On-the-fly, Sample-tailored Optimisation of NMR Experiments

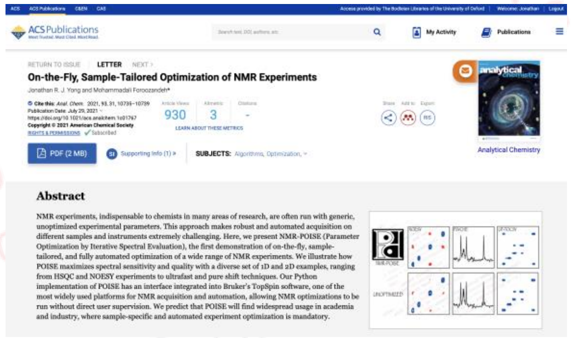
UNIVERSITY OF OXFORD

```
def converged(sim, xtol):
```

Python 3

NMR-POISE

Anal. Chem.



ACS | ACS Publications | C&EN | CAS | Access provided by The Bodleian Libraries of the University of Oxford | Welcome: Jonathan | Logout

ACS Publications — Most Trusted. Most Cited. Most Read.

RETURN TO ISSUE | LETTER | NEXT >

On-the-Fly, Sample-Tailored Optimization of NMR Experiments

Jonathan R. J. Yong and Mohammadali Foroozandeh*

Cite this: *Anal. Chem.* 2021, 93, 31, 10735–10739
Publication Date: July 29, 2021
https://doi.org/10.1021/acs.analchem.1c01767
Copyright © 2021 American Chemical Society
RIGHTS & PERMISSIONS ✓ Subscribed

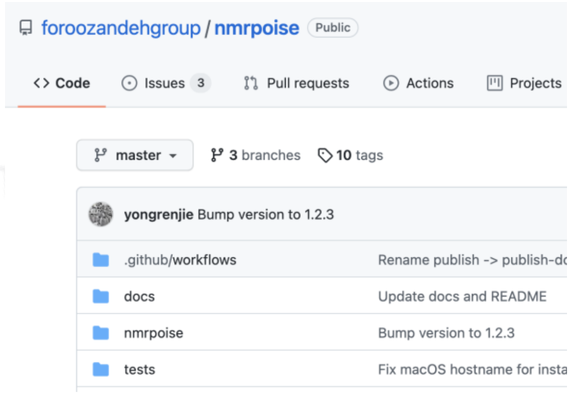
Article Views 930 | Altmetric 3 | Citations -

SUBJECTS: Algorithms, Optimization

Abstract

NMR experiments, indispensable to chemists in many areas of research, are often run with generic, unoptimized experimental parameters. This approach makes robust and automated acquisition on different samples and instruments extremely challenging. Here, we present NMR-POISE (Parameter Optimization by Iterative Spectral Evaluation), the first demonstration of on-the-fly, sample-tailored, and fully automated optimization of a wide range of NMR experiments. We illustrate how POISE maximizes spectral sensitivity and quality with a diverse set of 1D and 2D examples, ranging from HSQC and NOESY experiments to ultrafast and pure shift techniques. Our Python implementation of POISE has an interface integrated into Bruker's TopSpin software, one of the most widely used platforms for NMR acquisition and automation, allowing NMR optimizations to be run without direct user supervision. We predict that POISE will find widespread usage in academia and industry, where sample-specific and automated experiment optimization is mandatory.

GitHub



foroozandehgroup / nmrpoise Public

Code | Issues 3 | Pull requests | Actions | Projects

master | 3 branches | 10 tags

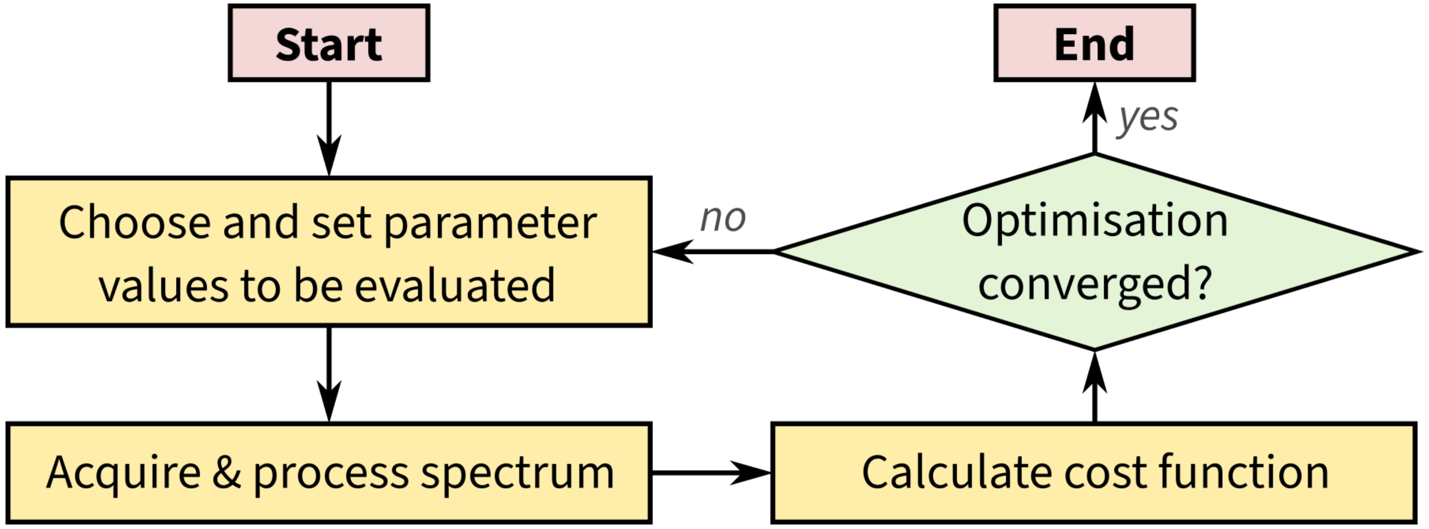
yongrenjie Bump version to 1.2.3

.github/workflows	Rename publish -> publish-dc
docs	Update docs and README
nmrpoise	Bump version to 1.2.3
tests	Fix macOS hostname for insta

