

▼ BPL_TEST2_Batch_calibration script with PyFMI ver 2.7.4

The key library PyFMI v2.7.4 is installed and downgrading is done Numpy v1.19.1. To simplify this we first install conda.

After the installation a small application BPL_TEST2_Batch_calibration is loaded and run. You can continue with this example if you like.

```
!lsb_release -a # Actual VM Ubuntu version used by Google
```

```
No LSB modules are available.
Distributor ID: Ubuntu
Description:   Ubuntu 18.04.6 LTS
Release:      18.04
Codename:     bionic
```

```
%env PYTHONPATH=
```



```
env: PYTHONPATH=
```

```
!wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh
!chmod +x Miniconda3-py37_4.12.0-Linux-x86_64.sh
!bash ./Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -f -p /usr/local
import sys
sys.path.append('/usr/local/lib/python3.7/site-packages/')
```

```
--2022-09-20 11:24:13-- https://repo.anaconda.com/miniconda/Miniconda3-py37_4
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.131.3, 104.16.130.3,
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.131.3|:443... conn
HTTP request sent, awaiting response... 200 OK
Length: 104996770 (100M) [application/x-sh]
Saving to: 'Miniconda3-py37_4.12.0-Linux-x86_64.sh'
```

```
Miniconda3-py37_4.1 100%[=====>] 100.13M 131MB/s in 0.8s
```

```
2022-09-20 11:24:14 (131 MB/s) - 'Miniconda3-py37_4.12.0-Linux-x86_64.sh' save
```

```
PREFIX=/usr/local
Unpacking payload ...
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

```
## Package Plan ##
```

```
environment location: /usr/local
```

```
added / updated specs:
```

```
- _libgcc_mutex==0.1=main
- _openmp_mutex==4.5=1_gnu
- brotli==0.7.0=py37h27cfd23_1003
- ca-certificates==2022.3.29=h06a4308_1
- certifi==2021.10.8=py37h06a4308_2
```

```

- cffi==1.15.0=py37hd667e15_1
- charset-normalizer==2.0.4=pyhd3eb1b0_0
- colorama==0.4.4=pyhd3eb1b0_0
- conda-content-trust==0.1.1=pyhd3eb1b0_0
- conda-package-handling==1.8.1=py37h7f8727e_0
- conda==4.12.0=py37h06a4308_0
- cryptography==36.0.0=py37h9ce1e76_0
- idna==3.3=pyhd3eb1b0_0
- ld_impl_linux-64==2.35.1=h7274673_9
- libffi==3.3=he6710b0_2
- libgcc-ng==9.3.0=h5101ec6_17
- libgomp==9.3.0=h5101ec6_17
- libstdcxx-ng==9.3.0=hd4cf53a_17
- ncurses==6.3=h7f8727e_2
- openssl==1.1.1n=h7f8727e_0
- pip==21.2.2=py37h06a4308_0
- pycosat==0.6.3=py37h27cfd23_0
- pycparser==2.21=pyhd3eb1b0_0
- pyopenssl==22.0.0=pyhd3eb1b0_0
- pysocks==1.7.1=py37_1
- python==3.7.13=h12debd9_0
- readline==8.1.2=h7f8727e_1
- requests==2.27.1=pyhd3eb1b0_0
- ruamel_yaml==0.15.100=py37h27cfd23_0
- setuptools==61.2.0=py37h06a4308_0
- six==1.16.0=pyhd3eb1b0_1
- sqlite==3.38.2=hc218d9a_0
- tk==8.6.11=h1ccaba5_0
- tqdm==4.63.0=pyhd3eb1b0_0
- urllib3==1.26.8=pyhd3eb1b0_0
- wheel==0.37.1=pyhd3eb1b0_0

```

