BPL_TEST2_Batch_design_space script with PyFMI

The key library PyFMI is installed.

After the installation a small application BPL_TEST2_Batch_design_space 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-py310_23.1.0-1-Linux-x86_64.
!chmod +x Miniconda3-py310_23.1.0-1-Linux-x86_64.sh
!bash ./Miniconda3-py310_23.1.0-1-Linux-x86_64.sh -b -f -p /usr/local
import sys
sys.path.append('/usr/local/lib/python3.10/site-packages/')
```

```
--2024-10-03 19:11:56--  https://repo.anaconda.com/miniconda/Miniconda3-py310
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.32.241, 104.16.191.
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.32.241|:443... con
HTTP request sent, awaiting response... 200 OK
Length: 74403966 (71M) [application/x-sh]
Saving to: ‘Miniconda3-py310_23.1.0-1-Linux-x86_64.sh’

Miniconda3-py310_23 100%[===================>]  70.96M   159MB/s    in 0.4s

2024-10-03 19:11:57 (159 MB/s) - ‘Miniconda3-py310_23.1.0-1-Linux-x86_64.sh’

PREFIX=/usr/local
Unpacking payload ...

Installing base environment...


Downloading and Extracting Packages


Downloading and Extracting Packages

Preparing transaction: done
Executing transaction: done
installation finished.
```

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

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

```
!conda --version
!python --version
```

```
conda 23.1.0
Python 3.10.15
```

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

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

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

- FMU - BPL_TEST2_Batch_design_space_no_noise_linux_om_me.fmu
- Setup-file - BPL_TEST2_Batch_design_space_no_noise_explore.py
- FMU - BPL_TEST2_Batch_design_space_with_noise_linux_om_me.fmu
- Setup-file - BPL_TEST2_Batch_design_space_with_noise_explore.py

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

```
%%bash
git clone https://github.com/janpeter19/BPL_TEST2_Batch_design_space
```

```
Cloning into 'BPL_TEST2_Batch_design_space'...
```

```
%cd BPL_TEST2_Batch_design_space
```

```
/content/BPL_TEST2_Batch_design_space
```

BPL_TEST2_Batch_design_space - demo

In this notebook the design space for a batch cultivation process is determined and visualized. The example is kept as simple as possible. The culture grow on a substrate S and the cell conentration X inrease until the substrate is consumed. We study the problem first without any measurement noise and then later with measurement noise and use one separate FMU for each.

The end criteria for a batch is here when the subdstrate level has decreased below a certain predefined level and that time is called time_final:

- S < Smin

The evaluation of the batch culture is just in terms of the obtained value of cell concentration at the end in combination with how long time the culture took. The batch is accepted provided the culture fullfil the two requirements:

- X_final > X_final_min
- Time_final < time_final_max

The question is what range of process parameters Y and qSmax that can be allowed to still get accepted batches.

Here we simply use brute force and sweep through a number combinations of process parameters and evaluate by simulation the result for each parameter setting. We get rather clear-cut corners in the process parameter space that result in acceptable batches.

In the later part we introduce substrate measurement error and in this way introduce some uncertainty in the determination of end of batch. The impact of this measurement noise is that the design space get more rounded corners.

The practical experimental approach is usually to just use a few parameter combinations and evaluate these and from that information calculate the design space. Usually "process linearity" assumption is used. The combination of this experimental approach with brute force simulation is discussed in reference [1].

1 Batch end detection - no measurement noise

Here we load a system model without noise. Thus detection of end of batch is an event in continuous time.

```
run -i BPL_TEST2_Batch_no_noise_explore.py
```

```
Linux - run FMU pre-compiled OpenModelica 1.23.0-dev

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 simulati
 - describe()  - describe culture, broth, parameters, variables with values/u

Note that both disp() and describe() takes values from the last simulation
and the command process_diagram() brings up the main configuration

