SFIT4 uncertainties python implementation

- Starting point is Rodgers2000 formula

  error on retrieval = smoothing error + other

$$\hat{x} - x_{\text{true}} = \left(\hat{A} - I_n\right)(x_{\text{true}} - x_a) + \hat{G}\left(\delta + \epsilon + K_b\epsilon_b\right)$$

- continuation of work done by Eric Nussbaumer
- error is measured value - true → unknown
- $\hat{x} = (p, t, t_i)$
- $\delta =$ error in forward model
- $\epsilon =$ error on measured spectrum
- $\epsilon_b =$ error on parameters in forward model that are not retrieved
- in slides: italics = function/variable in Layer1Mods.py

- Clive D. Rodgers. Inverse Methods for Atmospheric Sounding : Theory and Practice (Series on Atmospheric Oceanic and Planetary Physics). World Scientific Publishing Company, June 2000.
- R. Sussmann and T. Borsdorff. Technical note: Interference errors in infrared remote sounding of the atmosphere. *Atmos. Chem. Phys.*, 7:3537–3557, 2007.
- J.R. Taylor. *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements*. Physics - Chemistry - Engineering. University Science Books, 1997.
- T. von Clarmann. *Smoothing error pitfalls*. Atmospheric Measurement Techniques, 7(9) 3023–3034, 2014.

SFIT4 uncertainties python implementation

$$\hat{A} = \begin{pmatrix} \hat{A}_{rr} & \hat{A}_{rt} & \hat{A}_{ri} \\ \hat{A}_{tr} & \hat{A}_{tt} & \hat{A}_{ti} \\ \hat{A}_{ir} & \hat{A}_{it} & \hat{A}_{ii} \end{pmatrix}$$

- Take t component from Rodgers' formula to get error on target profile

$$\left[\hat{x} - x_{\text{true}} = \left(\hat{A} - I_n\right)(x_{\text{true}} - x_a) + \hat{G}\left(\delta + \epsilon_y + K_b\epsilon_b\right)\right]_t$$

$$\epsilon_t = \hat{A}_{tr}\epsilon_r + (A_{tt} - I)\epsilon_{a,t} + A_{ti}\epsilon_{a,i} + \hat{G}_t\delta + \hat{G}_t\epsilon_y + \hat{G}_tK_b\epsilon_b$$

