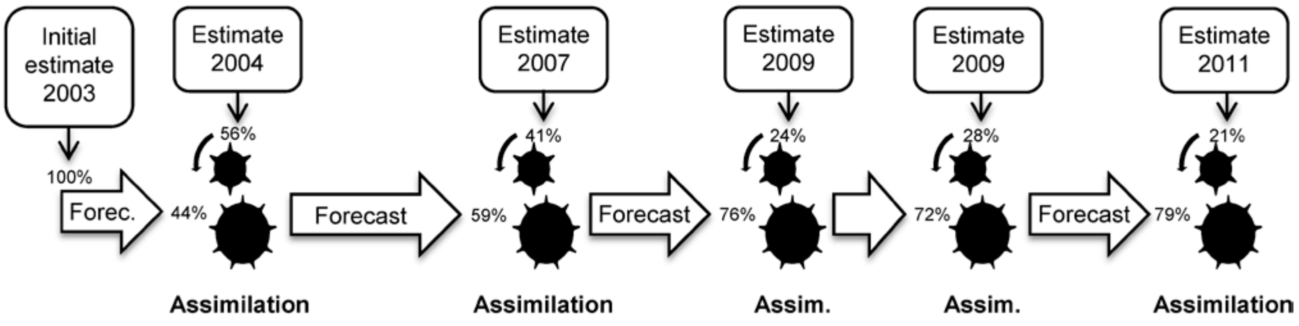
Data assimilation – A prototype system to assimilate forest stand information



Mattias Nyström, Anton Grafström, Nils Lindgren,
Håkan Olsson och Göran Ståhl

Working report 448 2016

Swedish University of Agricultural Sciences
Department of Forest Resource Management
901 83 UMEÅ
www.slu.se/srh
090/7868100

ISSN 1401–1204
ISRN SLU–SRG–AR–448–SE

SP
Certifierad
ISO 14001

Data assimilation – A prototype system to assimilate forest stand information

Mattias Nyström, Anton Grafström, Nils Lindgren,
Håkan Olsson och Göran Ståhl

Keywords: data assimilation, extended Kalman filter, forestry, forest inventory, forest data stand

Working Report 448 2016

Swedish University of Agricultural Sciences
Department of Forest Resource Management
Place of publication: Umeå
Year of publication: 2016

ISSN 1401–1204
ISRN SLU–SRG–AR–448–SE

Preface

This report is a technical documentation of a prototype program for data assimilation of forest stand data. The program is developed as part of a project (1314-130/165-9) funded by the Forest Society of Sweden (Skogssällskapet) with Professor Göran Ståhl as principal investigator. The de-veloped computer program and the studies made with it will be a foundation for further research about data assimilation of forest data, for which additional funding has been granted also from the Swedish National Space Board and Formas.

Umeå, Sweden, February 2016

Cover photo: Illustration of the data assimilation procedure supported in the prototype of the program.

Table of contents

Summary

The purpose of this report is to describe a data assimilation prototype program (Appendix A) developed to estimate forest stand data. The program was developed and tested on data collected on the forest estate Remningstorp in southern Sweden. Data assimilation can be used to sequentially combine remote sensing based estimates of forest variables with predictions from growth models. The assimilation routine implemented was the extended Kalman Filter.

The program supports two different ways to assimilate the forest data: (1) pixel-wise and (2) stand-wise. In the pixel-wise way, raster cells are used as assimilation unit and can be aggregated to a stand for evaluation. In the stand-wise way, the whole stand is assimilated as one unit. The two methods has pros and cons. The pixel-wise way is simple to use as no stand-delineation is needed, but might be subject to boundary effects and noise due to geometric errors. Using the developed code, it has been shown in three case studies that the combination of time series of remote sensing data and forest growth functions provides better estimates of forest variables than only using forecasting, or only using the latest remote sensing data. This opens up for a new way to keep forest stand registers up to date.

1 Introduction

Data assimilation is a technique that offers great potential for combining all new sources of data of relevance for forest estimates (Ehlers *et al.*, 2013; Nyström *et al.*, 2015). Existing information about a forest area is forecasted using a model that provides an estimate for the date of the next data acquisition and an estimate of the precision of the forecasted information. Thus, the precision of the forecasted information can be compared with the precision of the new information. In the assimilation step, the two sources of information are combined through weights that are inversely proportional to their uncertainties (Figure 1). The combined estimate is then forecasted to the time-point of the next data acquisition, and repeated when new data become available.

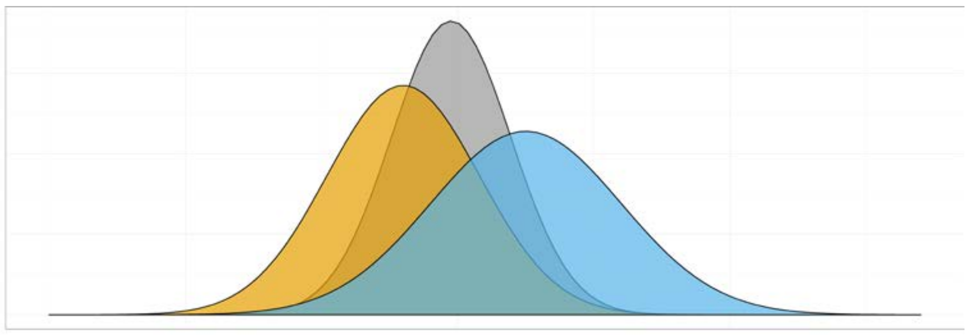


Figure 1. An illustration of the basic principle of data assimilation applied to a Gaussian model and estimate. The figure shows how the prior forecast (the center of the orange distribution) is updated to the posterior forecast (the center of the grey distribution) when a new estimate (the center of the blue distribution) is taken into account. Notice that the grey distribution is narrower than the orange distribution, indicating that the posterior forecast is more precise (i.e., the estimate has lower variance) than the prior forecast.

This report describes a data assimilation prototype program, essentially a code in the programming language R (R Core Team, 2015). The code was developed and tested using remote sensing and field inventory data from a forest estate in Remningstorp in southern Sweden.

2 Methods

2.1 Methods for representing data for any stand shape

The data assimilation prototype (Appendix A) is developed to support two different ways to assimilate forest stand data. The first is by dividing the stand into raster cells (Figure 2), i.e. cells with a fixed size, for example 18 m × 18 m. To evaluate the assimilation, a mean value of the assimilated cells is calculated within a defined unit, for example a stand (green polygon in Figure 2) or a circular sample plot (red circle in Figure 2). When calculating the mean value, the raster cells with its center point within the polygon are included. For example, the light green cells are included when calculating the mean for the green polygon and the light red cells are included in the mean of the red circle.

The second way is to assimilate the whole stand as one unit. That means that the whole green polygon will be assimilated at once instead of first splitting into smaller areas. The polygon can then have any shape and only the area covered by the polygon will be included in the assimilation. The stand-wise method is debated as the area used to develop the remote sensing predictions as well as the growth models are often developed from sample plots with a radius of about 10 m, which are arguments for applying the assimilation routines on areas approximately the same size. On the other hand, the pixel-wise approach might be sensitive to geometrical errors in the used data sets.

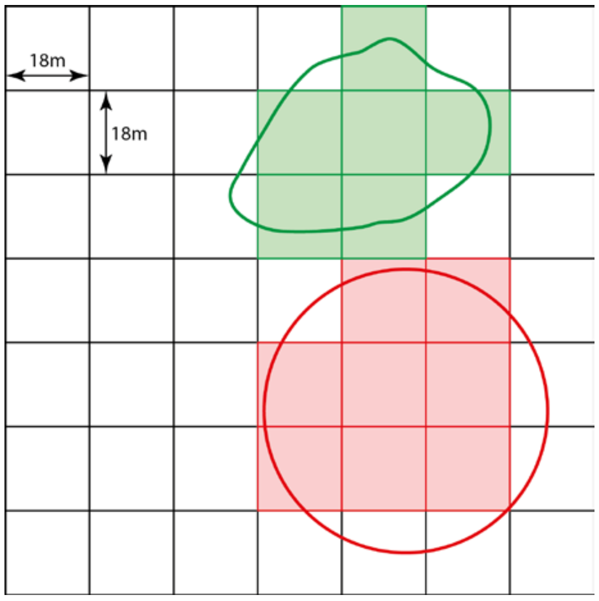


Figure 2. Illustration of the pixel-wise approach. The forest landscape in the initial case studies were divided into 18 m × 18 m raster cells and each cell was used as an assimilation unit. For evaluation, the raster cells within a stand or a sample plot were aggregated. Two examples of aggregation are given in the figure (green polygon and red circle) where the cells included in the aggregation are highlighted. The aggregation could simply be a mean value of the cells with its center point within the polygon.

