# Qslib: Python control of qPCR machines for molecular programming experiments

Constantine Evans

**Library (and poster) available at: https://github.com/cgevans/qslib**

**Documentation at: https://qslib.readthedocs.io/en/stable**

## Introduction

Real-time quantitative PCR (qPCR) machines are well-suited for fluorescence measurements of molecular programming systems: they can precisely and programmably control the temperature of many samples while taking multiple-color, real-time fluorescence readings. However, their software is often narrowly focused on qPCR and other specific uses. Qslib is a Python library that adapts the Applied Biosystems QuantStudio 5 qPCR machine for more flexible and convenient use as a general-purpose programmable temperature control and fluorescence measurement device. As a Python library rather than a GUI, qslib is designed both to allow everyday use through Jupyter notebooks, and to be readily extended for more complex applications.

## The QuantStudio 5 machine and low-level interface

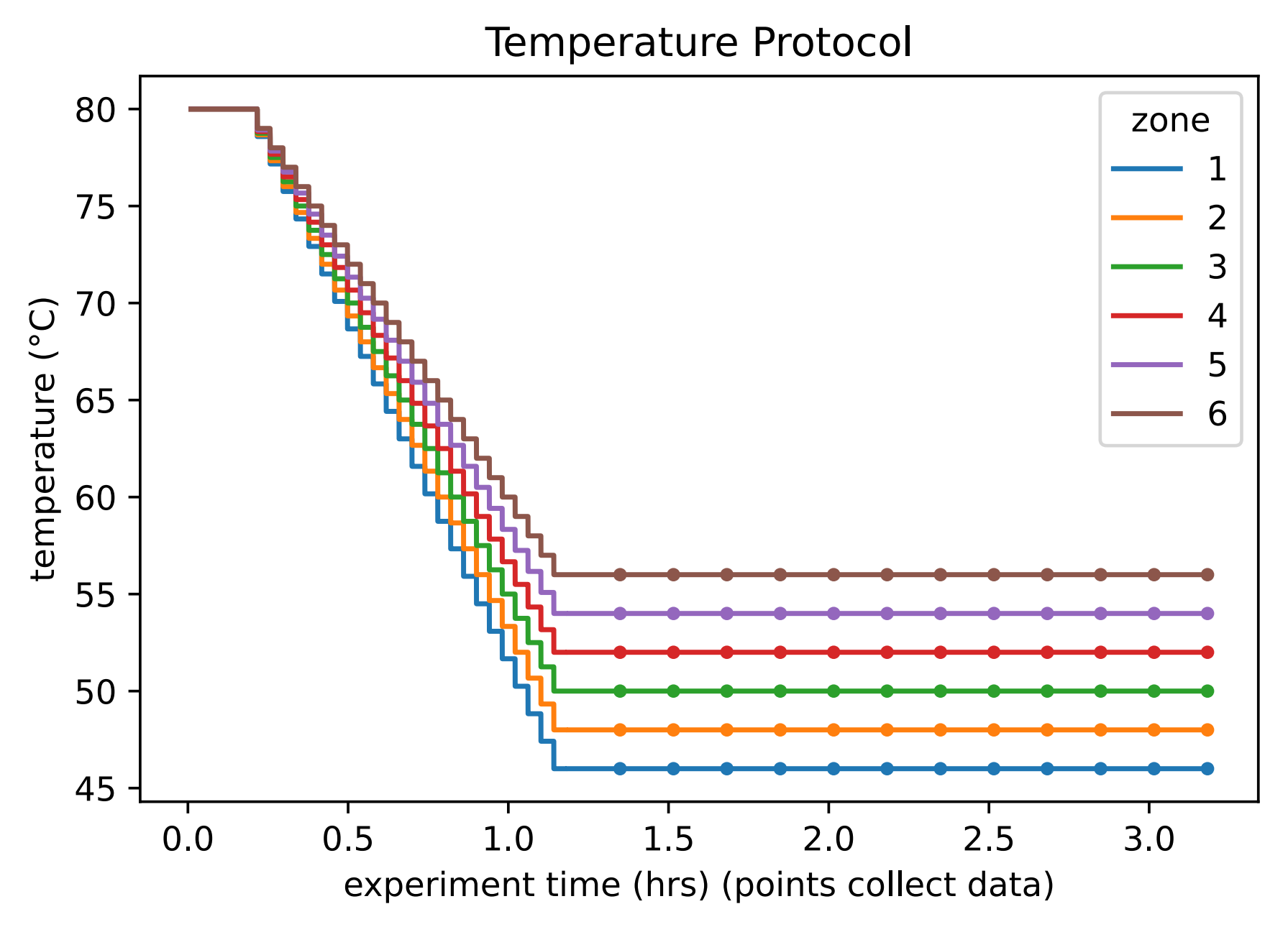
The QuantStudio 5 is a real-time PCR machine using an LED and camera for fluoroscence readings. It allows collection of data with arbitrary combinations of six excitation and six emission filters, and allows temperatures to be set independently in six groups of columns. Unusually, the machine runs Linux, and has a low-level, telnet-like interface with simple text commands, with extensive built-in documentation. It allows complete access and control of the machine's operation from running experiments down to moving motors, setting temperature controllers, and loading raw image files from the device's camera. Data files from the machine are also built on text and XML formats. Qslib is built on this openness.