Brief information about a command by help(), eg help(simu)
Key system information is listed with the command system_info()
```

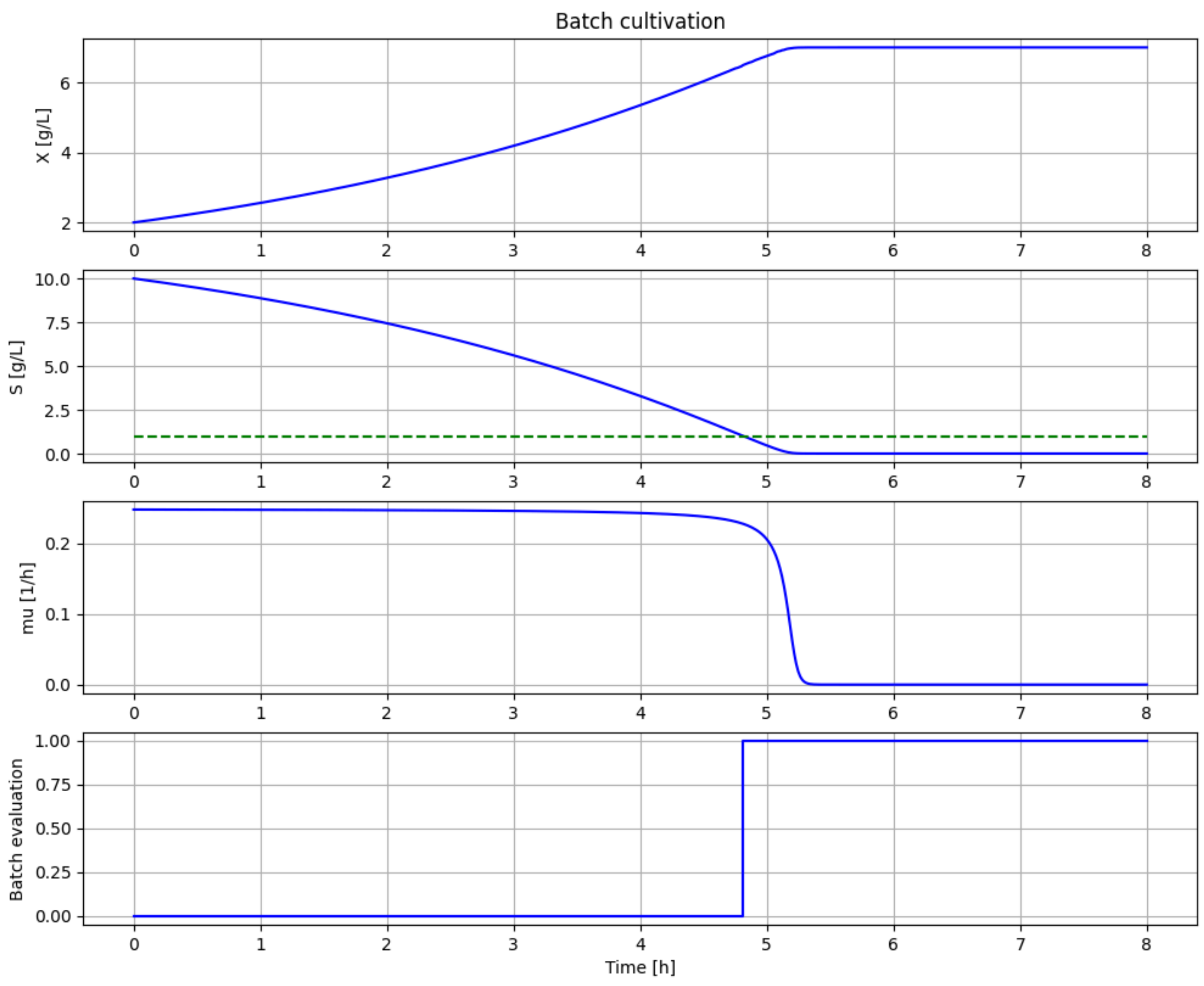
```
# Adjust the diagram size
%matplotlib inline
plt.rcParams['figure.figsize'] = [30/2.54, 24/2.54]
```

1.1 Batch evaluation

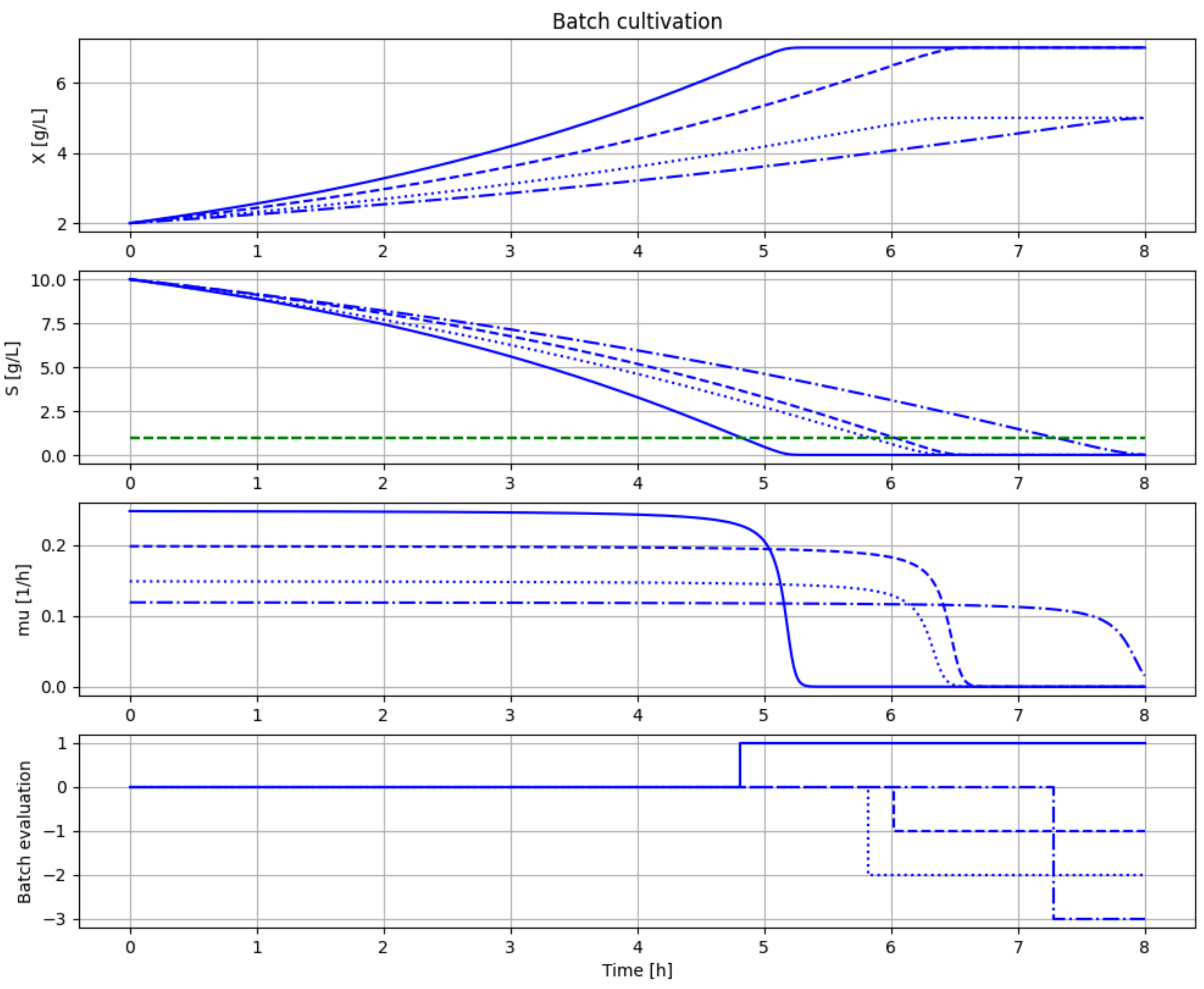
The first an example of batch that has an end of batch that fulfills the criteria for acceptance. In the following diagram we see examples of impact of variation on the criteria for acceptance.

The variable batch_evaluation goes from 0 to either 1 or a negative value when end of batch is detected. A positive value 1 means that the acceptance criteria is fullfilled and a negative value -1, -2 or -3 is obtained if one or more criteria for acceptance is not fullfilled.

