

```
#####  
#                                                                 #  
# BAYESIAN UPDATING OF GERMAN AGRICULTURAL YIELD EXPECTATIONS #  
#                               Jette Krause                    #  
#                               22.01.2008                     #  
#                                                                 #  
#                                                                 #  
#####
```

...

This is a programme carrying out Bayesian updating of German agricultural yield expectations. Its aim is to derive second order probabilities (SOP) for a set of predefined hypotheses on agricultural yield development.

In the present version, the programme processes data on German cereal, winter wheat and corn yields per unit area from 1950 to 2004. Updating proceeds chronologically from year to year. Results depict how an agent who gets to know more and more data adjusts her initial expectations. The adjustment of expectations includes changes in SOP as well as adaptation of the hypotheses to the new set of background knowledge in every updating cycle. Background knowledge consists of all yield data that is known from periods prior to the respective updating cycle, here called old data.

It is assumed that agricultural yields can be explained by a linear trend function with normally distributed variance. Developments investigated by this programme concern changes in the slope of the trend function and in variance. A set of hypotheses is specified through different hypotheses on changes in these two parameters. The programme adapts the SOP of models in the light of new yield data which are treated as becoming known subsequently. Initial prior SOP are assumed to be distributed uniformly over all models.

The general structure of an updating cycle is as follows:

- 1) From old data, parameters of a linear trend function (slope and axis intercept) as well as variance are estimated by regression analysis.
- 2) From values of slope and variance calculated in the first step and the set of hypotheses on possible changes in these parameters, a set of probability distributions on yields in the following period is determined. These models vary in regard to the expected value of yields and their variance.
- 3) By application of Bayes' Rule, SOP of models are adapted to new data. This procedure is called "updating". From the prior SOP of models and the likelihood of new yield data within each model, which is computed from the respective probability distribution of yields, posterior SOP of models are generated.

This process is repeated in a loop. For each subsequent updating cycle, the posterior SOP of the previous cycle is used as prior SOP for the next one. Updating stops when the last data value available, i.e., 2006 yields, has been processed. The final resulting posterior SOP along with the models specified in the last cycle describe yield expectations for the future. Probability distributions on yields and their posterior SOP over all updating cycles illustrate the learning process the agent has gone through, that is, the stepwise adaptation of her expectations to the data.

...

```
from math import sqrt
try: from transcendental import erf
except: pass
import Numeric
import Gnuplot

#-----
'''
Inputs: Yield Data

German agricultural yield data is used as an input to the programme.
1950-2006 data is used as published by Statistisches Bundesamt (StatBA).
All data is given in Hg/Ha, that is 100g per hectare.
'''

StatBA_Cereals=[22200, 25900, 24700, 24200, 24700, 25600, 25800, 26200, \
26100, 27500, 30300, 24200, 29600, 29300, 31200, 28500, 28800, 34800, \
36200, 34500, 31800, 37900, 37700, 38700, 41800, 38600, 35000, 38900, \
42900, 41100, 42300, 41600, 45700, 43600, 50700, 50700, 50900, 48900, \
51700, 52000, 54100, 59900, 53400, 57100, 58300, 61100, 62800, 64900, \
63300, 67000, 64600, 70600, 62500, 57700, 73600, 67300, 64900]
StatBA_Cereals_descr="Data: German Cereal Yields 1950-2006 in Hg/Ha, \
Sources: StatBA, Data for Germany has been calculated ex post"

StatBA_WiWheat=[25900, 30000, 28600, 27600, 25800, 29400, 30200, 31800, \
29200, 33700, 35700, 29100, 34500, 34500, 35400, 32600, 32800, 40900, \
42700, 39400, 37700, 44600, 40600, 43800, 46700, 43700, 39800, 43800, \
49700, 48200, 47900, 49400, 53500, 52700, 59700, 58700, 61500, 58200, \
62700, 57200, 63000, 68200, 60300, 66200, 68200, 69200, 73200, 73400, \
72400, 76400, 73200, 79200, 69400, 65500, 82100, 75100, 72400]
StatBA_WiWheat_descr="Data: German Winter Wheat Yields 1950-2006 in Hg/Ha, \
Sources: StatBA, Data for Germany has been calculated ex post"

StatBA_Corn=[25000, 28200, 22600, 28100, 26900, 26100, 26600, 26200, \
26700, 26100, 29400, 29000, 32300, 34500, 34600, 35900, 40100, 47000, \
48900, 48800, 49700, 50400, 46600, 53400, 48400, 55200, 46800, 58100, \
52900, 64100, 56400, 64400, 65700, 55200, 56500, 66600, 69400, 62800, \
76400, 74900, 68100, 68800, 72600, 80500, 71100, 74600, 78600, 87200, \
82600, 88400, 92800, 88900, 93900, 74700, 91300, 92700, 80700]
StatBA_Corn_descr="Data: German Corn Yields 1950-2006 in Hg/Ha, \
Sources: StatBA, Data for Germany has been calculated ex post"

Cer_ti="Updating Results Using Cereal Yields 1950-2006"
WiWheat_ti="Updating Results Using Winter Wheat Yields 1950-2006"
Corn_ti="Updating Results Using Corn Yields 1950-2006"

#-----
'''
Hypotheses

Different hypotheses on the changes in slope of the regression line
and variance of data from one period to the next are considered.
h_s lists contain factors the slope is multiplied by, and
h_v lists contain factors the variance is multiplied by.
'''
```

```

h_s_1=[1,2,.5]
h_s_2=[.5,.8,1,1.1,1.3,1.5,1.8,2]
h_s_3=[1,1.2,0.8]
h_s_4=[1,1.05,0.95]

