

## ✓ BPL\_YEAST\_AIR\_Fedbatch script with FMPy

The key library FMPy is installed.

After the installation a small application BPL\_YEAST\_AIR\_Fedbatch 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.
!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/')
```

```
⇒ --2024-08-13 09:02:43-- https://repo.anaconda.com/miniconda/Miniconda3-py312
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.191.158, 104.16.32.1
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.191.158|:443... co
HTTP request sent, awaiting response... 200 OK
Length: 143351488 (137M) [application/octet-stream]
Saving to: 'Miniconda3-py312_24.3.0-0-Linux-x86_64.sh'
```

```
Miniconda3-py312_24 100%[=====>] 136.71M 99.1MB/s in 1.4s
```

```
2024-08-13 09:02:45 (99.1 MB/s) - 'Miniconda3-py312_24.3.0-0-Linux-x86_64.sh'
```

```
PREFIX=/usr/local
Unpacking payload ...
```

```
Installing base environment...
```

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

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

```
⇒
```

```
certifi
conda
openssl
```

```
2024.7.4-py312h06a4308_0 --> 2024.7.4-py312h
24.3.0-py312h06a4308_0 --> 24.7.1-py312h
3.0.13-h7f8727e_0 --> 3.0.14-h5eee1f
```

## Downloading and Extracting Packages:

openssl-3.0.14	5.2 MB	:	0% 0/1 [00:00<?, ?it/s]
conda-24.7.1	1.2 MB	:	0% 0/1 [00:00<?, ?it/s]
certifi-2024.7.4	159 KB	:	0% 0/1 [00:00<?, ?it/s]
ca-certificates-2024	127 KB	:	0% 0/1 [00:00<?, ?it/s]
openssl-3.0.14	5.2 MB	:	0% 0.003006342237126712/1 [00:00<01:51
ca-certificates-2024	127 KB	:	13% 0.12647440251960723/1 [00:00<00:02,
frozendict-2.4.2	36 KB	:	44% 0.43853215920344746/1 [00:00<00:00,
certifi-2024.7.4	159 KB	:	10% 0.10045740493212503/1 [00:00<00:03,
conda-24.7.1	1.2 MB	:	1% 0.013060714305643354/1 [00:00<00:28
openssl-3.0.14	5.2 MB	:	50% 0.49604646912590744/1 [00:00<00:00,
frozendict-2.4.2	36 KB	:	100% 1.0/1 [00:00<00:00, 1.25it/s]
certifi-2024.7.4	159 KB	:	100% 1.0/1 [00:00<00:00, 2.62it/s]
certifi-2024.7.4	159 KB	:	100% 1.0/1 [00:00<00:00, 2.62it/s]
conda-24.7.1	1.2 MB	:	100% 1.0/1 [00:00<00:00, 1.31it/s]

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

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

```
🔄 conda 24.7.1
Python 3.12.2
```

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



```
executing transaction: done
```

```
#!conda install matplotlib --yes
```

```
#!conda install scipy --yes
```

```
#!conda install xlrd --yes
```

```
#!conda install openpyxl --yes
```

## ✓ Notes of BPL\_YEAST\_AIR\_Fedbatch

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

- FMU - BPL\_YEAST\_AIR\_Fedbatch\_linux\_jm\_cs.fmu
- Setup-file - BPL\_YEAST\_AIR\_Fedbatch\_fmpy\_explore

```
%%bash
```

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

```
📂 Cloning into 'BPL_YEAST_AIR_Fedbatch'...
```

```
%cd BPL_YEAST_AIR_Fedbatch
```

```
📂 /content/BPL_YEAST_AIR_Fedbatch
```

## ✓ BPL\_YEAST\_AIR\_Fedbatch - demo

Author: Jan Peter Axelsson

This notebook demonstrate yeast fedbatch cultivation. We look at impact of changes in the glucose feeding. We also take a look at tuning of the DO-control system. Both liquid- and gasphase are included in the model. The culture growth and metabolism are formulated in relation to to the respiratory capacity [1] and the model is exapanded to describe also the gas phase as well as the culture heat production [2]. The model was derived mainly from continuous culture data but proved to capture dynamic aspects well of ethanol production and consumption [3].

Interaction with the compiled model as FMU is mainly through the simplified commands: `par()`, `init()`, `newplot()`, `simu()` etc. The last simulation is always available in the workspace and called 'sim\_res'. The command `describe()` brings mainly up description infomration from the actual

Modelica code from the FMU but is complemented with information given in the dedicated Python setup-file.

The idea is to demonstrate how simulations and varying conditions can provide some process insight that can support the experimental work. I hope that at the end of this session you are ready to formulate your own questions you want to address with simulations - and you can just go on in this notebook! Just press the field "+Code" in the upper left part of notebook interface and you get a new "cell" where you write your own code. You can copy and paste from cells above using `ctrl-c` and `ctrl-p` as usual and edit the cell. When you are ready to execute the cell just press the "play button" to the left in the cell or press `shift-enter` as in "ordinary" Jupyter notebooks.

After a session you may want to save your own notebook. That you can do on your Google Drive account and I refer to Colab instructions for how to do this. It is easy.

Enjoy!

