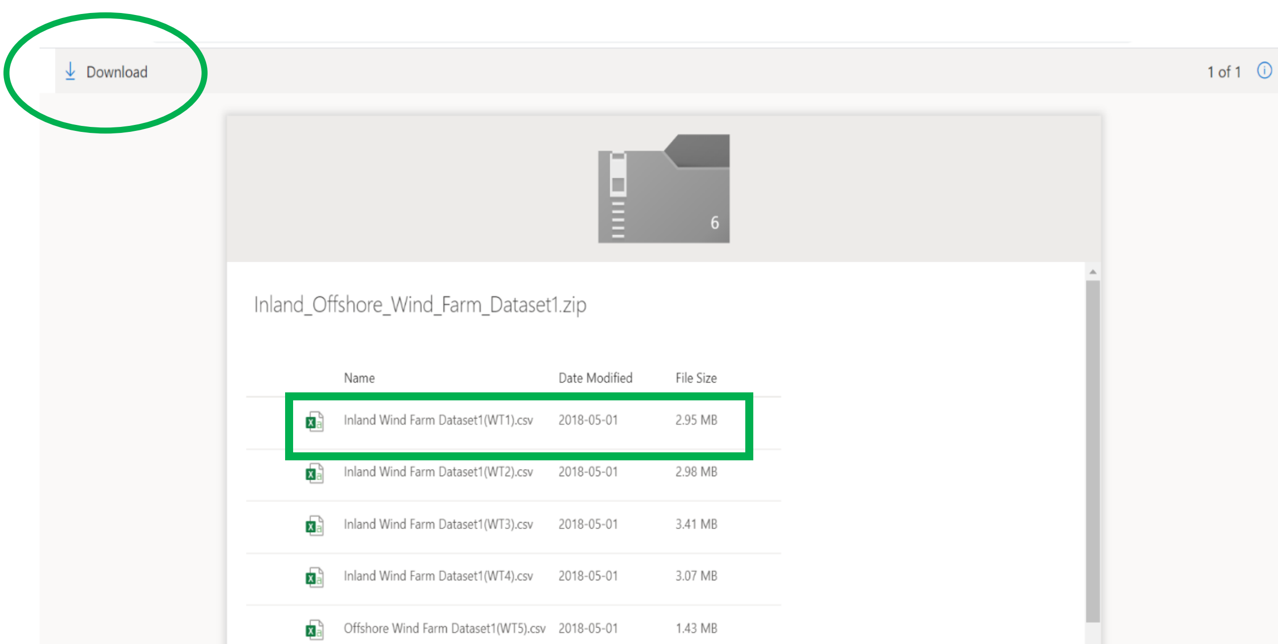
3 How to use KNN to fit a multi-dimensional power curve?

Step1: Download the sample data set as shown.

Visit site using the following link - *https://sites.google.com/view/yuding/book-dswe/dswe-datasets*. The page looks like as shown below and select the option 5.



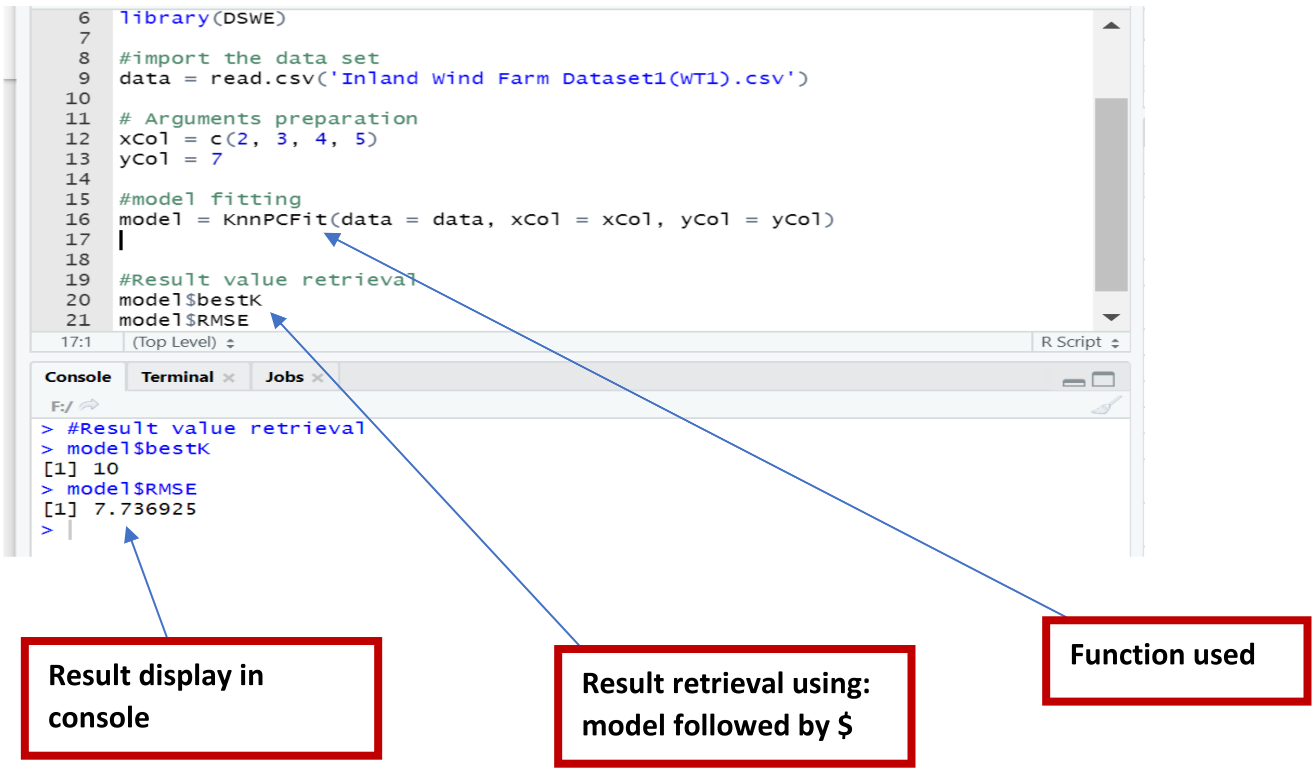
Download the sample data set as shown below in green boxes. After downloading, save the file in working directory.



Step 2: Set the path containing data set to a current working directory. Further load the package and import the data set as shown.

