CNES

IASI

Direction des Systèmes Orbitaux
Systèmes Instrumentaux
Sondage de l'Atmosphère

IA-TN-0000-3274-CNE
Change : 03 Date : 20/11/2019
Issue : 00 Date : 20/11/2019
Ref. : Distribution Code : E

TECHNICAL NOTE

NOISE COVARIANCE MATRIX

Written by :		Date :	
CHINAUD Jordi	DSO/SI /SA	20/11/2019	Signature numérique de CHINAUD Jordi DN : cn=CHINAUD Jordi, o=CNES, ou=DSO/SI/SA, email=jordi.chinaud@cnes.fr, c=FR Date : 2019.11.28 10:30:17 +01'00'
JACQUETTE Elsa	DSO/SI /SA		Signature numérique de Elsa Jacquette DN : cn=Elsa Jacquette, o=CNES, ou=DSO/SI/SA, email=elsa.jacquette@cnes.fr, c=FR Date : 2019.11.28 18:50:10 +01'00'
LE BARBIER Laura	DSO/SI /SA		Signature numérique de Le Barbier Laura DN : c=FR, o=CNES, ou=0002 775665912, ou=Agents CNES, 2.5.4.45=lebarbil@cnes.fr, cn=Le Barbier Laura Date : 2019.11.29 10:49:24 +01'00'
Approved by :		Date :	
PIERANGELO Clemence	DSO/SI /SA	26/11/2019	Signature numérique de Pierangelo Clemence DN : c=FR, o=CNES, ou=0002 775665912, ou=Agents CNES, 2.5.4.45=PierangeloC@cnes.fr, cn=Pierangelo Clemence Date : 2019.11.26 17:22:24 +01'00'
For application :		Date :	
VANDERMARCQ Olivier	DNO/OT /TA		Signature numérique de Olivier Vandermarcq DN : cn=Olivier Vandermarcq, o=CNES, ou=DNO/OT/TA, email=olivier.vandermarcq@cnes.fr, c=FR Date : 2019.11.28 10:24:10 +01'00'

CNES TOULOUSE : 18 av. E.Belin 31401 TOULOUSE Cedex 9

INDEX SHEET

CONFIDENTIALITY : P	KEYWORDS : Noise Covariance Matrix

TITLE :

Technical Note
Noise Covariance Matrix

AUTHOR(S) :

CHINAUD Jordi	DSO/SI /SA
JACQUETTE Elsa	DSO/SI /SA
LE BARBIER Laura	DSO/SI /SA
PEQUIGNOT Eric – DCT/SI/MO (v1.0)	

SUMMARY :

RELATED DOCUMENTS :	LOCALIZATION :

VOLUME : 1	TOTAL NUMBER OF PAGES : 22 INCLUDING PRELIMINARY PAGES : 6 NUMBER OF SUPPL. PAGES : 0	COMPOSITE DOCUMENT : N	LANGUAGE : EN

CONFIGURATION MANAGEMENT : No	CM RESP. :

REASONS FOR EVOLUTION : Update of the content of NCM v2 : adding off-diagonal elements

CONTRACT :

HOST SYSTEM :
Microsoft Word 10.0 (10.0.6838)
\\Bacchus\GDOC\ModeleGDOC_en.dot
Version GDOC : v4.2.0.2
Base projet : \\to05res04\GdocBasesPartagees\Projets\CST\Iasi\IASI

INTERNAL DISTRIBUTION

Name	Entity	Internal Postal Box	Observations
BERMUDO Francois	DSO/OT/IAS	2501	
BES Caroline	DSO/SI/SA	811	
BUFFET Laurence	DSO/SI/IN	3601	
CHINAUD Jordi	DSO/SI/SA	811	
DESCHAMPS Adrien	DSO/SI/SA	811	
DOCUMENTATION IASI	DCT/AQ/GP	2513	
FAILLOT Mathilde	DNO/OT/PE	902	
JACQUETTE Elsa	DSO/SI/SA	811	
JOUGLET Denis	DSO/SI/SA	811	
JURADO Eric	DSO/OT/IAS	2501	
LAFRIQUE Pierre	DSO/SI/SA	811	
LE BARBIER Laura	DSO/SI/SA	811	
LE FEVRE Clemence	DSO/OT/IAS	1324	
LUITOT Clément	DSO/SI/SA	811	
MAKHMARA Hassan	DSO/SI/MO	811	
PENQUER Antoine	DSO/SI/IN	3601	
PEUS Alain	DSO/SI/SA	811	
PIERANGELO Clemence	DSO/SI/SA	811	
PITTET Christelle	DSO/SI/SA	811	
SCHMISSER Roseline	DSO/SI/SA	811	
SYLVANDER Sylvia	DNO/OT/PE	902	
VANDERMARCQ Olivier	DNO/OT/TA	612	

EXTERNAL DISTRIBUTION

Name	Entity	Observations
BAQUE Claire	DNO/OT /PE (AKKA IS) (Claire.Baque@cnes.fr)	
CALVEL Jean-Christophe	DNO/OT /PE (AKKA) (Jean-Christophe.Calvel@cnes.fr)	
COPPENS Dorothée	EUMETSAT (dorothee.coppens@eumetsat.int)	
GUEDJ Stéphanie	EUMETSAT (Stephanie.Guedj@eumetsat.int)	

KANGAH Yannick	DSO/SI/SA (SPASCIA) (yannick.kangah@cnes.fr)
LALANNE Tristan	DSO/SI/SA (NOVELTIS) (Tristan.Lalanne@cnes.fr)
TOURNIER Bernard	DSO/SI/SA (SPASCIA) (bernard.tournier@cnes.fr)
TSCHIMMEL Martin	EUMETSAT (martin.tschimmel@eumetsat.int)
VASQUEZ Mayte	EUMETSAT (Mayte.Vasquez@external.eumetsat.int)
VERNIER Bénédicte	DNO/OT/PE (THALES) (benedicte.vernier@cnes.fr)

CHANGES

Issue	Rev.	Date	Reference, Author(s), Reasons for evolution
03	00	20/11/2019	CHINAUD Jordi DSO/SI /SA JACQUETTE Elsa DSO/SI /SA LE BARBIER Laura DSO/SI /SA Update of the content of NCM v3 : updating parameters
02	00	30/11/2018	CHINAUD Jordi DSO/SI /SA JACQUETTE Elsa DSO/SI /SA LE BARBIER Laura DSO/SI /SA Update of the content of NCM v2 : updating parameters
01	00	26/09/2008	PEQUIGNOT Eric DCT/SI /MO Creation of the document

TABLE OF CONTENTS

GLOSSARY AND LIST OF TBC AND TBD ITEMS

CGS	Core Ground Segment
EPS	Eumetsat Polar System
EUMETSAT	EUropean organisation for the exploitation of METeorological SATellites
IASI	Infrared Atmospheric Sounding Interferometer
OPS	Operational Processing Software (Level 1 IASI data processing)
TEC	Technical Expertise Center
UTC	Universal Time Coordinates