retrieval param error **"smoothing" error** **interfering species error**

noise error **forward model params error**

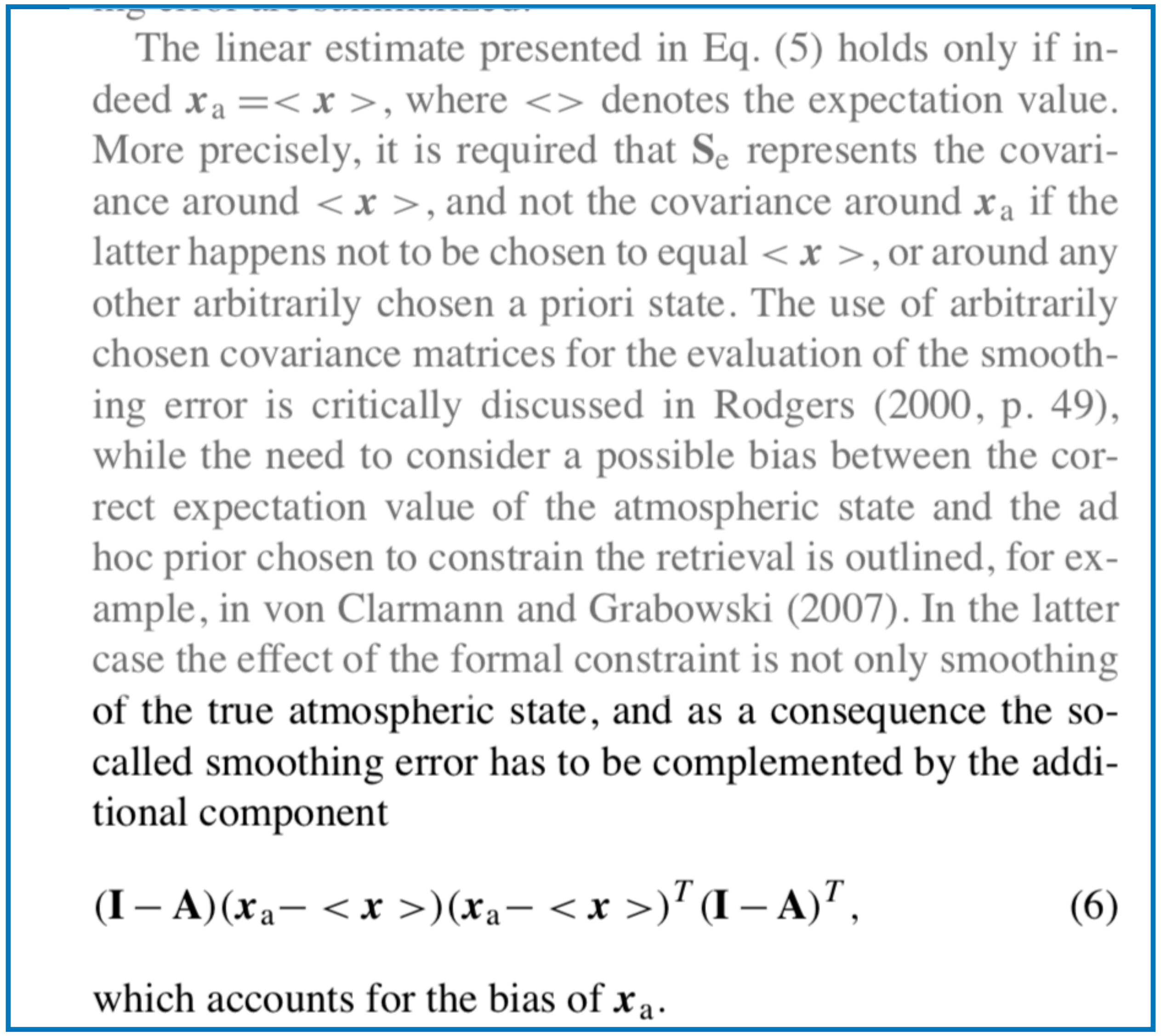
SFIT4 uncertainties python implementation

- each error $\epsilon_r, \epsilon_{t,a}, \epsilon_n, \ldots$ is unknown and requires an estimate from us
- if ϵ is Gaussian, an estimate of the mean error and of the covariance matrix (or standard deviation if scalar) is enough
- If ϵ is not Gaussian, we assume ϵ is small (then mean and cov allow 1st and 2e order approximation)
- uncertainty is estimate of the error
- estimate of the **mean** of an error is called **systematic uncertainty**
  estimate of the **covariance/std** is called **random uncertainty**

Random uncertainty

- estimate of covariance/standard deviation ~ random uncertainty
- we use standard covariance estimates to obtain “random covariance uncertainty matrix”
- follows known transformation rules: $T = \hat{A}_{tt} - I$ or $\hat{A}_{tr}$ or $\hat{G}K_b$ or… then propagated error on retrieved profile is TST^*
- take into account “precision” of uncertainty: does it make sense to input NCEP temperature uncertainty with 0.1K precision?

Systematic uncertainty



The linear estimate presented in Eq. (5) holds only if indeed $\boldsymbol{x}_\mathrm{a} = <\boldsymbol{x}>$, where $<>$ denotes the expectation value. More precisely, it is required that $\mathbf{S}_\mathrm{e}$ represents the covariance around $<\boldsymbol{x}>$, and not the covariance around $\boldsymbol{x}_\mathrm{a}$ if the latter happens not to be chosen to equal $<\boldsymbol{x}>$, or around any other arbitrarily chosen a priori state. The use of arbitrarily chosen covariance matrices for the evaluation of the smoothing error is critically discussed in Rodgers (2000, p. 49), while the need to consider a possible bias between the correct expectation value of the atmospheric state and the ad hoc prior chosen to constrain the retrieval is outlined, for example, in von Clarmann and Grabowski (2007). In the latter case the effect of the formal constraint is not only smoothing of the true atmospheric state, and as a consequence the so-called smoothing error has to be complemented by the additional component