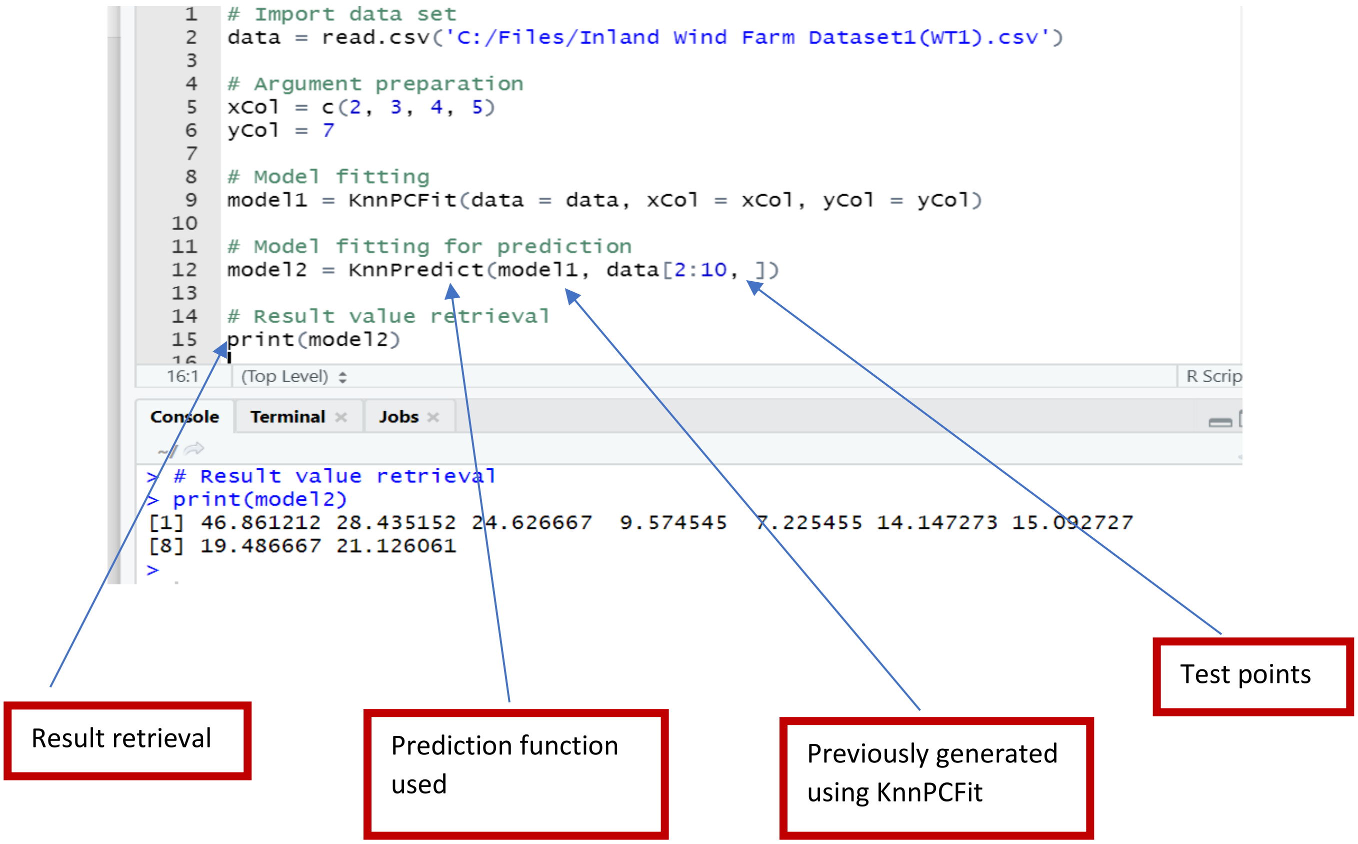
```r
#setting the work directory which contains data set
setwd('F:/')

#loading the package
library(DSWE)

#import the data set
data = read.csv('F:/Inland Wind Farm Dataset1(WT1).csv')

```

Step 3: Prepare the arguments to fit a power curve as shown below.



Note: - The RMSE reported is based on generalized cross validation on training set. To obtain a prediction on a test point, follow the next step.

Step 4: Prediction on a test point using the model generated from KnnPCFit and using the function KnnPredict as shown below.



How to select the best subset of variables in building a multi-dimensional power curve?

Step 1: The data set mentioned in previous questions will be used to demonstrate the usability of subset selections. The package is loaded and data is imported as shown.

```r
#setting the work directory which contains data set
setwd('F:/')

#loading the package
library(DSWE)

#import the data set
data = read.csv('F:/Inland Wind Farm Dataset1(WT1).csv')

```

Step 2: Prepare the arguments to fit the KNN power curve and retrieve the best subset as shown.

```r

#setting the work directory which contains data set
setwd('F:/')

#loading the package
library(DSWE)

#import the data set
data = read.csv('Inland Wind Farm Dataset1(WT1).csv')

# Arguments preparation
xCol = c(2, 3, 4, 5)
yCol = 7

#model fitting
model = KnnPCFit(data = data, xCol = xCol, yCol = yCol, subsetSelection = T)


#Result value retrieval
model$xCol

```

```
21:1    (Top Level) ÷                                        R Script ÷
Console   Terminal ×   Jobs ×
F:/
> #Result value retrieval
> model$xCol
[1] 2 3 4 5
>
```

Feature Column supplied

Retrieve best subset column

KNN model fit with subset selection

Suppose one has built a KNN power curve model using some data. When the new data comes in, how can one update it periodically?

Step 1: The data set mentioned in previous questions will be used to demonstrate model update. The package is loaded and data is imported as shown.

```r
#setting the work directory which contains data set
setwd('F:/')

#loading the package
library(DSWE)

#import the data set
data = read.csv('F:/Inland Wind Farm Dataset1(WT1).csv')

```

Step 2: Prepare the arguments to update the previously obtained model using new data as shown.

```r
setwd('F:/')

#loading the package
library(DSWE)

#import the data set
data = read.csv('Inland Wind Farm Dataset1(WT1).csv')

# Arguments preparation
xCol = c(2, 3, 4, 5)
yCol = 7

#model fitting
model = KnnPCFit(data = data, xCol = xCol, yCol = yCol)

#new data
newdata = data[300:600, ]

#update model
modelupdate = KnnUpdate(knnMdl = model, newData = newdata)

```

23:1 (Top Level) R Script

Console Terminal Jobs

F:/

```
> #update model
> modelupdate = KnnUpdate(knnMdl = model, newData = newdata)
>
```

Model Update Function

Previous model

New data

4 How to use temporal Gaussian process (tempGP) model to fit a multi-dimensional power curve?

Note:

The temporal Gaussian Process, or tempGP, is a Gaussian process-based model, and hence the inference (fitting) can be computationally expensive for large datasets. In order to overcome this problem, we implemented an acceleration method using the method of Vecchia.

Vecchia has a few hyperparameters to tune. We have tuned those parameters to strike a balance between prediction accuracy and computation time. We advise to use the default parameter settings. The fast computation is expected to be within four minutes on a single core of a modern computer, in contrast to hours of computation required by the original tempGP. The accelerated version is set as the default option for the tempGP function.

We expect the fast computation results to be 1-5% worse than that of the original version of tempGP. If accuracy is extremely important to the user at the cost of orders of magnitude of increased computation time, then one can easily set the argument `vecchia` to `FALSE` and `limit_memory` to None to use the original version of tempGP.

Example:

For this example, we use data1 in the DSWE package.

Step 1: Divide the training dataset into an input variable matrix and a response vector. One can also create a vector of time indices for the training data points. If the time indices are not created, the code assigns positive integers starting from 1 as the time indices. For example, if there are 100 training data points, the code will assign the time indices from 1 to 100.

```
data = DSWE::data1
trainindex = 1:5000 #using the first 5000 data points to train the model
traindata = data[trainindex,]
xCol = c(2:6) #input variable columns
yCol = 7 #response column
tCol = 1 #column with the time indices
trainX = as.matrix(traindata[,xCol])
trainY = as.numeric(traindata[,yCol])
trainT = as.numeric(traindata[,tCol])
```

Step 2: Call the tempGP function using the training data. There are two ways to call tempGP, with training time indices or without training time indices. As explained in Step #1, when training time indices are not assigned, tempGP automatically assigns the time indices starting from 1.

```
## Two ways to call the tempGP function
# 1. Using user defined trainT
tempGPObject = tempGP(trainX, trainY, trainT)

# 2. Generating time indices internally as the sequence of integers starting from 1.
tempGPObject = tempGP(trainX, trainY)
```

Step 3: Use the predict method to get predictions from the learned model. Again, one can either use just environmental input variables alone to predict the response, or also use the time indices of the test data points. We first show the case for using just the environmental input variables and provide details of using time indices in Section 3.1.

```
testdata = data[5001:10000,] # defining test data as the next 5000 data points after
train indices

## Predict only the function f(x) and ignore temporal component g(t)
testX = as.matrix(testdata[,xCol,drop = F])
testY = as.numeric(testdata[,yCol])
predF = predict(tempGPObject, testX)
rmseF = sqrt(mean((testY - predF)^2)) #rmse
cat('RMSE using just f(x):',round(rmseF,3),'\n')
```

How to update the training data in the tempGP model when new data is available?

