On Implementing CTR's Extension to Holt-Winters Exponential Smoothing

Ronald D. Notestine
Faculty of Management
Chukyo University

Introduction

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.

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.)

Using "Y" for the known data, and "Y" ("y-hat") for the forecast value, we can write exponential smoothing as, shown in Figure 3, at right.

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

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

Figure 1

The form at right is commonly expressed in the following manner: The Forecast for the next period can be expressed as the sum of the forecast for the last period, and a fraction of the error in that forecast. The fraction taken is given by the smoothing constant, commonly expressed as alpha.

Incidentally, the former expression clearly gives the forecast as a weighted average of the observed present value and the forecast value. The weighting factor is again alpha. A philosophical point is whether one should use the word "actual" when referring to the observed value "Y". It is certainly what one has actually observed as the value of y for that time. It is the realization of y that occurred this time. But, there may be a great deal of randomness superimposed upon a "true" value of y. In which case it is perfectly possible that the forecast value might tell us more about the underlying process than the value we actually observe. This is a major reason why smoothing techniques are used, even when no "future forecasting" is needed.

Holt's technique of exponential smoothing is an extension to simple exponential smoothing in which a trend is taken into account. The trend is forecast according to the same idea as above, and then superimposed upon the forecast for the level.

Winter's extension to Holt's technique attempts to account for seasonal variation. It again follows the same idea as for simple exponential smoothing. The seasonal variation is estimated by considering the series of past observations falling in the same period of the season.

Cipra-Trujillo-Rubio (CTR)

Introduction

Cipra, Trujillo, and Rubio (hereafter referred to as CTR) present what they describe as a simple procedure for interpolating, smoothing and predicting the values of a seasonal time series with missing observations. This case is particularly important, as the data one encounters in practice often has some data points missing, was taken at irregular intervals, the interval of data collection changed at some time, or some combination of these problems. The standard methods, mentioned above, assume that all data occurs at regular interval, with none missing, so some way to modify the method to account for irregularly spaced data is quite important.

The Basic Model

As with all exponential smoothing models, the basic philosophy is given by CTR in their first equation (See Figure 2) : That is, the new estimate is a weighted average of the observed value, and the old estimate. But, now

1) the weighting factor changes, and

2) the time is indexed: some times are missing, so there may be a gap between time t-sub-n and time t-sub-(n-1). That is, between two successive observed times, there may be a missing observation.

The Estimates of the level (S), the trend (T), and the seasonal index (I) are given by CTR's

$$\hat{Y}_{t_n} = V_{t_n} Y_{t_n} + (1 - V_{t_n})\hat{Y}_{t_{n-1}}$$

Figure 2

equations (21), (6), and (22) (See Figure 3) :

$$S_{t_n} = V_{t_n} \frac{y_{t_n}}{I^*_{t_{n-1}}} + (1 - V_{t_n})[S_{t_{n-1}} + (t_n - t_{n-1})T_{t_{n-1}}]$$

$$T_{t_n} = U_{t_n} \frac{S_{t_n} - S_{t_{n-1}}}{t_n - t_{n-1}} + (1 - U_{t_n})T_{t_{n-1}}$$

$$I_{t_n} = W_{t_n} \frac{y_{t_n}}{S_{t_n}} + (1 - W_{t_n})I^*_{t_n}$$

Figure 3

The recursive forms of the smoothing coefficients are given by equations (8), (9), and (10) (See Figure 4) : In the equations at right, the asterisk refers to the most recent non-missing data for a period corresponding to the same time of the season. (E.g. the most recent non-missing December observation.) While this may be inconvenient in most programming languages (the author is not a professional programmer, and so makes no claims regarding this), it can be done quite nicely using features of the mathematica language. As an example of the non recursive form, we give that for the smoothing coefficient for the level (See Figure 5). Mathematica is well adapted to the use of recursive formulations, so we shall not use the form at right

$$V_{t_n} = \frac{V_{t_{n-1}}}{b_{t_n} + V_{t_{n-1}}}, \quad b_{t_n} = (1-\alpha)^{t_n - t_{n-1}}$$

$$U_{t_n} = \frac{U_{t_{n-1}}}{d_{t_n} + U_{t_{n-1}}}, \quad d_{t_n} = (1-\gamma)^{t_n - t_{n-1}}$$

$$W_{t_n} = \frac{W^*_{t_{n-1}}}{f_{t_n} + W^*_{t_{n-1}}}, \quad f_{t_n} = (1-\delta)^{(t_n - t_{n-1})/p}$$