$$(\mathbf{I}-\mathbf{A})(\boldsymbol{x}_\mathrm{a}- <\boldsymbol{x}>)(\boldsymbol{x}_\mathrm{a}- <\boldsymbol{x}>)^T(\mathbf{I}-\mathbf{A})^T, \tag{6}$$

which accounts for the bias of $\boldsymbol{x}_\mathrm{a}$.

- bias ~ systematic uncertainty
- v. Clarmann: creates a 2d matrix from mean of error

$$x_a - < x > = < x_a - x_\text{true} > = < \epsilon_{a,t} >$$

- "systematic matrix" = $S = (< \epsilon_{a,t} >)_i (< \epsilon_{a,t} >)_j$
- transforms as covariance matrix

Systematic uncertainty: summary

- bias ~ systematic uncertainty
- "systematic matrix" = $S = (<\epsilon_{a,t}>)_i(<\epsilon_{a,t}>)_j$
- transforms as covariance matrix

Systematic uncertainty: some remarks

- "if systematic error is known: one should correct for it" but this is not always desirable:
    - do we use NCEP temperature at all stations or a station dependent bias corrected version of NCEP temperature?
    - do we use WACCM or …
    - spectroscopic error ~ bias ~ HITRAN uncertainty: sign is unknown
- "systematic matrix" = $S = (<\epsilon_{a,t}>)_i(<\epsilon_{a,t}>)_j$ does not contain sign of mean error estimate
- we do not distinguish between estimate of bias and estimate of dependency in repeated measurements

SFIT4 python implementation: sb.ctl input file

```
sb.lineInt_H2CO.systematic                  = 0.10
sb.lineTAir_H2CO.systematic                 = 0.10
sb.linePAir_H2CO.systematic                 = 0.10
sb.profile.H2CO.grid= -0.02 0.1 10 16 40 120
sb.profile.H2CO.correlation.width=4
sb.profile.H2CO.random =.30 0.5 0.5 0.5 .20 .20 #relative units
sb.profile.H2CO.systematic = 0.2 0.2 0.1 0.1 0.1 0.1
```

- all input has default values (or as many as possible), inputs are **<u>std'</u>**s
- profile input on coarse grid allowed (-> user friendly)
- systematic profile input: only needs “bias” estimation profile
- random input: similar to sfit4 implementation with correlation width
  $S_{ij} = \sigma_i \sigma_j e^{-\frac{|z_i - z_j|}{w}}$ where w is correlation width
- *createCovar* determines this matrix (both random/systematic case)
  multiplication matrix $e^{-\frac{|z_i - z_j|}{w}}$ has precision of 1/100 (too far off-diagonal elements are set to zero)
- use correlation width $w = 0$ for diagonal random S matrix, correlation width settings is ignored for systematic input

