How to CURUC

Michael Taylor

University of Reading

16/12/2020 14:47:00 version 0.2

FIDUCEO has received funding from the European Union's Horizon 2020 Programme for Research and Innovation, under Grant Agreement no. 638822

1 Introduction

1.1 Scope

This document outlines how to set-up the conda environment and necessary dependencies to run Gerrit Holl's python implementation of the CURUC recipes produced by Merchant et al.

The scope of this document is to:

- **Give technical details of the set-up procedure**
- **Present a semi-real model as an illustration**
- **Provide code to perform the CURUC calculation for the semi-real model data**
- **Display both exponential and vector forms of cross-line and cross-element correlation scales**
- **Analyse timing consideration associated with orbit sub-sampling**

1.2 Version Control

Date	Person	Version	Action / Reason
25/May/2018	M Taylor	v0.1	First draft.
29/May/2018	M Taylor	v0.2	Added missing git clone command to update Section 2.1. Minor formatting changes.

1.3 Applicable and Reference Documents

1.3.1 FIDUCEO reports and deliverable documents

- D2-2a Woolliams, E.J., Merchant, C.J., Harris, P. (2017). Principles behind the FCDR effects table (v1.a 25/08/2017), http://www.fiduceo.eu/content/mathematical-notation-fiduceo-publications.
- D2-2d Taylor, M. (2018). Report on the AVHRR FCDR: Uncertainty.
- D3.1 Block, T., Embacher, S. (2017). CDR/FCDR File Format Specification.
- RD.4 Merchant, C.J., Woolliams, E., Mittaz, J.P.D. (2018). Uncertainty and Error Correlation Quantification for FIDUCEO "easy-FCDR" Products: Mathematical Recipes. Version 0.9 (16/2/2018), http://www.fiduceo.eu/content/mathematical-notation-fiduceo-publications.
- RD.2 Merchant, C.J., Woolliams, E.J. (2017). Mathematical notation for FIDUCEO publications (v1a 17/11/2017), http://www.fiduceo.eu/content/mathematical-notation-fiduceo-publications.

1.3.2 References

- GUM, 2008, JCGM, 100:2008, Evaluation of measurement data – Guide to the expression of uncertainty in measurement: http://www.bipm.org/en/publications/guides
- VIM, 2008, JCGM, 200:2008, International vocabulary of metrology - basic and general concepts and associated terms: http://www.bipm.org/en/publications/guides
- Woolliams, E. R., Mittaz, J. P., Merchant, C. J., Hunt, S. E., & Harris, P. M. (2018, February). Applying Metrological Techniques to Satellite Fundamental Climate Data Records. In Journal of Physics: Conference Series (Vol. 972, No. 1, p. 012003). IOP Publishing.

1.4 Glossary

AVHRR	Advanced Very High Resolution Radiometer
FCDR	Fundamental Climate Data Record
HIRS	High resolution Infrared Radiation Sounder
IR	Infrared
NetCDF	Network Common Data Format

2 Set-up procedure

2.1 Installation

First, obtain anaconda ('conda') for linux (python 3.6x) with via wget:
https://repo.anaconda.com/archive/Anaconda3-5.1.0-Linux-x86_64.sh

Follow the installation instructions: https://docs.anaconda.com/anaconda/install/linux.

Next, install anaconda in your GWS:

```
./Anaconda3-5.1.0-Linux-x86_64.sh
```

In order to resolve dependencies associated with Gerrit's CURUC python code, it is also necessary to copy his conda environment to your home directory:

```
$ cp /home/users/gholl/.condarc /home/users/username/
```

You then need to create a conda clone of Gerrit's conda environment: (e.g. 'py36_gh'):

```
$ conda create -n py36_gh python=3 anaconda
```

Check that it has been created with:

```
$ conda env list
```

There are dependencies associated with xarray that mean that it is also necessary to install the typhon package from ARTS (http://radiativetransfer.org/tools/):

```
$ conda install typhon
```

Now set-up the conda environment:

```
$ source /path_to_anaconda_installation_dir/bin/activate py36_gh
```

Now, you will be ready to install and run Gerrit's CURUC python package from Github to your git checkouts:

```
$ mkdir /home/users/dirname/checkout/FCDR_HIRS/
```

```
$ git clone https://github.com/FIDUCEO/FCDR_HIRS.git
$ pip install .
```

