

Get to know Fann2Mql

Learn from the [Fann2Mql website](#)

1. Install it on your computer
2. Check the documents

Task

1. Run the code and check the result
2. Combine the QFL(in Assignment 1) into this program and compare the MSE vs. iterations for the two systems

You can remove the other products which not listed on your MT4 server. But you should compare **at least 5 products**.

Comparison:

- 2 charts

Chart 1: mean MSE vs. iterations of the two systems

Chart 2: standard deviation of MSE vs. iterations of the two systems

These are standard method in Journal and papers.

Code

```
//*****
*****
// Date      : 8 JAN 2019
// Created by : Dr. Raymond LEE
// Objective  : QUANTUM FINANCE DAILY FORECAST FOR 120 FINANCIAL PRODUCTS FROM
PROVIDED BY FOREX.COM
//          VERSION NO: V7
//          (WITH QPL EVALUATED FROM QF PATENTED SCHRODINGER EQUATION
EVALUATION TECHNIQUE)
//
//
//*****
*****

#include <Fann2MQL.mqh>
#property copyright "Copyright © 2019, DR. RAYMOND LEE"
#property link      "http://QFFC.ORG"

string      DL_Directory  = "120FC_DL";    // FC DAILY Directory
```

```

string    DT_Directory    = "120FC_DT";        // FC DATA  Directory
string    HL_Directory    = "120FC_HL";        // FC HI/L0 Directory
string    PF_Directory    = "120FC_PF";        // FC PERF  Directory
string    NQPR_Directory  = "QPL";             // NQPR Directory
string    DL_FileName     = "";                // File name for FC_DL
int       DL_FileHandle;   // File Handle for FC_DL
string    DT_FileName     = "";                // File name for FC_DT
int       DT_FileHandle;   // File Handle for FC_DT
string    HL_FileName     = "";                // File name for FC_HL
int       HL_FileHandle;   // File Handle for FC_HL
string    PF_FileName     = "";                // File name for FC_PF
int       DPF_FileHandle;  // File Handle for Daily FC_PF
string    DPF_FileName    = "";                // File name for Daily FC_PF
int       PF_FileHandle;   // File Handle for FC_PF
string    NQPR_FileName   = "";                // File name for NQPR
int       NQPR_FileHandle; // File Handle for NQPR
int       nn_layer        = 4;                // Number of layers (4 including input
and output layers)
int       nn_input        = 20;                // Number of input neurones, 5-Day time
series = 20 neurons
int       nn_hidden1      = 10;                // Number of neurones on the first
hidden layer
int       nn_hidden2      = 10;                // number on the second hidden layer
int       nn_output       = 4;                // number of outputs
double    trainingData[][24];                // IMPORTANT! size = nn_input +
nn_output
int       maxTraining     = 2000;              // maximum number of time we will train
double    targetMSE       = 0.00002;          // the Mean-Square Error of the RANN
int       TRAIN_SIZE      = 400;              // USE 500 TRAINING RECORDS
int       TSDATA_SIZE     = 505;              // USE PAST 505 TIME SERIES DATA
RECORDS
uint      stime=0;
uint      etime=0;
uint      Gstime=0;
uint      Getime=0;
uint      tlapse=0;
uint      Gtlapse=0;
int       nTraining=0;
int       nTP=0;
string    TPSymbol        = "";                // Current Trading duct Symbol
string    TP_Code[120]={ "XAGUSD", "CORN", "US30", "AUDUSD", "EURCHF",
"GBPCAD", "NZDJPY", "USDCNH", "XAUAUD", "XAUCHF",
"XAUEUR", "XAUGBP", "XAUJPY", "XAUUSD", "COPPER",
"PALLAD", "PLAT", "UK_OIL", "US_OIL", "US_NATG",
"HTG_OIL", "COTTON", "SOYBEAN", "SUGAR", "WHEAT",
"IT40", "AUS200", "CHINAA50", "ESP35", "ESTX50",
"FRA40", "GER30", "HK50", "JPN225", "N25",
"NAS100", "SIGI", "SPX500", "SWISS20", "UK100",
"US2000", "AUDCAD", "AUDCHF", "AUDCNH", "AUDJPY",
"AUDNOK", "AUDNZD", "AUDPLN", "AUDSGD", "CADCHF",
"CADJPY", "CADNOK", "CADPLN", "CHFHFUF", "CHFJPY",
"CHFNOK", "CHFPLN", "CNHJPY", "EURAUD", "EURCAD",
"EURCNH", "EURCZK", "EURDKK", "EURGBP", "EURHKD",
"EURHUF", "EURJPY", "EURMXN", "EURNOK", "EURNZD",
"EURPLN", "EURRON", "EURRUB", "EURSEK", "EURSGD",

```

```
"EURTRY", "EURUSD", "EURZAR", "GBPAUD", "GBPCHF",
"GBPDKK", "GBPHKD", "GBPJPY", "GBPMXN", "GBPNOK",
"GBPNZD", "GBPPLN", "GBPSEK", "GBPSGD", "GBPUSD",
"GBPZAR", "HKDJPY", "NOKDKK", "NOKJPY", "NOKSEK",
"NZDCAD", "NZDCHF", "NZDUSD", "SGDHKD", "SGDJPY",
"TRYJPY", "USDCAD", "USDCHF", "USDCZK", "USDDKK",
"USDHKD", "USDHUF", "USDILS", "USDJPY", "USDMXN",
"USDNOK", "USDPLN", "USDRON", "USDRUB", "USDSEK",
"USDSGD", "USDTHB", "USDTRY", "USDZAR", "ZARJPY"};
```

```
string TP_CName[120]={ "白银/美元", "玉米", "道琼斯指数", "澳元/美元", "欧元/瑞士法郎", "英镑/加元", "新西兰元/日元",
"美元/人民币", "黄金/澳元", "黄金/瑞士法郎", "黄金/欧元", "黄金/英镑", "黄金/日元", "黄金/美元",
"铜", "钯金", "铂金", "布伦特原油", "纽约原油", "美国天然气", "HTG 燃油", "棉花", "黄豆", "原糖", "小麦",
"意大利40指数", "澳大利亚200", "中国A50指数", "西班牙35指数", "欧洲斯托克50", "巴黎CAC", "德国DAX",
"恒生指数", "日经指数", "荷兰25指数", "纳斯达克", "新加坡指数", "标准普尔500", "瑞士20指数",
"英国富时100", "美国小型股2000", "澳元/加元", "澳元/瑞士法郎", "澳元/人民币", "澳元/日元",
"澳元/挪威克朗", "澳元/新西兰元", "澳元/波兰兹罗提", "澳元/新加坡元", "加元/瑞士法郎",
"加元/日元", "加元/挪威克朗", "加元/波兰兹罗提", "瑞士法郎/匈牙利福林", "瑞士法郎/日元",
"瑞士法郎/挪威克朗", "瑞士法郎/波兰兹罗提", "人民币/日元", "欧元/澳元", "欧元/加元", "欧元/人民币",
"欧元/捷克克朗", "欧元/丹麦克朗", "欧元/英镑", "欧元/港元", "欧元/匈牙利福林", "欧元/日元",
"欧元/墨西哥元", "欧元/挪威克朗", "欧元/新西兰元", "欧元/波兰兹罗提", "欧元/罗马尼亚列伊",
"欧元/俄罗斯卢布", "欧元/瑞典克朗", "欧元/新加坡元", "欧元/新土耳其里拉", "欧元/美元",
"欧元/南非兰特", "英镑/澳元", "英镑/瑞士法郎", "英镑/丹麦克朗", "英镑/港元", "英镑/日元",
"英镑/墨西哥元", "英镑/挪威克朗", "英镑/新西兰元", "英镑/波兰兹罗提", "英镑/瑞典克朗",
"英镑/新加坡元", "英镑/美元", "英镑/南非兰特", "港元/日元", "挪威克朗/丹麦克朗", "挪威克朗/日元",
"挪威克朗/瑞典克朗", "新西兰元/加元", "新西兰元/瑞士法郎", "新西兰元/美元", "新加坡元/港元",
"新加坡元/日元", "新土耳其里拉/日元", "美元/加元", "美元/瑞士法郎", "美元/捷克克朗", "美元/丹麦克朗",
"美元/港元", "美元/匈牙利福林", "美元/以色列新锡克尔", "美元/日元", "美元/墨西哥元", "美元/挪威克朗",
"美元/波兰兹罗提", "美元/罗马尼亚列伊", "美元/俄罗斯卢布", "美元/瑞典克朗", "美元/新加坡元",
"美元/泰铢", "美元/新土耳其里拉", "美元/南非兰特", "南非兰特/日元"};
```

```
string TP_ELName[120]={ "Silver/US Dollar", "Corn", "Dow Jones Index", "Australian Dollar/US Dollar", "EUR/Swiss Franc",
"British Pound/Canadian Dollar", "New Zealand Dollar/Japanese Yen", "US Dollar/Chinese Yuan",
```