```
run -i BPL_YEAST_AIR_Fedbatch_D0control_fmpy_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 simulation
- `describe()` - describe culture, broth, parameters, variables with values/units

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()`

```
%matplotlib inline
plt.rcParams['figure.figsize'] = [36/2.54, 30/2.54]
```

## ✓ About the process model

We can get information about the process, liquid- and gas-phase by the command `describe()`.

This command can also be used to bring up information about a specific variable or parameter. However, you should use `describe()` after a simulation to get the values used during the simulation.

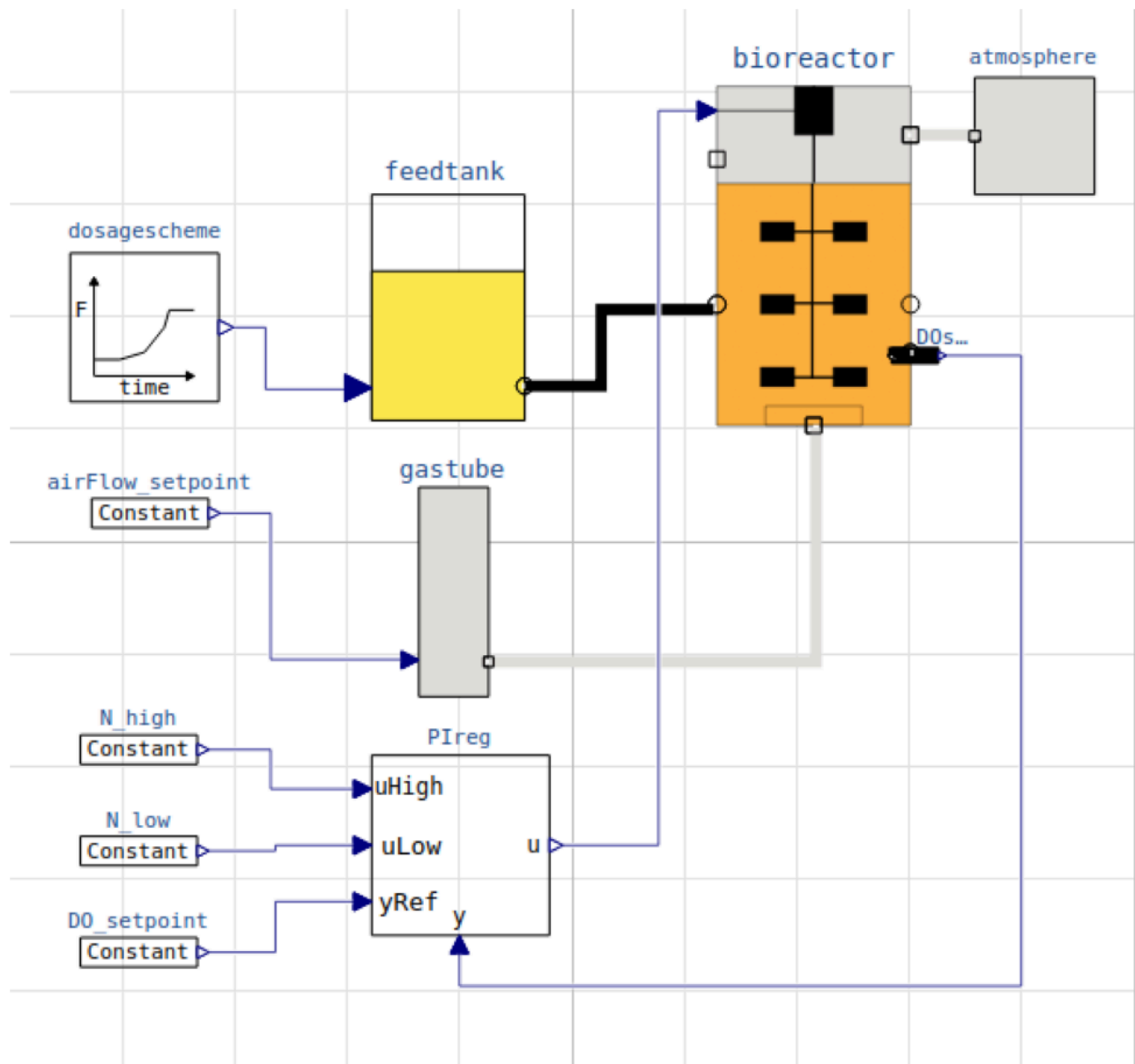
```
describe('culture'); print(); #describe('liquidphase'); print(); describe('gaspha
```



Saccharomyces cerevisiae – default parameters for strain H1022

```
process_diagram()
```

⇒ No processDiagram.png file in the FMU, but try the file on disk.



The model of the process has parameters both for culture, gas\_liquid\_transfer, as well as feeding procedure. The parameters that are available for changes you find by the command `disp()` and you

get a long list and you change by them by command `par()`. The model has even more parameters in the background but not made available for interaction.