2.2 Build considerations

If you run ipython3 and import the module FCDR_HIRS.metrology then the help file and source code can be accessed in the normal way by adding a single or double question mark:

```
[help file]   In: FCDR_HIRS.metrology?
[source code] In: FCDR_HIRS.metrology??
```

To run the CURUC python module there are 3 steps that need to be followed:

1. initialize FCDR_HIRS.metrology.allocate_curuc:
2. allocate the xarrays
3. call FCDR_HIRS.metrology.apply_curuc

Again, I recommend reading Gerrit's CURUC code and documentation:

```
[help file]   In: FCDR_HIRS.metrology.allocate_curuc?
[source code] In: FCDR_HIRS.metrology.allocate_curuc??
[help file]   In: FCDR_HIRS.metrology.apply_curuc?
[source code] In: FCDR_HIRS.metrology.apply_curuc??
```

It is important to note that **for each effect** there will be 14 xarray.DataArrays:

4 x C arrays (structured, independent x 2 views of same array)
4 x U arrays (structured, independent x 2 views of same array)
4 x R arrays (cross-channel structured and independent, cross-element, cross-line)
2 x boolean (dtype='?') masks for bad channels [n_c] and bad lines [n_l]

For example, the command I used to allocate the arrays for the semi-real model presented in the example code is:

```
A = FCDR_HIRS.metrology.apply_curuc(m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,coords,
mask1,mask2,return_vectors=True,interpolate_lengths=True,cutoff_l=ny,cutoff_e=nx)
```

and the key command I used to make the call to the CURUC code with orbit sub-sampling along-track: ys = 50 lines and across-track: xs = 5 elements, is:

```
A = run_curuc(ds,ys,xs)
```

The most important step in a real-world model run is to properly incorporate the information gleaned from the effects tables on the expected correlation functional forms outlined in the D2.2 documentation.

Case study and example run

2.3 A 2-effect toy model

At the Lisbon Workshop, Emma Woolliams presented a good example to illustrate CURUC that included a single structured (correlated) effect and a single independent (uncorrelated) effect:

EXAMPLE: What is the cross-channel covariance $\boldsymbol{S}$ between the Earth radiance values L_E in 3 different spectral channels A,B,C due to the uncertainty associated with the common structured effect in the internal calibration target temperature T and the uncertainty associated with the independent effect Earth Counts C_E?

$$\boldsymbol{S}_{L_E} = \boldsymbol{C}_T \boldsymbol{U}_T \boldsymbol{R}_T \boldsymbol{U}_T^T \boldsymbol{C}_T^T + \boldsymbol{C}_{C_E} \boldsymbol{U}_{C_E} \boldsymbol{R}_{C_E} \boldsymbol{U}_{C_E}^T \boldsymbol{C}_{C_E}^T \text{ (i.e. a sum over 2 effects)}$$

Covariance matrix for Earth Radiances in different channels due to common temperature error (**fully-correlated** cross-channel R):

$$\boldsymbol{S}_{L_E,T} = \begin{pmatrix} c_{LA,T} & 0 & 0 \\ 0 & c_{LB,T} & 0 \\ 0 & 0 & c_{LC,T} \end{pmatrix} \begin{pmatrix} u_T & 0 & 0 \\ 0 & u_T & 0 \\ 0 & 0 & u_T \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} u_T & 0 & 0 \\ 0 & u_T & 0 \\ 0 & 0 & u_T \end{pmatrix}^T \begin{pmatrix} c_{LA,T} & 0 & 0 \\ 0 & c_{LB,T} & 0 \\ 0 & 0 & c_{LC,T} \end{pmatrix}^T$$

where, $c_{LA,T} = \frac{\partial L_{E,A}}{\partial L_{ICT,A}} \frac{\partial L_{ICT,A}}{\partial T}$ is the sensitivity coefficient to convert from ICT temperature to Earth radiance uncertainty.