List of TBC items:

List of TBD items:

1. OVERVIEW

1.1. REFERENCE DOCUMENTS

RD1 IASI TEC/CGS and GIF Interface Control Document
24/08/2009, Issue 1, Rev. 5P
IA-ID-2200-2609-CNE

RD2 Dossier de définition des algorithmes IASI
JACQUETTE Elsa, MARALDI Claire, 15/06/2015, Issue 6, Rev. 9
IA-DF-0000-2006-CNE

RD3 IASI L1C Noise Covariance Matrix
KANGAH Yannick, TOURNIER Bernard, 21/03/2018, Issue 1, Rev. 0
SPA-018-TN-001

1.2. APPLICABLE DOCUMENTS

1.3. INTRODUCTION

Section 2 describes the principal and rationale of the Noise Covariance Matrix (NCM).

Section 3 describes the binary files which contains the noise covariance matrix. An example of Python reader is delivered to EUMETSAT with the noise covariance matrix files.

2. NOISE COVARIANCE MATRIX

2.1. INTRODUCTION

Let's note:

$S^t(\nu)$ the theoretical spectrum convolved by the instrument spectral response functions (ISRF)

$S^m(\nu)$ the spectrum calculated by on-board and ground processing

We can write:

$$S^m(\nu_i) = S^t(\nu_i) + \eta_i(S^t) + \varepsilon_i \quad (1)$$

In equation (1) we have split the errors into:

$\eta_i(S^t)$ which depend on the measure itself or are varying slowly. These errors cannot be considered as noise and are taken into account by all the calibration scheme

ε_i which are, on the contrary, really white noise

The noise covariance matrix is a characteristic of errors distribution ε_i (amplitude and correlation between errors observed on different channels for the same spectrum). By definition we have

$$C = E(\varepsilon\varepsilon^T) \quad (2)$$

with:

C the Noise Covariance Matrix, $\varepsilon^T = [\varepsilon_1\ \varepsilon_2\ \ldots\ \varepsilon_N]$, N the number of IASI channels (8461), E the mathematical mean operator.

Note that:

- Diagonal coefficients of C are dominated by instrument radiometric noise.

- There are 2 Noise Covariance Matrices. The first one related to L1B, the second one to L1C.

2.2. COMPRESSED REPRESENTATION OF THE NCM

The NCM is delivered by the TEC using a compressed representation. For the sake of this compression the NCM (C) is decomposed as followed:

$$C = C' + C'' \tag{3}$$

where:

C' is a band diagonal matrix which lower bandwidth and upper bandwidth are equal to 4 (diagonal + 4 upper/lower extra-diagonal elements)

$$C' = \begin{pmatrix} C_{11} & \cdots & C_{15} & 0 & \cdots & 0 \\ \vdots & C_{22} & \cdots & C_{26} & \ddots & \vdots \\ C_{51} & \vdots & \ddots & \vdots & \ddots & 0 \\ 0 & C_{62} & \vdots & \ddots & \vdots & C_{(n-4)n} \\ \vdots & \ddots & \ddots & \vdots & C_{(n-1)(n-1)} & \vdots \\ 0 & \cdots & 0 & C_{n(n-4)} & \cdots & C_{nn} \end{pmatrix}$$

n is the number of IASI channels;

C'' is the matrix of the extra diagonal elements.

$$C'' = \begin{pmatrix} 0 & \cdots & 0 & C_{16} & \cdots & n \\ \vdots & 0 & \cdots & \ddots & \ddots & \vdots \\ 0 & \vdots & \ddots & \vdots & \ddots & C_{(n-5)n} \\ C_{61} & \ddots & \vdots & \ddots & \vdots & 0 \\ \vdots & \ddots & \ddots & \vdots & 0 & \vdots \\ C_{n1} & \cdots & C_{n(n-5)} & 0 & \cdots & 0 \end{pmatrix}$$

C'' can be decomposed as followed:

$$C'' = VDV^T \tag{4}$$

where:

$$V = \begin{pmatrix} V_1^1 & \cdots & V_1^{neival} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ V_n^1 & \cdots & V_n^{neival} & 0 & \cdots & 0 \end{pmatrix}$$

$$D = \begin{pmatrix} \lambda_1 & 0 & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & \vdots & \cdots & \cdots & 0 \\ 0 & \cdots & \lambda_{neival} & 0 & \cdots & \vdots \\ \vdots & 0 & 0 & 0 & \cdots & \vdots \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 \end{pmatrix}$$

neival is the number of significant eigen values;

V is the matrix of right (column) eigen vectors (V^T is the transposed matrix of V) corresponding to the eigen values and

D is a diagonal matrix whose diagonal elements are the significant eigen values corresponding to the eigen vectors.

neival is currently fixed to 2 to describe C" using the eigen decomposition.

2.3. ESTIMATION OF NCM

The C matrix is estimated as follows:

1) IASI Full Noise Covariance Matrix is estimated using the hot Black Body spectra from External Calibration measurements in flight. L1B spectra are used for NCM 1B, L1C spectra are used for NCM 1C.

Note that due to the in-flight variation of Black Body temperature, each L1C spectrum is scaled with respect to the ratio between the Planck function at a reference temperature and the Planck function applied during the post-radiometric calibration in the level 1 processing:

$$S_{Sc}(v,N) = \frac{Planck(v,T_{Ref})}{Planck(v,T(N))} S(v,N)$$

Where $S_{Sc}(v,N)$ is the scaled spectrum at the wave number v and for the scan line LN (N);

$S(v,N)$ is the original L1B or L1C spectrum;

$T(N)$ is the filtered temperature of the BB at the scan line LN(N);

T_{Ref} =$\overline{T(N)}$ is the average temperature over the collection;

$Planck(v,T_{Ref})$ and $Planck(v,T(N))$ are the Planck function at v corresponding to the temperatures T_{Ref} and $T(N)$, respectively.

Finally, the NCM is performed as the variance-covariance matrix of the scaled spectra S_{sc}.

For i from 1 to n,

For j from i to n:

C(i,j)=Covariance(S_{sc}(i), S_{sc}(j))

C(j,i)=C(i,j)

2) The 5 vectors of the diagonal band (diagonal+4 upper diagonal vectors) of C are stored in C'.

   NB : as the covariance matrix is symmetric, C(j,i)=C(i,j), the lower diagonal coefficients are equal to the upper diagonal coefficients, so only the upper diagonal coefficients are stored.

3) We perfom eigen decomposition of C'' = C – C'.

The C" neival largest eigen values of λ_1, …, λ_{neival} and the associated eigen vectors V^1, …, V^{neival} are computed.