## ✓ First simulations - adjusting start of substrate feeding

```
# Culture parameters and others at default values
par(q02lim=0.0069)

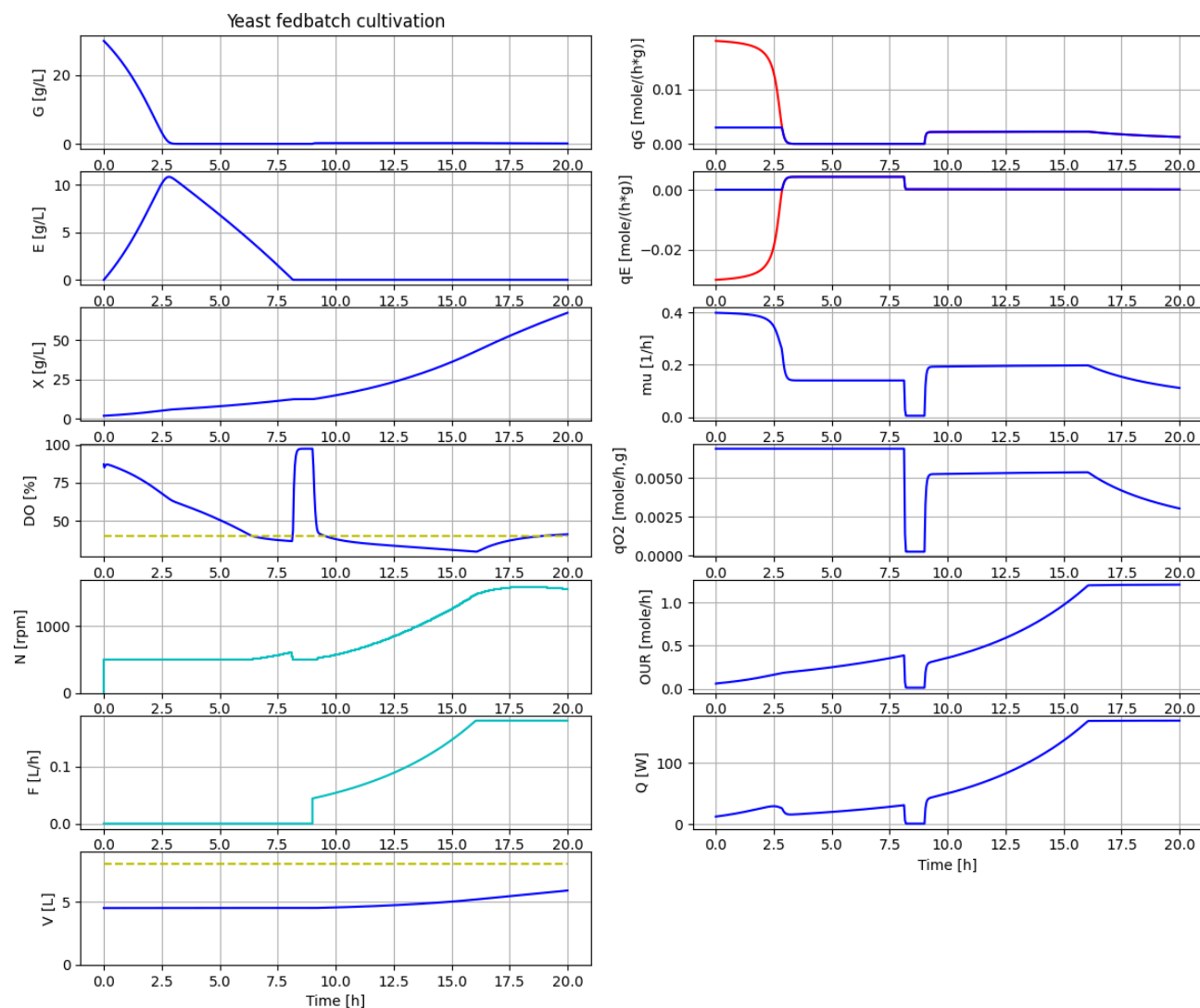
# Process initial conditions
init(V_start=4.5, VG_start=4.5*30, VX_start=4.5*2, VE_start=4.5*0)

# Feed profile
par(t_startExp=9, F_startExp=0.044, mu_feed=0.20, F_max=0.18)

# DO-control parameters
par(K=10, Ti=0.5)

# Simulate and plot
newplot(title='Yeast fedbatch cultivation', plotType='Overview')
simu(20)
```





Now we can get value of broth volume as well as the headspace and values are the last ones in the simulation

```
describe('bioreactor.V')
```

```
⇒ Reactor broth volume : 5.892 [ L ]
```

```
describe('bioreactor.V_gasphase')
```

