Appendix I

// Rough membership neuron program//

```
#include <fstream>
#include <cstdlib>
#include <iostream>
//#include <stdio.h>
#include <string>

using namespace std;

#define numberOfFiles 508
#define numberOfAttributes 11
#define numberOfFaults 12
#define MaximumFaultsCombination 4
#define maxNumberofAtrributesInComb 5
#define NumberofPossibleElements 300 // now maximum is 256 which is in fault of
minor AC error
#define numberOfTestFiles 169

main()
{
  struct roughset
  {
    int setSampleInfo[NumberofPossibleElements][maxNumberofAtrributesInComb];
    int setSampleInfoCounter;
    int setYes[NumberofPossibleElements][maxNumberofAtrributesInComb];
    int setYesCounter;
    int setYesOrNo[NumberofPossibleElements][maxNumberofAtrributesInComb];
    int setYesOrNoCounter;
    int setYesOrNoElem[NumberofPossibleElements][maxNumberofAtrributesInComb];//
store each type of element in set YesOrNo
    int setYesOrNoEachElemNum[NumberofPossibleElements]; // store the number of
each type of element in set YesOrNo
    int setYesOrNoElemNum;// store the number of type for
    int flag;
    int YesOrNoElem[NumberofPossibleElements][maxNumberofAtrributesInComb];//
store each type of element in set YesOrNo for each information
    int YesOrNoEachElemNum[NumberofPossibleElements]; // store the number of each
type of element in set YesOrNo for each information
    int YesOrNoElemNum;// store the number of type for each information
    float YesProbability[NumberofPossibleElements];
  };

  static roughset FS[numberOfFaults+1][numberOfAttributes];
```

```
int i,j,k,l,m,n,s,grobleFlag, subFlag;
int c[maxNumberofAtrributesInComb], FaultType[MaximumFaultsCombination];
FILE *fpin, *fpout1, *fpout2, *fpout3;
FILE *fpout5[numberOfFaults];
FILE *fpout7[numberOfFaults];
int c_sum = 0;

char  FaultTypeIndicator0[] = "Minor AC disturb";
char  FaultTypeIndicator1[] = "AC Disturb";
char  FaultTypeIndicator2[] = "Valve Current Closed/blocked/deblock";
char  FaultTypeIndicator3[] = "Line Fault";
char  FaultTypeIndicator4[] = "Commutation Failure";
char  FaultTypeIndicator5[] = "Pole Voltages/Current Closed/blocked/deblock";
char  FaultTypeIndicator6[] = "Current Arc Back";
char  FaultTypeIndicator7[] = "Parallel operation";
char  FaultTypeIndicator8[] = "Pole Current Oscillation";
char  FaultTypeIndicator9[] = "Normal affected by another pole";
char  FaultTypeIndicator10[] ="Asym protection";
char  FaultTypeIndicator11[] ="Disturbance on DC voltage";
int testFileIndex[numberOfTestFiles];
int attributeCombIndex[] = {1,1,5,3,1,1,1,1,1,1,1};

for (i=0; i<numberOfTestFiles; i++){
  testFileIndex[i] = i * (int)(numberOfFiles / numberOfTestFiles);
  printf("%d ", testFileIndex[i]);
}

for (i=0; i<numberOfFaults+1; i++){
  for (j=0; j<numberOfAttributes; j++){
    FS[i][j].setSampleInfoCounter = 0;
    FS[i][j].setYesCounter = 0;
    FS[i][j].setYesOrNoCounter = 0;
    FS[i][j].YesOrNoElemNum = 0;

    for ( k = 0; k<NumberofPossibleElements; k++)
    {
      FS[i][j].YesOrNoEachElemNum[k] = 0;
      for (l = 0; l<maxNumberofAtrributesInComb; l++)
      {
        FS[i][j].setSampleInfo[k][l] = 0;
        FS[i][j].setYes[k][l] = 0;
        FS[i][j].setYesOrNo[k][l] = 0;
        FS[i][j].setYesOrNoElem[k][l] = 0;
        FS[i][j].YesOrNoElem[k][l] = 0;
      }
    }
```

```
  }
}

fpin = fopen("train_PT_VTCombe_CAB_PCO_LFrevised_PI_sorted_input.txt","r");
fpout1 = fopen("FaultSample_11 atrributes_train.txt","wr");
fpout2 = fopen("FaultYesOrNo_11 atrributes_train.txt","wr");
fpout3 = fopen("FaultYes_11 atrributes_train.txt","wr");

fpout5[0] = fopen("FTrain0_11 atrributes.txt","wr");
fpout5[1] = fopen("FTrain1_11 atrributes.txt","wr");
fpout5[2] = fopen("FTrain2_11 atrributes.txt","wr");
fpout5[3] = fopen("FTrain3_11 atrributes.txt","wr");
fpout5[4] = fopen("FTrain4_11 atrributes.txt","wr");
fpout5[5] = fopen("FTrain5_11 atrributes.txt","wr");
fpout5[6] = fopen("FTrain6_11 atrributes.txt","wr");
fpout5[7] = fopen("FTrain7_11 atrributes.txt","wr");
fpout5[8] = fopen("FTrain8_11 atrributes.txt","wr");
fpout5[9] = fopen("FTrain9_11 atrributes.txt","wr");
fpout5[10] = fopen("FTrain10_11 atrributes.txt","wr");
fpout5[11] = fopen("FTrain11_11 atrributes.txt","wr");


fpout7[0] = fopen("FTrainT0_11 atrributes.txt","wr");
fpout7[1] = fopen("FTrainT1_11 atrributes.txt","wr");
fpout7[2] = fopen("FTrainT2_11 atrributes.txt","wr");
fpout7[3] = fopen("FTrainT3_11 atrributes.txt","wr");
fpout7[4] = fopen("FTrainT4_11 atrributes.txt","wr");