Jonathan Yong
University of Oxford

UK Magnetic Resonance Managers' Meeting
Manchester, 30 June 2022

Accelerated overview

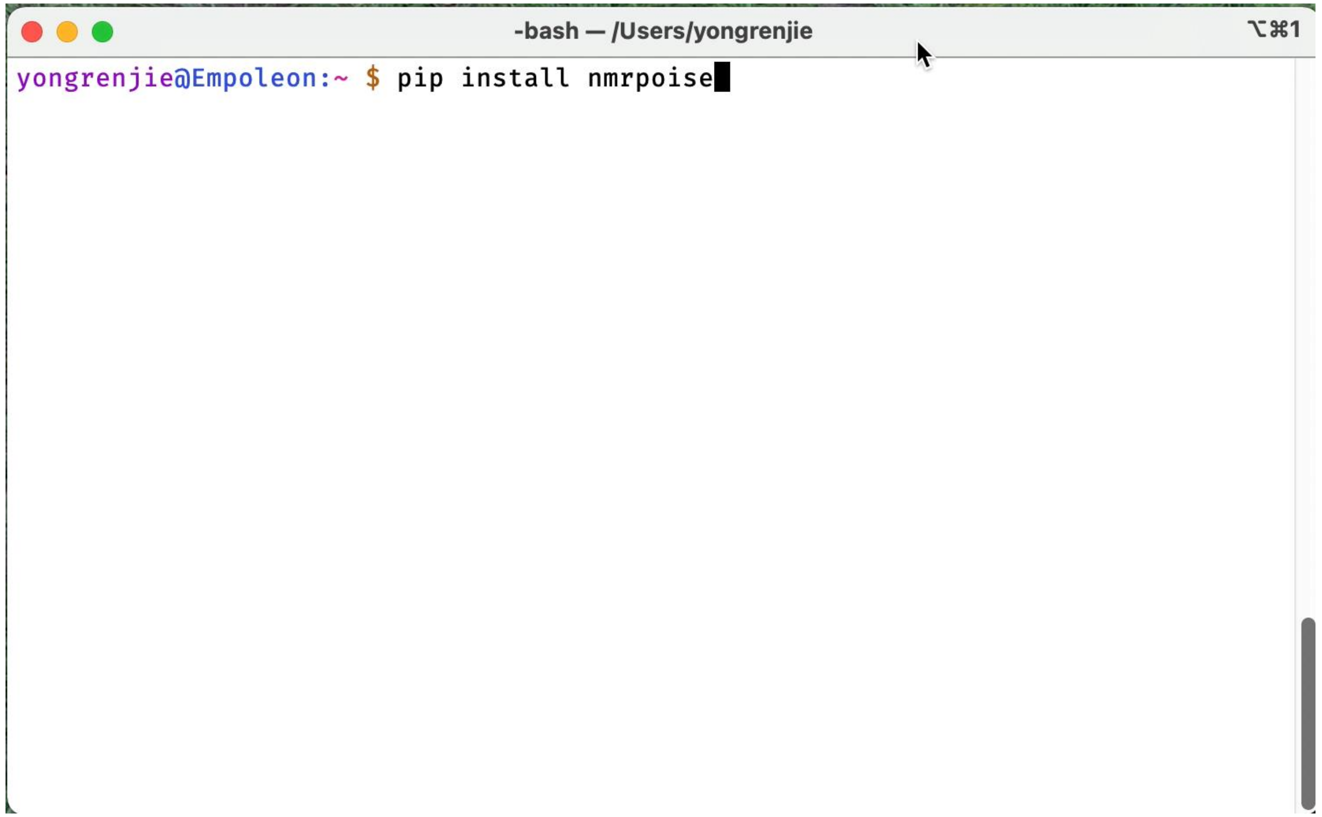

Start
Choose and set parameter values to be evaluated
Acquire & process spectrum
Calculate cost function
Optimisation converged?
no
yes
End

Optimise NMR acquisition parameters

Installation

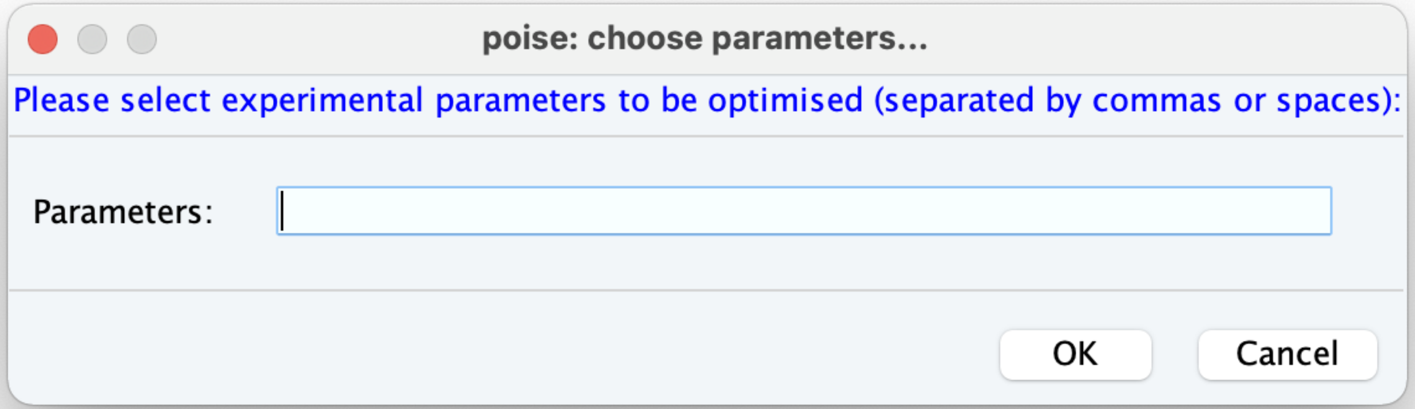
Python 3 package — just use `pip`

See online instructions if your spectrometer doesn't have Internet access



```
yongrenjie@Empoleon:~ $ pip install nmrpoise
```

Setting up an optimisation routine


poise: choose parameters...