SFIT4 python implementation: default inputs

```
VMRoutFlg                          = T
MolsoutFlg                         = T
out.total                          = T
out.srandom                        = T
out.ssystematic                    = T
SeInputFlg = T
sb.sza.random.scaled                                = T
sb.sza.systematic.scaled                            = T
sb.omega.random.scaled                              = T
sb.omega.systematic.scaled                          = T
file.out.total                                = Stotal.output
file.out.total.vmr                            = Stotal.vmr.output
file.out.srandom                              = Srandom.output
file.out.srandom.vmr                          = Srandom.vmr.output
file.out.ssystematic                          = Ssystematic.output
file.out.ssystematic.vmr                      = Ssystematic.vmr.output
file.out.error.summary                        = Errorsummary.output
file.out.avk                                  = avk.output
sb.temperature.random.scaled                        = F #in Kelvin
sb.temperature.systematic.scaled                    = F #in Kelvin
sb.temperature.grid                 =-0.020    4     6     10    13     25     40     120
sb.temperature.correlation.width    =2
sb.temperature.random               =  2        2     4     4     2      3      6       1
sb.temperature.systematic           =  1        1     1     2     2      2      4       5
sb.profile.H2O.grid                 =-0.020   1     6     10    13    25    40     120
sb.profile.H2O.correlation.width    =4
sb.profile.H2O.random               = 0.10 0.30  0.60  0.50  0.30  0.10  0.10   0.10
#relative units
sb.profile.H2O.systematic           = 0.10  0.4  0.20  0.20  0.20  0.20  0.20   0.20
sb.profile.HDO.grid                 =-0.020   1     6     10    13    25    40     120
sb.profile.HDO.correlation.width    =4
sb.profile.HDO.random               = 0.10 0.30  0.60  0.50  0.30  0.10  0.10   0.10
#relative units
sb.profile.HDO.systematic           = 0.10  0.4  0.20  0.20  0.20  0.20  0.20   0.20
sb.profile.*.grid= -0.02 120
sb.profile.*.correlation.width=4
sb.profile.*.random =.10 .10 #relative units
sb.profile.*.systematic =.10 .10 #relative units
```

```
sb.sza.random             = 0.005
sb.sza.systematic         = 0.001
sb.phase.*                = 0.001
sb.wshift.*               = 0.001
sb.slope.*                = 0.001
sb.curvature.*            = 0.001
sb.max_opd.*              = 0.0
sb.band.*.zshift.*        = 0.01
sb.solshft.*              = 0.005
sb.solstrnth.*            = 0.01
sb.apod_fcn.*             = 0.05
sb.phase_fcn.*            = 0.05
sb.line*_*.random         = 0.
sb.lineInt_CH4.systematic             = 0.03
sb.lineInt_CO.systematic              = 0.02
sb.lineInt_NO2.systematic             = 0.10
sb.lineInt_HNO3.systematic            = 0.1
sb.lineInt_O3.systematic              = 0.05
sb.lineInt_N2O.systematic             = 0.02
sb.lineInt_HCl.systematic             = 0.05
sb.lineInt_O3.systematic              = 0.05
sb.lineInt_HF.systematic              = 0.05
sb.lineInt_OCS.systematic             = 0.02
sb.lineInt_NO.systematic              = 0.05
sb.lineInt_C2H6.systematic            = 0.05
sb.lineInt_HCN.systematic             = 0.1
sb.lineInt_ClONO2.systematic          = 0.1
sb.lineInt_H2O.systematic             = 0.15
sb.lineInt_HDO.systematic             = 0.15

sb.lineTAir_*.systematic               = 0.05
sb.linePAir_*.systematic               = 0.05

#include everything in the error budget, except smoothing!!
sb.total.smoothing   =F
sb.total.*           =T
```

- to construct S matrix for eg H2O: sb.profile.H2O.random is chosen above wildcards sb.profile.*.random -> new class *DictWithDefaults*

SFIT4 python implementation: default settings

- a matrix (rand & syst) is constructed on the full state space:
  random Sa starts from sfit4 output '**file.out.sa_matrix**'
  systematic Sa starts from a **<u>zero</u>** matrix

- gas submatrices in S ran/sys are then filled with the sb.ctl input or default values:
  at present **<u>all gases</u>** have default values for target species!

- **<u>default values</u>** are important because tikhonov type retrievals have an incorrect 'file.out.sa_matrix' (the sa does not exist)

```
sb.profile.*.grid= -0.02 120
sb.profile.*.correlation.width=4
sb.profile.*.random =.10 .10 #relative units
sb.profile.*.systematic =.10 .10 #relative units
```

- the state vector random sa matrix is checked on being "symmetric" and "positive definite" *matPosDefTest*

- transformation function *calcCoVar* makes sure these 2 properties are preserved when propagating uncertainties

- Important: retrieval paramater have zero systematic contribution!

