COSTA
common set of tools for the assimilation of data

| | |
|---|---|
| MEMO | CTA memo200602 |
| Date | 26-05-2006 |
| Author(s) | drs. C. van Velzen |
| Subject | Pilot WAQUA-TRIWAQ and COSTA |

**Table of contents**



# 1 Introduction

The COSTA environment contains a number of small models and an RRSQRT-filter. Other filters like an Ensemble Kalman and particle filter are under development. However the available models are small test models. These models are less complicated then real-life simulation models both numerically as in software complexity.

WAQUA/TRIWAQ is a shallow water simulation software system used operationally by the Dutch national institute for coastal and marine management (Rijkswaterstaat). This software package contains various data assimilation methods like: steady-state Kalman, Ensemble Kalman and RRSQRT Kalman. The data assimilation techniques, implemented in WAQUA/TRIWAQ are fully entangled with the simulation model itself. It is therefor far from trivial to try-out an alternative data assimilation method or to apply the

data assimilation methods that are developed for WAQUA/TRIWAQ for other simulation models.

In the pilot project, described in this memo, COSTA is used for the WAQUA/TRIWAQ model. With this pilot project we want to accomplice the following:

- Create an initial version of a COSTA model component for the WAQUA/TRIWAQ model. This model component can be used as one of the test-models for testing COSTA assimilation methods.
- Find out what the weak points are of the COSTA design especially the model component.
- Create an RRSQRT-filter that is equivalent to the RRSQRT-filter that is currently available in WAQUA/TRIWAQ.
- Obtaining experience in creating COSTA-model components leading to a better design of the various COSTA model builders that will be developed in the future.

The usage of WAQUA/TRIWAQ has a number of advantages:

- Data assimilation is already available, therefore it is possible to compare the original simulation results and performance with the version using COSTA.
- WAQUA/TRIWAQ is a complicated model and complicated software. If it is possible to apply COSTA with WAQUA/TRIWAQ it is likely that COSTA can also be applied for other real-life models.
- Rijkswaterstaat wants to use COSTA as tool for data assimilation. This pilot is the first step in introducing COSTA in their simulation software.

The different steps that are made towards the creation of the pilot version are described in Section 2. The created COSTA model component of WAQUA/TRIWAQ is described in more detail in Section 3. A number of complications have emerged in this pilot project. There are also some outstanding items for future enhancement. Some of these items have already solved, others are solved temporarily but need more research in the future. These issues are discussed in Section 4. Finally the COSTA model component and created COSTA RRSQRT-filter are tested and compared with the original WAQUA/TRIWAQ software. The results of this test are given in Section 5

# 2 Development steps

The Pilot is divided in a number of steps. Each step results in a working simulation package with data assimilation that can be tested. The different steps are described in detail in the following sections:

## 2.1 Root covariance matrix

The model error covariance matrix $C$ is approximated by its root $L$ such that $C \approx LL^T$. The rank of the root matrix $L$ is bounded such that it can be stored in memory. The dimension of $L$ is $nstate \times nmode$. Where $nstate$ denotes the dimension of the state-vector of stochastic model and $nmode$ the number of modes in the RRSQRT-filter.

The root covariance matrix is represented by a FORTRAN array `rmatL`. In this phase `rmatL` is replaced by $nmode$ COSTA state-vectors. The handles of these state-vectors are stored in the $nmode$-dimensional COSTA vector of handles `vMATL`. For this purpose it is necessary to crate a COSTA state-vector for the stochastic WAQUA-TRIWAQ model. In this phase this is done at the initialization of the Kalman filter and the original initialization (read from file) of rmatL. The COSTA-state-vector is now in its most simple form and does not contain any sub-states. Within the filter, the content of rmatL is copied were necessary. The state vector is split up in different sub-states in the other steps.

An other issue that is handled in this phase is the translation between the COSTA-model state (stored in `rmatL`) and the for this purpose extra routines are added
`user_model_get_state` and
`user_model_set_state`. This routine copies the values from the work arrays in the simulation process to the state-vector and vice versa. These user-routines are incorporated in the COSTA model-component of WAQUA/TRIWAQ in the following step.

## 2.2 COSTA model component

In this phase the COSTA model component is introduced. All communication between the data assimilation algorithm and the model is now performed through the COSTA model interface. The model that is created in this phase is a stochastic model, the deterministic model and the noise model are still mixed.