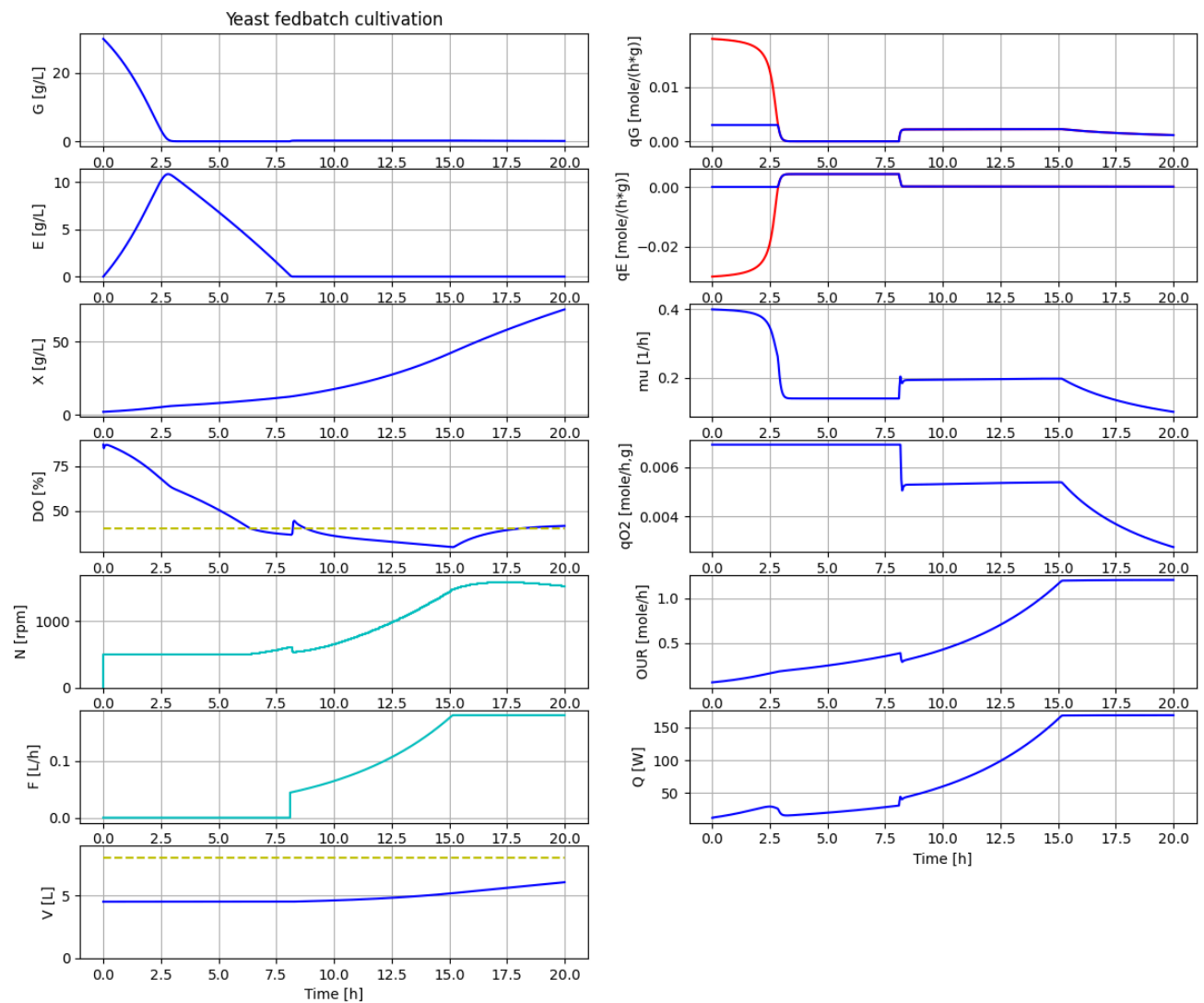
```
⇒ Volume of the gas phase : 2.108 [ L ]
```

```
# Take a look at the parameters available to adjust the dosage scheme  
disp('dosage', decimals=4)
```

```
⇒ F_start : 0.0  
   mu_feed : 0.2  
   t_startExp : 2.0  
   F_startExp : 0.12  
   F_max : 3.0
```

```
# Let us start the feeding just after the batch phase has ended and keep other pa  
par(t_startExp=8.1)
```

```
# Simulate and plot  
newplot(title='Yeast fedbatch cultivation', plotType='Overview')  
simu(20)
```



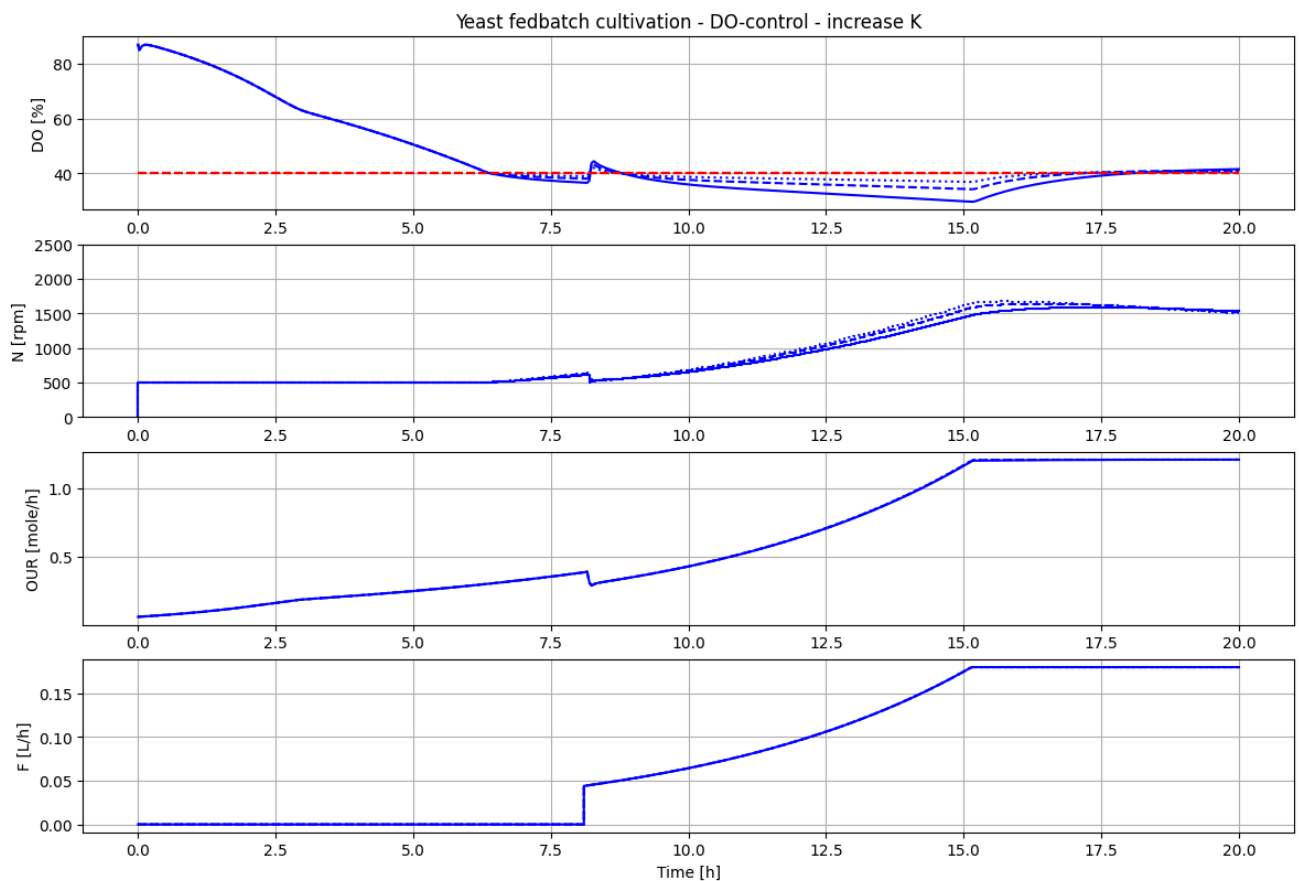
The increase of DO to about 50 % at end of batch phase should be possible to detect easily. This simulation is more realistic and we use these settings from now on.