## Before experiments: creating protocols

```
from qslib import *
from qslib.protocol import Exposure
```

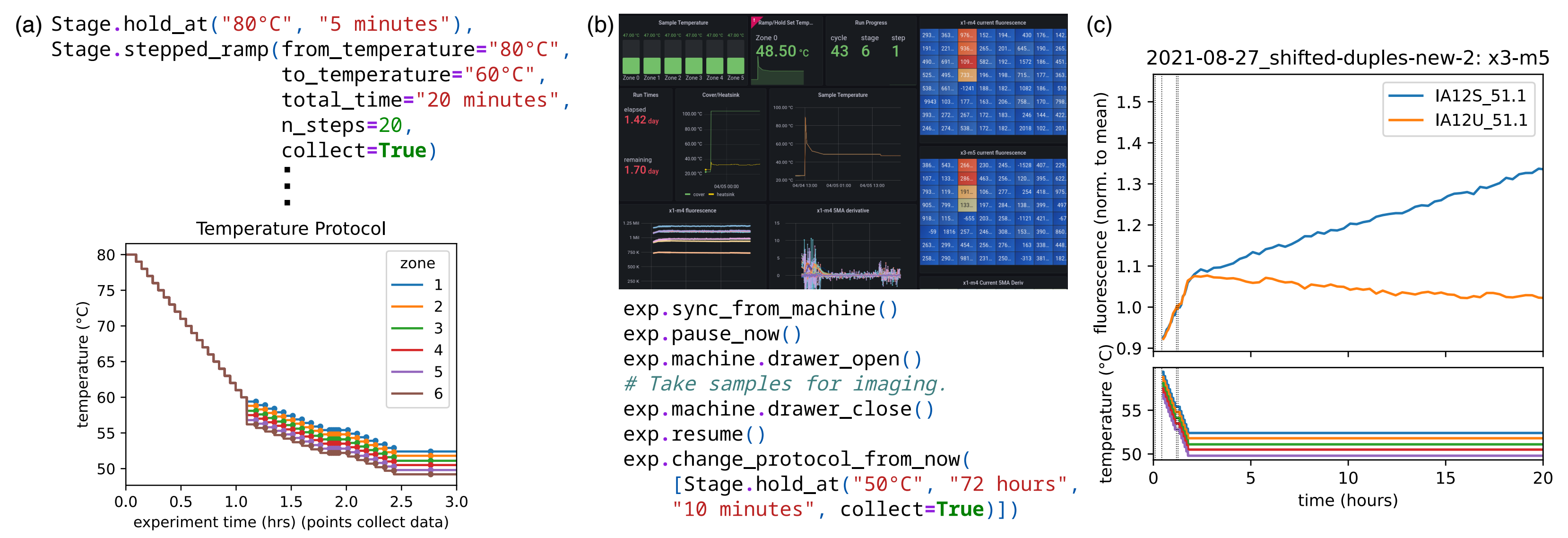
Within Python, qslib allows experiments to be specified as a combination of a `Protocol`, controlling temperature, and `PlateSetup`, giving sample information (whole-plate data is collected, so sample information is for convenience only). Protocols follow the usual qPCR machine concept of being a list of Stages, which each repeat a list of Steps for some number of repeats/cycles. But while Qslib allows stages to be quickly specified with convenience functions, and these can implement protocols that would be very difficult to specify in normal qPCR software. For example, say we'd like to ramp down to six different temperatures, and hold at each one, taking data every ten minutes during the holds, on two filter combinations:

```
protocol = Protocol([Stage.hold_at("80°C", "10 minutes"),
            Stage.stepped_ramp(from_temperature="80°C",
                    to_temperature=[46,48,50,52,54,56],
                    total_time="1 hour"),
            Stage.hold_at([46,48,50,52,54,56], "2 hours",
                    step_time="10 minutes", collect=True)
          ], filters=["x1-m4", "x3-m3"])
protocol.plot_protocol();
```