SFIT4 python implementation: some maths

calcCoVar: purpose to transform “covariance” matrix S (**<u>symmetric</u>** and **<u>semi-positive definite</u>**) under a transformation T (to target state vecotr

- $TST* = (TS)T* = T(ST*)$ is not necessarily symmetric
- decomposes $S = DD*$ (using eigenvalues and eigenvectors) and compute TD first, then$TST* = (TD)(TD)*$ is symmetric
- if input is 1D vector (spectral noise is diagonal covariance): $TD = T\text{diag}(\sqrt{S}) = \sqrt{S} \times T$ (where $\times$ is **<u>numpy multiplication</u>**: $(a \times b)_{ij} = a_j b_{ij}$ (**<u>fast and avoids multiplying with 0</u>**!))
- returns transformed S in different units: vmr, pc and the std profile on the total column (again using numpy multiplication: *_diagtransform*)

SFIT4 python implementation: routine setup

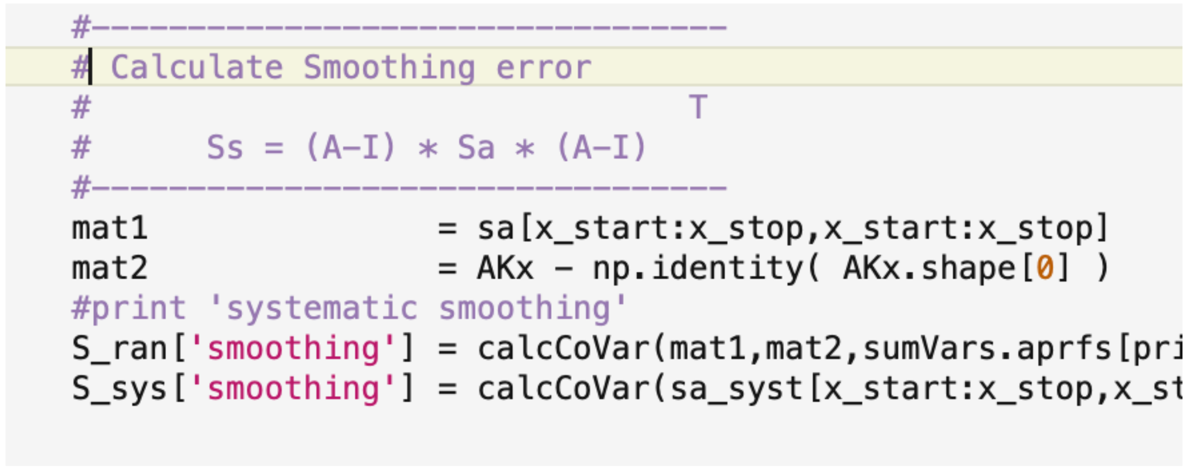
- process the sb.ctl input file (and setup S matrices)

```
    # Insert retrieval grid in sbctldefaults and substitute default values for
SbctlFileVars  ±line720
```

- determine the transformation matrices (A-I,G,GKb, vmr ap, airmass,…)

```
# Read in Gain matrix, ±line920
```

- loop over all uncertainty contributions available:
    1. smoothing error
    2. measurement noise
    3. retrieval parameters
    4. interfering species
    5. all parameters in Kb output



```
#-----------------------------------
# Calculate Smoothing error
#                                T
#      Ss = (A-I) * Sa * (A-I)
#-----------------------------------
mat1                  = sa[x_start:x_stop,x_start:x_stop]
mat2                  = AKx - np.identity( AKx.shape[0] )
#print 'systematic smoothing'
S_ran['smoothing'] = calcCoVar(mat1,mat2,sumVars.aprfs[pri
S_sys['smoothing'] = calcCoVar(sa_syst[x_start:x_stop,x_st
```