2.2 Methods for combining growth data and remote sensing estimates

The assimilation routine is the same for the two ways except that the first method (pixel-wise) requires a mean value to be calculated for the stand or sample plot when evaluating. Figure 3 shows an example of the data assimilation processing chain using the extended Kalman filter.

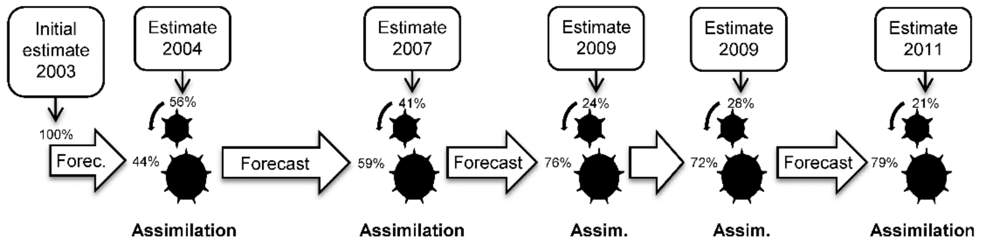


Figure 3. Illustration of the data assimilation procedure supported in the prototype. The years state the growth season for each data acquisition in the first study with empirical data (Nyström *et al.*, 2015). The percentage numbers are examples of the Kalman Gains, i.e.., the amount of information that is included from each source when assimilating. The Kalman Gain value is different for each assimilation unit. In this example, there were two acquisitions with growth season 2009 and therefore no forecast of the growth was needed between these two.

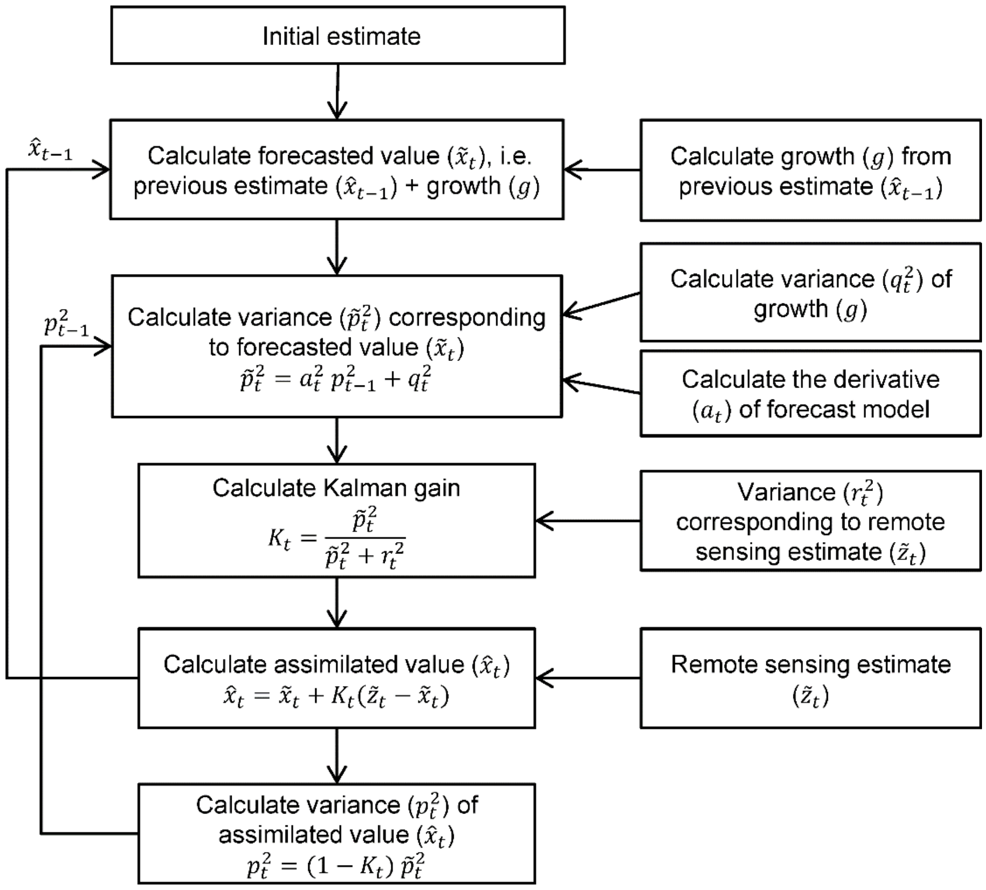
Figure 4 gives an overview how extended Kalman filtering (Welch & Bishop, 2006) was implemented to assimilate the target variables over time using remote sensing data for adjusting the development given by the growth model. The following sections describe the data assimilation in more detail. The notations are similar as in the R programming code (Appendix A).

The development over time of the target variable can be formulated as:

$$X_t = f(x_{t-1}, W_t, t-1) = x_{t-1} + g(x_{t-1}, t-1) + W_t \tag{1}$$

where the random variable X_t denotes the state of the target variable at time t. The model describes the conditional distribution of X_t, given that the observed value of the variable at time $t-1$ was x_{t-1}. Thus, the forecasted value at time t was given by the previous value, to which the expected growth $g(x_{t-1}, t-1)$ and a random term (W_t) were added.

In this prototype, the growth models were developed to estimate the growth for a five year period; predictions for any shorter periods were obtained through computing a share corresponding to the share of the length of the prediction period. The variance of the predictions was estimated from a separate linear regression model as it varied with the growth. The random term W_t was assumed to be normally distributed with zero mean and a variance q_t^2 that is dependent on the previous state as well as on time; $q_t = \alpha + \beta \cdot g(x_{t-1}, t-1)$, where $t \geq 1$ and α and β are parameters used for estimating the standard deviation of the residual errors. The parameters are different for each target variable and for different forest type classes.



Initial estimate
$\hat{x}_{t-1}$
Calculate forecasted value ($\tilde{x}_t$), i.e. previous estimate ($\hat{x}_{t-1}$) + growth (g)
Calculate growth (g) from previous estimate ($\hat{x}_{t-1}$)
p_{t-1}^2
Calculate variance ($\tilde{p}_t^2$) corresponding to forecasted value ($\tilde{x}_t$) $\tilde{p}_t^2 = a_t^2\, p_{t-1}^2 + q_t^2$
Calculate variance (q_t^2) of growth (g)
Calculate the derivative (a_t) of forecast model
Calculate Kalman gain $K_t = \frac{\tilde{p}_t^2}{\tilde{p}_t^2 + r_t^2}$
Variance (r_t^2) corresponding to remote sensing estimate ($\tilde{z}_t$)
Calculate assimilated value ($\hat{x}_t$) $\hat{x}_t = \tilde{x}_t + K_t(\tilde{z}_t - \tilde{x}_t)$
Remote sensing estimate ($\tilde{z}_t$)
Calculate variance (p_t^2) of assimilated value ($\hat{x}_t$) $p_t^2 = (1 - K_t)\, \tilde{p}_t^2$

Figure 4. Flowchart of the implementation of the data assimilation framework.

The deviations between the state of the target variable and estimates of it – based on remotely sensed data – were assumed to be independent normally distributed random variables. These normal distributions were assumed to have zero mean and time-dependent variances according to

$$Z_t = x_t + V_t, \tag{2}$$

where Z_t is the (random) estimator, x_t is the true state at time-point t, and V_t is a random deviation with zero mean and variance, r_t^2, which is estimated from the residual errors obtained in connection with the development of estimators for the target variables based on the remotely sensed data.

$\tilde{z}_t$ is used as notation for the actual value observed of Z_t. Similarly, $\tilde{x}_t$ is used as notation for the actual value obtained through the growth predictions, i.e., $\tilde{x}_t = f(\hat{x}_{t-1}, 0, t-1)$, where $\hat{x}_{t-1}$ is the assimilated variable at time-point $t-1$; its variance is denoted p_{t-1}^2. In case no estimates are made at time-point t, the assimilated variable $\hat{x}_t$ will obtain the value $\tilde{x}_t$ and the variance $\tilde{p}_t^2 = a_t^2 p_{t-1}^2 + q_t^2$. This is a consequence of the use of first order Taylor linearization, in connection with the extended Kalman filter, to linearize the growth model in order to compute the variance. Thus, $a_t = \frac{d}{dx} f(\hat{x}_{t-1}, 0, t-1)$, or in other words, a_t is the partial derivative of the growth model with respect to the target variable.