"Gold/Australian Dollar", "Gold/Swiss Franc", "Gold/Euro", "Gold/British Pound",
 "Gold/Japanese Yen", "Gold/US Dollar", "Copper", "Palladium", "Platinum", "Brent Crude Oil",
 "WTI Crude Oil", "US Natural Gas", "HTG Oil", "Cotton", "Soybean", "Sugar", "Wheat",
 "Italy 40 Index", "AUSSIE 200", "China A50 Index", "Spain 35 Index", "EURO STOXX 50 Index",
 "CAC 40 Index", "DAX 30 Index", "Hang Seng Index", "Nikkei Index", "Netherlands 25 Index",
 "Nasdaq Index", "Singapore Index", "SP500 Index", "Switzerland 20 Index", "FTSE 100 Index",
 "US Small Cap 2000", "Australian Dollar/Canadian Dollar", "Australian Dollar/Swiss Franc",
 "Australian Dollar/Chinese Yuan", "Australian Dollar/Japanese Yen", "Australian Dollar/Norwegian Krone",
 "Australian Dollar/New Zealand Dollar", "Australian Dollar/Polish Zloty", "Australian Dollar/Singapore Dollar",
 "Canadian Dollar/Swiss Franc", "Canadian Dollar/Japanese Yen", "Canadian Dollar/Norwegian Krone",
 "Canadian Dollar/Polish Zloty", "Swiss Franc/Hungarian Forint", "Swiss Franc/Japanese Yen",
 "Swiss Franc/Norwegian Krone", "Swiss Franc/Polish Zloty", "Chinese Yuan/Japanese Yen", "EUR/Australian Dollar",
 "EUR/Canadian Dollar", "Euro/Chinese Yuan", "EURCzech Koruna", "EUR/Danish Krone", "EUR/British Pound",
 "Euro/Hong Kong Dollar", "EURHungarian Forint", "EUR/Japanese Yen", "Euro/Mexican Peso", "EUR/Norwegian Krone",
 "EUR/New Zealand Dollar", "EUR/Polish Zloty", "Euro/Romanian Leu", "Euro/Russian Ruble", "EUR/Swedish Krona",
 "Euro/Singapore Dollar", "EUR/Turkish Lira", "EUR/US Dollar", "Euro/South African Rand", "British Pound/Australian Dollar",
 "British Pound/Swiss Franc", "British Pound/Danish Krone", "British Pound/Hong Kong Dollar",
 "British Pound/Japanese Yen", "British Pound/Mexican Peso", "British Pound/Norwegian Krone",
 "British Pound/New Zealand Dollar", "British Pound/Polish Zloty", "British Pound/Swedish Krona",
 "British Pound/Singapore Dollar", "British Pound/US Dollar", "British Pound/South African Rand",
 "Hong Kong Dollar/Japanese Yen", "Norwegian Krone/Danish Krone", "Norwegian Krone/Japanese Yen",
 "Norwegian Krone/Swedish Krona", "New Zealand Dollar/Canadian Dollar", "New Zealand Dollar/Swiss Franc",
 "New Zealand Dollar/US Dollar", "Singapore Dollar/Hong Kong Dollar", "Singapore Dollar/Japanese Yen",
 "Turkish Lira/Japanese Yen", "US Dollar/Canadian Dollar", "US Dollar/Swiss Franc", "US DollarCzech Koruna",
 "US Dollar/Danish Krone", "US DollarHong Kong Dollar", "US Dollar/Hungarian Forint", "US Dollar/Israeli Shekel",
 "US Dollar/Japanese Yen", "US Dollar/Mexican Peso", "US Dollar/Norwegian Krone", "US Dollar/Polish Zloty",
 "US Dollar/Romanian Leu", "US Dollar/Russian Ruble", "US Dollar/Swedish Krona", "US Dollar/Singapore Dollar",
 "US Dollar/Thai Baht", "US Dollar/Turkish Lira", "US

```
Dollar/South African Rand","South African Rand/Japanese Yen"};
```

```
string TP_ESName[120]={ "XAGUSD", "Corn", "Dow Jones  
Index", "AUDUSD", "EURCHF", "GBPCAD", "NZDJPY", "USDCNH",
```

```
"XAUAUD", "XAUCHF", "XAUEUR", "XAUGBP", "XAUJPY", "XAUUSD", "Copper", "Palladium", "Plati  
num",
```

```
"UK Crude Oil", "US Crude Oil", "US Natural Gas", "HTG  
Oil", "Cotton", "Soybean", "Sugar", "Wheat",
```

```
"Italy 40", "AUSSIE 200", "China A50", "Spain 35", "Euro  
Stoxx 50", "CAC 40", "DAX 30", "Hang Seng Index",
```

```
"Nikkei", "N25", "Nasdaq", "SIGI", "SP500", "Swiss  
20", "FTSE 100", "US2000",
```

```
"AUDCAD", "AUDCHF", "AUDCNH", "AUDJPY", "AUDNOK", "AUDNZD", "AUDPLN", "AUDSGD", "CADCHF",  
"CADJPY",
```

```
"CADNOK", "CADPLN", "CHFCHF", "CHFJPY", "CHFNO", "CHFPLN", "CNHJPY", "EURAUD", "EURCAD",  
"EURCNH",
```

```
"EURCZK", "EURDKK", "EURGBP", "EURHKD", "EURHUF", "EURJPY", "EURMXN", "EURNOK", "EURNZD",  
"EURPLN",
```

```
"EURRON", "EURRUB", "EURSEK", "EURSGD", "EURTRY", "EURUSD", "EURZAR", "GBPAUD", "GBPCHF",  
"GBPDKK",
```

```
"GBPHKD", "GBPJPY", "GBPMXN", "GBPNOK", "GBPNZD", "GBPPLN", "GBPSEK", "GBPSGD", "GBPUSD",  
"GBPZAR",
```

```
"HKDJPY", "NOKDKK", "NOKJPY", "NOKSEK", "NZDCAD", "NZDCHF", "NZDUSD", "SGDHKD", "SGDJPY",  
"TRYJPY",
```

```
"USDCAD", "USDCHF", "USDCZK", "USDDKK", "USDHKD", "USDHUF", "USDILS", "USDJPY", "USDMXN",  
"USDNOK",
```

```
"USDPLN", "USDRON", "USDRUB", "USDSEK", "USDUSD", "USDTHB", "USDTRY", "USDZAR", "ZARJPY"}  
;
```

```
string TP_CCat[120] = { "金属", "商品", "指数", "外汇1", "外汇2", "外汇3", "外汇7", "外  
汇8",
```

```
"金属", "金属", "金属", "金属", "金属", "金属", "金属", "金  
属", "金属",
```

```
"商品", "商品", "商品", "商品", "商品", "商品", "商品", "商品",  
"指数", "指数", "指数", "指数", "指数", "指数", "指数", "指数",  
"指数", "指数", "指数", "指数", "指数", "指数", "指数", "指数",  
"外汇1", "外汇1", "外汇1", "外汇1", "外汇1", "外汇1", "外汇
```

```
1", "外汇1", "外汇1", "外汇1",
```

```
"外汇2", "外汇2", "外汇2", "外汇2", "外汇2", "外汇2", "外汇
```

```
2", "外汇2", "外汇2", "外汇2",
```

```
"外汇3", "外汇3", "外汇3", "外汇3", "外汇3", "外汇3", "外汇
```

```
3", "外汇3", "外汇3", "外汇3",
```

```
"外汇4", "外汇4", "外汇4", "外汇4", "外汇4", "外汇4", "外汇
```

```
4", "外汇4", "外汇4", "外汇4",
```

```
"外汇5", "外汇5", "外汇5", "外汇5", "外汇5", "外汇5", "外汇
```

```
5", "外汇5", "外汇5", "外汇5",
```

```

        "外汇6","外汇6","外汇6","外汇6","外汇6","外汇6","外汇
6","外汇6","外汇6","外汇6",
        "外汇7","外汇7","外汇7","外汇7","外汇7","外汇7","外汇
7","外汇7","外汇7","外汇7",
        "外汇8","外汇8","外汇8","外汇8","外汇8","外汇8","外汇
8","外汇8","外汇8"};

string      TP_ECat[120] =
{"Metals","Commodity","Index","FX#1","FX#2","FX#3","FX#7","FX#8",

"Metals","Metals","Metals","Metals","Metals","Metals","Metals","Metals","Metals",

"Commodity","Commodity","Commodity","Commodity","Commodity","Commodity","Commodit
y","Commodity",

"Index","Index","Index","Index","Index","Index","Index","Index",

"Index","Index","Index","Index","Index","Index","Index","Index",

"FX#1","FX#1","FX#1","FX#1","FX#1","FX#1","FX#1","FX#1","FX#1","FX#1",

"FX#2","FX#2","FX#2","FX#2","FX#2","FX#2","FX#2","FX#2","FX#2","FX#2",

"FX#3","FX#3","FX#3","FX#3","FX#3","FX#3","FX#3","FX#3","FX#3","FX#3",

"FX#4","FX#4","FX#4","FX#4","FX#4","FX#4","FX#4","FX#4","FX#4","FX#4",

"FX#5","FX#5","FX#5","FX#5","FX#5","FX#5","FX#5","FX#5","FX#5","FX#5",

"FX#6","FX#6","FX#6","FX#6","FX#6","FX#6","FX#6","FX#6","FX#6","FX#6",

"FX#7","FX#7","FX#7","FX#7","FX#7","FX#7","FX#7","FX#7","FX#7","FX#7",

"FX#8","FX#8","FX#8","FX#8","FX#8","FX#8","FX#8","FX#8","FX#8"};

int          TP_No[120]=
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30
,

31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,
58,59,60,

61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,
88,89,90,

91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,11
3,114,115,

116,117,118,119,120};

int          TP_nD[120]=
{3,2,1,5,5,5,3,5,2,2,2,2,0,2,2,2,2,0,2,3,0,2,2,2,2,0,0,0,0,0,1,1,0,0,2,1,1,1,0,1,
1,5,5,4,

3,4,5,5,5,5,3,5,5,3,3,5,5,4,5,5,4,4,5,5,4,3,3,5,5,5,5,4,3,5,4,5,5,5,5,5,5,4,3,4,5
,5,5,5,