Please select experimental parameters to be optimised (separated by commas or spaces):

Parameters:

Prompts for required information with a series of popups

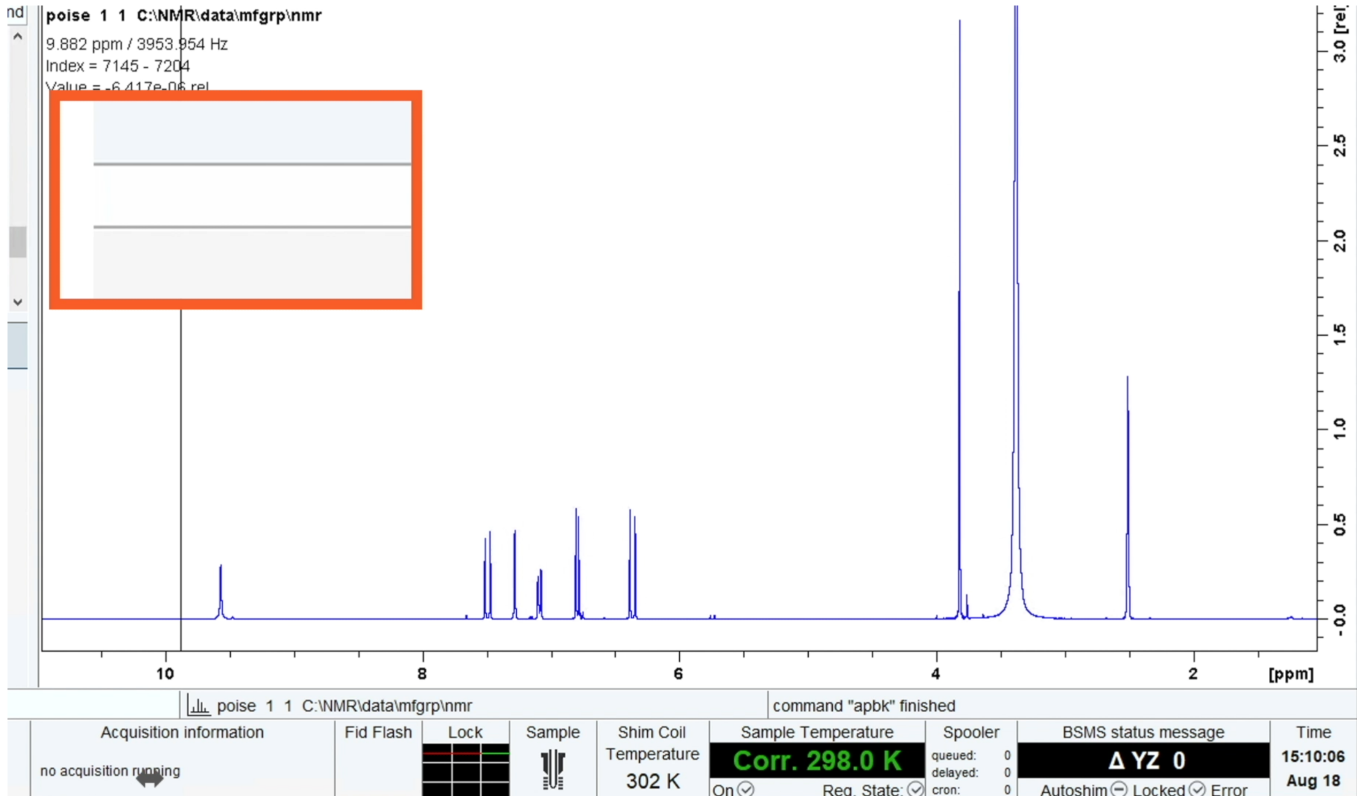
More info later

Running an optimisation routine

(Read in some parameter set first)

Run via TopSpin command line

You can script this



That’s the nice bits done…

Ingredients of a routine

Initial point

Minimum / maximum

Tolerance

$$\begin{tikzcd}
 & \text{AU programme} \\
\text{Cost function} \arrow[ru] \arrow[rd] & \\
 & \text{Python cost function}
\end{tikzcd}$$

Ingredients of a routine

Initial point

Minimum / maximum

Tolerance

Cost function

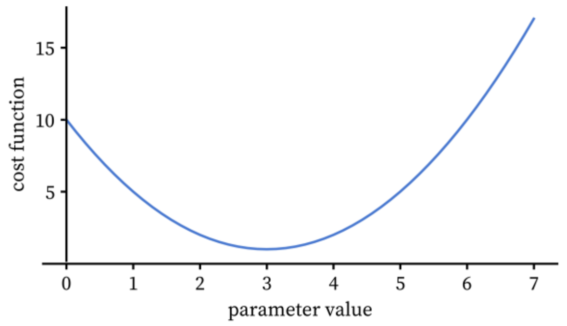
AU programme

Python cost function

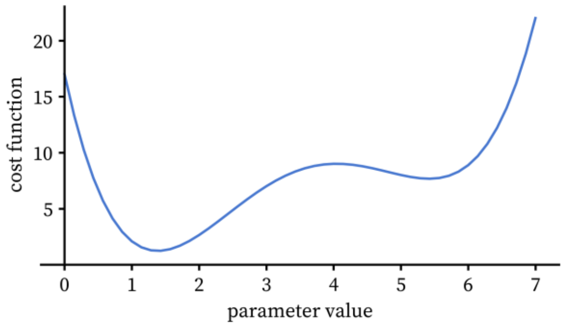
Should be your “best guess”.

Q: Does optimisation struggle if you give it a bad initial point?

A: Depends on the scientific problem you’re trying to solve



“easy” optimisation



“harder” optimisation

Ingredients of a routine

Initial point

Minimum / maximum

Usually common sense – based on instrument limitations etc.