In case an estimate is made at time-point t, the forecast and the estimate are weighted inversely proportional to the variances to obtain the assimilated variable, i.e.

$$\hat{x}_t = (1 - K_t)\tilde{x}_t + K_t \tilde{z}_t \tag{3}$$

with the Kalman gain $K_t = \frac{\tilde{p}_t^2}{\tilde{p}_t^2 + r_t^2}$. If the estimator (Eq. 2) has considerable variance then the Kalman gain becomes almost zero, implying that the estimate does not contribute to the assimilation. On the other hand, if the estimator is very precise, then the Kalman gain becomes almost 1, implying that the forecast does not contribute much to the assimilation. Eq. 3 is based on the assumption that $\tilde{x}_t$ and $\tilde{z}_t$ are independent. Further, the variance of the assimilated variable is

$$p_t^2 = (1 - K_t)\tilde{p}_t^2. \tag{4}$$

3 Results and discussion

The data assimilation prototype program (Appendix A) was successfully used in three case studies which have been presented at four international conferences (ForestSat 2014, IBFRA 2015, EARSeL 2015, SilviLaser 2015): The first study, where canopy height data from a time series of matched aerial images were used has also been published in a scientific journal (Nyström *et al.*, 2015). In the two not yet published studies, a series of airborne laser scanning data, and a series of canopy height from interferometric SAR, respectively, have been used. In future studies multiple remote sensing data sources will be mixed which will make use each sensors' strength.

In Nyström *et al.* (2015), the pixel-wise way of assimilating was used and evaluation was done on 40 m radius sample plots. In the two ongoing studies, the assimilation is done using the stand-wise method at 10 m radius sample plots. In the case studies, pros and cons were found with either method of choosing assimilation units. Pros and cons with the pixel-wise assimilation method:

- The geographic unit remain consistent over time.
- Boundary pixels will not resolve the exact polygon area and will include some of the surrounding areas. The problem will be smaller for larger stands though.
- Geometric errors in the sensor data might cause the remote sensing data to be noisy on the pixel level.

Pros and cons with the stand-wise assimilation method:

- The exact polygon is included which reduces the boundary effect.
- The stand boundaries may however change which might require the assimilation to restart; alternatively this problem might be resolved by implementing a merging/splitting functionality.
- The metrics calculated from the remote sensing data will be calculated on an area different to the area where the model was developed. How this affects the prediction from the remote sensing data need to be furthered studied.

It should be clarified that this prototype program was developed for research purposes and further development need to be done before data assimilation can be applied more operationally. The work with this prototype program has found several issues that need to be addressed in future work. These issues are, among other, the effects on the assimilation from: temporal correlation, spatial correlation and abrupt changes, for example windthrown trees.

References

Ehlers, S., Grafström, A., Nyström, K., Olsson, H. & Ståhl, G. (2013). Data assimilation in stand-level forest inventories. *Canadian Journal of Forest Research* 43(12), 1104–1113.

Nyström, M., Lindgren, N., Wallerman, J., Grafström, A., Muszta, A., Nyström, K., Bohlin, J., Willén, E., Fransson, J. E. S. & Ehlers, S. (2015). Data assimilation in forest inventory: first empirical results. *Forests* 6(12), 4540–4557.

R Core Team (2015). R: A Language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing. Available from: https://www.r-project.org/.

Welch, G. & Bishop, G. (2006). *An introduction to the Kalman filter* [online]. *TR 95-041*. Chapel Hill, NC, USA.

Appendix A

In this appendix a copy of the source code used in the data assimilation prototype using the extended Kalman Filter is supplied. The main-script (main.R) loads two other files, functions.R and growthModels.R. The file functions.R contains functions used in main.R. In growthModels.R there are growth models developed for stem volume, basal area, and Lorey's mean height. These models were specifically developed for the case study where the programming code was tested (Nyström *et al.*, 2015).