## 2.3 Stochastic model

The noise model is extracted from the stochastic COSTA model component. Two separate models are the product of this step:

- Deterministic WAQUA/TRIWAQ model
- Stochastic WAQUA/TRIWAQ model. This stochastic model has no knowledge of the deterministic model and uses the COSTA model interface for handling the deterministic model

The created COSTA model is described in more detail in Section 3.

## 2.4 COSTA main program

The WAQUA/TRIWAQ model is changed in a library. A COSTA main program will use functions from this library to run the simulation. The run-scripts are no longer necessary.

A result of removing the run-script is that is no longer possible to use alternative buffer-sizes.

## 2.5 COSTA RRSQRT-filter

The original RRSQRT-Filter is now part of the COSTA environment. The SIMONA memory management and tools library are no longer used. The input consists of the following data

- Time-span of the simulation and the used time-steps of the RRSQRT-filter
- two stochastic observers:
  - The Kalman observations. This is a stochastic observer containing all the observations that should be assimilated.
  - The validation observations. These observations are not assimilated but are compared to the model predictions.

  The observations and corresponding predictions are written to standard out in a human readable form and to an output-file that can be read by Octave or MATLAB for post-processing.
- The model instances of the modes (columns of the L-matrix) and a 0-mode (central mode) and a 1-mode (This mode is used for propagating the L-matrix work-variable will be deleted in future versions)
- The name of the sub-state containing all noise parameters and name of the sub-state of the deterministic model. This should be avoided in future versions. It will be discussed in more detail in Section 4.4.
- Optional the handles of one or more user routines:
  - Uncertainty output: Plots information on the uncertainty.
  - Scaling vector of deterministic model: Computation of new scaling vector for the deterministic model (called every time-step).
  - Post filter step: Possibility to handle all kind of model specific issues after a time-step like writing results, restarts etc to file.

  None of the user routines are mandatory. Currently, only a default implementation is available for the scaling (scale by identity). If no function is provided these steps will just be skipped.

The algorithm in pseudo-code is given by:

```
DO timestep
   IF first iteration
      Get models noise covariance matrix
   ENDIF

   propagate_mode0
   ScaleDeterministicModel (=Optional user function)

   IF (recompute scaling noise)
      ScaleNoiseParameters (=AR1 processes)
   ENDIF

   propagate_Lmatrix

   print_uncertainty (=Optional user function)

   reduce_width_of_Lmatrix

   print_uncertainty (=Optional user function)

   timeselection_observations
   assimilate_the_observations %One at a time

   print_predicionts_and_observations

   post_simulation step (=Optional user function)

ENDDO timestep
```

# 3 Design of the COSTA model component

Two COSTA model components are created:

1. Deterministic WAQUA/TRIWAQ model
2. Stochastic WAQUA/TRIWAQ model

The stochastic model is used by the assimilation method. The Stochastic model only uses the interface of the deterministic model. The Stochastic model is now configured using the information on the SDS-file. When the stochastic model is configured using e.g. an XML-file it is possible to use the stochastic model implementation in combination with an other deterministic model.

The deterministic model handles the propagation of the noise model and smoother and indirectly the deterministic model using the COSTA model interface of the deterministic model.

The state-vector of the stochastic model has the following form:

```
waqua-all-stoch
  +-- waqua-all-determ
  |   +-- waqua-up
  |   +-- waqua-vp
  |   +-- waqua-wp
  |   +-- waqua-sep
  |   +-- waqua-rp
  |   +-- waqua-turb.ener
  |   +-- waqua-turb.dissip
  +-- waqua-circul
  +-- waqua-all-noise
        +-- waqua-all-bound-noise
        |     +-- waqua-bnoise-wl
        |     +-- waqua-bnoise-vel
        |     +-- waqua-bnoise-ds
        |     +-- bnoise-rm
        +-- waqua-noise-windu
        +-- waqua-noise-windv
        +-- waqua-noise-visc
```

Depending on the model, some sub-states are optional.

