Experience with Google Colab for running Modelica FMU in notebooks and no installations

Jan Peter Axelsson

E-mail: jan.peter.axelsson@vascaia.se

Vascaia AB, Stockholm, Sweden

OpenModelica workshop 2025-02-03

Background

- Simplify *usage* of simulation (of bioprocesses)
- Part of larger story
  - Teaching
  - Development
  - Sensitivity analysis
  - Calibration
  - Optimization
  - Control system
  - …
- Protect IP of model
- Readable interaction
- Minimize installation/maintenance questions

Background

- Simplify *usage* of simulation (of bioprocesses)
- Part of larger story – **Jupyter notebook with Python**
  - Teaching
  - Development
  - Sensitivity analysis
  - Calibration
  - Optimization
  - Control system
  - …
- Protect IP of model – **FMU shared (i.e. not Modelica code)**
- Readable interaction – **FMU-explore "package"**
- Minimize installation questions – **Use Google Colab + Github**

Outline

- Background
- Google Colab
- Hands on – series of screenshots Github and Golab
- FMU-explore “package”
- FMU re-use different simulation tasks
- Concluding remarks

Google Colab

- Google provides: Gmail, Drive, Colab…
- Colab
  - Private VM Ubuntu LTS for 24 hours in the cloud
  - Jupyter notebook and Python pre-installed
  - Performance like a good laptop
  - Today 3 VM can run in parallell
- Colab interact with
  - Drive - user
  - Github - general

Google Colab experience

- Experience over about 2 years
- Works all the time
- Updates of VM software – litte information from Google?
- Colab notebook
  - Branch of an older Jupyter notebook
  - Jupyter notebook locally – transfer to Colab notebook
  - Widgets need different implementation – complicates usage
  - Printing of notebook to PDF difficult…
- Wish to improve
  - Centralize common setup, shorten start-up time
  - Printing of notebooks

Jan Peter Axelsson

janpeter19

Edit profile

4 followers · **0** following

Vascaia AB

Stockholm

Achievements

janpeter19 / README.md

- 👋 Hi, I'm @janpeter19
- 👀 I'm interested in Modelica and especially biotechnical applications, see paper.
- 🌱 I'm currently learning how to reach out more using Google Colab. I have prepared several examples for you starting with the repository BPL_TEST2_Batch. Here you follow a Jupyter notebook and can continue interact using Python and modifiy the simulations and graphs shown. More about the technology used here and about the modelling here. No installation needed!

  Note 2025-01-25 Please use notebooks using FMPy instead of PyFMI. There is an installation problem with the notebooks using PyFMI and reported and got issue number #287.

  Note 2024-11-11 The BPL is updated to ver 2.3.0 and now used in all examples. The GUI part has been developed in parallell with the main development of the library for more than a year. Now it is fully integrated and this update is an important consolidating step. Focus has been to use standard Modelica GUI facilities and to simplify the code. The main structure of the code still follows well the outline in section 6 in the paper referred above. The library has been tested mainly with the GUI of OpenModelica and Modelon Impact.

  Earlier notes you find here.
- 💞 I'm looking forward to collaborate on expanding the examples of use of Modelica in Colab. I also tailor-make models of biotechnical processes for your needs on a consultancy basis. Work with processes involves broader data analysis and is a part of my work, and simulation is just one tool, but an important one. Digital Twins is a catchword these days and part of what I do.
- 📫 You can reach me at: jan.peter.axelsson@vascaia.se

Pinned

Customize your pins

BPL_TEST2_Batch Public

Material for standard text book model of batch cultivation. Updated.