Protocols can also include custom steps to change device settings, and even run arbitrary low-level interface commands, allowing tuning of data collection, and exploration of different conditions. Fox example, a raw command could change the temperature of the heated cover, or a stage could change exposure settings:

```
Stage(CustomStep([Exposure([("x1-m4", (2000,2500))])]))
```



```
Stage.hold_at("80°C", "5 minutes"),
Stage.stepped_ramp(from_temperature="80°C",
                   to_temperature="60°C",
                   total_time="20 minutes",
                   n_steps=20,
                   collect=True)
        ⋮
```

```
exp.sync_from_machine()
exp.pause_now()
exp.machine.drawer_open()
# Take samples for imaging.
exp.machine.drawer_close()
exp.resume()
exp.change_protocol_from_now(
    [Stage.hold_at("50°C", "72 hours",
    "10 minutes", collect=True)])
```

Plate layouts can be quickly specified as Python dictionaries, or samples can be given more information, and descriptions. Qslib can give both lists of samples and formatted plate layouts.

```
layout = PlateSetup({"sample1": "A5"});
```

Experiments can be simply created, saved, and run (remotely) from these components.

```
experiment = Experiment(name="example", protocol=protocol,
    plate_setup=layout)
experiment.save_file("example.eds"); experiment.run("machine-
hostname")
```

The user doesn't even need to use the machine's screen to load samples:

```
machine = Machine("machine-hostname")
machine.drawer_open(); machine.drawer_close()
```

## During experiments: control and data access

During experiments, qslib allows complete control of machine functions, and access to data. Experiments can be loaded directly from the machine, mid-run.

```
experiment = Experiment.from_running("machine-hostname")
```

Already-loaded experiments can be efficiently updated with the latest data:

```
experiment.sync_from_machine()
```

Runs can be paused and resumed at any time, allowing samples to be physically accessed and modified, while keeping samples at fixed temperatures in the heat block:

```
experiment.pause_now(); experiment.resume()
```