```
# Nominal parameters
par(S_min=1.0, time_final_max=6.0, X_final_min=5.0)
init(VX_start=2, VS_start=10)
par(Y=0.5, qSmax=0.5, Ks=0.1)


# Simulation of nominal parameters that gives a batch that meed the end criteria
newplot(plotType='TimeSeries_2')
simu(8)
```



```
# Exammple of process parameter changes and how they meet the end criteria
newplot(plotType='TimeSeries_2')
par(Y=0.50, qSmax=0.50); simu(8)  # - pass (solid line)
par(Y=0.50, qSmax=0.40); simu(8)  # - fail criteria time_final < 6.0 (dashed line
par(Y=0.30, qSmax=0.50); simu(8)  # - fail criteria X_final > 5.0 (dotted line)
par(Y=0.30, qSmax=0.40); simu(8)  # - fail both criteria (dash dotted line)
```

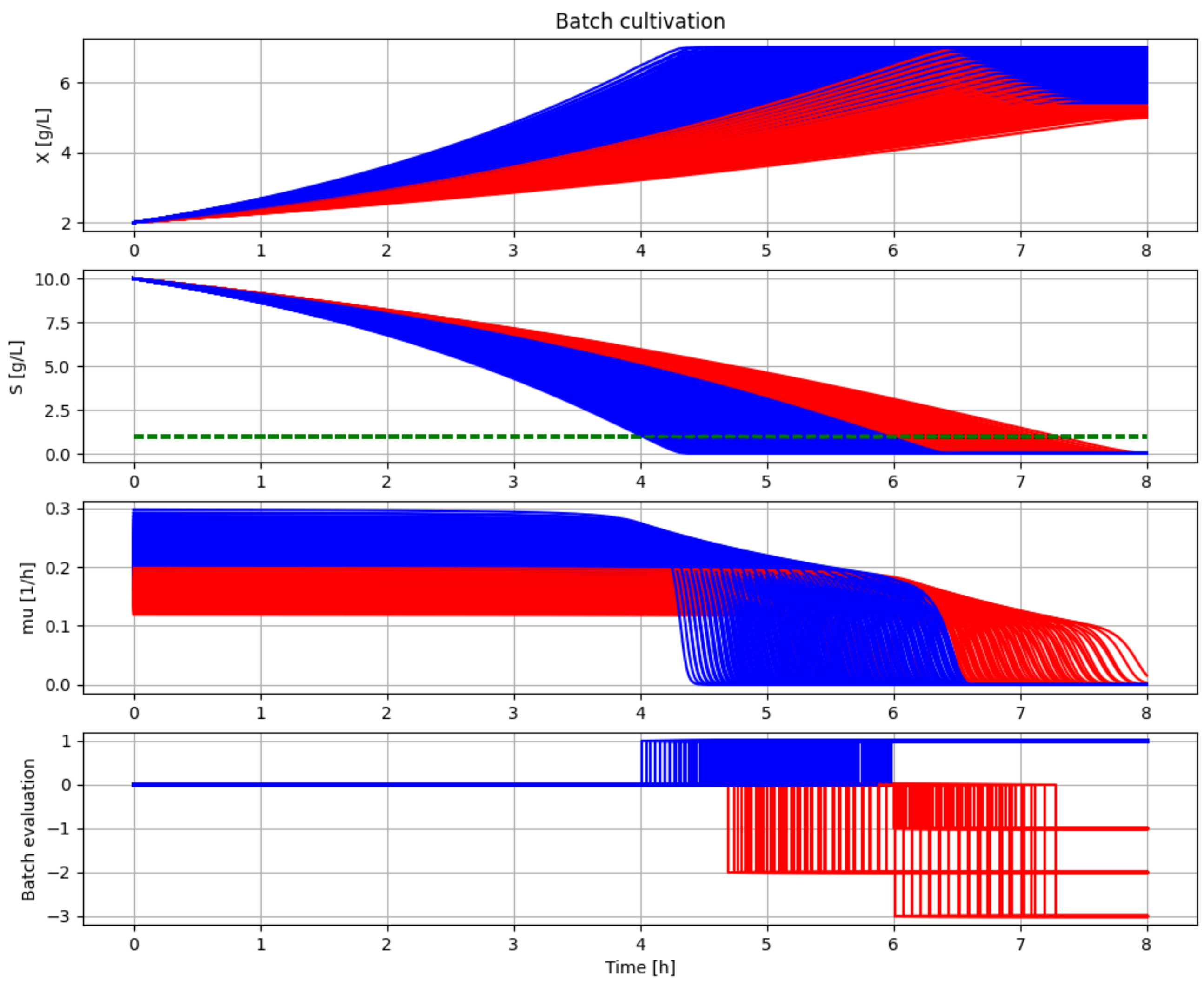


We see that the accepted batch (solid line) finish first. The batches that fail take longer time and two of them has also lower cell concentration at the end.

1.2 Batch evaluation under process variation - parameter sweep

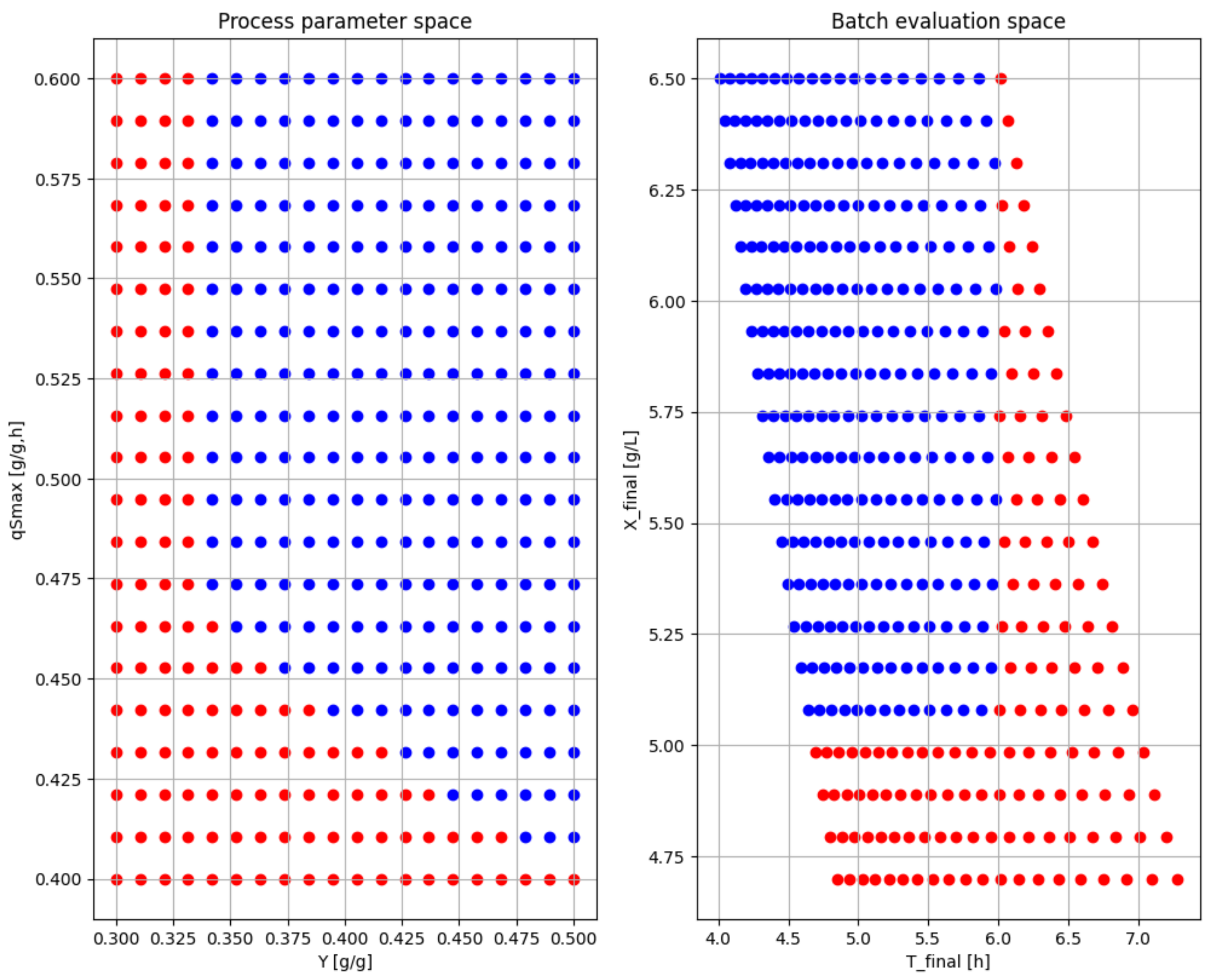
Now let us systematically sweep through a number of combinations of process parameters Y and qSmax and evaluate the batches and visualise the result.