Currently, we consider only the 2 largest eigen values of C" and their 2 associated eigen vectors.

4) We verify that the reconstructed NCM using the two largest eigen values of C" and their associated eigen vectors lead to a negligible error (rmse error of about 0.01 in correlation).

3. DESCRIPTION OF BINARY FILE

3.1. DEFINITION

This file contains the values associated to the IASI Noise Covariance Matrix updated by the TEC.

File naming convention:

Each file will comply with the following naming convention:

IASI_NCM_xx_M0x_YYYYMMDDHHMMSSZ_YYYYMMDDHHMMSSZ_YYYYMMDDHHMMSSZ_IAST_SPECTRESPO where:

- xx for the pixel number 01, 02, 03 or 04
- M0x= M01 for METOP-B, M02 for METOP-A, M03 for METOP-C
- the first date is the UTC time Start of validity: it the date when the matrix is considered as applicable by the TEC
- the second date is the UTC time Stop of validity: the first and second date are the same (this second date is not used by the TEC)
- the third date is the creation date of the file

3.2. FILE STRUCTURE

FIELD	DESCRIPTION	UNITS	DIM1	DIM2	DIM3	DIM4	DIM5	DIM6	TYPE	TYPE SIZE	FIELD SIZE	
IDefIssueIcd	Issue number of the format of the configuration files	n/a	1	1	1	1	1	1	Integer4	4	4	0
IDefRevisionIcd	Revision number of the format of the configuration files	n/a	1	1	1	1	1	1	Integer4	4	4	4
IDefCovID	Identification of this Covariance Matrix	n/a	1	1	1	1	1	1	Integer4	4	4	8
IDefCovDate	Time Start of validity of the IASI NCM	Date	1	1	1	1	1	1	Date	8	8	12
IDefCovarMatEigenVal1b	Eigen Values vector - Level 1b	(W/m²/st/m-1)²	2	100	1	1	1	1	Real64	8	1 600	20
IDefCovarMatEigenVal1c	Eigen Values vector - Level 1c	(W/m²/st/m-1)²	2	100	1	1	1	1	Real64	8	1 600	1 620
IDefRnmSize1_1b	Covariance Matrix 1b : first part size (diagonal vectors)	n/a	1	1	1	1	1	1	Integer 4	4	4	3 220
IDefRnmSize2_1b	Covariance Matrix 1b : second part size (eigen vectors)	n/a	1	1	1	1	1	1	Integer 4	4	4	3 224
IDefRnmSize1_1c	Covariance Matrix 1c : first part size (diagonal vectors)	n/a	1	1	1	1	1	1	Integer 4	4	4	3 228
IDefRnmSize2_1c	Covariance Matrix 1c : second part size (eigen vectors)	n/a	1	1	1	1	1	1	Integer 4	4	4	3 232
IRnmRadNoiseCovarMatDiagonalVectors1b	Covariance Matrix Level 1b : diagonal vectors	(W/m²/st/m-1)²	8500	50	1	1	1	1	Real32	4	1,70E+06	3,24E+03
IRnmRadNoiseCovarMatEigenVectors1b	Covariance Matrix Level 1b : eigen vectors	(W/m²/st/m-1)²	8500	50	1	1	1	1	Real32	4	1,70E+06	1,70E+06
IRnmRadNoiseCovarMatDiagonalVectors1c	Covariance Matrix Level 1c : diagonal vectors	(W/m²/st/m-1)²	8500	50	1	1	1	1	Real32	4	1,70E+06	3,40E+06
IRnmRadNoiseCovarMatEigenVectors1c	Covariance Matrix Level 1b : eigen vectors	(W/m²/st/m-1)²	8500	50	1	1	1	1	Real32	4	1,70E+06	5,10E+06

Corresponding variables:

IDefCovarMatEigenVal1b	Eigen values λ_1, …, λ_{neival} for L1B Dim 1 = 2 (for the 2 Cube Corner Direction)
IDefCovarMatEigenVal1c	Eigen values λ_1, …, λ_{neival} for L1C Dim 1 = 2 (for the 2 Cube Corner Direction)
IDefRnmSize1_1b	5
IDefRnmSize2_1b	neival = 2
IDefRnmSize1_1c	5
IDefRnmSize2_1c	neival = 2
IRnmRadNoiseCovarMatDiagonalVectors1b	5 vectors of the diagonal band for L1B: diagonal (1st column) + 4 upper diagonal vectors: 1st column = diagonal vector of C for L1B 2nd column = 1st upper diagonal vector of C for L1B 3rd column = 2nd upper diagonal vector of C for L1B 4th column = 3rd upper diagonal vector of C for L1B 5th column = 4th upper diagonal vector of C for L1B
IRnmRadNoiseCovarMatEigenVectors1b	Neival eigen vectors V^1, …, V^{neival} for L1B 1st column = 1st eigen vector V^1 for L1B 2nd column = 2nd eigen vector $V^{neival} = V^2$ for L1B
IRnmRadNoiseCovarMatDiagonalVectors1c	5 vectors of the diagonal band for L1C: diagonal (1st column) + 4 upper diagonal vectors: 1st column = diagonal vector of C for L1C 2nd column = 1st upper diagonal vector of C for L1C 3rd column = 2nd upper diagonal vector of C for L1C 4th column = 3rd upper diagonal vector of C for L1C 5th column = 4th upper diagonal vector of C for L1C
IRnmRadNoiseCovarMatEigenVectors1c	Neival eigen vectors V^1, …, V^{neival} for L1C 1st column = 1st eigen vector V^1 for L1C 2nd column = 2nd eigen vector $V^{neival} = V^2$ for L1C

3.3. PYTHON READER TOOL

The NCM 1C is provided to EUMETSAT as well as a Python tool to properly read it. The reader program has been tested on Linux system.

The package contains sample readers for NCM products in Python. It is intended solely to show how NCM data can be read, and is not a supported TEC product.

