Systematic Analysis

Christophe Pouzat

May 29, 2019

Contents

1 Introduction

Now that the C code, aba_ratio, doing the "heavy" job have been developed, we want to apply them in a systematic way to each experiment of our two data sets: beta-escin perforated patch and "classical" whole-cell. To that end we define Python scripts (watch out, we are using Python 3) that call aba_ratio before extracting from the results (output) of the latter, summary statistics we want to compare across experiments. At the end, these summary statistics are compiled and an HTML file is created making individual experiments inspection (hopefully) easy.

2 A worked out example

We want here to deal with a single experiment, running aba_ratio on it, then inspecting the results, finding out the "good" transients, before re-running aba_ratio on the good transients (if necessary).

2.1 Starting the Python session

The reader is invited here to follow step by step, copying and pasting the commands in his favorite Python console. We start by making sure that the working directory of the Python session is right (the path should end with hess-et-al-beta-escin-2019/analysis, use os.chdir() otherwise) :

```python
import os
print(os.getcwd())
```

```
/home/xtof/github/hess-et-al-beta-escin-2019/analysis
```

This analysis concerns data set DA_120906_E1 in folder hess-et-al-beta-escin-2019/data_paper/data_
We start by creating a dedicated folder to store our analysis and change our working directory to it:

```python
file="DA_120906_E1"
os.mkdir(os.getcwd()+'/'+file+'_analysis')
os.chdir(os.getcwd()+'/'+file+'_analysis')
```

We then copy the data to the working directory:

```
import shutil
data_dir = "../../data_paper/data_whole_cell"
shutil.copy(data_dir+'/'+file+'.h5',file+'.h5')
print(os.listdir())
```

```
['DA_120906_E1.h5']
```

We fit the whole series of transients using the subprocess module:

```
import subprocess
cmd = '../../code/aba_ratio -i '+ file + '.h5 -g -b 7'
p = subprocess.Popen(cmd,shell=True,stderr=subprocess.PIPE)
stderr_msg = p.stderr.read().decode()
print(stderr_msg)
```

```
**********************************
* Doing now stimulation 1
**********************************
iter  0: baseline = 0.0354, delta = 0.0242, tau = 16.6000, RSS = 2499.3412
iter  1: baseline = 0.0274, delta = 0.0189, tau = 1.7927, RSS = 376.4205
iter  2: baseline = 0.0294, delta = 0.0321, tau = 2.5327, RSS = 193.7312
iter  3: baseline = 0.0295, delta = 0.0327, tau = 2.1560, RSS = 189.5791
iter  4: baseline = 0.0294, delta = 0.0328, tau = 2.1992, RSS = 189.5219
iter  5: baseline = 0.0294, delta = 0.0328, tau = 2.1958, RSS = 189.5217
iter  6: baseline = 0.0294, delta = 0.0328, tau = 2.1961, RSS = 189.5217
iter  7: baseline = 0.0294, delta = 0.0328, tau = 2.1961, RSS = 189.5217
iter  8: baseline = 0.0294, delta = 0.0328, tau = 2.1961, RSS = 189.5217
iter  9: baseline = 0.0294, delta = 0.0328, tau = 2.1961, RSS = 189.5217
iter 10: baseline = 0.0294, delta = 0.0328, tau = 2.1961, RSS = 189.5217
iter 11: baseline = 0.0294, delta = 0.0328, tau = 2.1961, RSS = 189.5217
Fitted model Ca = baseline+delta*exp(-(t-t0)/tau)
Summary from method 'trust-region/levenberg-marquardt'
number of iterations: 11
function evaluations: 62
Jacobian evaluations: 0
reason for stopping: small step size
initial RSS = 2499.341231
final   RSS = 189.521712

Number of observation: 174
Number of degrees of freedom: 171
Baseline length: 7
Fit started from point 33
Estimated baseline 0.0294043 and standard error 0.000495979
Estimated delta 0.0328198 and standard error 0.00229856
Estimated tau 2.19606 and standard error 0.245396
RSS per degree of freedom: 1.10831
Probability of observing a larger of equal RSS per DOF under the null hypothesis: 0.15787

**********************************
* Stimulation 1 done
**********************************

**********************************
```

```
* Doing now stimulation 2
***********************************
iter  0: baseline = 0.0584, delta = 0.0335, tau = 16.2000, RSS = 471.2905
iter  1: baseline = 0.0579, delta = 0.0362, tau = 6.2313, RSS = 389.2593
iter  2: baseline = 0.0614, delta = 0.0367, tau = 6.6333, RSS = 220.5901
iter  3: baseline = 0.0611, delta = 0.0368, tau = 6.7976, RSS = 220.4922
iter  4: baseline = 0.0610, delta = 0.0368, tau = 6.8709, RSS = 220.4736
iter  5: baseline = 0.0610, delta = 0.0368, tau = 6.9028, RSS = 220.4702
iter  6: baseline = 0.0610, delta = 0.0368, tau = 6.9166, RSS = 220.4696
iter  7: baseline = 0.0609, delta = 0.0368, tau = 6.9224, RSS = 220.4694
iter  8: baseline = 0.0609, delta = 0.0368, tau = 6.9249, RSS = 220.4694
iter  9: baseline = 0.0609, delta = 0.0368, tau = 6.9260, RSS = 220.4694
iter 10: baseline = 0.0609, delta = 0.0368, tau = 6.9265, RSS = 220.4694
iter 11: baseline = 0.0609, delta = 0.0368, tau = 6.9267, RSS = 220.4694
iter 12: baseline = 0.0609, delta = 0.0368, tau = 6.9268, RSS = 220.4694
iter 13: baseline = 0.0609, delta = 0.0368, tau = 6.9268, RSS = 220.4694
iter 14: baseline = 0.0609, delta = 0.0368, tau = 6.9268, RSS = 220.4694
iter 15: baseline = 0.0609, delta = 0.0368, tau = 6.9268, RSS = 220.4694
iter 16: baseline = 0.0609, delta = 0.0368, tau = 6.9268, RSS = 220.4694
iter 17: baseline = 0.0609, delta = 0.0368, tau = 6.9268, RSS = 220.4694
iter 18: baseline = 0.0609, delta = 0.0368, tau = 6.9268, RSS = 220.4694
iter 19: baseline = 0.0609, delta = 0.0368, tau = 6.9268, RSS = 220.4694
iter 20: baseline = 0.0609, delta = 0.0368, tau = 6.9268, RSS = 220.4694
Fitted model Ca = baseline+delta*exp(-(t-t0)/tau)
Summary from method 'trust-region/levenberg-marquardt'
number of iterations: 20
function evaluations: 92
Jacobian evaluations: 0
reason for stopping: small step size
initial RSS = 471.290532
final   RSS = 220.469411

Number of observation: 170
Number of degrees of freedom: 167
Baseline length: 7
Fit started from point 37
Estimated baseline 0.0609398 and standard error 0.00117512
Estimated delta 0.0368121 and standard error 0.00138225
Estimated tau 6.92681 and standard error 0.655161
RSS per degree of freedom: 1.32018
Probability of observing a larger of equal RSS per DOF under the null hypothesis: 0.00349917
WARNING: THE FIT IS NOT GOOD!

***********************************
* Stimulation 2 done
***********************************

***********************************
* Doing now stimulation 3
***********************************
iter  0: baseline = 0.1257, delta = 0.0296, tau = 15.7000, RSS = 581.4789
iter  1: baseline = 0.1266, delta = 0.0247, tau = 4.5210, RSS = 243.5397
iter  2: baseline = 0.1303, delta = 0.0292, tau = 3.0779, RSS = 181.4660
iter  3: baseline = 0.1305, delta = 0.0308, tau = 3.0580, RSS = 179.5041
iter  4: baseline = 0.1305, delta = 0.0308, tau = 3.0543, RSS = 179.5039
iter  5: baseline = 0.1305, delta = 0.0308, tau = 3.0534, RSS = 179.5039
iter  6: baseline = 0.1305, delta = 0.0308, tau = 3.0532, RSS = 179.5039
iter  7: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
iter  8: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
iter  9: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
iter 10: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
iter 11: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
iter 12: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
iter 13: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
Fitted model Ca = baseline+delta*exp(-(t-t0)/tau)
Summary from method 'trust-region/levenberg-marquardt'
number of iterations: 13
function evaluations: 63
Jacobian evaluations: 0
reason for stopping: small step size
initial RSS = 581.478933
final   RSS = 179.503934

Number of observation: 165
Number of degrees of freedom: 162
Baseline length: 7
Fit started from point 42
Estimated baseline 0.130521 and standard error 0.000725029
Estimated delta 0.0308078 and standard error 0.00203744
Estimated tau 3.05313 and standard error 0.389237
RSS per degree of freedom: 1.10805
Probability of observing a larger of equal RSS per DOF under the null hypothesis: 0.164483

***********************************
* Stimulation 3 done
***********************************

***********************************
* Doing now stimulation 4
***********************************
iter  0: baseline = 0.1374, delta = 0.0362, tau = 34.2000, RSS = 1347.0368
iter  1: baseline = 0.1353, delta = 0.0351, tau = 3.0793, RSS = 1112.5823
iter  2: baseline = 0.1409, delta = 0.0334, tau = 7.1045, RSS = 261.9884
iter  3: baseline = 0.1375, delta = 0.0383, tau = 10.7317, RSS = 190.4950
iter  4: baseline = 0.1354, delta = 0.0412, tau = 11.8196, RSS = 182.4684
iter  5: baseline = 0.1351, delta = 0.0416, tau = 11.9372, RSS = 182.3598
iter  6: baseline = 0.1351, delta = 0.0416, tau = 11.9331, RSS = 182.3597
iter  7: baseline = 0.1351, delta = 0.0416, tau = 11.9336, RSS = 182.3597
iter  8: baseline = 0.1351, delta = 0.0416, tau = 11.9335, RSS = 182.3597
iter  9: baseline = 0.1351, delta = 0.0416, tau = 11.9336, RSS = 182.3597
iter 10: baseline = 0.1351, delta = 0.0416, tau = 11.9335, RSS = 182.3597
iter 11: baseline = 0.1351, delta = 0.0416, tau = 11.9335, RSS = 182.3597
iter 12: baseline = 0.1351, delta = 0.0416, tau = 11.9335, RSS = 182.3597
iter 13: baseline = 0.1351, delta = 0.0416, tau = 11.9335, RSS = 182.3597
Fitted model Ca = baseline+delta*exp(-(t-t0)/tau)
Summary from method 'trust-region/levenberg-marquardt'
number of iterations: 13
function evaluations: 71
Jacobian evaluations: 0
reason for stopping: small step size
initial RSS = 1347.036788
final   RSS = 182.359681

Number of observation: 179
Number of degrees of freedom: 176
Baseline length: 7
Fit started from point 28
Estimated baseline 0.135092 and standard error 0.0011255
Estimated delta 0.0415961 and standard error 0.00148409
Estimated tau 11.9335 and standard error 1.10399
RSS per degree of freedom: 1.03613
Probability of observing a larger of equal RSS per DOF under the null hypothesis: 0.355538

**********************************
* Stimulation 4 done
**********************************

******************************************
* Doing tau vs mean kappa_Fura regression *
******************************************
Best fit: tau = -13.0348 + 0.200292 kappa_Fura
Covariance matrix:
[ +2.94880e+00, -3.60631e-02
  -3.60631e-02, +4.46790e-04  ]
Total sum of squares (TSS) = 111.697
chisq (Residual sum of squares, RSS) = 21.9078
Probability of observing a larger of equal RSS per DOF under the null hypothesis: 1.74892e-05
R squared (1-RSS/TSS) = 0.803864
Estimated gamma/v with standard error: 4.9927 +/- 0.526893
Estimated kappa_S with standard error (using error propagation): -66.0788 +/- 10.9852
kappa_S confidence intervals based on parametric bootstrap
0.95 CI for kappa_S: [-69.2475,-61.3125]
0.99 CI for kappa_S: [-70.0881,-59.1478]
******************************************
* tau vs mean kappa_Fura regression done *
******************************************
******************************************
* Doing tau vs min kappa_Fura regression *
******************************************
Best fit: tau = -10.1116 + 0.175325 kappa_Fura
Covariance matrix:
[ +2.15995e+00, -2.80921e-02
  -2.80921e-02, +3.71894e-04  ]
Total sum of squares (TSS) = 111.697
chisq (Residual sum of squares, RSS) = 29.0429
Probability of observing a larger of equal RSS per DOF under the null hypothesis: 4.93634e-07
R squared (1-RSS/TSS) = 0.739985
Estimated gamma/v with standard error: 5.70371 +/- 0.627371
Estimated kappa_S with standard error (using error propagation): -58.6739 +/- 10.5124
kappa_S confidence intervals based on parametric bootstrap
0.95 CI for kappa_S: [-62.5005,-53.1186]
0.99 CI for kappa_S: [-63.4582,-50.724]
******************************************
* tau vs min kappa_Fura regression done  *
******************************************
******************************************
* Doing tau vs max kappa_Fura regression *
******************************************
Best fit: tau = -15.2462 + 0.216393 kappa_Fura
Covariance matrix:
[ +3.62094e+00, -4.21878e-02
  -4.21878e-02, +4.96739e-04  ]
Total sum of squares (TSS) = 111.697
chisq (Residual sum of squares, RSS) = 17.4307
Probability of observing a larger of equal RSS per DOF under the null hypothesis: 0.000164049
R squared (1-RSS/TSS) = 0.843947
Estimated gamma/v with standard error: 4.62122 +/- 0.475968
Estimated kappa_S with standard error (using error propagation): -71.456 +/- 11.4012
kappa_S confidence intervals based on parametric bootstrap
0.95 CI for kappa_S: [-74.4272,-67.2971]
0.99 CI for kappa_S: [-75.193,-65.6481]
******************************************
* tau vs max kappa_Fura regression done  *
******************************************
```