```
# Define sweep ranges and storage of final data
nY = 20
nqSmax = 20
Y_range = np.linspace(0.3,0.5,nY)
qSmax_range = np.linspace(0.4,0.6,nqSmax)
data = np.zeros([nY,nqSmax,5])


# Run parameter sweep - takes a few minutes
newplot(plotType='TimeSeries_2_diagrams')
init(VX_start=2, VS_start=10)

for j in range(nY):
    for k in range(nqSmax):
        par(Y=Y_range[j])
        par(qSmax=qSmax_range[k])
        simu(8)

        # Store final results
        data[j,k,0] = Y_range[j]
        data[j,k,1] = qSmax_range[k]
        data[j,k,2] = sim_res['monitor.time_final'][-1]
        data[j,k,3] = sim_res['monitor.X_final'][-1]
        data[j,k,4] = sim_res['monitor.batch_evaluation'][-1]

        # Plot simulation results
        if sim_res['monitor.batch_evaluation'][-1] > 0:
            ax1.plot(sim_res['time'], sim_res['bioreactor.c[1]'],'b-')
            ax2.plot(sim_res['time'], sim_res['bioreactor.c[2]'],'b-')
            ax2.plot([0, simulationTime], [model.get('monitor.S_min'), model.get(
            ax3.plot(sim_res['time'], sim_res['bioreactor.culture.q[1]'],'b-')
            ax4.step(sim_res['time'],sim_res['monitor.batch_evaluation'],where='p
        else:
            ax1.plot(sim_res['time'], sim_res['bioreactor.c[1]'],'r-')
            ax2.plot(sim_res['time'], sim_res['bioreactor.c[2]'],'r-')
            ax2.plot([0, simulationTime], [model.get('monitor.S_min'), model.get(
            ax3.plot(sim_res['time'], sim_res['bioreactor.culture.q[1]'],'r-')
            ax4.step(sim_res['time'],sim_res['monitor.batch_evaluation'],where='p

plt.show()
```



Batches represented by blue lines are those that in the end got accepted. The red ones failed.

```
# Show end results
plt.figure()
ax1 = plt.subplot(1,2,1)
ax2 = plt.subplot(1,2,2)

for j in range(nY):
    for k in range(nqSmax):
        if data[j,k,4] > 0:
            ax1.scatter(data[j,k,0],data[j,k,1],c='b')
        else:
            ax1.scatter(data[j,k,0],data[j,k,1],c='r')
ax1.grid()
#plt.axis([0, 0.8, 0, 0.8])
ax1.set_ylabel('qSmax [g/g,h]')
ax1.set_xlabel('Y [g/g]')
ax1.set_title('Process parameter space')