```
!conda update -n base -c defaults conda --yes
```

```

colorama-0.4.4-pyhd3eb1b0_0
conda-content-trust-0.1.1-pyhd3eb1b0_0
six-1.16.0-pyhd3eb1b0_1

```

The following packages will be UPDATED:

_openmp_mutex	4.5-1_gnu -->	5.1-1_gnu
ca-certificates	2022.3.29-h06a4308_1 -->	2022.07.19-h06
certifi	2021.10.8-py37h06a4308_2 -->	2022.9.14-py37
cffi	1.15.0-py37hd667e15_1 -->	1.15.1-py37h74
conda	4.12.0-py37h06a4308_0 -->	4.14.0-py37h06
conda-package-handling	1.8.1-py37h7f8727e_0 -->	1.9.0-py37h5ee
cryptography	36.0.0-py37h9ce1e76_0 -->	37.0.1-py37h9c
ld_impl_linux-64	2.35.1-h7274673_9 -->	2.38-h1181459_
libgcc-ng	9.3.0-h5101ec6_17 -->	11.2.0-h123456
libgomp	9.3.0-h5101ec6_17 -->	11.2.0-h123456
libstdcxx-ng	9.3.0-hd4cf53a_17 -->	11.2.0-h123456
ncurses	6.3-h7f8727e_2 -->	6.3-h5eee18b_3
openssl	1.1.1n-h7f8727e_0 -->	1.1.1q-h7f8727
pip	21.2.2-py37h06a4308_0 -->	22.1.2-py37h06
requests	pkgs/main/noarch::requests-2.27.1-pyh~ -->	pkgs/main/linu
setuptools	61.2.0-py37h06a4308_0 -->	63.4.1-py37h06
sqlite	3.38.2-hc218d9a_0 -->	3.39.2-h508229
tk	8.6.11-h1ccaba5_0 -->	8.6.12-h1ccaba
tqdm	pkgs/main/noarch::tqdm-4.63.0-pyhd3eb~ -->	pkgs/main/linu
urllib3	pkgs/main/noarch::urllib3-1.26.8-pyhd~ -->	pkgs/main/linu
...

xz

5.2.5-n/D644/C_0 --> 5.2.5-n/r8/z/e

zlib

1.2.12-h7f8727e_1 --> 1.2.12-h5eee18

Downloading and Extracting Packages

libstdc++-ng-11.2.0	4.7 MB	: 100% 1.0/1 [00:00<00:00, 6.85it/s]
toolz-0.11.2	49 KB	: 100% 1.0/1 [00:00<00:00, 26.02it/s]
openssl-1.1.1q	2.5 MB	: 100% 1.0/1 [00:00<00:00, 11.84it/s]
cffi-1.15.1	227 KB	: 100% 1.0/1 [00:00<00:00, 24.22it/s]
tk-8.6.12	3.0 MB	: 100% 1.0/1 [00:00<00:00, 9.73it/s]
cryptography-37.0.1	1.3 MB	: 100% 1.0/1 [00:00<00:00, 8.27it/s]
setuptools-63.4.1	1.1 MB	: 100% 1.0/1 [00:00<00:00, 9.96it/s]
zlib-1.2.12	103 KB	: 100% 1.0/1 [00:00<00:00, 25.84it/s]
sqlite-3.39.2	1.1 MB	: 100% 1.0/1 [00:00<00:00, 17.19it/s]
cytoolz-0.11.0	328 KB	: 100% 1.0/1 [00:00<00:00, 22.31it/s]
certifi-2022.9.14	155 KB	: 100% 1.0/1 [00:00<00:00, 24.72it/s]
requests-2.28.1	92 KB	: 100% 1.0/1 [00:00<00:00, 28.94it/s]
pip-22.1.2	2.4 MB	: 100% 1.0/1 [00:00<00:00, 6.58it/s]
tqdm-4.64.0	126 KB	: 100% 1.0/1 [00:00<00:00, 23.60it/s]
conda-4.14.0	909 KB	: 100% 1.0/1 [00:00<00:00, 12.28it/s]
ca-certificates-2022	124 KB	: 100% 1.0/1 [00:00<00:00, 32.69it/s]
libgcc-ng-11.2.0	5.3 MB	: 100% 1.0/1 [00:00<00:00, 8.00it/s]
ld_impl_linux-64-2.3	654 KB	: 100% 1.0/1 [00:00<00:00, 23.50it/s]
ncurses-6.3	781 KB	: 100% 1.0/1 [00:00<00:00, 5.65it/s]
_openmp_mutex-5.1	21 KB	: 100% 1.0/1 [00:00<00:00, 20.54it/s]
conda-package-handli	887 KB	: 100% 1.0/1 [00:00<00:00, 19.72it/s]
libgomp-11.2.0	474 KB	: 100% 1.0/1 [00:00<00:00, 20.97it/s]
urllib3-1.26.11	181 KB	: 100% 1.0/1 [00:00<00:00, 23.81it/s]
xz-5.2.5	339 KB	: 100% 1.0/1 [00:00<00:00, 22.65it/s]

Preparing transaction: done

Verifying transaction: done

Executing transaction: done

!conda --version

!python --version

conda 4.14.0

Python 3.7.13

!conda install -c conda-forge pyfmi==2.7.4 --yes # Install the key package

liblapack	conda-forge/linux-64::liblapack-0.3.0-15_linux64_openblas
libopenblas	conda-forge/linux-64::libopenblas-0.3.20-pthreads_h78a641
libxml2	conda-forge/linux-64::libxml2-2.9.12-h72842e0_0
libxslt	conda-forge/linux-64::libxslt-1.1.33-h15afd5d_2
lxml	conda-forge/linux-64::lxml-4.8.0-py37h540881e_2
metis	conda-forge/linux-64::metis-5.1.0-h58526e2_1006
mpfr	conda-forge/linux-64::mpfr-4.1.0-h9202a9a_1
numpy	conda-forge/linux-64::numpy-1.21.6-py37h976b520_0
pyfmi	conda-forge/linux-64::pyfmi-2.7.4-py37h161383b_0
python_abi	conda-forge/linux-64::python_abi-3.7-2_cp37m
scipy	conda-forge/linux-64::scipy-1.7.3-py37hf2a6cf1_0
suitesparse	conda-forge/linux-64::suitesparse-5.10.1-h9e50725_1
sundials	conda-forge/linux-64::sundials-5.8.0-h558c624_0
tbb	conda-forge/linux-64::tbb-2021.5.0-h924138e_1

The following packages will be UPDATED:

ca-certificates pkgs/main::ca-certificates-2022.07.19~ --> conda-forge::c

The following packages will be SUPERSEDED by a higher-priority channel:

```
certifi          pkgs/main/linux-64::certifi-2022.9.14~ --> conda-forge/nc
conda            pkgs/main::conda-4.14.0-py37h06a4308_0 --> conda-forge::c
openssl         pkgs/main::openssl-1.1.1q-h7f8727e_0 --> conda-forge::c
```

Downloading and Extracting Packages

```
metis-5.1.0      | 4.1 MB | : 100% 1.0/1 [00:00<00:00, 1.84it/s]
icu-68.2         | 13.1 MB | : 100% 1.0/1 [00:01<00:00, 1.47s/it]
certifi-2022.9.14 | 156 KB | : 100% 1.0/1 [00:00<00:00, 20.50it/s]
libxml2-2.9.12   | 772 KB | : 100% 1.0/1 [00:00<00:00, 7.85it/s]
libblas-3.9.0    | 12 KB | : 100% 1.0/1 [00:00<00:00, 28.34it/s]
python_abi-3.7   | 4 KB | : 100% 1.0/1 [00:00<00:00, 26.12it/s]
tbb-2021.5.0     | 1.9 MB | : 100% 1.0/1 [00:00<00:00, 3.69it/s]
lxml-4.8.0       | 1.4 MB | : 100% 1.0/1 [00:00<00:00, 4.59it/s]
libcblas-3.9.0   | 12 KB | : 100% 1.0/1 [00:00<00:00, 22.37it/s]
suitesparse-5.10.1 | 2.4 MB | : 100% 1.0/1 [00:00<00:00, 2.95it/s]
pyfmi-2.7.4      | 12.4 MB | : 100% 1.0/1 [00:01<00:00, 1.13s/it]
libgfortran-ng-12.1. | 23 KB | : 100% 1.0/1 [00:00<00:00, 23.77it/s]
scipy-1.7.3      | 21.8 MB | : 100% 1.0/1 [00:02<00:00, 2.68s/it]
libopenblas-0.3.20 | 10.1 MB | : 100% 1.0/1 [00:01<00:00, 1.27s/it]
assimulo-3.2.9   | 2.6 MB | : 100% 1.0/1 [00:00<00:00, 2.70it/s]
liblapack-3.9.0  | 12 KB | : 100% 1.0/1 [00:00<00:00, 36.89it/s]
conda-4.14.0     | 1010 KB | : 100% 1.0/1 [00:00<00:00, 4.46it/s]
sundials-5.8.0   | 1.0 MB | : 100% 1.0/1 [00:00<00:00, 5.26it/s]
libgfortran5-12.1.0 | 1.8 MB | : 100% 1.0/1 [00:00<00:00, 3.70it/s]
numpy-1.21.6     | 6.1 MB | : 100% 1.0/1 [00:00<00:00, 1.07it/s]
libiconv-1.16    | 1.4 MB | : 100% 1.0/1 [00:00<00:00, 6.68it/s]
fmilib-2.2.3     | 532 KB | : 100% 1.0/1 [00:00<00:00, 8.92it/s]
openssl-1.1.1o   | 2.1 MB | : 100% 1.0/1 [00:00<00:00, 3.84it/s]
mpfr-4.1.0       | 2.6 MB | : 100% 1.0/1 [00:00<00:00, 3.14it/s]
ca-certificates-2022 | 152 KB | : 100% 1.0/1 [00:00<00:00, 22.82it/s]
gmp-6.2.1        | 806 KB | : 100% 1.0/1 [00:00<00:00, 7.26it/s]
libxslt-1.1.33   | 522 KB | : 100% 1.0/1 [00:00<00:00, 7.76it/s]
```

Preparing transaction: done

Verifying transaction: done

Executing transaction: done

Retrieving notices: ...working... done

