An Extension of the CTR Extension of Holt-Winters Exponential Smoothing

Ronald D. Notestine
Faculty of Management
Chukyo University

Introduction

Cipra et al presented an extension to Holt-Winters method of exponential smoothing for the case of incomplete seasonal data [Cipra et. al. 1995]. The method follows a suggestion of Wright [Wright 1986], for interpolating, smoothing, and predicting seasonal data when data collection has been irregular. The details of the method, and an implementation of it using the Mathematica programming language was given in an earlier paper [Notestine 1995]. An obvious possible extension to this method is to smooth both forward and backward in time, rather than just forward. The author has not found any references in the literature to such an extenson. In this paper, a simple extension of the method smoothing both forward and backward is introduced.

BiDirectional Exponential Smoothing

First, a few words about exponential smoothing. In the earler paper referred to above, I described exponential smoothing : "Smoothing techniques are commonly used in business and industry both for forecasting, as well as for attempting to discern the 'true' path of some variable amidst the noise of a somewhat random world. Among the various smoothing methods, exponential smoothing is among the most popular [Gardner 1985]. Exponential models are relatively simple to set up, and readily amenable to computerization. According to Gardner, the 'surprising accuracy' of exponential smoothing models may be the primary reason for their great popularity."

$$\hat{Y}_{t+1} = \alpha Y_t + (1-\alpha)\hat{Y}_t$$

$$\hat{Y}_{t+1} = \hat{Y}_t + \alpha(Y_t-\hat{Y}_t)$$

Figure 1

In exponential smoothing, the forecast for any particular time is a weighted average of all of the past values. Thus, exponential smoothing is a variety of moving average. It is called exponential smoothing, because the weights assigned to each of the past values declines geometrically. (That is, according to an exponent given by the number of periods separating the forecast time and the past value in question.)”

This is illustrated in Figure 2, which shows a hypothetical series of data.. Suppose we wish to smooth the data for t=15. We take a weighted average of the actual data for t=15, and the data preceding t=15. The weights assigned to the preceding data decline geometrically with their distance in time from t=15. The relative weight to be assigned to the current data and to the past data is a parameter. The figure does not show the weight assigned to the data of t=15, but rather the manner in which the weights assigned the previous smoothed data points declines geometrically with distance in time.

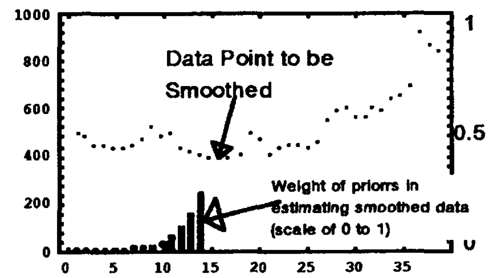

Figure 2 Normal Smoothing

One question that can be asked of the normal methods of smoothing is : Why just use data from the past? The reason we smooth at all is that we believe that the phenomena producing the data change slowly with respect to the time scale at which we collect it, and that much of the short scale variaton is dominated by noise. But, if this is so, then these phenomena will change little for time after the data point in question, as well time before. (In any case, once we do the analysis, it is ALL time past!) In fundamental physical theory, there is nothing to distinguish the flow of time. All phenomena are theoretically reversible, and flow with the arrow of time reversed is just as theoretically possible as that with the normal flow of time. Of course, in cases such as the scrambling of eggs, the difficulty of reversal becomes more than trivial. However, returning to a more realistic example, if an economic time series is being driven up by slowly varying inflation rate, then we have every reason to believe that “future” values will be every bit as valuable in smoothing a data point or estimating a missing data point as the past data.

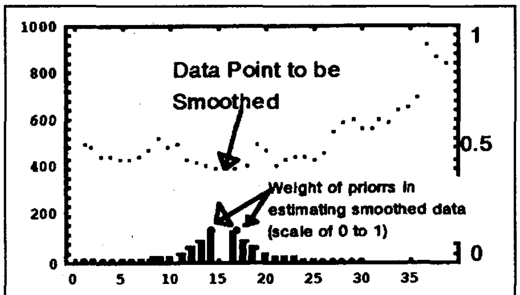

Figure 3 BiDirectional Smoothing

This is illustrted in Figure 3. In this case the date point is smoothed using a weighted average of smoothed values in both directions. The weighting factors decrease geometrically with distance form the time in question.

The Implementation

An objection to the description as literally given above, is that it requires smoothed values to be used before they have been smoothed! That is before we can smooth for time 15, we need to smooth for time 16, which requires we smooth for time 15 etc. The most obvious solution is to smooth forward in the normal way, reverse the data and smooth it "forward", and then comine the results. That is the method used here.

