Appendix-A

Code in R language for estimation of wind energy potential

```
>library(MASS)
>hist(ANNUAL.100$V1,breaks=94)
>plot(density(ANNUAL.100$V1))
>rug(ANNUAL.100$V1)
>mean(ANNUAL.100$V1)
>min(ANNUAL.100$V1)
>max(ANNUAL.100$V1)
>library(fitdistrplus)
>fitgmme<-fitdist(ANNUAL.100$V1,"gamma",method="mme")
>summary(fitgmme)
>fitw<-fitdist(ANNUAL.100$V1,"weibull")
>fitg<-fitdist(ANNUAL.100$V1,"gamma")
>fitln<-fitdist(ANNUAL.100$V1,"lognormal")
>summary(fitg)
>summary(fitln)
>cdfcomp(list(fitw,fitg,fitln),legendtext=c("weibull","gamma","lognormal"))
>denscomp(list(fitw,fitg,fitln),legendtext=c("weibull","gamma","lognormal"))
>qqcomp(list(fitw,fitg,fitln),legendtext=c("weibull","gamma","lognormal"))
>ppcomp(list(fitw,fitg,fitln),legendtext=c("weibull","gamma","lognormal"))
>gofstat(list(fitw,fitg,fitln),fitnames=c("weibull","gamma","lognormal"))
```

Appendix-B

Code in R language for estimation of load demand distribution

```
>library(MASS)

>hist(mydata$V1,breaks=94)

>plot(density(mydata$V1))

>rug(mydata$V1)

>mean(mydata$V1)

>min(mydata$V1)

>max(mydata$V1)

>library(fitdistrplus)

>fitln<-fitdist(mydata$V1,"lnorm")

>fitg<-fitdist(mydata$V1,"gamma")

>fitw<-fitdist(mydata$V1,"weibull")

>summary(fitln)

>summary(fitg)

>summary(fitw)

>cdfcomp(list(fitln,fitg,fitw),legendtext=c("lognormal","gamma","weibull"))

>denscomp(list(fitln,fitg,fitw),legendtext=c("lognormal","gamma","weibull"))

>qqcomp(list(fitln,fitg,fitw),legendtext=c("lognormal","gamma","weibull"))

>ppcomp(list(fitln,fitg,fitw),legendtext=c("lognormal","gamma","weibull"))

>gofstat(list(fitln,fitg,fitw),fitnames=c("lognormal","gamma","weibull"))
```