fpout7[5] = fopen("FTrainT5_11 atrributes.txt","wr");
fpout7[6] = fopen("FTrainT6_11 atrributes.txt","wr");
fpout7[7] = fopen("FTrainT7_11 atrributes.txt","wr");
fpout7[8] = fopen("FTrainT8_11 atrributes.txt","wr");
fpout7[9] = fopen("FTrainT9_11 atrributes.txt","wr");
fpout7[10] = fopen("FTrainT10_11 atrributes.txt","wr");
fpout7[11] = fopen("FTrainT11_11 atrributes.txt","wr");


for (i=0; i<numberOfFiles; i++){
  for (n = 0; n<MaximumFaultsCombination; n++){
     fscanf(fpin, "%d", &FaultType[n]);
     FaultType[n] = FaultType[n]-1;
  }
  for (j=0; j<numberOfAttributes; j++)
  {
    for (s=0; s<attributeCombIndex[j]; s++)  fscanf(fpin, "%d", &c[s]);
    for (n = 0; n<MaximumFaultsCombination; n++) {
      if (FaultType[n]>=0){
```

```
        for (s=0; s<attributeCombIndex[j]; s++)
FS[FaultType[n]][j].setSampleInfo[FS[FaultType[n]][j].setSampleInfoCounter][s] = c[s];
        FS[FaultType[n]][j].setSampleInfoCounter ++;
       }
      }
     }
    }

/*  for (i = 0; i < numberOfFaults; i++){
    for (j=0; j<numberOfAttributes; j++){
      printf("%d ", FS[i][j].setSampleInfoCounter);
      for (k =0; k<FS[i][j].setSampleInfoCounter; k++){
        for (s = 0; s<attributeCombIndex[j]; s++){
          printf("%d ", FS[i][j].setSampleInfo[k][s]);
        }
        printf("\t");
      }
    }
    printf("\n");
  }  */

  fclose(fpin);
  fpin = fopen("train_PT_VTCombe_CAB_PCO_LFrevised_PI_sorted_input.txt","r");

  fprintf(fpout1,"\t%s\t","Pole Voltage Sharp Drop");
  fprintf(fpout1,"%s\t","AC Phase Voltage Disturb Severity");
  fprintf(fpout1,"%s\t","P1 , p2, volts and currents trend");
  fprintf(fpout1,"%s\t","current closed  Voltage not closed");
  fprintf(fpout1,"%s\t","Valve Current Trend_vg1_2_3");
  fprintf(fpout1,"%s\t","3 valve current trends last digits are all 1");
  fprintf(fpout1,"%s\t","3 valves current trend are same");
  fprintf(fpout1,"%s\t"," 6 Pulse Spikes_max_vg1_2_3");
  fprintf(fpout1,"%s\t", " valve current spikes_comb_vg1_2_3");
  fprintf(fpout1,"%s\t", " Spikes Not following the closure of Valve Current");
  fprintf(fpout1,"%s\n", " short time close or minor disturb in valve
current_comb_vg1_2_3");

  fprintf(fpout2,"%s\n","YesOrNo SET");
  fprintf(fpout2,"\t%s\t","Pole Voltage Sharp Drop");
  fprintf(fpout2,"%s\t","AC Phase Voltage Disturb Severity");
  fprintf(fpout2,"%s\t","P1 , p2, volts and currents trend");
  fprintf(fpout2,"%s\t","current closed  Voltage not closed");
  fprintf(fpout2,"%s\t","Valve Current Trend_vg1_2_3");
  fprintf(fpout2,"%s\t","3 valve current trends last digits are all 1");
  fprintf(fpout2,"%s\t","3 valves current trend are same");
  fprintf(fpout2,"%s\t"," 6 Pulse Spikes_max_vg1_2_3");
```

```
  fprintf(fpout2,"%s\t", " valve current spikes_comb_vg1_2_3");
  fprintf(fpout2,"%s\t", " Spikes Not following the closure of Valve Current");
  fprintf(fpout2,"%s\n", " short time close or minor disturb in valve
current_comb_vg1_2_3");

  fprintf(fpout3,"%s\n","Yes SET");
  fprintf(fpout3,"\t%s\t","Pole Voltage Sharp Drop");
  fprintf(fpout3,"%s\t","AC Phase Voltage Disturb Severity");
  fprintf(fpout3,"%s\t","P1 , p2, volts and currents trend");
  fprintf(fpout3,"%s\t","current closed Voltage not closed");
  fprintf(fpout3,"%s\t","Valve Current Trend_vg1_2_3");
  fprintf(fpout3,"%s\t","3 valve current trends last digits are all 1");
  fprintf(fpout3,"%s\t","3 valves current trend are same");
  fprintf(fpout3,"%s\t"," 6 Pulse Spikes_max_vg1_2_3");
  fprintf(fpout3,"%s\t", " valve current spikes_comb_vg1_2_3");
  fprintf(fpout3,"%s\t", " Spikes Not following the closure of Valve Current");
  fprintf(fpout3,"%s\n", " short time close or minor disturb in valve
current_comb_vg1_2_3");

  for (i=0; i<numberOfFaults; i++){
    if (i==0) fprintf(fpout1,"%s\t",FaultTypeIndicator0);
    if (i==1) fprintf(fpout1,"%s\t",FaultTypeIndicator1);
    if (i==2) fprintf(fpout1,"%s\t",FaultTypeIndicator2);
    if (i==3) fprintf(fpout1,"%s\t",FaultTypeIndicator3);
    if (i==4) fprintf(fpout1,"%s\t",FaultTypeIndicator4);
    if (i==5) fprintf(fpout1,"%s\t",FaultTypeIndicator5);
    if (i==6) fprintf(fpout1,"%s\t",FaultTypeIndicator6);
    if (i==7) fprintf(fpout1,"%s\t",FaultTypeIndicator7);
    if (i==8) fprintf(fpout1,"%s\t",FaultTypeIndicator8);
    if (i==9) fprintf(fpout1,"%s\t",FaultTypeIndicator9);
    if (i==10) fprintf(fpout1,"%s\t",FaultTypeIndicator10);
    if (i==11) fprintf(fpout1,"%s\t",FaultTypeIndicator11);

    for (j=0; j<numberOfAttributes; j++){