```

```
5,5,4,4,5,4,5,5,5,5,4,3,3,5,5,3,5,5,3,5,5,5,3,5,5,5,3,5,5,3,5,5,3};
```

```
double TP_Norm[120]=
```

```
{60,650,50000,2,2.5,4,180,15,3000,2500,2300,2000,250000,2500,500,2000,2200,12000,
```

```
150,
```

```
8,35000,140,2000,40,1000,40000,12000,30000,20000,7000,10000,25000,50000,38000,120
```

```
0,
```

```
12000,1000,5000,20000,13000,3000,2,2,12,180,12,2,6,2,2,180,12,6,600,220,20,10,35,
```

```
3,3,
```

```
20,50,15,2.5,20,600,250,50,20,4,9,10,150,20,3,
```

```
20,3,40,4,4,20,20,350,50,25,4,12,20,
```

```
4,4,50,30,2,26,2,2.5,1.5,1.5,12,180,80,3,2,50,15,18,600,8,250,45,20,8,8,180,20,3,
```

```
80,15,40,18};
```

```
double TP_Range[120]={ 1.2, 25.00, 900, 0.04, 0.04, 0.05, 4, 0.1,
```

```
35, 35,
```

```
35, 35, 2500, 35, 35, 25, 30, 400,
```

```
4, 0.15,
```

```
1200, 6, 50, 3, 40, 1200, 120, 600,
```

```
400, 150,
```

```
200, 1200, 1500, 1800, 12, 280, 15, 125,
```

```
400, 200,
```

```
100, 0.04, 0.04, 0.1, 3, 0.1, 0.04, 0.05,
```

```
0.02, 0.04,
```

```
8, 0.2, 0.08, 10, 3, 0.3, 0.2, 1,
```

```
0.04, 0.04,
```

```
0.4, 0.6, 0.02, 0.07, 0.4, 9, 10, 4,
```

```
0.4, 0.06,
```

```
0.2, 0.07, 8, 0.4, 0.05, 0.2, 0.05, 1.1,
```

```
0.16, 0.16,
```

```
0.9, 1.5, 8, 3.6, 0.9, 0.2, 0.4, 4,
```

```
0.2, 0.04,
```

```
2, 1, 0.08, 1.7, 0.03, 0.02, 0.03, 0.03,
```

```
0.16, 7,
```

```
16, 0.04, 0.03, 1.3, 0.3, 0.04, 19, 0.12,
```

```
8, 2.5,
```

```
0.6, 0.3, 0.23, 17, 0.7, 0.03, 0.6, 0.17,
```

```
1.3, 1.1};
```

```
double TP_SL[120] = { 0.3, 3, 120, 0.003, 0.003, 0.003, 0.3,
```

```
0.003, 3, 3,
```

```
3, 3, 300, 3, 3, 3, 3,
```

```
60, 0.3, 0.03,
```

```
100, 0.6, 3, 0.3, 3, 100, 50,
```

```
100, 100, 30,
```

```
30, 100, 200, 200, 2.5, 30, 2,
```

```
3, 50, 30,
```

```
5, 0.003, 0.003, 0.03, 0.3, 0.03, 0.003,
```

```
0.003, 0.003, 0.003,
```

```
0.3, 0.03, 0.01, 0.5, 0.3, 0.03, 0.02,
```

```
0.05, 0.003, 0.003,
```



```

0.03, 0.03,0.0025,0.0025, 0.03, 0.8, 0.4,
0.1, 0.03, 0.003,
0.005, 0.006, 0.4, 0.01, 0.003, 0.005, 0.003,
0.05, 0.003, 0.003,
0.03, 0.03, 0.3, 0.05, 0.03, 0.003, 0.01,
0.03, 0.003, 0.003,
0.05, 0.03, 0.003, 0.03, 0.003, 0.003, 0.003,
0.003, 0.007, 0.3,
0.3, 0.003, 0.003, 0.1, 0.005, 0.003, 0.3,
0.005, 0.3, 0.003,
0.003, 0.003, 0.003, 0.3, 0.03, 0.003, 0.06,
0.01, 0.07, 0.03};

double TP_TP[120] =
{0.3,3,120,0.003,0.003,0.003,0.3,0.003,3,3,3,3,300,3,3,3,3,60,0.3,0.03,100,0.6,3,
0.3,3,100,50,100,100,30,30,100,200,200,2.5,30,2,3,50,30,5,0.003,0.003,0.03,0.3,
0.03,0.003,0.003,0.003,0.003,0.3,0.03,0.01,0.5,0.3,0.03,0.02,0.05,0.003,0.003,
0.03,0.03,0.0025,0.0025,0.03,0.8,0.4,0.1,0.03,0.003,0.005,0.006,0.4,0.01,0.003,
0.005,0.003,0.05,0.003,0.003,0.03,0.03,0.3,0.05,0.03,0.003,0.01,0.03,0.003,0.003,
0.05,0.03,0.003,0.03,0.003,0.003,0.003,0.003,0.003,0.007,0.3,0.3,0.003,0.003,0.1,0.005,
0.003,0.3,0.005,0.3,0.003,0.003,0.003,0.003,0.3,0.03,0.003,0.06,0.01,0.07,0.03};

int ann[120]; // Identifier of the RANN
int MAX_TP=120; // Max NO OF Forecasting TPs
bool bPass=false;
int nRound=0;
int today;
int DoW;
bool bWorkDay;
bool b_Forecast;
bool b_PF;
bool b_Skip;

// Declare Daily Forecast elements
int D1_YEAR;
int D1_MONTH;
int D1_DAY;
double D0_OPEN;
double D1_OPEN;
double D1_HIGH;
double D1_LOW;
double D1_CLOSE;
double FC_AOPEN;
double FC_AHIGH;
double FC_ALOW;
double FC_ACLOSE;
double FC_AJOPEN;
double FC_AJHIGH;
double FC_AJLOW;