```
!conda install numpy=1.19.1 --yes # Need to downgrade numpy
```

Collecting package metadata (current_repodata.json): done

Solving environment: failed with initial frozen solve. Retrying with flexible

Collecting package metadata (repodata.json): done

Solving environment: done

Package Plan

environment location: /usr/local

added / updated specs:

- numpy=1.19.1

The following packages will be downloaded:

package	build
---------	-------

blas-1.0	openblas	46 KB
numpy-1.19.1	py37h30dfecb_0	21 KB
numpy-base-1.19.1	py37h75fe3a5_0	4.1 MB
Total:		4.2 MB

The following NEW packages will be INSTALLED:

blas	pkgs/main/linux-64::blas-1.0-openblas
numpy-base	pkgs/main/linux-64::numpy-base-1.19.1-py37h75fe3a5_0

The following packages will be UPDATED:

openssl	conda-forge::openssl-1.1.1o-h166bdaf_0 --> pkgs/main::ope
---------	-----------------------------------------------------------

The following packages will be SUPERSEDED by a higher-priority channel:

ca-certificates	conda-forge::ca-certificates-2022.9.1~ --> pkgs/main::ca-
certifi	conda-forge/noarch::certifi-2022.9.14~ --> pkgs/main/linu
conda	conda-forge::conda-4.14.0-py37h89c186~ --> pkgs/main::cor
numpy	conda-forge::numpy-1.21.6-py37h976b52~ --> pkgs/main::num

Downloading and Extracting Packages

numpy-base-1.19.1	4.1 MB	: 100% 1.0/1 [00:00<00:00, 4.55it/s]
numpy-1.19.1	21 KB	: 100% 1.0/1 [00:00<00:00, 17.29it/s]
blas-1.0	46 KB	: 100% 1.0/1 [00:00<00:00, 30.59it/s]

Preparing transaction: done

Verifying transaction: done

Executing transaction: done

Retrieving notices: ...working... done

Now specific installation and the run simulations. Start with connecting to Github. Then upload the four files:

- FMU - BPL_TEST2_Batch_linux_jm_cs.fmu
- Setup-file - BPL_TEST2_Batch_explore.py

```
# Filter out DeprecationWarnings for 'np.float as alias' is needed - wish I could m
import warnings
warnings.filterwarnings("ignore")
```

```
%bash
```

```
git clone https://github.com/janpeter19/BPL_TEST2_Batch_calibration
```

```
Cloning into 'BPL_TEST2_Batch_calibration'...
```

```
%cd BPL_TEST2_Batch_calibration
```

```
/content/BPL_TEST2_Batch_calibration
```

▼ BPL_TEST2_Batch_calibration - demo

This notebook shows the possibilities for calibration of the model BPL_TEST2_Batch using `scipy.optimize.minimize()` routine. There are several different methods to choose between. In this notebook we work with simulated data.

The text-book model of batch cultivation we simulate is the following where S is substrate, X is cell concentration, and V is volume of the broth

$$\frac{d(VS)}{dt} = -q_S(S) \cdot VX$$

$$\frac{d(VX)}{dt} = \mu(S) \cdot VX$$

and where specific cell growth rate μ and substrate uptake rate q_S are

$$\mu(S) = Y \cdot q_S(S)$$

$$q_S(S) = q_S^{max} \frac{S}{K_s + S}$$

where Y is the yield, q_S^{max} is the maximal specific substrate uptake rate and K_s is the corresponding saturation constant.

The parameter estimation is done with optimization methods that only require evaluation of the mismatch between simulation with given parameters and data. At start the allowed range for each parameter is given. The method used for optimization is SLSQP but can easily be changed [1].

In the near future the FMU may provide first derivative gradient information, that will make it possible to choose corresponding method of `minimize()` for improved performance. This possibility is related to the upgrade to the FMI-standard ver 3.0 for the Modelica compiler.

The Python package PyFMI [2] that is the base for FMU-explore has a simplified built-in functionality for parameter estimation that also use `scipy.optimize.minimize()`. However, there is estimated and the purpose seems to only address smaller examples. Therefore we here define a no possibility to include parameter changes to the compiled model that should not be Python function evaluation() that facilitate the formulation of the parameter estimation and bring flexibility to choice of optimization method.