```

```

h_v_1=[1,2,.5]
h_v_2=[.5,.8,1,1.1,1.3,1.5,1.8,2]
h_v_3=[1,1.2,0.8]
h_v_4=[1,1.05,0.95]

```

```

#-----
...

```

(1) Estimation of trend model parameters

The following definitions are used to determine the parameters of the trend line (axis intercept and slope) according to the method of least squares, as well as the variance from old data.

Abbreviations*:

L_act - list of old yield data known before updating
t - list containing all points in time before current updating
a - axis intercept of the trend line
b - slope of the trend line

*Only abbreviations which will reoccur in other parts of the programme are given here. All others are explained in the respective parts or definitions.

```

...

def mean(val):
    ...
    mean calculates the arithmetic mean of a list of values.
    val - list of values
    ...
    sum=0
    for i in range(len(val)):
        sum+=val[i]
    return float(sum)/float(len(val))

```

```

def den(L_act,t):
    ...
    den calculates the denominator of the equation for computing the
    slope of the regression line.
    ...
    sum=0
    for i in range(len(L_act)):
        sum+=(t[i]-mean(t))**2
    return sum

```

```

def num(L_act,t):
    ...
    num calculates the numerator of the equation for computing the slope
    of the regression line.
    ...
    sum=0
    for i in range(len(L_act)):
        sum+=(t[i]-mean(t))*(L_act[i]-mean(L_act))
    return sum

```

```
def act_regr(L_act,t):
    """
    act_regr calculates the parameters of the trend function from yield
    data values contained in list L_act and time values contained in
    list t.
    b - slope of the trend line
    a - axis intercept
    """
    b=float(num(L_act,t))/float(den(L_act,t))
    a=float(mean(L_act)) - float(b*mean(t))
    global a
    global b
    return a,b
```

```
def act_var(L_act,t):
    """
    act_var calculates the variance of data, i.e., the mean squared
    deviation of measured yield values from the regression line.
    y_est - yields at time t[i] as estimated from the fitted regression
            function
    """
    sum=0
    for i in range(len(L_act)):
        y_est=a+b*t[i]
        sum+=(L_act[i]-y_est)**2
    return float(sum)/float(len(L_act)-2)
```

```
#-----
```

(2) Specification of hypotheses

Departing from parameter values computed and assumptions on changes in them, the following definitions determine different probability distributions on future yield data. Since these distributions are normal distributions, they are specified by their respective mean (which corresponds to the expected value of a distribution) and variance.

Abbreviations:

h_s - list of hypotheses on changes in slope
h_v - list of hypotheses on changes in variance

```
"""
```

```
def time(ev):
    """
    time assigns a point in time [0,1,2,...] to each event contained
    in a list.
    ev - list of events
    """
    t=[]
    k=0
    for i in range(len(ev)):
        t.append(k)
        k+=1
    return t
```

```
def hyp_exp(L_act,t,h_s):
    """
    hyp_exp creates a list of hypothetical (predicted) expected values
    of yields for the following period. It uses trend line parameters
    calculated from old data. Different expected values of yields are
    calculated by extrapolating the trend line with the slope changed
    as suggested by the different hypotheses on changes in slope,
    successively.
    m - list which gets successively filled with the expected values
        calculated
    """
    act_regr(L_act,t)
    m=[]
    for i in range(len(h_s)):
        m.append(a+b*(h_s[i])*(len(t)))
    return m

def hyp_var(L_act,t,h_v):
    """
    hyp_var generates a list of hypothetical values of variance in the
    period to come. It takes the value of variance calculated from old
    data and successively applies hypotheses on possible changes to it.
    var - value of variance calculated from old data
    v - list which gets successively filled with the hypothetical
        values of variance
    """
    var=act_var(L_act,t)
    v=[]
    for i in range(len(h_v)):
        v.append(var*(h_v[i]))
    return v

def models(L_act,h_s,h_v):
    """
    models generates a list of tuples [expected value; variance],
    where each tuple gives the parameters of a different normally
    distributed probability density function on yields in the
    upcoming period. This is done by calling the two lists containing
    hypothetical expected values and hypothetical values of variance
    and then combining each element of the first list with each one
    of the second.
    h_exp - list of hypothetical expected values for the upcoming
        period
    h_var - list of hypothetical values of variance for the upcoming
        period
    ND - list in which two parameters [expected value; variance] are
        assembled
    M - list where all lists of parameter combinations
        [expected value; variance] are attached, successively
    """
    t=time(L_act)
    h_exp=hyp_exp(L_act,t,h_s)
    h_var=hyp_var(L_act,t,h_v)
    M=[]
    ND=[]
    for i in range(len(h_exp)):
        for j in range(len(h_var)):
            ND.append(h_exp[i])
```

```

        ND.append(h_var[j])
        M.append(ND)
        ND=[]
    return M

```

```

#-----

```

```

'''

```

(3) Bayesian Updating of SOP

The following definitions carry out the updating process according to Bayes' Rule.

```

'''

```

```

def display_1(years,i,new_data,L_descr):

```

```

'''

```

display_1 produces a header for the output of each updating cycle. The header contains a description of data processed, gives the point in time assigned by the programme as well as the year and the values of new data to be processed in that cycle. Furthermore, it prints column headlines for the values that will be given.

```

'''

```

```

print L_descr
print"year",years[i+1],"/",years[i+1]+1950,"      ", \
"yield", new_data,"\\\"
print "mean", "      ", "variance", "      ", \
"likelihood", "      ", "posterior sop","\\\"
return

```

```

def display_2(mean, var,li,sop):

```

```

'''

```

display_2 hands the values of mean, variance, likelihood and posterior SOP calculated in an updating cycle to the output window.

```

'''

```

```

print round(mean,2), " & ",round(var,2)," & ", round(li,10), \
" & ", round(sop,10),"\\\"
return

```

```

def tot_prob(lik,sop_old):

```

```

'''

```

tot_prob computes the total probability of yield data, which is the likelihood of a model multiplied with its prior SOP, summed up over all models.

```

'''

```

```

sum=0
for i in range(len(sop_old)):
    sum+=lik[i]*sop_old[i]
return sum

```

```

def lik(models,x):

```

```

'''

```

lik calculates the likelihood of data x within each model from the respective probability density function (pdf). As each single value of x has zero probability within the pdf of a normally distributed variable, probabilities are calculated for x-values within a small interval around x, specified by its lower and upper limits [x_ll;x_ul]. Probabilities are computed as the integral over the pdf within this interval. Integrating over a pdf, values of the cumulative distribution function (cdf) are obtained. cdf-values for normally

```

distributed variables have to be computed numerically. For
implementation, the error function can be used to calculate values
of the cdf of a standard normally distributed variable. To use this
relation, x-values first have to be linearly transformed into
standardized z-values. Then, likelihood of data is calculated as
the probability of z-values to lie within the interval [z_ll;z_ul].
It is computed from the cdf of a standard normally distributed
variable implemented via the error function.
likely - list to which the likelihood of different models is appended,
        subsequently
x_ul - upper limit of integration
x_ll - lower limit of integration
z_ul - standardized upper limit of integration
z_ll - standardized lower limit of integration
fop - likelihood (or first order probability) of models, subsequently
      calculated for different z-values
erf - error function, used to calculate values of the cdf for a
      standard normally distributed variable
...
likely=[]
x_ul=x+0.0005
x_ll=x-0.0005
for i in range(len(models)):
    z_ul=(float(x_ul-models[i][0])/float(sqrt(models[i][1])))
    z_ll=(float(x_ll-models[i][0])/float(sqrt(models[i][1])))
    fop=(erf(z_ul/sqrt(2))+1)/2 - (erf(z_ll/sqrt(2))+1)/2
    likely.append(fop)
return likely

```

```

def sop_start(N):
    """
    sop_start determines a set of uniformly distributed initial prior SOP.
    sop - list where initial prior SOP of all models are subsequently
          appended
    """
    sop=[]
    for i in range(N):
        sop.append(float(1.0)/float(N))
    return sop