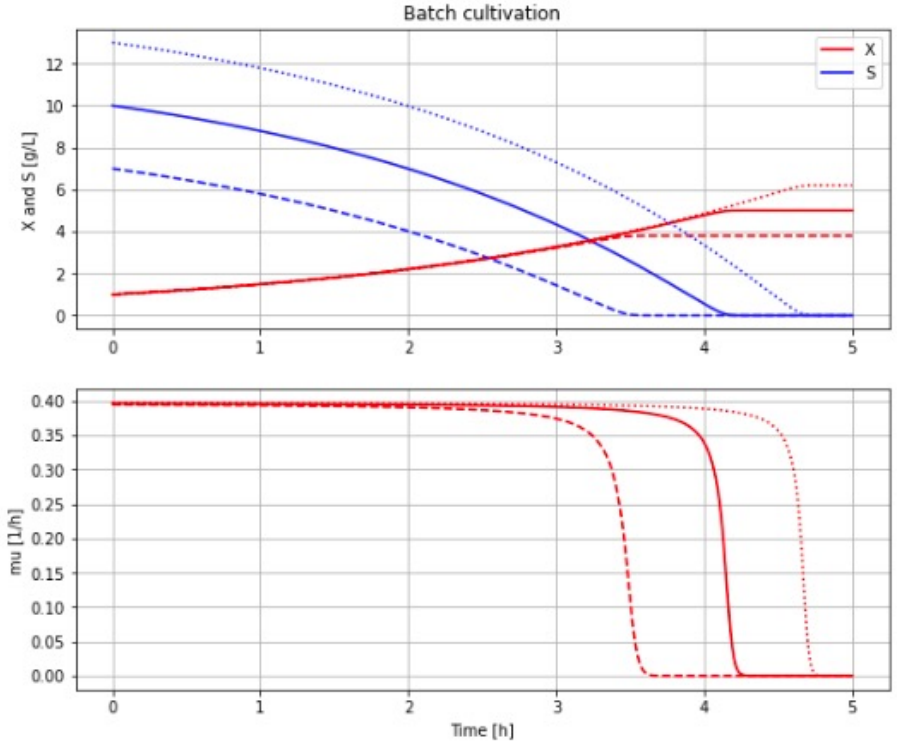
Jupyter Notebook ☆ 2

BPL_TEST2_Chemostat Public

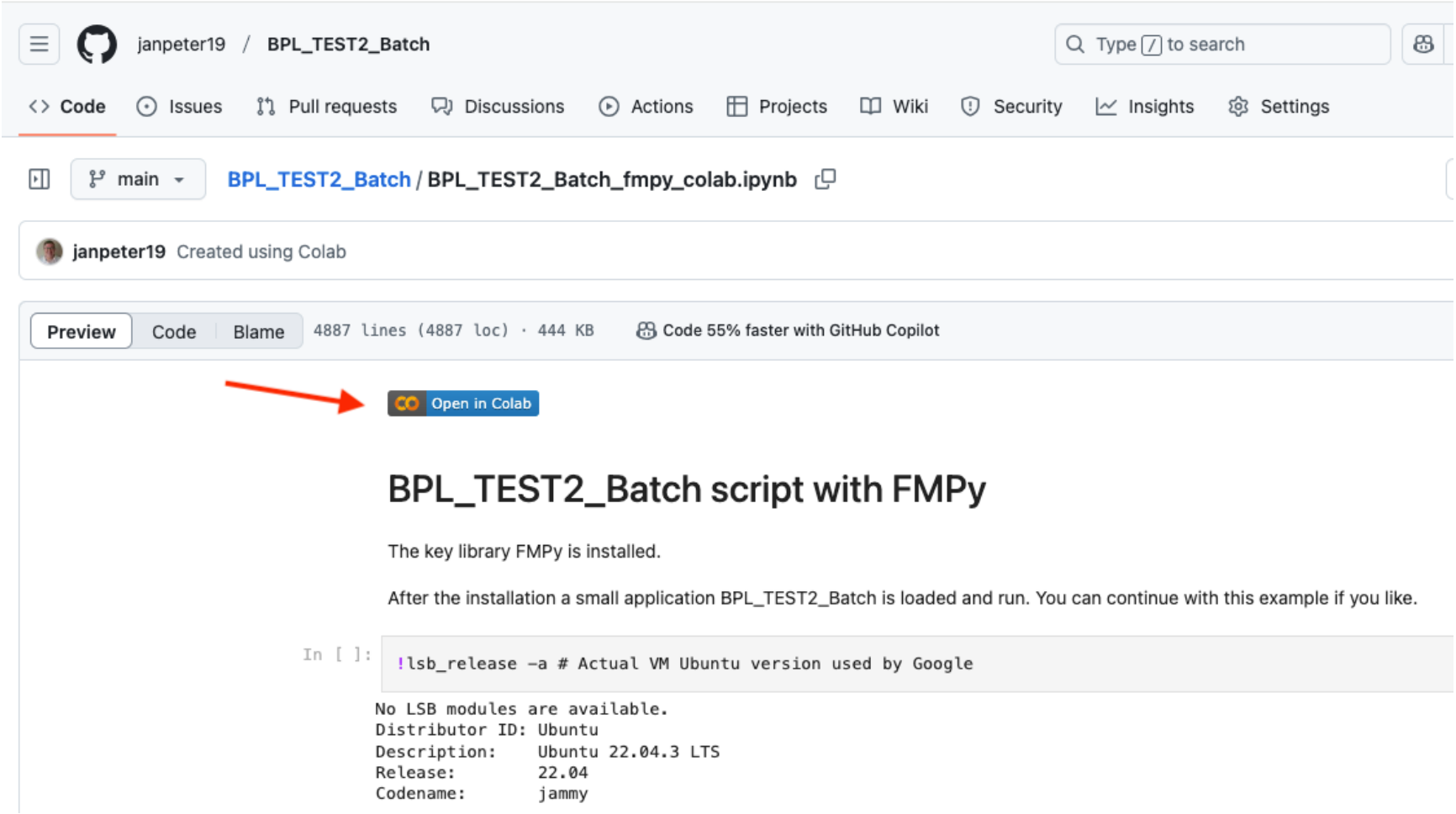
Material for standard text book model of chemostat (continuous) cultivation. Updated.