```
run -i BPL_TEST2_Batch_explore.py
```

```
Linux - run FMU pre-compiled JModelica 2.4
```

```
Model for bioreactor has been setup. Key commands:
```

- `par()` - change of parameters and initial values
- `init()` - change initial values only
- `simu()` - simulate and plot
- `newplot()` - make a new plot

- show() - show plot from previous simulation
- disp() - display parameters and initial values from the last simulation
- describe() - describe culture, broth, parameters, variables with values /

Note that both disp() and describe() takes values from the last simulation

Brief information about a command by help(), eg help(simu)

Key system information is listed with the command system_info()

```
# Adjust the size of diagrams
```

```
plt.rcParams['figure.figsize'] = [15/2.54, 12/2.54]
```

▼ 1 Generate data later used for parameter estimation

```
import pandas as pd
```

```
# Data generated
```

```
simulationTime = 6.0
```

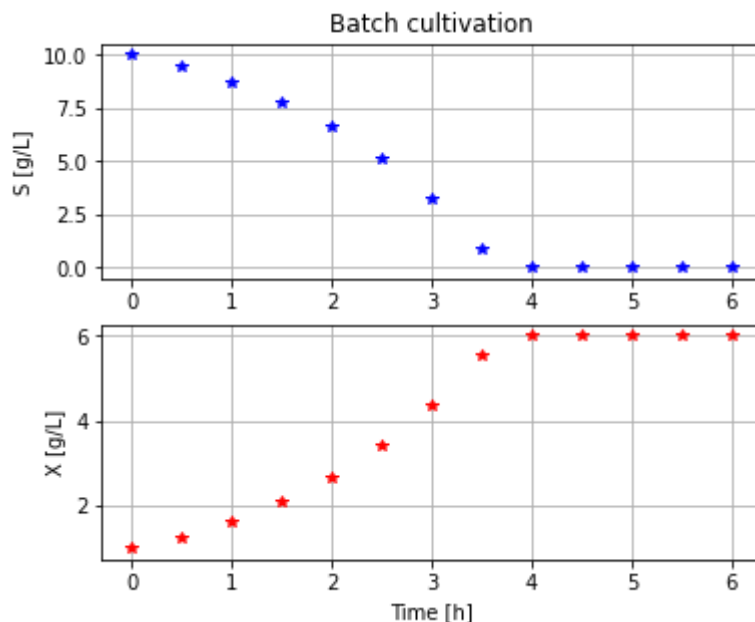
```
par(Y=0.50, qSmax=1.00, Ks=0.1)
```

```
init(V_0=1.0, VS_0=10, VX_0=1.0)
```

```
newplot(plotType='Demo_2')
```


```
opts['ncp'] = 12
```

```
simu(simulationTime)
```



```
# Store data in a DataFrame for later use
```

```
data = pd.DataFrame(data={'time':sim_res['time'], 'X':sim_res['bioreactor.c[1]'], 'S':sim_res['bioreactor.c[2]']})
```

	time	x	s	
0	0.0	1.000000	1.000000e+01	
1	0.5	1.280773	9.438453e+00	
2	1.0	1.640079	8.719842e+00	
3	1.5	2.099615	7.800770e+00	
4	2.0	2.686770	6.626459e+00	
5	2.5	3.435479	5.129043e+00	
6	3.0	4.385325	3.229350e+00	
7	3.5	5.559252	8.814967e-01	
8	4.0	6.000000	1.048673e-08	
9	4.5	6.000000	-6.547559e-11	

➤ 2 Simulation with initial guess of parameters compared with data

Here we define the parameters that should be estimated and specify allowed ranges. Nominal parameters are chosen as the mid-point of the allowed parameter range.

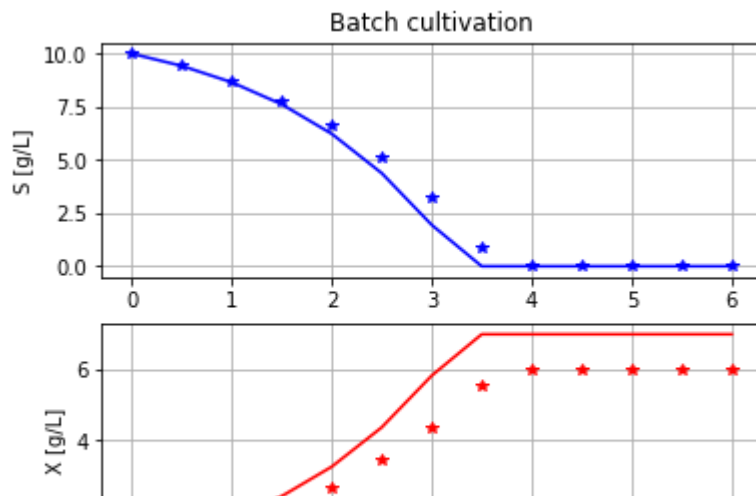
Simulation with these nominal parameter set and compare with data give an idea of how well the model fit data.