We extract the analysis results, a bit boring text file manipulation–boring but much easier to achieve with `Python` than with `C` code–:

```python
with open(file+'_aba_tau_vs_mean_kappa','r') as fin:
    tau_vs_kappa = fin.read()

tau_vs_kappa.find('# Using stimulation:')
debut = tau_vs_kappa.find('# Using stimulation:')
fin = tau_vs_kappa.find('\n',debut)
working_string = tau_vs_kappa[(debut+len('# Using stimulation:')):fin]
nb_transients = int(working_string[-1])
rss_per_dof = []
fit_info = []
kept_transients = []
for t_idx in range(1,nb_transients+1):
    with open(file+'_aba_RatioFit_s'+str(t_idx),'r') as fin:
        RatioFit = fin.read()

    debut = RatioFit.find('# nobs = ')
    fin = RatioFit.find('# rss per degree of freedom: ')
    fit_info.append(RatioFit[debut:fin])
    if RatioFit.find("WARNING: THE FIT IS NOT GOOD!",debut,fin) == -1:
        kept_transients.append(t_idx)

    debut = fin + len('# rss per degree of freedom: ')
    fin = RatioFit.find('\n',debut)
    rss_per_dof.append(float(RatioFit[debut:fin]))
```

We find the bad transients:

```python
start = 0
bad_pos = stderr_msg.find("WARNING: THE FIT IS NOT GOOD!",start)
bad_stim = []
while bad_pos > -1:
    pos1 = stderr_msg.rfind("* Doing now stimulation ", start,  bad_pos)
    bad_stim.append(int(stderr_msg[pos1+24]))
    start = bad_pos+30
    bad_pos = stderr_msg.find("WARNING: THE FIT IS NOT GOOD!",start)

if len(bad_stim) > 0:
    print(bad_stim)
```

```
[2]
```

We find the total number of transients:

```
start = 0
pos1 = stderr_msg.rfind("* Doing now stimulation ")
total_stim = int(stderr_msg[pos1+24])
print("The number ofstimulations is: " + stderr_msg[pos1+24])
```

```
The number ofstimulations is: 4
```