Figure 4

$$V_{t_n} = \frac{1}{\sum_{i=1}^{n} (1 - \alpha)^{t_n - t_{n-1}}}$$

Figure 5

Forecasts and interpolations are calculated from the same formula. Thus, interpolations under this scheme are actually just forecasts using only the data prior to the period of the "interpolation" (See Figure 6).

$$\hat{Y}_{m+t_n}(t_n) = (S_{t_n} + mT_{t_n})I_{m+t_n}{}^*$$

Figure 6

Smoothed values for non-missing observations are calculated according to (See Figure 7) :

$$\hat{Y}_{t_n}(t_n) = S_{t_n} I_{t_n}$$

Figure 7

Initial trend and initial level of the series are calculated according to (See Figure 8) :

$$T_0 = \frac{\bar{y}_{k_1} - \bar{y}_{k_0}}{(k_1 - k_0)p} \qquad S_0 = \bar{y}_{k_0} - \left(pk_0 - \frac{p-1}{2}\right)T_0$$

Figure 8

The initial seasonal adjustment indices need a tad more calculation. It is necessary to set initial values of the smoothing coefficients for a full period prior to the beginning of the data. (E.g. a full year, in the most common case.) Then, it is necessary to find the first season with any data at all (not hard) and the first season by which at least one observation is present for all periods. (E.g. the year by which there is at least one observation for every month.) (See Figure 9).

The sum is over all periods of the same position within the seasonal period. For example, all January's, or all Junes. k(i) is the number of terms in the sum for time period i.

y-bar is the mean of data observations for seasonal period k (ie year k).

$$I_i = \frac{1}{k(i)} \sum_k I_{i+kp}$$

$$I_{i+kp} = \frac{y_{i+kp}}{\bar{y}_k + \left(i + \frac{p-1}{2}\right)T_0}$$

Figure 9

About the Implementation

data is read in from an external data file. The external file is presumed to be a text file, such as might be written to disk by any data base, statistics, or spread sheet program. The structure of the file is assumed to be a repetition of (year, month, date), although other structures can be easily accommodated. Missing data points are assumed to marked by an ascii text upper case "M". Again, any other marker is easily accommodated.

(It is important to note that the 'month' numbers in the data set do NOT repeat. The months for the second year will begin with 13 for January, or some later month.)

Named functions are defined for extracting year, month, or date from an element of the data set. The form of the mathematica programming language make these especially handy in the sequel. Initializing the trend coefficient requires that the average of the y data be found for a seasonal period (year). It is exceptionally easy to construct such a function in mathematica. A major point being that it is of no concern whether any data are missing for the year, the average automatically accounts for missing data.

By using the Apply ('@@' below) and 'Length', The average for a list easily results regardless of the number of elements. (The result is converted to decimal to avoid the possibility of fractions being displayed.)

```
In[1]:=mean [lis__List] := (Plus@@lis)/Length [lis]  //N
```

By using an anonymous function, the relevant data is selected. By using the 'Map' capability ('/@' below), only the y-data are extracted before being fed into the averaging function.

```
In[1]:=meanYforSeasonalPeriod [data__, k__, per__:12] :=
    mean [
      yData/@
        Select [
          data,
            (k-1) per <month [#] <=k*per&
        ]
    ]
```

It is also quite easy to write a function that will gather all of the data belonging to the same month. The following function takes all of the data, which is in ascending order, and gathers together all of the records for January, all of the records for February, etc. Or, if the period is different from 12, another period can be given as a second argument. Note that the anonymous function for selecting data changes all zeroe's to 12 (or whatever the period is). This is because Mod [12, 12] is 0, but we want this to be 12 for the 12th month, of course. The grouping is done for every month 'k' from 1 to the number of periods in one season, usually 12.

```
In[2]:=groupByMonth [data__, per__:12] :=
    Table [
      Select [
        data,
          (Mod [month [#] , per]/.0-> per) ==k&] ,
            {k, per}
        ]
```

We also need to test a list of records to see if we have included at least one for each month. This is easily done by extracting the months from the records, applying the 'Union' function, which essentially converts the list to a set by eliminating duplicates, and then asking if the result is the same as the set of all integers from 1 to the number of periods in one season. This is more easily coded than explained.

```
In [ 3 ] : =FullSetQ [data__, per__: 12]  : =
        Union [
          Mod [month [#] , per] &/@
              data/. 0 -> per
        ] ==Range [per]
```

Normally, the non recursive form for the smoothing coefficients is used. However, when data are missing, one must be very careful about the exponents, as they no longer increase by one, but rather by the amount of gap in between non-missing observations. By using mathematica's 'FoldList' function, this is automatically done. First, we generate a list of the time differences between successive observations. Then, we define a general function for the relation between successive values of any of the smoothing coefficients.