Appendix-C

Code in Java language for optimal planning of renewable distributed generation

```
package optimization.project;

public class OptimizationProject {

System.out.println("Objective" + calculation.optimizationFunction());

System.out.println("Surplus All Years" + calculation.cForAllYears());

System.out.println("Consumption " + calculation.cForAllYearsConsumptionPower());

System.out.println("Investment Cost" + calculation.investmentCost());

System.out.println("Operation & Maintenance Cost" +
calculation.cDistributorGenerationForAllYears());

System.out.println("Power Purchase" + calculation.cPowerPurchaseForAllYears());

System.out.println("RDGSUM=" + calculation.RDGUnitsSum);

        }

}

package optimization.project;

import java.util.HashMap;

public class OptimizationCalculation{
//*********************************************************************
***//

public String toString() {

      return "Objective="+ optimizedvalue + ", Surplus All Years=" + cForAllYears() +
",Consumption=" + cForAllYearsConsumptionPower() + ",Investment Cost=" +
investmentCost() + ",Operation & Maintenance Cost=" +
cRenewableDistributorGenerationForAllYears() + ",Power Purchase=" +
cPowerPurchaseForAllYears() + ",RDGSUM=" + RDGUnitsSum;

              }

//*********************************************************************
***//

double ldrandomnumber=Utilities.random();

double wsrandomnumber=Utilities.random();

public int lifeinyears,numberofunit,costofeachturbine;
public double ratedwindturbinepowercapacity,sellingprice/*in
rupees/kwh*/,purchasecost/*in rupees/kwh*/,discount/*in
percent*/,maintenancecost/*rupee/kwh*/;

public double
lognormallocationparameter,lognormalscaleparameter,weibullscaleparameter,weibullsh
apeparameter,cutinwindvelocity,ratedwindvelocity,cutoutwindvelocity;

public double optimizedvalue;

//*******************************************************************
***//

double surpluspowersum=0;

public int totalhours,totalyears;

public HashMap<String,InputParameters> lst=new HashMap<>();

public int[] RDGUnits;

public int RDGUnitsSum;

public OptimizationCalculation(OptimizationCalculation input,double random1,double
random2,int[] RDG)

   {

this(input.totalhours,input.totalyears,input.lifeinyears,input.ratedwindturbinepowercapa
city,input.costofeachturbine,input.sellingprice,input.purchasecost,input.discount,input.m
aintenancecost,input.lognormallocationparameter,input.lognormalscaleparameter,input.
weibullscaleparameter,input.weibullshapeparameter,input.cutinwindvelocity,input.rated
windvelocity,input.cutoutwindvelocity);

        this.wsrandomnumber=random1;

        this.ldrandomnumber=random2;

        this.RDGUnits=RDG;

optimizedvalue=optimizationFunction();

   }

        public OptimizationCalculation(int hours,int years, int lifeinyears,  double
ratedwindturbinepowercapacity, int costofeachturbine, double sellingprice, double
purchasecost, double discount, double maintenancecost, double
lognormallocationparameter, double lognormalscaleparameter, double
weibullscaleparameter, double weibullshapeparameter, double cutinwindvelocity,
double ratedwindvelocity, double cutoutwindvelocity)

   {

RDGUnitsPer=new int[numberofbus];

for(int i=0;i<=RDGUnits.length-1;i++)
```

```
        {
RDGUnits[i]=Utilities.getNoOfRDGUnits();
RDGUnitsSum+=RDGUnits[i];
        }
//****************************************************************//
        this.lifeinyears = lifeinyears;
        this.ratedwindturbinepowercapacity = ratedwindturbinepowercapacity;
        this.costofeachturbine = costofeachturbine;
        this.sellingprice = sellingprice;
        this.purchasecost = purchasecost;
        this.discount = discount;
        this.maintenancecost = maintenancecost;
        this.lognormallocationparameter = lognormallocationparameter;
        this.lognormalscaleparameter = lognormalscaleparameter;
        this.weibullscaleparameter = weibullscaleparameter;
        this.weibullshapeparameter = weibullshapeparameter;
        this.cutinwindvelocity = cutinwindvelocity;
        this.ratedwindvelocity = ratedwindvelocity;
        this.cutoutwindvelocity = cutoutwindvelocity;
//****************************************************************//
totalhours=hours;
totalyears=years;
for(int year =1;year<=years;year++)
        {
for(int hour=1;hour<=hours;hour++)
        {
           String index=year + "," + hour;
lst.put(index,new InputParameters(year, hour));
```

```
        }
        }
optimizedvalue=optimizationFunction();
    }
   //***************Total investment
cost************************************//
public double investmentCost()
    {
      // System.out.println("discount" + discount);
      // System.out.println("total years=" + totalyears);
double x1=Math.pow(1+ discount, totalyears)-1;
       // System.out.println("x1=" + x1);
double x2=discount*Math.pow(1+discount,totalyears);
     //  System.out.println("x2=" + x2);
double x3=x1/x2;
      // System.out.println("x3=" + x3);
double x4=discount*Math.pow (1+discount,lifeinyears);
      // System.out.println("x4=" + x4);
double x5=Math.pow(1+discount,lifeinyears)-1;
       //System.out.println("x5=" + x5);
double x6=x4/x5;
      // System.out.println("x6=" + x6);
double x7=x3*x6*ratedwindturbinepowercapacity*costofeachturbine;
       //System.out.println("x7=" + x7);
return x7*DGUnitsSum;
    }
   //**************Objective function
minimization*****************************//
public double optimizationFunction()
    {
```

```