Using the time indices for test data points improves the prediction accuracy of the tempGP model if the time indices of the test points are close to that of the training points, as the temporal dependence in response vanishes after a short period of time. Thus, we have provided another function *updateData* to keep updating the training data as the new data becomes available.

```
## Predict both f(x) and g(t) using a rolling window data update with the help of
updateData() method for tempGP objects.

predY = rep(0,nrow(testdata)) #vector to store the rolling predictions
```

Step 1: Do an *i*-step ahead prediction given the input variables. In this example, we use the actual input variable values; if the input variable values are not available, replace with their forecast.

```
#starting a loop for doing the rolling predctions
for (i in 1:nrow(testdata)){

 testX_i = as.matrix(testdata[i,xCol,drop = F]) #input variables for time point i;
replace with forecast when actual data not available.

 testT_i = testdata[i,tCol] #time index for i

 predY[i] = predict(tempGPObject, testX_i, testT_i) #predict both f(x) and g(t)
```

Step 2: Once the data for time point ‘ *i* ’ is available, update the tempGP model using the updataData function.

```
 #After time point i, the data for time point i would be available. Update the data
and residuals in the tempGP object.

 tempGPObject = updateData(tempGPObject, newX = testX_i, newY = testY[i], newT =
testT_i)
}

rmseY = sqrt(mean((testY - predY)^2)) #rmse

cat('RMSE using rolling update and f(x) + g(t):',round(rmseY,3),'\n')
```

5 How to use AMK to fit a multi-dimensional power curve?

Step 1: The data set mentioned in previous question will be used to demonstrate the usability of AMK. The package is loaded and data is imported as shown.

```r
#setting the work directory which contains data set
setwd('F:/')

#loading the package
library(DSWE)

#import the data set
data = read.csv('F:/Inland Wind Farm Dataset1(WT1).csv')

```

Step 2: Prepare the arguments to fit a power curve as shown.

```r
#loading the package
library(DSWE)

#import the data set
data = read.csv('Inland Wind Farm Dataset1(WT1).csv')

# Arguments preparation
trainX = data[, c(2, 3, 4)]
trainY = data[, 7]
testX = data[30:50, c(2, 3, 4)]
cirCov = 2

#model fitting
model = AMK(trainX = trainX, trainY = trainY, testX = testX, cirCov = cirCov)


#Result value retrieval
print(model)

```

19:1 (Top Level) R Script

Console Terminal Jobs

F:/

```
> #Result value retrieval
> print(model)
 [1] 43.75839 47.05013 42.76889 39.88256 34.08364 31.54263 37.97694 41.69231
 [9] 38.80486 46.18037 44.88440 49.20261 56.27825 50.38758 48.07383 53.07551
[17] 55.41698 61.68535 33.73371 38.47867 35.89820
> 
```

Result display in the console

Function used

6 How to use the spline model to fit a multi-dimensional power curve?

Step 1: The data set mentioned in previous question will be used to demonstrate the usability of SplinePCFit. The package is loaded and data is imported as shown.

```r
#setting the work directory which contains data set
setwd('F:/')

#loading the package
library(DSWE)

#import the data set
data = read.csv('F:/Inland Wind Farm Dataset1(WT1).csv')

```

Step 2: Prepare the arguments to fit a power curve as shown.

```r
# loading library
library(DSWE)

# import the data set
data = read.csv('Inland Wind Farm Dataset1(WT1).csv')

# Arguments preparation
data = data
xCol = c(2, 3, 4)
yCol = 7
testX = data[30:50, ]

# model fitting
model = SplinePCFit(data = data, xCol = xCol, yCol = yCol, testX = testX)

# Result retrieval
print(model)
```

18:1 (Top Level)

Console Terminal Jobs

c:/Files/

```
> # Result retrieval
> print(model)
 [1] 37.59174 41.61286 36.88821 33.39156 24.65419 22.96355 27.54371 31.79511 30.85075 39.72490
[11] 37.75112 43.87695 50.80703 44.55413 42.34244 47.66733 53.02681 62.49079 32.56616 38.26896
[21] 34.95442
>
```

Result display in the console

Function used

7 How to use the Bayesian tree model to fit a multi-dimensional power curve?

Due to some issues related to CRAN package BayesTree, that has not been resolved by its maintainer, the BayesTreePCFit function in DSWE, which depends on BayesTree, has to be discontinued for now.

Step 1: The data set mentioned in previous question will be used to demonstrate the usability of BayesTreePCFit. The package is loaded and data is imported as shown.

```r
#setting the work directory which contains data set
setwd('F:/')

#loading the package
library(DSWE)

#import the data set
data = read.csv('F:/Inland Wind Farm Dataset1(WT1).csv')

```

Step 2: Prepare the arguments to fit a power curve as shown.

```r
# loading library
library(DSWE)

# import the data set
data = read.csv('Inland Wind Farm Dataset1(WT1).csv')

# Arguments preparation
trainX = data[, c(2, 3, 4)]
trainY = data[, 7]
testX = data[30:50, c(2, 3, 4)]
ntree = 50

# model fitting
model = BayesTreePCFit(trainX = trainX, trainY = trainY, testX = testX, nTree = ntree)

# Result retrieval
print(model)

```

18:1 (Top Level) R Script

Console Terminal Jobs

c:/Files/

```
> # Result retrieval
> print(model)
 [1] 38.44786 40.42227 36.09804 33.53336 22.90946 23.26647 27.63178 32.23961 28.92553 38.44786
[11] 38.44786 42.14432 53.71039 45.09118 42.14432 48.26174 60.49887 67.63969 29.05692 40.07448
[21] 34.79524
> 
```

Result display in the console

Function used

8 How to use the XGBoost model to fit a multi-dimensional power curve?

Step 1: The data set mentioned in the previous question will be used to demonstrate the usability of XgbPCFit, i.e., the XGBoost-based power curve model. The package is loaded and data is imported as shown

```
#setting the work directory which contains data set
setwd('F:/')

#loading the package
library(DSWE)

#import the data set
data = read.csv('F:/Inland Wind Farm Dataset1(WT1).csv')

```

Step 2: Prepare the arguments to fit a power curve as shown.

```
# loading library
library(DSWE)

# import the dataset
data=read.csv('Inland Wind Farm Dataset1(WT1).csv')

# arguments preparing
trainX = as.matrix(data[ , c(2, 3, 4)])
trainY = as.matrix(data[ , 7])
testX = as.matrix(data[30:50, c(2, 3, 4)])

# model fitting
model = XgbPCFit(trainX = trainX, trainY = trainY, testX = testX)

# result retrieval
print(model)

```

```
18:3   (Top Level)
Console   Background Jobs
R 4.1.3 · C:/Users/19797/OneDrive/Desktop/Dataset5/
> print(model)
 [1] 29.37282 31.71445 27.46764 26.04504 20.40950
 [6] 18.84400 20.85572 24.76675 22.52099 31.71445
[11] 29.37282 34.80626 39.61592 34.75877 31.71445
[16] 38.83776 41.55432 50.92761 23.20947 26.49713
[21] 23.20947
```

Result display in the console

Function used

9 How to use the support vector machine to fit a multi-dimensional power curve?

Step 1: The data set mentioned in previous question will be used to demonstrate the usability of SvmPCFit. The package is loaded and data is imported as shown.

```r
#setting the work directory which contains data set
setwd('F:/')

#loading the package
library(DSWE)

#import the data set
data = read.csv('F:/Inland Wind Farm Dataset1(WT1).csv')

```

Step 2: Prepare the arguments to fit a power curve as shown.

```r
# loading library
library(DSWE)

# import the data set
data = read.csv('Inland Wind Farm Dataset1(WT1).csv')

# arguments preparing
trainX = data[, c(2, 3, 4)]
trainY = data[, 7]
testX = data[30:50, c(2, 3, 4)]

# model fitting
model = SvmPCFit(trainX = trainX, trainY = trainY, testX = testX)

# result retrieval
print(model)

```

```
18:12   (Top Level) ÷                                                R Scri
Console   Terminal ×   Jobs ×
C:/Files/
> # result retrieval
> print(model)
      30       31       32       33       34       35       36       37       38
39.68693 43.32704 38.39222 35.12603 26.78749 23.98868 30.69213 35.08113 32.13036
      39       40       41       42       43       44       45       46       47
41.47650 39.31682 45.59180 53.14763 46.67026 44.13884 51.48125 55.21317 63.31669
      48       49       50
27.62785 34.00319 30.17626
```

Result display in the console

Function used

10 How do the power curve functions compare with each other?

We tested the power curve functions implemented both in R and Python packages using Dataset#5 (Inland and Offshore Wind Farm Dataset1) of the *Data Science for Wind Energy* book. Please follow the instruction of Step 1 to Question 3 “How to use KNN to fit a multi-dimensional power curve?” to download the datasets. There are six turbine datasets in Dataset#5, four from onshore turbines and two from offshore turbines.