main.R

```
rm(list = ls())

# Load used libraries
require(ggplot2)
require(grid)
require(gridExtra)
require(reshape)
require(dplyr)

# Load R-functions from files
source("functions.R")
source("growthModels.R")

########################################################################################
# Parameters
########################################################################################
# Set to true if you want to assimilate for each cell within a larger plot, e.g. 40m radius.
# The cells will be aggregated in the end for each plot.
useCells <- FALSE

# set TRUE if a model for variance is to be used
useFnPredVar <- FALSE

# Set the year when we want to evaluate the assimilation
evaluationYear <- 2014

# Variables selected to assimilate
varSelected <- c("HGV", "BA","Vol")

########################################################################################
# Read predictions for the assimilation units
########################################################################################

# If useCells = FALSE
# One value for each assimilation unit.
# This value is the value found in the input-files to this project (Predicted_yyyy-mm-dd.txt)

# If useCells = TRUE
# Each cell value within the assimilation unit is stored in the file (Predicted_yyyy-mm-dd.txt)
# E.g. for a 40m radius plot, there will be around 21 raster cells (18m x 18m)

# Read model statistics (e.g. sigma if regression models were used)
# Only used if useFnPredVar set to FALSE
# Headings in the file: "Year" "sigma_HGV" "sigma_DGV" "sigma_BA" "sigma_Vol"
modelStats <- read.csv("indata/ModelStats.txt", header = T, sep=" ")

# Dates for the predictions (date when the data was acquired)
# Headings in the file: acquisitionDate fieldData
predDates <-  read.csv("indata/predDates.txt",  header  =  T,  sep="  ",  colClasses  =  c("Date",
"character"))

# Decide the state year for the prediction. The assimilation is always defined to be after the
growing season.
# Example 1: If photo was taken before 1 June 2002, then the assimilation year is set to be
2001.
# Example 2: If photo was taken after 1 June 2002, then the assimilation year is set to be 2002.
```

```
predDates$stateYear    <-    sapply(predDates$acquisitionDate,    simplify=T,    FUN=function(k)
{return(calculateStateYear(k))})

# Read predicted data from files
# These files could either contain plot-by-plot data or plots consisting of raster cells.
# Name the files: “Predicted_2003-10-13.txt” where 2003-10-13 is the acquisition dates found in
predDates.
# Header in the files: ID cellID HGV DGV BA Vol (if using cells)
# Header in the files: ID HGV DGV BA Vol (if not using cells)
predPath <- paste0("indata/predictedPlots")
ass <- assObj(path=predPath, sigma=modelStats, predDates=predDates, varSelected=varSelected,
useFnPredVar=useFnPredVar)

# Create a list of the growth functions used
tv_fkn <- list("HGV"=hgv_tv, "BA"=ba_tv, "Vol"=vol_tv)
tv_prim_fkn <- list("HGV"=hgv_tv_prim, "BA"=ba_tv_prim, "Vol"=vol_tv_prim)
tv_var_fkn <- list("HGV"=hgv_tv_var, "BA"=ba_tv_var, "Vol"=vol_tv_var)

# Parameters to the growth models
# These are different for different tree species.
# Which tree species to use is decided depending on the amount of each species.
treespeciesParam <- returnTreespeciesParam()

# Read field evaluation data (ground truth)
# The file need to contain ID, SIS, age, HGV, Vol, BA, stateYear and the tree species mix as
volume of pine, spruce and decidious and the total volume, see treespeciesEval()-function for
more information
plotsEval_subs <- read.csv("fieldData/forestState.csv", sep=";", dec=",", stringsAsFactors=F)

#Calculate the percentage of each treespecies on each plot
treespeciesEvalPercent_tmp <- treespeciesEval(plotsEval)

# Assign treespecies-ID to each plot with respect to the volume percentage of each treespecies
plotsEval_subs                  <-                  calcTreespeciesID(df=plotsEval_subs_merge,
dfPercentage=treespeciesEvalPercent_tmp)

# Fore- or backcast plots to the user specified evaluation year (evaluationYear)
plotsEval_subs   <-   calcSpecificStateYear_EvalPlots(df=plotsEval_subs,   year=evaluationYear,
treespecies=treespeciesParam,              tv_fkn=tv_fkn,              varSelected=varSelected,
treespeciesPercent=treespeciesEvalPercent)


#########################################################################################
# Sample plot IDs chosen for assimilation
#########################################################################################
# These plots should be unchanged in the time period for assimilation.
assPlotsSelected  <- c(1,2,4,16,18,20,32,34,35,38,39,45,46,47,48,49,50,51,52,58,60,61)


#########################################################################################
# Create a data.frame preparing for saving assimilated values and their variance
#########################################################################################
for(k in 1:length(ass)) {
  ass[[k]] <- assDfs(ass = ass, i = k, year = ass[[k]]$stateYear, useCells = useCells)
}



#########################################################################################
# Notations
#########################################################################################
# Article     code       Explanation
# ==============================================
# p^2_t-1     p2_t_1     Variance at t-1
# p^2_t       p2_t       Variance of new assimilated value
# p~^2_t      p2tilde_t  New variance on the previous with growth added
# r^2_t       r2_t       Variance of the remote sensing "measurement"
# g(x_t-1)    tillvaxt   Growth between t-1 and t
# q^2_t       q2_t       Variance on the growth
# a_t         a_t        Derivative of the forecasting model
# K_t         K_t        Kalman gain
# xhat_t      xhat_t     New assimilated value
# z~_t        ztilde_t   Measurement (remote sensing)
```

```
#######################################################################################
# Assimilation
#######################################################################################
# Loop over the plots to be assimilated
for(p in assPlotsSelected) {

  # Extract which indices where data exist for the current plot (p)
  assInd <- assimilationIndicesForId(ass, p)

  j <- 0
  for(k in assInd[-length(assInd)]) { # Loop over all except the last k, because we assimilate
using k+1.
    # Counter that goes from 1, 2, 3... Used to get the k+1 assimilation index.
    j <- j+1

    # Index of the next measurement
    k1 <- assInd[j+1]

    # year1 is the year where we have an initial measurement or an assimilated value
    year1 <- ass[[k]]$stateYear

    # year2 is the year where we do the assimilation with the new measurement from remote sensing
    year2 <- ass[[k1]]$stateYear

    # Select the treespecies parameters for the specified treespecies on the current plot
    treespecies <- selectTreespecies(speciesID=plotsEval_subs$speciesID[plotsEval_subs$ID==p],
param=treespeciesParam)

    # Select the treespecies composition for the current plot
    thisTreespeciesPercent <- treespeciesEvalPercent[treespeciesEvalPercent$ID==p, ]

    # Store cell id if cells are used, otherwise set to -1
    if(useCells) {
      cIDs <- ass[[k1]]$pred$cellID[ass[[k1]]$pred$ID==p]
    } else {
      cIDs <- -1
    }

    # Loop cells to be assimilated for the current year (year2)
    for(cID in cIDs) {
      # Index for the plot p in the data.frames included in ass
      # According to the article Ehlers et al. 2013, the notations are as follows:
      # t   = k1
      # t-1 = k
      if(useCells) {
        p_assIndex_k <- which(x=(ass[[k]]$pred$ID==p & ass[[k]]$pred$cellID==cID), arr.ind=T)
        p_assIndex_k1 <- which(x=(ass[[k1]]$pred$ID==p & ass[[k1]]$pred$cellID==cID), arr.ind=T)
        p_fieldIndex <- which(x=(plotsEval_subs$ID==p), arr.ind=T)
      } else {
        #Assimilate for the whole evaluation plot at once
        p_assIndex_k <- which(x=(ass[[k]]$pred$ID==p), arr.ind=T)
        p_assIndex_k1 <- which(x=(ass[[k1]]$pred$ID==p), arr.ind=T)
        p_fieldIndex <- which(x=(plotsEval_subs$ID==p), arr.ind=T)
      }

      # Stop if more than one row in predict-file shares the same plot ID.
      stopifnot( length(p_assIndex_k)==1, length(p_assIndex_k1)==1, length(p_fieldIndex)==1 )

      # Save prediction and field data for this plot
      thisMeas <- ass[[k1]]$pred[p_assIndex_k1, ]
      thisField <- plotsEval_subs[p_fieldIndex, ]

      birthyear <- thisField$stateYear-thisField$age #Calculate when the plot was "born"
      age_year1 <- (year1-birthyear)
      age_year2 <- (year2-birthyear)
      SIS <- thisField$SIS

      #Save properties
      ass[[k1]]$properties$age[p_assIndex_k1] <- age_year2
      ass[[k1]]$properties$SIS[p_assIndex_k1] <- thisField$SIS #SIS for the sample plot

      # Run only if it's the first time-point
      if(k==assInd[1]) {
        #First "measurement"
        firstPred <- ass[[k]]$pred[p_assIndex_k, ]

        #First measurement of the selected variables
        x_t_1 <- firstPred[, varSelected]

        # Variance at t = 0
        p2_t_1 <-  ass[[k]]$predVar[p_assIndex_k,  varSelected]  #  TODO:  Here  we  need  to  add
 variance for each plot. i.e. add p_assIndex_k

        # Assimilated values are the same as the predictions/measurements in the starting year.
        ass[[k]]$assimilated[p_assIndex_k,    varSelected]    <-    ass[[k]]$pred[p_assIndex_k,
 varSelected]

        # The variance for the starting year is the same as for the predictions/measurements
 (from JW)
        ass[[k]]$assimilatedVariance[p_assIndex_k,               varSelected]               <-
 as.data.frame(ass[[k]]$predVar[p_assIndex_k, varSelected])

        # Weight is 1 for the starting year. Only based on the measurement...
        ass[[k]]$kalmanGain[p_assIndex_k, varSelected] <- 1

        # Save properties for the starting year
        ass[[k]]$properties$age[p_assIndex_k] <- age_year1
        ass[[k]]$properties$SIS[p_assIndex_k] <- thisField$SIS #SIS for the sample plot

      } else { # All other time-points
        # Load data from previous assimilation
        x_t_1 <- ass[[k]]$assimilated[p_assIndex_k, varSelected]
        p2_t_1 <- ass[[k]]$assimilatedVariance[p_assIndex_k, varSelected]
      }



      ##############################################################################
      # Growth
      ##############################################################################
      # Growth
      tillvaxt  <-  tv(fknList=tv_fkn,     x=x_t_1,  treespecies=treespecies,  age=age_year1,
 period=year2-year1,                    SIS=SIS,                      varSelected=varSelected,
 treespeciesPercent=thisTreespeciesPercent)

      # Variance on growth
      q2_t  <-  tv_var(fknList=tv_var_fkn,  x=tillvaxt,  treespecies=treespecies,  period=year2-
 year1, varSelected=varSelected)

      # Derivative of the forecasting model
      a_t <- 1 + tv_prim(fknList=tv_prim_fkn, x=x_t_1, treespecies=treespecies, age=age_year1,
 period=year2-year1,                    SIS=SIS,                      varSelected=varSelected,
 treespeciesPercent=thisTreespeciesPercent) #linj?risering f?r varians

      # Store a_t for future analysis
      ass[[k]]$a_t[p_assIndex_k, varSelected] <- a_t

      # Add growth to the previuos value
      x_t <- x_t_1 + tillvaxt

      # Calculate the new variance on the "previous+growth" value
      p2tilde_t <- (a_t)^2 * p2_t_1 + q2_t


      ##############################################################################
      # Measurement
      ##############################################################################
      # Remote sensing "measurement"
      ztilde_t <- thisMeas[, varSelected] # At time point t (=k1)

      # Variance of the remote sensing "measurement"
      r2_t <- ass[[k1]]$predVar[p_assIndex_k1, varSelected]


      ##############################################################################
      # Assimilation
      ##############################################################################
```

```
      # Kalman gain
      K_t <- p2tilde_t / (p2tilde_t + r2_t)

      # New assimilated value
      xhat_t <- x_t + K_t * (ztilde_t-x_t)

      # New variance of assimilated value
      p2_t <- (1-K_t)*p2tilde_t

      # Save assimilation
      ass[[k1]]$assimilated[p_assIndex_k1, varSelected] <- xhat_t
      ass[[k1]]$assimilatedVariance[p_assIndex_k1, varSelected] <- p2_t
      ass[[k1]]$kalmanGain[p_assIndex_k1, varSelected] <- K_t

      ass[[k1]]$assForecast[p_assIndex_k1, varSelected] <- x_t
      ass[[k1]]$assForecastVar[p_assIndex_k1, varSelected] <- p2tilde_t

    } #End for-loop over cells within the evaluation plot
  } # End for-loop over assimilation units (The whole plots)
} #End for-loop over assimilation time points



################################################################################
# Evaluation of the results in numbers
################################################################################

cat("If the Kalman Gain is 1, all of the measurement is used, if 0.5, the assimilated value is
the mean between forecasted and measured.\n")

if(useCells) {
  sink(file="outdata/Results_HGV_plotCells.txt")
  printAssResultsCell(ass=ass, variable="HGV", plots=assPlotsSelected)
} else {
  sink(file="outdata/Results_HGV_plot.txt")
  printAssResults(ass=ass, variable="HGV", plots=assPlotsSelected)
}
print2011Measures(fieldData=plotsEval_subs, variable="HGV", plots=assPlotsSelected)
sink()


if(useCells) {
  sink(file="outdata/Results_BA_plotCells.txt")
  printAssResultsCell(ass=ass, variable="BA", plots=assPlotsSelected)
} else {
  sink(file="outdata/Results_BA_plot.txt")
  printAssResults(ass=ass, variable="BA", plots=assPlotsSelected)
}
print2011Measures(fieldData=plotsEval_subs, variable="BA", plots=assPlotsSelected)
sink()


if(useCells) {
  sink(file="outdata/Results_Vol_plotCells.txt")
  printAssResultsCell(ass=ass, variable="Vol", plots=assPlotsSelected)
} else {
  sink(file="outdata/Results_Vol_plot.txt")
  printAssResults(ass=ass, variable="Vol", plots=assPlotsSelected)
}
print2011Measures(fieldData=plotsEval_subs, variable="Vol", plots=assPlotsSelected)
sink()


################################################################################
# Make plots
################################################################################
savePlots <- F # Set to TRUE if you want to save the plots to file
source("makePlot.R") #Not included in this report.
```