Tolerance

$$\begin{tikzcd}
 & \text{AU programme} \\
\text{Cost function} \arrow[ur] \arrow[dr] & \\
 & \text{Python cost function}
\end{tikzcd}$$

Ingredients of a routine

Initial point

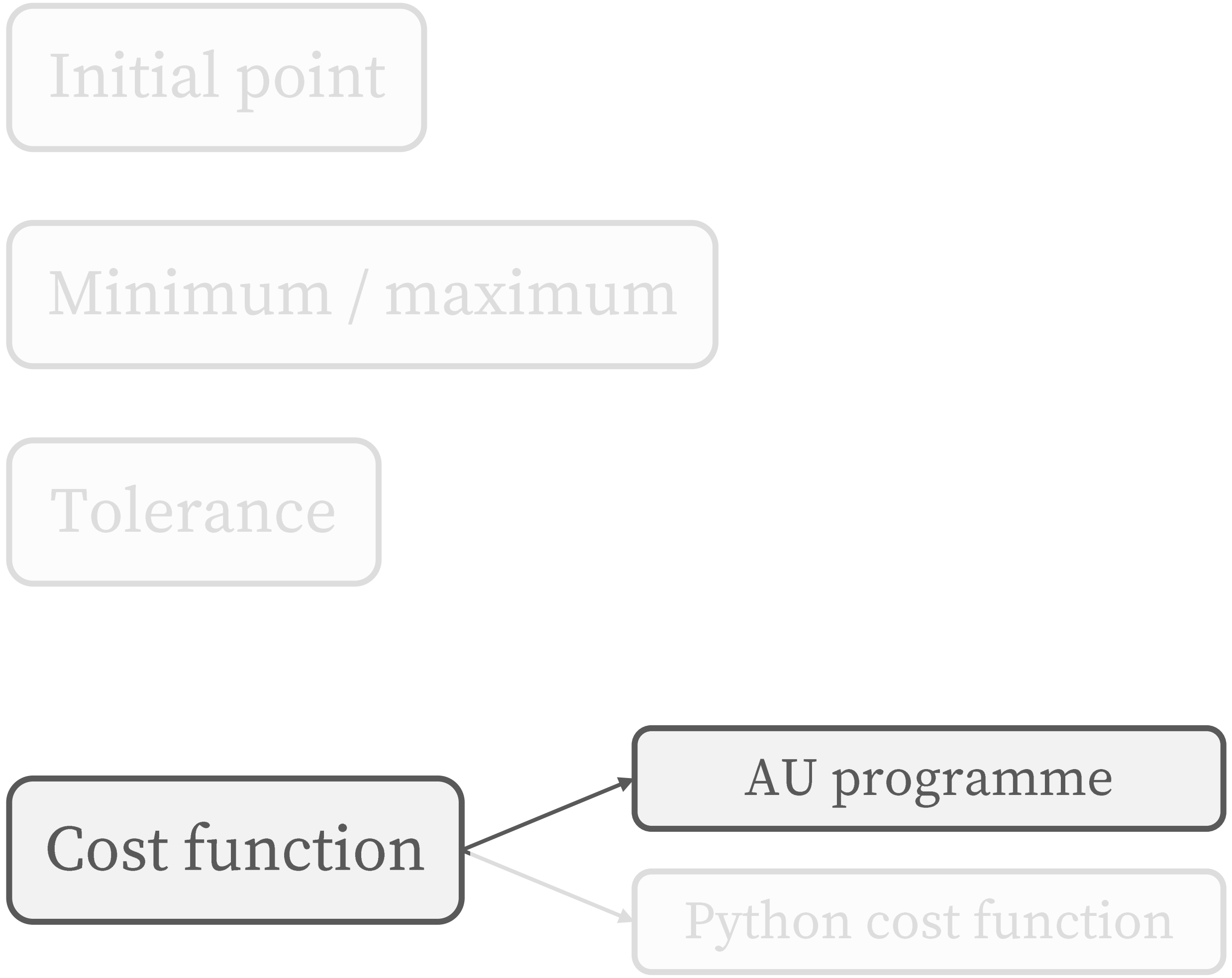
Minimum / maximum

Tolerance

Doesn’t matter as much as one might think
(as long as it’s sensible)

$$\begin{tikzcd}
 & \text{AU programme} \\
\text{Cost function} \arrow[ur] \arrow[dr] & \\
 & \text{Python cost function}
\end{tikzcd}$$

Ingredients of a routine



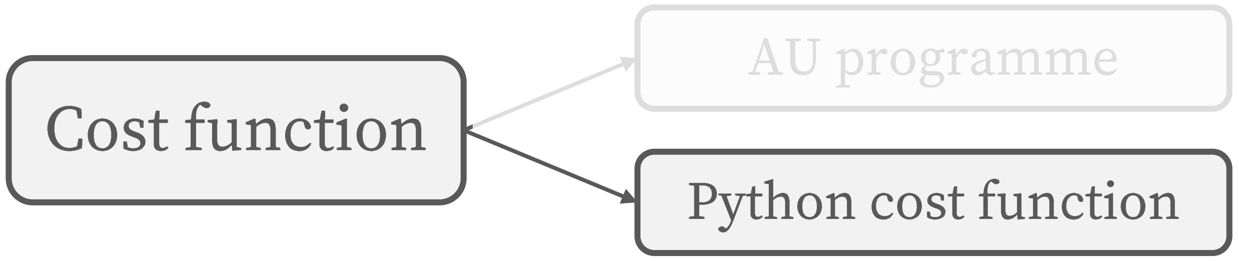
Default is usually OK (`zg; efp; apk; abs`)

Ingredients of a routine

Initial point

Minimum / maximum

Tolerance



Often simple (“integrate this peak”) but can be tricky to find something generally applicable!

Many builtins are available...