The following power curve functions are tested: tempGP, AMK, KNN, DNN, SSANONA, SVM, XGBoost, and the binning method (the IEC standard procedure).

The following table presents the root mean square error (RMSE) based on a randomized five-fold cross-validation. The power is normalized, with each turbine’s rated power as 100%. So the values reported below are relative to the rated power. For instance, 0.1 means 10% of the rated power.

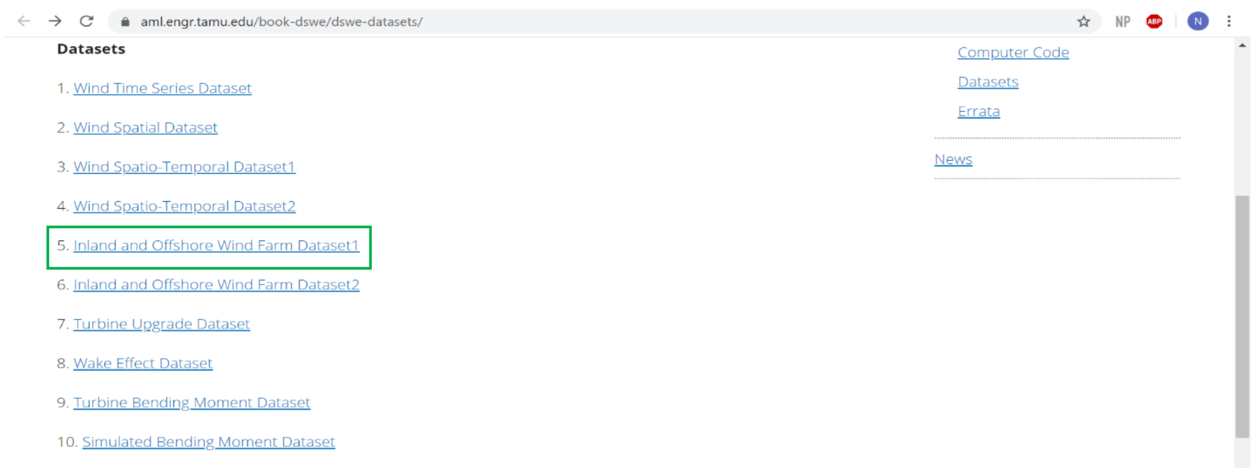
In the table, the results of AMK, KNN, SSANOVA, and the binning method are taken directly from Table 5.7 of the *Data Science for Wind Energy* book, the results of tempGP, TempGP(fast computation), XGBoost and SVM are obtained using the respective R package functions, and the results of DNN are obtained using its Python package function.

	tempGP	tempGP (fast computation)	AMK	KNN	DNN	SSANOVA	SVM	XGBoost	BIN
WT1	0.064	0.065	0.074	0.077	0.082	0.087	0.100	0.115	0.131
WT2	0.070	0.070	0.080	0.083	0.088	0.091	0.099	0.115	0.116
WT3	0.056	0.055	0.065	0.067	0.074	0.077	0.091	0.109	0.122
WT4	0.079	0.079	0.100	0.104	0.111	0.112	0.125	0.131	0.152
WT5	0.066	0.068	0.079	0.081	0.089	0.095	0.094	0.116	0.097
WT6	0.066	0.069	0.080	0.083	0.094	0.104	0.104	0.119	0.109
Average	0.067	0.068	0.080	0.082	0.090	0.094	0.102	0.118	0.121
Relative to the original tempGP	-	1.01	1.19	1.23	1.34	1.41	1.53	1.75	1.81

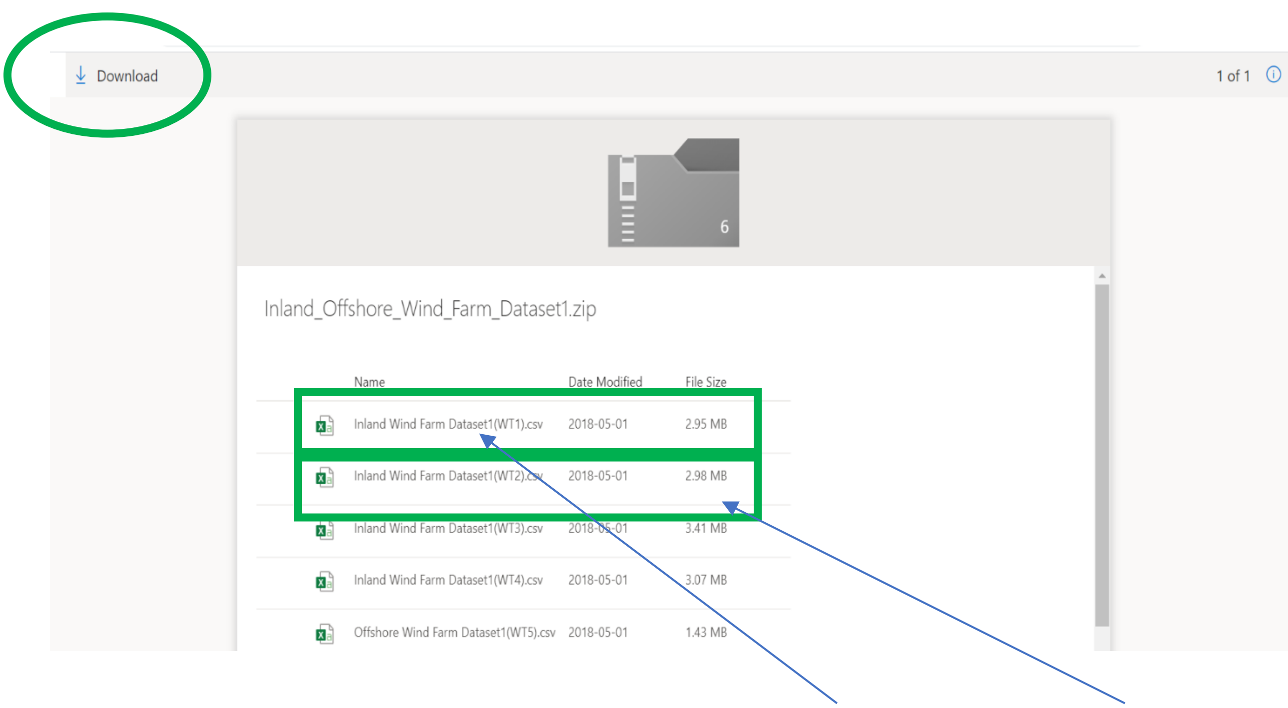
11 How to select the subsets of data, before and after a decision point, so that they can be deemed probabilistically comparable?

Step1: Download the sample data set.