By taking a mean of the forward and backward smoothings, the result is exactly that illustrated above, contributions decrease geometrically both forward and backward in time from the point being smoothed. An objection to this would concern the end regions. Smoothing at the beginning of the data is less accurate because it is based on less data. Near the end of the data, a smoothed value is based on contributions from much more data, and, if the smoothing method is justified, should be more accurate. This in fact can be seen with artificial test data sets, where the smoothed values for the early data are systematically "pulled away" from the observed data. Given this, it seems reasonable to believe that a filter combining information both forward and backward in time, but reducing the influence in one direction as the ends of the data set are approached might be reasonable.

The method has been implemented in the Mathhematica programming language. This allows considrable flexiblilty in the analyysis, and the automaitcc creation of variable names to hold results. The implementation used was designed for clarity and flexibility and is not yet optimised for efficiency.

Some Results

In order to see an example of the method applied, we use a data set of the number of passenger airline miles flown in the United Kingdom from January 1949 to December 1960. This is a much used data set consisting of twelve years of well-behaved, seasonal data.

We examine the method in two situations : No data missing, that is we smooth on the entire data set, and see how forward smoothing only compares with smoothing in both directions in time. Second, we randomly remove data from the set to simulate a situation of missing data, but using actual data to do so.

Situation 1 : All data smoothed. Fugure 4 shows the raw data, with the data points connectd by straight lines for clarity. Figure 5 shows the same data after the application of forward smoothing according to the CTR method. In this case, however, only the dots represent smoothed values, the lines are the lines of the raw values. To differentiate the smoothed dots of figure 5 from the raw of figure 4, the smoothed dots have been slightly enlarged and rendered in gray. As can be seen, the figure, the smoothed values follow the raw data quite closely. This indicates that it might be difficult to improve upon the fit by further smoothing. Indeed, as can be seen by looking at the Root Mean Square Errors for smoothing forward in time only, and for smoothing in both directions in time, the improvement from smoothing in both directions is slight. In fact, this improvement appears to come entirely from the fact that this set seems to "smooth better" in a backward direction, as can be seen by looking at the value of the root mean square error for smoothing backwards in time.

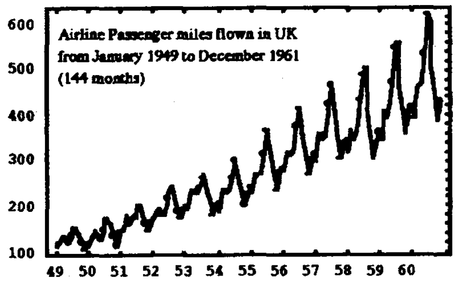

Figure 4 Airline Data (no smoothing)

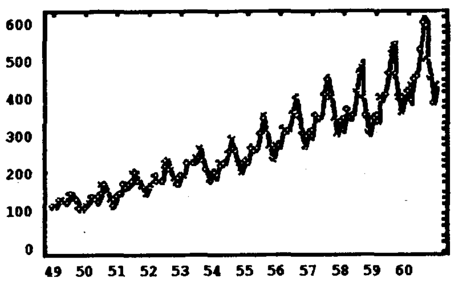

Figure 5 Airline Data
no missing data, but smoothed

RMSE(Forward) = 0.5879
RMSE(Backward) = 0.4112
RMSE(BiDirectional) = 0.5192

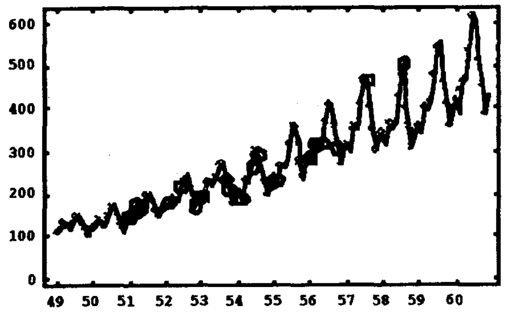

Figure 6 Airline Data
with missing data (circles) smoothed

RMSE(Forward) = 4.3803
RMSE(Backward) = 5.0802
RMSE(BiDirectional) = 1.9406

Situation 2 : a total of 24 points of data have been randomly set to missing (ie "M" was subsituted for the actual data for 24 randomly chosen months.) The only restriction on the months chosen is that they could not be in either the first two or last two years. This is a very unrealisitc restriction, of course. But it is plausible that it provides an advantageous situation for the bi-directional amoothing method. If the method cannot pass muster here, it is unlikiely to do so if there were no restrictions on the months chosen. So, we make this a first test. As can be seen, the root mean square error for the bidirectional smoothing is less than half of either of the forward or backward cases. An anomoly must be mentioned here, however. There appears to be a characteristic of this data that drives the forward smoothed value for month 104 (August 1957) to negative levels (-22), and the following month to less than half the actual value (about 170 for 400 plus). The reason should be investigated in the future to find out if this is merely "local anomoly", or a more serious instability in the method. In the meantime, the single value for August 1957 was averaged with the preceding and following smoothed values. Having done this, the result is illustrated in Figure 6, where the dots are the smoothed values, circles indicate missing values, and the lines graph the original data.