It might be worth looking at this in more detail. 'dataObs' is the list of all non-missing records. It is of the form {year, time (month), y-data}

```
In [ 4 ] : =dataObs//Short
```

```
Out [ 4 ]={{1991,  1 , 491}, {1991,  2 , 475}, <<36>>, {1994, 48, 832}}
```

We extract the times (months).

```
In [ 5 ] : =month/@dataObs//Short
```

```
Out [ 5 ]={ 1 ,  2 ,  4 ,  5 ,  6 ,  8 ,  9 , 10, 11, <<25>>, 44, 45, 46, 47, 48}
```

Next, we partition this list into pairs of successive times (months)

```
In [ 6 ] : = Partition [month/@dataObs,  2 ,  1 ]//Short
```

```
Out[6]={{1, 2}, {2, 4}, {4, 5}, <<33>>, {46, 47}, {47, 48}}
```

Finally, we map an anonymous function to take the differences

```
In[7]:=tTimeDiffList=(# [[2]] -# [[1]] &)/@
                            Partition [month/@dataObs, 2, 1] ;
        tTimeDiffList//Short

Out[7]={1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, <<21>>, 1, 1, 1, 1, 1, 1}
```

Here is our general function for smoothing coefficients.

```
In[8]:=next [previous__, exponent__, constant__] :=
            previous/(previous+(1-constant)exponent)
```

To get the list of smoothing coefficients 'V [t [n]]', we need only fold this into the list of time differences, using V [0], and the factor 'a' (alpha). The function 'FoldList' acts in the following way:

```
In[9]:=FoldList [f, x0, {x1, x2}]

Out[9]={x0, f [x0, x1], f [f [x0, x1], x2]}
```

So, starting with 'V [0]', the following expression "folds together" the entire list of 'V' values.

```
In[10]:=Vlist=
        FoldList [
            next [#1, #2, a] &,
            V [0] ,
            tTimeDiffList
        ] ;
        Vlist//Short

Out[10]={0.468372, 0.438398, 0.549097, <<35>>, 0.403139}
```

The smoothing coefficients for the seasonal index, must be "folded" separately for each month of the year, and scaled by a factor of 12, or whatever the number of time periods per seasonal period is. Again this can be done neatly. First, for each month, the initial month for year 0 (one of months minus11 through 0) is added to the list.

For example, for month 3 (March), the augmented list of non-missing months is :

```
In [11] : =k= 3 ;p=12;
        mons=Join [{k-p}, month/@dataGroupedByMonth [[k]]]
```

```
Out [11]={-9, 15, 27, 39}
```

Partitioning, and taking differences

```
In [12] : =(# [[2]] -# [[1]] &)/@Partition [mons, 2, 1]
```

```
Out [12]={24, 12, 12}
```

Then, folding the 'next' function, starting with W [3-12], we get the list of the rest of the W [n] for n corresponding to a non-missing march :

```
In [14] : =FoldList [next [#1, #2 /p, d] &, W [k-p] , %]
```

```
{0.49394, 0.578425, 0.490846, 0.449968}
```

Putting this all into one expression, and using module to provide for local variables, We get the complete list of seasonal smoothing factors. However, there is one remaining wrinkle : they are still grouped by month name. So, at the end, we sort by time period, so that January of the second year will come immediately after December of the first year, etc.

```
In [15] : =Wlist=
        Table [
          Module [{mons, WSet},
            mons=Join [
              {k-p},
              month/@
                dataGroupedByMonth [[k]]
            ] ;
            WSet [k] =
              FoldList [
                next [#1, #2 /p, d] &,
                W [k-p] ,
            (# [[2]] -# [[1]] &)/@
                  Partition [mons, 2, 1]
          ] ;
        Transpose [{mons, WSet [k]}]
      ] ,
      {k, p}
    ]//(Sort [Flatten [#, 1] , (#1  [[1]] <#2  [[1]] &)] &);
    Wlist//Short
```

```
Out [15]={{-11, 0.49394}, <<49>>, {48, 0.554905}}
```