```



```
double    FC_AJCLOSE;
double    FC_Diff;
double    FC_NOPEN;
double    FC_NHIGH;
double    FC_NLOW;
double    FC_NCLOSE;
double    FC_RANGE;
double    MSE;
bool      bMatch;

// Daily Forecast Display Items
string    PName_EN;
string    PName_CH;
string    PCode;
string    PCat_EN;
string    PCat_CH;
double    TP_Low;
double    SL_Low;
double    TP_High;
double    SL_High;
double    FC_MID;
double    QPL1;
double    QPL2;
double    QPL3;
double    QPL4;
double    QPL5;
double    QPL6;
double    QPL7;
double    QPL8;
bool      bQPLFound;

// REMOVE THE RANN FROM MEMORY
int deinit() {
    f2M_destroy_all_anns();
    return(0);
}

int init() {
    int      i,d,t,k,q,nL,nt,nD,ToW;

    // Declare Read DT Data
    int      RDT_YY;
    int      RDT_MM;
    int      RDT_DD;
    double    RDT_OP;
    double    RDT_HI;
    double    RDT_LO;
    double    RDT_CL;

    // Declare Read HL Data
    int      RHL_YY;
    int      RHL_MM;
    int      RHL_DD;
    double    RHL_HI;
    double    RHL_LO;
```

```
// Declare Read PF Data
int      RPF_YY;
int      RPF_MM;
int      RPF_DD;
double   RPF_OP;
double   RPF_HI;
double   RPF_LO;
double   RPF_CL;
double   RPF_FH;
double   RPF_FL;
double   RPF_EH;
double   RPF_EL;
double   RPF_AE;
double   RPF_PE;

// Declare FC_DT Arrays
int      DT_YY[505];
int      DT_MM[505];
int      DT_DD[505];
double   DT_OP[505];
double   DT_HI[505];
double   DT_LO[505];
double   DT_CL[505];

// Declare FC_HL Arrays
int      HL_YY[505];
int      HL_MM[505];
int      HL_DD[505];
double   HL_HI[505];
double   HL_LO[505];

// Declare FC_PF Arrays
int      PF_YY[505];
int      PF_MM[505];
int      PF_DD[505];
double   PF_OP[505];
double   PF_HI[505];
double   PF_LO[505];
double   PF_CL[505];
double   PF_FH[505];
double   PF_FL[505];
double   PF_EH[505];
double   PF_EL[505];
double   PF_AE[505];
double   PF_PE[505];

// Declare NQPR AND QPL ARRAYS
double   NQPR[120][21]; // Array to store ALL NQPR (each 21 energy levels)

// Set Global Start Time
Gstime   = GetTickCount();

// Get Current Day
today    = TimeDay(TimeLocal());
```

```

Dow                = TimeDayOfWeek(TimeLocal());

// Check Whether Need to Forecast or NOT
b_Forecast         = false;
if ((Dow>0) && (Dow<6))
{
    // Mon - Fri -> Forecast
    b_Forecast      = true;
}

// Prepare for Forecast if needed
if (b_Forecast)
{
    // Read ALL NQPR into array NQPR[][]
    ResetLastError();
    NQPR_FileName    = "FX_NQPR.csv";
    NQPR_FileHandle  =
FileOpen(NQPR_Directory+"//"+NQPR_FileName,FILE_COMMON|FILE_READ|FILE_CSV,',');

    if(NQPR_FileHandle!=INVALID_HANDLE)
    {
        for (nTP=0;nTP<MAX_TP;nTP++)
        {
            // Get ALL the 21 NQPR for the first 20 Energy Levels
            for (nL=0;nL<21;nL++)
            {
                NQPR[nTP][nL] = StringToDouble (FileReadString(NQPR_FileHandle));
            }
        } // Loop over ALL products
    } // Valid NQPR File Handle
    // Close NQPR File
    FileClose(NQPR_FileHandle);

    // OPEN THE DAILY FOECAST DATAFILE
    DL_FileName      = "FC_DAILY.csv";
    FileDelete(DL_Directory+"//"+DL_FileName,FILE_COMMON);
    ResetLastError();

    // CREATE FC_DL HANDLE
    DL_FileHandle    =
FileOpen(DL_Directory+"//"+DL_FileName,FILE_COMMON|FILE_READ|FILE_WRITE|FILE_CSV,
',');

    // Write Header Line

FileWrite(DL_FileHandle,"TP_No","TP_Code2","FC_LOW","TP_LOW","SL_LOW","FC_HIGH","
TP_HIGH","SL_HIGH",
        "QPL#1","QPL#2","QPL#3","QPL#4","QPL#5","QPL#6","QPL#7","QPL#8");
} // b_Forecast is TRUE

// CREATE Daily DPF_HANDLE
DPF_FileName        = "PF_DAILY.csv";
FileDelete(DL_Directory+"//"+DPF_FileName,FILE_COMMON);
ResetLastError();
DPF_FileHandle      =

```

```

FileOpen(DL_Directory+"//"+DPF_FileName,FILE_COMMON|FILE_READ|FILE_WRITE|FILE_CSV
,',');

// Write Header Line
FileWrite(DPF_FileHandle,"YEAR","MONTH","DAY",

"TP_No","TP_CODE2","TP_CCat","TP_ECat","TP_CNAME","TP_ESNAME","TP_ELNAME","Digit"
,

"OPEN","HIGH","LOW","CLOSE",
"FC_HIGH","FC_LOW",
"ERR_HIGH","ERR_LOW","AV_ERR","%ERROR");

// LOOP OVER ALL TP
for (nTP=0;nTP<MAX_TP;nTP++)
{
    TPSymbol    = TP_Code[nTP];
    stime       = GetTickCount();
    nTraining    = 0;
    nt          = 0;
    b_Skip      = false;

//*****
//
//  READ AND SET ALL THE D0 AND D1 DATA VALUES
//
//*****

// Check whether the current D1 Bar is today
Print(TPSymbol);
if (TimeDay(iTime(TPSymbol,PERIOD_D1,0)) == today)
{
    // Check previous D1 bar is Mon-Fri or NOT
    nD    = 1;
    bWorkDay = false;
    while (!bWorkDay)
    {
        ToW =TimeDayOfWeek(iTime(TPSymbol,PERIOD_D1,nD));
        if ((ToW>0) && (ToW<6))
        {
            bWorkDay = true;
        }else{
            b_Skip = true;
            nD++;
        }
    }
}

// Today D1 bar exist, get Today OPEN, D-1 OPEN HIGH LOW CLOSE VOL
D0_OPEN    = iOpen(TPSymbol,PERIOD_D1,0);
D1_YEAR    = TimeYear(iTime(TPSymbol,PERIOD_D1,nD));
D1_MONTH   = TimeMonth(iTime(TPSymbol,PERIOD_D1,nD));
D1_DAY     = TimeDay(iTime(TPSymbol,PERIOD_D1,nD));
D1_OPEN    = iOpen(TPSymbol,PERIOD_D1,nD);