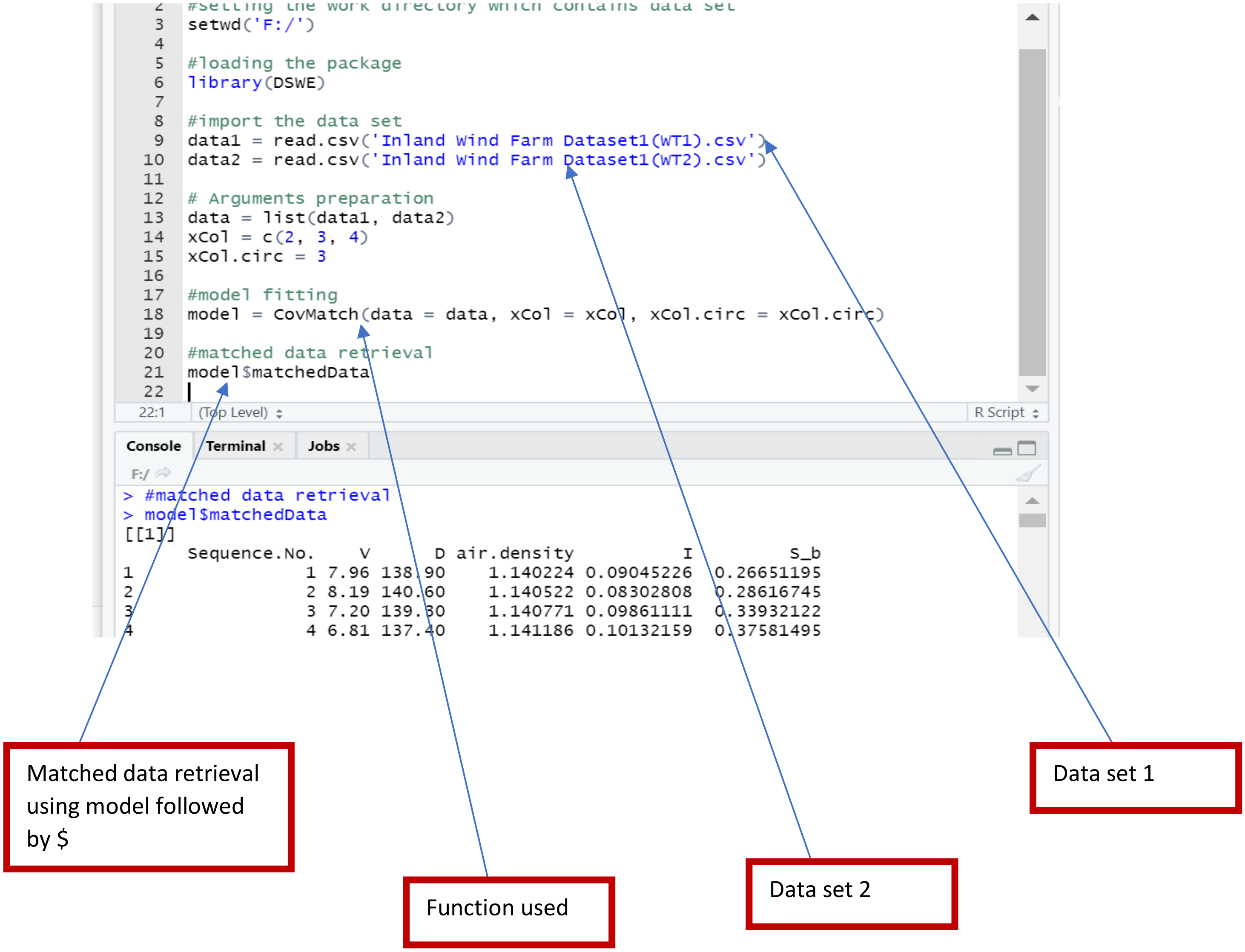
Visit site using following link - *https://sites.google.com/view/yuding/book-dswe/dswe-datasets*
The page looks like as shown below and select Dataset #5.



Download the data set as shown below in green boxes. After downloading, save the file in working directory.



Step 3: The package is loaded and data is imported. Also, arguments are prepared and matching function is employed as shown



12 How to compare performance of two turbine or two data set in different time period?

Step 1: The data set mentioned in previous question will be used to demonstrate the performance quantification. The package is loaded and data is imported as shown.

```
#setting the work directory which contains data set
setwd('F:/')

#loading the package
library(DSWE)

#import the data set
data1 = read.csv('Inland Wind Farm Dataset1(WT1).csv')
data2 = read.csv('Inland Wind Farm Dataset1(WT2).csv')
```

Step 2: Prepare the arguments to use performance comparison function as shown.

```
#loading the package
library(DSWE)

#import the data set
data1 = read.csv('Inland Wind Farm Dataset1(WT1).csv')
data2 = read.csv('Inland Wind Farm Dataset1(WT2).csv')

# Arguments preparation
data = list(data1, data2)
xCol = c(2, 3, 4)
xCol.circ = 3
yCol = 7
testCol = c(2, 4)

#model fitting
model = ComparePCurve(data = data, xCol = xCol, xCol.circ = xCol.circ, yCol = yCol, testCol = tes

#Result retrieval
model$weightedDiff
model$scaledDiff
```

Result retrieval

Function used

Argument preparation

Data set

How to use a different probability distribution than that computed from the data to compute the weighted difference between the power curves?

Step 1: Construct a desired ***testset*** and a probability distribution over that ***testset*** as shown below as an example:

```
## Construct a custom testset and custom weights.
ws_min = 5 #minimum value of wind speed for constructing the testset.
ws_max = 15 #maximum value of wind speed for constructing the testset.
ws_test = seq(ws_min, ws_max, length.out = 50) #generate 50 grid points for wind speed.
rho_min = 1.1 #minimum value of air density for constructing the testset.
rho_max = 1.3 #maximum value of air density for constructing the testset.
rho_test = seq(rho_min, rho_max, length.out = 50) #generate 50 grid points for air density.

#Combine ws_test and rho_test to create a 50 by 50 mesh grid.
testset = expand.grid(ws_test, rho_test)

#For example, We use a Weibull distribution for windspeed with shape = 2.25 and scale = 6.5.
#and uniform distribution for air density. Please change as desired.
#Multiplying the weights by 1 to denote a uniform distribution for air density.
weights = dweibull(testset[,1], shape = 2.25, scale = 6.5)*1
weights = weights/sum(weights) #normalizing weights to ensure that they sum to 1.
```

Step 2: Run ***ComparePCurve()*** function as described earlier with the ***testset*** generated in Step 1 as one of the inputs:

```
#Loading the package and reading the data
library(DSWE)
data1 = read.csv('Inland Wind Farm Dataset1(WT1).csv')
data2 = read.csv('Inland Wind Farm Dataset1(WT2).csv')
datalist = list(data1,data2)

#Calling ComparePCurve() function
output = ComparePCurve(data = datalist, xCol = c(2,3,4), xCol.circ = 3,
                       yCol = 7, testCol = c(2,4), testSet = testset)
```

Step 3: Use the output from ***ComparePCurve()*** function to compute the weighted difference and statistically significant weighted difference with the weights computed in Step 1 as follows:

```
# Computing weighted difference with mu1 as the base for percentage calculation
muDiff = output$muDiff
base = output$mu1
weightedDiff = ComputeWeightedDifference(muDiff = muDiff, weights = weights, base = base)

# Computing statistically significant weighted difference
confBand = output$band
weightedStatDiff = ComputeWeightedDifference(muDiff = muDiff, weights = weights, base = base,
                                             statDiff = TRUE, confBand = confBand)
```

13 A case study of estimating the effect associated with turbine upgrades.

This case study applies the functions in the DSWE package to the Turbine Upgrade Dataset, associated with the book, *Data Science for Wind Energy*, and available from the website below. The case study is explained in Section 4.1 of the preprint https://arxiv.org/pdf/2005.08652.pdf. Additional information about the dataset and turbine upgrades can be found in Section 1.2.3 and Chapter 7 of *Data Science for Wind Energy*. The dataset includes two cases of upgrades–one is the pitch angle adjustment and the second is the vortex generator installation. The steps below explain how the top rows of Table 3 of the preprint https://arxiv.org/pdf/2005.08652.pdf are obtained as well as how the VG effect is estimated.

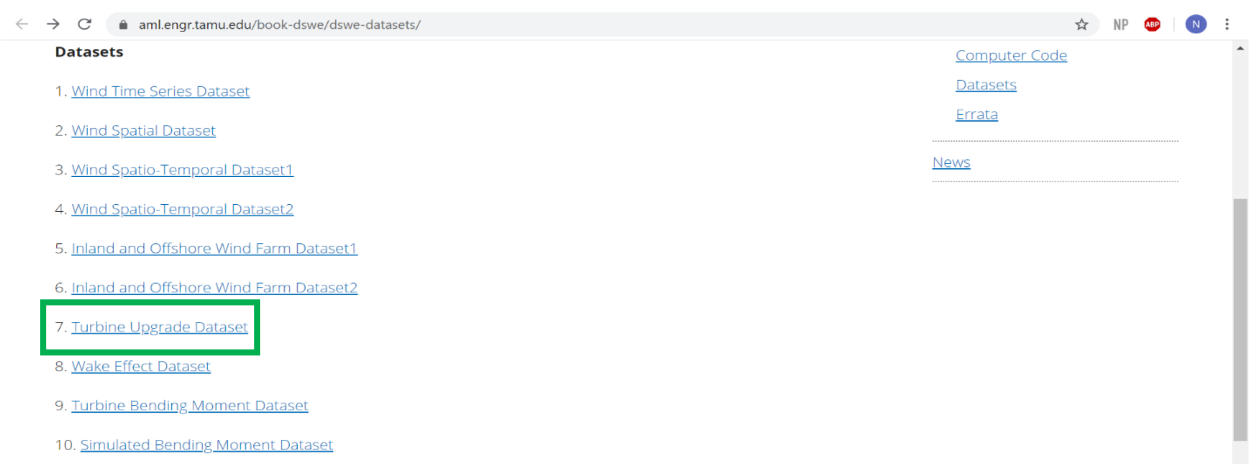
The above preprint is now published in the journal of *Renewable Energy*. The paper's full citation is

> Ding, Kumar, Prakash, Kio, Liu, Liu, and Li, 2021, "A case study of space-time performance comparison of wind turbines on a wind farm," *Renewable Energy*, Vol. 171, pp. 735-746.