Jupyter Notebook ☆ 1

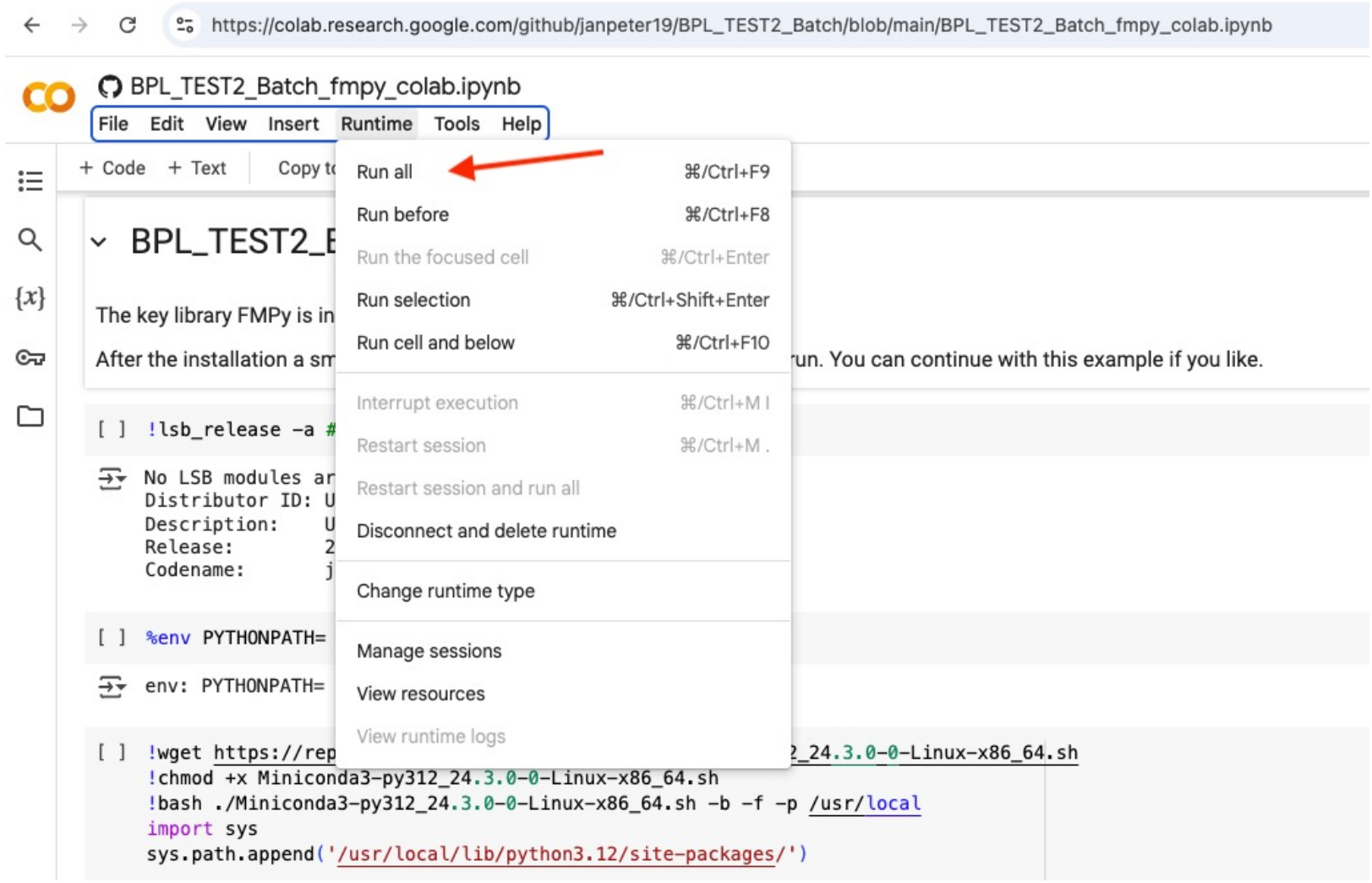
```
newplot(plotType='TimeSeries')
for value in [10, 7, 13]: init(VS_0=value); simu(5)
```



You start up the notebook in Colab by pressing here start BPL notebook or alternatively (experimentally) start BPL notebook with FMPy. Then you in the menu choose Runtime/Run all. If you have chosen the altarnative with FMPy click first on the symbol Open in Colab.


janpeter19 / BPL_TEST2_Batch
Type / to search
Code
Issues
Pull requests
Discussions
Actions
Projects
Wiki
Security
Insights
Settings
main
BPL_TEST2_Batch / BPL_TEST2_Batch_fmpy_colab.ipynb
janpeter19 Created using Colab
Preview
Code
Blame
4887 lines (4887 loc) · 444 KB
Code 55% faster with GitHub Copilot
Open in Colab
BPL_TEST2_Batch script with FMPy
The key library FMPy is installed.
After the installation a small application BPL_TEST2_Batch is loaded and run. You can continue with this example if you like.
In [ ]:
!lsb_release -a # Actual VM Ubuntu version used by Google
No LSB modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 22.04.3 LTS
Release: 22.04
Codename: jammy