Concluusion

The testing done above is preliminary and tentative only. Many more cases with many more reptitions would be needed to provide any confidence at all on an ad hoc basis. This is left for future work. The program listing, including a superset of the functions needed for this analysis is listed below.

```
(* *** *** *** *** UberCTR *** *** *** ***     *)
(* *** *** *** *** UberCTR *** *** *** ***     *)
(* *** *** *** *** UberCTR *** *** *** ***     *)

Clear[UberCTR]
Options[UberCTR] =
    {period->12,
     smoothingExponents->{.4,.1,.4},
     variableSuffix->None,
     predictAhead->0
    };

UberCTR[dataMat_?MatrixQ, opts___Rule] :=
  Module[{data,p,len,a,g,d,accu,pred,
     varSuf,forwSuf,backSuf,
     yhFyhF,yhByhB,yhDyhD,yvector,
     yhFvector,yhBvector,yhDvector,
     yhPvector,predTable},
    data=dataMat;
    len = Length[data];
    p = period /. {opts} /. Options[UberCTR];
  {a,g,d} = smoothingExponents /.
       {opts} /. Options[UberCTR];
(* nbr of digits of accuracy to keep     *)
  accu = Max[3,2+Round@Log[10.,Max@(yData/@observed[data])]];
(* if y-data is Real, reduce digits          *)
  If[MemberQ[Head/@yData/@data, Real],
  (* then reduce digits of y-data *)
   data =
     {year[#],month[#],N[yData[#],accu]}&/@data
  (* else continue *)
  ];
(* How many months of predictions for y-hat? *)
  pred = predictAhead /. (* default = 0 *)
       {opts} /. Options[UberCTR];
  If[Head[pred]=!=Integer,
  (* then *)
   Print[StringForm[
     “Option predictAhead is ``, it must be type Integer!”,
     predictAhead
   ]];
   pred=Input[“Type an Integer for months to predictAhead”]
  (* else continue *)
  ];
  varSuf = (variableSuffix /. (* default = “” *)
       {opts} /. Options[UberCTR])//ToString;
(* Include varSuf in Global ‘SuffixList’ *)
  SuffixList=Append[SuffixList,varSuf]//Union;
(* Create new Globals, set locals to them *)
  setNames[yhFyhF,yhByhB,yhDyhD,varSuf];
(* Suffixes used in calling CTR *)
  forwSuf = StringJoin[“F”,varSuf];
  backSuf = StringJoin[“B”,varSuf];
(* Name the y-data vector *)
  yvector = yData[data//Transpose];
(* Smooth Forward *)
  yhFvector = (
   Print[“yhFvector”];
   CTR[data,
     periodCTR->p,
     smoothingExponents->{a,g,d},
     variableSuffix->forwSuf
   ]//N[#,accu]&
  );
(* Smooth Backward *)
  yhBvector = (
   Print["yhBvector"];
   CTR[reverseData[data],
     periodCTR->p,
     smoothingExponents->{a,g,d},
     variableSuffix->backSuf
   ]//Reverse//N[#,accu]&

  );
(* Take Goemetric Mean of F & B smoothings   *)
  yhDvector = Sqrt[yhFvector yhBvector]//N[#,accu]&;
(* Predict requested number of months        *)
  yhPvector = Table[YH[data,n,forwSuf],{n,len+1,pred}];
  predTable =
   Table[
     {year[data,mon,p],
      mon,
      "P",
      YH[data,mon,forwSuf],
      "-",
      "-"
    },
    {mon,len+1,len+pred}
  ];
  Do[Print[predTable[[n]]],{n,Length[predTable]}];
(* Create Output Table *)
  Join[
   Transpose[
     Join[
      data//Transpose,
      {yhFvector,
       yhBvector,
       yhDvector}
     ]
   ],
   predTable
```

```
    ]
  ]


(* *** END *** *** UberCTR *** *** END ***   *)
(* *** END *** *** UberCTR *** *** END ***   *)
(* *** END *** *** UberCTR *** *** END ***   *)



(* *** *** ***   CTR   *** *** ***   *)
(* Function to call the rest                 *)
(* Note, this function does no smoothing *)
(* it calculates all info needed to smooth   *)
(* y-hats must be calculated separately  *)
(* *** *** ***   CTR   *** *** ***   *)

(* *** *** Globals Defined by CTR *** ****)
(* Defines symbols with names constructed*)
(* by joining the characters in the list *)
(* just below a suffix supplied by the       *)
(* user. The suffix is specified by the      *)
(* option ‘variableSuffix’. The default is   *)
(* ‘variableSuffix->None’. If a suffix is*)
(* specified by the user, it must be a       *)