functions.R

```
#####################################################################
# Add tree species parameters to data.frame
#####################################################################
addSpecies <- function(df, species, id, B0, B1, B2, B3=0, B4=0, B5, B6, mse_hgv5, BV0, BV1, D0,
D1, D2, D3, D4=0, D5, D6=0, mse_ba5, DV0, DV1, F0, F1, F2, F3, F4, F5, F6=0, mse_vol5, FV0,
FV1) {
  rbind(df, data.frame("species"=species, "id"=id,
              "B0"=B0, "B1"=B1, "B2"=B2, "B3"=B3, "B4"=B4, "B5"=B5, "B6"=B6, "HGV"=mse_hgv5/5,
              "BV0"=BV0, "BV1"=BV1,
              "D0"=D0, "D1"=D1, "D2"=D2, "D3"=D3, "D4"=D4, "D5"=D5, "D6"=D6, "BA"=mse_ba5/5,
              "DV0"=DV0, "DV1"=DV1,
              "F0"=F0, "F1"=F1, "F2"=F2, "F3"=F3, "F4"=F4, "F5"=F5, "F6"=F6, "Vol"=mse_vol5/5,
              "FV0"=FV0, "FV1"=FV1
  ))
}


#####################################################################
# Select ID where no predictions are NA
#####################################################################
noNAID <- function(ass, id) {
  nonNA_id <- NULL
  for(i in id) {
    na <- F
    for(a in 1:length(ass)) {
      idRow <- ass[[a]]$pred$ID==i
      if( sum(is.na(ass[[a]]$pred[idRow, ]))>0 | sum(idRow)==0 ) {
        na <- T
      }
    }
    if(na==F) {
      nonNA_id <- c(nonNA_id, i)
    }
  }
  return(nonNA_id)
}


#####################################################################
# Read from prediciton files and create an ass-object
#####################################################################
assObj <- function(path, sigma, predDates, varSelected, useFnPredVar=F) {
  #List all txt-files in the catalog
  flist <- list.files(path=path, pattern="Predicted.*txt")

  # Create a list containing all predictions
  ass <- list()

  i <- 0

  for(fname in flist) {
    i <- i+1

    t <- read.csv(paste0(path, "/", fname), header = T, sep=" ")
    #tSelected <- merge(t, assPlotsSelected, by.x="ID", by.y="ID", all.x=F)
    y <- as.Date(strsplit(x = strsplit(x = fname, split = "_")[[1]][2], split = "\\.")[[1]][1])

    # Select row in predDates
    predDates_i <- which(x=predDates$acquisitionDate==y, arr.ind=T)

    # Create the ass-object for the current year
    ass[[i]]                <-                list("year"=format(y,                   "%Y"),
"acquisitionDate"=predDates$acquisitionDate[predDates_i],
"stateYear"=predDates$stateYear[predDates_i], "pred" = t)

    if(useFnPredVar){
      # Sigma / variance
      ass[[i]]$predVar <- fnPredVar(ass[[i]], varSelected)

    } else {
      ass[[i]]$predVar <- ass[[i]]$pred
```

```
     ass[[i]]$predVar[    ,    varSelected]    <-    as.data.frame((sigma[i,    paste0("sigma_",
varSelected)])^2) # Calculate variance directly, sigma is in the file
    }
  }
  return(ass)
}


####################################################################
# Create the basic data.frames in the assimilation object
####################################################################
assDfs <- function(ass, i, year, useCells) {

  rList <- ass[[i]]

  # Set specific values depending on using cells or not.
  if(useCells) {
    startColumn <- 3 # Different start column if cellID-column is available.
    rList$properties    <-    data.frame("ID"=ass[[i]]$pred$ID,    "cellID"=ass[[i]]$pred$cellID,
"age"=NA, "SIS"=NA)
  } else {
    startColumn <- 2
    rList$properties <- data.frame("ID"=ass[[i]]$pred$ID, "age"=NA, "SIS"=NA)
  }

  rList$assimilated <- ass[[i]]$pred
  rList$assimilated[ , startColumn:ncol(rList$assimilated)] <- NA

  rList$assimilatedVariance <- ass[[i]]$pred
  rList$assimilatedVariance[ , startColumn:ncol(rList$assimilatedVariance)] <- NA


  rList$assForecast <- ass[[i]]$pred
  rList$assForecast[ , startColumn:ncol(rList$assForecast)] <- NA

  rList$assForecastVar <- ass[[i]]$pred
  rList$assForecastVar[ , startColumn:ncol(rList$assForecastVar)] <- NA


  rList$kalmanGain <- ass[[i]]$pred
  rList$kalmanGain[ , startColumn:ncol(rList$kalmanGain)] <- NA

  rList$a_t <- ass[[i]]$pred
  rList$a_t[ , startColumn:ncol(rList$a_t)] <- NA

  rList$assYear <- year

  return(rList)
}


####################################################################
# Calculate the state year
# Before 15 June, then state year is the year before, otherwise the same
####################################################################
calculateStateYear <- function(predDate) {

  month    <-    as.numeric(format(predDate,    format="%m"))    +    as.numeric(format(predDate,
format="%d"))/30
  if( month >= 6.5  ) {
    year <- format(predDate, format="%Y")
  } else {
    year <- format(predDate-365, format="%Y")
  }
  return(as.integer(year))
}


####################################################################
# Functions to extract variables from ass
####################################################################
assimilationYear <- function(ass, assInd=NA) {
  if(sum(is.na(assInd)>0)) {
    assInd <- 1:length(ass)
  }

  sapply(ass[assInd], simplify=T, FUN=function(k) {return(k$assYear)})
}

assimilationYearsForId <- function(ass, id) {
  dates <- integer()
  for(i in 1:length(ass)) {
    if(nrow(dplyr::filter(ass[[i]]$pred, ID==id))>0) {
      dates <- c(dates, ass[[i]]$assYear)
    }
  }
  return(dates)
}

assimilationIndicesForId <- function(ass, id) {
  ind <- integer()
  for(i in 1:length(ass)) {
    if(nrow(dplyr::filter(ass[[i]]$pred, ID==id))>0) {
      ind <- c(ind, i)
    }
  }
  return(ind)
}

assimilationDatesForId <- function(ass, id) {
  dates <- Sys.Date() # Format the vector dates as class Date. Didn't find a way to define the
vector as Date without first storing today's date and then reassign it in the for-loop
  j <- 0 # Counter for the vector where dates are stored.
  for(i in 1:length(ass)) {
    if(nrow(dplyr::filter(ass[[i]]$pred, ID==id))>0) {
      j <- j + 1
      dates[j] <- as.Date(ass[[i]]$acquisitionDate)
    }
  }
  return(dates)
}

# If assInd is supplied, the user choose which assIndices to return.
# If assInd is not supplied, all assIndices where values exist are returned
extractAssimilated <- function(ass, variable, id, assInd=NA) {
  if(sum(is.na(assInd)>0)) {
    assInd <- assimilationIndicesForId(ass=ass, id=id)
  }

  sapply(ass[assInd], simplify=T, FUN=function(k) {return(k$assimilated[k$assimilated$ID==id,
variable])})
}

extractAssimilatedCells <- function(ass, variable, id, assInd=NA) {
  if(sum(is.na(assInd)>0)) {
    assInd <- assimilationIndicesForId(ass=ass, id=id)
  }

  df <- data.frame("x"=NULL, "y"=NULL)
  for(i in assInd) {
    df <- rbind(df, data.frame("x"=ass[[i]]$assYear,
ass[[i]]$assimilated[ass[[i]]$assimilated$ID==id, c(variable, "cellID")]))
  }

  names(df) <- c("x", "y", "cellID")
  return(df)
}

extractAssimilatedVar <- function(ass, variable, id, assInd=NA) {
  if(sum(is.na(assInd)>0)) {
    assInd <- assimilationIndicesForId(ass=ass, id=id)
  }

  sapply(ass[assInd], simplify=T, FUN=function(k)
{return(k$assimilatedVariance[k$assimilatedVariance$ID==id, variable])})
}

extractAssForecast <- function(ass, variable, id, assInd=NA) {
  if(sum(is.na(assInd)>0)) {
    assInd <- assimilationIndicesForId(ass=ass, id=id)
  }

  sapply(ass[assInd], simplify=T, FUN=function(k) {return(k$assForecast[k$assForecast$ID==id,
variable])})
}

extractAssForecastVar <- function(ass, variable, id, assInd=NA) {
  if(sum(is.na(assInd)>0)) {
    assInd <- assimilationIndicesForId(ass=ass, id=id)
  }

  sapply(ass[assInd],                        simplify=T,                        FUN=function(k)
{return(k$assForecastVar[k$assForecastVar$ID==id, variable])})
}

extractAge <- function(ass, id, assInd=NA) {
  if(sum(is.na(assInd)>0)) {
    assInd <- assimilationIndicesForId(ass=ass, id=id)
  }