Covariance matrix for Earth Radiances in different channels due to errors in Earth counts (**no correlation** cross-channel R):

$$\boldsymbol{S}_{L_E,C_E} = \begin{pmatrix} c_{LA,CE} & 0 & 0 \\ 0 & c_{LA,CE} & 0 \\ 0 & 0 & c_{LA,CE} \end{pmatrix} \begin{pmatrix} u_{CA} & 0 & 0 \\ 0 & u_{CB} & 0 \\ 0 & 0 & u_{CC} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_{CA} & 0 & 0 \\ 0 & u_{CB} & 0 \\ 0 & 0 & u_{CC} \end{pmatrix}^T \begin{pmatrix} c_{LA,CE} & 0 & 0 \\ 0 & c_{LA,CE} & 0 \\ 0 & 0 & c_{LA,CE} \end{pmatrix}^T$$

where, $c_{LA,T} = \frac{\partial L_{E,A}}{\partial C_{E,A}}$ is the sensitivity coefficient to convert from Earth counts to Earth radiance uncertainty.

2.4 Semi-real model for an AVHRR orbit from the Easy FCDR

By analogy, I have created a variation of the toy model to test Gerrit's CURUC python implementation. In particular, I used a full processed orbit from NOAA-18 (07/07/2010) from v0.4 of the Easy AVHRR FCDR (pre-β) on CEMS. This provides total pixel-level independent and structured uncertainty for 3 IR channels.

We note that, in brightness temperature space, all sensitivity coefficients are = 1 as both the uncertainties and the channel data are in Kelvin. The simplest model that can be set-up to test the CURUC python implementation is then to assign the total independent uncertainty per pixel per channel to one combined independent 'effect' and the total structured uncertainty per pixel per channel to one combined structured 'effect' (in reality, the total uncertainties result from the propagation of the component effects).

By analogy with the toy model of the previous section, correlation matrices for the independent effect are assumed to represent no correlation and are the identity matrix (1s on diagonal) and correlation matrices for the structured effect are effect are assumed to represent full correlation and are therefore matrices of 1s (everywhere).

The python code to allocate these arrays, read in the pixel-level uncertainty data and apply the CURUC recipes using Gerrit's python implementation is given in section 5.

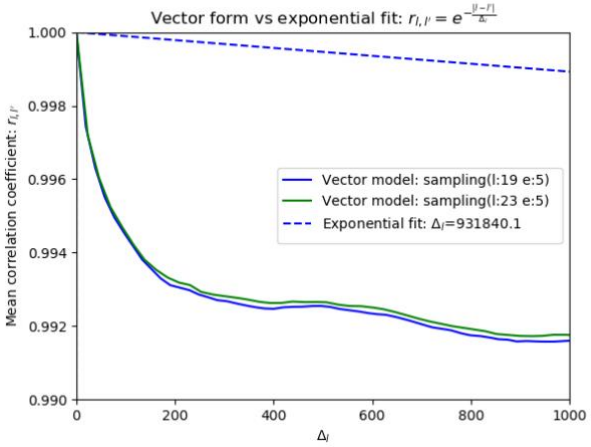
2.5 CURUC output

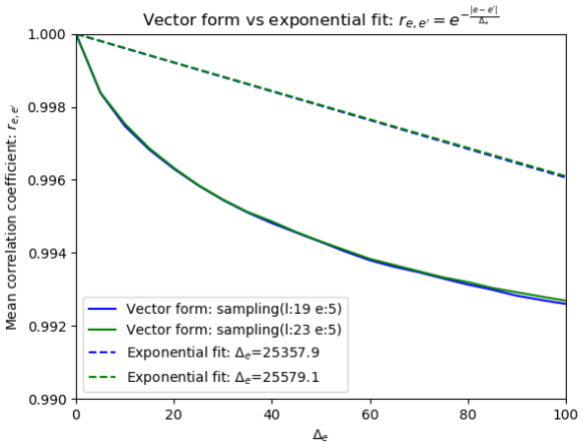
Gerrit's CURUC python code outputs 6 arrays:

1. Cross-channel correlation matrix for structured effects: R_cs [n_c, n_c] which, for the semi-real model reproduces a matrix of 1s (to 2 d.p.)
2. Cross-channel correlation matrix for independent effects: R_ci [n_c, n_c] which, for the semi-real model reproduces the identity matrix (to 2 d.p.)
3. Cross-element correlation length scale for each channel: Δ_e used to model the cross-element correlation with an exponential fit: $r_{e,e'} = \exp(-\frac{|e-e'|}{\Delta_e})$
4. Cross-line correlation length scale for each channel: Δ_l used to model the cross-line correlation with an exponential fit: $r_{l,l'} = \exp(-\frac{|l-l'|}{\Delta_l})$
5. Average cross-element correlation length for each channel and element separation in vector form
6. Average cross-line correlation length for each channel and line separation in vector form

