```
//+------------------------------------------------------------------+
//|                                        Inc_CyclicalFunctions.mqh |
//|                                                         M Wilson |
//|                                      https://www.algotrader.blog |
//+------------------------------------------------------------------+
#property copyright "M Wilson"
#property link      "https://www.algotrder.blog"
#property strict
//+------------------------------------------------------------------+
//| Enumerations                                                     |
//+------------------------------------------------------------------+
enum CyclicalLevel
{
   LevelUnspecified,
   AboveUpperBand,
   InUpperBand,
   InLowerBand,
   BelowLowerBand
};

enum CyclicalCross
{
   CrossUnspecified,
   AboveUpper,
   BelowUpper,
   BelowLower,
   AboveLower
};
//+------------------------------------------------------------------+
//| Functions - ATR                                                  |
//+------------------------------------------------------------------+
CyclicalLevel CyclicalLevelForATR(const int intShift)
{

//For the input candle intShift, calculate the ATR.   Then based upon the current chart char

//calculate the average over 24 hours and the standard deviation over 24 hours.   Then retur
   //which indicates if the ATR is in the lower band etc etc.


//First, this only works for periods less than 1 day, so return NA if it is greater than 1 c
   if(PeriodSeconds()>=86400) return LevelUnspecified;


//Get the number of points in the ATR.   Use 10 for 1 hour, then scale up for smaller timefa
   int intPeriod=10*(PeriodSeconds(PERIOD_H1)/PeriodSeconds());
   int intAveragePeriod=24*(PeriodSeconds(PERIOD_H1)/PeriodSeconds());

   //Now get the ATR
   double dblATR=iATR(Symbol(),0,intPeriod,intShift);

   //Now work out the average ATR
   double dblAverage=_CyclicalAverageATR(intShift, intPeriod, intAveragePeriod);

   //Now work out the standard deviation
   double dblStdDev=_CyclicalBandWidthATR(intShift, intPeriod, intAveragePeriod);

   //Now work out which level we are in.
   CyclicalLevel eRet=BelowLowerBand;
   if(dblATR>dblAverage-dblStdDev && dblATR<=dblAverage)
   {
      eRet=InLowerBand;
   }
   else if(dblATR>dblAverage && dblATR<=dblAverage+dblStdDev)
   {
      eRet=InUpperBand;
   }
   else
   {
      eRet=AboveUpperBand;
   }

   return eRet;
```

```
}
int CyclicalCrossForATR(const int intShift, const CyclicalCross eInputCross, double &
dblNumberOfDaysAway)
{

//Given the input intShift, this routine scans backwards in time and attempts to find a poir

//crosses above or below one of the outer bands.   It the returns the candle number of where

//because the period and averaging period etc are store within the routine (to prevent lots

//we are unsure how far away the cross is, so dblNumberOfDaysAway is populated with the numb
   //and the cross.


//WARNING - You may get rounding errors if you compare it to an onscreen indicator due to pc


//If the input eInputCross is unspecified, return intShift and set dblNumberOfDaysAway to 0;
   if(eInputCross==CrossUnspecified)
   {
      dblNumberOfDaysAway=0;
      return intShift;
   }


//Get the number of points in the ATR.   Use 10 for 1 hour, then scale up for smaller timefa
   int intPeriod=10*(PeriodSeconds(PERIOD_H1)/PeriodSeconds());
   int intAveragePeriod=24*(PeriodSeconds(PERIOD_H1)/PeriodSeconds());

   //Define the maximum number of bars that we can process
   int intMax=Bars-intAveragePeriod-intPeriod-1;

   //Define the return values etc
   int intRet=intShift;


//Start on candle intShift an compare it to intShift+1, if we have found the desired cross,
   for(int intOffset=0;intOffset<intMax;intOffset++)
   {
      //Now get the ATR
      double dblATRT=iATR(Symbol(),0,intPeriod,intShift+intOffset);
      double dblATRTm1=iATR(Symbol(),0,intPeriod,intShift+intOffset+1);

      //Get the averages.
      double dblAverageT=_CyclicalAverageATR(intShift+intOffset,intPeriod,
intAveragePeriod);
      double dblAverageTm1=_CyclicalAverageATR(intShift+intOffset+1,intPeriod,
intAveragePeriod);

      //Get the band widths.
      double dblBandWidthT=_CyclicalBandWidthATR(intShift+intOffset,intPeriod,
intAveragePeriod);
      double dblBandWidthTm1=_CyclicalBandWidthATR(intShift+intOffset+1,intPeriod,
intAveragePeriod);

      //Look for the cross
      switch(eInputCross)
      {
         case AboveUpper:
            if(dblATRT>dblAverageT+dblBandWidthT && dblATRTm1<=dblAverageTm1+
dblBandWidthTm1)
            {
               //Calculate the number of days difference
               dblNumberOfDaysAway=((double)intOffset)*PeriodSeconds()/PeriodSeconds(
PERIOD_D1);
               //Return the candle number of the cross.