      for (k=0; k<FS[i][j].setSampleInfoCounter; k++){
        if (attributeCombIndex[j] > 1) {
          fprintf(fpout1, "{");
          for (s=0; s<attributeCombIndex[j]; s++) {
            fprintf(fpout1, "%d ",  FS[i][j].setSampleInfo[k][s]);
          }
          fprintf(fpout1, "} ");
        }
        else fprintf(fpout1, "%d ",  FS[i][j].setSampleInfo[k][0]);
      }
      fprintf(fpout1, "\t");
    }
```

```
    fprintf(fpout1, "\n");
  }

  for (i=0; i<numberOfFiles; i++){
    for (n = 0; n<MaximumFaultsCombination; n++){
       fscanf(fpin, "%d", &FaultType[n]);
       FaultType[n] = FaultType[n]-1;
//       printf ("%d\n", FaultType[n]);
    }

    for (j=0; j<numberOfAttributes; j++){
      for (n = 0; n<MaximumFaultsCombination; n++){
        if (FaultType[n]>=0) FS[FaultType[n]][j].flag = 0;
      }
      for (s=0; s<attributeCombIndex[j]; s++)  {
        fscanf(fpin, "%d", &c[s]);
//        if (j==2) printf ("%d ", c[s]);
      }
//      if (j==2) printf ("\n");

      if ((FaultType[0]>=0)&(FaultType[1]>=0)){            // has more than 1 faults
        for (n = 0; n<MaximumFaultsCombination; n++){
          if (FaultType[n] >=0) FS[FaultType[n]][j].flag=1;
//          printf ("%s\n", "flag=1");
        }
      }
      else{           // has only one fault which is FaultType[0]
        for (k=0; k<numberOfFaults; k++){
          if ((k != FaultType[0])&(FS[FaultType[0]][j].flag==0)){
            for (l=0; l<FS[k][j].setSampleInfoCounter; l++){
              if (FS[FaultType[0]][j].flag==0){
                c_sum = 0;
//                if (j==2) printf("\ni = %d k=%d j=%d l=%d ", i, k,j,l);
                for (s=0; s<attributeCombIndex[j]; s++)
                {
                  c_sum = c_sum + abs(FS[k][j].setSampleInfo[l][s]-c[s]);
//                  if (j==2) printf("%d ", FS[k][j].setSampleInfo[l][s]);
                }
/*                for (s=0; s<attributeCombIndex[j]; s++)
                {
                  if (j==2) printf("%d ", c[s]);
                }    */
                if (c_sum == 0) FS[FaultType[0]][j].flag=1;
              }
            }
          }
```

```
      }
    }

//    if (j==2) printf ("FS[%d][%d].flag=%d\n", FaultType[0], j, FS[FaultType[0]][j].flag);

    if (FS[FaultType[0]][j].flag==0) {
      for (s=0; s<attributeCombIndex[j]; s++)
FS[FaultType[0]][j].setYes[FS[FaultType[0]][j].setYesCounter][s] = c[s];
      FS[FaultType[0]][j].setYesCounter ++;
    }
    else{
      for (n = 0; n<MaximumFaultsCombination; n++){
        if (FaultType[n]>=0){
          if (FS[FaultType[n]][j].flag==1) {
            for (s=0; s<attributeCombIndex[j]; s++)
FS[FaultType[n]][j].setYesOrNo[FS[FaultType[n]][j].setYesOrNoCounter][s] = c[s];
            FS[FaultType[n]][j].setYesOrNoCounter ++;
          }
        }
      }
    }
  }


 /* for (i = 0; i < numberOfFaults; i++){
   for (j=0; j<numberOfAttributes; j++){
     printf("%d %d ", FS[i][j].setYesCounter, FS[i][j].setYesOrNoCounter);
   }
   printf("\n");
  }  */



  for (i=0; i<numberOfAttributes; i++){
    // initialize the FS[numberOfFaults][i]: the first element and YesOrNoElemNum
    FS[numberOfFaults][i].YesOrNoElemNum = 1;

    grobleFlag = 0;
    for (j=0; j<numberOfFaults; j++){
      if ((FS[j][i].setYesOrNoCounter!=0)&(grobleFlag==0)){
        for (s=0; s<attributeCombIndex[i]; s++) FS[numberOfFaults][i].YesOrNoElem[0][s]
= FS[j][i].setYesOrNo[0][s];
        grobleFlag = 1;
      }
    }
```

```
//   for (s=0; s<maxNumberofAtrributesInComb; s++) printf("%d",
FS[numberOfFaults][i].YesOrNoElem[0][s]);
//   printf("\n");

   for (j=0; j<numberOfFaults; j++){
     for (k=0; k<FS[j][i].setYesOrNoCounter; k++){
       grobleFlag = 0;
       for (l=0; l<FS[numberOfFaults][i].YesOrNoElemNum; l++){
         c_sum =0;
         for (s=0; s<attributeCombIndex[i]; s++) c_sum = c_sum +
abs(FS[numberOfFaults][i].YesOrNoElem[l][s] - FS[j][i].setYesOrNo[k][s]);
         if ((c_sum == 0)&(grobleFlag==0)){
           FS[numberOfFaults][i].YesOrNoEachElemNum[l] =
FS[numberOfFaults][i].YesOrNoEachElemNum[l] + 1;
           grobleFlag = 1;
         }
       }
       if (grobleFlag == 0){
         FS[numberOfFaults][i].YesOrNoElemNum =
FS[numberOfFaults][i].YesOrNoElemNum + 1;
         for (s=0; s<attributeCombIndex[i]; s++)
FS[numberOfFaults][i].YesOrNoElem[FS[numberOfFaults][i].YesOrNoElemNum-1][s]
= FS[j][i].setYesOrNo[k][s];

FS[numberOfFaults][i].YesOrNoEachElemNum[FS[numberOfFaults][i].YesOrNoElemN
um-1] = 1;
       }
     }