The prerequisite concerning Python libraries are the following:

Python version: 2.7.5

and following modules:

- ✓ datetime
- ✓ math
- ✓ matplotlib 1.2.0
- ✓ numpy 1.7.1

We give hereafter a copy of the python script file ncm_format.py:

```
# -*- coding: utf-8 -*-
"""
ncm_format.py python script

usage:
python ncm_format.py formatted_fname
where:
- formatted_fname is 6803236 bytes input file

Main program is only an example and has only stdout ouputs.

higher level function is:
read_ncm_compute_nedt : read NCM structure from formatted binary file, keep only 1 diagonal vector (there are 5)
and compute nedt, reconstruct covariance matrix from this NCM structure read from formatted binary file (only keep
covariance output, not separated diagonal and extradiagonal parts), compute nedt for diagonal elements; nedt
results are the same (for diagonal elements)

usefull user functions are :
ncm_format_write : write NCM structure in formatted binary file
ncm_format_read : read NCM structure from previously written formatted binary file
reconstruct_cov_from_struct_ncm : reconstruct covariance matrix from this NCM structure

first developper function is :
load_struct_ncm : load NCM structure from npz files (eig_decompos.py output, reconstruct_NCM_1{B,C}_PN1.npz
for example)
```

```
NCM structure is 6803236 bytes one defined in IA-TN-0000-3274-CNE 02.00 (remains big endian)
with one change : IRnmRadNoiseCovarMat1{b,c}  are splited into IRnmRadNoiseCovarMatDiagonalVectors1b{b,c}
(5 diagonal vectors) / IRnmRadNoiseCovarMatDiagonalVectorsEigenVectors1b{b,c} (2 eigenvectors)

sotware prerequisites :
- python 2.7.5 and following modules :
 - datetime
 - math
 - matplotlib 1.2.0
 - numpy 1.7.1

"""

import matplotlib.pyplot as plt
import numpy as np
from sys import argv
import struct
import math

# for planck : begin
pi = np.pi
Cst_h = 6.6260755e-34    # Joules*Secondes
Cst_c = 2.99792458e+8    # m/s
Cst_k = 1.380658e-23     # Joules/Kelvin
Cst_sca1  = 2*Cst_h*Cst_c**2
Cst_sca2  = Cst_h*Cst_c/Cst_k
# for planck : end

# empty 6803236 bytes structure : begin
STRUCT_NCM_EMPTY={
  'IDefIssueIcd':-1,
  'IDefRevisionIcd' : -2,
  'IDefCovID' : -3,
  'IDefCovDate_day' : -4,
  'IDefCovDate_ms' : -5,
  'IDefCovarMatEigenVal1b' : np.zeros((2,100),dtype=np.double),
  'IDefCovarMatEigenVal1c' : np.zeros((2,100),dtype=np.double),
  'IDefRnmSize1_1b' : 5,
  'IDefRnmSize2_1b' : 2,
  'IDefRnmSize1_1c' : 5,
  'IDefRnmSize2_1c' : 2,
  'IRnmRadNoiseCovarMatDiagonalVectors1b' : np.zeros((8500,50),dtype=np.float),
  'IRnmRadNoiseCovarMatEigenVectors1b' : np.zeros((8500,50),dtype=np.float),
  'IRnmRadNoiseCovarMatDiagonalVectors1c' : np.zeros((8500,50),dtype=np.float),
  'IRnmRadNoiseCovarMatEigenVectors1c' : np.zeros((8500,50),dtype=np.float),
  }
# empty 6803236 bytes structure : end

def ncm_format_write(fname_output, struct_ncm, verbose=True):
  """
  write struct_ncm in fname_output file
  """
  idfct="[ncm_format_write]"
  if verbose: print idfct, "begin"
```

```
  if verbose: print idfct, "fname_output=", fname_output
  fout = open(fname_output, 'wb')
  if verbose: print idfct, "struct_ncm['IDefIssueIcd']", struct_ncm['IDefIssueIcd']
  fout.write(struct.pack('>i', struct_ncm['IDefIssueIcd']))
  if verbose: print idfct, "struct_ncm['IDefRevisionIcd']", struct_ncm['IDefRevisionIcd']
  fout.write(struct.pack('>i', struct_ncm['IDefRevisionIcd']))
  if verbose: print idfct, "struct_ncm['IDefCovID']", struct_ncm['IDefCovID']
  fout.write(struct.pack('>i', struct_ncm['IDefCovID']))
  if verbose: print idfct, "struct_ncm['IDefCovDate_day']", struct_ncm['IDefCovDate_day']
  fout.write(struct.pack('>i', struct_ncm['IDefCovDate_day']))
  if verbose: print idfct, "struct_ncm['IDefCovDate_ms']", struct_ncm['IDefCovDate_ms']
  fout.write(struct.pack('>i', struct_ncm['IDefCovDate_ms']))
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(struct_ncm['IDefCovarMatEigenVal1b'])),
"IDefCovarMatEigenVal1b", struct_ncm['IDefCovarMatEigenVal1b'].shape,
(np.nanmax(struct_ncm['IDefCovarMatEigenVal1b'])))
    print idfct, my_str
  for i in range(struct_ncm['IDefCovarMatEigenVal1b'].shape[0]):
    for j in range(struct_ncm['IDefCovarMatEigenVal1b'].shape[1]):
      fout.write(struct.pack('>d', struct_ncm['IDefCovarMatEigenVal1b'][i,j]))
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(struct_ncm['IDefCovarMatEigenVal1c'])),
"IDefCovarMatEigenVal1c", struct_ncm['IDefCovarMatEigenVal1c'].shape,
(np.nanmax(struct_ncm['IDefCovarMatEigenVal1c'])))
    print idfct, my_str
  for i in range(struct_ncm['IDefCovarMatEigenVal1c'].shape[0]):
    for j in range(struct_ncm['IDefCovarMatEigenVal1c'].shape[1]):
      fout.write(struct.pack('>d', struct_ncm['IDefCovarMatEigenVal1c'][i,j]))
  if verbose: print idfct, "struct_ncm['IDefRnmSize1_1b']", struct_ncm['IDefRnmSize1_1b']
  fout.write(struct.pack('>i', struct_ncm['IDefRnmSize1_1b']))
  if verbose: print idfct, "struct_ncm['IDefRnmSize2_1b']", struct_ncm['IDefRnmSize2_1b']
  fout.write(struct.pack('>i', struct_ncm['IDefRnmSize2_1b']))
  if verbose: print idfct, "struct_ncm['IDefRnmSize1_1c']", struct_ncm['IDefRnmSize1_1c']
  fout.write(struct.pack('>i', struct_ncm['IDefRnmSize1_1c']))
  if verbose: print idfct, "struct_ncm['IDefRnmSize2_1c']", struct_ncm['IDefRnmSize2_1c']
  fout.write(struct.pack('>i', struct_ncm['IDefRnmSize2_1c']))
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1b'])),
"IRnmRadNoiseCovarMatDiagonalVectors1b", struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1b'].shape,
(np.nanmax(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1b'])))
    print idfct, my_str
  for i in range(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1b'].shape[0]):
    for j in range(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1b'].shape[1]):
      fout.write(struct.pack('>f', struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1b'][i,j]))