return -cForAllYears()-
cForAllYearsConsumptionPower()+cDistributorGenerationForAllYears()+cPowerPurc
haseForAllYears()+investmentCost();

   }

   //****************Total operating and maintenance
cost************************//

public double cRenewableDistributorGenerationForAllYears()

   {

double sum=0;

for(int year=1;year<=totalyears;year++)

       {

sum=sum + cRenewableDistributorGenerationPerYear(year);

       }

return sum;

   }

public double cRenewableDistributorGenerationPerYear(int year)

   {

double surplussum=doRenewableDistributorGenerationHourSum(year);

return surplussum*(maintenancecost)/Math.pow(1+ discount,year);

   }

public double doRenewableDistributorGenerationHourSum(int year)

   {

double sum=0;

for(int hour=1;hour<=totalhours;hour++)

       {

           String index=year + "," + hour;

           InputParameters ip=lst.get(index);

double dg=ip.renewabledistributorGeneration() * Math.pow( RDGUnitsSum,year);

sum=sum + dg;

       }

return sum;
```

```
    }

    //******Total revenue due to power saving by renewable distributed
generation********//

public double cForAllYearsConsumptionPower()

    {

double sum=0;

for(int year=1;year<=totalyears;year++)

        {

sum=sum + consumptionPerYear(year);

        }

return sum;

    }

public double consumptionPerYear(int year)

    {

double surplussum=doLoadDemandHourSum(year);

return surplussum*(purchasecost)/Math.pow(1+ discount,year);

    }

public double doLoadDemandHourSum(int year)

    {

double sum=0;

for(int hour=1;hour<=totalhours;hour++)

        {

            String index=year + "," + hour;

            InputParameters ip=lst.get(index);

double ld=ip.loadDemand();

double dg=ip.renewabledistributorGeneration();

          //  System.out.printf("ld is %f,  dg= %f,  dg*RDGUnitsSum %f
\n",ld,dg,dg*RDGUnitsSum);

if(ld<=dg*RDGUnitsSum)

sum=sum + ld;
else
sum=sum + dg*RDGUnitsSum;
        }
return sum;
    }
    //******************Revenue from total power supplied to
grid******************//
public double cForAllYears()
    {
double sum=0;
for(int year=1;year<=totalyears;year++)
        {
sum=sum + cPerYear(year);
        }
return sum;
    }
public double cPerYear(int year)
    {
double surplussum=doSurplusHourSum(year);
return surplussum*(sellingprice)/Math.pow(1+discount,year);
    }
public double doSurplusHourSum(int year)
    {
double sum=0;
for(int hour=1;hour<=totalhours;hour++)
        {
            String index=year + "," + hour;
            InputParameters ip=lst.get(index);
sum=sum + ip.surplusPowerToGrid();
        }
```

```
return sum;
   }
   //**************Total power purchase
cost***********************************//
public double cPowerPurchaseForAllYears()
   {
double sum=0;
for(int year=1;year<=totalyears;year++)
      {
sum=sum + cPowerPurchasePerYear(year);
      }
return sum;
   }
public double cPowerPurchasePerYear(int year)
   {
double surplussum=doPowerPurchaseHourSum(year);
return surplussum*(purchasecost)/Math.pow(1+ discount,year);
   }
public double doPowerPurchaseHourSum(int year)
   {
double sum=0;
for(int hour=1;hour<=totalhours;hour++)
      {
         String index=year + "," + hour;
         InputParameters ip=lst.get(index);
sum=sum + ip.powerPurchasedFromGrid();
      }
return sum;
   }
   //************************Sub
Class***********************************//
class InputParameters {
double ldrandomnumber=Utilities.random();
double wsrandomnumber=Utilities.random();
public int planperiodinyears,operatinghours;
public InputParameters( int year,int hour) {
      this.planperiodinyears=year;
      this.operatinghours=hour;
   }
public double powerPurchasedFromGrid()
   {
       //Power purchased from grid when demand is greater than RDG generation
double ld=loadDemand();
doublerdg=renewabledistributorGeneration();
if(ld>rdg)
return ld-rdg;
else
return 0;
   }
public double surplusPowerToGrid()
   {
       //Surplus power supplied to grid
double ld=loadDemand();
doublerdg=renewabledistributorGeneration();
if(rdg>ld)
returnrdg-ld;
else
return 0;
   }
```

```
public double renewabledistributorGeneration()
   {
       //Hourly power generation based on generated wind speed values for a single rdg
unit
double ws=windSpeed();
if(ws>=0 && ws< cutinwindvelocity || ws>=cutoutwindvelocity)
return 0;
if(ws>=cutinwindvelocity && ws<=ratedwindvelocity)
return ratedwindturbinepowercapacity*(ws-cutinwindvelocity)/(ratedwindvelocity-
cutinwindvelocity);
return ratedwindturbinepowercapacity;
   }
public double loadDemand()
   {
       //Random generation of load demand values
while(true)
       {
ldrandomnumber=Utilities.random();
double rand=ldrandomnumber;
double x1=2*lognormallocationparameter -Math.sqrt(2)*lognormalscaleparameter;
double x2=x1*x1 + 4*(lognormallocationparameter*lognormallocationparameter +
2*Math.sqrt(Math.PI)* lognormalscaleparameter*lognormalscaleparameter*rand);
double retvalue=Math.exp(0.5*(x1+x2));
if(retvalue<=11)
continue;
return retvalue;
       }
   }
public double windSpeed()
          //Random generation of wind speed values
   {
```

```
double rand=wsrandomnumber;

double exp= Math.exp(Math.log((weibullscaleparameter*rand)/weibullshapeparameter)
+ (weibullshapeparameter-1)* Math.log(weibullscaleparameter)) ;

        // System.out.println(exp);

double temp=(weibullshapeparameter/weibullscaleparameter) * exp;

temp=Math.pow(temp,1/(weibullshapeparameter-2));

return temp;

    }

}

    //*****************************************************************//
```