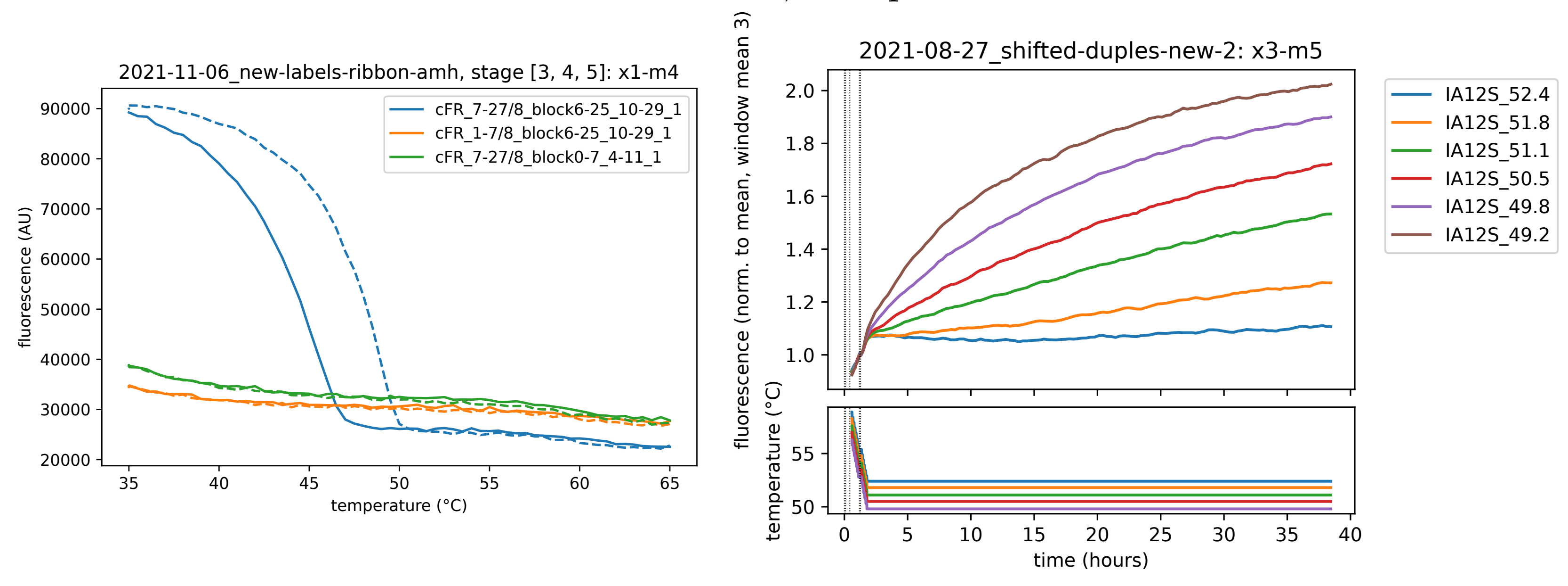
Qslib also has a monitor for *continuously* monitoring the machine from a server, and transmitting updates to databases like Influx. This lets users build, eg, real-time web dashboards for monitoring runs remotely. **See the laptop nearby for our monitoring website,** using Grafana.

While an experiment is running, qslib uniquely allows complete changes to the remaining protocol, or simple termination of current stages and replacement with new ones. For example, to lower temperatures:

```
experiment.change_protocol_from_now([
    Stage.hold_at([42,44,46,48,50,52], "2 hours",
            step_time="5 minutes", collect=True,
            filters=["x1-m1","x1-m4","x4-m4"])])
```