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1b'])),
"IRnmRadNoiseCovarMatEigenVectors1b", struct_ncm['IRnmRadNoiseCovarMatEigenVectors1b'].shape,
(np.nanmax(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1b'])))
    print idfct, my_str
  for i in range(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1b'].shape[0]):
    for j in range(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1b'].shape[1]):
      fout.write(struct.pack('>f', struct_ncm['IRnmRadNoiseCovarMatEigenVectors1b'][i,j]))
  if verbose :
```

```
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1c'])),
"IRnmRadNoiseCovarMatDiagonalVectors1c", struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1c'].shape,
(np.nanmax(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1c'])))
    print idfct, my_str
  for i in range(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1c'].shape[0]):
    for j in range(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1c'].shape[1]):
      fout.write(struct.pack('>f', struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1c'][i,j]))
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1c'])),
"IRnmRadNoiseCovarMatEigenVectors1c", struct_ncm['IRnmRadNoiseCovarMatEigenVectors1c'].shape,
(np.nanmax(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1c'])))
    print idfct, my_str
  for i in range(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1c'].shape[0]):
    for j in range(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1c'].shape[1]):
      fout.write(struct.pack('>f', struct_ncm['IRnmRadNoiseCovarMatEigenVectors1c'][i,j]))
  fout.close()
  if verbose: print idfct, "end"


def ncm_format_read(fname_input, verbose=True):
  """
  read struct_ncm frim fname_input file
  """
  idfct="[ncm_format_read]"
  if verbose: print idfct, "begin"
  if verbose: print idfct, "fname_input=", fname_input
  struct_ncm = STRUCT_NCM_EMPTY
  fin = open(fname_input, 'rb')
  fileContent = fin.read()
  fin.close()
  offset=0
  mysize=4
  if verbose: print idfct, "offset", offset
  struct_ncm['IDefIssueIcd'] = struct.unpack('>i',fileContent[offset:offset+mysize])[0]
  if verbose: print idfct, "struct_ncm['IDefIssueIcd']", struct_ncm['IDefIssueIcd']
  offset=offset+mysize
  if verbose: print idfct, "offset", offset
  struct_ncm['IDefRevisionIcd'] = struct.unpack('>i',fileContent[offset:offset+mysize])[0]
  if verbose: print idfct, "struct_ncm['IDefRevisionIcd']", struct_ncm['IDefRevisionIcd']
  offset=offset+mysize
  if verbose: print idfct, "offset", offset
  struct_ncm['IDefCovID'] = struct.unpack('>i',fileContent[offset:offset+mysize])[0]
  if verbose: print idfct, "struct_ncm['IDefCovID']", struct_ncm['IDefCovID']
  offset=offset+mysize
  if verbose: print idfct, "offset", offset
  struct_ncm['IDefCovDate_day'] = struct.unpack('>i',fileContent[offset:offset+mysize])[0]
  if verbose: print idfct, "struct_ncm['IDefCovDate_day']", struct_ncm['IDefCovDate_day']
  offset=offset+mysize
  if verbose: print idfct, "offset", offset
  struct_ncm['IDefCovDate_ms'] = struct.unpack('>i',fileContent[offset:offset+mysize])[0]
  if verbose: print idfct, "struct_ncm['IDefCovDate_ms']", struct_ncm['IDefCovDate_ms']
  verbose2 = False
  for i in range(struct_ncm['IDefCovarMatEigenVal1b'].shape[0]):
    verbose2 = verbose
    for j in range(struct_ncm['IDefCovarMatEigenVal1b'].shape[1]):
```

```
      offset=offset+mysize
      mysize=8
      struct_ncm['IDefCovarMatEigenVal1b'][i,j] = struct.unpack('>d',fileContent[offset:offset+mysize])[0]
      if verbose2 :
        print idfct, "offset", offset
        my_str= 'IDefCovarMatEigenVal1b[' + str(i) +  ',' + str(j) +  ']=' + str(  struct_ncm['IDefCovarMatEigenVal1b'][i,j])
        print idfct, my_str
      if j>5: verbose2 = False
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(struct_ncm['IDefCovarMatEigenVal1b'])),
"IDefCovarMatEigenVal1b", struct_ncm['IDefCovarMatEigenVal1b'].shape,
(np.nanmax(struct_ncm['IDefCovarMatEigenVal1b'])))
    print idfct, my_str
  for i in range(struct_ncm['IDefCovarMatEigenVal1c'].shape[0]):
    verbose2 = verbose
    for j in range(struct_ncm['IDefCovarMatEigenVal1c'].shape[1]):
      offset=offset+mysize
      mysize=8
      struct_ncm['IDefCovarMatEigenVal1c'][i,j] = struct.unpack('>d',fileContent[offset:offset+mysize])[0]
      if verbose2 :
        print idfct, "offset", offset
        my_str= 'IDefCovarMatEigenVal1c[' + str(i) +  ',' + str(j) +  ']=' + str(  struct_ncm['IDefCovarMatEigenVal1c'][i,j])
        print idfct, my_str
      if j>5: verbose2 = False
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(struct_ncm['IDefCovarMatEigenVal1c'])),
"IDefCovarMatEigenVal1c", struct_ncm['IDefCovarMatEigenVal1c'].shape,
(np.nanmax(struct_ncm['IDefCovarMatEigenVal1c'])))
    print idfct, my_str
  offset=offset+mysize
  if verbose: print idfct, "offset", offset
  mysize=4
  struct_ncm['IDefRnmSize1_1b'] = struct.unpack('>i',fileContent[offset:offset+mysize])[0]
  if verbose: print idfct, "struct_ncm['IDefRnmSize1_1b']", struct_ncm['IDefRnmSize1_1b']
  offset=offset+mysize
  if verbose: print idfct, "offset", offset
  mysize=4
  struct_ncm['IDefRnmSize2_1b'] = struct.unpack('>i',fileContent[offset:offset+mysize])[0]
  if verbose: print idfct, "struct_ncm['IDefRnmSize2_1b']", struct_ncm['IDefRnmSize2_1b']
  offset=offset+mysize
  if verbose: print idfct, "offset", offset
  mysize=4
  struct_ncm['IDefRnmSize1_1c'] = struct.unpack('>i',fileContent[offset:offset+mysize])[0]
  if verbose: print idfct, "struct_ncm['IDefRnmSize1_1c']", struct_ncm['IDefRnmSize1_1c']
  offset=offset+mysize
  if verbose: print idfct, "offset", offset
  mysize=4
  struct_ncm['IDefRnmSize2_1c'] = struct.unpack('>i',fileContent[offset:offset+mysize])[0]
  if verbose: print idfct, "struct_ncm['IDefRnmSize2_1c']", struct_ncm['IDefRnmSize2_1c']
  verbose2 = False