## ✓ DO-control - tuning of PI-regulator parameters

Let us focus on the DO-control system and choose a more limited plotType. We study the impact of PI control parameters and see if we can decrease the control error without losing stability.

```
# Let us take a closer look at the DO-control system and try to make control error
newplot(title='Yeast fedbatch cultivation - DO-control - increase K', plotType='F'
for value in [10, 20, 40]: par(K=value); simu(20)
```

```
# Reset K to the original value
par(K=10)
```



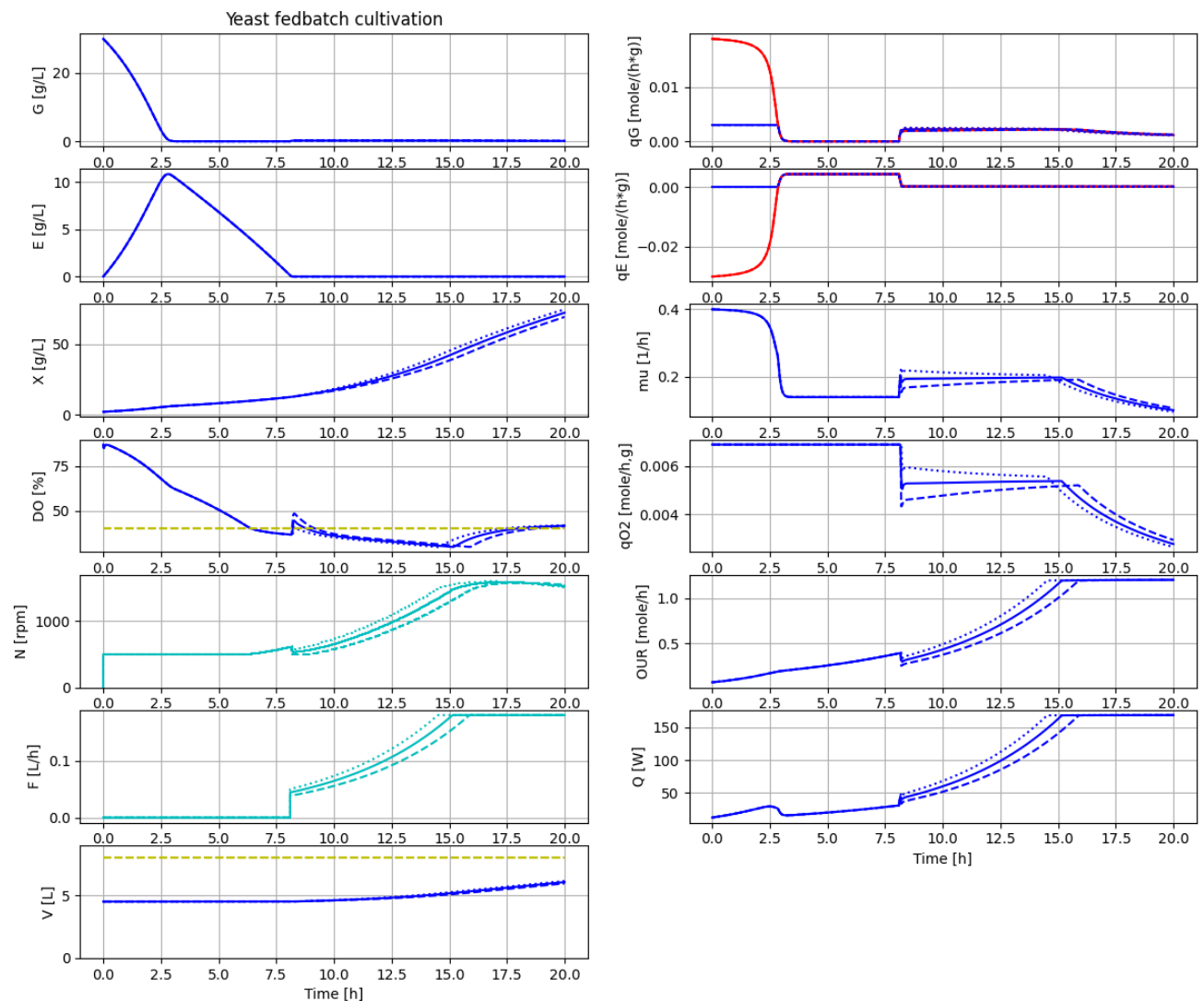
We see that by a higher control gain  $K$  the DO-control error get smaller and the stability of the control system is maintained.