If necessary (that is, if there are "bad" trials), we redo the analysis without them:

```
good_stim = [i for i in range(1,total_stim+1) if i not in bad_stim]
cmd += ' -s ' + str(good_stim).strip('[]').replace(' ','')
p = subprocess.Popen(cmd,shell=True,stderr=subprocess.PIPE)
stderr_msg = p.stderr.read().decode()
print(stderr_msg)
```

```
**********************************
* Doing now stimulation 1
**********************************
iter  0: baseline = 0.0354, delta = 0.0242, tau = 16.6000, RSS = 2499.3412
iter  1: baseline = 0.0274, delta = 0.0189, tau = 1.7927, RSS = 376.4205
iter  2: baseline = 0.0294, delta = 0.0321, tau = 2.5327, RSS = 193.7312
iter  3: baseline = 0.0295, delta = 0.0327, tau = 2.1560, RSS = 189.5791
iter  4: baseline = 0.0294, delta = 0.0328, tau = 2.1992, RSS = 189.5219
iter  5: baseline = 0.0294, delta = 0.0328, tau = 2.1958, RSS = 189.5217
iter  6: baseline = 0.0294, delta = 0.0328, tau = 2.1961, RSS = 189.5217
iter  7: baseline = 0.0294, delta = 0.0328, tau = 2.1961, RSS = 189.5217
iter  8: baseline = 0.0294, delta = 0.0328, tau = 2.1961, RSS = 189.5217
iter  9: baseline = 0.0294, delta = 0.0328, tau = 2.1961, RSS = 189.5217
iter 10: baseline = 0.0294, delta = 0.0328, tau = 2.1961, RSS = 189.5217
iter 11: baseline = 0.0294, delta = 0.0328, tau = 2.1961, RSS = 189.5217
Fitted model Ca = baseline+delta*exp(-(t-t0)/tau)
Summary from method 'trust-region/levenberg-marquardt'
number of iterations: 11
function evaluations: 62
Jacobian evaluations: 0
reason for stopping: small step size
initial RSS = 2499.341231
final   RSS = 189.521712

Number of observation: 174
Number of degrees of freedom: 171
Baseline length: 7
Fit started from point 33
Estimated baseline 0.0294043 and standard error 0.000495979
Estimated delta 0.0328198 and standard error 0.00229856
Estimated tau 2.19606 and standard error 0.245396
RSS per degree of freedom: 1.10831
Probability of observing a larger of equal RSS per DOF under the null hypothesis: 0.15787

**********************************
* Stimulation 1 done
**********************************

**********************************
```

```
* Doing now stimulation 3
***********************************
iter  0: baseline = 0.1257, delta = 0.0296, tau = 15.7000, RSS = 581.4789
iter  1: baseline = 0.1266, delta = 0.0247, tau = 4.5210, RSS = 243.5397
iter  2: baseline = 0.1303, delta = 0.0292, tau = 3.0779, RSS = 181.4660
iter  3: baseline = 0.1305, delta = 0.0308, tau = 3.0580, RSS = 179.5041
iter  4: baseline = 0.1305, delta = 0.0308, tau = 3.0543, RSS = 179.5039
iter  5: baseline = 0.1305, delta = 0.0308, tau = 3.0534, RSS = 179.5039
iter  6: baseline = 0.1305, delta = 0.0308, tau = 3.0532, RSS = 179.5039
iter  7: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
iter  8: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
iter  9: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
iter 10: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
iter 11: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
iter 12: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
iter 13: baseline = 0.1305, delta = 0.0308, tau = 3.0531, RSS = 179.5039
Fitted model Ca = baseline+delta*exp(-(t-t0)/tau)
Summary from method 'trust-region/levenberg-marquardt'
number of iterations: 13
function evaluations: 63
Jacobian evaluations: 0
reason for stopping: small step size
initial RSS = 581.478933
final   RSS = 179.503934

Number of observation: 165
Number of degrees of freedom: 162
Baseline length: 7
Fit started from point 42
Estimated baseline 0.130521 and standard error 0.000725029
Estimated delta 0.0308078 and standard error 0.00203744
Estimated tau 3.05313 and standard error 0.389237
RSS per degree of freedom: 1.10805
Probability of observing a larger of equal RSS per DOF under the null hypothesis: 0.164483

***********************************
* Stimulation 3 done
***********************************

***********************************
* Doing now stimulation 4
***********************************
iter  0: baseline = 0.1374, delta = 0.0362, tau = 34.2000, RSS = 1347.0368
iter  1: baseline = 0.1353, delta = 0.0351, tau = 3.0793, RSS = 1112.5823
iter  2: baseline = 0.1409, delta = 0.0334, tau = 7.1045, RSS = 261.9884
iter  3: baseline = 0.1375, delta = 0.0383, tau = 10.7317, RSS = 190.4950
iter  4: baseline = 0.1354, delta = 0.0412, tau = 11.8196, RSS = 182.4684
iter  5: baseline = 0.1351, delta = 0.0416, tau = 11.9372, RSS = 182.3598
iter  6: baseline = 0.1351, delta = 0.0416, tau = 11.9331, RSS = 182.3597
iter  7: baseline = 0.1351, delta = 0.0416, tau = 11.9336, RSS = 182.3597
iter  8: baseline = 0.1351, delta = 0.0416, tau = 11.9335, RSS = 182.3597
iter  9: baseline = 0.1351, delta = 0.0416, tau = 11.9336, RSS = 182.3597
iter 10: baseline = 0.1351, delta = 0.0416, tau = 11.9335, RSS = 182.3597
iter 11: baseline = 0.1351, delta = 0.0416, tau = 11.9335, RSS = 182.3597
iter 12: baseline = 0.1351, delta = 0.0416, tau = 11.9335, RSS = 182.3597
iter 13: baseline = 0.1351, delta = 0.0416, tau = 11.9335, RSS = 182.3597
Fitted model Ca = baseline+delta*exp(-(t-t0)/tau)
Summary from method 'trust-region/levenberg-marquardt'
number of iterations: 13
```

```
function evaluations: 71
Jacobian evaluations: 0
reason for stopping: small step size
initial RSS = 1347.036788
final   RSS = 182.359681

Number of observation: 179
Number of degrees of freedom: 176
Baseline length: 7
Fit started from point 28
Estimated baseline 0.135092 and standard error 0.0011255
Estimated delta 0.0415961 and standard error 0.00148409
Estimated tau 11.9335 and standard error 1.10399
RSS per degree of freedom: 1.03613
Probability of observing a larger of equal RSS per DOF under the null hypothesis: 0.355538

**********************************
* Stimulation 4 done
**********************************

******************************************
* Doing tau vs mean kappa_Fura regression *
******************************************
Best fit: tau = -13.6918 + 0.208884 kappa_Fura
Covariance matrix:
[ +5.13322e+00, -6.46311e-02
  -6.46311e-02, +8.20406e-04  ]
Total sum of squares (TSS) = 74.8946
chisq (Residual sum of squares, RSS) = 21.7103
Probability of observing a larger of equal RSS per DOF under the null hypothesis: 3.17091e-06
R squared (1-RSS/TSS) = 0.710122
Estimated gamma/v with standard error: 4.78733 +/- 0.65645
Estimated kappa_S with standard error (using error propagation): -66.5471 +/- 14.0865
kappa_S confidence intervals based on parametric bootstrap
0.95 CI for kappa_S: [-69.7983,-60.9418]
0.99 CI for kappa_S: [-70.6262,-58.1578]
******************************************
* tau vs mean kappa_Fura regression done *
******************************************
******************************************
* Doing tau vs min kappa_Fura regression *
******************************************
Best fit: tau = -9.42271 + 0.165536 kappa_Fura
Covariance matrix:
[ +3.25022e+00, -4.35833e-02
  -4.35833e-02, +5.92004e-04  ]
Total sum of squares (TSS) = 74.8946
chisq (Residual sum of squares, RSS) = 28.6076
Probability of observing a larger of equal RSS per DOF under the null hypothesis: 8.86332e-08
R squared (1-RSS/TSS) = 0.618028
Estimated gamma/v with standard error: 6.04099 +/- 0.88793
Estimated kappa_S with standard error (using error propagation): -57.9226 +/- 13.7337
kappa_S confidence intervals based on parametric bootstrap
0.95 CI for kappa_S: [-62.3143,-50.5641]
0.99 CI for kappa_S: [-63.3537,-46.825]
******************************************
* tau vs min kappa_Fura regression done  *
******************************************
******************************************
```

```
* Doing tau vs max kappa_Fura regression *
******************************************
Best fit: tau = -17.8818 + 0.248863 kappa_Fura
Covariance matrix:
[ +7.21813e+00, -8.65053e-02
  -8.65053e-02, +1.04273e-03  ]
Total sum of squares (TSS) = 74.8946
chisq (Residual sum of squares, RSS) = 15.4997
Probability of observing a larger of equal RSS per DOF under the null hypothesis: 8.25195e-05
R squared (1-RSS/TSS) = 0.793047
Estimated gamma/v with standard error: 4.01827 +/- 0.521392
Estimated kappa_S with standard error (using error propagation): -72.8537 +/- 14.2644
kappa_S confidence intervals based on parametric bootstrap
0.95 CI for kappa_S: [-75.5726,-68.6405]
0.99 CI for kappa_S: [-76.2269,-66.7985]
******************************************
* tau vs max kappa_Fura regression done  *
******************************************
```

We make the plots:

```python
files = os.listdir()
for f in files:
    if ".gp" in f:
        gp_name = f
        gp_out = f.replace(".gp",".png")
        gp_cmd = ("set terminal pngcairo size 800,800 enhanced font 'Verdana,9';\n"
                  "set o '" + gp_out + "';\n"
                  "load '" + gp_name + "'\n")
        subprocess.run('gnuplot',input=gp_cmd,text=True)
```

3 The `aba_boring` program

Now that we know what we want to do on each experiment, we write a `Python` script that does automatically what we did step by step in the previous section.