(* String. eg ‘variableSuffix->″f″’. If the  *)
(* head is  not String, the user is scolded  *)
(* and the option is reset to ‘None’         *)
(*                  *)
(* List of lead chars for names:             *)
(*                  *)
(* {t, s, y, U, V, W, Sl, Tr, Id}            *)
(*                  *)
(* t[k] =     Months w/ Observations         *)
(* s[k] =     Months w/o Observations        *)
(* y[k] =     Data for month k               *)
(* U[k] =     Level Smoothng Factor      *)
(* V[k] =     Trend Smoothng Factor      *)
(* W[k] =     Seasonal Smoothng Factor   *)
(*                  *)
(* *** *** *** *** *** *** **** *** ***      *)
```

```
Needs[“Utilities`FilterOptions`”];
Clear[CTR,period,smoothingExponents];
Options[CTR] =
   {variableSuffix->None,
    periodCTR->12,
    smoothingExponents->{0.4,0.1,0.4}
   };

CTR[ data_?MatrixQ, opts___Rule ]:=
  Module[
   {dataObs,dataMis,numObs,p,a,g,d,varSuf,
    tt,ss,yy,SlSl,TrTr,IdId,UU,VV,WW,yhyh,
    len},
   dataObs = observed[data];
   dataMis = missing[data];
   len = Length[data];
   numObs = Length[dataObs];
   {a,g,d} = smoothingExponents /.
         {opts} /. Options[CTR];
   varSuf = variableSuffix /.
         {opts} /. Options[CTR];
   p = periodCTR /. {opts} /. Options[CTR];
   (* error checking                     *)
   (* Does data include rep of each month     *)
   If[
    MemberQ[groupByMonth[dataObs,p],{}],
    (* then scold and abort! *)
    Print[StringForm[
      “The following months have no data: ``”,
      Position[groupByMonth[dataObs,p],{}]//Flatten
    ]];
    Return[$Abort]
    (* else continue *)
   ];
   (* Is varSuf a string? *)
   If[!(StringQ[varSuf] || varSuf===None),
   (* then, if NOT a string, do: *)
    Print[StringForm[
      “‘variableSuffix’ is ``, it must be a string!”,
    varSuf
```

```
 ]];
 Print[“Variables will be given NO suffix.”];
 varSuf = None
(* else do nothing *)
];
(* END error checking                      *)
Which[
 varSuf === None,
   setNames[
     tt,ss,yy,SlSl,TrTr,IdId,UU,VV,WW,yhyh
     (* no suffix *)
   ],
 True,
   setNames[
     tt,ss,yy,SlSl,TrTr,IdId,UU,VV,WW,yhyh,
     varSuf (* suffix *)
   ]
];
  (* ***  ‘t’,’s’,’y’      ***      *)
IndexTimePeriodsAndData[
     data,
     tt,ss,yy,
     p
];
  (* ***  ‘Sl’,’Tr’,’Id’   ***      *)
InitializeIndices[
  data,
  yy,
  SlSl,TrTr,IdId,
  p
];
  (* ***   ‘U’,’V’,’W’     ***      *)
InitializeSmoothingCoefficients[
   dataObs,
   UU,VV,WW,
   p,
   {a,g,d}
];
  (* Sets Rest of ‘U’,’V’,’W’             *)
SetSmoothingCoefficients[
      data,
      tt,
      UU,VV,WW,
      p,
      {a,g,d}
   ];
     (* Sets Rest of 'Sl','Tr','Id'      *)
   SetAdjustmentIndices[
      data,
      p,
      yy,
      tt,
      UU,VV,WW,
      SlSl,TrTr,IdId
   ];
   (* set the y-hat for observed t           *)
   setYHatObserved[
      data,
      yhyh,
      tt,
      SlSl,
      IdId
   ];
   (* set the y-hat for missing t            *)
   setYHatMissing[
      data,
      yhyh,
      ss,
      SlSl,
      TrTr,
      IdId,
      p
   ];
   Table[yhyh[k],{k,Length[data]}]
   ]
(* *** END *** ***  CTR ***    *** END *** ***       *)
(* *** END *** ***  CTR ***    *** END *** ***       *)
(* *** END *** ***  CTR ***    *** END *** ***       *)
(* *** *** *** BASIC Functions: *** *** *** *)
(* *** *** *** BASIC Functions: *** *** *** *)
(* *** *** *** BASIC Functions: *** *** *** *)
(*
  generateNames,
  roundReal,
  squareDif,
  reverseData,
  halfString,
  setNames,
  year,
  month,
  yData,
  yHat,
  yFHat,
  yBHat,
  yDHat,
  mean,
  meanYforYear,
  groupByMonth,
  FullSetQ,
  observed,
  missing,
  FutureQ,
  FutureBySameMonthQ,
  MissingQ,
  nextSmooth,
  priorSameMonth,
  timeIntervals
*)
(* *** *** *** BASIC Functions: *** *** *** *)

(* *** *** *** SuffixList *** *** *** ***      *)
(* *** *** *** Some Utilities *** *** ****)