**Exercise** I leave for you to study the impact variation of the  $T_i$ -parameter. Just make a new cell below. Then copy and paste the cell above and change parameter to  $T_i$ .

## ✓ Sensitivity to changes in feed-profile

Now, let us focus on investigating impact of changes in the feed-profile. The goal is to increase the produced cell mass without accumulation of by-product ethanol. Simulation can bring some insight into how behaviour of the different variables change when by-product is formed. This insight can help to interpret experimental results.

```
# Let us check the sensitivity to changes in the feed profile design
newplot(title='Yeast fedbatch cultivation', plotType='Overview')
for value in [0.044, 0.038, 0.050]: par(F_startExp=value); simu(20)
```



The variation in  $F_{\text{start}}$  has an impact and we see that the actual growth rate during fedbatch phase do converge to the set growth rate of the feed, but it takes more than 5 hours.

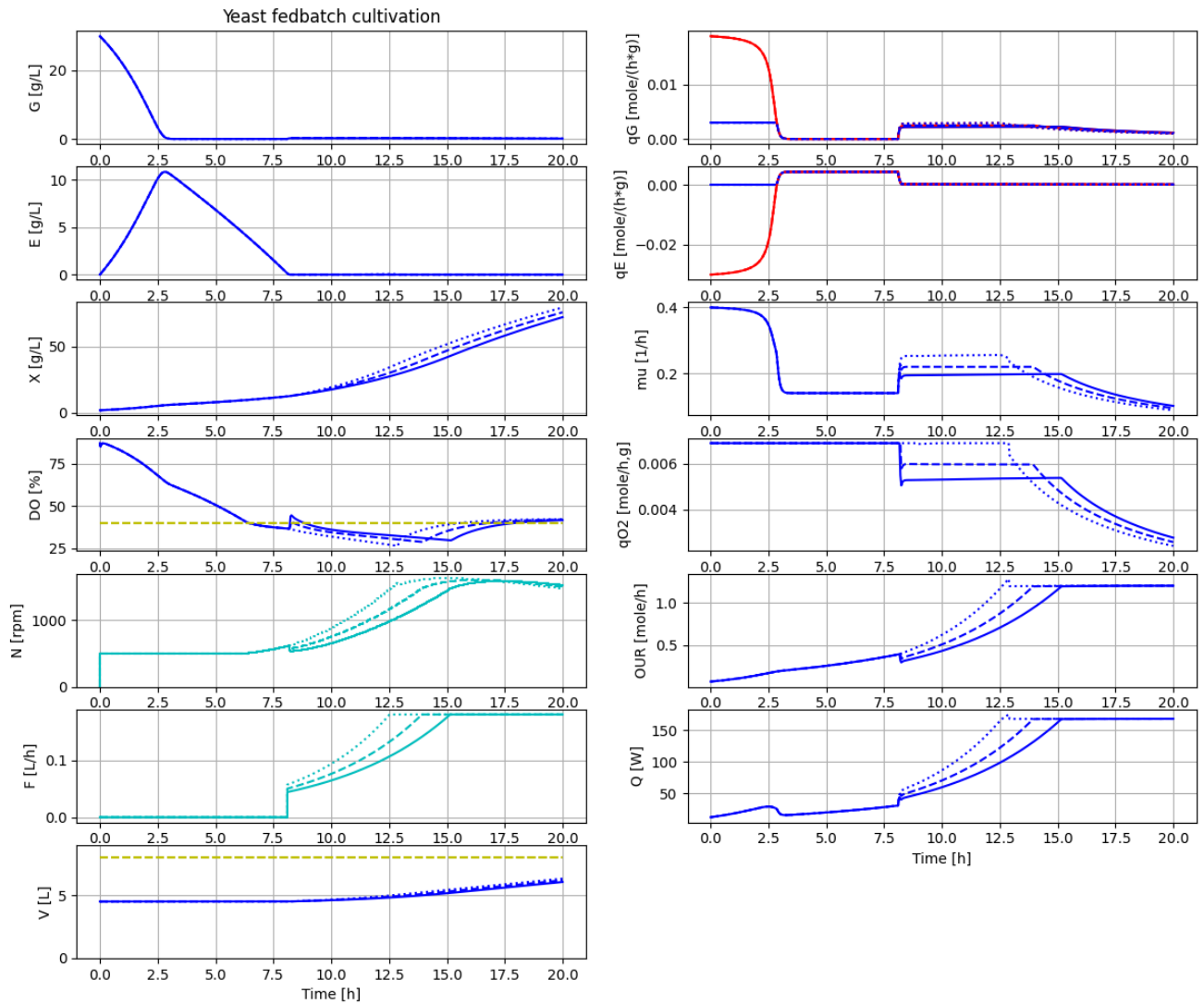
```
# Let us investigate a feedprofile that is closer to the maximal capacity
```

```
newplot(title='Yeast fedbatch cultivation', plotType='Overview')
```

```
par(F_startExp=0.044, mu_feed=0.20); simu(20)
```