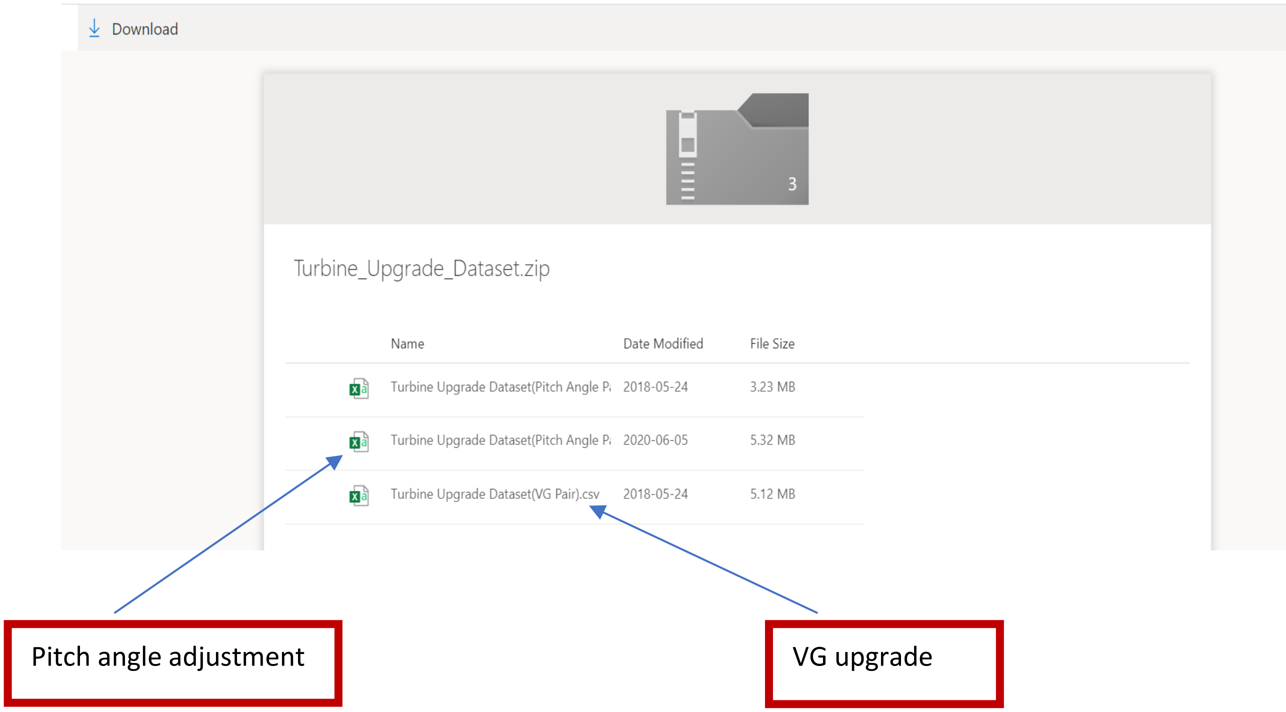
One can go to *https://sites.google.com/view/yuding/publications* (and then go to J77) to get the reproducibility report and R code for reproducing the majority of the results in this paper.

Step1: Download the sample data set as shown.

Visit site using the following link - *https://sites.google.com/view/yuding/book-dswe/dswe-datasets*. The page looks like as shown below and select Dataset #7.



Download the sample data set as shown below. After downloading, save the file in working directory.



Step 2: Set the path containing data set to a current working directory. Further load the package and import the data set as shown

For pitch angle pair:

```
# setting the working directory which contains data set
setwd('F:/')

# loading library
library(DSWE)

# import the data set
data = read.csv('Turbine Upgrade Dataset(Pitch Angle Pair, Table7.3).csv')

```

For VG upgrade:

```r
# setting the working directory which contains data set
setwd('F:/')

# loading library
library(DSWE)

# import the data set
data = read.csv('Turbine Upgrade Dataset(VG Pair).csv')
```

Step 3: Use the performance comparison function on pitch angle adjustment and VG upgrade as shown below. In case of pitch angle adjustment, user just needs to import the appropriate data set and change the value of 'input' variable as shown below in the script

For Pitch Angle adjustment:

```r
# import the data set
data = read.csv('Turbine Upgrade Dataset(Pitch Angle Pair, Table7.3).csv')

#creating a result data frame
input = 2   # enter a value between 2 - 9 (integer)
result_table = data.frame(matrix(NA, nrow = 3, ncol = 2))
colnames(result_table) = c('r', paste(input, '%', sep = ''))
result_table[, 1] = c('r\'', 'Our estimate', 'Our estimate/rprime')

# Code for sensitivity analysis
data1 = data[data$upgrade.status==0, ]
data2 = data[data$upgrade.status==1, ]
r = c(2, 3, 4, 5, 6, 7, 8, 9)
rprime  = c(1.25, 1.87, 2.49, 3.11, 3.74, 4.36, 4.98, 5.60)
xCol = c(3, 4, 5, 6, 7)
xCol.circ = 4
dataList = list(data1, data2)

# control model
control = ComparePCurve(data = dataList, xCol = xCol, xCol.circ = xCol.circ, testCol = c(3, 4), thrs = 0.2, yCol = 18)
```

Data set

Control model

User defined input (2-9)

```r
# test model
yCol = c(10, 11, 12, 13, 14, 15, 16, 17)
rprime = rprime[r == input]
yCol = yCol[r == input]
r = r[r == input]
test = ComparePCurve(data = dataList, xCol = xCol, xCol.circ = xCol.circ, testCol = c(3, 4), thrs = 0.2, yCol = yCol)
result_table[2, 2] = test$weightedDiff - control$weightedDiff
result_table[3, 2] = round(result_table[2, 2] / rprime, 2)
result_table[1, 2] = paste(rprime, '%', sep = '')
result_table[2, 2] = paste(result_table[2, 2], '%', sep = '')

# display result
print(result_table)
```

35:1 (Top Level)

Console Terminal Jobs

```
# display result
print(result_table)
                   r    2%
                  r' 1.25%
        Our estimate 1.12%
Our estimate/rprime   0.9
```

Result display

Test model

For VG upgrade:

```r
# import the data set
data = read.csv('Turbine Upgrade Dataset(VG Pair).csv')

# argument preparation
data1 = data[data$upgrade.status == 0, ]
data2 = data[data$upgrade.status == 1, ]
dataList = list(data1, data2)
xCol = c(4, 5, 6, 7)
xCol.circ = 5

# control model
control = ComparePCurve(data = dataList, xCol = xCol, xCol.circ = xCol.circ, testCol = c(4, 5), thrs = 0.2, yCol = 12)

# test model
test = ComparePCurve(data = dataList, xCol = xCol, xCol.circ = xCol.circ, testCol = c(4, 5), thrs = 0.2, yCol = 11)

# VG effect
VG_Effect = paste(test$weightedDiff - control$weightedDiff, '%', sep = '')

# result retrieval
print(VG_Effect)
```