Clear[generateNames,SuffixList,
   PrefixList,DirectionList,sendMissing]

(* Initialize holder of the suffixes     *)
SuffixList = {};(* GLOBAL *)
```

```
(* The following do not change *)
PrefixList = {“t”,”s”,”y”,”Sl”,”Tr”,”Id”,
            “U”,”V”,”W”,”yh”};(* GLOBAL *)
DirectionList = {“F”,”B”,”D”};(* GLOBAL *)
(* result is a list of strings: *)
generateNames := Module[{lis1},
  lis1=Outer[
     StringJoin,
     PrefixList,
     DirecList
   ]//Flatten;
  Outer[StringJoin,lis1,SuffixList]//Transpose
]

(* changes data at ‘places’ to missing *)
sendMissing[data_?MatrixQ, places_List] :=
  Module[{thedata=data},
   ((thedata[[#,-1]] = “M”)&) /@ places;
   thedata
  ]

appendColumn[data_?MatrixQ, col_?VectorQ] :=
   If[Length[col]==Length[data],
   (* then append *)
     Transpose[Append[data//Transpose,col]],
   (* else incompatible lengths ABORT! *)
     Return[“Column wrong length, caannot append!”]
   ]

(* *** END *** SuffixList *** *** END ***     *)
(* *** END *** Some Utilities *** END *** *)

Clear[roundReal,squareDif,reverseData]

(* round reals, leave others as is *)
Attributes[roundReal]= {Listable};
roundReal[x_Real] := Round[x]
roundReal[x_?(Head[x]=!=Real&)] := x

(* square diff of numbers, return string as is  *)
```

```
squareDif[x_?NumberQ,y_?NumberQ] := (x-y)^2
squareDif[x_String,y_?NumberQ] := x
squareDif[x_?NumberQ,y_String] := y
squareDif[{x_,y_}] := squareNumDif[x,y]

(* number data by months in reverse direction *)
(* ie, reverse all except months                  *)
reverseData[data_?MatrixQ] :=
  Transpose[
    {Reverse[year/@data],
     month/@data,
     Reverse[yData/@data]
    }
  ]
Clear[halfString, setNames]

  (* Returns the first half of a string that is:   *)
  (* of even length, with both  halves the same    *)

halfString[str_String] := (* Returns GLOBAL! *)
  Module[{$pos,len},
    If[MemberQ[Characters[str],"$"],
    (* Then we are inside a module       *)
    (* must strip off $moduleNbr         *)
      $pos=StringPosition[str,"$"][[1,1]];
      StringTake[str, Floor[($pos-1)/2] ],
    (* Else halve the string as is          *)
      len = StringLength[str];
      StringTake[str, Floor[len/2] ]
    ]
  ];

  (* *** *** *** setNames *** *** ***    *)
  (* Takes a sequence of symbols, which are   *)
  (* required to be "doubled", ie 'tt' etc,   *)
  (* and sets them to be the undoubled symbol *)
  (* with suffix appended. ie 'tt = tf'       *)
  (* Notice that 'symList' is a sequence of   *)
  (* symbols. The suffix is a string, and the *)
  (* result is that symlist is set to be a    *)
  (* sequence of symbols.                         *)

setNames[ syms__Symbol,suffix_String:""]:=
  Module[{symList={syms}, symHalfStrings},
    (* Convert to string, take 1st half *)
    symHalfStrings =
      halfString /@
        (ToString /@ symList);
    (* affix suffix, convert from string *)
    {syms}//Evaluate =
      ToExpression /@
        (StringJoin[#,suffix]& /@ symHalfStrings)
  ];

(* end of auxilliary naming functions           *)

Clear[year,month,yData,yHat,yFHat,yBHat,yDHat]

year[record_List] := record[[1]];
month[record_List] := record[[2]];
yData[record_List] := record[[3]];
(* the following can be used on output of UberCTR: *)
yHat[record_List] := record[[4]];
yFHat[record_List] := record[[4]];
yBHat[record_List] := record[[5]];
yDHat[record_List] := record[[6]];

year[data_?MatrixQ,mon_Integer,period_Integer:12] :=
  Module[{year1},
    year1 = data//First//year;
    year1+Quotient[mon,period]
  ]

Clear[mean,RMSE,meanYforYear,groupByMonth,FullSetQ,
  observed,missing,FutureQ,FutureBySameMonthQ,
  MissingQ,nextSmooth,priorSameMonth,timeIntervals]

mean[lis_List] := (Plus@@lis)/Length[lis]//N;
mean[seq__?NumberQ] := Plus[seq]/Length[{seq}]//N;
mean[seq__?(NumberQ[#//N]&)] := Plus[seq]/Length[{seq}]//N;
```

```