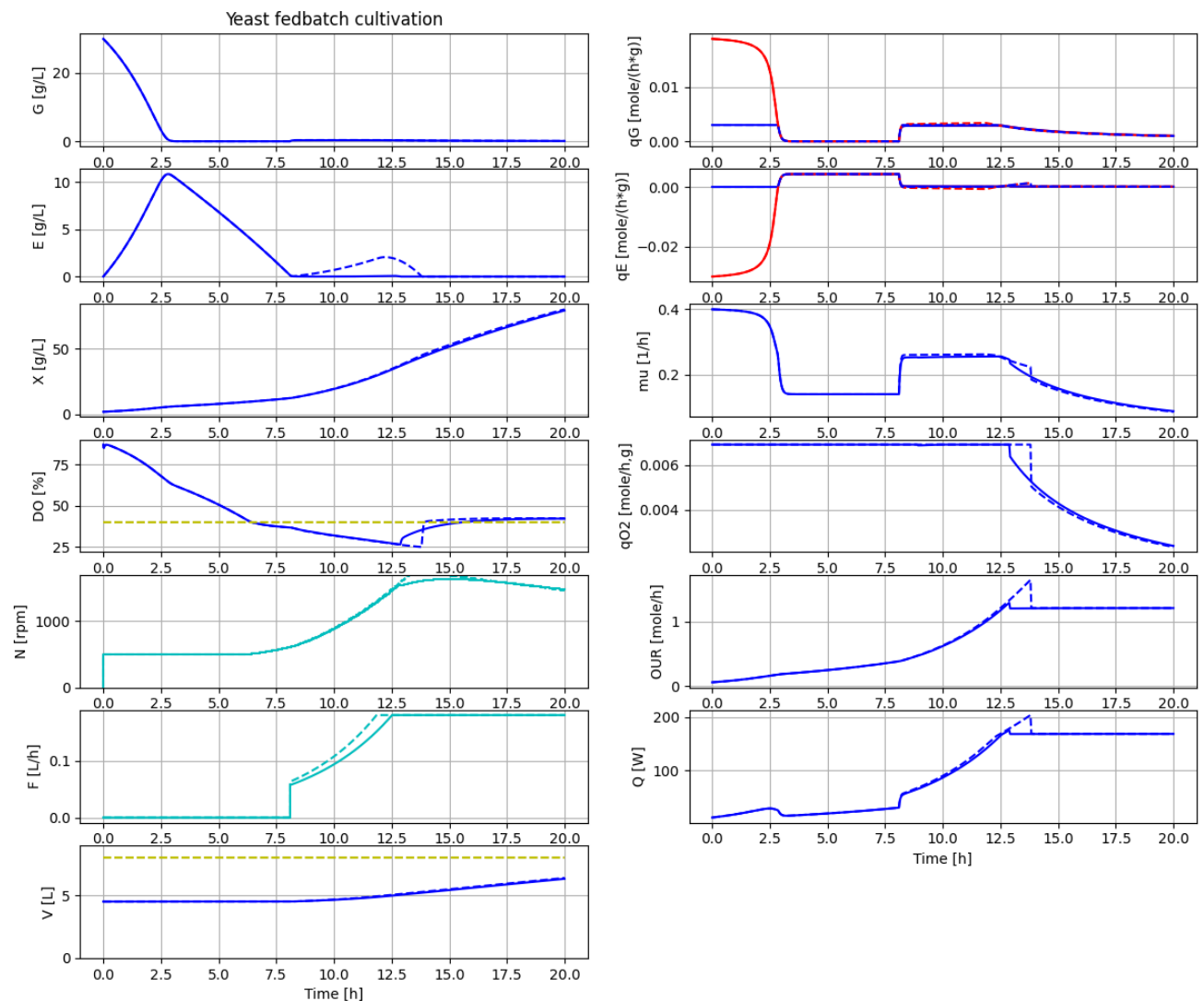
```
par(F_startExp=0.050, mu_feed=0.22); simu(20)
```

```
par(F_startExp=0.057, mu_feed=0.26); simu(20)
```



```
# And let us see what happens if the feedprofile exceed the culture capacity
newplot(title='Yeast fedbatch cultivation', plotType='Overview')
par(F_startExp=0.057, mu_feed=0.26); simu(20)
par(F_startExp=0.063, mu_feed=0.28); simu(20)
par(F_startExp=0.044, mu_feed=0.20)
```





Note that with the feedprofile that exceed culture respiratory capacity, ethanol is accumulated during time 8-12.5 hours. When the feedprofile then is constant from time 12.5 hours and on, then the accumulated ethanol is consumed over about an hour. This leads to a higher oxygen

demand and heat production during this time. The specific cell growth rate is also slightly higher during this period.