22:1 (Top Level) R Script

Console Terminal Jobs

c:/Files/

```
> # result retrieval
> print(VG_Effect)
[1] "1.32%"
>
```

Result display in the console

Target for control and test model

14 How to use the Energy Decomposition approach and the deltaEnergy function?

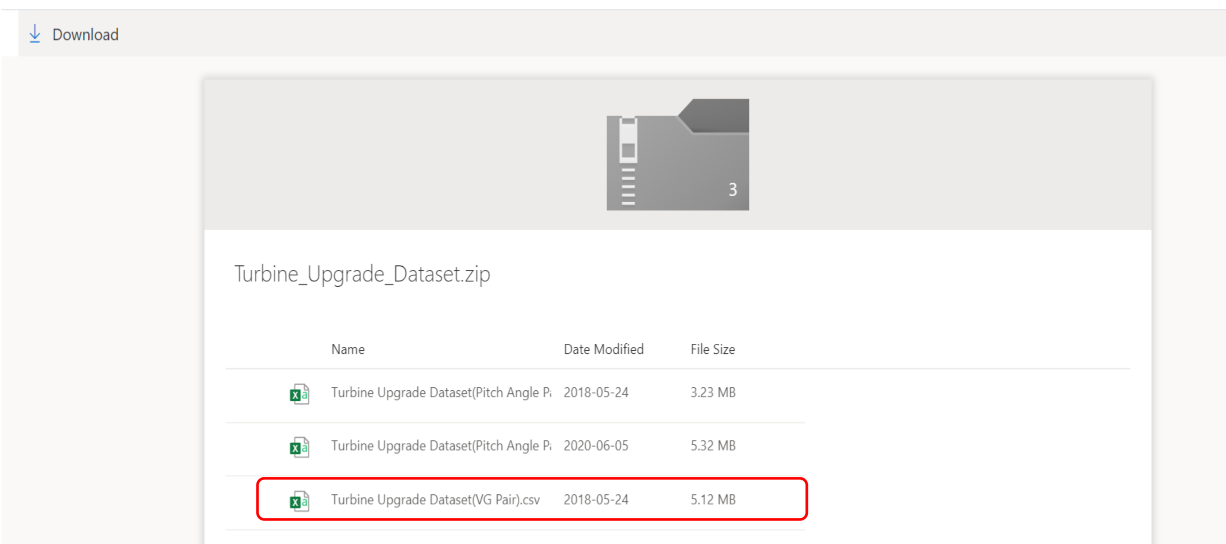
The following example applies the functions to the Turbine Upgrade Dataset, which bears similarity to the case study in:

> Latiffianti, E, Ding, Y, Sheng, S, Williams, L, Morshedizadeh, M, Rodgers, M. Analysis of leading edge protection application on wind turbine performance through energy and power decomposition approaches. Wind Energy. 2022; 25(7): 1203- 1221. doi:10.1002/we.2722. Available online: https://onlinelibrary.wiley.com/doi/full/10.1002/we.2722

This approach can be used for two purposes: 1) estimating the effect associated with turbine upgrades, 2) comparing the performance of two turbines on the same period of operations. Both will be demonstrated in the following.

Step1: The data set mentioned in the previous question will be used to demonstrate the performance quantification. Perform **Step 1** from the previous question.

Download the sample data set as shown below. After downloading, save the file in the working directory.



Step 2: Set the path containing data set to a current working directory. Further load the package and import the data set as shown.

```
# Setting the working directory that contains data sets
setwd('~F:/')

#loading library
library(DSWE)

#import the data sets
data1 = read.csv('Turbine Upgrade Dataset(VG Pair).csv')
```

Step 3: To apply performance comparison using energy decomposition, the data pair needs synchronization and imputation. There are three functions available related to the energy decomposition: `syncSize`, `imptPower` and `deltaEnergy`. The `deltaEnergy` automatically includes `syncSize` and it also provides the option to carry `imptPower`. With the available functions, there are two ways to obtain the energy decomposition: A) using `deltaEnergy` to perform synchronization, imputation, and decomposition in one go, or B) using `imptPower` and then apply `deltaEnergy` on the output. The following are the script for each Approach A and Approach B separately.

Energy decomposition using approach A.

```
# Step 3 Approach A
Before = data1[data1$upgrade.status==0,]
After = data1[data1$upgrade.status==1,]

set.seed(50)
upgrade.effect = deltaEnergy(data=list(Before,After), powercol = 11, xcol = c(4:7), timecol=2,
                             sync.method = "random", vcol=4, vrange = c(5,12,25))
set.seed(70)
control.effect = deltaEnergy(data=list(Before,After), powercol = 12, xcol = c(4:7), timecol=2,
                             sync.method = "random", vcol=4,vrange = c(5,12,25))
control.effect$deltaE.turb
# VG effects (energy difference)
deltaE.turb = round(upgrade.effect$deltaE.turb - control.effect$deltaE.turb,2)
cat(paste('delta E from VG effects is ',deltaE.turb,'%',sep=''),'\n')
```

```
> # VG effects (energy difference)
> deltaE.turb = round(upgrade.effect$deltaE.turb - control.effect$deltaE.turb,2)
> cat(paste('delta E from VG effects is ',deltaE.turb,'%',sep=''),'\n')
delta E from VG effects is 1.46%
```

Result display

Note that the result from energy decomposition and power difference (`ComparePCurve`) are not expected to be the same, but intuitively it should be close enough when data pairs are adequate in size. In this particular example, each pair has 5,000 data points after synchronization. Ideally, 1-year worth of data should be used. Please refer to Latiffianti et al, 2022 as mentioned above.

Energy decomposition using approach B.

```
# Using approach B ------------------------------

#Synchronization
set.seed(10)
sync.test = syncSize(data=list(Before,After), powercol=11, xcol=c(4:7), method='random',timecol = 2)
set.seed(10)
sync.control = syncSize(data=list(Before,After), powercol=12, xcol=c(4:7), method='random',timecol = 2)

# Imputation
imput.test = imptPower(data=sync.test, powercol=2, vcol = 3, vrange = c(5,12,25), rated.power = 1)
imput.control = imptPower(data=sync.control, powercol=2, vcol = 3, vrange = c(5,12,25), rated.power = 1)

# Energy decomposition
set.seed(10)
deltaE.test = deltaEnergy(data = imput.test, powercol=2, timecol=1, vcol = 3, xcol = c(3:6),imput=FALSE)
deltaE.control = deltaEnergy(data = imput.control, powercol=2, timecol=1, vcol = 3, xcol = c(3:6),imput=FALSE)

#Print result
deltaE = round(deltaE.test$deltaE.turb - deltaE.control$deltaE.turb,2)
cat(paste(deltaE,'%',sep=''),'\n')
```

```
> #Print result
> deltaE = round(deltaE.test$deltaE.turb - deltaE.control$deltaE.turb, 2)
> cat(paste(deltaE,'%',sep=''))
1.2%
```

Result display

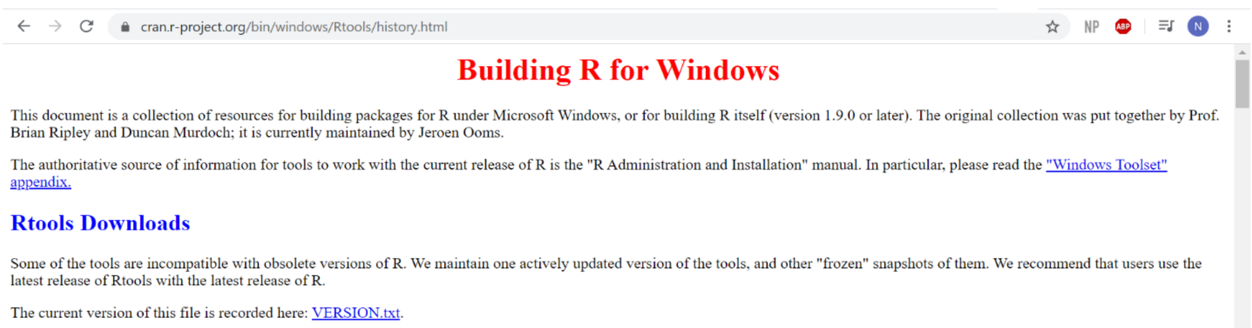
Results from Approach A and Approach B may be slightly different due to randomization in the power curve modeling (`tempGP`) and sometimes synchronization (`syncSize`) when the synchronization method chosen is ‘`random`’.