Appendix-D

VIKOR method

Opricovic (1998), Opricovic and Tzeng (2002) developed VIKOR, the Serbian name: VlseKriterijumska Optimizacija I KompromisnoResenje, means multi-criteria optimization and compromise solution (Chu et al., 2007). The VIKOR method was developed for multi-criteria optimization of complex systems (Opricovic and Tzeng, 2004). This method focuses on ranking and selecting from a set of alternatives, and determines compromise solutions for a problem with conflicting criteria, which can help the decision makers to reach a final decision. Here, the compromise solution is a feasible solution which is the closest to the ideal, and a compromise means an agreement established by mutual concessions (Opricovic and Tzeng, 2007). It introduces the multi-criteria ranking index based on the particular measure of ‘‘closeness” to the ‘‘ideal” solution (Opricovic, 1998).

Ranking by VIKOR may be performed with different values of criteria weights, analyzing the impact of criteria weights on proposed compromise solution. The VIKOR method determines the weight stability intervals, using the methodology presented in Opricovic (1998). The compromise solution obtained with initial weights (w_i, i = 1; . . . ; n), will be replaced if the value of a weight is not within the stability interval. The analysis of weight stability intervals for a single criterion is performed for all criterion functions, with the same (given) initial values of weights. In this way, the preference stability of an obtained compromise solution may be analyzed using the VIKOR program.

Matching MCDM methods with classes of problems would address the correct applications, and for this reason the VIKOR characteristics are matched with a class of problems as follows (Opricovic and Tzeng, 2007):

• Compromising is acceptable for conflict resolution.

• The decision maker (DM) is willing to approve solution that is the closest to the ideal.

• There exist a linear relationship between each criterion function and a decision maker's utility.

• The criteria are conflicting and noncommensurable (different units).

• The alternatives are evaluated according to all established criteria (performance matrix).

• The DM's preference is expressed by weights, given or simulated.

• The VIKOR method can be started without interactive participation of DM, but the DM is in charge of approving the final solution and his/her preference must be included.

• The proposed compromise solution (one or more) has an advantage rate.

• A stability analysis determines the weight stability intervals.

The main steps of multi-criteria decision making are the following (Opricovic and Tzeng, 2004):

(a) Establishing system evaluation criteria that relate system capabilities to goals;

(b) Developing alternative systems for attaining the goals (generating alternatives);

(c) Evaluating alternatives in terms of criteria (the values of the criterion functions);

(d) Applying a normative multi-criteria analysis method;

(e) Accepting one alternative as ‘‘optimal’’ (preferred);

(g) If the final solution is not accepted, gather new information and go into the next iteration of multi-criteria optimization.