for j in range(nY):
    for k in range(nqSmax):
        if data[j,k,4] > 0:
            ax2.scatter(data[j,k,2],data[j,k,3],c='b')
        else:
            ax2.scatter(data[j,k,2],data[j,k,3],c='r')
ax2.grid()
#plt.axis([0, 8, 0, 8])
ax2.set_xlabel('T_final [h]')
ax2.set_ylabel('X_final [g/L]')
ax2.set_title('Batch evaluation space')
plt.show()
```



Here we visualize the previous simulations results in a different way with foucse on the end result. Each dot in the left diagram (process parameter space) represent a simulation that give a result in the rigiht diagam (batch evaluation space). The blue dots are those batches that were accepted and the red ones those that failed.

The blue dots in the process parameter space show the "design space" for the acceptance criteria we have.

2 Batch end detection - with measurement noise

Here we load a system model with normal noise added to the sampled value of substrate concentration. The measurement of substrate conentration usually has a higher variation than measuremetn of cell concentration and therefore we focus here on the impact on substrate conentrations.

Thus detection of end of batch is now in discrete time with a give samplePeriod (default 0.1 hour). This discreteization also introduce an error in detection of the end point. By changing this sample intervall to shorter values you can see the impact of this error but not done here.

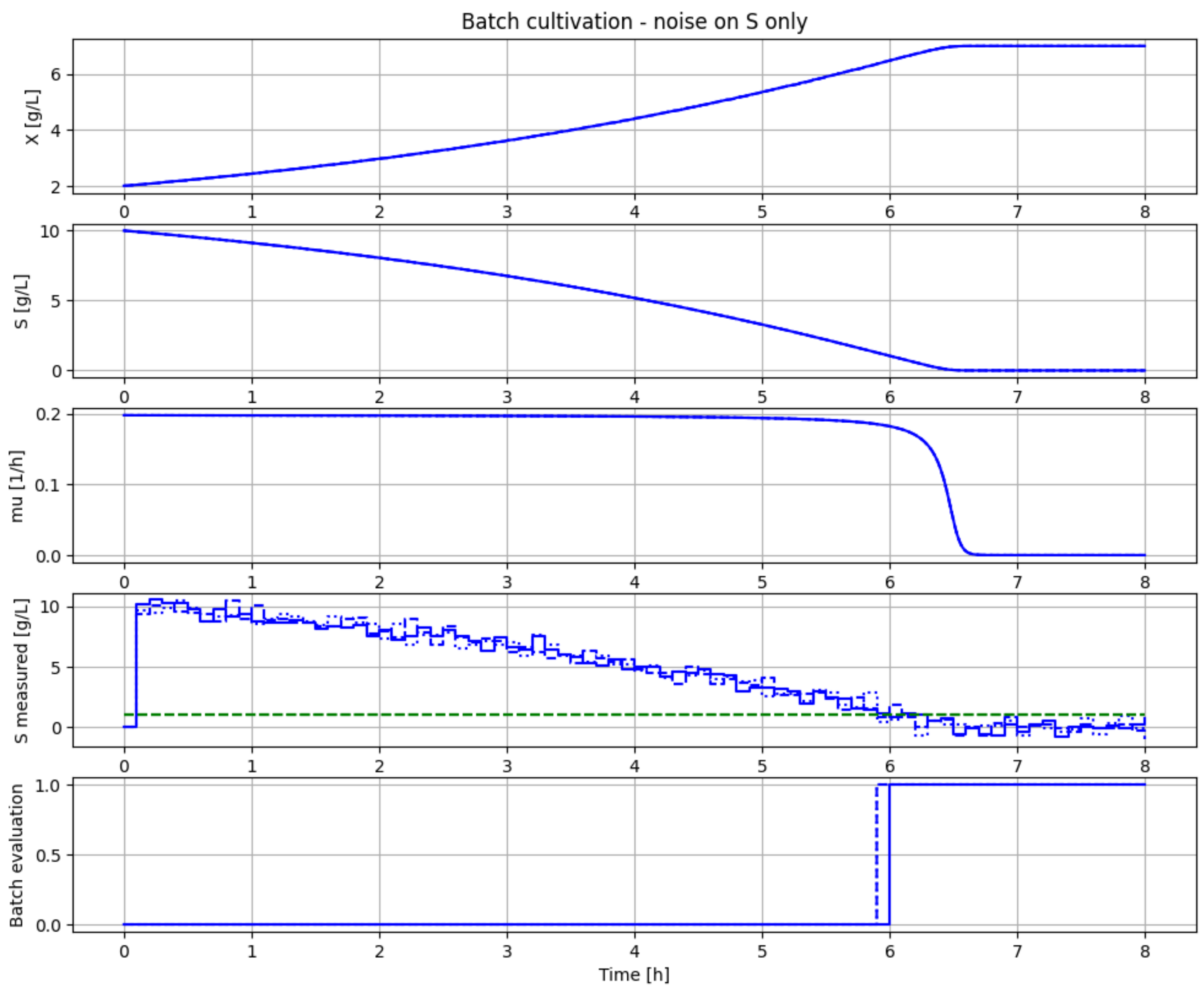
```
run -i BPL_TEST2_Batch_with_noise_explore.py
```

```
Linux - run FMU pre-compiled OpenModelica 1.23.0-dev

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 simulati
 - describe()  - describe culture, broth, parameters, variables with values/u

Note that both disp() and describe() takes values from the last simulation
and the command process_diagram() brings up the main configuration

Brief information about a command by help(), eg help(simu)
Key system information is listed with the command system_info()
<Figure size 1181.1x944.882 with 0 Axes>
```

2.1 Batch evaluation under substrate measurement error

Here we see an example of how substrate measurement noise directly affect the evaluation of the batch from accetable to not acceptable.