Where are **routines** stored?

```
exp/stan/nmr/py/user
├── poise.py
└── poise_backend
    ├── __init__.py
    ├── backend.py
    ├── cfhelpers.py
    ├── costfunctions.py
    ├── costfunctions_user.py
    ├── example_routines
    │   ├── asaphsqc.json
    │   ├── p1cal.json
    │   ├── [...]
    │   └── solvsupp4.json
    ├── get_cfs.py
    ├── optpoise.py
    ├── routines
    └── shared.py
```

```
{
    "name": "p1cal",
    "pars": ["p1"],
    "lb": [40.0],
    "ub": [56.0],
    "init": [48.0],
    "tol": [0.2],
    "cf": "minabsint",
    "au": "poise_1d”
}
```

Where are cost functions stored?

```
exp/stan/nmr/py/user
├── poise.py
└── poise_backend
    ├── __init__.py
    ├── backend.py
    ├── cfhelpers.py
    ├── costfunctions.py
    ├── costfunctions_user.py
    ├── example_routines
    │   ├── asaphsqc.json
    │   ├── p1cal.json
    │   ├── [...]
    │   └── solvsupp4.json
    ├── get_cfs.py
    ├── optpoise.py
    ├── routines
    └── shared.py
```

```
{
    "name": "p1cal",
    "pars": ["p1"],
    "lb": [40.0],
    "ub": [56.0],
    "init": [48.0],
    "tol": [0.2],
    "cf": "minabsint",
    "au": "poise_1d”
}
```

Where are cost functions stored?

```
exp/stan/nmr/py/user
├── poise.py
└── poise_backend
    ├── __init__.py
    ├── backend.py
    ├── cfhelpers.py
    ├── costfunctions.py
    ├── costfunctions_user.py
    ├── example_routines
    │   ├── asaphsqc.json
    │   ├── p1cal.json
    │   ├── [...]
    │   └── solvsupp4.json
    ├── get_cfs.py
    ├── optpoise.py
    ├── routines
    └── shared.py
```

```python
import numpy as np

def minabsint():
    r = get1d_real()
    i = get1d_imag()
    mag = np.abs(r + 1j * i)
    return np.sum(mag)
```

Where are cost functions stored?

```
exp/stan/nmr/py/user
├── poise.py
└── poise_backend
    ├── __init__.py
    ├── backend.py
    ├── cfhelpers.py
    ├── costfunctions.py
    ├── costfunctions_user.py
    ├── example_routines
    │   ├── asaphsqc.json
    │   ├── p1cal.json
    │   ├── [...]
    │   └── solvsupp4.json
    ├── get_cfs.py
    ├── optpoise.py
    ├── routines
    └── shared.py
```

These are “system defaults”

They are overwritten if you reinstall POISE

Your own stuff should go here

Reinstalling POISE leaves this untouched

Defining cost functions

"Full" Python 3, *separate from TopSpin's Python interface!*

Conscious design choice as TS Python doesn't work with things like numpy

We already needed numpy for the core optimisation algorithms

```
import numpy as np

def minabsint():
    r = get1d_real()
    i = get1d_imag()
    mag = np.abs(r + 1j * i)
    return np.sum(mag)
```

Minimise the **abs**olute **int**ensity of the spectrum (useful when searching for 360° null)