BPL_TEST2_Batch_fmpy_colab.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text Copy to Drive
BPL_TEST2_Batch script with FMPy
The key library FMPy is installed.
After the installation a small application BPL_TEST2_Batch 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 22.04.3 LTS
Release: 22.04
Codename: jammy
[ ] %env PYTHONPATH=
env: PYTHONPATH=
[ ] !wget https://repo.anaconda.com/miniconda/Miniconda3-py312_24.3.0-0-Linux-x86_64.sh
!chmod +x Miniconda3-py312_24.3.0-0-Linux-x86_64.sh
!bash ./Miniconda3-py312_24.3.0-0-Linux-x86_64.sh -b -f -p /usr/local
import sys
sys.path.append('/usr/local/lib/python3.12/site-packages/')


https://colab.research.google.com/github/janpeter19/BPL_TEST2_Batch/blob/main/BPL_TEST2_Batch_fmpy_colab.ipynb
BPL_TEST2_Batch_fmpy_colab.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
Run all ⌘/Ctrl+F9
Run before ⌘/Ctrl+F8
Run the focused cell ⌘/Ctrl+Enter
Run selection ⌘/Ctrl+Shift+Enter
Run cell and below ⌘/Ctrl+F10
Interrupt execution ⌘/Ctrl+M I
Restart session ⌘/Ctrl+M .
Restart session and run all
Disconnect and delete runtime
Change runtime type
Manage sessions
View resources
View runtime logs
BPL_TEST2_
The key library FMPy is in
After the installation a sm
run. You can continue with this example if you like.
[ ] !lsb_release -a #
No LSB modules ar
Distributor ID: U
Description: U
Release: 2
Codename: j
[ ] %env PYTHONPATH=
env: PYTHONPATH=
[ ] !wget https://rep
2_24.3.0-0-Linux-x86_64.sh
!chmod +x Miniconda3-py312_24.3.0-0-Linux-x86_64.sh
!bash ./Miniconda3-py312_24.3.0-0-Linux-x86_64.sh -b -f -p /usr/local
import sys
sys.path.append('/usr/local/lib/python3.12/site-packages/')

BPL_TEST2_Batch setup

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

- FMU - BPL_TEST2_Batch_linux_om_me.fmu
- Setup-file - BPL_TEST2_Batch_fmpy_explore.py

```
[11] %%bash
     git clone https://github.com/janpeter19/BPL_TEST2_Batch
```

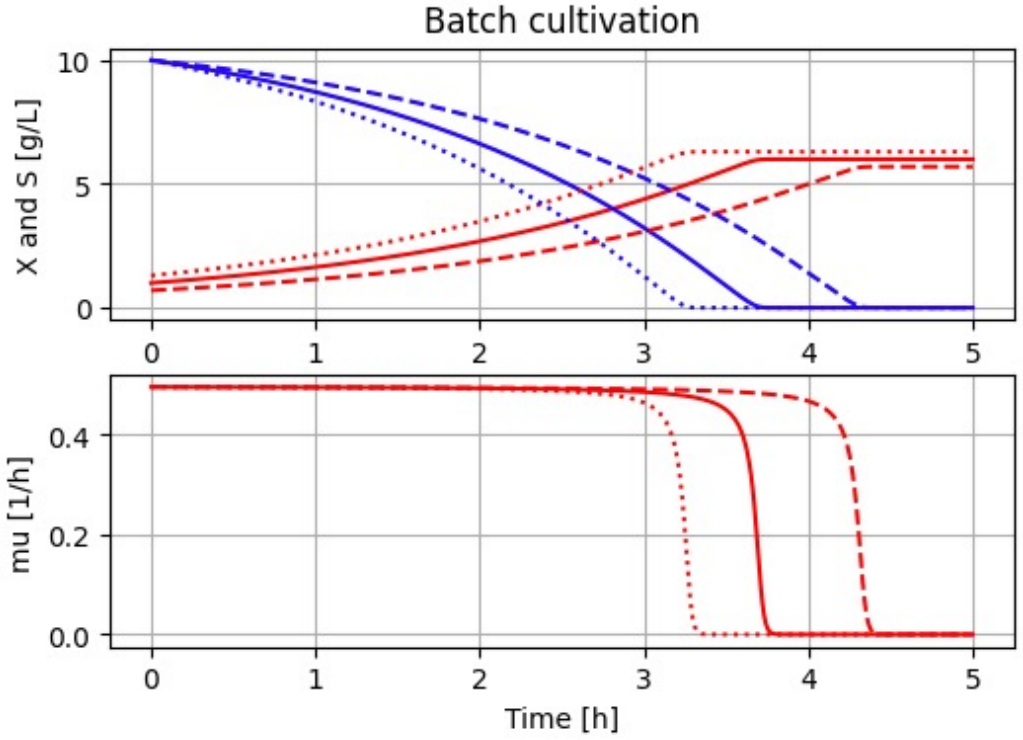
```
Cloning into 'BPL_TEST2_Batch'...
```

```
[12] %cd BPL_TEST2_Batch
```