```
# Nominal parameters
par(S_min=1.0, time_final_max=6.0, X_final_min=5.0)
init(VX_start=2, VS_start=10)
par(Y=0.5, qSmax=0.5, Ks=0.1)
par(sigma=0.48, samplePeriod=0.1)


# Simulation of nominal parameters that gives a batch that meed the end criteria
newplot(plotType='TimeSeries_2')
par(Y=0.5, qSmax=0.4);
for value in [2,3,5]: par(seed=value); simu(8)
```

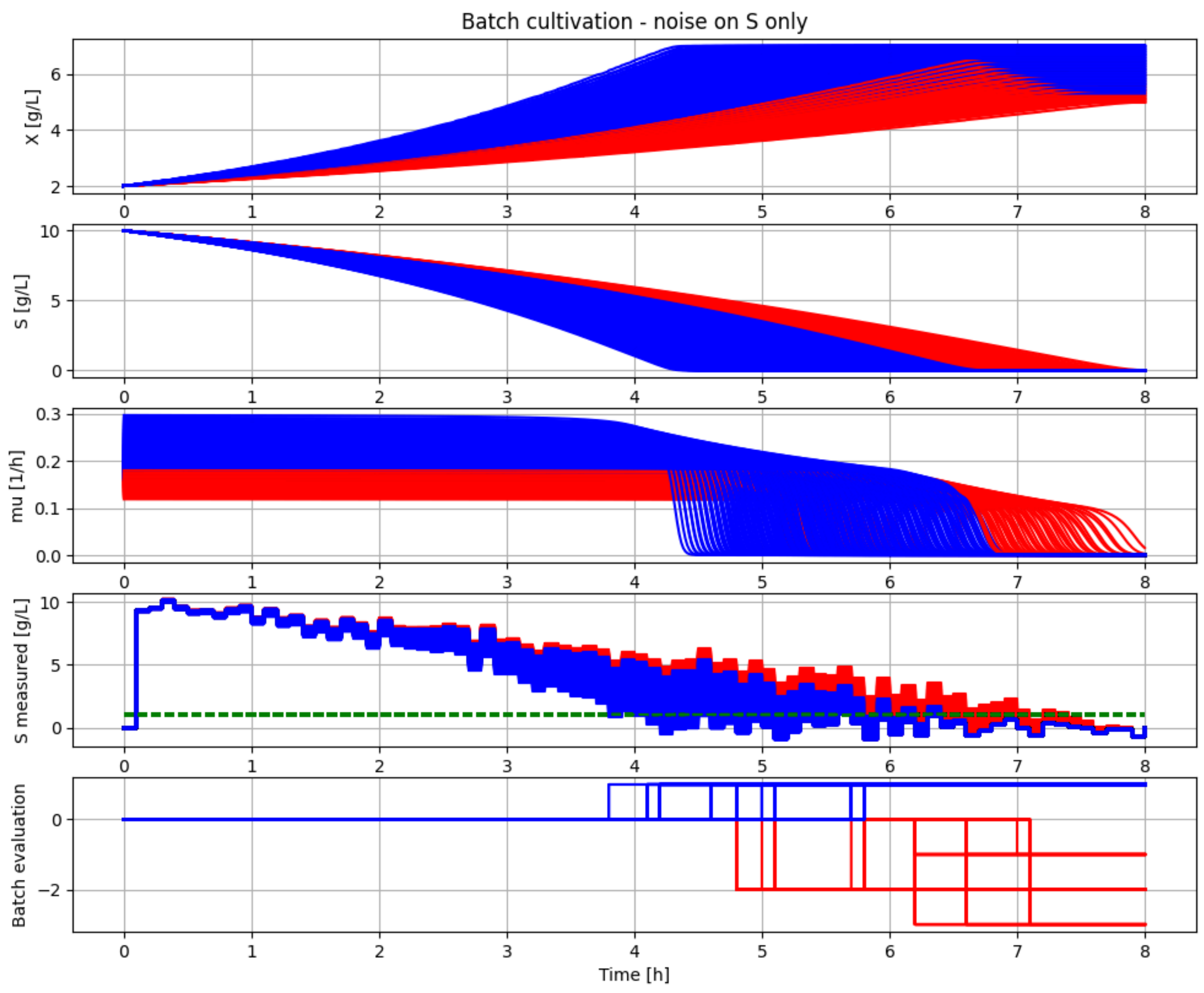


2.2 Batch evaluation under process variation and measurement error - parameter sweep

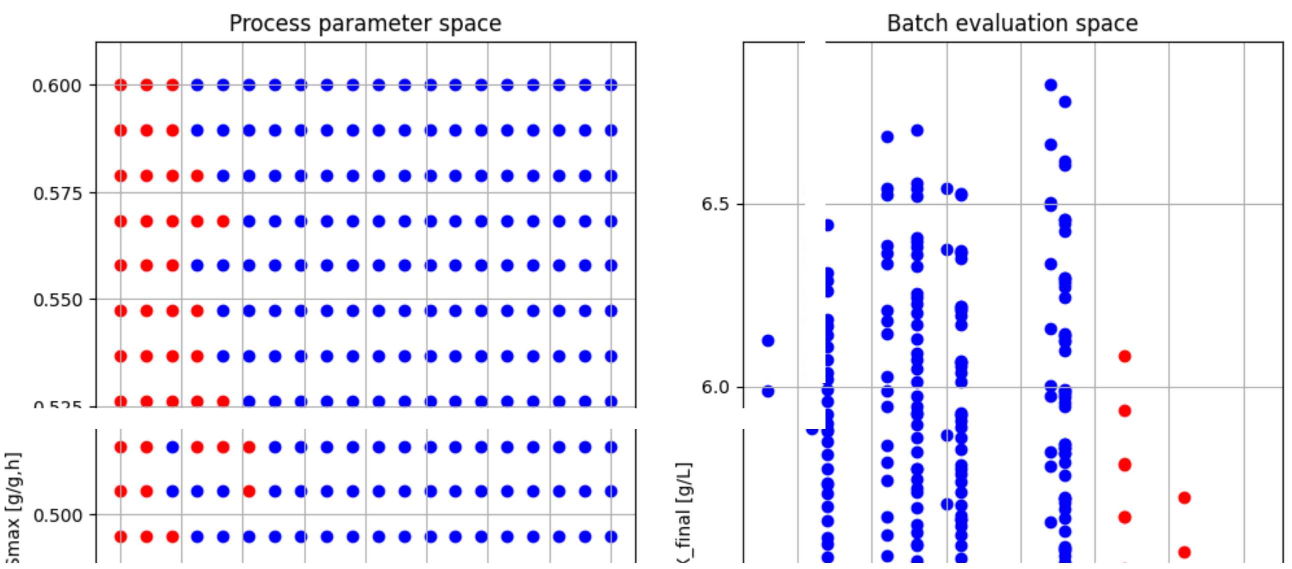
Now let us again systematically sweep through a number of combinations of process parameters Y and qSmax and evaluate the batches and visualise their result.