```

```

D1_HIGH      = iHigh(TPSymbol,PERIOD_D1,nD);
D1_LOW       = iLow(TPSymbol,PERIOD_D1,nD);
D1_CLOSE     = iClose(TPSymbol,PERIOD_D1,nD);

}else
{
    // Check previous D1 bar is Mon-Fri or NOT
    nD      = 0;
    bWorkDay = false;
    while (!bWorkDay)
    {
        ToW =TimeDayOfWeek(iTime(TPSymbol,PERIOD_D1,nD));
        if ((ToW>0) && (ToW<6))
        {
            bWorkDay = true;
        }else{
            b_Skip = true;
            nD++;
        }
    }
}

// Today D1 Bar does not exist, get yesterday data and set today open
as yestersday close
D1_YEAR      = TimeYear(iTime(TPSymbol,PERIOD_D1,nD));
D1_MONTH     = TimeMonth(iTime(TPSymbol,PERIOD_D1,nD));
D1_DAY       = TimeDay(iTime(TPSymbol,PERIOD_D1,nD));
D1_OPEN      = iOpen(TPSymbol,PERIOD_D1,nD);
D1_HIGH      = iHigh(TPSymbol,PERIOD_D1,nD);
D1_LOW       = iLow(TPSymbol,PERIOD_D1,nD);
D1_CLOSE     = iClose(TPSymbol,PERIOD_D1,nD);
D0_OPEN      = D1_CLOSE;
}

//if(D0_OPEN == 0.0 || D1_CLOSE==0.0) continue;

//*****
//
//  READ FC_DT / FC_HL / FC_PF FILES
//
//*****

Print("#",nTP," ",TPSymbol,": 1.1 READING FC_DT FILE ...");
ResetLastError();
DT_FileName      = TP_No[nTP]+" "+TPSymbol+"_DT.csv";
DT_FileHandle    =
FileOpen(DT_Directory+"//"+DT_FileName,FILE_COMMON|FILE_READ|FILE_CSV,',');

Print("#",nTP," ",TPSymbol,": 1.2 READING FC_HL FILE ...");
ResetLastError();
HL_FileName      = TP_No[nTP]+" "+TPSymbol+"_HL.csv";
HL_FileHandle    =

```

```

FileOpen(HL_Directory+"//"+HL_FileName,FILE_COMMON|FILE_READ|FILE_CSV,',');

Print("#",nTP," ",TPSymbol,": 1.3 READING FC_PF FILE ...");
ResetLastError();
PF_FileName      = TP_No[nTP]+" "+TPSymbol+"_PF.csv";
PF_FileHandle    =
FileOpen(PF_Directory+"//"+PF_FileName,FILE_COMMON|FILE_READ|FILE_CSV,',');

// READ FC_DT[]
if(DT_FileHandle!=INVALID_HANDLE)
{
    for (t=0;t<TSDATA_SIZE;t++)
    {
        // READ FC_DT DATA
        RDT_YY  = StringToInteger(FileReadString(DT_FileHandle));
        RDT_MM  = StringToInteger(FileReadString(DT_FileHandle));
        RDT_DD  = StringToInteger(FileReadString(DT_FileHandle));
        RDT_OP  = StringToDouble (FileReadString(DT_FileHandle));
        RDT_HI  = StringToDouble (FileReadString(DT_FileHandle));
        RDT_LO  = StringToDouble (FileReadString(DT_FileHandle));
        RDT_CL  = StringToDouble (FileReadString(DT_FileHandle));

        // CHECK WHETHER THE FIRST RECORD IS LATEST D1 BAR VALUES
        if (t==0)
        {
            if ((D1_DAY == RDT_DD)|| (DoW<2)|| (b_Skip))
            {
                //FIRST DATA RECORD IS THE CURRENT D1 BAR, NO NEED TO STORE
                INTO ARRAY
                nt      = 0;
                b_PF    = false;
            }else
            {
                //FIRST DATA RECORD IS NOT THE CURRENT D1 BAR, STORE VALUE INTO
                FIRST ARRAY ELEMENT
                nt      = 1;
                b_PF    = true;

                // STORE ALL THE BAR VALUES INTO TS_[0]
                DT_YY[0]  = D1_YEAR;
                DT_MM[0]  = D1_MONTH;
                DT_DD[0]  = D1_DAY;
                DT_OP[0]  = (double)D1_OPEN/TP_Norm[nTP];
                DT_HI[0]  = (double)D1_HIGH/TP_Norm[nTP];
                DT_LO[0]  = (double)D1_LOW/TP_Norm[nTP];
                DT_CL[0]  = (double)D1_CLOSE/TP_Norm[nTP];
            }
        } // Check first record

        // Set the Array Values
        if (nt<TSDATA_SIZE)
        {
            DT_YY[nt]    = RDT_YY;
            DT_MM[nt]    = RDT_MM;
            DT_DD[nt]    = RDT_DD;

```

```

        DT_OP[nt]    = RDT_OP;
        DT_HI[nt]    = RDT_HI;
        DT_LO[nt]    = RDT_LO;
        DT_CL[nt]    = RDT_CL;
    } //if (nt<TSDATA_SIZE)

    // Upcount nt
    nt++;
} // READING FC_DT DATA RECORDS

// Print("Reading ",TPSymbol,"READING FC_DT DATA FILE COMPLETED !!!");

}else
{
    PrintFormat("Failed to open %s file, Error code =
%d",DT_FileName,GetLastError());
    Print("Reading FC_DT DataFile ERROR!!!");
} // CHECK DT_FILEHANDLE

// CLOSE TIME SERIES DATA FILE
FileClose(DT_FileHandle);

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

//
//  WRITE FC_DT DATA FILE IF b_PF IS TRUE
//

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

if (b_PF==true)
{
    Print("#",nTP," : ",TPSymbol,": 1.1 WRITING FC_DT DATA FILE ...");

    FileDelete(DT_Directory+"//"+DT_FileName,FILE_COMMON);
    ResetLastError();

    DT_FileHandle =
FileOpen(DT_Directory+"//"+DT_FileName,FILE_COMMON|FILE_READ|FILE_WRITE|FILE_CSV,
',');

    if(DT_FileHandle!=INVALID_HANDLE)
    {
        for (t=0;t<TSDATA_SIZE;t++)
        {
            FileWrite(DT_FileHandle,DT_YY[t],DT_MM[t],DT_DD[t],
                DoubleToString(DT_OP[t],5), DoubleToString(DT_HI[t],5),
                DoubleToString(DT_LO[t],5), DoubleToString(DT_CL[t],5));
        }
    } // TS_FileHandle

    // CLOSE TIME SERIES DATA FILE

```



```

        FileClose(DT_FileHandle);
    } //CHECK bWriteTS

// *****
// READ FC_HL[]
//

if(HL_FileHandle!=INVALID_HANDLE)
{
    for (t=0;t<TSDATA_SIZE;t++)
    {
        // READ FC_HL DATA
        RHL_YY = StringToInteger(FileReadString(HL_FileHandle));
        RHL_MM = StringToInteger(FileReadString(HL_FileHandle));
        RHL_DD = StringToInteger(FileReadString(HL_FileHandle));
        RHL_HI = StringToDouble (FileReadString(HL_FileHandle));
        RHL_LO = StringToDouble (FileReadString(HL_FileHandle));

        // CHECK WHETHER IT IS WORKING DAY OR THE FIRST RECORD IS TODAY
        if (t==0)
        {
            if ((RHL_DD == today)|| (TimeDayOfWeek(TimeLocal()) == 0) ||
(TimeDayOfWeek(TimeLocal())== 6))
            {
                nt = 0;
            }else
            {
                nt = 1;
                // STORE ALL THE BAR VALUES INTO TS_[0]
                HL_YY[0] = TimeYear(TimeLocal());
                HL_MM[0] = TimeMonth(TimeLocal());
                HL_DD[0] = TimeDay(TimeLocal());
            }
        } // Check first record

        // Set the Array Values
        if (nt<TSDATA_SIZE)
        {
            HL_YY[nt] = RHL_YY;
            HL_MM[nt] = RHL_MM;
            HL_DD[nt] = RHL_DD;
            HL_HI[nt] = RHL_HI;
            HL_LO[nt] = RHL_LO;
        } //if (nt<TSDATA_SIZE)

        // Upcount nt
        nt++;
    } // READING FC_HL DATA RECORDS

    // Print("Reading ",TPSymbol,"READING FC_HL DATA FILE COMPLETED !!!");

}else
{
    PrintFormat("Failed to open %s file, Error code =