```
# Parameters to be estimated using parDict names and their bounds
parEstim = ['Y', 'qSmax', 'Ks']
parBounds = [(0.4, 0.8), (0.7, 1.3), (0.05, 0.20)]
parEstim_0 = [np.mean(parBounds[k]) for k in range(len(parBounds))]
```

```
# Parameters to be estimated using parDict names and their bounds
parEstim = ['Y', 'qSmax', 'Ks']
parBounds = [(0.4, 0.8), (0.7, 1.3), (0.05, 0.20)]
parEstim_0 = [np.mean(parBounds[k]) for k in range(len(parBounds))]
```

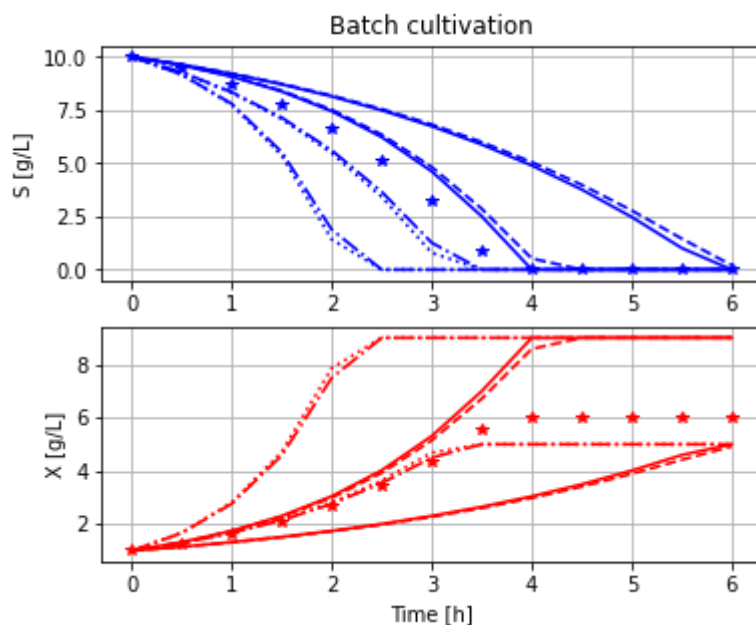
```
# Simulation with nominal parameters
newplot(plotType='Demo_1')
par(Y=parEstim_0[0], qSmax=parEstim_0[1], Ks=parEstim_0[2])
simu(simulationTime)
```

```
# Show data
ax1.plot(data['time'], data['S'], 'b*')
ax2.plot(data['time'], data['X'], 'r*')
plt.show()
```

```
# Simulation over the parameter ranges given
newplot(plotType='Demo_1')
for Y_value in parBounds [0]:
    for qSmax_value in parBounds[1]:
        for Ks_value in parBounds[2]:
            par(Y=Y_value, qSmax=qSmax_value, Ks=Ks_value)
            simu(simulationTime)

# Show data
ax1.plot(data['time'], data['S'],'b*')
ax2.plot(data['time'], data['X'],'r*')
plt.show()
```



Simulation over the different parameter combinations of the parameter bounds shows that data is "covered" and we have good hope to find a parameter combination that fits data well.

▼ 3 Parameter estimation

Here we use the `scipy.optimize.minimize()` procedure which contains a family of different methods [1]. Since we have chosen to work with bounds on the parameters to be estimated there

are only three methods to choose between. Here the method Sequential Least Squares Programming SLSQP is chosen.

Note that we in the definition of `evaluation()` make use of PyFMI-functions to administrate the simulation parameters as well as running it, instead of using the simplified `simu()` function we are used to.

```
# Optimization routine import
import scipy.optimize

# Parameters to be estimated using parDict names and their bounds
extra_args = (parEstim, data, fmu_model, simulationTime, parDict, parLocation)

# Modified evaluation function tailored for Python optimization algorithms
def evaluation(x, parEstim, data=data, fmu_model=fmu_model, simulationTime=simulationTime,
               parDict=parDict, parLocation=parLocation):
    """The parameter list is tailored for scipy optimization algorithms interface,
    where the first parameter x is an array with parameters that are tuned
    and evaluated and parEstim is a list of the names of these parameters."""