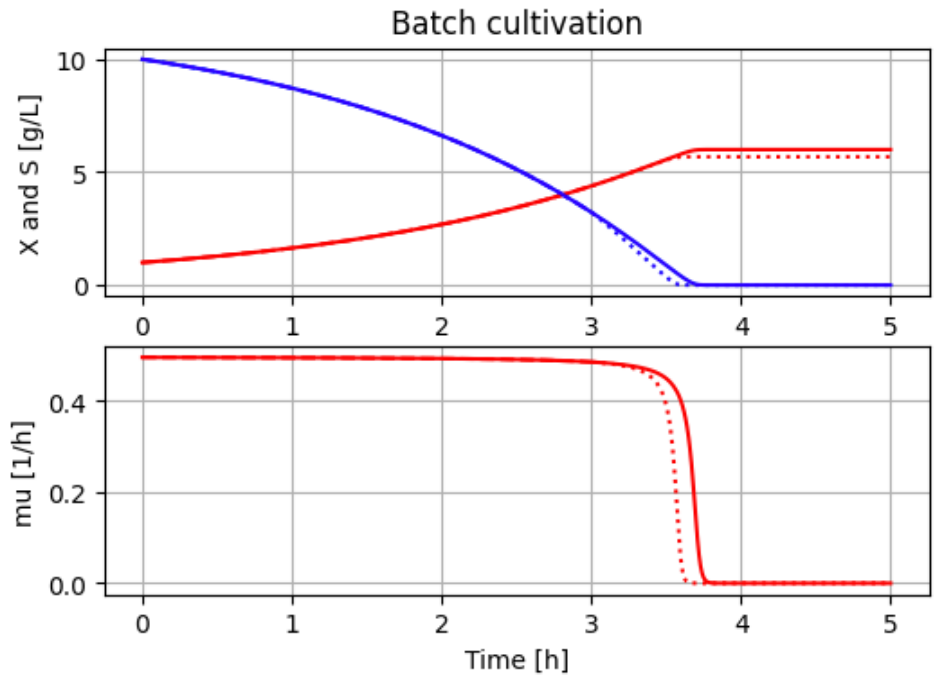
```
/content/BPL_TEST2_Batch
```

```
[13] run -i BPL_TEST2_Batch_fmpy_explore.py
```

```
[30] # Simulation were initial value of biomass VX_start is varied
     newplot(plotType='TimeSeries')
     for value in [1.0, 0.7, 1.3]: init(VX_start=value); simu(5)

     # Restore default value of VX_start
     init(VX_start=1.0)
```



```
[31] # Simulation where metabolism is changed after 3 hours
     newplot(plotType='TimeSeries')
     simu(5)

     simu(3)
     par(Y=0.4, qSmax=1.0/(0.4/0.5)); simu(2, 'cont')

     # Restore default value of Y and qSmax
     par(Y=0.5, qSmax=1.0)
```