```

```

%d",HL_FileName,GetLastError());
    Print("Reading FC_HL DataFile ERROR!!!");
} // CHECK HL_FILEHANDLE

// CLOSE FC_HL DATA FILE
FileClose(HL_FileHandle);

// *****
// READ FC_PF[]
//
if(PF_FileHandle!=INVALID_HANDLE)
{
    for (t=0;t<TSDATA_SIZE;t++)
    {
        // READ FC_DT DATA
        RPF_YY = StringToInteger(FileReadString(PF_FileHandle));
        RPF_MM = StringToInteger(FileReadString(PF_FileHandle));
        RPF_DD = StringToInteger(FileReadString(PF_FileHandle));
        RPF_OP = StringToDouble (FileReadString(PF_FileHandle));
        RPF_HI = StringToDouble (FileReadString(PF_FileHandle));
        RPF_LO = StringToDouble (FileReadString(PF_FileHandle));
        RPF_CL = StringToDouble (FileReadString(PF_FileHandle));
        RPF_FH = StringToDouble (FileReadString(PF_FileHandle));
        RPF_FL = StringToDouble (FileReadString(PF_FileHandle));
        RPF_EH = StringToDouble (FileReadString(PF_FileHandle));
        RPF_EL = StringToDouble (FileReadString(PF_FileHandle));
        RPF_AE = StringToDouble (FileReadString(PF_FileHandle));
        RPF_PE = StringToDouble (FileReadString(PF_FileHandle));

        // CHECK WHETHER THE FIRST RECORD IS LATEST D1 BAR VALUES
        if (t==0)
        {
            if (!b_PF)
            {
                //FIRST DATA RECORD IS THE CURRENT D1 BAR, NO NEED TO STORE
                INTO ARRAY
                nt          = 0;
            }else
            {
                //FIRST DATA RECORD IS NOT THE CURRENT D1 BAR, STORE VALUE INTO
                FIRST ARRAY ELEMENT
                nt          = 1;

                // STORE ALL THE BAR VALUES INTO TS_[0]
                PF_YY[0]    = D1_YEAR;
                PF_MM[0]    = D1_MONTH;
                PF_DD[0]    = D1_DAY;
                PF_OP[0]    = (double)D1_OPEN;
                PF_HI[0]    = (double)D1_HIGH;
                PF_LO[0]    = (double)D1_LOW;
                PF_CL[0]    = (double)D1_CLOSE;

                // CHECK WITH HL_YY, HL_MM, HL_DD TO LOCATE HL_HI & HL_LO
                bMatch = false;
                k        = 0;
            }
        }
    }
}

```

```

        while ((!bMatch)&&(k<10))
        {
            if ((PF_YY[0]==HL_YY[k])&&(PF_MM[0]==HL_MM[k])&&
(PF_DD[0]==HL_DD[k]))
            {
                bMatch = true;

                // CALCULATE THE PF_EH, PF_EL, PF_AE, PF_PE
                PF_FH[0] = HL_HI[k];
                PF_FL[0] = HL_LO[k];
                PF_EH[0] = MathAbs(PF_HI[0]-HL_HI[k]);
                PF_EL[0] = MathAbs(PF_LO[0]-HL_LO[k]);
                PF_AE[0] = MathAbs((PF_EH[0]+PF_EL[0])/2);
                PF_PE[0] = MathAbs(PF_AE[0]/PF_CL[0]);
            }
            k++;
        }
    } // Check first record

    // Set the Array Values
    if (nt<TSDATA_SIZE)
    {
        PF_YY[nt] = RPF_YY;
        PF_MM[nt] = RPF_MM;
        PF_DD[nt] = RPF_DD;
        PF_OP[nt] = RPF_OP;
        PF_HI[nt] = RPF_HI;
        PF_LO[nt] = RPF_LO;
        PF_CL[nt] = RPF_CL;
        PF_FH[nt] = RPF_FH;
        PF_FL[nt] = RPF_FL;
        PF_EH[nt] = RPF_EH;
        PF_EL[nt] = RPF_EL;
        PF_AE[nt] = RPF_AE;
        PF_PE[nt] = RPF_PE;
    } //if (nt<TSDATA_SIZE)

    // Upcount nt
    nt++;
} // READING FC_DT DATA RECORDS

// Print("Reading ",TPSymbol,"READING FC_PF DATA FILE COMPLETED !!!");

}else
{
    PrintFormat("Failed to open %s file, Error code =
%d",PF_FileName,GetLastError());
    Print("Reading FC_PF DataFile ERROR!!!");
} // CHECK PF_FILEHANDLE

// CLOSE FC_PF DATA FILE
FileClose(PF_FileHandle);

```

```

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

//
//  WRITE FC_PF DATA FILE IF bWriteTS IS TRUE
//

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

if (b_PF==true)
{

    Print("#",nTP," : ",TPSymbol,": 1.1 WRITING FC_PF DATA FILE ...");

    // First Write to DPF Summary File
    FileWrite(DPF_FileHandle,PF_YY[0],PF_MM[0],PF_DD[0],
              TP_No[nTP],TP_Code[nTP],TP_CCat[nTP],TP_ECat[nTP],
              TP_CName[nTP],TP_ESName[nTP],TP_ELName[nTP],TP_nD[nTP],

    DoubleToString(PF_OP[0],TP_nD[nTP]),DoubleToString(PF_HI[0],TP_nD[nTP]),DoubleToS
tring(PF_LO[0],TP_nD[nTP]),DoubleToString(PF_CL[0],TP_nD[nTP]),

    DoubleToString(PF_FH[0],TP_nD[nTP]),DoubleToString(PF_FL[0],TP_nD[nTP]),

    DoubleToString(PF_EH[0],5),DoubleToString(PF_EL[0],5),DoubleToString(PF_AE[0],5),
    DoubleToString(PF_PE[0],5));

    FileDelete(PF_Directory+"//"+PF_FileName,FILE_COMMON);
    ResetLastError();

    PF_FileHandle =
    FileOpen(PF_Directory+"//"+PF_FileName,FILE_COMMON|FILE_READ|FILE_WRITE|FILE_CSV,
    ',');

    if(PF_FileHandle!=INVALID_HANDLE)
    {
        for (t=0;t<TSDATA_SIZE;t++)
        {
            FileWrite(PF_FileHandle,PF_YY[t],PF_MM[t],PF_DD[t],

            DoubleToString(PF_OP[t],TP_nD[nTP]),DoubleToString(PF_HI[t],TP_nD[nTP]),DoubleToS
tring(PF_LO[t],TP_nD[nTP]),DoubleToString(PF_CL[t],TP_nD[nTP]),

            DoubleToString(PF_FH[t],TP_nD[nTP]),DoubleToString(PF_FL[t],TP_nD[nTP]),

            DoubleToString(PF_EH[t],5),DoubleToString(PF_EL[t],5),DoubleToString(PF_AE[t],5),
            DoubleToString(PF_PE[t],5));
        }
    } // PF_FileHandle

    // CLOSE TIME SERIES DATA FILE
    FileClose(PF_FileHandle);
} //CHECK bWriteTS