The plots below show the result for the AVHRR orbit semi-real model output for the vector and exponential fits to the correlation scale for 2 different runs:

Case 1: sample every 5 elements across scan, every 19 scanlines along track
Case 2: sample every 5 elements across scan, every 23 scanlines along track



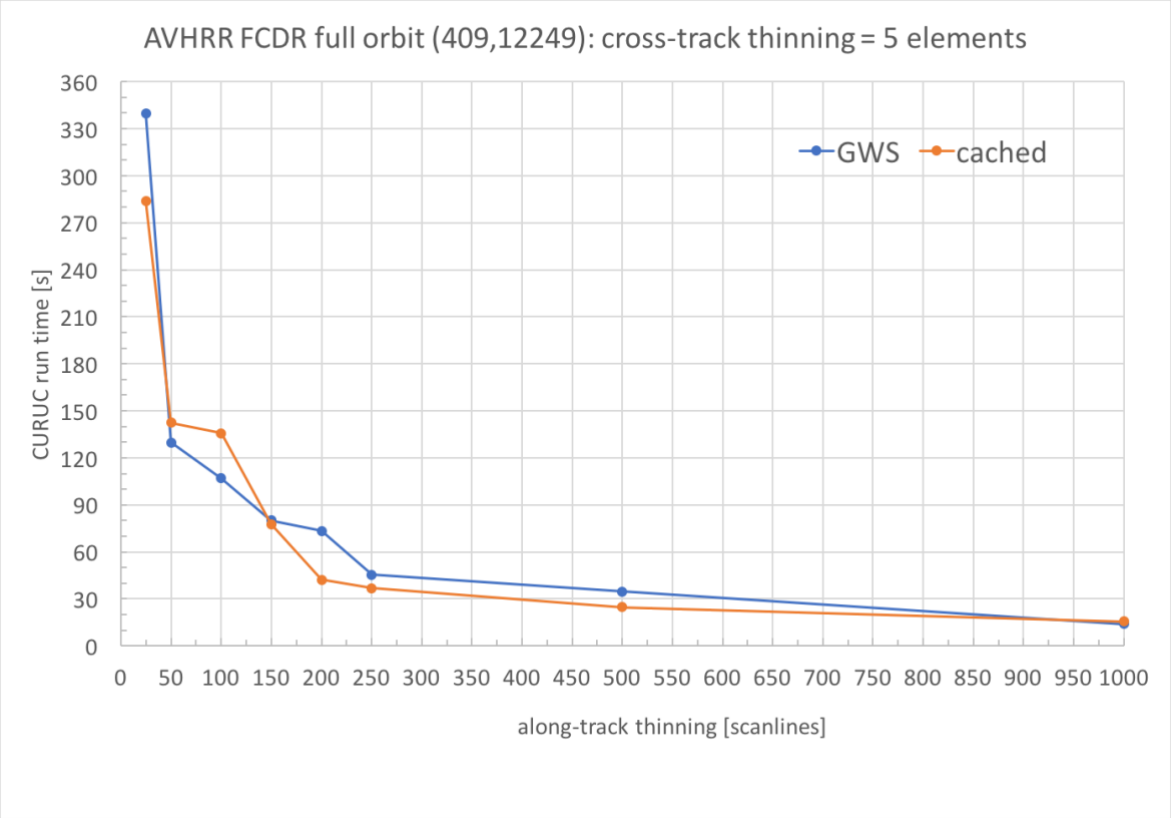


A (single) scalar correlation scale value used in exponential fit may be more user-friendly but does not reveal variability of the cross-element (cross-line) for each channel and element (line) separation. The vector of mean correlation as a function of Δl or Δe appears to reflect better the reality of along-track and cross-track variation in correlation scale properties.