**Exercise** You can investigate the impact of changing the maximal feedrate  $F_{\max}$ . Make sure that the DO level do not get too low.

## ✓ Make your own diagrams

There are a couple of pre-defined plotType for the application that you make by the command `newplot()`. The command result in a list "diagrams" that describe the commands that make the plot when you call `simu()` or you just want to look at the last simulation again with a changed plotType using `show()`.

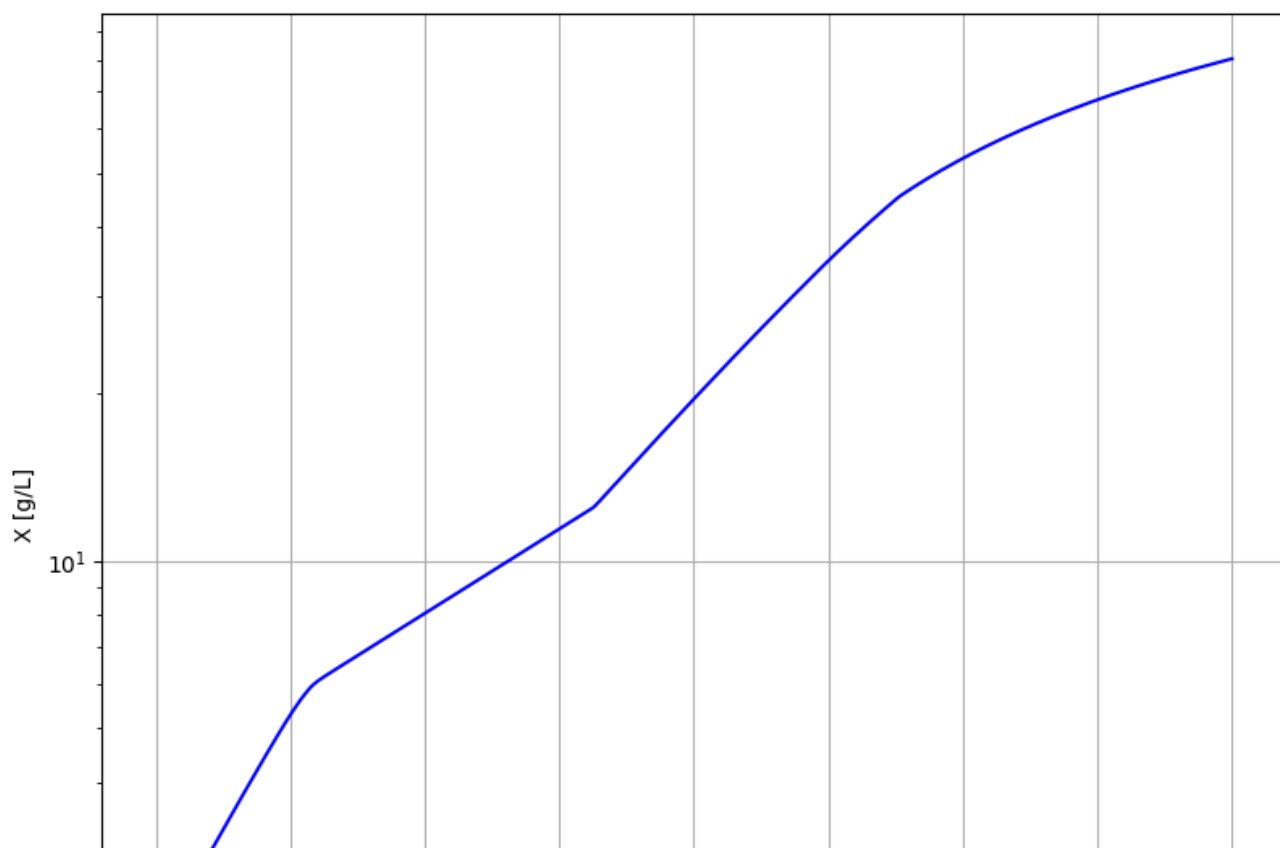
You can also in Jupyter notebook directly define the list "diagrams" and then that will be used for subsequent calls of `simu()` or `show()`. When you have made a diagram that you want to reuse many times you can bring it into the python-setup file and edit the `newplot()` command and add a new plotType.

Below a few simple examples that show how to do a diagram directly in the notebook

```
# First decrease the diagram size
plt.rcParams['figure.figsize'] = [24/2.54, 20/2.54]

# Improvise and make your own diagram - cell concentration in a logarithmic plot
plt.figure()
ax1 = plt.subplot(1,1,1)
ax1.set_ylabel('X [g/L]')
ax1.set_xlabel('Time [h]')
ax1.grid()

setLines()
diagrams.clear()
diagrams.append("ax1.semilogy(sim_res['time'], sim_res['bioreactor.c[1]'], color=
show()")
```



```
# - study the variation of Kla together with DO and N during cultivation
plt.figure()
ax1 = plt.subplot(3,1,1); ax2 = plt.subplot(3,1,2); ax3 = plt.subplot(3,1,3)
ax1.set_ylabel('DO [%]'); ax1.grid()
ax2.set_ylabel('Kla [1/h]'); ax2.grid()
ax3.set_ylabel('N [rpm]'); ax3.grid()
ax3.set_xlabel('Time [h]')

setLines()
diagrams.clear()
```