   }

  /*****************************************
   if (i==2){
     printf("%d\t%d\n", numberOfFaults, FS[numberOfFaults][i].YesOrNoElemNum );
     for (m=0; m<FS[numberOfFaults][i].YesOrNoElemNum; m++)
     {
       for (s=0; s<maxNumberofAtrributesInComb; s++) printf("%d ",
FS[numberOfFaults][i].YesOrNoElem[m][s]);
       printf("\t");
       printf("%d\t", FS[numberOfFaults][i].YesOrNoEachElemNum[m]);
     }
     printf("\n");
   }
   //************************************************/
  }
```

```
  for (i=0; i<numberOfFaults; i++){
    for (j=0; j<numberOfAttributes; j++){
      if (FS[i][j].setYesOrNoCounter!=0){
        for (s=0; s<attributeCombIndex[j]; s++) FS[i][j].setYesOrNoElem[0][s] =
FS[i][j].setYesOrNo[0][s];
        FS[i][j].setYesOrNoElemNum = 1;
        for (k=0; k<FS[i][j].setYesOrNoCounter; k++){
          grobleFlag=0;
          for (l=0; l<FS[i][j].setYesOrNoElemNum; l++){
            if (grobleFlag==0){
              c_sum = 0;
              for (s=0; s<attributeCombIndex[j]; s++) c_sum = c_sum +
abs(FS[i][j].setYesOrNo[k][s] - FS[i][j].setYesOrNoElem[l][s]);
              if (c_sum == 0){
                grobleFlag = 1;
                FS[i][j].setYesOrNoEachElemNum[l] = FS[i][j].setYesOrNoEachElemNum[l]
+1;
              }
            }
          }
          if (grobleFlag==0){
            FS[i][j].setYesOrNoElemNum ++;
            for (s=0; s<attributeCombIndex[j]; s++)
FS[i][j].setYesOrNoElem[FS[i][j].setYesOrNoElemNum-1][s] =
FS[i][j].setYesOrNo[k][s];
            FS[i][j].setYesOrNoEachElemNum[FS[i][j].setYesOrNoElemNum-1] = 1;
          }
        }
      }
/*      if ((i==11)&(j==2)){
        for (m = 0; m < FS[i][j].setYesOrNoElemNum; m++){
          printf("Faults = %d , Attributes = %d , setYesOrNoEachElemNum = %d\n",
i,j,FS[i][j].setYesOrNoEachElemNum[m]);
        }
      }*/
    }
  }


// calculate the probability

  for (i=0; i<numberOfFaults; i++){
    for (j=0; j<numberOfAttributes; j++){
      if (FS[i][j].setYesOrNoCounter !=0){
        for (k=0; k<FS[i][j].setYesOrNoElemNum; k++){
```

```
          grobleFlag = 0;
          for (l=0; l<FS[numberOfFaults][j].YesOrNoElemNum; l++){
            if (grobleFlag==0){
              c_sum = 0;
              for (s=0; s<attributeCombIndex[j]; s++) c_sum = c_sum +
abs(FS[i][j].setYesOrNoElem[k][s] - FS[numberOfFaults][j].YesOrNoElem[l][s]);
              if (c_sum == 0){
                grobleFlag = 1;
                FS[i][j].YesProbability[k] =
(float)FS[i][j].setYesOrNoEachElemNum[k]/(float)FS[numberOfFaults][j].YesOrNoEac
hElemNum[l];
//                if (j==2){
 //                 printf("i=%d\t%f\t%d\t%d\n", i, FS[i][j].YesProbability[k],
FS[i][j].setYesOrNoEachElemNum[k],
FS[numberOfFaults][j].YesOrNoEachElemNum[l]);
//                }
              }
            }
          }
        }
      }
//      printf ("Faults = %d, Attributes = %d , k = %d ,  FS[i][j].YesProbability[%d] =
%d\n", i,j,k,k,FS[i][j].YesProbability[k] );
    }
  }

  fclose(fpin);
  for (m=0; m<numberOfFaults; m++){
    fprintf(fpout5[m], "%d\t%d\t%d\t%d\n",numberOfAttributes, 1, numberOfFiles, 1);
//    fprintf(fpout6[m], "%d\t%d\t%d\t%d\n",numberOfAttributes, 1, numberOfTestFiles,
1);
  }

  fpin = fopen("train_PT_VTCombe_CAB_PCO_LFrevised_PI_sorted_input.txt","r");
  for (m=0; m<numberOfFaults; m++){ // generate 12 files for 12 faults
    fclose(fpin);
    fpin = fopen("train_PT_VTCombe_CAB_PCO_LFrevised_PI_sorted_input.txt","r");
    for (i=0; i<numberOfFiles; i++){
      subFlag = 0;
//      fprintf(fpout5[m], "%s%d\t%s%d\t","Fault",m+1,"file",i+1);
      for (l=0; l<numberOfTestFiles; l++){
        if ((i+1)==testFileIndex[l]) subFlag = 1;
      }
//      if (subFlag==1) fprintf(fpout6[m], "%s%d\t%s%d\t","Fault",m+1,"file",i+1);

      // generate target files for train and test
```

```
      grobleFlag = 0;
      for (n = 0; n<MaximumFaultsCombination; n++)
      {
       fscanf(fpin, "%d", &FaultType[n]);
//       fprintf(fpout5[m], "%d\t", FaultType[n]);
       if (( m == FaultType[n]-1 )&(FaultType[n]>0)&(grobleFlag == 0))
       {
        fprintf(fpout7[m], "%d\n", 1);
        grobleFlag = 1;
       }
      }
      if (grobleFlag == 0)
       fprintf(fpout7[m], "%.2f\n", 0.01);

      if (subFlag==1){
       grobleFlag = 0;
       for (n = 0; n<MaximumFaultsCombination; n++)
       {
//        fprintf(fpout6[m], "%d\t", FaultType[n]);
        if (( m == FaultType[n]-1 )&(FaultType[n]>0)&(grobleFlag == 0))
        {