               return intShift+intOffset;
            }
            break;
         case BelowUpper:
            if(dblATRT<dblAverageT+dblBandWidthT && dblATRTm1>=dblAverageTm1+
dblBandWidthTm1)
```

```
            {
               //Calculate the number of days difference
               dblNumberOfDaysAway=((double)intOffset)*PeriodSeconds()/PeriodSeconds(
PERIOD_D1);
               //Return the candle number of the cross.
               return intShift+intOffset;
            }
            break;
         case BelowLower:
            if(dblATRT<dblAverageT-dblBandWidthT && dblATRTm1>=dblAverageTm1-
dblBandWidthTm1)
            {
               //Calculate the number of days difference
               dblNumberOfDaysAway=((double)intOffset)*PeriodSeconds()/PeriodSeconds(
PERIOD_D1);
               //Return the candle number of the cross.
               return intShift+intOffset;
            }
            break;
         case AboveLower:
            if(dblATRT>dblAverageT-dblBandWidthT && dblATRTm1<=dblAverageTm1-
dblBandWidthTm1)
            {
               Print(" ATR T: ",dblATRT," ATR Tm1: ",dblATRTm1, " Bound T: ",dblAverageT
-dblBandWidthT," Bound Tm1: ",dblAverageTm1-dblBandWidthTm1);
               //Calculate the number of days difference
               dblNumberOfDaysAway=((double)intOffset)*PeriodSeconds()/PeriodSeconds(
PERIOD_D1);
               //Return the candle number of the cross.
               return intShift+intOffset;
            }
            break;
      };
   }

   return intRet;
}
double _CyclicalAverageATR(const int intShift, const int intPeriod, const int
intAveragePeriod)
{
   //Now work out the average ATR
   double dblAverage=0.0;
   for(int iOffset=0;iOffset<intAveragePeriod;iOffset++)
   {
      dblAverage+=iATR(Symbol(),0,intPeriod,intShift+iOffset);
   }
   dblAverage/=intAveragePeriod;

   return dblAverage;
}
double _CyclicalBandWidthATR(const int intShift, const int intPeriod, const int
intAveragePeriod)
{
   double dblAverage=_CyclicalAverageATR(intShift, intPeriod, intAveragePeriod);

   //Now work out the standard deviation
   double dblStdDev=0.0;
   for(int iOffset=0;iOffset<intAveragePeriod;iOffset++)
   {
      double dblTemp=iATR(Symbol(),0,intPeriod,intShift+iOffset);
      dblStdDev+=((dblTemp-dblAverage)*(dblTemp-dblAverage));
   }
   dblStdDev/=intAveragePeriod;
   dblStdDev=sqrt(dblStdDev);


//Multiply the StdDev by 0.5.   This is my hard coded constant for the BandWidth.   I do not
   //inputs into the EA's and so have decided to set this in code.
   dblStdDev*=0.5;

   return dblStdDev;
}
//+------------------------------------------------------------------+
//| Functions - StdDev                                               |
//+------------------------------------------------------------------+
CyclicalLevel CyclicalLevelForStdDev(const int intShift)
{

//For the input candle intShift, calculate the StdDev.   Then based upon the current chart c

//calculate the average over 24 hours and the standard deviation over 24 hours.   Then retur
   //which indicates if the StdDev is in the lower band etc etc.


//First, this only works for periods less than 1 day, so return NA if it is greater than 1 c
   if(PeriodSeconds()>=86400) return LevelUnspecified;


//Get the number of points in the ATR.   Use 10 for 1 hour, then scale up for smaller timefa
   int intPeriod=10*(PeriodSeconds(PERIOD_H1)/PeriodSeconds());
   int intAveragePeriod=24*(PeriodSeconds(PERIOD_H1)/PeriodSeconds());

   //Now get the StdDev
   double dblStdDev=iStdDev(Symbol(),0,intPeriod,0,MODE_SMA,PRICE_CLOSE,intShift);

   //Now work out the average StdDev
   double dblAverage=_CyclicalAverageStdDev(intShift,intPeriod,intAveragePeriod);

   //Now work out the standard deviation
   double dblStdDevOfStdDev=_CyclicalBandWidthStdDev(intShift,intPeriod,intAveragePeriod
);

   //Now work out which level we are in.
   CyclicalLevel eRet=BelowLowerBand;
   if(dblStdDev>dblAverage-dblStdDevOfStdDev && dblStdDev<=dblAverage)
   {
      eRet=InLowerBand;
   }
   else if(dblStdDev>dblAverage && dblStdDev<=dblAverage+dblStdDevOfStdDev)
   {
      eRet=InUpperBand;
   }
   else
   {
      eRet=AboveUpperBand;
   }

   return eRet;
}
int CyclicalCrossForStdDev(const int intShift, const CyclicalCross eInputCross, double &
dblNumberOfDaysAway)
{

//Given the input intShift, this routine scans backwards in time and attempts to find a poir

//crosses above or below one of the outer bands.   It the returns the candle number of where

//because the period and averaging period etc are stored within the routine (to prevent lots

//we are unsure how far away the cross is, so dblNumberOfDaysAway is populated with the numb
   //and the cross.