However, while we have calculated the values for the seasonal smoothing constant, W, we have not actually yet set the W to the values we found. Ordinarily, a Do loop might be used. However, since each record includes not just the value, but the time period, there is a very compact option available. It is based on the fact that we are setting W only for observed times, and the observed times are the first elements of the elements of the list ' Wlist'. We wish to set W [t [n]], where the nth element of 'Wlist' is the pair {t [n], W [t[n]]}. That, for each element, the first sub-element is the index, and the second is the corresponding value. So we map the following function onto the list :

```
In [16] : =(W [# [[1]]] =# [[2]])&/@Wlist;
```

```
In [17] : =Table [{t [k] , W [t [k]]}, {k, numObs}]//Take [#, 4] &
```

```
Out [17]={{1, 0.451524}, {2, 0.451524}, {4, 0.4}, {5, 0.4}}
```

The formulae for the level, trend, and seasonal index are connected, and thus cannot be calculated separately. While it would be possible to construct a folding operation, it seems easiest to use a Do loop. Because the time has been indexed for all non-missing data, it is particularly straightforward.

```
In [18] : =
Do [
    Sl [t [n]] =
    (V [t [n]] y [t [n]]/Id [prevSameMonth [t [n]]])+
      (1-V [t [n]])(Sl [t [n-1]]+(t [n] -t [n-1])Tr [t [n-1]]);
    Tr [t [n]] =U [t [n]] *
        (Sl [t [n]] -Sl [t [n-1]])/(t [n] -t [n-1])+
                                                (1-U [t [n]])Tr [t [n-1]] ;
    Id [t [n]] =W [t [n]] *
              y [t [n]]/Sl [t [n]]+
                (1-W [t [n]])Id [prevSameMonth [t [n]]] ;
    yh [t [n]] =Sl [t [n]] Id [t [n]]
    ),
    {n, numObs}
]
```

The data used

The data used is hypothetical data based on a set used by CTR in their paper (prices of pigs at the market).

```
In [19] : =yHats=Table [yh [k] , {k, numMis+numObs}] ;
```

```
In [20] : =finalTableau=
        Append [dataAll//Transpose, yHats]//Transpose;
```

Showing this is table form, with the markers "M" eliminated:

```
In [21] : =TableForm [finalTableau,
        TableHeadings->{None, {"year", "time", "Y", "Y-hat"}},
        TableSpacing->{ 0 , 3 }
]/."M"->""
```

```
    year   time    Y      Y-hat
    1991     1    491    476.75
    1991     2    475    462.497
    1991     3           422.047
    1991     4    441    444.223
    1991     5    439    440.592
    1991     6    425    428.213
    1991     7           444.36
```

1991	8	434	441.08
1991	9	450	453.831
1991	10	466	488.937
1991	11	523	522.153
1991	12	483	482.987
1992	13	494	470.158
1992	14	437	431.365
1992	15	412	405.884
1992	16	403	410.336
1992	17	394	400.659
1992	18	384	388.164
1992	19	389	392.676
1992	20	402	403.019
1992	21		419.781
1992	22		480.448
1992	23	487	492.824
1992	24	461	460.971
1993	25		421.407
1993	26		378.693
1993	27	404	395.21
1993	28	433	423.164
1993	29	446	436.656
1993	30	441	436.233
1993	31	435	439.675
1993	32	451	455.461
1993	33		483.825
1993	34	539	543.554
1993	35	585	586.835
1993	36		557.223
1994	37	595	579.42
1994	38	560	549.025
1994	39	560	545.078
1994	40		555.899

1994	41	598	588.642
1994	42	580	580.021
1994	43	636	624.741
1994	44	655	654.606
1994	45	685	688.518
1994	46	908	878.04
1994	47	864	890.921
1994	48	832	843.816

Plots

We generate the plots of the smoothed and of the actual data, but do not show them. We join the points of the smoothed plot, but not the actual data. The forecast for one year ahead is shown in red (gray) dots.