3.1 Code presentation

The literate programming approach is used here. This means that the code–at the beginning at least–is broken into "manageable" pieces that are individually explained (when just reading the code is not enough), they are then pasted together to give the code that will actually make the functions. These manageable pieces are called blocks and each block gets a name like: `<<name-of-the-block>>` upon definition. It is then referred to by this name when used in subsequent codes. See Schulte, Davison, Dye and Dominik (2010) A Multi-Language Computing Environment for Literate Programming and Reproducible Research for further explanations.

3.2 aba_boring

We define now our Python program, aba_boring.py, doing the "boring part" of the analysis for us. The skeleton of this program is:

```
# import modules
<<aba_boring-imports>>

# parse program arguments
<<aba_boring-parse-arguments>>

# create directory where results will be kept
<<aba_boring-prepare-dir-and-data>>

# run the analysis of the whole set of transients
<<aba_boring-run-on-whole-set>>

# get the total number of transients
<<aba_boring-find-number-of-transients>>

# Get key statistics on each transient
<<aba_boring-get-single-transient-statistics>>

# If some transients are not well fitted redo the
# analysis using only the good ones
<<aba_boring-redo-fits-on-good-transients-if-necessary>>

# Generates the figures and the output summary file
# in markdown (MD) format
<<aba_boring-generate-md-output>>

# Do some clean up
<<aba_boring-clean-up>>

# Try to generate an HTML version of the MD summary file
<<aba_boring-generate-html-output>>
```

3.2.1 <<aba_boring-imports>>

Five modules of the standard library are used in our code:

- os
- shutil
- subprocess
- random
- argparse

```
import os
import shutil
import subprocess
import random
import argparse
```

3.2.2 <<aba_boring-parse-arguments>>

This code block reads `aba_boring` arguments and prints the help message. The arguments that are initialized at the end of this block are:

- `data_name`: name of the file containing the dataset.
- `data_dir`: name of the directory containing the dataset.
- `path`: the path to `aba_ratio` executable code.
- `baseline_length`: length of the baseline used in transient fits.
- `rng_seed`: random number generator seed used to estimate the null distribution of the residuals correlation coefficient.
- `nperm`: number of permutations used to estimate the null distribution of the residuals correlation coefficient.

```
script_description = ("Runs aba_ratio in a systematic way on a given data set."
                      " All transients are analyzed first. The ones having a too "
                      "large residual sum of square (RSS) or a too large residuals "
                      "lag 1 correlation are detected and a new "
                      "aba_ratio run is performed with the other ones only. "
                      "The diagnostic figures are generaed in png format and "
                      "summary reports are generated in MarkDown and HTML "
                      "formats. These figure and reports, together with aba_ratio "
                      "output are stored in a directory called 'data_name_analysis' "
                      "created as a sub-directory of the one from which this "
                      "program is executed. The null disdribution of the residuals "
                      "lag 1 correlation is obtained by simulation (permutations "
                      "of the observed sequence of residuals).")
parser = argparse.ArgumentParser(description=script_description)
parser.add_argument('-f', '--data-file-name',
                    dest='data_name',
                    help = ('The data file name without the ".h5" suffix.'),
                    required=True)
parser.add_argument('-d', '--data-directory',
                    dest='data_directory',
                    help = ('The directory name where the data are found.'),
                    required=True)
parser.add_argument('-p', '--path-to-code',
                    dest='path_to_aba_ratio',
                    help = ('The directory name where the code is found. '
                            'Only necessary if it not in the user PATH.'),
                    required=False)
# Get the baseline length
```

```
def _baseline_length(string):
    n = int(string)
    if n <= 0:
        msg = "The baseline length must be > 0"
        raise argparse.ArgumentTypeError(msg)
    return n
parser.add_argument('-l', '--baseline-length',
                    type=_baseline_length, dest='baseline_length',
                    help=('The baseline length for the fits '
                          '(default 7)'),
                    default=7)
# Set the (pseudo)random number generator seed
parser.add_argument('-s', '--rng-seed',
                    type=int, dest='seed',
                    help=('The random number generator seed '
                          '(default 18710305)'),
                    default=18710305)
# Get the number of permutations
def _nb_perm(string):
    n = int(string)
    if n <= 0:
        msg = "The number of permutations must be > 0"
        raise argparse.ArgumentTypeError(msg)
    return n
parser.add_argument('-r', '--number-of-permutations',
                    type=_nb_perm, dest='nperm',
                    help=('The number of permutations used for '
                          'the simulations (default 1000)'),
                    default=1000)
args = parser.parse_args()

data_name = args.data_name
data_dir = os.path.abspath(args.data_directory)
path = os.path.abspath(args.path_to_aba_ratio)
baseline_length = args.baseline_length
rng_seed = args.seed
random.seed(rng_seed)
nperm = args.nperm
```

3.2.3 <<aba_boring-prepare-dir-and-data>>

The code creates first a directory `data_name_analysis` (if it does not already exist) as a sub-directory of the one from which the code is executed. The data set is then copied into it (it is removed at the end of the analysis).

```
# Make directory to store results and copy data into it
new_dir = os.path.abspath(data_name+'_analysis')
if not os.path.exists(new_dir):  # Check if directory already exists
    os.makedirs(new_dir)  # if not, create it
os.chdir(new_dir)
shutil.copy(data_dir+'/'+ data_name +'.h5',data_name +'.h5')
```

3.2.4 <<aba_boring-run-on-whole-set>>

This code block runs program `aba_ratio` on the whole set of transients. The information on the fit of `aba_ratio` appear of the `stderr` exactly as if `aba_ration` was run directly from the command line.

```python
# run aba_ratio on the whole set of transients
print("Doing now analysis with all the transients.\n")
cmd = path + '/aba_ratio -i ' + data_name + '.h5 -g -b ' + str(baseline_length)
p = subprocess.Popen(cmd,shell=True,stderr=subprocess.PIPE)
print(p.stderr.read().decode())
```

3.2.5 <<aba_boring-find-number-of-transients>>

A short "utility" block getting the number of transients in the dataset.

```python
# Figure out first the number of transients
with open(data_name + '_aba_tau_vs_mean_kappa','r') as fin:
    tau_vs_kappa = fin.read()

tau_vs_kappa.find('# Using stimulation:')
debut = tau_vs_kappa.find('# Using stimulation:')
fin = tau_vs_kappa.find('\n',debut)
working_string = tau_vs_kappa[(debut+len('# Using stimulation:')):fin]
nb_transients = int(working_string[-1])
```

3.2.6 <<aba_boring-get-single-transient-statistics>>

This block extracts the key transients fit quality information from the output files generated by `aba_ratio`. The lag 1 residual correlation is also computed and the probability of the observed value under the null hypothesis of no correlation is estimated by generating a sample of nperm permutations. Here "probability of the observed value under the null" means the probability of observing a lag 1 correlation smaller or equal to the observed one (under the null hypothesis).

This block creates five lists that are subsequently used:

- `rss_per_dof`: contains the value of the RSS (residual sum of squares) per DOF (degrees of freedom) for each transient.
- `tau_se`: contains the standard error on tau (the time constant) for each transient.
- `res_corr_lag1`: contains a tuple with the lag 1 correlation and the probability under the null for each transient.
- `fit_info`: contains a string with general information on the fit for each transient
- `kept_transients`: contains the indices of the 'good' transients.