The deterministic model component implements the following methods:

- `cta_model_create`
- `cta_model_free`
- `cta_model_compute`
- `cta_model_setstate`
- `cta_model_getstate`
- `cta_model_getobsvalues`
- `cta_model_axpy_model`
- `cta_model_axpyforc`

The stochastic model component implements the following methods:

- `cta_model_create`
- `cta_model_compute`
- `cta_model_setstate`
- `cta_model_getstate`
- `cta_model_getobsvalues`
- `cta_model_axpy_model`
- `cta_model_free`
- `cta_model_getnoisecount`
- `cta_model_getnoisecovar`

# 4 Complications

## 4.1 Noise model

For the boundary noise an AR(1) process is used for WAQUA/TRIWAQ. This model has the form

$$\gamma(t_{i+1}) = \alpha\gamma(t_i) + \eta(t_i) \tag{1}$$

The implementation of the noise model in WAQUA/TRIWAQ is however slightly different. The time correlation $\alpha$ is applied every half time-step.

After the first half time-step the time-correlation is applied. The problem of this approach is that the noise model is integrated into the ADI-scheme. The actual scheme of the noise model in WAQUA-TRIWAQ is implemented as:

$$\gamma(t_{i+\frac{1}{2}}) = \alpha_{waq}\gamma(t_i) \tag{2}$$

$$\gamma(t_{i+1} = \alpha_{waq}\gamma(t_{i+\frac{1}{2}}) + \eta(t_i) \tag{3}$$

In order to separate the noise model from the deterministic simulation model we have changed the noise model into

$$\gamma(t_{i+\frac{1}{2}}) = \gamma(t_i)\gamma(t_{i+1} = \alpha^2_{waq}\gamma(t_i) + \eta(t_i) \tag{4}$$

This change results in a different scaling and covariance approximation. The differences are of the same order as small changes of the parameter $\alpha_{waq}$. This nose model is not worst than the original model from our opinion.

## 4.2 Hidden part of state

There is a part of the model state that is not a part of the exported state-vector that is used in the RRSQRT-filter. This hidden part of the state-vector concerns the (integer) flags that indicate the drying and flooding of the model.

WAQUA/TRIWAQ is a nonlinear model. The columns of the $L$-matrix contain non-feasible values and these cannot be propagated directly in a non-linear model. Therefore we add the state of the 0-mode to the columns of the $L$-matrix and propagate this state. The propagated column of $L$ is then computed by subtracting the state of the propagated 0-mode. In COSTA this is done using the following algorithm:

```
x_old=mode0.GetState()
mode0.Compute(timestep)
DO i=1,nmode
  l=mode(i).GetState()
  l.axpy(1.0,x_old)
  l=mode(i).SetState()
  mode(i).Compute(timestep)
  l=mode(i).GetState()
  mode(i).axpy(-1.0,x_new)
ENDDO
```

If the set of dry points in the propagated 0-mode and column of $L$ we can have (locally) huge difference when we subtract these states. These large differences will cause the filter to blow-up. To avoid this we need the hidden part of the state in the model-axpy operation. In the model axpy, all differences at locations that are dry in one state and wet in the other are set to 0. The hidden part of the state is therefore very important.

Unfortunately there are some complications at this time. The current WAQUA-TRIWAQ model component always assumes that the state of 0-mode is subtracted (the hidden part is only saved for the 0-mode). As a result we cannot use the model axpy for other purposes and it must always be applied directly after the propagation of the mode. We cannot propagate all modes first and later subtract with the 0-mode because the hidden part of the state is not saved for the modes. An other problem is that for now we can only use the model-axpy at this place and nowhere else.

For this reason $x_{old}$ is not added using the model axpy. An option is to make the model axpy smarter such that it can detect problematic areas without using the hidden part of the state. An other options is to add some extra information in the state-vector like model-id and timestamp. In this way the model-axpy can determine whether the special drying-flooding option can be used. Finally we suggest to store the hidden state for all models. This will increase the memory usage but the increment is limited. It is even possible to use a sparse format for storing this information.

This is an interesting issue that needs some discussion.

## 4.3 Observations

The implemented version only supports "linear" interpolation for the observations (water-levels). In the future is should be possible to handle all kinds of observations and be able to compute the corresponding prediction especially for the columns of the $L$-matrix.

The idea is to adjust the filter to handle the interpolation:

```
!Note mode0 contains state x0(t)
mode0.getobsvalues(...,obs0)
!Note contains propagation of x0(t-1)+l(i)(t-1)
mode(i).getobsvalues(...,obs1)
! Compute H*l(i)
hl=obs1-obs0
```

An other small problem is that the interpolation matrices are created at the beginning of the simulation. As a result the stochastic observer is needed when creating a model instance. The interpolation matrices must be created on the fly in a next version of the model component for simplicity.