```

```

// DO DAILY FC IF b_Forecast is TRUE
if (b_Forecast)
{

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

//
// STEP 2 - CREATE RANN
//

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

Print("#",nTP," : ",TPSymbol,": 2 CREATE RANN ...");

IndicatorBuffers(0);
IndicatorDigits(6);

// We resize the trainingData array, so we can use it.
// We're gonna change its size one size at a time.
ArrayResize(trainingData,1);

// CREATE NEW ANN
ann[nTP] = f2M_create_standard(nn_layer, nn_input, nn_hidden1, nn_hidden2,
nn_output);

// CHECK IF CREATED SUCCESFULLY. 0 = OK, -1 = error
// debug("f2M_create_standard()",ann);

// SET ACTIVATION FUNCTION
f2M_set_act_function_hidden (ann[nTP], FANN_SIGMOID_SYMMETRIC_STEPWISE);
f2M_set_act_function_output (ann[nTP], FANN_SIGMOID_SYMMETRIC_STEPWISE);
f2M_randomize_weights (ann[nTP], -0.77, 0.77);

// Just to check. Just for debug purpose.
// debug("f2M_get_num_input(ann)",f2M_get_num_input(ann));
// debug("f2M_get_num_output(ann)",f2M_get_num_output(ann));

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

//
// STEP 3 : REGISTER TIME SERIES DATA
// INSERT TIME SERIES DATA FOR TRAINING ONE-BY-ONE
// INPUTS : 5-DAY TIME SERIES (OPEN/HIGH/LOW/CLOSE) - TOTALLY 20 DATA
// D/D-1/D-2/D-3/D-4
// OUTPUT : NEXT-DAY (D+1) OPEN/HIGH/LOW/CLOSE
// Note : d = 0 (First record is the test set)
//

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

```

```

Print("#",nTP," : ",TPSymbol,": 3 REGISTER TIME SERIES DATA ...");

for (d=0;d<TRAIN_SIZE;d++)
{
    prepareData(ann[nTP],"train",
DT_OP[d+1],DT_HI[d+1],DT_LO[d+1],DT_CL[d+1],
                DT_OP[d+2],DT_HI[d+2],DT_LO[d+2],DT_CL[d+2],
                DT_OP[d+3],DT_HI[d+3],DT_LO[d+3],DT_CL[d+3],
                DT_OP[d+4],DT_HI[d+4],DT_LO[d+4],DT_CL[d+4],
                DT_OP[d+5],DT_HI[d+5],DT_LO[d+5],DT_CL[d+5],
                DT_OP[d],DT_HI[d],DT_LO[d],DT_CL[d]);
} // INSERT 500 TIME SERIES DATA INTO TRAINING ARRAY

// Now we print the full training set to the console, to check how it looks
like.
// this is just for debug purpose.
// printDataArray(ann[nTP]);

//*****
//
// STEP 4 - RANN TRAINING
//

//*****

Print("#",nTP," : ",TPSymbol,": 4 TRAINING TIME SERIES DATA ... Target
MSE=",DoubleToString(targetMSE,10));

bPass    = false;
nRound   = 1;

while (!bPass)
{
    for (i=0;i<maxTraining;i++)
    {
        MSE = teach(ann[nTP]);
        if (MSE < targetMSE)
        {
            i = maxTraining; // and we go out of this loop
        }
        nTraining++;
    }

    // we print to the console the MSE value once the training is completed
    // debug("MSE",f2M_get_MSE(ann));

//*****
//
// STEP 5 - RANN RUNNING, USING THE FIRST 5 TIME SERIES RECORD TO PREDICT

```

TODAY

```

//

//*****

// Print(TPSymbol,": 5 Forecast Today ...");

    prepareData(ann[nTP],"compute",DT_OP[0],DT_HI[0],DT_LO[0],DT_CL[0],
                DT_OP[1],DT_HI[1],DT_LO[1],DT_CL[1],
                DT_OP[2],DT_HI[2],DT_LO[2],DT_CL[2],
                DT_OP[3],DT_HI[3],DT_LO[3],DT_CL[3],
                DT_OP[4],DT_HI[4],DT_LO[4],DT_CL[4],
                0,0,0,0);

// De-normalize FC Results
FC_AOPEN    = FC_NOPEN * TP_Norm[nTP];
FC_AHIGH    = FC_NHIGH * TP_Norm[nTP];
FC_ALOW     = FC_NLOW * TP_Norm[nTP];
FC_ACLOSE   = FC_NCLOSE * TP_Norm[nTP];

// Calculate the FC_OPEN DIFF AND CALCULATE THE AJ FC
FC_Diff     = D0_OPEN - FC_AOPEN;
FC_AJOPEN   = FC_AOPEN + FC_Diff;
FC_AJHIGH   = FC_AHIGH + FC_Diff;
FC_AJLOW    = FC_ALOW + FC_Diff;
FC_AJCLOSE  = FC_ACLOSE + FC_Diff;

// Calculate TP_Range
FC_RANGE = FC_AJHIGH - FC_AJLOW;

//          if ((FC_AJHIGH>FC_AJOPEN)&&(FC_AJHIGH>FC_AJLOW)&&
(FC_AJHIGH>FC_AJCLOSE)&&
//          (FC_AJLOW<FC_AJOPEN) &&(FC_AJLOW<FC_AJHIGH) &&
(FC_AJLOW<FC_AJCLOSE)&&(FC_RANGE<TP_Range[nTP]))

// Check bPASS
if ((FC_AJHIGH>FC_AJOPEN)&&(FC_AJHIGH>FC_AJLOW)&&(FC_AJHIGH>FC_AJCLOSE)&&
    (FC_AJLOW<FC_AJOPEN) &&(FC_AJLOW<FC_AJHIGH) &&(FC_AJLOW<FC_AJCLOSE)&&
    (FC_RANGE<(TP_Range[nTP]*2)))
{
    bPass = true;
    Print("#",nTP," : ",TPSymbol," : Round #",nRound," PASSED!!!");
}else
{
    Print("#",nTP," : ",TPSymbol," : Round #",nRound," NOT PASSED!!!");
}

nRound++;

// Randam Weight
f2M_randomize_weights (ann[nTP], -0.77, 0.77);

} // WHILE LOOP ON bPASS

```



```

// Check time
etime    = GetTickCount();
tlapse   = etime - stime;

// Output to Console
Print(TP_No[nTP], " : ", TP_Symbol, ": 6 FORECAST COMPLETED !!! Time Taken :
", tlapse, " msec.", " #Training =", nTraining+1, " MSE
=", DoubleToString(f2M_get_MSE(ann[nTP]), 10));
Print(TP_No[nTP], " : ", TP_Symbol, ": Today Forecast OPEN(", FC_AJOPEN, "),
HIGH(", FC_AJHIGH, "), LOW(", FC_AJLOW, "), CLOSE(", FC_AJCLOSE, ")");

// UPDATE HL_HI[0], HL_LO[0]
HL_HI[0] = DoubleToString(FC_AJHIGH, TP_nD[nTP]);
HL_LO[0] = DoubleToString(FC_AJLOW, TP_nD[nTP]);

// Update ALL Daily Forecast Items
TP_Low   = FC_AJLOW + TP_TP[nTP];
SL_Low   = FC_AJLOW - TP_SL[nTP];
TP_High  = FC_AJHIGH - TP_TP[nTP];
SL_High  = FC_AJHIGH + TP_SL[nTP];

// Calculate the EIGHT closest QPL
if(NQPR[nTP][3] != 0.0) QPL1 = D1_CLOSE / NQPR[nTP][3];
if(NQPR[nTP][2] != 0.0) QPL2 = D1_CLOSE / NQPR[nTP][2];
if(NQPR[nTP][1] != 0.0) QPL3 = D1_CLOSE / NQPR[nTP][1];
if(NQPR[nTP][0] != 0.0) QPL4 = D1_CLOSE / NQPR[nTP][0];

QPL5 = D1_CLOSE * NQPR[nTP][0];
QPL6 = D1_CLOSE * NQPR[nTP][1];
QPL7 = D1_CLOSE * NQPR[nTP][2];
QPL8 = D1_CLOSE * NQPR[nTP][3];

// PRINTOUT to DailyForecast.csv DataFile
FileWrite(DL_FileHandle, TP_No[nTP], TP_Code[nTP],
DoubleToString(FC_AJLOW, TP_nD[nTP]), DoubleToString(TP_Low, TP_nD[nTP]), DoubleToStr
ing(SL_Low, TP_nD[nTP]),
DoubleToString(FC_AJHIGH, TP_nD[nTP]), DoubleToString(TP_High, TP_nD[nTP]), DoubleToS
tring(SL_High, TP_nD[nTP]),
DoubleToString(QPL1, TP_nD[nTP]), DoubleToString(QPL2, TP_nD[nTP]), DoubleToString(QP
L3, TP_nD[nTP]), DoubleToString(QPL4, TP_nD[nTP]),
DoubleToString(QPL5, TP_nD[nTP]), DoubleToString(QPL6, TP_nD[nTP]), DoubleToString(QP
L7, TP_nD[nTP]), DoubleToString(QPL8, TP_nD[nTP]));

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

//
//  WRITE FC_HL DATA FILE
//