RMSE[lis_?VectorQ] := Sqrt[Plus@@(lis^2)/Length[lis]]//N;

meanYforYear[data_, k_, period_:12] :=
  mean[yData/@Select[data,(k-1)period<month[#]<=k*period&]];

groupByMonth[data_, period_:12] :=
  Table[
   Select[
    data,
    (Mod[month[#],period] /. 0->period)==k&
   ],
   {k,period}
  ];

FullSetQ[data_, period_:12] :=
  Union[
   (Mod[month[#],period]& /@ data) /. 0->period
  ] == Range[period];

observed[data_] := Select[data, #[[3]]=!="M"&];

missing[data_] := Select[data, #[[3]]=="M"&];

FutureQ[data_?MatrixQ,mon_Integer] :=
      (mon > Max[month/@observed[data]]);

FutureBySameMonthQ[mon_Integer, per_Integer:12] :=
  Module[{sameMonthList},
   sameMonthList =
    month/@
     Part[
      groupByMonth[data,per],
      Mod[mon-1,per]+1
     ];
   mon > Max[sameMonthList]
  ];

MissingQ[data_?MatrixQ,mon_Integer] :=
   !FutureQ[mon] && MemberQ[month/@missing[data]];
```

```
(* The smoothing function, our 'workhorse' *)
nextSmooth[previous_, exponent_, constant_] :=
   previous /(previous + (1-constant)^exponent);

priorSameMonth[      data_?MatrixQ,
       mon_Integer,
       per_Integer:12 ] :=
  Max@Select[
     Join[
       {Mod[mon-1,per]+1-per},
       month/@
         Part[
           groupByMonth[data//observed,per],
           Mod[mon-1,per]+1
         ]
     ],
     (#<mon&)
   ];

(* Time units separating adjacent observations *)
timeIntervals[data_] := (#[[2]]-#[[1]]&) /@
          Partition[month/@data,2,1];



(* For future values of y-hat only        *)
(* Must supply suffix to variable name    *)
(* which is also used to reconstruct      *)
(* other needed names eg 'Id..' the       *)
(* seasonal adjustment index for that     *)
(* data set.          *)

Clear[YH]

YH[data_?MatrixQ,
  mon_Integer?(#>0&),(*must be future*)
  varSuf_String,
  period_Integer:12
] :=
  Module[{dataObs,yhyh,
     SlSl,TrTr,IdId},
   dataObs = data//observed;
  (* Suitability Check: is month ‘mon’ future? *)
   If[Not[FutureQ[dataObs,mon]],
   (* then ABORT *)
    Print[“In ‘YH’, month not a future month!”];
    Return[Null] (* ABORT *)
   (* else continue *)
   ];
  (* create the symbols we will use                *)
   setNames[yhyh,S1S1,TrTr,IdId,varSuf];
  (* DO IT: set yh..[n] to a value                    *)
   yhyh[mon] =
    With[{tn=Max@Select[month/@dataObs,#<mon&]},
      (S1S1[tn] + (mon-tn) TrTr[tn]) *
        IdId[priorSameMonth[dataObs,mon]]
    ]
  ]


(* *** *** END *** BASIC Functions: *** END *** *** *)
(* *** *** END *** BASIC Functions: *** END *** *** *)
(* *** *** END *** BASIC Functions: *** END *** *** *)

(* *** *** *** *** CREATE Globals 1 *** *** *** *** *)
(* *** *** *** *** CREATE Globals 1 *** *** *** *** *)
(* *** *** *** *** CREATE Globals 1 *** *** *** *** *)
(* ‘t’,‘s’,‘y’,‘Sl’,‘Tr’,‘Id’,‘U’,‘V’,‘W’           *)
(* Actual Globals names given in args,                  *)
(*       e.g. ‘tF’ or ‘tB’ etc          *)
(* GLOBALS are all in Arguments to be NAMED         *)
(* by USER                          *)
(* Functions:                          *)
(*     IndexTimePeriodsAndData,               *)
(*     InitializeIndices,                           *)
(*     InitializeSmoothingCoefficients              *)
(* *** *** *** *** CREATE Globals 1 *** *** *** *** *)

Clear[
  IndexTimePeriodsAndData,InitializeIndices,
  InitializeSmoothingCoefficients];
```

```
(* Creates Globals 't', 's', and 'y' *)
(* no output *)
IndexTimePeriodsAndData[data_?MatrixQ,
          tt_Symbol:t,
          ss_Symbol:s,
          yy_Symbol:y,
          period_Integer:12]:=
  Module[{p,dataObs,dataMis,numObs,numMis},
   p=period;
   dataObs=observed[data];
   dataMis=missing[data];
   numObs=Length[dataObs];
   numMis=Length[dataMis];
     (* Before and up to month 0: *)
   Do[tt[k] = k, {k,1-p,0}];
     (* Observed time periods: *)
   Do[tt[k] = (month/@dataObs)[[k]], {k,numObs}];