A fit is classified as 'bad' if any of the following three conditions is true:

- The RSS per DOF is so large that it should be observed in less than 1% of the cases under the null hypothesis.
- A `NaN` resulted at some point during the fitting procedure.
- the lag 1 correlation is so large that it should be observed in less than 1% of the cases under the null hypothesis.

```python
# Next get: the RSS per dof for each transient
#           the key part of the fit info for each
#           as well as the standard error on tau
#           and statistics on the residuals correlation
# Create a list with the good transients

<<aba_boring-corr-function-definition>>

rss_per_dof = []  # a list of values
tau_se = []  # a list of values
res_corr_lag1 = []  # a list of tuples (correlation and Proba of a smaller value under the null
fit_info = []  # a list strings
kept_transients = []  # a list of integers
for t_idx in range(1,nb_transients+1):
    <<aba_boring-residual-lag-1>>
    <<aba_boring-rss-per-dof-etc>>
    # Find out if the fit was good enough
    condition1 = RatioFit.find("WARNING: THE FIT IS NOT GOOD!") == -1
    condition2 = RatioFit.find(("Probability of observing a larger of equal "
                                "RSS per DOF under the null hypothesis: "
                                "-nan")) == -1
    condition3 = RatioFit.find(("Probability of observing a larger of equal "
                                "RSS per DOF under the null hypothesis: "
                                "nan")) == -1
    condition4 = res_corr_lag1[-1][1] > 0.01
    if condition1 and condition2 and condition3 and condition4:
        kept_transients.append(t_idx)
```

`<<aba_boring-corr-function-definition>>` Function `corr` returns the lag 1 (auto)correlation of its argument.

```python
def corr(lst: list) -> float:
    """Returns lag 1 correlation of its input.

    The elements of lst are assumed centered (no mean subtraction is
    done).
    """
    n = len(lst) - 1
    return sum([lst[i+1]*lst[i] for i in range(n)])/n
```

`<<aba_boring-residual-lag-1>>` This code block gets the residuals from the file containing the whole results of the transient fit generated by `aba_ratio`. See the comments in the code for details. The list `res_corr_lag1` is updated.

```
# Get the residuals in order to study their lag 1 correlation
residual = []
for line in open(data_name + '_aba_RatioFit_s' + str(t_idx), 'r'):
    if ("#" in line) or ("\n" == line):
        continue
    else:
        residual.append(float(line.split(" ")[3]))
residual_corr = corr(residual)  # lag 1 correlation of the residuals
corr_null = [0]*nperm  # a list used for storing the corr. of the shuffled
                       # residuals
for i in range(nperm):
    random.shuffle(residual)
    corr_null[i] = corr(residual)
corr_null.sort()  # the corr. coef. of the shuffled residuals are now sorted
for i in range(-1,-len(corr_null)-1,-1):
    # find out the index of the first corr. coef. smaller or equal than
    # the observed one
    if corr_null[i] <= residual_corr:
        break
# Add to res_corr_lag1 a tuple with the observed value and its probability
res_corr_lag1.append((residual_corr,1.0-min(1.0,(nperm+i+1)/nperm)))
```

`<<aba_boring-rss-per-dof-etc>>` This code block gets:

- the standard error on tau
- the "general" fit information (a string)
- the RSS per DOF

It updates:

- `tau_se`
- `fit_info`
- `rss_per_dof`

```
with open(data_name + '_aba_RatioFit_s' + str(t_idx),'r') as fin:
        RatioFit = fin.read()
# Get the standard error on tau
debut = RatioFit.find("# estimated tau ")
debut = RatioFit.find(" and standard error ",debut)
fin = RatioFit.find("\n",debut)
debut = debut+len(" and standard error ")
tau_se.append(float(RatioFit[debut:fin]))
# Get the general information on the fit
debut = RatioFit.find('# nobs = ')
fin = RatioFit.find('# rss per degree of freedom: ')
fit_info.append(RatioFit[debut:fin])
# Get the RSS per DOF
debut = fin + len('# rss per degree of freedom: ')
fin = RatioFit.find('\n',debut)
rss_per_dof.append(float(RatioFit[debut:fin]))
```

3.2.7 <<aba_boring-redo-fits-on-good-transients-if-necessary>>

If the length of the list containing the "good" transients, `kept_transients`, is (strictly) shorter than the number of transients, then redo the fits on the good transient only in order to have a tau vs kappa regression using only the reliable transients.

```
# if there are bad transients redo analysis with the good ones
if len(kept_transients) < nb_transients:
    if len(kept_transients) >= 3:
        print("Doing now analysis with the 'good' transients.\n")
        cmd += ' -s ' + str(kept_transients).strip('[]').replace(' ','')
        p = subprocess.Popen(cmd,shell=True,stderr=subprocess.PIPE)
        stderr_msg = p.stderr.read().decode()
```

3.2.8 <<aba_boring-generate-md-output>>

Generates a summary file in `markdown` format.

```
fout = open(data_name+'_report.md','w')
fout.write("*Analysis of dataset " + data_name + "*\n")
fout.write("-----\n\n")
fout.write("[TOC]\n\n")

fout.write("The baseline length is: {0:d}.\n\n".format(baseline_length))
fout.write(("**When fitting tau against kappa_Fura only the transients "
            "for which the fit RSS and the lag 1 auto-correlation "
            "of the residuals were small enough, giving an overall "
            "probability of false negative of 0.02, were kept** (see "
            "the numerical summary associated with each transient).\n\n"))

fout.write("\nThe good transients are: " + str(kept_transients).strip('[]') + ".\n\n")
if len(kept_transients) < 3:
    fout.write("**Not enough good transients to keep going!**\n\n")

files = os.listdir()
files = [f for f in files if '.gp' in f]

<<aba_boring-mkfig-definition>>
<<aba_boring-loading-curve>>
<<aba_boring-transients>>
if len(kept_transients) >= 3:
    <<aba_boring-tau-vs-kappa>>
<<aba_boring-key-summary-stats>>
```

```
def mkfig(f: str) -> str:
    """Makes figures from gnuplot script file f and returns file name of created figure.
    """
    gp_name = f
    gp_out = f.replace(".gp",".png")
    gp_cmd = ("set terminal pngcairo size 800,800 enhanced font 'Verdana,12';\n"
              "set o '" + gp_out + "';\n"
```

```
               "load '" + gp_name + "'\n")
    subprocess.run('gnuplot',input=gp_cmd,text=True)
    return gp_out
```

<<aba_boring-mkfig-definition>>

```
# Start with the loading curve
loading_name = [f for f in files if 'loading_curve' in f][0]
fout.write("# Loading curve\n")
fout.write(("The time at which the 'good' transients were recorded"
            " appear in red.\n\n"))
gp_out = mkfig(loading_name)
fout.write("!["+loading_name.strip(".gp")+"]("+gp_out+")\n\n")
```

<<aba_boring-loading-curve>>

```
# The transient
fout.write("# Transients \n")
fout.write(("On each graph, the residuals appear on top.\n"
            "**Under the null hypothesis**, if the monoexponential fit is correct "
            "**they should be centered on 0 and have a SD close to 1** (not "
            "exactly 1 since parameters were obtained through the fitting "
            "procedure form the data.\n\n"
            "The estimated [Ca2+] appears on the second row. The estimate "
            "is show in black together with pointwise 95% confidence intervals."
            " The fitted curve appears in red. **The whole transient is not "
            "fitted**, only a portion of it is: a portion of the baseline "
            "made of " + str(baseline_length) + " points and "
            "the decay phase starting at the time where the Delta[Ca2+] "
            "has reached 50% of its peak value.\n\n"
            "The time appearing on the abscissa is the time from the beginning "
            "of the experiment.\n\n"))
for t_idx in range(1,nb_transients+1):
    fout.write("## Transient " + str(t_idx) + "\n")
    transient_name = [f for f in files if 'RatioFit_s'+str(t_idx) in f][0]
    if t_idx in kept_transients:
        fout.write("**Transient "+str(t_idx)+" is 'good'.**\n\n")
    else:
        fout.write("**Transient "+str(t_idx)+" is a 'bad'.**\n\n")
    fout.write("### Fit graphical summary\n")
    gp_out = mkfig(transient_name)
    fout.write("!["+transient_name.strip(".gp")+"]("+gp_out+")\n\n")
    fout.write("### Fit numerical summary\n")
    #fout.write("\n\n~~~~~\n\n")
    fout.write("\n\n")
    fout.write(fit_info[t_idx-1].replace("#",">").replace("\n","\n\n"))
    fout.write("> Lag 1 residuals auto-correlation: {0:.3f}\n\n".\
               format(res_corr_lag1[t_idx-1][0]))
    fout.write("> Pr[Lag 1 auto-corr. > {0:.3f}] = {1:.3f}\n\n".\
               format(*res_corr_lag1[t_idx-1]))
    #fout.write("\n\n~~~~~\n\n")
    fout.write("\n\n")
```