  sapply(ass[assInd],  simplify=T,  FUN=function(k)  {return(k$properties[k$properties$ID==id,
"age"])})
}

extractMeasure <- function(ass, variable, id, assInd=NA) {
  if(sum(is.na(assInd)>0)) {
    assInd <- assimilationIndicesForId(ass=ass, id=id)
  }

  sapply(ass[assInd], simplify=T, FUN=function(k) {return(k$pred[k$pred$ID==id, variable])})
}

extractMeasureVar <- function(ass, variable, id, assInd=NA) {
  if(sum(is.na(assInd)>0)) {
    assInd <- assimilationIndicesForId(ass=ass, id=id)
  }

  sapply(ass[assInd], simplify=T, FUN=function(k) {return(k$predVar[k$pred$ID==id, variable])})
}


####################################################################
# Extract cell values for plotting
####################################################################
extractMeasureForPlot <-   function(ass, variable, id, assInd=NA) {
  if(sum(is.na(assInd))>0) {
    assInd <- assimilationIndicesForId(ass=ass, id=id)
  }

  dfMeas <- data.frame()

  for(i in assInd) {
    dfMeas  <- rbind(dfMeas, data.frame("x"=ass[[i]]$assYear,
                                        ass[[i]]$pred[ass[[i]]$pred$ID==id, variable],
                                        ass[[i]]$predVar[ass[[i]]$predVar$ID==id,  c(variable,
"cellID")],
                         ass[[i]]$acquisitionDate
     ))
  }

  names(dfMeas)  <- c("x", "y", "var", "cellID", "acquisitionDate")
  return(dfMeas)
}


####################################################################
# For line plots,  inkludes
####################################################################
extractAssimilatedCellsForPlot <- function(ass, variable, id, assInd=NA) {
  if(sum(is.na(assInd)>0)) {
    assInd <- assimilationIndicesForId(ass=ass, id=id)
  }
  dfAss <- data.frame()
  dfFoca <- data.frame()

  for(i in assInd) { #TODO: No forecast for first year, change to two and add first value after
order()-commands
    dfAss  <- rbind(dfAss, data.frame("x"=ass[[i]]$assYear,
                                      "y"=ass[[i]]$assimilated[ass[[i]]$assimilated$ID==id,
variable],

ass[[i]]$assimilatedVariance[ass[[i]]$assimilatedVariance$ID==id, c(variable, "cellID")],
                                      "valuetype" = "Assimilated",
                        "acquisitionDate" = ass[[i]]$acquisitionDate
    ))
    dfFoca  <- rbind(dfFoca, data.frame("x"=ass[[i]]$assYear,
                                        "y"=ass[[i]]$assForecast[ass[[i]]$assForecast$ID==id,
variable],

ass[[i]]$assForecastVar[ass[[i]]$assForecastVar$ID==id, c(variable, "cellID")],
                                        "valuetype" = "ForecastOfAss",
                         "acquisitionDate" = ass[[i]]$acquisitionDate
    ))

  }


  ## reorder so that for every cellID every second is assimilated and every other is forecast
value in time order
  dfAss  <- dfAss[order(dfAss$cellID), ]
  dfFoca <- dfFoca[order(dfFoca$cellID), ]

  dftot <- rbind(dfAss, dfFoca)
  dftot[ c(F,T) , ] <- dfAss
  dftot[ c(T,F) , ] <- dfFoca ##Order: First forecast then assimilated

  names(dftot)  <- c("x", "y", "var", "cellID", "valuetype", "acquisitionDate")

  return(dftot)
}


#####################################################################
# Funktion to use for plotts of normal distributions
#####################################################################
extractAssForDistPlot <- function(ass, variable, id, assInd=NA) {
  if(sum(is.na(assInd)>0)) {
    assInd <- assimilationIndicesForId(ass=ass, id=id)
  }

  dfDist  <- data.frame()

  for(i in assInd) { #TODO: No forecast for first year, change to two and add first value after
order()-commands
    dfDist  <- rbind(dfDist, data.frame("x"=ass[[i]]$assYear,

                                        ass[[i]]$assimilated[ass[[i]]$assimilated$ID==id,
c("cellID", variable)],

sqrt(ass[[i]]$assimilatedVariance[ass[[i]]$assimilatedVariance$ID==id, variable]),

                                        ass[[i]]$assForecast[ass[[i]]$assForecast$ID==id,
variable],

sqrt(ass[[i]]$assForecastVar[ass[[i]]$assForecastVar$ID==id, variable]),

                                        ass[[i]]$pred[ass[[i]]$pred$ID==id, variable],
                                        sqrt(ass[[i]]$predVar[ass[[i]]$predVar$ID==id,
variable])
    ))

  }

  names(dfDist)  <- c("x", "cellID", "assim", "SD_ass",  "FoCa", "SD_foca", "pred", "SD_pred")
  return(dfDist)
```

```
}


#####################################################################
# Print out results from assimilation
#####################################################################
printAssResults <- function(ass, variable, plots) {

  cat("*****************************************************************\n",        variable,
"\n*************************************************************\n")

  for(p in plots) {
    cat("****************************", p, "************************************\n")
    cat("Year  ", "age  ", "SIS  ", "meas  (var)        ", "assFcast (var)     ", "ass   (var)
", "a_t   ", "KalGain", "\n", sep="")
     assInd <- assimilationIndicesForId(ass=ass, id=p)
    for(a in ass[assInd]) {
      p_id <- which(x=a$pred$ID==p, arr.ind=T)

      #cat(a$assYear,  ",  meas=",  sprintf("%6.2f",  a$pred[p_id,  variable]),  ",  assForecast=",
sprintf("%.2f", a$assForecast[p_id, variable]), ", ass=", sprintf("%.2f", a$assimilated[p_id,
variable]),  ",  measVar=",  sprintf("%.2f",  a$predVar[,  variable]),  ",  assForecastVar=",
sprintf("%.2f",    a$assForecastVar[p_id,    variable]),    ",    assVar=",    sprintf("%.2f",
a$assimilatedVariance[p_id,  variable]),  ",  Kal  gain=",  sprintf("%.2f",  a$kalmanGain[p_id,
variable]), "\n", sep="")
      cat(a$assYear, "  ",
         a$properties[p_id, "age"], "  ",
         sprintf("%4.1f", a$properties[p_id, "SIS"]), "  ",

         sprintf("%6.2f", a$pred[p_id, variable]),
         " (", sprintf("%8.2f", a$predVar[p_id, variable]), ")  ",

         sprintf("%6.2f", a$assForecast[p_id, variable]),
         " (", sprintf("%8.2f", a$assForecastVar[p_id, variable]),  ")  ",

         sprintf("%6.2f", a$assimilated[p_id, variable]),
         " (", sprintf("%8.2f", a$assimilatedVariance[p_id, variable]),  ")  ",

         sprintf("%4.2f", a$a_t[p_id, variable]), "  ",
         sprintf("%4.2f", a$kalmanGain[p_id, variable]),
         "\n", sep="")
    }
    cat("\n")
  }
}


#####################################################################
# Print out results from assimilation
# by cell
#####################################################################
printAssResultsCell <- function(ass, variable, plots) {

  cat("*****************************************************************\n",        variable,
"\n*************************************************************\n")

  for(p in plots) {
    cat("****************************", p, "************************************\n")
    assInd <- assimilationIndicesForId(ass=ass, id=p)
    cIDs <- ass[[1]]$pred$cellID[ass[[assInd[1]]]$pred$ID==p]
    for(cID in cIDs) {
      cat("CellID ", "Year  ", "age  ", "SIS  ", "meas  (var)        ", "assFcast (var)     ",
"ass   (var)        ", "a_t   ", "KalGain", "\n", sep="")
      for(a in ass[assInd]) {
        p_id <- which(x=(a$pred$ID==p & a$pred$cellID==cID), arr.ind=T)

        #cat(a$assYear, ", meas=", sprintf("%6.2f", a$pred[p_id, variable]), ", assForecast=",
sprintf("%.2f", a$assForecast[p_id, variable]), ", ass=", sprintf("%.2f", a$assimilated[p_id,
variable]),  ",  measVar=",  sprintf("%.2f",  a$predVar[,  variable]),  ",  assForecastVar=",
sprintf("%.2f",    a$assForecastVar[p_id,    variable]),    ",    assVar=",    sprintf("%.2f",
a$assimilatedVariance[p_id,  variable]),  ",  Kal  gain=",  sprintf("%.2f",  a$kalmanGain[p_id,
variable]), "\n", sep="")
        cat(sprintf("%6i", a$pred$cellID[p_id]), " ",
           a$assYear, "  ",