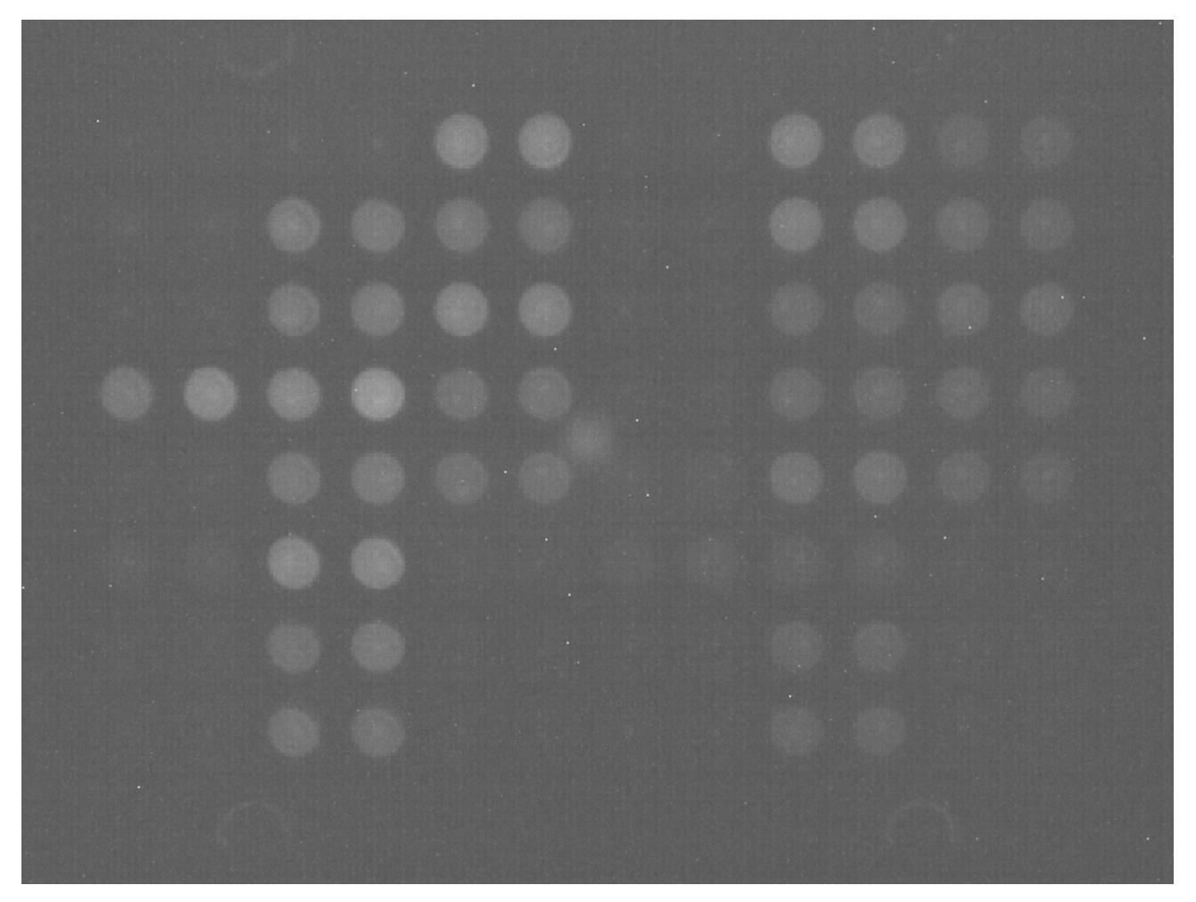
## After experiments: processing data

Qslib allows direct and convenient access to fluorescence, time, and temperature data as standard pandas dataframes (even for experiments run with the manufacturer's software), with `experiment.welldata`. It also includes convenient plotting methods for plotting fluorescence over time, and anneal-and-melt curves. These plots can include processing like per-well normalization to different parts of the experiment, smoothing, and averaging. Plots can include protocol and temperature information. Times for readings and temperatures are taken from exact clock times on the machine, with precise definitions.



Rather than using only temperature readings corresponding to fluorescence readings, qslib gives temperature readings at one-second resolution. Fluctuations in temperature during experiments, and temperatures after momentary power failures, can be seen precisely.

Qslib also allows access (under continuing development) to less processed data than normally used, included per-well brightness readings, and, if enabled, the raw TIFF images used for fluorescence readings:



## Future possibilities

Qslib started as a library to make using qPCR machines for molecular programming more convenient, but has evolved into a library that enables features rarely seen in qPCR machines, and allows exploration what features might be useful.

- More complex, possibly offline processing of raw image data into fluorescence readings, perhaps allowing more accurate or sensitive readings than currently possible.
- Better calibration and control for multi-exposure data points, camera gain settings, and exposure time changes, allowing more sensitive readings.
- Functions to allow feedback from fluorescence data to automatically control sample temperatures.
- More, and better, functions to allow convenient protocol specification, experiment control, and data processing and presentation.
- Let us know what else might be interesting.

*Qslib was developed using the QS5's SCPI interface and its documentation. It is not endorsed or supported by Applied Biosystems, and may limit your warranty. It may damage your machine, or injure you, especially if you use it to run direct, low-level commands to operate parts of the machine rather than qslib's methods, as with any software that gives you complete control over a piece of equipment.*