3 RAM and runtime considerations

For large orbit files containing many pixels (n_e, n_l), CURUC runtime can be prohibitively slow in terms of FCDR generation for sensor seires. In particular, the largest error-covariance matrices stored in RAM have dimension […. , n_l,n_l] and depend on the square of the number of lines stored. Because of a limit on peak RAM usage on the lotus queue on CEMS (= 64Gb), it is necessary to optimise the set-up by thinning the (data slicing) the allocated xarrays, i.e. orbit sub-sampling. In order to get a feel for the runtime overhead associated with calculating the cross-channel, cross-line and cross-element correlation scales, a small grid of runs was performed using a full orbit from the Easy AVHRR-FCDR on CEMS (at both the GWS and also cached) for a range of along-track thinning every n scanlines:

cross-track thinning	**along-track thinning [*n*]**	**Runtime [in s] when run from GWS**	**Runtime [in s] when cached at /dev/shm/username/)**
5	25	339.86	283.96
5	50	129.71	142.43
5	100	106.987	135.71
5	150	79.95	77.58
5	200	73.35	42.26
5	250	45.30	36.94
5	500	34.74	24.73
5	1000	13.88	15.54



A large decrease in runtime is observed when along track thinning of $n \geq 50$ scanlines is adopted. CURUC calculation is less than a minute per orbit for the semi-real model example when $n \geq 150$ scanlines.

4 Python (v3.6) code for the semi-real model

```
#!/usr/bin/env python3

from __future__ import print_function

# call as: python3.6 calc_curuc.py filename.nc
# or under ipython3: %run calc_curuc.py filename.nc

# ======================================
# Calculate CURUC for an FCDR orbit file
# ======================================
# Version 0.2
# 24 May, 2018
# michael.taylor AT reading DOT ac DOT uk
# ======================================

import os
import sys
import numpy as np
import numpy.ma as ma
import scipy
import netCDF4
import xarray
import FCDR_HIRS.metrology
from  optparse import OptionParser
from math import *
from netCDF4 import Dataset

def run_curuc(ds,ys,xs):

    us1 = ds['u_structured_Ch3b']
    us2 = ds['u_structured_Ch4']
    us3 = ds['u_structured_Ch5']
    us = [us1,us2,us3]

    ui1 = ds['u_independent_Ch3b']
    ui2 = ds['u_independent_Ch4']
    ui3 = ds['u_independent_Ch5']
```

```
    ui = [ui1,ui2,ui3]

    bitmask_lines = (ds['quality_scanline_bitmask']&1)!=0

    nc = 3
    nx = len(ds.x)
    ny = len(ds.y)
    ns = 1
    ni = 1

    print('IN: allocate_curuc')
    (m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,coords) =
FCDR_HIRS.metrology.allocate_curuc(nc,ny,nx,ns,ni,ys,xs)
    print('OUT: allocate_curuc')

    m9.loc[{'n_s':0}].values[...] = 1
    m12.loc[{'n_i':0}].values[...] = 1
    print('C: allocated')

    for i in range(1,4):
        m5.loc[{'n_c':i,'n_s':0}].values[...] = us[i-1].loc[{'y':coords['n_l'], 'x':coords['n_e']}]
        m8.loc[{'n_c':i,'n_i':0}].values[...] = ui[i-1].loc[{'y':coords['n_l'], 'x':coords['n_e']}]

    print('U: allocated')

    m1.loc[{'n_s':0}].values[...] = 1
    m2.loc[{'n_s':0}].values[...] = 1
    m4.loc[{'n_s':0}].values[...] = 1
    m3.loc[{'n_i':0}].values[...] = np.eye(3)[np.newaxis,np.newaxis,np.newaxis,:,:]

    print('R: allocated')

    mask1 = xarray.DataArray(np.zeros(3,dtype='?'),dims=['n_c'])
    mask2 = bitmask_lines.sel(y=coords['n_l'])

    A =
FCDR_HIRS.metrology.apply_curuc(m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,coords,mask1,mask2,return_vecto
rs=True,interpolate_lengths=True,cutoff_l=ny,cutoff_e=nx)

    return A
```

```
if __name__ == "__main__":
    parser = OptionParser('usage: %filename')
    (options, args) = parser.parse_args()
    filename = args[0]

    ds = xarray.open_dataset(filename)
    xs = 5
    ys = 50
    A = run_curuc(ds,ys,xs)

# Exponential fit using mean correlation scale value for comparison:
y_l = np.exp(-np.abs(np.arange(len(A[4]))) / A[0].sel(n_c=3)[0].values)
y_e = np.exp(-np.abs(np.arange(len(A[5]))) / A[1].sel(n_c=3)[0].values)
```