//          fprintf(fpout8[m], "%d\n", 1);
         grobleFlag = 1;
        }
       }
//      if (grobleFlag == 0)
//       fprintf(fpout8[m], "%.2f\n", 0.01);
      }

      // generate train and test files

      for (j=0; j<numberOfAttributes; j++){
       for (s=0; s<attributeCombIndex[j]; s++)  fscanf(fpin, "%d", &c[s]);
       grobleFlag = 0;
       if (FS[m][j].setYesCounter!=0){
        for (k=0; k<FS[m][j].setYesCounter; k++){
         if (grobleFlag==0) {
          c_sum = 0;
          for (s=0; s<attributeCombIndex[j]; s++)  c_sum = c_sum +
abs(FS[m][j].setYes[k][s] - c[s]);
          if (c_sum == 0 ) {
           grobleFlag = 1;
           fprintf(fpout5[m], "%.2f\t", 1.0);
//           if (subFlag==1) fprintf(fpout6[m], "%.2f\t", 1.0);
          }
```

```
          }
         }
        }
        if ((FS[m][j].setYesOrNoElemNum!=0)&(grobleFlag==0)){
         for (k=0; k<FS[m][j].setYesOrNoElemNum; k++){
          if (grobleFlag == 0) {
           c_sum = 0;
           for (s=0; s<attributeCombIndex[j]; s++)  c_sum = c_sum +
abs(FS[m][j].setYesOrNoElem[k][s] - c[s]);
            if (c_sum == 0 ) {
             grobleFlag = 1;
             if ((j<= 2 )&(m==11)) fprintf(fpout5[m], "%.2f\t", FS[m][j].YesProbability[k]);
             else  fprintf(fpout5[m], "%.2f\t", FS[m][j].YesProbability[k]);
             if (subFlag==1) {
//               if ((j<= 2 )&(m==11)) fprintf(fpout6[m], "%.2f\t",
FS[m][j].YesProbability[k]);
 //              else fprintf(fpout6[m], "%.2f\t", FS[m][j].YesProbability[k]);
              }
             }
            }
           }
        }
        if (grobleFlag==0){
         fprintf(fpout5[m], "%.2f\t", 0.0);
//         if (subFlag==1) fprintf(fpout6[m], "%.2f\t", 0.0);
        }
       // cout<<"*******"<<i<<"******grobleFlag="<<grobleFlag;
      }
      fprintf(fpout5[m], "\n");
//      if (subFlag==1) fprintf(fpout6[m], "\n");
    }
    fprintf(fpout5[m], "\n");
//    fprintf(fpout6[m], "\n");
  }
  for (i=0; i<numberOfFaults; i++){
    if (i==0) fprintf(fpout2,"%s\t",FaultTypeIndicator0);
    if (i==1) fprintf(fpout2,"%s\t",FaultTypeIndicator1);
    if (i==2) fprintf(fpout2,"%s\t",FaultTypeIndicator2);
    if (i==3) fprintf(fpout2,"%s\t",FaultTypeIndicator3);
    if (i==4) fprintf(fpout2,"%s\t",FaultTypeIndicator4);
    if (i==5) fprintf(fpout2,"%s\t",FaultTypeIndicator5);
    if (i==6) fprintf(fpout2,"%s\t",FaultTypeIndicator6);
    if (i==7) fprintf(fpout2,"%s\t",FaultTypeIndicator7);
    if (i==8) fprintf(fpout2,"%s\t",FaultTypeIndicator8);
    if (i==9) fprintf(fpout2,"%s\t",FaultTypeIndicator9);
    if (i==10) fprintf(fpout2,"%s\t",FaultTypeIndicator10);
```

```
   if (i==11) fprintf(fpout2,"%s\t",FaultTypeIndicator11);

   for (j=0;j<numberOfAttributes;j++){
     for (k=0; k<FS[i][j].setYesOrNoCounter; k++){
       if (attributeCombIndex[j] > 1) {
         fprintf(fpout2, "{");
         for (s=0; s<attributeCombIndex[j]; s++) fprintf(fpout2, "%d ",
FS[i][j].setYesOrNo[k][s]);
         fprintf(fpout2, "} ");
       }
       else fprintf(fpout2, "%d ", FS[i][j].setYesOrNo[k][0]);
     }
     fprintf(fpout2, "\t");
   }
   fprintf(fpout2, "\n");
  }


  for (i=0; i<numberOfFaults; i++){
   if (i==0) fprintf(fpout3,"%s\t",FaultTypeIndicator0);
   if (i==1) fprintf(fpout3,"%s\t",FaultTypeIndicator1);
   if (i==2) fprintf(fpout3,"%s\t",FaultTypeIndicator2);
   if (i==3) fprintf(fpout3,"%s\t",FaultTypeIndicator3);
   if (i==4) fprintf(fpout3,"%s\t",FaultTypeIndicator4);
   if (i==5) fprintf(fpout3,"%s\t",FaultTypeIndicator5);
   if (i==6) fprintf(fpout3,"%s\t",FaultTypeIndicator6);
   if (i==7) fprintf(fpout3,"%s\t",FaultTypeIndicator7);
   if (i==8) fprintf(fpout3,"%s\t",FaultTypeIndicator8);
   if (i==9) fprintf(fpout3,"%s\t",FaultTypeIndicator9);
   if (i==10) fprintf(fpout3,"%s\t",FaultTypeIndicator10);
   if (i==11) fprintf(fpout3,"%s\t",FaultTypeIndicator11);

   for (j=0;j<numberOfAttributes;j++){
     for (k=0; k<FS[i][j].setYesCounter; k++){
       if (attributeCombIndex[j] > 1) {
         fprintf(fpout3, "{");
         for (s=0; s<attributeCombIndex[j]; s++) fprintf(fpout3, "%d ",
FS[i][j].setYes[k][s]);
         fprintf(fpout3, "} ");
       }
       else fprintf(fpout3, "%d ", FS[i][j].setYes[k][0]);
     }
     fprintf(fpout3, "\t");
   }
   fprintf(fpout3, "\n");
  }
```

```
  fclose(fpin);
  fclose(fpout1);
  fclose(fpout2);
  fclose(fpout3);

  for (i = 0; i<numberOfFaults; i++){
    fclose(fpout5[m]);
    fclose(fpout7[m]);
  }

  return (0);
}
```