     (* Missing time periods: *)
   Do[ss[k]=(month/@dataMis)[[k]], {k,numMis}];
     (* Index the data observations: *)
   Do[yy[tt[k]] = (yData/@dataObs)[[k]], {k,1,numObs}];
  ];

(* Creates Globals 'Sl', 'Tr', and 'Id' *)
(* no output *)
InitializeIndices[   data_?MatrixQ,
        yy_Symbol:y,
        SlSl_Symbol:Sl,
        TrTr_Symbol:Tr,
        IdId_Symbol:Id,
        period_Integer:12]:=
  Module[{p,dataObs,k0,k1,dataByMonth,
     yMbyMonthGroup,yMMG,yMY},
    p = period;
    dataObs = observed[data];
    dataByMonth=groupByMonth[dataObs];
    k0 = 1+Quotient[-1+month@dataObs[[1]], p];
    k1 = Module[{maxOfMins},
        maxOfMins =
         Max[month/@
```

```
              (First/@dataByMonth)];
           Max[
             1+Quotient[-1+maxOfMins, p],
             k0+1
           ]
         ];
      yMbyMonthGroup=
        mean/@(Map[yData,#]&[#]&/@dataByMonth);
      yMMG[k_]:= yMMG[k] = yMbyMonthGroup[[k]];
      yMY[k_] := yMY[k] =
         meanYforYear[dataObs,k,p];
      TrTr[0] = (yMY[k1]-yMY[k0]) / ((k1-k0) p);
      S1S1[0] = yMY[k1] - (k0 p - (p-1)/2)TrTr[0];
      Do[     (* CHANGE FROM ADDITIVE CASE *)
        IdId[k-p] =
          mean[  (* DIVISION HERE *)
            yy[month[#]] (**) / (**) (
              yMY[1+Quotient[month[#]-1,p]] +
              TrTr[0]((-Mod[p-month[#],p])+(p-1)/2)
            )& /@ dataByMonth[[k]]
          ],
        {k,p}
      ];
    ];


  (* Initialize smoothing coeffs U, V, and W *)
  (* creates U[0], V[0], and W[-p+1] thru W[0] *)
  (* no output *)

  InitializeSmoothingCoefficients[
      dataObs_?MatrixQ,
      UU_Symbol:U,
      VV_Symbol:V,
      WW_Symbol:W,
      period_Integer:12,
      smoothingExponents_List:{0.4, 0.1, 0.4}
  ]:=
    Module[{alpha,gamma,delta,q,Q,
       dataObsObs,dataGroupedByMonth},
    p = period;
    (* make sure this is observed data *)
    dataObsObs = observed[dataObs];
    (* do not depend on global!! *)
    dataGroupedByMonth = groupByMonth[dataObsObs];
    {alpha,gamma,delta} = smoothingExponents;
    q = mean[#[[2]]-#[[1]]& /@
        Partition[month/@dataObsObs,2,1]];
    VV[0] = 1 - (1-alpha)^q;
    UU[0] = 1 - (1-gamma)^q;
    Do[
      Q[k-p] = Module[{mons},
        mons = month/@(dataGroupedByMonth[[k]]);
        (1+Quotient[Last[mons],p]) / Length[mons]//N
      ];
      WW[k-p] = 1 - (1-delta)^Q[k-p],
    {k,p}
    ]
  ];

(* *** *** END *** CREATE Globals 1 *** END *** *** *)
(* *** *** END *** CREATE Globals 1 *** END *** *** *)
(* *** *** END *** CREATE Globals 1 *** END *** *** *)

(* *** *** *** *** EXTEND Globals 1 *** *** *** *** *)
(* *** *** *** *** EXTEND Globals 1 *** *** *** *** *)
(* *** *** *** *** EXTEND Globals 1 *** *** *** *** *)
(*                     *)
(* ‘U’,’V’,’W’,’Sl’,’Tr’,’Id’,’yh’-obs only              *)
(* GLOBALS are args to be NAMED by USER (e.g. UF)   *)
(* Functions:                       *)
(*    SetSmoothingCoefficients,                     *)
(*    SetAdjustmentIndices                          *)
(*                    *)
(* *** *** *** *** EXTEND Globals 1 *** *** *** *** *)

(* extends Globals ‘U’, ‘V’, and ‘W’ *)
(* no output *)
SetSmoothingCoefficients[
   data_?MatrixQ,
   tt_Symbol:t,
   UU_Symbol:U,
   VV_Symbol:V,
   WW_Symbol:W,
   period_Integer:12,
   smoothingExponents_:{.4, .1, .4}
] :=
  Module[{p,alpha,gamma,delta,dataGroupedByMonth,
     dataObs,numObs,observationIntervals,
     Vlist,Ulist,Wlist},
   (* ASSUMES UU,VV,WW ARE INITIALIZED! *)
   p = period;
   {alpha,gamma,delta} = smoothingExponents;
   dataGroupedByMonth = groupByMonth[data];
   dataObs = observed[data];
   numObs = Length[dataObs];
   observationIntervals =
     timeIntervals[dataObs];
   Vlist =
     FoldList[
       nextSmooth[#1, #2, alpha]&,
       VV[0],
       observationIntervals
     ];
   Ulist =
     FoldList[
       nextSmooth[#1, #2, gamma]&,
       UU[0],