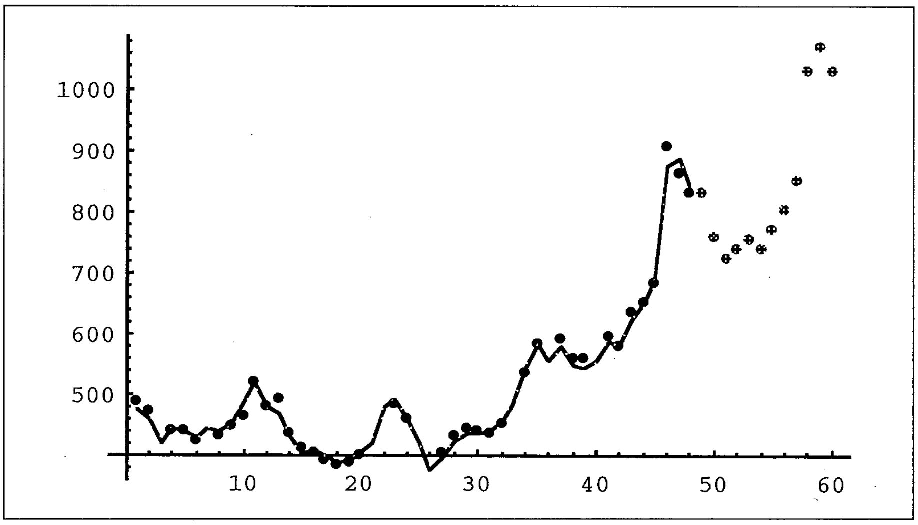
```
In [22] : =rawDataPlot=
        ListPlot [Rest/@dataObs,
              PlotStyle->{PointSize [.015]},
              DisplayFunction->Identity
        ] ;
```

```
In [23] : =numTot=numObs+numMis;
```

```
In [24] : =smoothedPlot=
        ListPlot [Table [{n, yh [n]}, {n, numTot}] ,
              PlotStyle->{PointSize [.015] , RGBColor [ 0, 0, 1 ]},
              PlotJoined->True,
              DisplayFunction->Identity
        ] ;
```

```
In [25] : =forecastPlot=
        ListPlot [Table [{n, yh [n]}, {n, numTot+1, numTot+p}] ,
              PlotStyle->{PointSize [.015] , RGBColor [ 1, 0, 0 ]},
              DisplayFunction->Identity
        ] ;
```

```
In [26] : =Show [rawDataPlot, smoothedPlot, forecastPlot,
  DisplayFunction->$DisplayFunction] ;
```