15 (Optional) Installation using source code

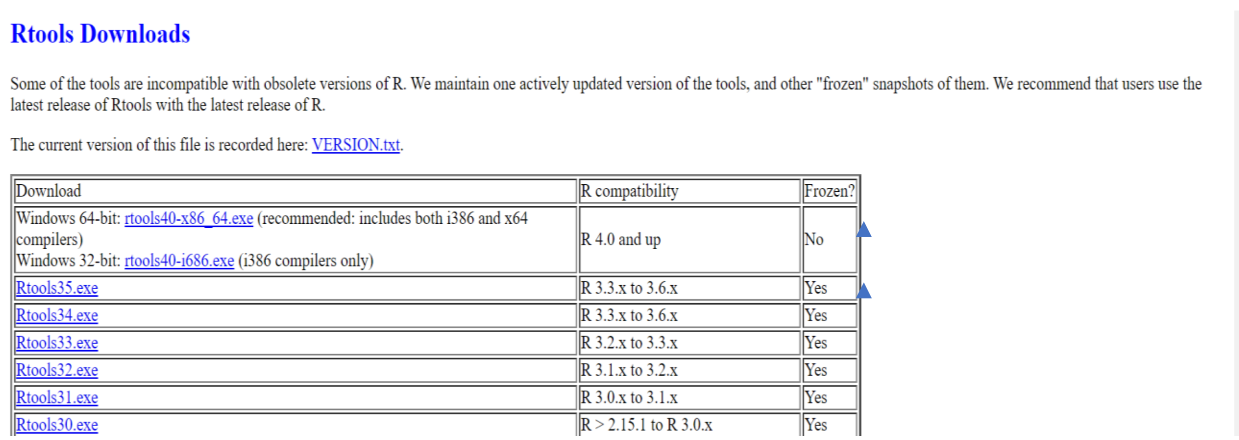
The package contains some C++ code for fast computation, and thus requires compiling C++ code if one wishes to install the package using the source code. Following are the necessary steps in order to get the required compilation tools and install the package from source:

Step 1: Install C++ tool chains, which is the Rtools for Windows and the GFortran for Mac. The guidelines to install are:

For Windows:
Visit site using the following link: *https://cran.r-project.org/bin/windows/Rtools/history.html*. The page directed looks like the below.



Choose the compatible Rtools version from the table below and follow the installation process. If one uses R 4.0 or up, please select 'Rtools40-x86_64'. If one uses R 3.5.x-3.6.x, please choose 'Rtools35.exe'.



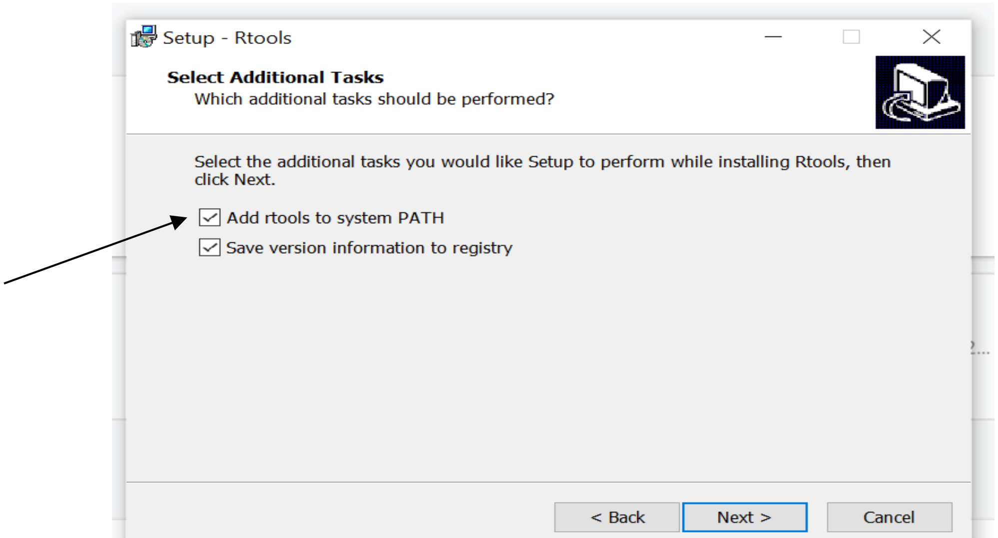
Rtools Downloads

Some of the tools are incompatible with obsolete versions of R. We maintain one actively updated version of the tools, and other "frozen" snapshots of them. We recommend that users use the latest release of Rtools with the latest release of R.

The current version of this file is recorded here: VERSION.txt.

Download	R compatibility	Frozen?
Windows 64-bit: rtools40-x86_64.exe (recommended: includes both i386 and x64 compilers) Windows 32-bit: rtools40-i686.exe (i386 compilers only)	R 4.0 and up	No
Rtools35.exe	R 3.3.x to 3.6.x	Yes
Rtools34.exe	R 3.3.x to 3.6.x	Yes
Rtools33.exe	R 3.2.x to 3.3.x	Yes
Rtools32.exe	R 3.1.x to 3.2.x	Yes
Rtools31.exe	R 3.0.x to 3.1.x	Yes
Rtools30.exe	R > 2.15.1 to R 3.0.x	Yes

If the following prompt message box appears with the option of **'Add rtools to system path,' please make sure that option is checked.** If the option is not shown, then just proceed, and Step 2 provides the information for manually adding rtools to system path.



For MacOS:

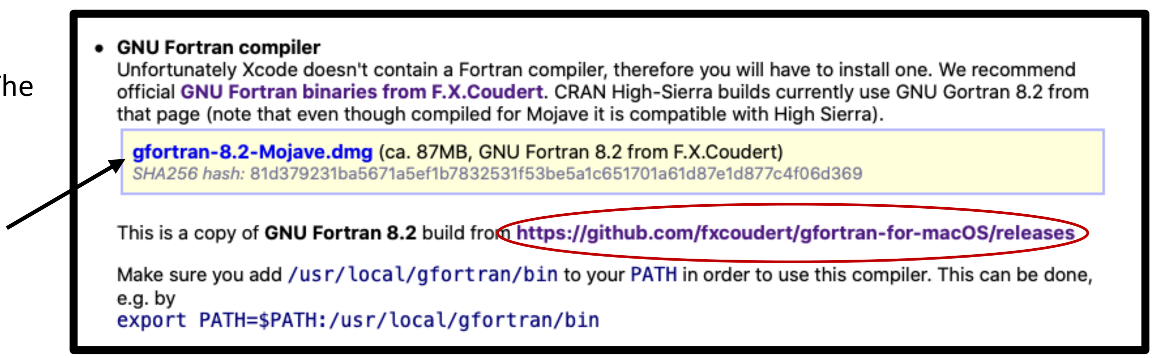
You would need C++ and Fortran compilers to build the package. Apple's official C++ compilers can be downloaded by installing the command line developer tools using the following steps:

- Open the Terminal app
- Type the command: xcode-select –install

An installation window will open up. Click install and the installation would be begin.

Apple's command line tools do not have a Fortran compiler. It can be downloaded from the official website of R using the link: *https://mac.r-project.org/tools/*

The



latest version is for MacOS Mojave but works for MacOS Catalina too. If an older version of MacOS is in use, use the link in the circle to download for older versions.

Step 2: While installing Rtools, if the prompt message box did not appear with a message **'Add rtools to system path'**, please follow the steps below to manually add Rtools to system path.

- First locate Rtools bin location on your computer. The default location for Rtools35 is "C:\\Rtools\\bin" and the default location for Rtools40 is "C:\\Rtools40\\usr\\bin". Please double check and make sure using the File Explorer on your computer.
- Next, use the following command in R, **`Rtool_bin_location = "C:\\Rtools\\bin"`** if using Rtools35, or, **`Rtool_bin_location = "C:\\Rtools40\\usr\\bin"`** if using Rtools40. If the Rtools bin is not located in directory, please enter the right location identified in the above step.
    - Last, execute the following command in R, **`Sys.setenv(PATH = paste(Rtool_bin_location, Sys.getenv("PATH"), sep=";"))`** , to set up Rtools in path temporarily.

Step 3: Use the standard **`install.packages()`** command as: **`install.packages("DSWE")`** to install the package and use the install from source option when asked.

Note: Some message boxes may pop up asking for user input.

- One pop-up message box asks "***Do you want to install from sources the package which needs compilation***?" Upon prompted, please click on "Yes".
- Another pop-up message box asks for updates. Upon prompted, type "1" and press enter, as it is safe to overwrite the installed dependencies with the recent ones. The layout of prompt may differ, depending on R versions in use. Always select the option to update the package.