           a$properties[p_id, "age"], "  ",
           sprintf("%4.1f", a$properties[p_id, "SIS"]), "  ",

           sprintf("%6.2f", a$pred[p_id, variable]),
           " (", sprintf("%8.2f", a$predVar[p_id, variable]), ")  ",

           sprintf("%6.2f", a$assForecast[p_id, variable]),
           " (", sprintf("%8.2f", a$assForecastVar[p_id, variable]),  ")  ",

           sprintf("%6.2f", a$assimilated[p_id, variable]),
           " (", sprintf("%8.2f", a$assimilatedVariance[p_id, variable]),  ")  ",

           sprintf("%4.2f", a$a_t[p_id, variable]), "  ",
           sprintf("%4.2f", a$kalmanGain[p_id, variable]),
           "\n", sep="")
      }
      cat("\n")
    }
  }
}


#####################################################################
# Print out measures from 2011
#####################################################################
print2011Measures <- function(fieldData, variable, plots) {
  options(digits=4)
  for(p in plots) {
    p_id <- which(x=fieldData$ID==p, arr.ind=T)
    cat("Year ", fieldData$stateYear[p_id], ", measured at the 40m plots, ID: ", p, ", ",
variable, ": ", fieldData[p_id, variable], "\n", sep="")
  }
}


#####################################################################
# plotta resultat. Kräver att assimPlot skapas innan funktionen exv.:
# assimPlot <- ggplot(forecastDF, aes(x = forecastDF$forecastTimes, y = forecastHGV))
#####################################################################
fnAssPlot <- function(assimPlot, limY_min, limY_max, limX_min, limX_max, plotTitle, x_breaks,
y_breaks, evalMeas) {

  assimPlot <- assimPlot +
    geom_ribbon(aes(ymin = lower, ymax = upper), colour = "orange", linetype = 2, size = 1, fill
= "orange", alpha= 0.6)  +
    geom_line(colour = "purple", size = 1.5) +
    theme_minimal( base_size = 24) +
    theme(legend.position = "none", plot.margin = unit(c(0,0,0,0), "cm"), axis.title.x =
element_blank(), axis.title.y = element_blank()) +#, axis.text.x = element_blank(), axis.text.y
= element_blank()) +
    geom_errorbar(aes(x=x, ymax=lowerLast, ymin=upperLast), width=20, size=1.5) +
    geom_point(aes(x=x, y=yLast), size=5, colour="purple", shape=16) + #Last assimilation
    geom_point(aes(x=acquisitionDate, y=meas), size=5, colour="red", shape=4, data=evalMeas) +
#Evalation against field truth
    xlab("") +
    ylab("") +
    #ggtitle(plotTitle) +
    scale_y_continuous(breaks=y_breaks, limits=c(limY_min, limY_max), expand=c(0,0)) +
    scale_x_date(labels=date_format("%Y"), breaks=x_breaks)
    #scale_x_continuous(breaks=x_breaks, limits=c(limX_min, limX_max), expand=c(0,0))

  return(assimPlot)
}


#####################################################################
# Calculate percentage of each tree species on the plots
#####################################################################
treespeciesEval <- function(df) {
  treesp <- df[, c("pine", "spruce", "Deciduous", "Vol")]
  temp <- sapply(X = 1:ncol(treesp), FUN = function(i) {return( as.numeric(treesp[, i]) /
as.numeric(df[, "Vol"]) * 100)})

  treespPercent <- data.frame(cbind(df[ , "ID"], temp))
  names(treespPercent) <- c("ID", "pine", "spruce", "Deciduous", "Vol")

  return(treespPercent)
}

treespeciesWithNames <- function(df) {
  named <- sapply(X=1:ncol(df), FUN = function(i) { return(paste(names(df[i]), round(df[, i],
1), sep="="))})
  return(named)
}


#####################################################################
# Calculate ID depending on percentage in each class of tree species
#####################################################################
calcTreespeciesID <- function(df, dfPercentage) {

  df$speciesID <- NA

  for(i in 1:nrow(df)) {
    id <- 100 #100 will be default. A broadleaved category should be added later... TODO

    if(dfPercentage[i, "pine"] >= 65) {
      # ID=1, (pine >=65%)
      id <- 1
    } else if(dfPercentage[i, "spruce"] >= 65) {
      # ID=2, (spruce >=65%)
      id <- 2
    } else if((dfPercentage[i, "pine"] + dfPercentage[i, "spruce"]) >= 65) {
      # ID=12, conifer (spruce+pine >= 65%)
      id <- 12
    }  else  if(  ((dfPercentage[i,  "pine"]  +  dfPercentage[i,  "spruce"])  <  65)  &&
(sum(dfPercentage[i, "Deciduous"]) >=35) && (sum(dfPercentage[i, "Deciduous"]) <65) ) {
      # ID=100, mix (spruce+pine <65%, broadleaved >=35%, <65%)
      id <- 100
    }

    df[i, "speciesID"] <- id
  }

  return(df)
}


############################################################
# Select tree species from the parameters list given a speciesID
############################################################
selectTreespecies <- function(speciesID, param) {
  return(param[param$id==speciesID, ])
}


###########################################################
# Calculate HGV, DGV, Vol for the year inputed
# Base the calculations on the year the plot was inventoried
# At the moment, no tree species information is used.
###########################################################
calcSpecificStateYear_evalPlots  <-  function(df,  year,  treespecies,  tv_fkn,  varSelected,
treespeciesPercent) {
  for(i in 1:nrow(df)) {
    thisTreespeciesPercent <- treespeciesPercent[treespeciesPercent$ID==df$ID[i], ]
    x_t <- df[i, varSelected]
    age_year1 <- df[i, "age"]
    stateYear <- df[i, "stateYear"]
    SIS <- df[i, "SIS"]

    if(year==stateYear) {
      next
    }

    year1 <- min(year, stateYear)
    year2 <- max(year, stateYear)
    tillvaxt              <-              tv(fknList=tv_fkn,                              x=x_t,
treespecies=selectTreespecies(speciesID=df$speciesID[i],  param=treespecies),  age=age_year1,
period=year2-year1,                      SIS=SIS,                     varSelected=varSelected,
treespeciesPercent=thisTreespeciesPercent)

    if(year < stateYear) {
      tillvaxt <- -tillvaxt
    }
    df[i, varSelected] <- df[i, varSelected] + tillvaxt
    df[i, "age"] <- df[i, "age"] + (year-stateYear)
    df[i, "stateYear"] <- year
  }
  return(df)
}


###########################################################
# Plot difference between ass, rep and for and the field measured value
###########################################################
ggplotDeviation <- function(variable, data, title, selVar=NULL, ylab) {

  selVar <- paste0(variable, c("_assDiff", "_repDiff", "_forDiff"))
  dataMelt <- melt(data=data, id.vars="ID", measure.vars=selVar)

  ggResultsPlot <- ggplot(data=dataMelt, mapping=aes(x=factor(ID), y=value, fill=variable))
  ggResultsPlot <- ggResultsPlot +
    theme_bw( base_size = 10) +
    theme(legend.position  =  "right",  plot.margin  =  unit(c(0,0,0,0),  "cm"))  +  #,
axis.ticks.x=element_line(size=1)) +
    geom_bar(stat="identity", position=position_dodge()) +
    ylab(ylab) +
    xlab("Forest stand ID") +
    scale_y_continuous() +
    ggtitle(title) +
    scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"),
              name="Method",
              breaks=selVar,
              labels=c("Assimilated", "Most recent est.", "Forecasted")
              )

  ggsave(filename=paste0("plotsCells/resultDiffEval_", variable, ".png"), width=17.5, height=7,
units="cm", dpi=300, plot=ggResultsPlot)
}


################################################################################
# Calculate RMSE
# given that df contains diff (estimated-field)
################################################################################
rmseDiffAsInput <- function(df) {
  return(sqrt(colSums(df^2) / nrow(df)))
}


################################################################################
# Calculate Bias
# given that df contains diff (estimated-field)
################################################################################
biasDiffAsInput <- function(df) {
  return(colSums(df) / nrow(df))
}
```

