BPL_TEST2_Batch_calibration - demo

This notebook shows the possibilities for calibration of the model BPL_TEST2_Batch using scipy.optimize.minimize() routine.

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

In [1]:
```
run -i BPL_TEST2_Batch_explore.py
```

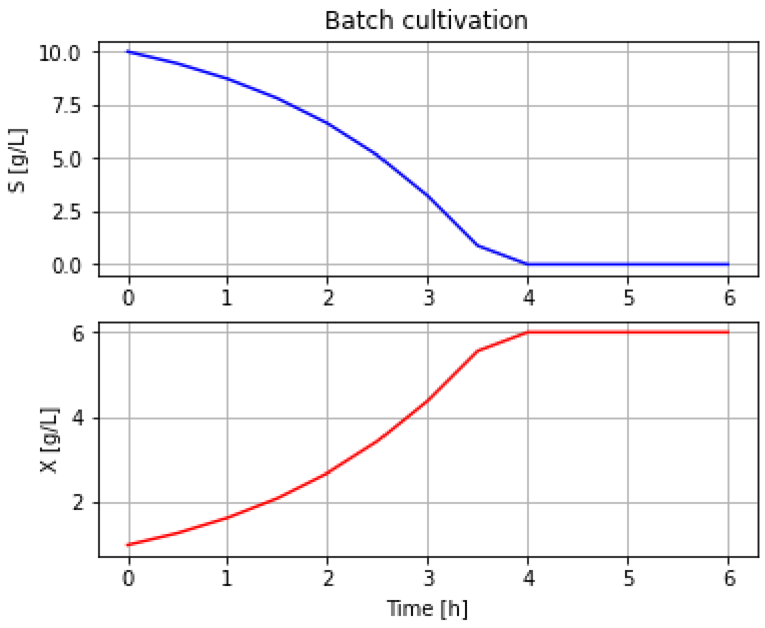
```
Windows - run FMU pre-compiled JModelica 2.14

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 / units

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()
```

In [2]:
```
# Adjust the size of diagrams
plt.rcParams['figure.figsize'] = [15/2.54, 12/2.54]
```

A first simulation and store data in a dataframe

In [3]:
```
import pandas as pd
```

In [4]:
```
par(Y=0.5, qSmax=1.0, Ks=0.1)
init(V_0=1.0, VS_0=10, VX_0=1.0)
newplot(plotType='Demo_1')
opts['ncp'] = 12
simu(6)
```



In [5]:
```
# Store data in a DataFrame for later use
data = pd.DataFrame(data={'time':sim_res['time'], 'X':sim_res['bioreactor.c[1]'], 'S
data
```

Out[5]:

	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	1.048375e-08
9	4.5	6.000000	-1.936268e-10
10	5.0	6.000000	2.156125e-12
11	5.5	6.000000	9.975889e-14
12	6.0	6.000000	4.189854e-15

Try minimize() for parameter estimation

Here we try the same scipy-algorithm family that is actually used in pyfmi model.estimate()

In [6]:
```
# For clarity of code here we import again
import scipy.optimize
```

In [7]:
```
# Parameters to be estimated using parDict names and their bounds
```

```
parEstim = ['Y', 'qSmax', 'Ks']
parEstim_0 = np.array([0.4, 1.2, 0.15])

extra_args = (parEstim, data, 'BPL_TEST2_Batch_windows_jm_cs.fmu', parDict, parLocat
```

In [8]:

```
# Modified evaluation function tailored for Python optimization algorithms
def evaluation(x, parEstim, data=data, fmu_model=fmu_model,
                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 evalauted."""

    # 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=6.0, options=opts)

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

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

In [9]:

```
# Run minimize()
result = scipy.optimize.minimize(evaluation, x0=parEstim_0, args=extra_args,
                                 method='Nelder-Mead', options={"disp":True})
#                                 method='BFGS', options={"disp":True})
```

```
Optimization terminated successfully.
         Current function value: 0.069422
         Iterations: 42
         Function evaluations: 83
```

In [10]:

```
result
```

Out[10]:

```
 final_simplex: (array([[0.50010848, 1.01030323, 0.16246785],
       [0.50010547, 1.01023377, 0.16248744],
       [0.50007476, 1.01031636, 0.16249674],
       [0.50015623, 1.01021132, 0.16250092]]), array([0.06942214, 0.06944698, 0.0694
6086, 0.06947121]))
           fun: 0.06942213758121146
       message: 'Optimization terminated successfully.'
          nfev: 83
           nit: 42
        status: 0
```

```
     success: True
           x: array([0.50010848, 1.01030323, 0.16246785])
```

The estimated parameters x are very close to the original, but why not a success?

But if a use method 'Nelder-Mead' instead of the default 'BFGS' then I get success but needs doulbe amount of function evaluations.

Concluding remarks

- It is rather easy to make scipy function evaluations with flexibility to use compiled FMUs and provide parameters different fram default and that should not be tuned.
- FMU-explore with workspace dictionaries parDict[] and parLocation[] etc are useful also in this scipy-optimization context. We see also that the broad optimization family scipy.minimization() used in pyfmi for model.estimate() does work.

Appendix

In [11]:
```
describe('parts')
```

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

In [12]:
```
describe('MSL')
```

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

In [13]:
```
system_info()
```

```
System information
 -OS: Windows
 -Python: 3.9.5
 -PyFMI: 2.9.5
 -FMU by: JModelica.org
 -FMI: 2.0
 -Type: FMUModelCS2
 -Name: BPL_TEST2.Batch
 -Generated: 2022-09-13T11:19:04
 -MSL: 3.2.2 build 3
 -Description: Bioprocess Library version 2.1.0 beta
 -Interaction: FMU-explore ver 0.9.3
```

In [14]:
```
optse = model.estimate_options()
```

In [15]:
```
optse
```

Out[15]:
```
{'tolerance': 1e-06,
 'result_file_name': '',
 'filter': None,
 'method': 'Nelder-Mead',
 'scaling': 'Default',
 'simulate_options': 'Default'}
```

In [ ]:
```

```