<<aba_boring-transients>>

```
# tau vs kappa figures
fout.write("# tau vs kappa \n")
fout.write(("Since the [Fura] changes during a transient (and it can change a lot "
            "during the early transients), the _unique_ value to use as '[Fura]' "
            "is not obvious. We therefore perform 3 fits: one using the minimal "
            "value, one using the mean and one using the maximal value.\n\n"
            "The observed tau (shown in red) are displayed with a 95% confidence "
            "interval that results from the fitting procedure and _is_ therefore "
            "_meaningful only if the fit is correct_!\n\n"
            "No serious attempt at quantifying the precision of [Fura] and "
            "therefore kappa_Fura has been made since the choice of which [Fura] "
            "to use has a larger effect and since the other dominating effect "
            "is often the certainty we can have that the saturating value (the "
            "[Fura] in the pipette) has been reached.\n\n"
            "The straight line in black is the result of a _weighted_ linear "
            "regression. The blue dotted lines correspond to the limits of "
            "_pointwise 95% confidence intervals_.\n\n"))
suffix = ["min","mean","max"]
for s in suffix:
    fout.write("## tau vs kappa  using the " + s + " [Fura] value\n")
    fout.write("### Fit graphical summary\n")
    tau_name = [f for f in files if 'tau_vs_'+s in f][0]
    gp_out = mkfig(tau_name)
    fout.write("!["+tau_name.strip(".gp")+"]("+gp_out+")\n\n")
    fout.write("### Fit numerical summary\n")
    with open(data_name + '_aba_tau_vs_' + s + '_kappa','r') as fin:
        tau_vs_kappa = fin.read()
        debut = tau_vs_kappa.find("# Best fit:")
        fin = tau_vs_kappa.find("\n\n# The data")
        #fout.write("\n\n~~~~~\n\n")
        fout.write("\n\n")
        fout.write(tau_vs_kappa[debut:fin].replace("#",">").replace("\n","\n\n"))
        #fout.write("\n\n~~~~~\n\n")
        fout.write("\n\n")
```

<<aba_boring-tau-vs-kappa>>

```
# write the RSS per dof for each good transient
rss_per_dof = [rss_per_dof[i] for i in range(nb_transients) if i+1 in kept_transients]
tau_se = [tau_se[i] for i in range(nb_transients) if i+1 in kept_transients]
res_corr_lag1 = [res_corr_lag1[i] for i in range(nb_transients) if i+1 in kept_transients]
corr_lag1 = [res_corr_lag1[i][0] for i in range(len(res_corr_lag1))]
corr_prob = [res_corr_lag1[i][1] for i in range(len(res_corr_lag1))]
fout.write(("# RSS per DOF, standard error of tau and lag 1 residual "
            "correlation for each 'good' tansient\n"))
fout.write("{0:d} out of {1:d} transients  were kept.\n\n".\
           format(len(kept_transients), nb_transients))
fout.write("sigma(tau): "+str(tau_se).strip('[]')+"\n\n")
fout.write("Residual correlation at lag 1: "+str(corr_lag1).strip('[]')+"\n\n")
fout.write(("Probablity of a correlation at lag 1 smaller or equal than "
```

```
            "observed: ")+str(corr_prob).strip('[]')+"\n\n")
fout.write("RSS/DOF: "+str(rss_per_dof).strip('[]')+"\n")
```

<<aba_boring-key-summary-stats>>

3.2.9 <<aba_boring-clean-up>>

Close summary file and remove data copy.

```
fout.close()
os.remove(data_name + ".h5")
```

3.2.10 <<aba_boring-generate-html-output>>

If pandoc is installed, use it to generate a summary in HTML.

```
# Try generating the html output with the markdown module
# You can install it since it is not part of the standard library
# with: pip install markdown
# See https://python-markdown.github.io/
try:
    import markdown
    import codecs  # part of the standard library
    fin = codecs.open(data_name+"_report.md", mode="r", encoding="utf-8")
    md = fin.read()
    fin.close()
    html = markdown.markdown(md,extensions=["extra","toc"])
    fout = codecs.open(data_name+"_report.html", mode="w",
                       encoding="utf-8", errors="xmlcharrefreplace")
    fout.write(html)
    fout.close()
except ModuleNotFoundError:
    # check if pandoc is installed
    pandoc = subprocess.run("pandoc -v", shell=True)
    if pandoc.returncode == 0:
        # if yes, call pandoc
        pandoc_cmd = 'pandoc -s ' + data_name + '_report.md -o ' + data_name + '_report.html'
        subprocess.call(pandoc_cmd, shell=True)
    else:
        print(("Neither 'markdown' (Python module), nor 'pandoc' found,"
               " so no 'html' output was generated."))
```

3.3 Test aba_boring

As a first test we repeat what our worked out example was doing (on Unix/Linux the properties of aba_boring.py must be set such that the file is executable, otherwise, python3 aba_boring.py -f DA_120906_E1 -d ../data_paper/data_whole_cell -p ../code can be used):

```
./aba_boring.py -f DA_120906_E1 -d ../data_paper/data_whole_cell -p ../code
```

4 The `aba4comp` program

We now want to have a code that takes folder names indicating where the beta-escin and the whole cell data sets are found, some extra parameters required by `aba_boring` and running the latter on each experiment of the two data sets before generating a useful summary.

4.1 `aba4comp` skeleton

```
# Imports required modules
<<aba4comp-imports>>

# Makes sure that path are adapted to OS
<<aba4comp-clean-paths>>

# beta-escin data set analysis
<<aba4comp-beta-analysis>>

# whole-cell data set analysis
<<aba4comp-wc-analysis>>

#  Define read_report function
<<read-report-definition>>

# reads beta-escin analysis results
<<aba4comp-read-beta-results>>

# reads whole-cell analysis results
<<aba4comp-read-wc-results>>

# writes analysis summary to disk in md format
<<DA-analysis-summary-write>>

# converts md to html
<<DA-analysis-summary-to-html>>
```

4.1.1 `<<aba4comp-imports>>`

The required modules are imported here. Module `markdown` is the only one that is not part of the standard library; it must therefore be installed by the user first.

```
import os
from multiprocessing import Pool
from statistics import mean, stdev
from math import sqrt
from collections import OrderedDict
import markdown # Not part of standard lib. must be installed by user
import codecs
```

4.1.2 <<aba4comp-clean-paths>>

Makes OS independent paths and stores them in varables:

- DA_beta_dir: where the beta-escin data are located.
- DA_wc_dir: where the whole-cell data are located.
- path2aba_ratio: where the C codes are located.
- path2aba_boring: where aba_boring.py is located.

```
DA_beta_dir = os.path.abspath('../data_paper/data_beta_escin')
DA_wc_dir = os.path.abspath('../data_paper/data_whole_cell')
path2aba_ratio = os.path.abspath('../code')
path2aba_boring = os.path.abspath('.')
```

4.1.3 <<aba4comp-beta-analysis>>

This code block performs the systematic analysis of the beta-escin data set.

```
new_dir = os.path.abspath('DA-beta')
if not os.path.exists(new_dir):  # Check if directory already exists
    os.makedirs(new_dir)  # if not, create it
os.chdir(new_dir)
DAfiles = [file for file in os.listdir(DA_beta_dir) if file.endswith(".h5")]

def call_aba_boring(file_name):
    import subprocess
    da_cmd = (path2aba_boring + "/aba_boring.py "
              "-f " + file_name[:file_name.find(".h5")] + " "
              "-d " + DA_beta_dir + " "
              "-p " + path2aba_ratio)
    subprocess.call(da_cmd,shell=True)

with Pool() as p:
    p.map(call_aba_boring, DAfiles)

#  Go back to where we started
os.chdir('../')
```

4.1.4 <<aba4comp-wc-analysis>>

This code block performs the systematic analysis of the whole data set.

```
new_dir = os.path.abspath('DA-wc')
if not os.path.exists(new_dir):  # Check if directory already exists
    os.makedirs(new_dir)  # if not, create it
os.chdir(new_dir)

DAfiles = [file for file in os.listdir(DA_wc_dir) if file.endswith(".h5")]
```

```
def call_aba_boring(file_name):
    import subprocess
    da_cmd = (path2aba_boring + "/aba_boring.py "
              "-f " + file_name[:file_name.find(".h5")] + " "
              "-d " + DA_wc_dir + " "
              "-p " + path2aba_ratio)
    subprocess.call(da_cmd,shell=True)

with Pool() as p:
    p.map(call_aba_boring, DAfiles)

#  Go back to where we started
os.chdir('../')
```

4.1.5 `<<aba4comp-read-beta-results>>`

This block reads part of the analysis results of each beta-escin experiment.

```
beta_dir = os.path.abspath('DA-beta')
os.chdir(beta_dir)
the_list = os.listdir(beta_dir)
the_list = [n for n in the_list if 'DA_' in n]
the_list.sort()
beta = OrderedDict([(dn.replace("_analysis",""), read_report(dn)) for dn in the_list])
#  Go back to where we started
os.chdir('../')
```

4.1.6 `<<aba4comp-read-wc-results>>`

This block reads part of the analysis results of each whole-cell experiment.

```
wc_dir = os.path.abspath('DA-wc')
os.chdir(wc_dir)
the_list = os.listdir(wc_dir)
the_list = [n for n in the_list if 'DA_' in n]
the_list.sort()
wc = OrderedDict([(dn.replace("_analysis",""), read_report(dn)) for dn in the_list])
#  Go back to where we started
os.chdir('../')
```

4.2 Prepare summaries to compare the two conditions

We now want to prepare a summary table of our fits for each condition, beta-escin and whole-cell. This table will contain for each dataset:

- the data set name,
- the `number of good transients` / `total number of transients`,
- the setting leading to the best τ vs κ fit (`min`, `mean` or `max`),

- the RSS (DOF) of this best fit,
- the probability of a larger χ^2 value under the null hypothesis,
- the bootstrap 99% confidence interval for κ_S.