  for i in range(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1b'].shape[0]):
    for j in range(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1b'].shape[1]):
      if j < 5 : verbose2 = verbose and i==0
      offset=offset+mysize
      mysize=4
```

```
      struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1b'][i,j] =
struct.unpack('>f',fileContent[offset:offset+mysize])[0]
      if verbose2 :
        print idfct, "offset", offset
        my_str= 'IRnmRadNoiseCovarMatDiagonalVectors1b[' + str(i) +  ',' + str(j) +  ']=' + str(
struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1b'][i,j])
        print idfct, my_str
      verbose2 = False
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1b'])),
"IRnmRadNoiseCovarMatDiagonalVectors1b", struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1b'].shape,
(np.nanmax(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1b'])))
    print idfct, my_str
  verbose2 = False
  for i in range(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1b'].shape[0]):
    for j in range(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1b'].shape[1]):
      if j < 2 : verbose2 = verbose and i==0
      offset=offset+mysize
      mysize=4
      struct_ncm['IRnmRadNoiseCovarMatEigenVectors1b'][i,j] =
struct.unpack('>f',fileContent[offset:offset+mysize])[0]
      if verbose2 :
        print idfct, "offset", offset
        my_str= 'IRnmRadNoiseCovarMatEigenVectors1b[' + str(i) +  ',' + str(j) +  ']=' + str(
struct_ncm['IRnmRadNoiseCovarMatEigenVectors1b'][i,j])
        print idfct, my_str
      verbose2 = False
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1b'])),
"IRnmRadNoiseCovarMatEigenVectors1b", struct_ncm['IRnmRadNoiseCovarMatEigenVectors1b'].shape,
(np.nanmax(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1b'])))
    print idfct, my_str
  verbose2 = False
  for i in range(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1c'].shape[0]):
    for j in range(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1c'].shape[1]):
      if j < 5 : verbose2 = verbose and i==0
      offset=offset+mysize
      mysize=4
      struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1c'][i,j] =
struct.unpack('>f',fileContent[offset:offset+mysize])[0]
      if verbose2 :
        print idfct, "offset", offset
        my_str= 'IRnmRadNoiseCovarMatDiagonalVectors1c[' + str(i) +  ',' + str(j) +  ']=' + str(
struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1c'][i,j])
        print idfct, my_str
      verbose2 = False
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1c'])),
"IRnmRadNoiseCovarMatDiagonalVectors1c", struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1c'].shape,
(np.nanmax(struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors1c'])))
    print idfct, my_str
  verbose2 = False
  for i in range(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1c'].shape[0]):
    for j in range(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1c'].shape[1]):
      if j < 2 : verbose2 = verbose and i==0
```

```
      offset=offset+mysize
      mysize=4
      struct_ncm['IRnmRadNoiseCovarMatEigenVectors1c'][i,j] =
struct.unpack('>f',fileContent[offset:offset+mysize])[0]
      if verbose2 :
        print idfct, "offset", offset
        my_str= 'IRnmRadNoiseCovarMatEigenVectors1c[' + str(i) +  ',' + str(j) +  ']=' + str(
struct_ncm['IRnmRadNoiseCovarMatEigenVectors1c'][i,j])
        print idfct, my_str
      verbose2 = False
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1c'])),
"IRnmRadNoiseCovarMatEigenVectors1c", struct_ncm['IRnmRadNoiseCovarMatEigenVectors1c'].shape,
(np.nanmax(struct_ncm['IRnmRadNoiseCovarMatEigenVectors1c'])))
    print idfct, my_str
  return struct_ncm
  if verbose: print idfct, "end"



def load_struct_ncm(npz_fname_1b, npz_fname_1c, verbose=True):
  """
  read  npz file (eig_decompos.py output, reconstruct_NCM_PN1.npz for example)
  return struct_ncm, cov_red_1b, cov_red_1c, w_cov_1b, w_cov_1c, v_cov_1b and v_cov_1c
  only cov_red_1b and  cov_red_1c are needed, others are for check

  note that struct_ncm elements are converted from (W/m2/st/cm-1)2 to (W/m2/st/m-1)2, and others outputs too
  """
  idfct="[load_struct_ncm]"
  if verbose: print idfct, "begin"
  if verbose: print idfct, "npz_fname_1b=", npz_fname_1b
  if verbose: print idfct, "npz_fname_1c=", npz_fname_1c
  struct_ncm = STRUCT_NCM_EMPTY
  # erase index1 and index2 from npz_lut : unused and may be null #'index1':'arr_4', 'index2':'arr_5'
  npz_lut={'cov':'arr_0','cov_red':'arr_1','cov_norm':'arr_2','cov_red_norm':'arr_3','v_cov':'arr_6','w_cov':'arr_7'}
  levels=['1b','1c']
  npz_fnames= {'1b':npz_fname_1b,'1c':npz_fname_1c}
  cov_red= {'1b':None,'1c':None}
  w_cov= {'1b':None,'1c':None}
  v_cov= {'1b':None,'1c':None}
  # levels loop : begin
  for level in levels:
    if verbose: print idfct, "level=", level
    npz_fname=npz_fnames[level]
    if verbose: print idfct, "npz_fname=", npz_fname
    npzfile=np.load(npz_fname)
    #np.savez(fname_output,cov, cov_red, cov_norm, cov_red_norm, index1, index2, v_cov, w_cov)
    if verbose:
      #for k in sorted(npzfile.keys()):
        #print idfct, ":", k,  type(npzfile[k]), npzfile[k].shape
          for k in sorted(npz_lut.keys()):
            #print idfct, ":", k, type(npzfile[npz_lut[k]]), npzfile[npz_lut[k]].shape
            try:
```

```
            my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(npzfile[npz_lut[k]])), k, npzfile[npz_lut[k]].shape,
(np.nanmax(npzfile[npz_lut[k]])))
          except:
            my_str= "%s (%s)" % (k, npzfile[npz_lut[k]].shape)
          print idfct, my_str
    IDefCovarMatEigenVal1 = np.zeros((2,100),dtype=np.double)
    for icd in range(2):
      for iev in range(2):
        IDefCovarMatEigenVal1[icd,iev] = npzfile[npz_lut['v_cov']][8460-iev] * 0.0001 #from (W/m2/st/cm-1)2 to
(W/m2/st/m-1)2