## 4.4 Scaling of the model

Scaling is essential for using WAQUA/TRIWAQ in combination with a RRSQRT-filter. The scaling is however not generic and dependent on the model. The COSTA model interface was not prepared for this extra information. For this reason it is chosen to supply two user dependent routines to the RRSQRT-filter that:

- compute scaling of the model-state
- compute the weights for the noise parameters

There is however a complication. It is very natural to extend the model interface for the scaling of the model-state. However the algorithm that computes the scaling parameters for the noise parameters involves the propagation of modes. This is a kind of computation that is not part of the model component. This algorithm is not model specific and can be used for other models having an AR(1) noise model.

The suggested approach is to add scaling to the model component. The scaling of the noise parameters can optional be done using an user defined function that is offered to the filter.

The generic scaling algorithm needs to know the name of the sub-state(s) that contain the noise parameters. probably we should extend the model interface with a method that returns the name(s) of the sub-states containing the noise parameters.

## 4.5 Writing to SDS-file

The computed results are written to the SDS-file after each time-step. Currently, this is handled by a user defined routine of the filter, that can write output. This is in general

not a problem. However when the time-steps in the filter are larger than the time-steps of the model not all data is written to de SDS-file.

It is quite complicated to handle this in the model component. An option is to write all data after the propagation of the 0-mode. Unfortunately no observations are assimilated at that time. The forecast is then written to the SDS-file instead of the analysis.

The SDS-file does not contain the observed values and the corresponding model forecast and analyzed prediction. This information now written to an file that can be read by MATLAB or Octave. The filter can be extended using a special user routine that replaces the writing to the MATLAB-file by writing to the SDS-file or any other format that is desired by the user.

### 4.6 Smoother and Scaling

The scaling of the variables in the smoother is not ok. The scaling parameters should be shifted at all time-steps. This problem will however be solved whenever the scaling is a part of the model.

### 4.7 Performance

The performance is comparable to initial version. The pilot version is 0-15% slower than the initial version. The performance seems to decrease when a large number of modes is selected.

In order to quickly create the RRSQRT-filter a number of the old routines from the original filter are used. This means however that the complete L-matrix is copied multiple times in each time-step. This can however be solved by optimizing the RRSQRT filter in the future.

## 5 Test

The CSM8 model with RRSQRT-filter is used for testing the various intermediate versions of the pilot. The simulation results are not the same as the original software. However the only change that has resulted in different simulation results is the adjusted AR(1) noise model as discussed in Section 4.1 . To illustrate this we have plotted the forecasted water-levels at the different observation stations of three different versions of the software:

1. Kalmina version: The original WAQUA/TRIWAQ software (version rkz1107-03-073)
2. Kalmina with adjusted noise model: The intermediate development version in which the adjusted noise model is introduced
3. Pilot-version: The result of the pilot project; the COSTA model component of WAQUA/TRIWAQ and the COSTA RRSQRT-filter.