// RNN calibration and test program, listed is example source code for fault 1 RNN//

```
#include <stdio.h>
#include <stdlib.h>  /* malloc */
#include <math.h>

/* user defined data types */
typedef double MATRIX[20][20];
typedef double VECTOR[20];

typedef struct record
{
  VECTOR        x;
  struct record  * next;
} RECORD;

typedef RECORD * POINTER;

/* global variables */
POINTER data_x, data_y, rel, rel_new, previous, cur, cur1, cur2, memb;
MATRIX  r, w, u, r_new, w_new, u_new;
int     NN, n, m, trans, iter, subiter;
VECTOR  x, z, res, y, target;
FILE    * inp1, * inp2, * inp3, * inp4;
FILE    * outpr, * outpp, * outpl, * outpv;

/* prototypes of procedures */
double random(void);
//double triangle(double, double, double, double);
void start_training_reading(void);
void training_reading(void);
void start_result_reading(void);
void result_reading(void);
//double min(double, double);
//double max(double, double);
double t_norm(double, double);
double s_norm(double, double);
double tr(double);
double dist (VECTOR, VECTOR);
double impl(double, double);
void petri_net(VECTOR);
double performance(void);
double deriv_u(int, int);
double dw_1(int, int);
double dw_2(int, int);
```

```
double deriv_w(int, int);
double dr(int, int);
double deriv_r(int, int);
void initial(void);
void show_results(void);
void show_connections(void);
FILE * SafeFopen(char *, char*);
void SafeFclose(FILE *);

/* user defined constants */
#define ALPHA 0.01
#define ITER  200

/* main entrance */
void main ()
{
  POINTER  cur1, cur2;
  int      index, sum, l, it, i, j, k, flag;
  double   error;

  outpr = SafeFopen("result0.txt", "w");
  outpp = SafeFopen("perfor0.txt", "w");
  outpl = SafeFopen("learn0.txt", "w");
  outpv = SafeFopen("verify0.txt", "w");

  sum = 0;
  flag = 0;
  trans = 11;
  n = 11;
  m = 1;

  initial();

  for (it=0; it<ITER; it++) {
    error = 0;
    inp1 = SafeFopen("FTrain0_11 atrributes.txt", "r");
    inp2 = SafeFopen("FTrainT0_11 atrributes.txt", "r");
    printf("%d\n", it);
    fscanf(inp1, "%d %d %d %d", &n, &m, &subiter, &NN);
    trans = n;
    for (l=0; l<subiter; l++) {
      if ((l == 0) & (it == 0)) {
        start_training_reading();
        flag = 1;
      }
      else {
```

```
      training_reading();
      flag = 0;
    }
    cur1 = data_x->next;
    cur2 = data_y->next;

    for (k=0; k<NN; k++) {
      sum = sum + 1;
      index = sum % subiter;
      for (i=0; i<n; i++) {
        x[i] = cur1->x[i];
      }
      for (j=0; j<m; j++) {
        target[j] = cur2->x[j];
      }
      petri_net(x);
      error = error + dist(y, target);
//      fprintf(outpr, "%d\t%.4lf\n", index, dist(y, target));
      if (it == 0) {
        for (j=0; j<m; j++)  fprintf(outpl, "%.4lf\t%.4lf\t", y[j], target[j]);
        fprintf(outpl, "\n");
      }

      if  (it == (ITER-1) ) {
        for (j=0; j<m; j++)   fprintf(outpl, "%.4lf\t%.4lf\t", y[j], target[j]);
        fprintf(outpl, "\n");
      }

      if ((error <= 0.0000001) & (it > 50)) {
        SafeFclose(inp1);
        SafeFclose(inp2);

        inp3 = SafeFopen("FTest0_11 atrributes.txt", "r");
        inp4 = SafeFopen("FTestT0_11 atrributes.txt", "r");
        fscanf(inp3, "%d %d %d %d\n", &n, &m, &subiter, &NN);
        start_result_reading();
        show_results();
        for (l=0; l<subiter-1; l++) {
          result_reading();
          show_results();
        }

        show_connections();
        SafeFclose(outpr);
        SafeFclose(outpp);
        SafeFclose(outpl);
```

```
        SafeFclose(outpv);
        SafeFclose(inp3);
        SafeFclose(inp4);

        return;
      }

    for (j=0; j<trans; j++) {
       for (i=0; i<n; i++) {
         r_new[j][i] = tr(r[j][i] + ALPHA * deriv_r(j, i));
         w_new[j][i] = tr(w[j][i] + ALPHA * deriv_w(j, i));
       }
      }

      for (j=0; j<m; j++) {
        for (i=0; i<trans; i++) {
          u_new[j][i] = tr(u[j][i] + ALPHA * deriv_u(j, i));
        }
      }

      for (j=0; j<trans; j++) {
        for (i=0; i<n; i++) {
          r[j][i] = r_new[j][i];
          w[j][i] = w_new[j][i];
        }
      }

      for (j=0; j<m; j++) {
        for (i=0; i<trans; i++) {
          u[j][i] = u_new[j][i];
        }
      }

      cur1 = cur1->next;
      cur2 = cur2->next;
    }
  }

    error = error/(subiter*NN);
    fprintf(outpp, "%.4lf\n", error);
    SafeFclose(inp1);
    SafeFclose(inp2);
}

 inp3 = SafeFopen("FTest0_11 atrributes.txt", "r");
 inp4 = SafeFopen("FTestT0_11 atrributes.txt", "r");
```

```
  fscanf(inp3, " %d %d %d %d\n", &n, &m, &subiter, &NN);
  start_result_reading();
  show_results();
  for (l=0; l<subiter-1; l++) {
    result_reading();
    show_results();
  }

  show_connections();
  SafeFclose(outpr);
  SafeFclose(outpp);
  SafeFclose(outpl);
  SafeFclose(outpv);
  SafeFclose(inp3);
  SafeFclose(inp4);

} /* end of main() */


/*************************************************************************
******/
double random(void)
{
  int seed;
  double random;

  seed = rand() + 1;
  seed = seed - (seed / 105867) * 104867;
  random = (double) (seed + 1) / 104857;

  return (random);
}

/*************************************************************************
******/
/*double triangle(double x, double a, double m, double b)