4.2.1 `read_report` definition

We define function `read_report` that, as its name says, reads a individual dataset "report" and outputs a `dictionnary` containing the key descriptive statistics of that transient's fit.

```
def read_report(report_name: str) -> dict:
    <<read_report-docstring>>
    from collections import OrderedDict
    import math  # get nan and inf
    <<read_report-open-and-read-report>>
    <<mk_get_value-definition>>
    <<get_value_initialization>>
    <<read_report-get-nb-transients>>
    <<read_report-get-transients-fit-info>>
    <<read_report-get-tau-vs-kappa-fit-info>>
    return {"nb_used": nb_used,
            "nb_total": nb_total,
            "setting": best_setting,
            "transients": transients,
            "tau_vs_kappa": tau_vs_kappa}
```

`<<read-report>>` skeleton

`<<read_report-docstring>>` Code block `<<read_report-docstring>>` contains the function's docstring:

```
"""Read report in directory report_name and returns a dictionary with
relevant statitics.

Parameters
----------
report_name: the name of the directory containing the report (if
             'DA_120906_E1_analysis' a file called 'DA_120906_E1_reprot.md'
             will be looked for in that directory)

Returns
-------
A dictionary with the following keys:
nb_used: the number of 'usable' transients
nb_total: the total number of transients
setting: the best settinf (min, mean or max), ie, the setting yielding the
         best tau vs kappa (straight line) fit
RSS: the residual sum of squares of the best fit
Prob: the probability for the Chi2 distribution to exceed the RSS under
      the null hypothesis
CI: the 99% confidence interval for kappa_s for the bset fit (using a
    bootstrap).
"""
```

`<<read_report-open-and-read-report>>` Code block `<<read_report-open-and-read-report>>` opens the `md` file containg the report and reads it, assigning the resulting string to variable `report`:

```
fn = report_name.replace("_analysis","_report.md")
with open(report_name+'/'+fn,'r') as fin:
    report = fin.read()
```

`<<mk_get_value-definition>>` Code block `<<mk_get_value-definition>>` defines a function returning a function (more precisely a *closure*) that will make it easy for us to extract the information we want from the report:

```
def mk_get_value(report: str):
    """Takes a strings and returns a function (more precisely a closure)
    making it easy to extract specific information contained in the strig.
    """
    import math
    def get_value(start: int,
                  bra: str,
                  cket: str = "\n",
                  what: str = "int") -> tuple:
        """Reads a value from report between bra and cket from start
        and returns a tuple containg the point where it stopped and
        the requested value.

        Pamaters
        --------
        start: a positive intege, the  position from which searched is done.
        bra: a string preceeding immediatly the value.
        cket: a string following the value (in other words we should have
              bra value cket).
        what: a string, either 'int' or 'float' depending on the type of
              value

        Returns
        -------
        A tuple with the position in report where cket was found and the
        value.
        """
        start = report.find(bra,start)+len(bra)
        end = report.find(cket,start)
        val = report[start:end]
        if what == "int":
            res = int(val)
        else:
            res = float(val)
        return end,res
    return get_value
```

`<<get_value_initialization>>` Code block `<<get_value_initialization>>` initializes a `get_value` closure designed to work on our `report` string:

```
get_value = mk_get_value(report)
```

`<<read_report-get-nb-transients>>` Code block `<<read_report-get-nb-transients>>` finds out the number of "good" transients and the total number of transients in the dataset and assigns them, respectively, to variables `nb_used` and `nb_total`:

```
get_value_par = {"start": 0,
                 "bra": ("# RSS per DOF, standard error of tau and lag 1 "
                         "residual correlation for each 'good' tansient\n"),
                 "cket": " out of ",
                 "what": "int"}
start, nb_used = get_value(**get_value_par)
get_value_par = {"start": start,
                 "bra": " out of ",
                 "cket": " transients",
                 "what": "int"}
start, nb_total = get_value(**get_value_par)
```

`<<read_report-get-transients-fit-info>>` Code block `<<read_report-get-transients-fit-info>>` gets the summary information for each transient and keeps it in an OrderedDict, `transients`:

```
transients = OrderedDict()
dict_lst = [{"bra": "> nobs = ",
             "cket": "\n",
             "what": "int"},  # n_obs
            {"bra": "> number of degrees of freedom = ",
             "cket": "\n",
             "what": "int"},  # n_dof
            {"bra": "> baseline length = ",
             "cket": "\n",
             "what": "int"},  # baseline_length
            {"bra": "> fit started from point ",
             "cket": "\n",
             "what": "int"},  # fit_start
            {"bra": "> estimated baseline ",
             "cket": " and standard error ",
             "what": "float"},  # Ca0_best
            {"bra": " and standard error ",
             "cket": "\n",
             "what": "float"},  # Ca0_se
            {"bra": "> estimated delta ",
             "cket": " and standard error ",
             "what": "float"},  # Delta_best
            {"bra": " and standard error ",
             "cket": "\n",
             "what": "float"},  # Delta_se
            {"bra": "> estimated tau ",
             "cket": " and standard error ",
             "what": "float"},  # Tau_best
            {"bra": " and standard error ",
             "cket": "\n",
             "what": "float"},  # Tau_se
            {"bra": "> residual sum of squares: ",
             "cket": "\n",
             "what": "float"},  # rss
            {"bra": "> RSS per degree of freedom: ",
             "cket": "\n",
             "what": "float"},  # rss_per_dof
            {"bra": ("> Probability of observing a larger of equal RSS "
                     "per DOF under the null hypothesis: "),
             "cket": "\n",
             "what": "float"},  # prob_rss
            {"bra": "> Lag 1 residuals auto-correlation: ",
             "cket": "\n",
             "what": "float"},  # lag1
            {"bra": "] = ",
             "cket": "\n",
             "what": "float"}]  # prob_lag1

for t_idx in range(1,nb_total+1):
    start=report.find("## Transient "+str(t_idx))
    start=report.find("### Fit numerical summary",start)
    get_value_par = {"start": start}
    val = []
    for par in dict_lst:
        get_value_par.update(par)
        get_value_par["start"] = start
        start, res = get_value(**get_value_par)
        val.append(res)
    transients["Transient"+str(t_idx)]={"n_obs": val[0],
                                        "n_dof": val[1],
                                        "baseline_length": val[2],
                                        "fit_start": val[3],
                                        "Ca0_best": val[4],
                                        "Ca0_se": val[5],
                                        "Delta_best": val[6],
                                        "Delta_se": val[7],
                                        "tau_best": val[8],
                                        "tau_se": val[9],
                                        "rss": val[10],
                                        "rss_per_dof": val[11],
                                        "prob_rss": val[12],
                                        "lag1": val[13],
                                        "prob_lag1": val[14]}
```