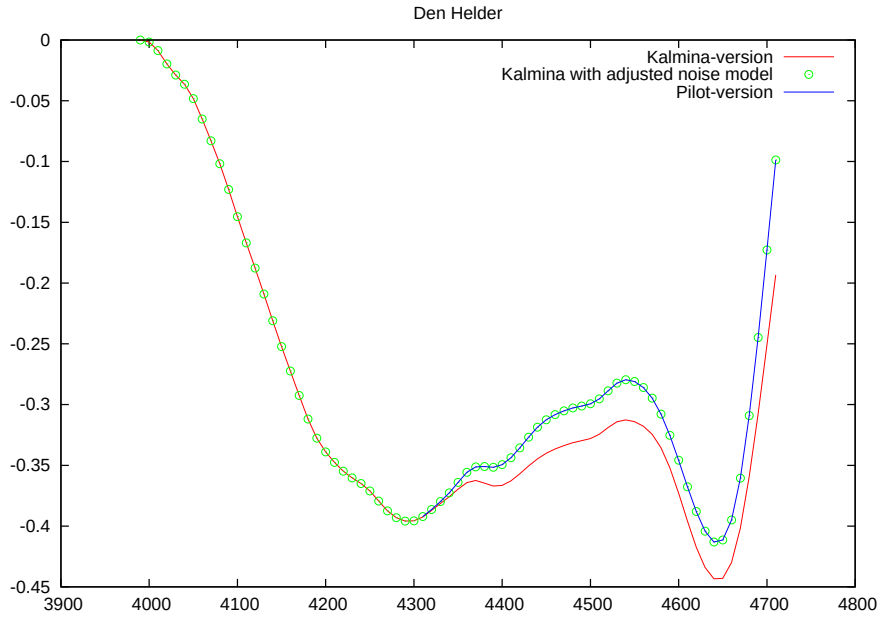


Figure 1: *Water level after analysis at Den Helder for three different versions of the software*

The water-levels are plotted in figures 1 to 5. Note that the first observations are available at $t = 4300$ from where, due to a different noise parameter, scaling and error covariance matrix different gain matrices are computed. This results in differences in simulation results.

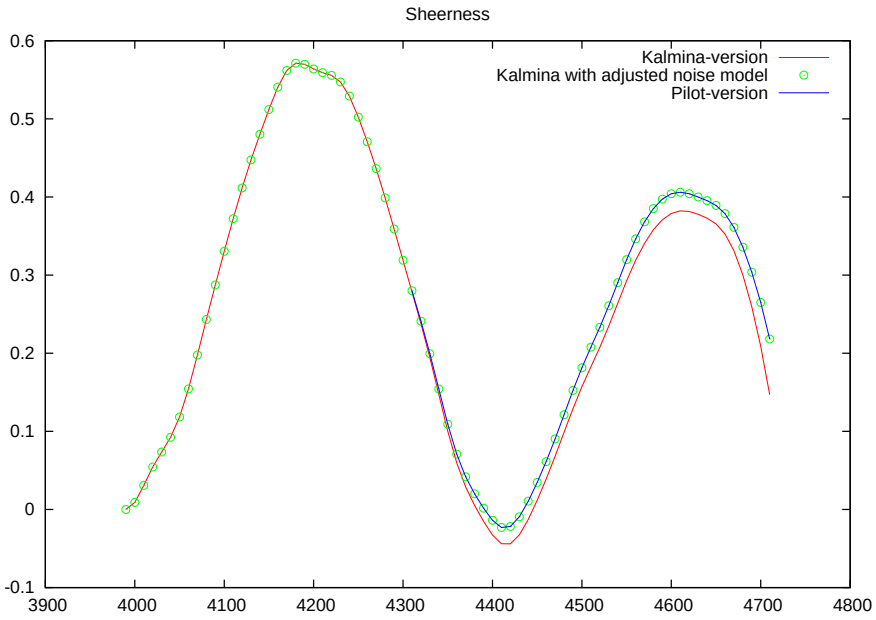


Figure 2: *Water level after analysis at Sheerness for three different versions of the software*

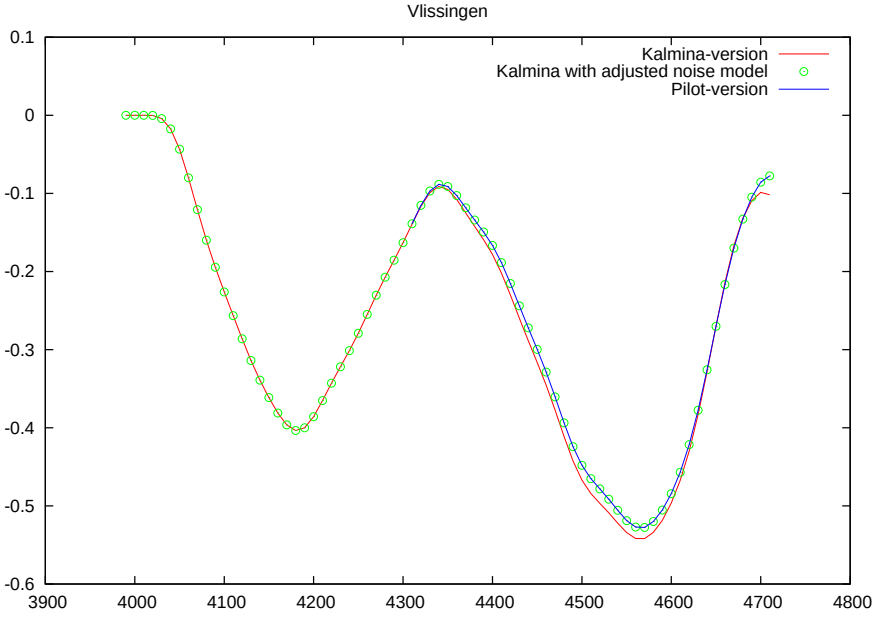


Figure 3: *Water level after analysis at Vlissingen for three different versions of the software*