{
  double y;

  if (x <= m) y = (x - a) / (m - a);
  if (x > m)  y = 1 - (x - m) / (b - m);
  if (y < 0)  y = 0;
  if (y > 1)  y = 1;

  return (y);
}
```

```
/*************************************************************************
******/
void start_training_reading(void)
{
  int k, i;
//  VECTOR aux;

  data_x =(RECORD *) malloc(sizeof(RECORD));
  data_x->next = NULL;
  previous = data_x;
  for (k=0; k<NN; k++) {
    memb = (RECORD *) malloc(sizeof(RECORD));
    memb->next = NULL;
    for (i=0; i<n; i++) {
      fscanf(inp1, "%lf", &memb->x[i]);
    }
    previous->next = memb;
    previous = previous->next;
  }

  data_y =(RECORD *) malloc(sizeof(RECORD));
  data_y->next = NULL;
  previous = data_y;
  for (k=0; k<NN; k++) {
    memb = (RECORD *) malloc(sizeof(RECORD));
    memb->next = NULL;
    for (i=0; i<m; i++) {
      fscanf(inp2, " %lf", &memb->x[i]);
/*      memb->x[i] = memb->x[i];  */
    }
    previous->next = memb;
    previous = previous->next;
  }

  return;
}

/*************************************************************************
******/
void training_reading(void)
{
  int k, i;
//  VECTOR aux;

  cur1 = data_x->next;
```

```
  cur2 = data_y->next;
  for (k=0; k<NN; k++) {
    for (i=0; i<n; i++) {
      fscanf(inp1, "%lf", &cur1->x[i]);
    }
    cur1 = cur1->next;
  }

  for (k=0; k<NN; k++) {
    for (i=0; i<m; i++) {
      fscanf(inp2, "%lf", &cur2->x[i]);
    }
    cur2 = cur2->next;
  }

  return;
}

/*************************************************************************
******/
void start_result_reading(void)
{
  int k, i;
//  VECTOR aux;

  data_x =(RECORD *) malloc(sizeof(RECORD));
  data_x->next = NULL;
  previous = data_x;
  for (k=0; k<NN; k++) {
    memb = (RECORD *) malloc(sizeof(RECORD));
    memb->next = NULL;
    for (i=0; i<n; i++) {
      fscanf(inp3, "%lf", &memb->x[i]);
/*      memb->x[i] = memb->x[i] */
    }
    previous->next = memb;
    previous = previous->next;
  }

  data_y =(RECORD *) malloc(sizeof(RECORD));
  data_y->next = NULL;
  previous = data_y;
  for (k=0; k<NN; k++) {
    memb = (RECORD *) malloc(sizeof(RECORD));
    memb->next = NULL;
    for (i=0; i<m; i++) {
```

```
      fscanf(inp4, "%lf", &memb->x[i]);
/*      memb->x[i] = memb->x[i];  */
    }
    previous->next = memb;
    previous = previous->next;
  }

  return;
}

/*************************************************************************
******/
void result_reading(void)
{
  int k, i;
//  VECTOR aux;

  cur1 = data_x->next;
  cur2 = data_y->next;
  for (k=0; k<NN; k++) {
    for (i=0; i<n; i++) {
      fscanf(inp3, " %lf", &cur1->x[i]);
/*      cur1->x[i] = cur1->x[i]; */
    }
    cur1 = cur1->next;
  }

  for (k=0; k<NN; k++) {
    for (i=0; i<m; i++) {
      fscanf(inp4, " %lf", &cur2->x[i]);
/*      cur2->x[i] = cur2->x[i]; */
    }
    cur2 = cur2->next;
  }

  return;
}

/*************************************************************************
******/
/*double min(double a, double b)
{
  double min;

  if (a < b) {
    min = a;
```

```
  else
    min = b;
  }

  return (min);
}

/*************************************************************************
******/
//double max(double a, double b)
/*{
  double max;

  if (a > b) {
    max = a;
  else
    max = b;
  }

  return (max);
}

/*************************************************************************
******/
double t_norm(double x, double y)
{
  return (x * y);
}

/*************************************************************************
******/
double s_norm(double a, double b)
{
  return (a + b - a * b);
}

/*************************************************************************
******/
double tr(double a)
{
  double q;

  q = a;
  if (q < 0) a = 0;
  if (q > 1) a = 1;
```

```
  return (a);
}

/*************************************************************************
******/
double dist (VECTOR a, VECTOR b)
{
  double sum, q;
  int i;

  q = 0;
  sum = 0;

  for (i=0; i<m; i++) {
    q = q + sqrt((a[i] - b[i]) * (a[i] - b[i]));
    sum = sum + b[i];
  }

  return (q /sum);
}

/*************************************************************************
******/
double impl(double a, double b)
{
  double impl;

  if (a > b)
    impl = 1 - a + b;
  else
    impl = 1;

  return (impl);
}

/*************************************************************************
******/
void petri_net(VECTOR x)
{
  double q;
  int j, i;

  for (j=0; j<trans; j++) {
    q = 1;
    for (i=0; i<n; i++) {
      q = t_norm(q, s_norm(impl(r[j][i], x[i]), w[j][i]));
```

```
    }
    z[j] = q;
  }

  for (j=0; j<m; j++) {
    q = 0;
    for (i=0; i<trans; i++) {
      q = s_norm(q, t_norm(z[i], u[j][i]));
    }
    y[j] = q;
  }

  return;
}


/*************************************************************************
******/
double performance(void)
{
  double   q;
  int     k, i, j;
  VECTOR  aux, bux;
  POINTER cur, cur1;

  q = 0;
  cur = data_x->next;
  cur1 = data_y->next;
  for (k=0; k<NN; k++) {
    for (i=0; i<n; i++) {
      aux[i] = cur->x[i];
    }
    petri_net(aux);
    for (j=0; j<m; j++) {
      bux[j] = cur1->x[j];
    }
    q = q + dist(bux, y);
    cur = cur->next;
    cur1 = cur1->next;
  }

  return (q);
}

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