    struct_ncm['IDefCovarMatEigenVal%s' % level] = IDefCovarMatEigenVal1
    struct_ncm['IDefRnmSize1_%s' % level] = 5
    struct_ncm['IDefRnmSize2_%s' % level] = 2
    IRnmRadNoiseCovarMatDiagonalVectors1 = np.zeros((8500,50),dtype=np.float)
    for idiag in range(struct_ncm['IDefRnmSize1_%s' % level]): # 5 diagonals
      IRnmRadNoiseCovarMatDiagonalVectors1[0:8461-idiag,idiag] = np.diag(npzfile[npz_lut['cov']],idiag) * 0.0001
#from (W/m2/st/cm-1)2 to (W/m2/st/m-1)2
    struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors%s' % level] = IRnmRadNoiseCovarMatDiagonalVectors1
    IRnmRadNoiseCovarMatEigenVectors1 = np.zeros((8500,50),dtype=np.float)
    for iev in range(struct_ncm['IDefRnmSize2_%s' % level]): # 2 last eigenvectors
      IRnmRadNoiseCovarMatEigenVectors1[0:8461,iev] = npzfile[npz_lut['w_cov']][0:8461,8460-iev]  #no * 0.0001
#from (W/m2/st/cm-1)2 to (W/m2/st/m-1)2
    struct_ncm['IRnmRadNoiseCovarMatEigenVectors%s' % level] = IRnmRadNoiseCovarMatEigenVectors1
    cov_red[level]=npzfile[npz_lut['cov_red']]  * 0.0001 #from (W/m2/st/cm-1)2 to (W/m2/st/m-1)2
    w_cov[level]=npzfile[npz_lut['w_cov']]  #no * 0.0001 #from (W/m2/st/cm-1)2 to (W/m2/st/m-1)2
    v_cov[level]=npzfile[npz_lut['v_cov']]  * 0.0001 #from (W/m2/st/cm-1)2 to (W/m2/st/m-1)2
  # levels loop : end
  if verbose: print idfct, "end"
  return struct_ncm, cov_red['1b'], cov_red['1c'], w_cov['1b'], w_cov['1c'], v_cov['1b'], v_cov['1c']

def reconstruct_cov_from_struct_ncm(struct_ncm, level, verbose=True):
  """
  reconstruct covariance matrix from struct_ncm
  return 3 matrices : cov (=covband+covextra), covband (diagonal), covextra (extradiagonal)
  """
  idfct="[reconstruct_cov_from_struct_ncm]"
  if verbose: print idfct, "begin"
  if verbose: print idfct, "level", level
  assert(level in ['1b', '1c'])
  cov = np.zeros((8461,8461))
  covband = np.zeros((8461,8461))
  w_cov = np.zeros((8461,8461))
  v_cov = np.zeros(8461)
  IDefCovarMatEigenVal1 = struct_ncm['IDefCovarMatEigenVal%s' % level]
  IRnmRadNoiseCovarMatDiagonalVectors1 = struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors%s' % level]
  IRnmRadNoiseCovarMatEigenVectors1 = struct_ncm['IRnmRadNoiseCovarMatEigenVectors%s' % level]
  for iev in range(struct_ncm['IDefRnmSize2_%s' % level]): # 2 last eigenvectors
    v_cov[8460-iev]=IDefCovarMatEigenVal1[0,iev] #icd=0
    w_cov[0:8461,8460-iev]=IRnmRadNoiseCovarMatEigenVectors1[0:8461,iev]
  # covband
  nb_diag = struct_ncm['IDefRnmSize1_%s' % level]
  for idiag in range(nb_diag): # 5 diagonals bands
    x = IRnmRadNoiseCovarMatDiagonalVectors1[0:8461-idiag,idiag]
    covband = covband + np.diag(x,idiag)
    if idiag>0 : covband = covband + np.diag(x,-idiag)
```

```
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(covband)), "covband", covband.shape, (np.nanmax(covband)))
    print idfct, my_str
  # covextra
  covextra=np.dot(w_cov,np.dot(np.diag(v_cov),np.transpose(w_cov))) # matrix computation
  for idiag in range(nb_diag): # remove 5 diagonals reconstruected from eigenvectors
    x_up = np.diag(np.diag(covextra,idiag),idiag)
    x_down = np.diag(np.diag(covextra,-idiag),-idiag)
    covextra = covextra - x_up
    if idiag>0 :
          covextra = covextra - x_down
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(covextra)), "covextra", covextra.shape, (np.nanmax(covextra)))
    print idfct, my_str
  cov = covextra + covband # add cov extra diagonal + diagonal
  if verbose :
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(cov)), "cov", cov.shape, (np.nanmax(cov)))
    print idfct, my_str
  if verbose: print idfct, "end"
  return cov, covband, covextra

def load_write_reconstruct_ncm(npz_fname_1b, npz_fname_1c, formatted_fname, IDefIssueIcd, IDefRevisionIcd,
IDefCovID, IDefCovDate_day, IDefCovDate_ms, verbose=True):
  """
  read NCM structure from npz files (eig_decompos.py output, reconstruct_NCM_PN1_1b.npz and
reconstruct_NCM_PN1_1c.npz for example)
  also keep cov_red output for checking, but not w_cov and v_cov

  copy input parameters into NCM structure (DefIssueIcd, IDefRevisionIcd, IDefCovDate_day, IDefCovDate_ms)

  reconstruct covariance matrix from this NCM structure directly read from npz file
  only keep covariance output, not separated diagonal and extradiagonal parts
  make difference with reduced covariance matrix read from npz file
  difference is expected to be low

  write NCM structure in formatted binary file
  read NCM structure from previously written formatted binary file

  reconstruct covariance matrix from this NCM structure read from formatted binary file
  only keep covariance output, not separated diagonal and extradiagonal parts
  make difference with reduced covariance matrix read from npz file
  difference is expected to be low
  """
  idfct="[load_write_reconstruct_ncm]"
  if verbose: print idfct, "begin"
  if verbose: print idfct, "npz_fname_1b", npz_fname_1b
  if verbose: print idfct, "npz_fname_1c", npz_fname_1c
  if verbose: print idfct, "formatted_fname", formatted_fname
  if verbose: print idfct, "IDefIssueIcd", IDefIssueIcd
  if verbose: print idfct, "IDefRevisionIcd", IDefRevisionIcd
  if verbose: print idfct, "IDefCovID", IDefCovID
  if verbose: print idfct, "IDefCovDate_day", IDefCovDate_day
  if verbose: print idfct, "IDefCovDate_ms", IDefCovDate_ms
  # read NCM structure from npz file (eig_decompos.py output, reconstruct_NCM_PN1.npz for example)
  # also keep cov_red output for checking, but not w_cov and v_cov
```

```
  struct_ncm_1, cov_red_1b, cov_red_1c, _, _, _, _ = load_struct_ncm(npz_fname_1b, npz_fname_1c,
verbose=verbose)
  # copy input parameters into NCM structure
  struct_ncm_1['IDefIssueIcd'] = IDefIssueIcd
  if verbose: print idfct, "struct_ncm_1['IDefIssueIcd']", struct_ncm_1['IDefIssueIcd']
  struct_ncm_1['IDefRevisionIcd'] = IDefRevisionIcd
  if verbose: print idfct, "struct_ncm_1['IDefRevisionIcd']", struct_ncm_1['IDefRevisionIcd']