When the decision maker is unable to take a decision or doesn't know to express their preferences at the beginning stage of the system design, the VIKOR method would be an effective tool for the multi-criteria decision-making process. For the value of a maximum group utility of the ‘‘majority” (min S, given by Eq. (2)), and a minimum individual regret of the ‘‘opponent” (min R, given by Eq. (3)), obtained compromise solution would be accepted by the decision makers. Based on the involvement of the decision-makers' preferences by weights of criteria, the compromise solutions would be the base for negotiation. The result of the VIKOR ranking depends on the ideal solution Q with values of *v*, which will be only for a given set of alternatives. Any changes to a given set of alternatives will lead to the result of modified VIKOR ranking for the new set of alternatives. The fixed ideal solution would be defined by the decision maker based on the best f_i and the worst f_i values, but it could be avoided.

Here each alternative would be evaluated with each criterion function and, the compromise ranking would be performed with the comparison of the measure of closeness to ideal solution F*. Compromise solution F^C will be a feasible solution that will be the closest to the ideal solution and will have a compromise established by mutual concessions (Polatidis et al., 2006). With multi-criteria measure for the compromise ranking of alternatives is developed from the Lp-metric by using an aggregating function from the compromise programming method (Yu, 1973; and Zeleny, (1982) :

$$L_{pj} = \left[\sum_{i=1}^{n}\left\{ {w_i\left(f_i^* - f_{ij}\right)}\Big/{(f_i^* - f_i^-)}\right\}^{p}\right]^{1/p} \tag{1}$$

$1 \leq p \leq \infty$, j=1, 2,....., J

where $L_{1,\,j}$ denoted as S_j in Eq. (2) and $L_{\infty,j}$ denoted as R_j in Eq. (3), are used to formulate the ranking measure.

For the VIKOR method, the number of j alternatives is denoted as $a_1, a_2, \ldots, a_j$. For any alternative a_j the rating of the i_{th} facet is denoted by f_{ij}, and this is the value of the i_{th} criterion for the alternative a_j; where j=1,2,....,m and i=1,2,.....,n. The compromise ranking algorithm of the VIKOR method is divided into the following four steps which are given below (Opricovic and Tzeng, 2004):

Step I: For all the criterion functions, find out the best f_i^* and the worst f_i^- values, i = 1,2,...,n. If the i_{th} function represents a benefit then $f_i^* = \max_j f_{ij}$ and $f_i^- = \min_j f_{ij}$, whereas if the i_{th} function represents a cost $f_i^* = \min_j f_{ij}$ and $f_i^- = \max_j f_{ij}$.

Step II: Compute the values of S_j and R_j, j = 1,2,...,m from the relations of

$$S_j = \sum_{i=1}^{n} {w_i\left(f_i^* - f_{ij}\right)}\Big/{(f_i^* - f_i^-)} \tag{2}$$

$$R_j = \max_i \left[{w_i\left(f_i^* - f_{ij}\right)}\Big/{(f_i^* - f_i^-)}\right] \tag{3}$$

Where w_i denotes the weights of criteria, which expresses the decision maker's preference for the relative importance of the criteria.

Step III: compute the values of Q_j, from the given relation

$$Q_j = {}^{v(S_j - S^*)}\Big/_{(S^- - S^*)} + {}^{(1 - v)(R_j - R^*)}\Big/_{(R^- - R^*)} \tag{4}$$

Where $S^* = \min_j S_j$; $S^- = \max_j S_j$; $R^* = \min_j R_j$; $R^- = \max_j R_j$ and as a weight v has been introduced for the strategy of maximum group utility, while (1 - *v*) is for the weight of the individual regret. The solution will be obtained by $\min_j S_j$ with a maximum group utility based on "majority" rule, where the solution will be obtained by $\min_j R_j$ with a minimum individual regret of the "opponent". In general, the value of the *v* is taken as 0.5, but we can take any value of *v* in the range of 0 to 1.