`<<read_report-get-tau-vs-kappa-fit-info>>` Code block `<<read_report-get-tau-vs-kappa-fit-i`
reads summary information of the tau vs kappa fits if such fits were actually performed, that is, if a list 3 'good' transients were obtained from the data set. The information is kept in an `OrderedDict`, `tau_vs_kappa`:

```
tau_vs_kappa = OrderedDict()
dict_lst = [{"bra": "> chisq (Residual sum of squares, RSS) = ",
             "cket": "\n",
             "what": "float"},  # rss
            {"bra": ("> Probability of observing a larger of equal "
                     "RSS per DOF under the null hypothesis: ")  ,
             "cket": "\n",
             "what": "float"},  # prob
            {"bra": "> Estimated gamma/v with standard error: ",
             "cket": " +/- ",
             "what": "float"},  # gamma_best
            {"bra": " +/- ",
             "cket": "\n",
             "what": "float"},  # gamma_se
            {"bra": ("> Estimates kappa_S with standard "
                     "error (using error propagation): "),
             "cket": " +/- ",
             "what": "float"},  # kappa_S_best
            {"bra": " +/- ",
             "cket": "\n",
             "what": "float"},  # kappa_S_se
            {"bra": "> 0.99 CI for kappa_S: [",
             "cket": ",",
             "what": "float"},  # 99% CI lower
            {"bra": ",",
             "cket": "]\n",
             "what": "float"}]  # 99% CI upper
settings = ["min","mean","max"]
best_setting = "min"
for suffix in settings:
    if nb_used < 3:
        tau_vs_kappa["tau_vs_kappa_"+suffix]={"rss": "NA",
                                              "prob": "NA",
                                              "gamma_best": "NA",
                                              "gamma_se": "NA",
                                              "kappa_S_best": "NA",
                                              "kappa_S_se": "NA",
                                              "CI_lower": "NA",
                                              "CI_upper": "NA"}
    else:
        bra = "## tau vs kappa  using the "+suffix+" [Fura] value\n"
        start = report.find(bra)
        get_value_par = {"start": start}
        val = []
        for par in dict_lst:
            get_value_par.update(par)
            get_value_par["start"] = start
            start, res = get_value(**get_value_par)
            val.append(res)
        if suffix == "min":
            best_rss = val[0]
        elif val[0] < best_rss:
            best_rss = val[0]
            best_setting = suffix
        CI = "["+str(round(val[6]))+","+str(round(val[7]))+"]"
        tau_vs_kappa["tau_vs_kappa_"+suffix]={"RSS": val[0],
                                              "Prob": val[1],
                                              "gamma_best": val[2],
                                              "gamma_se": val[3],
                                              "kappa_S_best": val[4],
                                              "kappa_S_se": val[5],
                                              "CI_lower": round(val[6]),
                                              "CI_upper": round(val[7]),
                                                  "CI": CI}
```

4.2.2 <<DA-analysis-summary-write>>

Create a string holding the results in MarkDown format:

```
md="# Summaries for the beta-escin and whole-cell data sets\n"
md+="[TOC]\n\n"
<<DA-analysis-tau-vs-kappa-explanations>>
<<DA-analysis-tau-vs-kappa-beta-escin-table>>
<<DA-analysis-tau-vs-kappa-wc-table>>

md+="# Transients numerical summaries of the beta-escin data sets\n"
<<DA-analysis-beta-escin-summaries>>

md+="# Transients numerical summaries of the whole-cell data sets\n"
<<DA-analysis-whole-cell-summaries>>
```

```
md+="## The content of the tables\n"
md+=("In the next two tables, the columns have the following meaning:\n\n"
     "+ **Data set** gives the data set name.\n"
     "+ **used/total** displays the number of 'good' transients over the "
     "total number of transients.\n"
     "+ **setting** is the [Fura] concentration choice (among 'min', 'mean' "
     "and 'max') that yielded the best straight line fit.\n"
     "+ **RSS** is the residual sum of squares of the best fit (its magnitude "
     "depends on the number of 'good' transients in the dataset).\n"
     "+ **Prob** is the probability for the RSS to *exceed* the observed "
     "value under the null hypothesis.\n"
     "+ **99% CI** is the 99% confidence interval for the endogenous kappa "
     "(kappa_s) obtained with a bootstrap simulation.\n\n"
     "**Warning** if the data set yieleded less than 3 'good' transients, no "
     "straight line fit was performed (it would have been meaningless) and "
     "'NA' (Not Available) appears in the last 4 columns of the table.\n\n")
```

```
md+="## Analysis summary of beta-escin data sets\n\n"
md+="**tau vs kappa(Fura) straight line regression summary**\n\n"
tbl_head=["Data set","used/total","setting","RSS","Prob","99% CI"]
md+="~~~~~\n"
md+="{0:^25}|{1:^14}|{2:^11}|{3:^9}|{4:^9}|{5:^25}\n".format(*tbl_head)
md+="\n"
for data_name,value in beta.items():
    link="[{0}_report]({0}_analysis/{0}_report.html) ".format(data_name)
    ratio="{nb_used}/{nb_total}".format(**value)
    md+="{0:^26}{1:^15}".format(data_name,ratio)
    if value["nb_used"] < 3:
        md+="{0:^12}{0:^10}{0:^10}{0:^26}".format("NA")
    else:
        par = {"nb_used": value['nb_used'],
               "nb_total": value['nb_total'],
               "setting": value['setting']}
        good_tau = value['tau_vs_kappa']['tau_vs_kappa_'+value['setting']]
        par.update(good_tau)
        md+="{setting:^11s}{RSS:^10.2f}{Prob:^10.2f}{CI:^26s}".format(**par)
```

```
    md+="\n"

md+="~~~~~\n"
```

```
md+="## Analysis summary of whole-cell data sets\n\n"
md+="**tau vs kappa(Fura) straight line regression summary**\n\n"
tbl_head=["Data set","used/total","setting","RSS","Prob","99% CI"]
md+="~~~~~\n"
md+="{0:^25}|{1:^14}|{2:^11}|{3:^9}|{4:^9}|{5:^25}\n".format(*tbl_head)
md+="\n"
for data_name,value in wc.items():
    link="[{0}_report]({0}_analysis/{0}_report.html) ".format(data_name)
    ratio="{nb_used}/{nb_total}".format(**value)
    md+="{0:^26}{1:^15}".format(data_name,ratio)
    if value["nb_used"] < 3:
        md+="{0:^12}{0:^10}{0:^10}{0:^26}".format("NA")
    else:
        par = {"nb_used": value['nb_used'],
               "nb_total": value['nb_total'],
               "setting": value['setting']}
        good_tau = value['tau_vs_kappa']['tau_vs_kappa_'+value['setting']]
        par.update(good_tau)
        md+="{setting:^11s}{RSS:^10.2f}{Prob:^10.2f}{CI:^26s}".format(**par)
    md+="\n"

md+="~~~~~\n"
```

For each experiment in the beta-escin data set a complete summary is created:

```
dir_name = "DA-beta"
for data_name,value in beta.items():
    <<DA-analysis-write-complete-summary>>
    md += txt
```

For each experiment in the whole-cell data set a complete summary is created:

```
dir_name = "DA-wc"
for data_name,value in wc.items():
    <<DA-analysis-write-complete-summary>>
    md += txt
```

What goes into the "complete summary":

```
report_name = data_name+"_report"
full_report_name = dir_name+"/"+data_name+"_analysis/"+report_name+".html"
txt = "## Data set "+data_name+" numerical summary\n"
txt += ("The full report with figures is located at ["
        "{report_name:s}]({full_report_name:s}).\n\n")
txt = txt.format(**{"report_name": report_name,
                    "full_report_name": full_report_name})
txt += ("This data set contains {nb_total:d} transients, "
        "{nb_used:d} of which were 'good' enough for the "
        "tau vs kappa fit.\n\n")
txt = txt.format(**value)
txt += ("The RSS per degree of freedom were (values in '()' correspond "
        "to 'bad' transients): ")
```

```
nb_total = value['nb_total']
for t_idx in range(1,nb_total+1):
    transient = value['transients']['Transient'+str(t_idx)]
    rss_per_dof = transient['rss_per_dof']
    if transient['prob_rss'] > 0.01:
        txt += "{0:.3f}".format(rss_per_dof)
    else:
        txt += "({0:.3f})".format(rss_per_dof)
    if t_idx < nb_total:
        txt += ", "
    else:
        txt += ".\n\n"
txt += ("The lag 1 residual autocorrelation coefficient were "
        "(values in '()' correspond to 'bad' transients): ")
for t_idx in range(1,nb_total+1):
    transient = value['transients']['Transient'+str(t_idx)]
    lag1 = transient['lag1']
    if transient['prob_rss'] > 0.01:
        txt += "{0:.3f}".format(lag1)
    else:
        txt += "({0:.3f})".format(lag1)
    if t_idx < nb_total:
        txt += ", "
    else:
        txt += ".\n\n"
```

4.2.3 `<<DA-analysis-summary-to-html>>`

Takes the string in Markdown format, converts it into html and writes result to file.

```
html = markdown.markdown(md,extensions=["extra","toc"])
fout = codecs.open('beta_wc_comp.html',mode='w',
                   encoding='utf-8',errors='xmlcharrefreplace')
fout.write(html)
fout.close()
```

4.3 Run `aba4comp`

After setting the properties of `aba4comp.py` such that the file is executable, the *whole analysis* is performed with:

```
./aba4comp.py
```

Once the execution is finished, two directories, `DA-beta` and `DA-wc` have been created and filed with the analysis results of each experiment, file `beta_wc_comp.html` has also been created. It gives an overview of the analysis results together with a quick access to each individual experiment.