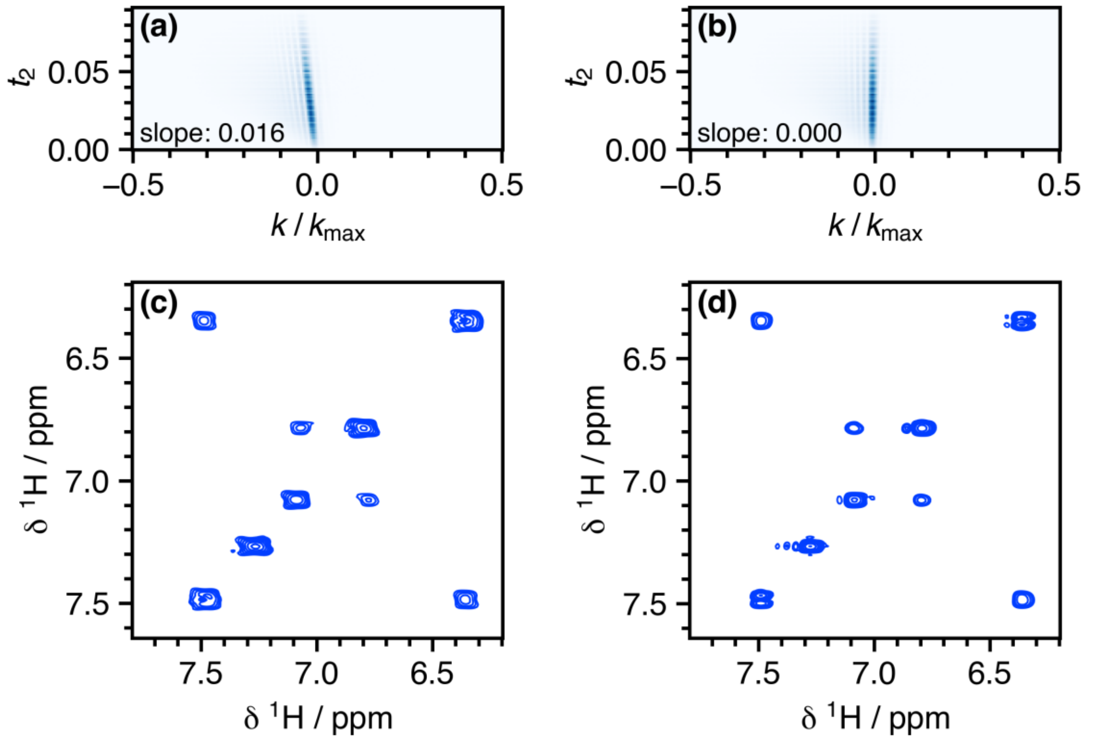
Defining cost functions

“Full” Python 3, *separate from TopSpin’s Python interface!*

Conscious design choice as TS Python doesn’t work with things like numpy

We already needed numpy for the core optimisation algorithms

This means you can make *really* complicated stuff work

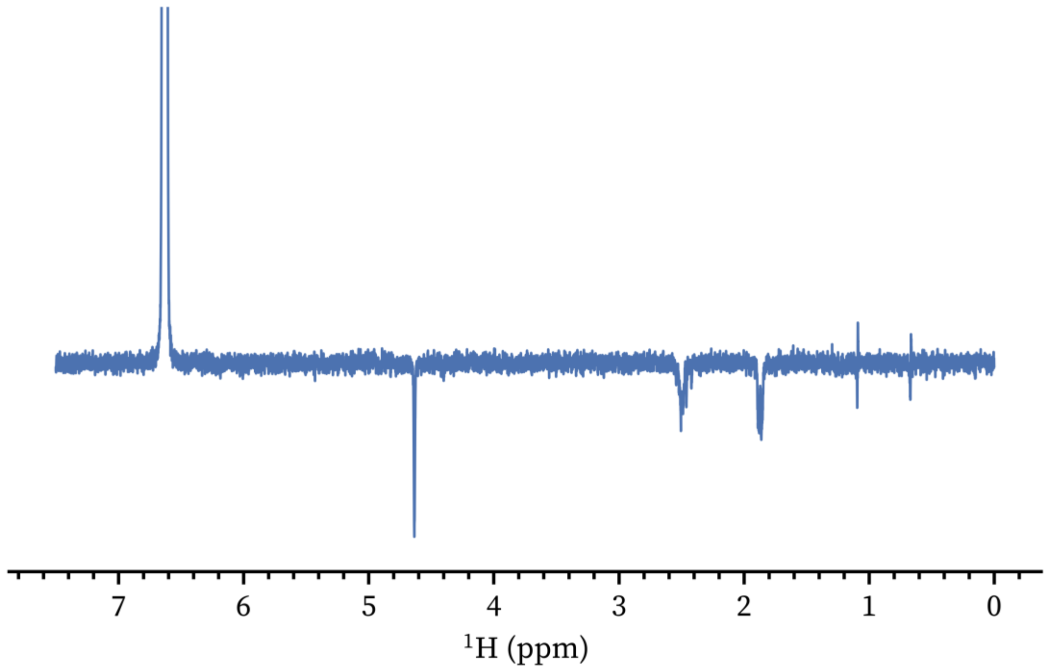


UF TOCSY unoptimised

UF TOCSY optimised

Defining cost functions

- It can be very hard to find something that works for “all” samples
- e.g. 1D NOE spectrum — you still have to pick the shift to irradiate



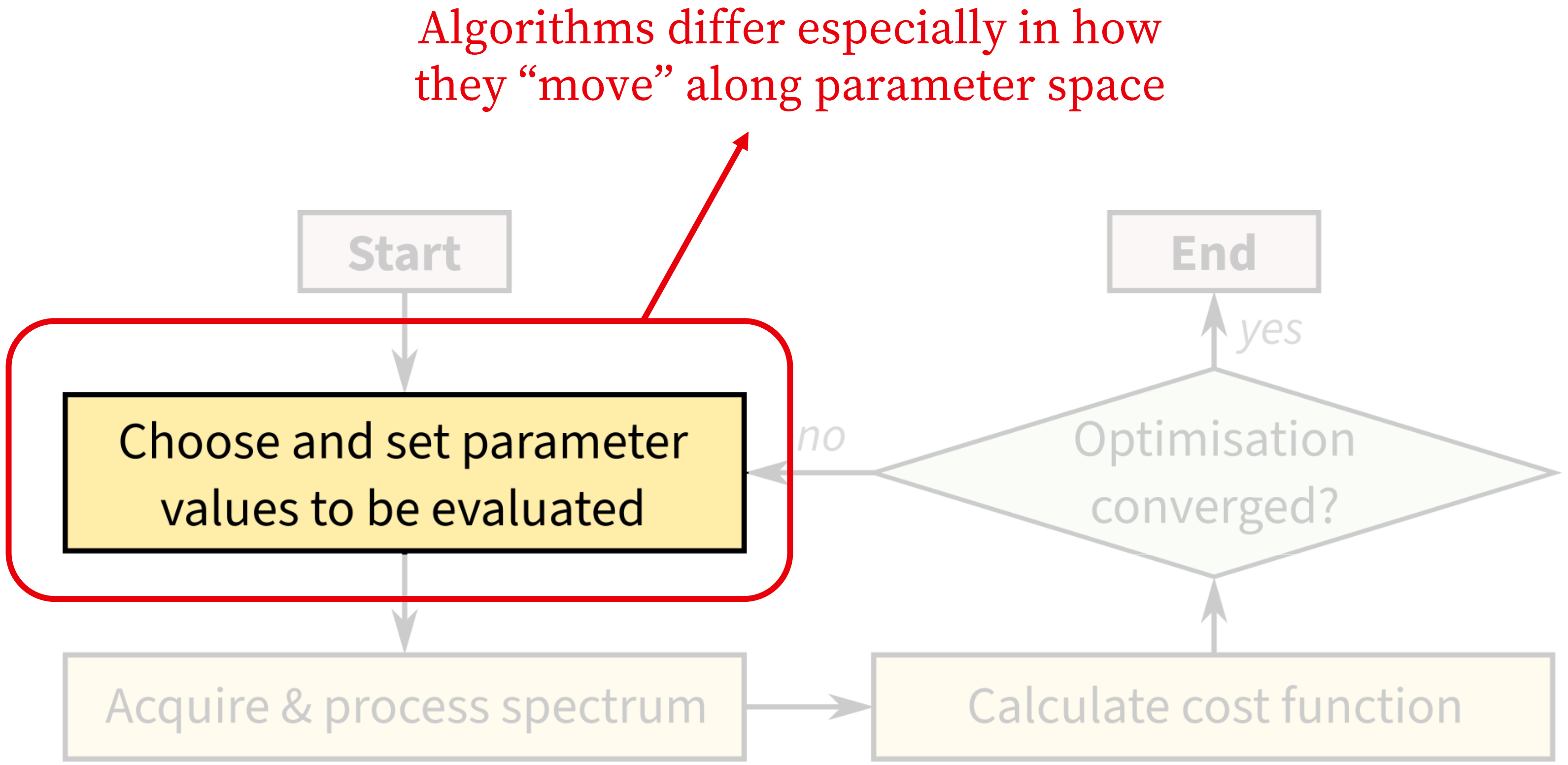
Especially problematic things

- Strong singlets (use `dpl`)
- Overlapping peaks
- Artefacts

The optimisation algorithm itself

Three algorithms available:

1. Nelder–Mead
2. Multidirectional search
3. BOBYQA



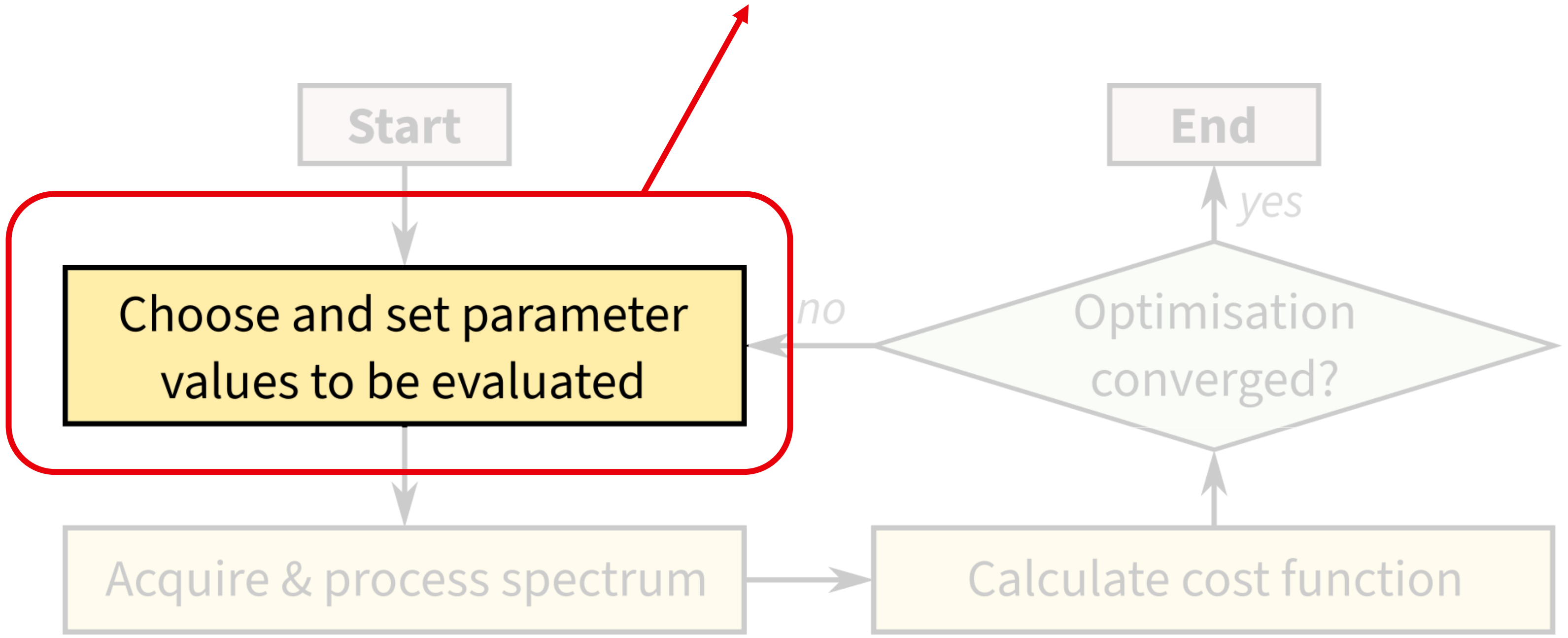
Algorithms differ especially in how they “move” along parameter space
Start
End
yes
Choose and set parameter values to be evaluated
no
Optimisation converged?
Acquire & process spectrum
Calculate cost function

The optimisation algorithm itself

Three algorithms available:

1. Nelder–Mead
2. ~~Multidirectional search~~ *Use NM*
3. BOBYQA

Algorithms differ especially in how they "move" along parameter space


Start
End
yes
Choose and set parameter values to be evaluated
no
Optimisation converged?
Acquire & process spectrum
Calculate cost function

NM vs BOBYQA

Three algorithms available:

1. Nelder–Mead
2. ~~Multidirectional search~~ *Use NM*
3. BOBYQA

$\longrightarrow$ Doesn't use the exact cost function value to decide where to move

"Less information" → slower but less affected by noise / poor CFs

NB This is solely anecdotal experience; I don't mean to make *theoretical* claims on performance

NM vs BOBYQA

Three algorithms available:

1. Nelder–Mead
2. ~~Multidirectional search~~ *Use NM*
3. BOBYQA $\longrightarrow$

Uses the exact cost function value to decide where to move

“**More information**” → faster but can go in a poor direction

NB This is solely anecdotal experience; I don’t mean to make *theoretical* claims on performance

Derivative-based algorithms

~~Newto[illegible]method~~

~~Gradien[illegible]scent~~

~~B[illegible]S~~

~~Conjuga[illegible]radient~~

Noisy data

Noisy cost function

Noisy derivatives

Derailed far too easily

How much SNR do you need?