- for each contribution store in *S_ran, S_syst* dictionaries

- concludes with summing all contributions and writing the output

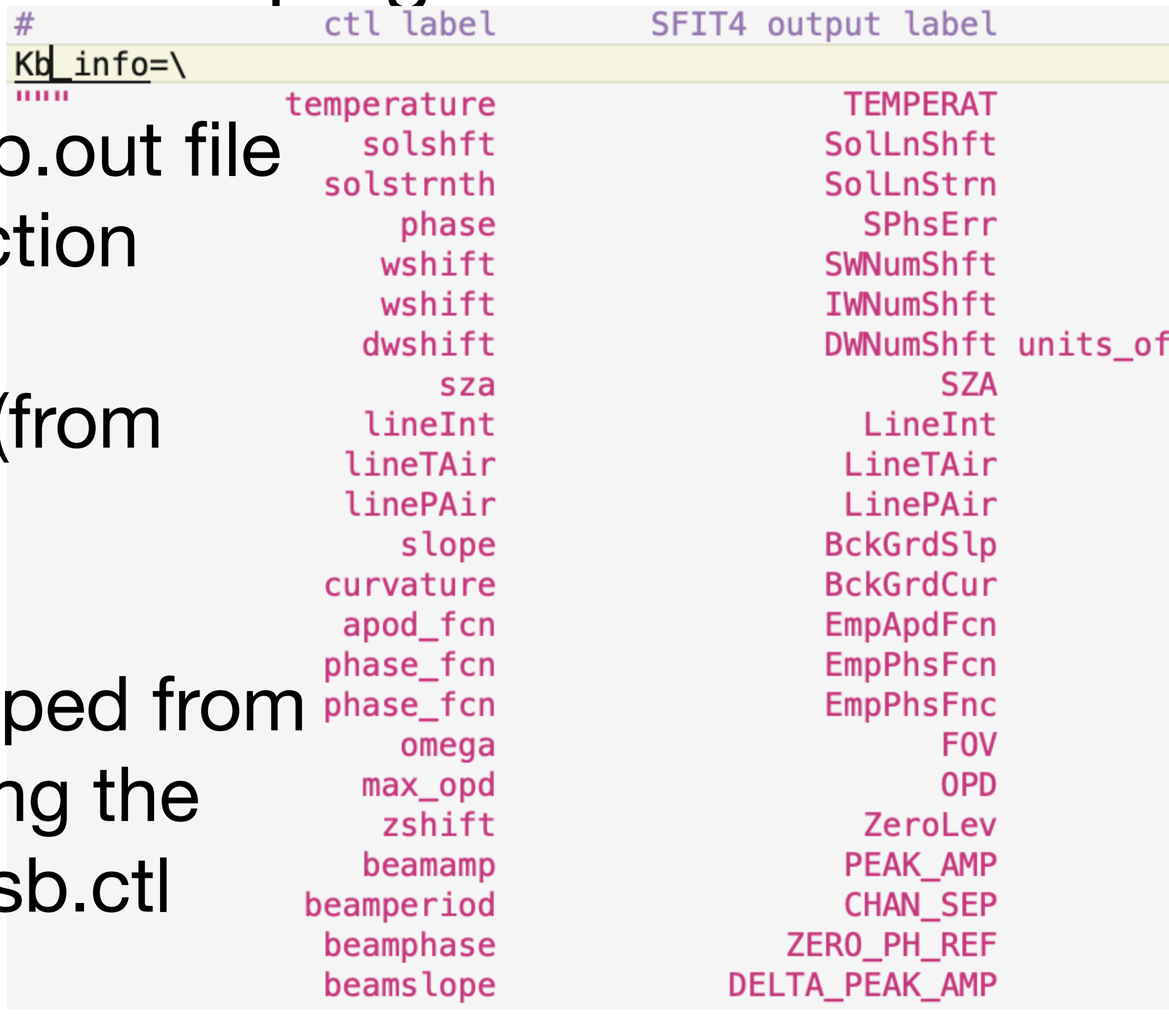
SFIT4 python implementation: housekeeping

- a match is required between labels in Kb.out file and labels in sb.ctl file, done with a function *param_map*
  uses a hardcoded dictionary *Kb_labels* (from *Kb_info*)