The Mathematica Implementation

All Calculations

```
In [1] :=
Clear [year, month, yData,
      mean, meanYforSeasonalPeriod, groupByMonth,
      FullSetQ, next, prevSameMonth] ;

year [record_] :=record [[1]] ;
month [record_] :=record [[2]] ;
yData [record_] :=record [[3]] ;

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
meanYforSeasonalPeriod [data_, k_, per_:12] :=
      mean [yData/@Select [data, (k-1)per<month [#] <=k*per&]]

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

FullSetQ [data_, per_ : 12] :=
      Union [
        Mod [month [#] , per] &/@
            data/. 0 ->per
      ] ==Range [per]

next [previous_, exponent_, constant_] :=
      previous/(previous+(1 -constant)exponent)

prevSameMonth [k_] :=
      Max@
        Select [
          Join [
            {Mod [k-1, 12]+ 1 -p},
            month/@
              dataGroupedByMonth [[
                  Mod [k- 1 , 12]+ 1
              ]]
          ] ,
          (#<k&)
        ]
( *data normally read from from a disk file : * )

In [16] : =dataAll=
{{1991,  1 , 491}, {1991,  2 , 475}, {1991,  3 ,"M"}, {1991,  4 , 441},
    {1991,  5 , 439}, {1991,  6 , 425}, {1991,  7 ,"M"}, {1991,  8 , 434},
    {1991,  9 , 450}, {1991, 10, 466}, {1991, 11, 523},
    {1991, 12, 483}, {1992, 13, 494}, {1992, 14, 437},
    {1992, 15, 412}, {1992, 16, 403}, {1992, 17, 394},
    {1992, 18, 384}, {1992, 19, 389}, {1992, 20, 402},
    {1992, 21,"M"}, {1992, 22,"M"}, {1992, 23, 487},
    {1992, 24, 461}, {1993, 25,"M"}, {1993, 26,"M"},
    {1993, 27, 404}, {1993, 28, 433}, {1993, 29, 446},
    {1993, 30, 441}, {1993, 31, 435}, {1993, 32, 451},
    {1993, 33,"M"}, {1993, 34, 539}, {1993, 35, 585},
    {1993, 36,"M"}, {1994, 37, 595}, {1994, 38, 560},
    {1994, 39, 560}, {1994, 40,"M"}, {1994, 41, 598},
    {1994, 42, 580}, {1994, 43, 636}, {1994, 44, 655},
    {1994, 45, 685}, {1994, 46, 908}, {1994, 47, 864},
    {1994, 48, 832}};

In [17] : =
dataObs=Select [dataAll, # [[-1]] =!="M"&] ;
(*form : {{year, tp, y}, {year, tp, y}...}*)

dataMis=Select [dataAll, # [[-1]] =="M"&] ;
(*form : {{year, tp, "M"}, {year, tp, "M"}...}*)

tTimeDiffList=(# [[2]] -# [[1]] &)/@
          Partition [month/@dataObs, 2, 1] ;

In [22] : =
Clear [InitializeCTR]

InitializeCTR [data_?MatrixQ] : =
          InitializeCTR [data, {0.4, 0.1, 0.4}]

InitializeCTR [data_?MatrixQ, per_Integer] : =
          InitializeCTR [data, {0.4, 0.1, 0.4}, per]

InitializeCTR [dataObs_?MatrixQ,
          {alphaC_ : 0.4, gamma_ : 0.1, delta_ : 0.4},
          per_Integer : 12
] : =Module [{},
      p=per;
      numObs=Length [dataObs] ;
```

```
numMis=Length [dataMis] ;
a=alpha;
g=gamma;
d=delta;
Do [t [k] =(month/@dataObs) [[k]] , {k, numObs}] ;
Do [t [k] =k, {k, 1-p, 0}] ;
Do [y [t [k]] =(yData/@dataObs) [[k]] , {k, 1, numObs}] ;
Do [s [k] =(month/@dataMis) [[k]] , {k, numMis}] ;
dataGroupedByMonth=groupByMonth [dataObs] ;
k0=1+Quotient [-1+month@dataObs [[1]] , p] ;
k1=Module [{maxOfMins},
  maxOfMins=
    Max [month/@(First/@dataGroupedByMonth)] ;
    1+Quotient [-1+maxOfMins, p]
  ] ;
yMbyMonthGroup=
  mean/@(Map [yData, #] & [#] &/@dataGroupedByMonth);
yMMG [k_] :=yMMG [k] =yMbyMonthGroup [[k]] ;
yMSP [k_] :=yMSP [k] =
    meanYforSeasonalPeriod [dataObs, k, p] ;
Tr [0] =(yMSP [k1] -yMSP [k0])/((k1-k0)p);
Sl [0] =yMSP [k1] -(k0p-(p-1)/2)Tr [0] ;
Do [      (* CHANGE FROM ADDITIVE CASE *)
  Id [k-p] =
    mean [      (* DIVISION HERE *)
      y [month [#]] (**)/(**)(
            yMSP [1+Quotient [month [#] -1, p]]+
            Tr [0] ((-Mod [p-month [#] , p])+(p-1)/2)
      )&/@dataGroupedByMonth [[k]]
    ],
  {k, p}
] ;
(* Initialize q and Qi *)
```

```
      q=mean [# [[2]] -# [[1]] &/@
         Partition [month/@dataObs, 2, 1]] ;
      V [0] =1-(1-alpha)q;
      U [0] =1-(1-gamma)q;
      Do [
        Q [k-p] =Module [{mons},
          mons=month/@(dataGroupedByMonth [[k]]);
          (1+Quotient [Last [mons] , p])/Length [mons]//N
        ] ;
        W [k-p] =1-(1-delta)Q [k-p] ,
      {k, p}
      ] ;
    ]
```

```
In [27] : =
InitializeCTR [dataObs]
```

```
In [28] : =
Vlist=
    FoldList [
      next [#1, #2, a] &,
      V [0] ,
      tTimeDiffList
    ] ;
Ulist=
    FoldList [
      next [#1, #2, g] &,
      U [0] ,
      tTimeDiffList
    ] ;
Wlist=
    Table [
      Module [{mons, WSet},
        mons=Join [
          {k-p},
          month/@
            dataGroupedByMonth [[k]]
        ] ;
        WSet [k] =
          FoldList [
            next [#1, #2/p, d] &,
            W [k-p] ,
            (# [[2]] -# [[1]] &)/@
                Partition [mons, 2, 1]
          ] ;
        Transpose [{mons, WSet [k]}]
      ] ,
      {k, p}
    ]//(Sort [Flatten [#, 1] , (#1  [[1]] <#2  [[1]] &)] &);

Do [
    V [t [k]] =Vlist [[k]] ;
    U [t [k]] =Ulist [[k]] ,
    {k, numObs}
] ;

(* We can be slicker with the W's, since they are indexed *)
(W [# [[1]]] =# [[2]])&/@Wlist;

Do [(
    SI [t [n]] =
      (V [t [n]] y [t [n]]/Id [prevSameMonth [t [n]]])+
        (1-V [t [n]])(SI [t [n-1]]+(t [n] -t [n-1])Tr [t [n-1]]);
    Tr [t [n]] =U [t [n]] *
          (SI [t [n]] -SI [t [n-1]])/(t [n] -t [n-1])+
                                      (1-U [t [n]])Tr [t [n-1]] ;
    Id [t [n]] =W [t [n]] *
            y [t [n]]/SI [t [n]]+
              (1-W [t [n]])Id [prevSameMonth [t [n]]] ;
    yh [t [n]] =Sl [t [n]] Id [t [n]]
    ),
    {n, numObs}
]
```