Step IV: Now rank the alternatives with the sorting of the results of S, R, and Q in decreasing order. From this we will have three ranking lists for S, R, and Q. Suppose we have a compromise solution of the alternative A^1 best ranked by the minimum value of the measure Q, then it should satisfy the given conditions. Propose as a compromise solution the alternative A^1, which is the best ranked by the measure Q (minimum), if t he following two conditions are satisfied:

a. First one is the acceptable advantage. $Q(A^2) - Q(A^1) \geq DQ$, where DQ=1/(J-1) and A^2 is the alternative with the second position on the ranking list by Q;
b. The second one is the acceptable stability in decision-making. The alternative A^1 should also be the best ranked by S or/and R. This compromise solution should be stable for a decision-making process, that could be the strategy of maximum group utility (when $v > 0.5$ is needed), or by consensus ($v \approx 0.5$), or with veto ($v < 0.5$).

If one of the above conditions is not satisfied, then we will have to propose a set of compromise solutions, which will consist of:

c. Alternative A^1 and A^2 when the condition b is not satisfied, or

d. Alternatives A^1, A^2,..., A^M when the condition a is not satisfied and, A^M is determined by the relation $Q(A^M) - Q(A^1) < DQ$ for maximum value n means the positions of these alternatives are "in closeness".

In brief, we can say that VIKOR method works on ranking and selection of the alternatives from the given one in the existence of conflicting criteria. It gives a compromise solution that will be accepted by the decision makers because of its maximum group utility for the ''majority'', and of the minimum individual regret for the ''opponent''. By the use of linear normalization, this method representing the closeness to the ideal solution based on aggregating function.

List of Papers Published/ Accepted/Communicated/Presented

A. Journals

1. Kumar, M., and Samuel, C. (2017). Selection of best Renewable Energy Source by using VIKOR Method. *Technology and Economics of Smart Grids and Sustainable Energy, 2(1): 8.* (Springer, Published)
2. Kumar, M., and Samuel, C. (2017). Stochastic Demand Side Management in Smart Grid. *International Journal of Networking and Virtual Organisations*. (Inderscience, Accepted)
3. Kumar, M., and Samuel, C. (2017). Wind energy potential estimation with prediction of wind speed distribution. *International Journal of Intelligent Systems Technologies and Applications*. (Inderscience, Accepted)
4. Kumar, M., and Samuel, C. (2017). Green Practices with Renewable Distributed Generation Technology in India. *Energy & Environment*. (SAGE, Accepted)
5. Kumar, M., and Samuel, C. (2017). Future of Renewable Distributed Generation in India.*TERI Information Digest on Energy and Environment.* (The Energy and Resources Institute, Accepted)
6. Kumar, M., Samuel, C. and Jaiswal, A. (2015). An overview of distributed generation in power sector. *International Journal of Science, Technology & Management, 4(1): 1407-1423.*
7. Kumar, M., and Samuel, C. (2017). Optimal Planning of Stochastic Renewable Distributed Generation System. *Energy Sources, Part B: Economics, Planning, and Policy.* (Taylor & Francis, Communicated)
8. Kumar, M., and Samuel, C. (2017). Increasing Demand of Renewable Distributed Generation Technologies in Modern Power System. *Renewable & Sustainable Energy Reviews*. (Elsevier, Communicated)
9. Kumar, M., and Samuel, C. (2017). The Economics of Smart Grid Integrated Renewable Distributed Generation. *Technology and Economics of Smart Grids and Sustainable Energy.* (Springer, Communicated)

B. Conferences

1. Kumar, M., and Samuel, C. (2016, September). Statistical analysis of load demand distribution at Banaras Hindu University, India. In Advances in Computing, Communications and Informatics-2016 (ICACCI-2016), Jaipur, India (pp. 2318-2323).IEEE.

2. Kumar, M., and Samuel, C. (2016, May). Modern Green Building Concept in India: A New Face of Ancient Methodologies with Novel Features. National Conference on Cultural Heritage and Management-2016 (NCCHM-2016), Indian Institute of Technology (Banaras Hindu University), Varanasi.

3. Kumar, M., and Samuel, C. (2016, February). Increasing Demand of Renewable Distributed Generation Technology in Power Sector. National Conference on