```
# Define sweep ranges and storage of final data
nY = 20
```

```
nqSmax = 20
Y_range = np.linspace(0.3,0.5,nY)
qSmax_range = np.linspace(0.4,0.6,nqSmax)
data = np.zeros([nY,nqSmax,5])


# Run parameter sweep - takes a few minuts
newplot(plotType='TimeSeries_2_diagrams')
par(sigma=0.48, seed=1, samplePeriod=0.1)

for j in range(nY):
    for k in range(nqSmax):
        par(Y=Y_range[j])
        par(qSmax=qSmax_range[k])
        simu(8)

        # Store final results
        data[j,k,0] = Y_range[j]
        data[j,k,1] = qSmax_range[k]
        data[j,k,2] = sim_res['monitor.time_final'][-1]
        data[j,k,3] = sim_res['monitor.X_final'][-1]
        data[j,k,4] = sim_res['monitor.batch_evaluation'][-1]

        # Plot simulation results
        if sim_res['monitor.batch_evaluation'][-1] > 0:
            ax1.plot(sim_res['time'], sim_res['bioreactor.c[1]'],'b-')
            ax2.plot(sim_res['time'], sim_res['bioreactor.c[2]'],'b-')
            ax3.plot(sim_res['time'], sim_res['bioreactor.culture.q[1]'],'b-')
            ax4.plot(sim_res['time'], sim_res['sensor.out.c[2]'],'b-')
            ax4.plot([0, simulationTime], [model.get('monitor.S_min'), model.get(
            ax5.step(sim_res['time'],sim_res['monitor.batch_evaluation'],where='p
        else:
            ax1.plot(sim_res['time'], sim_res['bioreactor.c[1]'],'r-')
            ax2.plot(sim_res['time'], sim_res['bioreactor.c[2]'],'r-')
            ax3.plot(sim_res['time'], sim_res['bioreactor.culture.q[1]'],'r-')
            ax4.plot(sim_res['time'], sim_res['sensor.out.c[2]'],'r-')
            ax4.plot([0, simulationTime], [model.get('monitor.S_min'), model.get(
            ax5.step(sim_res['time'],sim_res['monitor.batch_evaluation'],where='p

plt.show()
```



```
# Show end results
plt.figure()
ax1 = plt.subplot(1,2,1)
ax2 = plt.subplot(1,2,2)

for j in range(nY):
    for k in range(nqSmax):
        if data[j,k,4] > 0:
            ax1.scatter(data[j,k,0],data[j,k,1],c='b'
        else:
            ax1.scatter(data[j,k,0],data[j,k,1],c='r'
ax1.grid()
#plt.axis([0, 0.8, 0, 0.8])
ax1.set_ylabel('qSmax [g/g,h]')
ax1.set_xlabel('Y [g/g]')
ax1.set_title('Process parameter space')

for j in range(nY):
    for k in range(nqSmax):
        if data[j,k,4] > 0:
            ax2.scatter(data[j,k,2],data[j,k,3],c='b'
        else:
            ax2.scatter(data[j,k,2],data[j,k,3],c='r'
ax2.grid()
#plt.axis([0, 8, 0, 8])
ax2.set_xlabel('T_final [h]')
ax2.set_ylabel('X_final [g/L]')
ax2.set_title('Batch evaluation space')
```


Process parameter space
qSmax [g/g,h]
Batch evaluation space
X_final [g/L]