    # Load model
    global model
    if model is None:
        model = load_fmu(fmu_model)
    model.reset()

    # Change parameters and initial values from default
    for i, p in enumerate(parEstim): model.set(parLocation[p], x[i])
    for p in set(parDict)-set(parEstim): model.set(parLocation[p], parDict[p])

    # Simulation options
    opts = model.simulate_options()
    opts['ncp'] = 12
    opts['result_handling'] = 'memory'
    opts['silent_mode'] = True

    # Simulate
    sim_res = model.simulate(start_time=0.0, final_time=simulationTime, options=opts)

    # Calculate loss function V
    V={}
    V['X'] = np.linalg.norm(data['X'] - np.interp(data['time'], sim_res['time'], sim_res['X']))
    V['S'] = np.linalg.norm(data['S'] - np.interp(data['time'], sim_res['time'], sim_res['S']))

    return V['X'] + V['S']

import time

# Run minimize()
```

```

start_time = time.time()
result = scipy.optimize.minimize(evaluation, x0=parEstim_0, args=extra_args,
                                method='SLSQP', bounds=parBounds, options={"disp":
print('CPU-time =', time.time()-start_time)

```

```

Optimization terminated successfully      (Exit mode 0)
      Current function value: 0.00023713755940405281
      Iterations: 21
      Function evaluations: 114
      Gradient evaluations: 21
CPU-time = 0.14707636833190918

```

```
result
```

```

      fun: 0.00023713755940405281
      jac: array([ 5.38155632,  2.80042532, -0.71099749])
message: 'Optimization terminated successfully'
      nfev: 114
       nit: 21
      njev: 21
      status: 0
     success: True
         x: array([0.49999989, 0.99996316, 0.09977226])

```

The estimated parameters `result.x` are very close to the original values and no surprise.

Test of the three methods available that handle parameter bounds: TNC, L-BFGS-B and SLSQP. It turns out that SLSQP is by far the fastest. It is 3 times faster than L-BFGS-B which is faster than TNC. Can be that SLSQP is less robust though. The nit (number of iterations) does not differ that much though: 24 vs 30. The nfev (number of function evaluations) is perhaps more important 127 vs 256. A more precise timer function is likely `timeit` for this short times.

The Nelder-Mead algorithm has a good reputation to be very robust, but more slow, and with this method we cannot have bounds on the parameters.

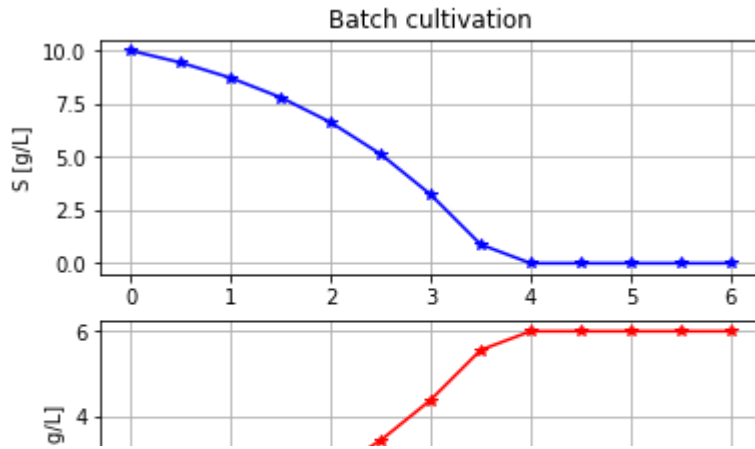
▼ 4 Simulation with estimated parameters compared with data

```

newplot(plotType='Demo_1')
par(Y=result.x[0], qSmax=result.x[1], Ks=result.x[2])
simu(simulationTime)

# Show data
ax1.plot(data['time'], data['S'], 'b*')
ax2.plot(data['time'], data['X'], 'r*')
plt.show()

```



```
# The estimated parameters are
for i in range(len(parEstim)): print(parEstim[i],':', result.x[i])
```