//WARNING - You may get rounding errors if you compare it to an onscreen indicator due to pc


//If the input eInputCross is unspecified, return intShift and set dblNumberOfDaysAway to 0;
   if(eInputCross==CrossUnspecified)
   {
      dblNumberOfDaysAway=0;
      return intShift;
   }


//Get the number of points in the StdDev.   Use 10 for 1 hour, then scale up for smaller tim
   int intPeriod=10*(PeriodSeconds(PERIOD_H1)/PeriodSeconds());
   int intAveragePeriod=24*(PeriodSeconds(PERIOD_H1)/PeriodSeconds());

   //Define the maximum number of bars that we can process
   int intMax=Bars-intAveragePeriod-intPeriod-1;

   //Define the return values etc
   int intRet=intShift;


//Start on candle intShift an compare it to intShift+1, if we have found the desired cross,
   for(int intOffset=0;intOffset<intMax;intOffset++)
   {
      //Now get the ATR
      double dblStdDevT=iStdDev(Symbol(),0,intPeriod,0,MODE_SMA,PRICE_CLOSE,intShift+
intOffset);
      double dblStdDevTm1=iStdDev(Symbol(),0,intPeriod,0,MODE_SMA,PRICE_CLOSE,intShift+
intOffset+1);

      //Get the averages.
      double dblAverageT=_CyclicalAverageStdDev(intShift+intOffset,intPeriod,
intAveragePeriod);
      double dblAverageTm1=_CyclicalAverageStdDev(intShift+intOffset+1,intPeriod,
intAveragePeriod);

      //Get the band widths.
      double dblBandWidthT=_CyclicalBandWidthStdDev(intShift+intOffset,intPeriod,
intAveragePeriod);
      double dblBandWidthTm1=_CyclicalBandWidthStdDev(intShift+intOffset+1,intPeriod,
intAveragePeriod);

      //Look for the cross
      switch(eInputCross)
      {
         case AboveUpper:
            if(dblStdDevT>dblAverageT+dblBandWidthT && dblStdDevTm1<=dblAverageTm1+
dblBandWidthTm1)
            {
               //Calculate the number of days difference
               dblNumberOfDaysAway=((double)intOffset)*PeriodSeconds()/PeriodSeconds(
PERIOD_D1);
               //Return the candle number of the cross.
               return intShift+intOffset;
            }
            break;
         case BelowUpper:
            if(dblStdDevT<dblAverageT+dblBandWidthT && dblStdDevTm1>=dblAverageTm1+
dblBandWidthTm1)
            {
               //Calculate the number of days difference
               dblNumberOfDaysAway=((double)intOffset)*PeriodSeconds()/PeriodSeconds(
PERIOD_D1);
               //Return the candle number of the cross.
               return intShift+intOffset;
            }
            break;
         case BelowLower:
            if(dblStdDevT<dblAverageT-dblBandWidthT && dblStdDevTm1>=dblAverageTm1-
dblBandWidthTm1)
            {
               //Calculate the number of days difference
               dblNumberOfDaysAway=((double)intOffset)*PeriodSeconds()/PeriodSeconds(
PERIOD_D1);
               //Return the candle number of the cross.
               return intShift+intOffset;
            }
            break;
         case AboveLower:
            if(dblStdDevT>dblAverageT-dblBandWidthT && dblStdDevTm1<=dblAverageTm1-
dblBandWidthTm1)
            {
               //Calculate the number of days difference
               dblNumberOfDaysAway=((double)intOffset)*PeriodSeconds()/PeriodSeconds(
PERIOD_D1);
```

```
               //Return the candle number of the cross.
               return intShift+intOffset;
            }
            break;
      };
   }

   return intRet;
}
double _CyclicalAverageStdDev(const int intShift, const int intPeriod, const int
intAveragePeriod)
{
   //Now work out the average StdDev
   double dblAverage=0.0;
   for(int iOffset=0;iOffset<intAveragePeriod;iOffset++)
   {
      dblAverage+=iStdDev(Symbol(),0,intPeriod,0,MODE_SMA,PRICE_CLOSE,intShift+iOffset);
   }
   dblAverage/=intAveragePeriod;

   return dblAverage;
}
double _CyclicalBandWidthStdDev(const int intShift, const int intPeriod, const int
intAveragePeriod)
{
   double dblAverage=_CyclicalAverageStdDev(intShift, intPeriod, intAveragePeriod);

   //Now work out the standard deviation
   double dblStdDevOfStdDev=0.0;
   for(int iOffset=0;iOffset<intAveragePeriod;iOffset++)
   {
      double dblTemp=iStdDev(Symbol(),0,intPeriod,0,MODE_SMA,PRICE_CLOSE,intShift+
iOffset);
      dblStdDevOfStdDev+=((dblTemp-dblAverage)*(dblTemp-dblAverage));
   }
   dblStdDevOfStdDev/=intAveragePeriod;
   dblStdDevOfStdDev=sqrt(dblStdDevOfStdDev);


//Multiply the StdDev by 0.5.   This is my hard coded constant for the BandWidth.   I do not
   //inputs into the EA's and so have decided to set this in code.
   dblStdDevOfStdDev*=0.5;

   return dblStdDevOfStdDev;
}
```