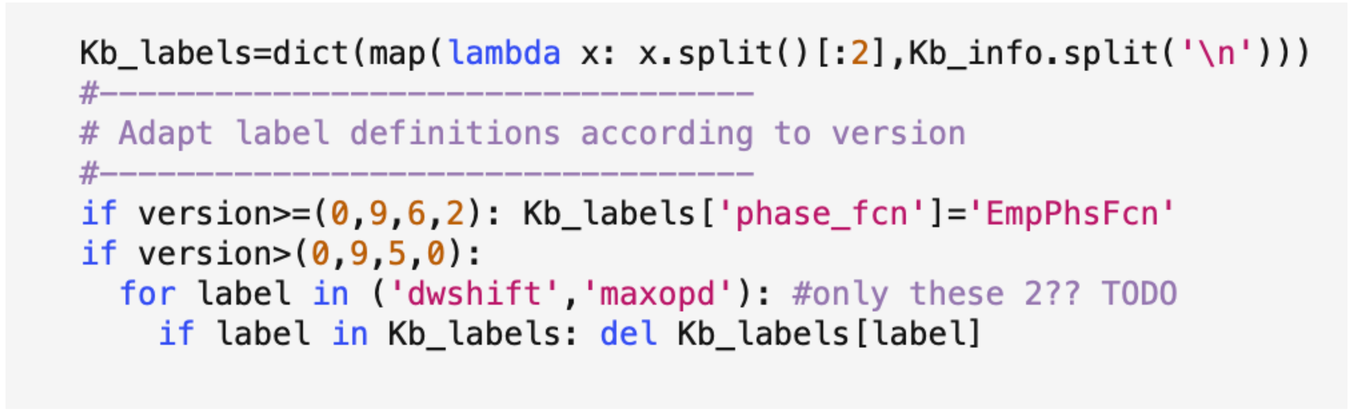
- *Kb_param* is a list of Kb.out labels (stripped from microwindow indices) used form mapping the Kb.out columns to the S matrices from sb.ctl

- *Kb* is a dictionary with keys sb.ctl labels and values are column index arrays (for slicing) from Kb.out

- *Kb_labels* is SFIT4 version dependent



```
#                  ctl label        SFIT4 output label
Kb_info=\
"""
                 temperature              TEMPERAT
                    solshft              SolLnShft
                  solstrnth              SolLnStrn
                      phase                SPhsErr
                     wshift              SWNumShft
                     wshift              IWNumShft
                    dwshift              DWNumShft units_of
                        sza                    SZA
                    lineInt                LineInt
                   lineTAir               LineTAir
                   linePAir               LinePAir
                      slope              BckGrdSlp
                  curvature              BckGrdCur
                   apod_fcn              EmpApdFcn
                  phase_fcn              EmpPhsFcn
                  phase_fcn              EmpPhsFnc
                      omega                    FOV
                    max_opd                    OPD
                     zshift                ZeroLev
                    beamamp               PEAK_AMP
                 beamperiod               CHAN_SEP
                  beamphase            ZERO_PH_REF
                  beamslope         DELTA_PEAK_AMP
```



```
Kb_labels=dict(map(lambda x: x.split()[:2],Kb_info.split('\n')))
#-----------------------------------
# Adapt label definitions according to version
#-----------------------------------
if version>=(0,9,6,2): Kb_labels['phase_fcn']='EmpPhsFcn'
if version>(0,9,5,0):
  for label in ('dwshift','maxopd'): #only these 2?? TODO
    if label in Kb_labels: del Kb_labels[label]
```

SFIT4 python implementation: improvements

- allow the user to have control on S values
  for retrieval parameters and column gas retrievals: use sb.ctl input for sa
- recalculate g matrix if not in output
- test on column only retrievals
- test for temperature retrievals
- calculate uncertainty for all profile gas retrievals
- "sb.file.random" in sb.ctl: obsolete?
- implement kb.profile.gas = H2O and gas.column.list=H2O … uncertainty contribution is taken into account 2x
- … kb for channeling? ils? …