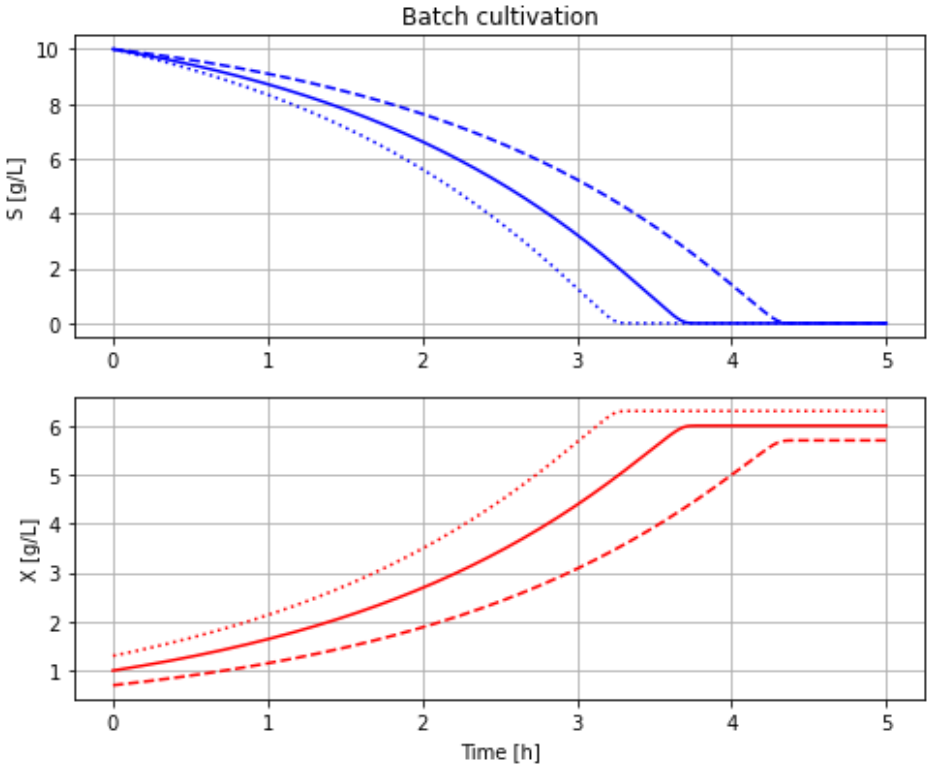
Outline

- Background
- Google Colab
- Hands on – series of screenshots Github and Golab
- **FMU-explore “package”**
- FMU re-use different simulation tasks
- Concluding remarks

FMU-explore “package”

- Command-line interaction vs scripting

A simple example



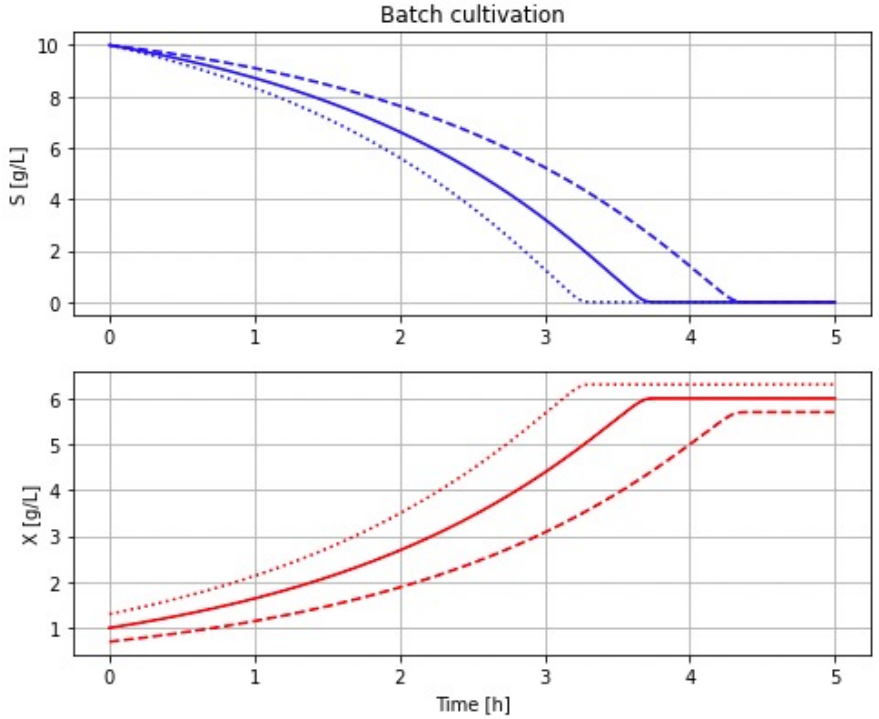
Scripting using PyFMI

In [9]:
```
# Setup-script defines fmu_model, parDict[], parLocation[] and some more

# - newplot()
plt.figure()
ax1=plt.subplot(2,1,1); ax1.grid(); ax1.set_ylabel('S [g/L]')
ax2=plt.subplot(2,1,2); ax2.grid(); ax2.set_ylabel('X [g/L]'); ax2.set_xlabel('Time [h]')
lines=['-','--',':']; linecycler = cycle(lines)

for value in [1.0, 0.7, 1.3]:
    # - init(VX_0=value)
    parDict['VX_0'] = value

    # - simu(5)
    model = load_fmu(fmu_model)
    for key in parDict.keys(): model.set(parLocation[key], parDict[key])
    sim_res = model.simulate(0, 5, options=opts)

    linetype = next(linecycler)
    ax1.plot(sim_res['time'], sim_res['bioreactor.c[2]'], color='b', linestyle=linetype)
    ax2.plot(sim_res['time'], sim_res['bioreactor.c[1]'], color='r', linestyle=linetype)
```

Much to write at the command line
Flexibility comes at a price
Use diagram “canvas” to collect results

FMU-explore with PyFMI

In [4]:
```
newplot(plotType='Demo_1')
for value in [1,0.7,1.3]: init(VX_0=value); simu(5)
```



In [5]:
```
describe('bioreactor.V')
```

```
Reactor broth volume 1.0 [ L ]
```

FMU-explore “package”

- Command-interaction vs scripting

- Command-line interaction
  - Concise
  - Facilitate incremental variation of simulations
- Solution approach – workspace framework
  - Introduce simplifying functions: par(), init(), simu()…
  - Introduce dictionaries and lists to faciliate incremental interaction
- Same notebook – use PyFMI or FMPy under the hood

Handfull commands
- general code independent of application

- newplot()
- par(), init()
- simu()
- disp(), describe()

Cf *Simnon* (H. Elmqvist, 1975)

Benefits
- Less to enter
- Readability
- Incremental changes…

Implementation:
Workspace dictionaries and lists

Information to provide for the application

- parDict[] – short name for parameter and value
- parLocation[] – short name > modelica model name
- stateDict[] – states to handle simu(mode='cont')
- diagrams[] – tailored standard diagram for results
  (type list of "command line" strings)

- sim_res[] – all simulation results stored
  (type FMIResult - extract numpy.ndarray)

Global variables

Global variables?