growthModels.R

```
##################################################################################
# Parameters to the growth models
# These are different for different tree species.
# Which tree species to use is decided depending on the amount of each species.
##################################################################################
returnTreespeciesParam <- function() {
  treespeciesParam <- data.frame()
  #B0, B1... = HGV
  #D0, D1... = BA
  #F0, F1... = Vol

  # ID=1, (pine >=65%)
  treespeciesParam <- addSpecies(df=treespeciesParam, species="pine", id="1",
                     B0=0.2966,  B1=-0.2598,  B2=0.4880,  B3=0,  B4=0,  B5=0.1488,  B6=0.0058,
mse_hgv5=1.46,
                     BV0=1.34, BV1=-0.131,
                     D0=0.3408, D1=-0.0071, D2=0.3599, D3=-0.7863, D4=0, D5=1.0175, D6=-0.5603,
mse_ba5=4.76,
                     DV0=1.43, DV1=0.2630,
                     F0=-0.6707,   F1=0,   F2=0.3706,   F3=-0.2897,   F4=-0.0046,   F5=1.1857,
mse_vol5=200,
                     FV0=8.68, FV1=0.248
  )

  # ID=2, (spruce >=65%)
  treespeciesParam <- addSpecies(df=treespeciesParam, species="spruce", id="2",
                     B0=-0.4226,  B1=-0.2156,  B2=0.4275,  B3=0,  B4=0,  B5=0.4140,  B6=0.0039,
mse_hgv5=1.92,
                     BV0=1.55, BV1=-0.113,
                     D0=0.6755,  D1=-0.0222,  D2=0.3172,  D3=-0.5512,  D4=0,  D5=0.6712,  D6=0,
mse_ba5=5.78,
                     DV0=1.92, DV1=0.183,
                     F0=0.2207, F1=-0.0003, F2=0.3907, F3=-0.2712, F4=-0.0048, F5=0.8671, F6=0,
mse_vol5=335,
                     FV0=13.67, FV1=0.147
  )

  # ID=12, conifer (spruce+pine >= 65%)
  treespeciesParam <- addSpecies(df=treespeciesParam, species="conifer", id="12",
                     B0=0.0679,  B1=-0.2301,  B2=0.4585,  B3=0,  B4=0,  B5=0.2800,  B6=0.0040,
mse_hgv5=1.84,
                     BV0=1.46, BV1=-0.103,
                     D0=0.2890,  D1=-0.0130,  D2=0.1860,  D3=-0.5307,  D4=0,  D5=0.8280,  D6=0,
mse_ba5=6.36,
                     DV0=1.83, DV1=0.260,
                     F0=0.8171, F1=0, F2=0.3867, F3=-0.3721, F4=-0.0038, F5=0.7302, F6=0.2805,
mse_vol5=317,
                     FV0=14.58, FV1=0.113
  )

  # ID=100, mix (spruce+pine <65%, broadleaved >=35%, <65%)
  # THIS ONE WILL BE SELECTED IF NONE of 1, 2, 12, 100 fits the conditions.
  treespeciesParam <- addSpecies(df=treespeciesParam, species="mix", id="100",
                     B0=-1.6143,  B1=-0.1472,  B2=0.2891,  B3=-0.0144,  B4=0.3900,  B5=0.5533,
B6=0.0020, mse_hgv5=2.24,
                     BV0=1.71, BV1=-0.133,
                     D0=0.4635,  D1=-0.0140,  D2=0.1718,  D3=-0.4725,  D4=0,  D5=0.7032,  D6=0,
mse_ba5=8.62,
                     DV0=2.65, DV1=0.096,
                     F0=0.3930,  F1=0,  F2=0.3867,  F3=-0.1555,  F4=-0.0091,  F5=0.6961,  F6=0,
mse_vol5=302,
                     FV0=11.20, FV1=0.260
  )

  return(treespeciesParam)
}


####################################################################
# HGV forecasting
# Forecast the growth in meters after "period" years,
#  i.e. height growth basal area weighted:
#  HGV(age+period) = HGV + hgv_tv(period)
####################################################################
hgv_tv <- function(x, treespecies, age, period, SIS, treespeciesPercent)
{
  if(period==0) { #Don't do forecast if year2 is the same as year1.
    return(0)
  } else{
    HGV1 <- x$HGV
    dHGV <- hgv2(HGV1, age, period, SIS, treespecies)

    return( dHGV )
  }
}

hgv2 <- function(HGV1, age, period, SIS, treespecies) {
  return(   exp(   treespecies$B0   +   treespecies$B1*HGV1   +   treespecies$B2*log(HGV1)   +
treespecies$B3*age      +      treespecies$B4*log(age)      +      treespecies$B5*log(SIS)      +
treespecies$B6*(HGV1*SIS) )/5*period )
}

hgv_tv_prim <- function(x, treespecies, age, period, SIS, treespeciesPercent)
{
  HGV1 <- x$HGV

  HGV_prim <- hgv2(HGV1, age, period, SIS, treespecies) * (treespecies$B1 + treespecies$B2/HGV1
+ treespecies$B6*SIS)

  return( HGV_prim )
}

hgv_tv_var <- function(x, treespecies, period) {
  if(period==0) {
    return(0)
  } else {
    return( ( ( treespecies$BV0 + treespecies$BV1*(x$HGV*5/(period)) ) / 5 * (period) )^2 )
  }
}


####################################################################
# BA forecasting
# The function predict the forecast for 5 years.
# Therefore it need to be divided by five to have the yearly growth.
####################################################################
ba_tv <- function(x, treespecies, age, period, SIS, treespeciesPercent) {
  if(period==0) { #Don't do forecast if year2 is the same as year1.
    return(0)
  } else {
    BA1 <- x$BA

    dBA <- ba2(BA1, age, period, SIS, treespecies, treespeciesPercent)
    return(dBA)
  }
}

ba2 <- function(BA1, age, period, SIS, treespecies, treespeciesPercent)  {

  return(    exp(treespecies$D0    +    treespecies$D1*BA1    +    treespecies$D2*log(BA1)    +
treespecies$D3*log(age) + treespecies$D5*log(SIS) + treespecies$D6*(BA1/age)) /5*period )
}


####################################################################
# BA forecasting, derivative
####################################################################
ba_tv_prim <- function(x, treespecies, age, period, SIS, treespeciesPercent) {
  BA1 <- x$BA

  BA_prim <- ba2(BA1, age, period, SIS, treespecies, treespeciesPercent) * (treespecies$D1 +
treespecies$D2/BA1 + treespecies$D6/age)

  return(BA_prim)
}

ba_tv_var <- function(x, treespecies, period) {
  if(period==0) {
    return(0)
  } else {
    return( ( ( treespecies$DV0 + treespecies$DV1*(x$BA*5/(period)) ) / 5 * (period) )^2 )
  }
}
```

```
####################################################################
# Vol forecasting
####################################################################
vol_tv <- function(x, treespecies, age, period, SIS, treespeciesPercent) {
  if(period==0) { #Don't do forecast if year2 is the same as year1.
    return(0)
  } else{
    Vol1 <- x$Vol

    dVol <- vol2(Vol1, age, period, SIS, treespecies, treespeciesPercent)

    return(dVol)
  }
}

vol2 <- function(Vol1, age, period, SIS, treespecies, treespeciesPercent) {
  #Extract proportion spruce (0-0.65)
  propSpruce <- treespeciesPercent$spruce/100

  return(   exp(   treespecies$F0   +   treespecies$F1*Vol1   +   treespecies$F2*log(Vol1)   +
treespecies$F3*log(age)     +     treespecies$F4*age     +     treespecies$F5*log(SIS)     +
treespecies$F6*log(1+propSpruce) )/5*period )
}

####################################################################
# Vol forecasting, derivative
####################################################################
vol_tv_prim <- function(x, treespecies, age, period, SIS, treespeciesPercent) {
  Vol1 <- x$Vol

  Vol_prim   <-   vol2(Vol1,   age,   period,   SIS,   treespecies,   treespeciesPercent)   *
(treespecies$F2/Vol1 + treespecies$F1)

  return(Vol_prim)
}

vol_tv_var <- function(x, treespecies, period) {
  if(period==0) {
    return(0)
  } else {
    return( ( ( treespecies$FV0 + treespecies$FV1*(x$Vol*5/(period)) ) / 5 * (period) )^2 )
  }
}

####################################################################
# Forecasting for all selected variables
####################################################################
tv <- function(fknList, x, treespecies, age, period, SIS, varSelected, treespeciesPercent) {
  r <- data.frame(NULL)
  for(v in varSelected) {
    r[1,v] <- fknList[[v]](x, treespecies, age, period, SIS, treespeciesPercent)
  }
  return(r)
}


####################################################################
# Forecasting, derivative of all selected variables
####################################################################
tv_prim <- function(fknList, x, treespecies, age, period, SIS, varSelected, treespeciesPercent)
{
  r <- data.frame(NULL)
  for(v in varSelected) {
    r[1,v] <- fknList[[v]](x, treespecies, age, period, SIS, treespeciesPercent)
  }
  return(r)
}


####################################################################
# Variance for all forecasting models
####################################################################
```

```
tv_var <- function(fknList, x, treespecies, period, varSelected) {
  r <- data.frame(NULL)
  for(v in varSelected) {
    r[1,v] <- fknList[[v]](x=x, treespecies=treespecies, period=period)
  }
  return(r)
}
```