  struct_ncm_1['IDefCovID'] = IDefCovID
  if verbose: print idfct, "struct_ncm_1['IDefCovID']", struct_ncm_1['IDefCovID']
  struct_ncm_1['IDefCovDate_day'] = IDefCovDate_day
  if verbose: print idfct, "struct_ncm_1['IDefCovDate_day']", struct_ncm_1['IDefCovDate_day']
  struct_ncm_1['IDefCovDate_ms'] = IDefCovDate_ms
  if verbose: print idfct, "struct_ncm_1['IDefCovDate_ms']", struct_ncm_1['IDefCovDate_ms']
  # reconstruct covariance matrix from this NCM structure directly read from npz file
  # only keep covariance output, not separated diagonal and extradiagonal parts
  cov_1b1, _, _ = reconstruct_cov_from_struct_ncm(struct_ncm_1, '1b', verbose=verbose)
  cov_1c1, _, _ = reconstruct_cov_from_struct_ncm(struct_ncm_1, '1c', verbose=verbose)
  # make difference with reduced covariance matrix read from npz file
  # difference is expected to be low
  cov_1b_diff1 = cov_1b1 - cov_red_1b
  my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(cov_1b_diff1)), "cov_1b_diff1", cov_1b_diff1.shape,
(np.nanmax(cov_1b_diff1)))
  print idfct, my_str
  cov_1c_diff1 = cov_1c1 - cov_red_1c
  my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(cov_1c_diff1)), "cov_1c_diff1", cov_1c_diff1.shape,
(np.nanmax(cov_1c_diff1)))
  print idfct, my_str
  # write NCM structure in formatted binary file
  ncm_format_write(formatted_fname, struct_ncm_1, verbose=verbose)
  # read NCM structure from previously written formatted binary file
  struct_ncm_2 = ncm_format_read(formatted_fname, verbose=verbose)
  # reconstruct covariance matrix from this NCM structure read from formatted binary file
  # only keep covariance output, not separated diagonal and extradiagonal parts
  cov_1b2, _, _ = reconstruct_cov_from_struct_ncm(struct_ncm_2, '1b', verbose=verbose)
  cov_1c2, _, _ = reconstruct_cov_from_struct_ncm(struct_ncm_2, '1c', verbose=verbose)
  # make difference with reduced covariance matrix read from npz file
  # difference is expected to be low
  cov_1b_diff2 = cov_1b2 - cov_red_1b
  my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(cov_1b_diff2)), "cov_1b_diff2", cov_1b_diff2.shape,
(np.nanmax(cov_1b_diff2)))
  print idfct, my_str
  cov_1c_diff2 = cov_1c2 - cov_red_1c
  my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(cov_1c_diff2)), "cov_1c_diff2", cov_1c_diff2.shape,
(np.nanmax(cov_1c_diff2)))
  print idfct, my_str
  print idfct, "end"


def plkdirect(t,wave):
  """
  calculate forward Planck function t in K wave in m-1
  resultat w en watt/m**2/steradian/m-1
  """
  sca = Cst_sca2*wave/t
  if ( sca < 100.0 ):
```

```
    denom = math.e**(sca)-1.0
    w = Cst_sca1*wave**3/denom
  else:
    w = 0.0
  return w

def plkinverse(wave,w):
  """
  Inverse Planch funcrion w in W/m**2/str/m-1 wave in m-1
  result t in K
  """
  if ( w > 0.0 ):
    denom = (1.0/w)*Cst_sca1*wave**3
    sca = math.log(denom+1.0)
    t = Cst_sca2*wave/sca
  else:
    t = 0.0
  return t


def plkderive(t,wave):
  """
  calculate forward Planck function t in K wave in m-1
  and its derivative with respect to the temperature t
  results
  w in w/m**2/str/m-1
  dwsdt in w/m**2/str/m-1/K
  """
  sca = Cst_sca2*wave/t
  if ( sca < 100.0 ):
    denom = math.e**(sca)-1.0
    w = Cst_sca1*wave**3/denom
    dwsdt = w/denom*(denom+1.0)*sca/t
  else:
    w = 0.0
    dwsdt = 0.0
  return w,dwsdt


def read_ncm_compute_nedt(formatted_fname, level, verbose=True):
  """
  read NCM structure from formatted binary file
  keep only 1 diagonal vector (there are 5) and compute nedt
  reconstruct covariance matrix from this NCM structure read from formatted binary file
  only keep covariance output, not separated diagonal and extradiagonal parts
  compute nedt on diagonal elements
  nedt results are the same (for diagonal elements)
  """
  idfct="[read_ncm_compute_nedt]"
  if verbose: print idfct, "begin"
  if verbose: print idfct, "formatted_fname", formatted_fname
  if verbose: print idfct, "level", level
  # read NCM structure from formatted binary file
  struct_ncm = ncm_format_read(formatted_fname, verbose=verbose)
  cov_nedt=np.zeros(8461)
```

```
    T0=280.0
    w=0.
    dwsdt=0.
    print idfct, "diagonal elements of struct_ncm (level : %s)" % level
    print idfct, "iSample, nu0, cov_nedt[i]"
    for iSample in range(0,8461,200):
        nu0=64500.0+25.*iSample  #m-1
        w, dwsdt = plkderive(T0, nu0)  #m-1
        #keep only 1 diagonal vector (there are 5)
        idiag = 0
        x = struct_ncm['IRnmRadNoiseCovarMatDiagonalVectors%s' % level][0:8461-idiag,idiag] #m-1
        cov_nedt[iSample] = np.sqrt(x[iSample])/dwsdt #m-1
        my_str= "%4i %8.2f %12.6e" % (iSample, nu0, cov_nedt[iSample])
        print my_str
    # reconstruct covariance matrix from this NCM structure
    # only keep covariance output, not separated diagonal and extradiagonal parts
    cov, _, _ = reconstruct_cov_from_struct_ncm(struct_ncm, level, verbose=verbose)
    my_str= "%e <= %s (%s) <= %e" % ((np.nanmin(cov)), "cov", cov.shape, (np.nanmax(cov)))
    cov_nedt=np.zeros(8461)
    T0=280.0
    w=0.
    dwsdt=0.
    print idfct, "diagonal elements of reconstructed matrix"
    print idfct, "iSample, nu0, cov_nedt[i]"
    for iSample in range(0,8461,200):
      nu0=64500.0+25.*iSample  #m-1
      w, dwsdt = plkderive(T0, nu0)
      #cov_nedt[iSample] = np.sqrt(cov[iSample,iSample])/dwsdt
      cov_nedt[iSample] = np.sqrt(cov[iSample,iSample])/dwsdt
      my_str= "%4i %8.2f %12.6e" % (iSample, nu0, cov_nedt[iSample])
      print my_str
    print idfct, "end"

if __name__ == "__main__":
    idfct="[ncm_format.py]"
    print idfct, "begin"
    #print argv
    #print len(argv)
    if (len(argv)!=2) :
      print __doc__
    if (len(argv)==2) :
      formatted_fname=argv[1]
      for level in ['1b', '1c']:
        read_ncm_compute_nedt(formatted_fname, level, verbose=True)
    print idfct, "end"
```