```

```

def update(L,L_descr,h_s,h_v,ti,saveplotas):
    """
    update coordinates the process of subsequent steps of Bayesian
    updating. It calls all other definitions, directly or indirectly.
    First, the number of models is determined from the number of
    hypotheses, and the initial prior SOP of models is calculated. A
    point in time is assigned to each annual yield data. Then, there
    is an outer loop over updating cycles. Yield data from the original
    data list L are appended one by one to an initially empty list
    called data. This list contains all data that are known from
    before the current updating cycle. When data contains three or
    more values, the following procedure takes place: The next data in
    list L is declared new data. This is the data to learn from in the
    current updating cycle. A set of models is created using old data
    and a set of hypotheses. The Likelihood of these models in regard
    to new data is calculated, as well as the total probability of data
    over all models. Then, there is an inner loop over all models. In
    this loop, prior SOP are updated which results in posterior SOP of
    all models. When this process has finished, posterior SOP of the

```

updating cycle just gone through become prior SOP for the next updating cycle. The programme continues from the beginning of the outer loop. It carries out subsequent updating cycles until the last data value contained in list L has been used as new data. Then, the process stops. Its results consist of posterior SOP of models along with their parameter values at all points in time during the updating cycle, which have been stepwise displayed as outputs.

```

L - list of all yield data available (from 1950 to 2004)
L_descr - description of yield data
N - number of models
h_v - set of hypotheses on changes in variance
h_s - set of hypotheses on changes in the slope of the trend line
sop_old - prior SOP
years - list of points in time assigned to the occurrence of yield
      data
data - list which is fed with old yield data step by step
new_data - data value to learn from in the current updating cycle
sop_new - posterior SOP
M_new - list of models for the current updating period
li - list of likelihood of all models
tp - total probability of current new data
prob - SOP of each model, subsequently
...

N=(len(h_v))*(len(h_s))
sop_old=sop_start(N)
years=time(L)
data=[]
sopdev=[]
sopdev.append(sop_old)
for i in range(len(L)-1):
    data.append(L[i])
    if i>=2:
        new_data=L[i+1]
        display_1(years,i,new_data,L_descr)
        M_new=models(data,h_s,h_v)
        li=lik(M_new,new_data)
        tp=tot_prob(li,sop_old)
        sop_new=[]
        for j in range(N):
            prob=float(li[j])*float(sop_old[j])/float(tp)
            sop_new.append(prob)
            display_2(M_new[j][0],M_new[j][1],li[j],prob)
        print "//"
        sopdev.append(sop_new)
    sop_old=sop_new
epsplot(sopdev,"sop",1952,ti,saveplotas)
return

```

```

#-----
...
Plotting the development of SOP over time
...
#-----

```

```

def hyptuples(sop_time):
    #rearranging the data list: forming "hypothesis-tuples":
    y=[]
    for i in range(len(sop_time[0])):

```

```

    sop_model=[]
    for j in range(len(sop_time)):
        sop_model.append(sop_time[j][i])
    y.append(sop_model)
return y

```

```

#-----
def epsplot(datalist,descr,startyear,ti,saveas):
    year=[]
    for i in range(len(datalist)):
        year.append(i+startyear)

#general settings for plotting:
g=Gnuplot.Gnuplot()
g('set terminal postscript solid linewidth 2.0')
g('set data style linespoints')
g('set output \"plot_\"+saveas+'.eps\"')
if descr=="sop":
    y=hyptuples(datalist)
    g.title(ti)
    g.xlabel('Year')
    g.ylabel('Second Order Probability')
    data = []
    for i in range(len(y)):
        model=zip(year,y[i])
        data.append(Gnuplot.Data(model,title='Hypothesis no. '+str(i+1)))
    g.plot(*tuple(data))
    z=0
    for i in range(2000000):
        z+=1
return

```

```

#-----
'''
Call of the updating process

```

The programme is started here, handing in the following inputs:

- the list of data to be used, e.g. 'StatBA_Cereals'
- a description of data, e.g. 'StatBA_Cereals_descr'
- a set of hypotheses on changes in the slope of the trend line, e.g. 'h_s_3' and,
- a set of hypotheses on the changes in variance, e.g. 'h_v_3'.

Depending on the choice of data and hypotheses used as inputs, the programme generates different outputs.

```

'''
update(StatBA_Cereals,StatBA_Cereals_descr,h_s_3,h_v_3,Cer_ti,"Cer33")
update(StatBA_WiWheat,StatBA_WiWheat_descr,h_s_3,h_v_3,WiWheat_ti,"WiWheat33")
update(StatBA_Corn,StatBA_Corn_descr,h_s_3,h_v_3,Corn_ti,"Corn33")

```

```

#-----

```