Workspace benefit from “global variables”

- Less to type
- Facilitate study of incremental changes…
  - i.e. you WANT side effects

Little written about proper use of global variables?

parDict[] and parLocation[]

parDict – short names and values
parLocation – short names -> modelica code names

```
for key in parDict.keys(): model.set(parLocation[key], parDict[key])
```

Focus/restrict interaction to certain parameters
- disp() – displays only these
  pyfmi: `model.get_model_variables().keys()` – displays all

Good for
- Teaching situation – avoid unnecessary information
- Consultancy – a step towards protect IP… (needs to do more)

Handling of model state

simu(mode='cont') need to set initial values

- Continuous time state
- Discrete time sampled systems (i.e. regulators)

- Continuous time – pyfmi: model.get_states_list()
- Discrete time? – today configure manually

  Interested in an automatic solution!

  Possibly pyfmi: model.get_fmu_state() …

Application setup-file

The setup needed:

- Import needed packages including PyFMI/FMPy
- fmu_model – name of FMU
- parDict[] – short names -> values
- parLocation[] – short names -> modelica address
- timeDiscreteStates[] – used to make stateDict[]

- newplot() – used to make different lists diagrams[]
- describe() – extend with information not in code

FMU re-use different simulation tasks

Stimulate “Github creativity” on Jupyter notebook level?

Same FMU

- Explore effect of parameter changes
- Sensitivity analysis
- **Calibration**
- Optimization
- …

Example Batch – now for calibration

[10]:
```
# Optimization routine import
import scipy.optimize
```

[11]:
```
# Parameters to be estimated using parDict names and their bounds
extra_args = (parEstim, data, fmu_model, simulationTime, parDict, parLocation)
```

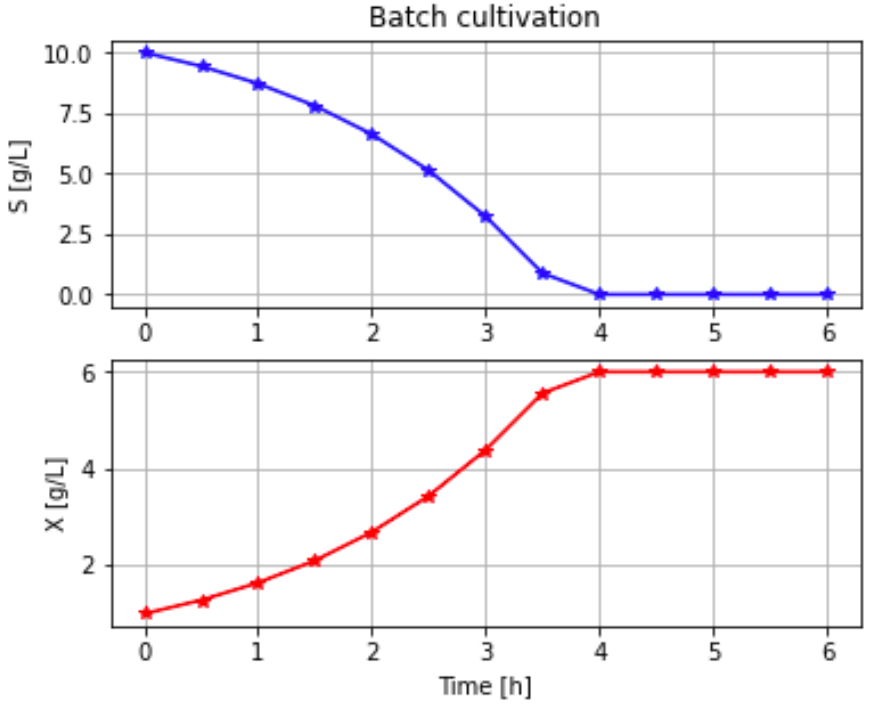
[12]:
```
# Modified evaluation function tailored for Python optimization algorithms
def objective(x, parEstim, data=data, fmu_model=fmu_model, simulationTime=simulationTime,
              parDict=parDict, parLocation=parLocation):
    """The parameter list is tailored for scipy optimization algorithms interface,
       where the first parameter x is an array with parameters that are tuned
       and evalauted and parEstim is a list of the names of these parameters.
       The code can be made faster, but longer, using fmpy-commands directly."""

    # Update parameters and simulate
    for i, p in enumerate(parEstim): par(**{p:x[i]})
    simu(simulationTime, options=opts_data)

    # Calculate loss function V
    V={}
    V['X'] = np.linalg.norm(data['X'] - np.interp(data['time'], sim_res['time'], sim_res['bioreactor.c[1]']))
    V['S'] = np.linalg.norm(data['S'] - np.interp(data['time'], sim_res['time'], sim_res['bioreactor.c[2]']))

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

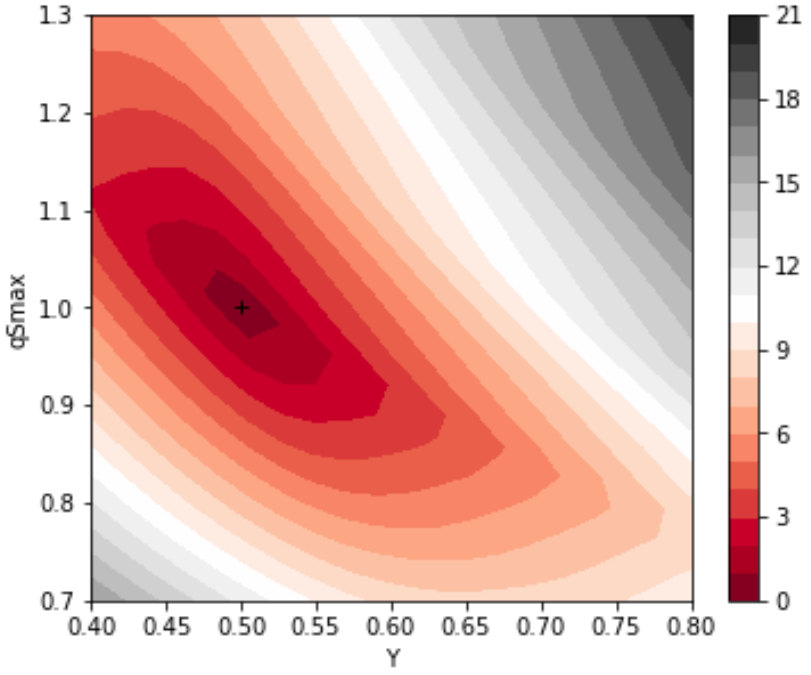
Example Batch – now for calibration