Newto nethod

Gradien scent

B s

Conjug radient

Noisy data

Noisy cost function

Noisy derivatives

Derailed far too easily

How much SNR do you need? (or NS)

- As much as possible! :-(

- Some trial and error involved here, sorry --- we don’t have a “magic threshold value”

- This again depends on the scientific problem you’re solving

Scripting POISE

poisecal

pw90° calibration routine

*simplified version for today

```
GETCURDATA
int old_expno = expno;
DATASET(name, 99999, procno, disk, user)
RPAR("P1_CALIBRATION", "all")
GETPROSOL

XCMD("sendgui xpy poise p1cal -q")

float p1opt;
FETCHPAR("P 1", &p1opt)
p1opt = p1opt/4;

DATASET(name, old_expno, procno, disk, user)
STOREPAR("P 1", p1opt)
Proc_err(INFO_OPT, "Optimised value of p1: %.3f", p1opt);
```

Scripting POISE

Set up for optimisation

```
GETCURDATA
int old_expno = expno;
DATASET(name, 99999, procno, disk, user)
RPAR("P1_CALIBRATION", "all")
GETPROSOL

XCMD("sendgui xpy poise p1cal -q")

float p1opt;
FETCHPAR("P 1", &p1opt)
p1opt = p1opt/4;

DATASET(name, old_expno, procno, disk, user)
STOREPAR("P 1", p1opt)
Proc_err(INFO_OPT, "Optimised value of p1: %.3f", p1opt);
```

Scripting POISE

```
GETCURDATA
int old_expno = expno;
DATASET(name, 99999, procno, disk, user)
RPAR("P1_CALIBRATION", "all")
GETPROSOL
```

Optimise

```
XCMD("sendgui xpy poise p1cal -q")
```

```
float p1opt;
FETCHPAR("P 1", &p1opt)
p1opt = p1opt/4;

DATASET(name, old_expno, procno, disk, user)
STOREPAR("P 1", p1opt)
Proc_err(INFO_OPT, "Optimised value of p1: %.3f", p1opt);
```

Scripting POISE

```
GETCURDATA
int old_expno = expno;
DATASET(name, 99999, procno, disk, user)
RPAR("P1_CALIBRATION", "all")
GETPROSOL

XCMD("sendgui xpy poise p1cal -q")

float p1opt;
FETCHPAR("P 1", &p1opt)
p1opt = p1opt/4;

DATASET(name, old_expno, procno, disk, user)
STOREPAR("P 1", p1opt)
Proc_err(INFO_OPT, "Optimised value of p1: %.3f", p1opt);
```

Retrieve optimised p1

You can get value of cost function via `TI` parameter

Scripting POISE

```
GETCURDATA
int old_expno = expno;
DATASET(name, 99999, procno, disk, user)
RPAR("P1_CALIBRATION", "all")
GETPROSOL

XCMD("sendgui xpy poise p1cal -q")

float p1opt;
FETCHPAR("P 1", &p1opt)
p1opt = p1opt/4;

DATASET(name, old_expno, procno, disk, user)
STOREPAR("P 1", p1opt)
Proc_err(INFO_OPT, "Optimised value of p1: %.3f", p1opt);
```

Set new parameter value

Generality is a tradeoff

- POISE is designed to be *general* — write your own routines, cost functions
- At the same time, this means that you can't make a highly specialised workflow
  - e.g. T_1 calculation routine: do inversion–recovery expt and search for null in intensity. Faster than 2D IR, but faster methods exist
  - `pulsecal fast` (doesn't do `rga`) is actually faster than `poisecal` (but still inaccurate)

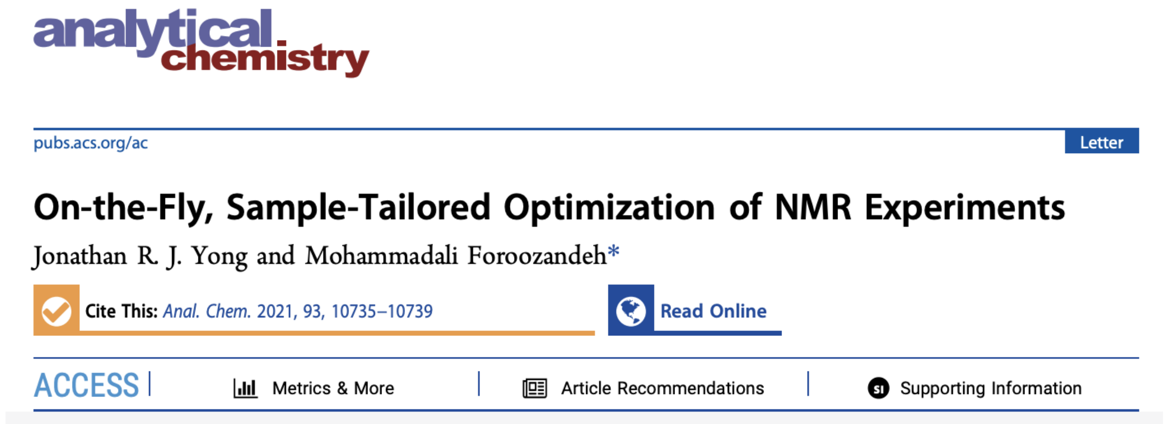
Thanks [and (other) questions]

Mohammadali Foroozandeh (Oxford)

Tim Claridge (Oxford)

Many others — see paper...

££:
Clarendon Fund (Oxford)
SBM CDT / EPSRC / industrial partners



analytical chemistry

pubs.acs.org/ac Letter

On-the-Fly, Sample-Tailored Optimization of NMR Experiments

Jonathan R. J. Yong and Mohammadali Foroozandeh*

Cite This: *Anal. Chem.* 2021, 93, 10735–10739 Read Online

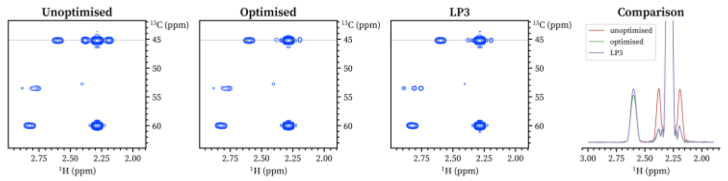
ACCESS | Metrics & More | Article Recommendations | Supporting Information

Anal. Chem. **2021,** *93* (31), 10735–10739.

Poster 42

for my other work on NOAH stuff...

Q: How do you make your plots?
A: I wrote a Python package to do it!
GitHub: `yongrenjie/penguins`


Unoptimised
Optimised
LP3
Comparison
^{13}C (ppm)
^{1}H (ppm)
unoptimised
optimised
LP3