```

```

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

Print("#",nTP," : ",TPSymbol,": 4.1 WRITING FC_HL DATA FILE ...");

FileDelete(HL_Directory+"//"+HL_FileName,FILE_COMMON);
ResetLastError();

HL_FileHandle =
FileOpen(HL_Directory+"//"+HL_FileName,FILE_COMMON|FILE_READ|FILE_WRITE|FILE_CSV,
',');

if(HL_FileHandle!=INVALID_HANDLE)
{
    for (t=0;t<TSDATA_SIZE;t++)
    {
        FileWrite(HL_FileHandle,HL_YY[t],HL_MM[t],HL_DD[t],
DoubleToString(HL_HI[t],TP_nD[nTP]),DoubleToString(HL_LO[t],TP_nD[nTP]));
        // Print("t=",t,":",HL_YY[t],":",HL_MM[t],":",HL_DD[t],":",
        //
DoubleToString(HL_HI[t],TP_nD[nTP]),":",DoubleToString(HL_LO[t],TP_nD[nTP]));
    }
} // PF_FileHandle

// CLOSE FC_HL DATA FILE
FileClose(HL_FileHandle);

} // IF WORKING DAY
} // LOOP OVER ALL TP

// Close FC_DL File
FileClose(DL_FileHandle);
FileClose(DPF_FileHandle);

// Check Global Time
Gettime = GetTickCount();
Gtlapse = Gettime - Gstime;

// Output time taken
Print("Total Time Taken : ",Gtlapse," msec");

return(0);
}

int start()
{
    return(0);
}

/*****
** printDataArray()
** Print the datas used for training the neurones

```

```

** This is useless. Just created for debug purpose.
*****/
void printDataArray(int rann) {
    int i,j;
    int bufferSize =
ArraySize(trainingData)/(f2M_get_num_input(rann)+f2M_get_num_output(rann))-1;
    string lineBuffer = "";
    for (i=0;i<bufferSize;i++) {
        for (j=0;j<(f2M_get_num_input(rann)+f2M_get_num_output(rann));j++) {
            lineBuffer = StringConcatenate(lineBuffer, trainingData[i][j], ",");
        }
        debug("dataArray["+i+"]", lineBuffer);
        lineBuffer = "";
    }
}

/*****
** prepareData()
** Prepare the data for either training or computing.
** It takes the data, put them in an array,
** and send them to the training or running function
** Update according to the number of input/output your code needs.
*****/
void prepareData(int rann, string action, double d1_open, double d1_high, double
d1_low, double d1_close,
                double d2_open, double d2_high, double d2_low, double d2_close,
                double d3_open, double d3_high, double d3_low, double d3_close,
                double d4_open, double d4_high, double d4_low, double d4_close,
                double d5_open, double d5_high, double d5_low, double d5_close,
                double out_open, double out_high, double out_low, double
out_close)
{
    double inputVector[];
    double outputVector[];
    // we resize the arrays to the right size
    ArrayResize(inputVector,f2M_get_num_input(rann));
    ArrayResize(outputVector,f2M_get_num_output(rann));

    // InputVectors
    inputVector[0] = d1_open;
    inputVector[1] = d1_high;
    inputVector[2] = d1_low;
    inputVector[3] = d1_close;
    inputVector[4] = d2_open;
    inputVector[5] = d2_high;
    inputVector[6] = d2_low;
    inputVector[7] = d2_close;
    inputVector[8] = d3_open;
    inputVector[9] = d3_high;
    inputVector[10] = d3_low;
    inputVector[11] = d3_close;
    inputVector[12] = d4_open;
    inputVector[13] = d4_high;
    inputVector[14] = d4_low;

```

```

    inputVector[15] = d4_close;
    inputVector[16] = d5_open;
    inputVector[17] = d5_high;
    inputVector[18] = d5_low;
    inputVector[19] = d5_close;

    // Output Vectors
    outputVector[0] = out_open;
    outputVector[1] = out_high;
    outputVector[2] = out_low;
    outputVector[3] = out_close;

    // CHECK WHETHER TRAINING OR RUNNING
    if (action == "train") {
        addTrainingData(rann,inputVector,outputVector);
    }
    if (action == "compute") {
        compute(rann,inputVector);
    }
}

/*****
** addTrainingData()
** Add a single set of training data
** (data example + expected output) to the global training set
*****/
void addTrainingData(int rann, double inputArray[], double outputArray[]) {
    int j;
    int bufferSize =
    ArraySize(trainingData)/(f2M_get_num_input(rann)+f2M_get_num_output(rann))-1;

    //register the input data to the main array
    for (j=0;j<f2M_get_num_input(rann);j++) {
        trainingData[bufferSize][j] = inputArray[j];
    }
    for (j=0;j<f2M_get_num_output(rann);j++) {
        trainingData[bufferSize][f2M_get_num_input(rann)+j] = outputArray[j];
    }

    ArrayResize(trainingData,bufferSize+2);
}

/*****
** teach()
** Get all the trainign data and use them to train the neurones one time.
** In order to perly train the neurones, you need to run ,
** this function many time until the Mean-Square Error get low enough.
*****/
double teach(int rann) {
    int i,j;
    double inputVector[];
    double outputVector[];
    ArrayResize(inputVector,f2M_get_num_input(rann));

```

```

    ArrayResize(outputVector, f2M_get_num_output(rann));
    int call;
    int bufferSize =
    ArraySize(trainingData)/(f2M_get_num_input(rann)+f2M_get_num_output(rann))-1;
    for (i=0; i<bufferSize; i++) {
        for (j=0; j<f2M_get_num_input(rann); j++) {
            inputVector[j] = trainingData[i][j];
        }
        outputVector[0] = trainingData[i][20];
        outputVector[1] = trainingData[i][21];
        outputVector[2] = trainingData[i][22];
        outputVector[3] = trainingData[i][23];
        //f2M_train() is showing the neurones only one example at a time.
        call = f2M_train(rann, inputVector, outputVector);
    }
    // Once we have show them an example,
    // we check if how good they are by checking their MSE.
    // If it's low, they learn good!
    MSE = f2M_get_MSE(rann);
    return(MSE);
}

/*****
** compute()
** Compute a set of data and returns the computed result
*****/
int compute(int rann, double inputVector[]) {
    int out;
    ArrayResize(inputVector, f2M_get_num_input(rann));

    // We sent new data to the neurones.
    out = f2M_run(rann, inputVector);

    // and check what they say about it using f2M_get_output().
    FC_NOPEN = f2M_get_output(rann, 0);
    FC_NHIGH = f2M_get_output(rann, 1);
    FC_NLOW = f2M_get_output(rann, 2);
    FC_NCLOSE = f2M_get_output(rann, 3);
    return(0);
}

/*****
** debug()
** Print data to the console
*****/
void debug(string a, string b) {
    Print(a+" ==> "+b);
}

```