```
double deriv_u(int s, int t)
{
//  POINTER cur;
//  VECTOR  rr;
  double   a, deriv_u;
  int     i;

  a = 0;
  for (i=0; i<trans; i++) {
    if (i != t) {
      a = s_norm(a, t_norm(z[i], u[s][i]));
    }
  }

  deriv_u = 2 * (target[s] - y[s]) * z[t] * (1 - a);

  return (deriv_u);
}

/*************************************************************************
******/
double dw_1(int j, int s)
{
  int i;
  double b, dw_1;

  b = 0;
  for (i=0; i<trans; i++) {
    if (i != s) {
      b = s_norm(b, t_norm(z[i], u[j][i]));
    }
  }
  dw_1 = u[j][s] * (1 - b);

  return (dw_1);
}

/*************************************************************************
******/
double dw_2(int s, int t)
{
  double c, dw_2;
  int i;

  c = 1;
  for (i=0; i<n; i++) {
```

```
   if (i != t) {
     c = t_norm(c, s_norm(impl(r[s][i], x[i]), w[s][i]));
   }
  }

  dw_2 = c * (1.0 - impl(r[s][t], x[t]));

  return (dw_2);
}

/*****************************************************************************
******/
double deriv_w(int s, int t)
{
  double q;
  int j;

  q = 0;
  for (j=0; j<m; j++) {
    q = q + 2.0 * (target[j] - y[j]) * dw_1(j, s) * dw_2(s, t);
  }

  return (q);
}

/*****************************************************************************
******/
double dr(int s, int t)
{
  double dd, c;
  int i;

  c = 1;
  for (i=0; i<n; i++) {
    if (i != t)
      c = t_norm(c, s_norm(impl(r[s][i], x[i]), w[s][i]));
  }
  if (r[s][t] > x[t])
    dd = -1;
  else
    dd = 0;

  return (c * (1.0 - w[s][t]) * dd);
}
```

```
/*************************************************************************
******/
double deriv_r(int s, int t)
{
  int j;
  double q;

  q = 0;
  for (j=0; j<m; j++)
     q = q + 2 * (target[j] - y[j]) * dw_1(j, s) * dr(s, t);

  return (q);
}

/*************************************************************************
******/
void initial(void)
{
  int i, j;

  for (j=0; j<trans; j++) {
    for (i=0; i<n; i++) {
      r[j][i] = random();
      w[j][i] = random();
    }
  }

  for (j=0; j<m; j++) {
    for (i=0; i<trans; i++) {
      u[j][i] = random();
    }
  }

  return;
}

/*************************************************************************
******/
void show_results(void)
{
  POINTER cur1, cur2;
  int k, i, j;

  cur1 = data_x->next;
  cur2 = data_y->next;
  for (k=0; k<NN; k++) {
```

```
    for (i=0; i<n; i++) {
      x[i] = cur1->x[i];
    }
    for (j=0; j<m; j++) {
      target[j] = cur2->x[j];
    }

    petri_net(x);

    for (j=0; j<m; j++) {
      fprintf(outpr, "%.4lf\t%.4lf\t", y[j], target[j]);
      fprintf(outpv, "%.4lf\t%.4lf\t", y[j], target[j]);
    }

    fprintf(outpr, "\n");
    fprintf(outpv, "\n");

    cur1 = cur1->next;
    cur1 = cur2->next;
  }

  return;
}

/*************************************************************************
******/
void show_connections(void)
{
  int i, j;

  fprintf(outpr, "r\n");
  for (j=0; j<trans; j++) {
    for (i=0; i<n; i++) {
      fprintf(outpr, "%.4lf\t", r[j][i]);
    }
    fprintf(outpr, "\n");
  }

  fprintf(outpr, "----------------------\n");
  fprintf(outpr, "w\n");
  for (j=0; j<trans; j++) {
    for (i=0; i<n; i++) {
      fprintf(outpr, "%.4lf\t", w[j][i]);
    }
    fprintf(outpr, "\n");
  }
```

```
  fprintf(outpr, "----------------------\n");
  fprintf(outpr, "u\n ");
  for (j=0; j<m; j++) {
    for (i=0; i<trans; i++) {
      fprintf(outpr, "%.4lf\t", u[j][i]);
    }
    fprintf(outpr, "\n");
  }

  return;
}

/*************************************************************************
******/
FILE * SafeFopen(char * f_name, char * mode)
{
  FILE * file_ptr;

  if ( (file_ptr = fopen(f_name, mode)) == NULL)
  {
                fprintf(stderr, "ERROR: File %s could not be opened.\n", f_name);
                exit(1);
  }

  return file_ptr;
}

/*************************************************************************
******/
void SafeFclose(FILE * file_ptr)
{
  if ( fclose(file_ptr) == EOF )
  {
          fprintf(stderr, "ERROR: File cannot be closed.\n");
     exit(1);
  }

  return;
}
```