       observationIntervals
     ];
   Wlist=
     Table[
       Module[{monthNbr,WSet},
         monthNbr = Join[
          {k-p},
          month/@
            dataGroupedByMonth[[k]]
         ];
         WSet[k] =
          FoldList[
            nextSmooth[#1, #2/p, delta]&,
            WW[k-p],
            (#[[2]]-#[[1]]&)/@
              Partition[monthNbr,2,1]
          ];
        (*index by monthNbr for sort at end:*)
        Transpose[{monthNbr,WSet[k]}]
       ],
       {k, p}
     ]//(Sort[Flatten[#,1],(#1[[1]]<#2[[1]]&)]&);

   Do[
     VV[tt[k]] = Vlist[[k]];
     UU[tt[k]] = Ulist[[k]],
     {k,numObs}
   ];

   (* Wlist is indexed by month, so just map: *)
   (WW[#[[1]]]=#[[2]])& /@ Wlist;
  ];


(* extends globals 'Sl', 'Tr', 'Id' *)
SetAdjustmentIndices[
      data_?MatrixQ,
      period_Integer:12,
      yy_Symbol:y,
      tt_Symbol:t,
      UU_Symbol:U,
      VV_Symbol:V,
      WW_Symbol:W,
      SlSl_Symbol:Sl,
      TrTr_Symbol:Tr,
      IdId_Symbol:Id
] :=
  Module[{numObs=Length[data//observed],priorMonth},
   Do[ (
     priorMonth =
     priorSameMonth[data//observed,tt[n],period];
     SlSl[tt[n]] =
          (VV[tt[n]] yy[tt[n]] / IdId[priorMonth]) +
            (1-VV[tt[n]]) *
              (SlSl[tt[n-1]] +(tt[n]-tt[n-1])TrTr[tt[n-1]]);
          TrTr[tt[n]] = UU[tt[n]] *
              (SlSl[tt[n]]-SlSl[tt[n-1]])/(tt[n]-tt[n-1]) +
                      (1-UU[tt[n]]) TrTr[tt[n-1]];
          IdId[tt[n]] =  WW[tt[n]] *
               yy[tt[n]] / SlSl[tt[n]] +
                 (1-WW[tt[n]])IdId[priorMonth]
          ),
          {n,numObs}
        ]
      ];

    setYHatObserved[
          data_?MatrixQ,
          yhyh_Symbol:yh,
          tt_Symbol:t,
          SlSl_Symbol:Sl,
          IdId_Symbol:Id
    ] :=
      Module[{numObs=data//observed//Length},
       Do[
         yhyh[tt[n]] = SlSl[tt[n]] IdId[tt[n]],
         {n, numObs}
       ]
      ]

    setYHatMissing[
          data_?MatrixQ,
          yhyh_Symbol:yh,
          ss_Symbol:s,
          SlSl_Symbol:Sl,
          TrTr_Symbol:Tr,
          IdId_Symbol:Id,
          period_Integer:12
    ] :=
      Module[{dataObs,numMis,tn,p},
       p=period;
       dataObs=data//observed;
    numMis=data//missing//Length;
    Do[
      yhyh[ss[n]] =
        With[
            {tn=Max@Select[month/@dataObs,#<ss[n]&]},
            (SlSl[tn] + (n-tn) TrTr[tn]) *
               IdId[priorSameMonth[data,ss[n],p]]
        ],
      {n, numMis}
    ]
   ]

(* *** END *** *** EXTEND Globals 1 *** *** END *** *)
(* *** END *** *** EXTEND Globals 1 *** *** END *** *)
(* *** END *** *** EXTEND Globals 1 *** *** END *** *)


(* *** END *** *** CTR PACKAGE CTR *** *** END ***   *)
(* *** END *** *** CTR PACKAGE CTR *** *** END ***   *)
(* *** END *** *** CTR PACKAGE CTR *** *** END ***   *)
```

References

Aldrin, M. and **Damsleth, E.** "Forecasting Non-seasonal Time Series with Missing Observatons" in The Journal of Forecasting, Vol 8, 97-116 (1989)

Box, E. P. B., et. al. Time Series Analysis: forecasting and control, 3rd ed., Prentice Hall, Englewood Cliffs, NJ, 1994

Cipra, T. et al, Holt-Winters Method with Missing Observations, in Management Science, INFORMS, vol 41, No.1, January 1995.

Gardner, E. S., Exponential Smoothing : The State of the Art in Journal of Forecasting, vol 4, 1-28 (1985)

Hamilton, James D., Time Series Analysis, Princeton University Press, Princeton, NJ, 1994

Notestine, Ronald D., "On Implementing CTR's Extension to Holt-Winters Exponential Smoothing" in Chukyo Deiei Kenkyu (in English in :中京経営研究), Vol 5., No.2 February 1996.

Wilson, J H. and **Barry Keating**, Business Forecasting, 2nd ed. Irwin Publishers, Burr Ridge, Illinois, 1994.

Wright, D. J., Forecasting Data Published at Irregular Intervals Using an Extension of Holt's Method in Management Science, INFORMS, vol 32, No.4, April 1986.