Finally Extrapolations

In [36] :=

```
yh [n_/;((!MemberQ [month/@dataObs, n])&&n> 0 )] :=
    yh [n] =
      With [{tn=Max@Select [month/@dataObs, #<n&]},
        (Sl [tn]+(n-tn)Tr [tn]) *
                          Id [prevSameMonth [n]]
      ]
```

Result

```
yHats=Table [yh [k] , {k, numMis+numObs}] ;

dataSmoothedTableau=
    Append [dataAll//Transpose, yHats]//Transpose;

forecastTableau=
    Table [
      { 1 +dataAll [[- 1 , 1 ]] ,
        k,
        "ForeCast",
        yh [k]
      },
      {k, numMis+numObs+ 1 , numMis+numObs+13}
    ] ;

TableForm [
    Join [dataSmoothedTableau, forecastTableau] ,
      TableHeadings->{None, {"Year", "t", "y", "y-hat"}},
  TableSpacing->{0, 3}
]/."M"->""/."ForeCast"->"FC"
```

```
Year  t   y    y-hat
1991  1   491  476.75
1991  2   475  462.497
1991  3        422.047
1991  4   441  444.223
1991  5   439  440.592
1991  6   425  428.213
1991  7        444.36
1991  8   434  441.08
1991  9   450  453.831
1991  10  466  488.937
1991  11  523  522.153
1991  12  483  482.987
1992  13  494  470.158
1992  14  437  431.365
1992  15  412  405.884
1992  16  403  410.336
1992  17  394  400.659
1992  18  384  388.164
1992  19  389  392.676
1992  20  402  403.019
1992  21       419.781
1992  22       480.448
1992  23  487  492.824
1992  24  461  460.971
1993  25       421.407
1993  26       378.693
1993  27  404  395.21
1993  28  433  423.164
1993  29  446  436.656
```

1993	30	441	436.233
1993	31	435	439.675
1993	32	451	455.461
1993	33		483.825
1993	34	539	543.554
1993	35	585	586.835
1993	36		557.223
1994	37	595	579.42
1994	38	560	549.025
1994	39	560	545.078
1994	40		555.899
1994	41	598	588.642
1994	42	580	580.021
1994	43	636	624.741
1994	44	655	654.606
1994	45	685	688.518
1994	46	908	878.04
1994	47	864	890.921
1994	48	832	843.816
1995	49	FC	832.445
1995	50	FC	761.294
1995	51	FC	726.55
1995	52	FC	739.467
1995	53	FC	755.476
1995	54	FC	741.86
1995	55	FC	775.188
1995	56	FC	806.755
1995	57	FC	853.808
1995	58	FC	1033.04
1995	59	FC	1073.92
1995	60	FC	1032.71
1995	61	FC	1015.39

References

Box, E.P.B., et. al. Time Series Analysis : forecasting and control, 3 rd 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

Wilson, J H. and Barry Keating, Business Forecasting, 2 nd 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.