```
Y : 0.49999989477908324
qSmax : 0.9999631628019081
Ks : 0.09977225678489653
```

▼ 5 Analysis of the loss function

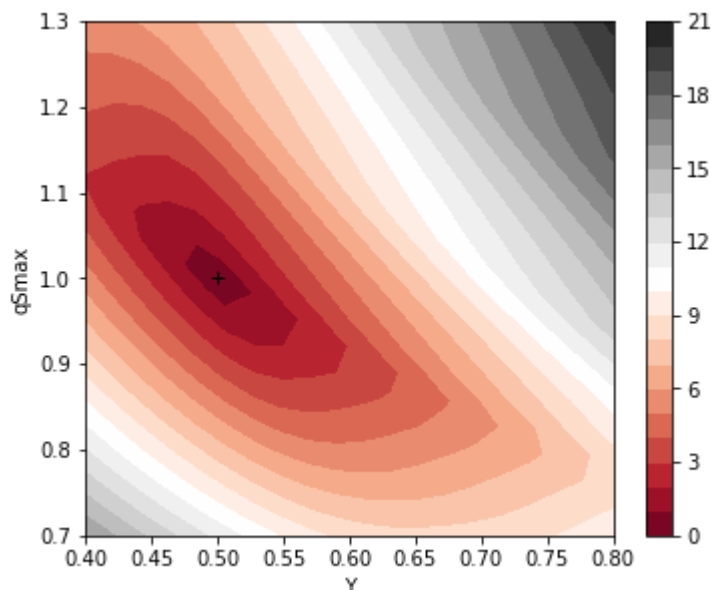
The problem is small and analysis of the loss function brings some insight. From the diagram above showing parameter sweep over combinations min- and max-parameters we see that the parameter K_s has little influence. Let us set that a fixed value and then plot the loss function in the parameters Y and $qSmax$. We do this by go through all the parametera combinations and evaluate each of them.

```
# Sweep through Y and qSmax variation and store the value of the loss-function for
nY = 20
nqSmax = 20
V = np.zeros((nY, nqSmax))

Y = np.linspace(parBounds[0][0],parBounds[0][1],nY)
qSmax = np.linspace(parBounds[1][0],parBounds[1][1],nqSmax)

for j in range(nY):
    for k in range(nqSmax):
        V[k,j] = evaluation([Y[j], qSmax[k], 0.1], parEstim)

# Contour plot
plt.figure()
plt.clf
plt.subplot(1,1,1)
plt.contourf(Y, qSmax, V, 20, cmap='RdGy')
plt.plot(result.x[0], result.x[1], 'k+')
plt.colorbar()
plt.ylabel('qSmax')
plt.xlabel('Y')
plt.show()
```



We see the following in the contour diagram of the loss function simplified:

- The minima is unique in the range of parameters we study. This is good news.
- The contour plot is ellipsoid and rather narrow. The more narrow the ellipsoid the more difficult and more time it takes to converge to the minima.
- The direction of the ellipsoid axis indicate the correlation you may get between the two parameters during the minimization process.

Note that the form of the contour plot change with the parameters (and initial values) of the actual process. You can see the impact by changing the parameters in "cell # 4" where data is generated and then just choose to run that cell and the cells below. No need to restart the notebook.

6 Summary

A choice was made to work with allowed ranges of parameters to be estimated and a start value was defined as the center point in this parameter space. There are only three methods available in `optimize.minimize()` that can handle bounds on parameters.

An `evaluation()` function was created that define how the difference between simulation and data is measured. The function is rather transparent and easy to modify and you may want to change weight on the loss in S and X, for instance. Here they have so far equal weight.

The FMU-explore workspace dictionaries `partDict[]` and `parLocation[]` are useful also here and simplify the code for the `evaluation()` function. But we also use the detailed PyFMI-functions to administrate and set parameters of the actual simulation.

The call `optimize.minimize()` has several parameters and can easily be modified, for instance change of method.

The estimated parameters were close to perfect!

The contour plot of the simplified loss function shows that the minima is unique and should not be difficult too difficult to obtain.

7 References

[1] Scipy Reference guide on optimize.minimize() [here](#)

[2] Andersson, C., Åkesson, J., Fuhrer C. : "PyFMI: A Python package for simulation of coupled dynamic models with the functional mock-up interface", Centre for Mathematical Sciences, Lund University, Report LUTFNA-5008-2016, 2016.

▼ Appendix

```
describe('parts')
```

```
['bioreactor', 'bioreactor.culture', 'liquidphase', 'MSL']
```

```
describe('MSL')
```

```
MSL: 3.2.2 build 3 - used components:
```

```
scipy.__version__
```

```
'1.7.3'
```

```
system_info()
```

```
System information
```

```
-OS: Linux
```

```
-Python: 3.7.14
```

```
-PyFMI: 2.7.4
```

```
-FMU by: JModelica.org
```

```
-FMI: 2.0
```

```
-Type: FMUModelCS2
```

```
-Name: BPL_TEST2.Batch
```

```
-Generated: 2022-09-19T14:20:20
```

```
-MSL: 3.2.2 build 3
```

```
-Description: Bioprocess Library version 2.1.0 beta
```

```
-Interaction: FMU-explore ver 0.9.3
```

[Colab paid products](#) - [Cancel contracts here](#)

✓ 0s completed at 13:27 ● ✕