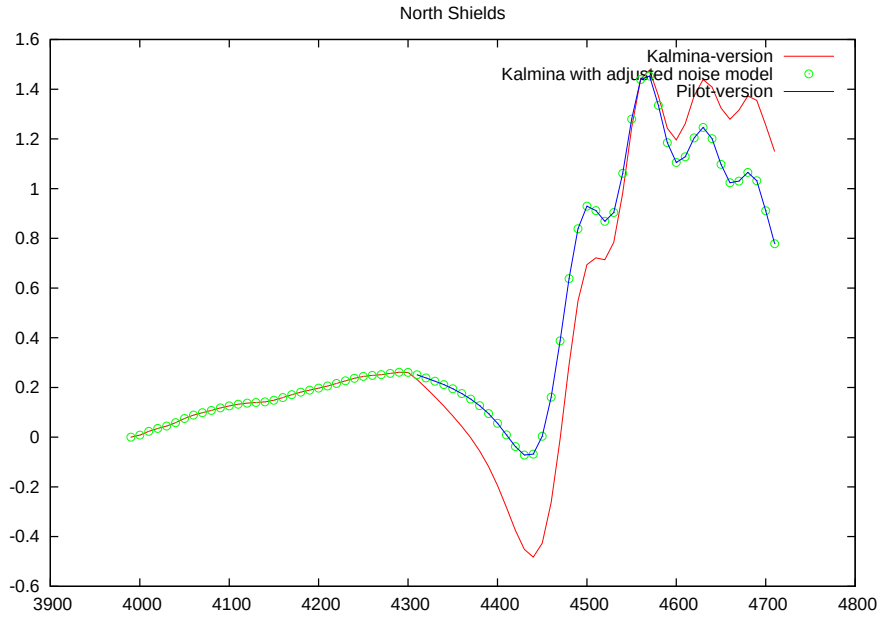


Figure 4: *Water level after analysis at North Shields for three different versions of the software*

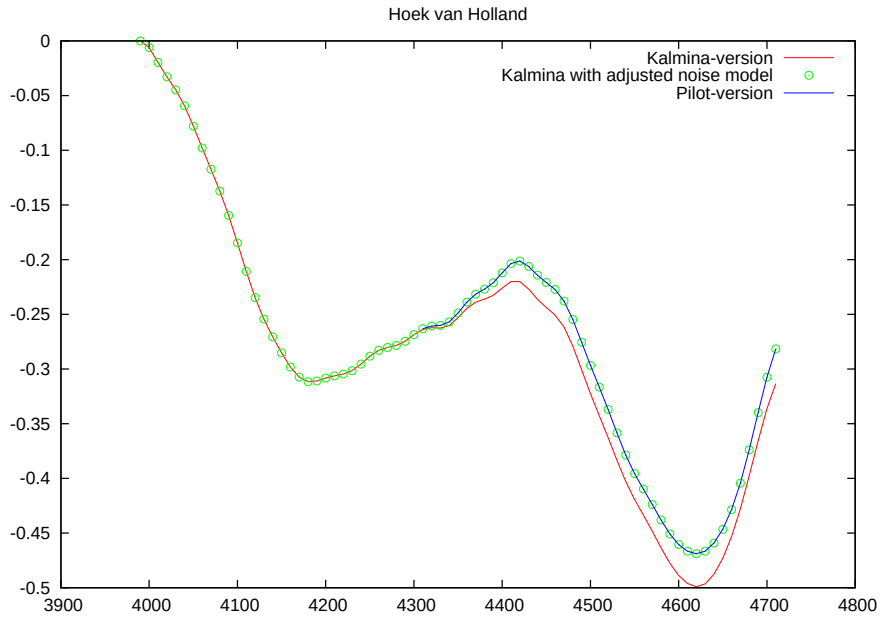


Figure 5: *Water level after analysis at Hoek van Holland for three different versions of the software*