```
[13]: import time

[14]: # Run minimize()
start_time = time.time()
result = scipy.optimize.minimize(objective, x0=parEstim_0, args=extra_args,
                                 method='Nelder-Mead', bounds=parBounds, options={"disp":True})
print('CPU-time =', time.time()-start_time)

Optimization terminated successfully.
         Current function value: 0.045546
         Iterations: 40
         Function evaluations: 77
CPU-time = 26.788382530212402

[15]: result

[15]:        message: Optimization terminated successfully.
             success: True
              status: 0
                 fun: 0.045545889241978756
                   x: [ 5.001e-01  1.007e+00  1.405e-01]
                 nit: 40
                nfev: 77
       final_simplex: (array([[ 5.001e-01,  1.007e+00,  1.405e-01],
                              [ 5.001e-01,  1.007e+00,  1.405e-01],
                              [ 5.001e-01,  1.007e+00,  1.405e-01],
                              [ 5.001e-01,  1.007e+00,  1.405e-01]]), array([ 4.555e-02,  4.555e-02,  4.555e-02,  4.559e-02]))
```

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

Results from calibration





Example Batch – now for calibration

- Convenient to integrate ”workspace items” in objective function
  - parDict, parLocation,…
  - simu()
- Improves readaiblity compared to “naked” PyFMI och FMPy
- Price: negligible over-head cost in simulation time

- Still, the full flexibility of scripting using PyFMI/FMPy is there

Type of interaction with users

1. Smaller changes of culture model etc – free
2. Smaller changes of ways to interact – ex change diagram pens..
3. Larger expansions of culture model – consultancy
4. Change IEC-process parametrization to “users process-view”
    1. Names
    2. Factorisation of parameters
    3. Simulate not in “time” but “accumulated flow”

Occasionally extended BPL and FMU-explore from user interaction

Concluding remarks

- Colab works pretty well
- Teaching situation
  - Local installation in the classroom perhaps
  - Homework for students over intenet using Colab
- DEMO-situation
  - Discuss around notebooks, easy to update
- FMU-explore
  - Command-line interaction facilitated
  - Readability improved
  - Same notebook for PyFMI and FMPy
  - FMI 3.0 implications?
  - Larger interest?

Links

Homepage: https://github.com/janpeter19

Ex 1: https://github.com/janpeter19/BPL_TEST2_Batch

Ex 2: https://github.com/janpeter19/BPL_TEST2_Batch_calibration

More on Colab: https://colab.research.google.com/