Optimising NMR Spectroscopy through Method and Software Development

Jonathan Yong

University of Oxford

Contents

refsection:1

refsection:2

refsection:3

Chapter 3

POISE

chpt:poise

This chapter describes the development of software for on-the-fly optimisation of NMR experimental parameters, titled POISE (*Parameter Optimisation by Iterative Spectral Evaluation*). The primary benefit of this is that parameters may be adjusted for individual spectrometers and samples, which may vary greatly in their chemical properties. POISE is primarily written in Python 3. In this chapter, I first provide some details about the implementation of POISE. The bulk of the text which follows is devoted to a number of applications in liquid-state NMR spectroscopy. At the end, the extension of the concept of on-the-fly optimisation to ESR spectroscopy is also briefly discussed: I contributed code for this, but the experimental ESR work and data analysis were carried out by Jean-Baptiste Verstraete (University of Oxford).

The work in this chapter forms the subject of two publications:

- Yong, J. R. J.; Foroozandeh, M. On-the-Fly, Sample-Tailored Optimization of NMR Experiments. *Anal. Chem.* **2021**, *93*, 10735–10739, DOI: 10.1021/acs.analchem.1c01767
- (JBV et al., manuscript submitted)

3.1 Introduction

ec:poise__introduction

In the previous chapter, I covered various approaches to improving pure shift NMR through the use of optimisation. Although the optimisation code written there was highly specialised and only designed to work on pure shift applications, it was envisioned that this optimisation approach could be applied to essentially *any* NMR experiment where parameter optimisation was required. In principle, this description is appropriate for *every* experiment: even the simplest pulse–acquire experiment can be optimised through the use of Ernst angle excitation. More complex examples, such as 2D experiments, typically have parameters which should be chosen to optimally match coupling constants (INEPT delay) or relaxation rates (NOE mixing time).

In practice, the need for accurate parameters is often 'solved' through the use of compromise values, which typically fall in the middle of an expected range for typical molecules. For example, these values may be stored as part of a parameter set designed to be reused. Alternatively, parameter values may be optimised 'by hand'. However, compared to these, the use of experimental optimisation has several benefits. It is:

1. *sample-specific*, and as long as the default values are within the optimisation bounds, the optimisation will yield performance which is no worse than the defaults;
2. more *robust* towards unusual molecular structures, which have physical or chemical properties which fall outside of an expected range;
3. *instrument-specific*, so can compensate for spectrometer imperfections.
4. *automated*, so does not require an expert to adjust parameter values manually, or even any user intervention for that matter;
5. *objective*, in that the quality of a spectrum can (in principle) be mathematically measured through a cost function; and
6. *fast*, in that it uses an algorithm which is designed to achieve rapid decreases in the objective function: many 'manual' optimisations involve either trial-and-error or an exhaustive grid search (i.e. increasing a parameter value one step at a time), neither of which are efficient.

Despite these advantages, experimental optimisation of NMR parameters has seen only limited use. In fact, although there are several examples of such optimisations in laser,[2] nuclear quadrupole resonance,[3–5] and ESR[6] spectroscopies, the only direct parallel in NMR which I have found is that of the eDUMBO pulses for heteronuclear[7,8] and homonuclear dipolar[9] decoupling in solid-state MAS experiments. In this work, the Emsley group used 'direct spectral optimisation' (equivalent to what I call 'experimental optimisation') to determine the best coefficients for a Fourier series pulse. The performance of these pulses was measured by a cost function which (primarily) took into account the intensity of the detected peaks: a larger intensity corresponds to better decoupling performance. Interestingly, the aim of using an experimental optimisation here was not to obtain sample-specific pulses (point (1)), but rather to account for the 'spectrometer response', i.e. instrumental non-idealities (point (3)). It was assumed that the compound used for the optimisation was a suitably representative choice, so that the optimisation result could simply be applied to other samples with no change.

The likely reason for the low popularity of experimental optimisations is *time*. In most cases, it is probably easier to run NMR optimisations in a theoretical manner, which can be much faster compared to the acquisition of a spectrum (depending on what simulations are involved), and also circumvents the effect of noise. Examples of such optimisations include the design of shaped pulses, either through optimal control theory[10–15] or by simple parameterisation:[16–21] these were briefly discussed in § 2.4.3. (In fact, even the aforementioned eDUMBO pulses were not *originally* designed as an experimental optimisation: they are actually an enhancement of the DUMBO decoupling schemes, which were optimised using numerical simulations.[22]) It is also possible to design entire pulse sequences using *in silico* optimisations:[23–27] this is essentially what I did with the dPSYCHE experiment (§ 2.6). However, doing this in an experimental fashion would almost certainly be prohibitively slow.

In this chapter, I aim to provide a convincing argument that experimental optimisation is not necessarily slow. In particular, I will show that it is often possible to devise optimisation routines which yield improved results in a matter of minutes. All the optimisations here are performed using a software package written by me, called POISE (Parameter Optimisation by Iterative Spectral Evaluation). POISE is open-source (`https://github.com/foroozandehgroup/nmrpoise`) and can be installed in a single step through `pip install nmrpoise`. Furthermore, it comes with extensive user documentation, both in the form of a text guide (`https://foroozandehgroup.github.io/nmrpoise`) as well as video (`https://www.youtube.com/watch?v=QTCeSCRZs4I`).

In contrast to previous work, which typically feature optimisations targeted at one specific application, I have endeavoured to make POISE as customisable and as broad as possible. This generality is what allows a single software package, POISE, to perform all the optimisations described in this chapter; it also means that other users can devise specific cost functions and optimisation procedures for their own use. Thus, *POISE is more than just the applications shown later in this chapter*: it is really a platform which makes it possible to carry out arbitrary optimisations on an NMR spectrometer.

3.2 Technical overview

sec:poise__technical

In this section, I first cover the general principles underlying, and the implementation of, POISE. The basic operation of POISE is summarised in FIG, which is essentially a generalised version of the pure shift optimisations carried out in § 2.4.

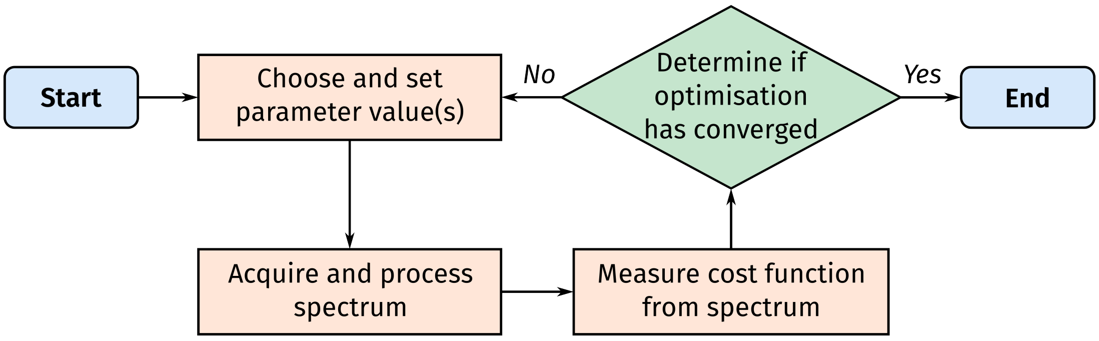
Start
Choose and set parameter value(s)
No
Determine if optimisation has converged
Yes
End
Acquire and process spectrum
Measure cost function from spectrum

fig:poise_flowchart

Figure 3.1: Flowchart depicting the main steps in a POISE optimisation.

Almost all aspects of this can be customised by the user, which I will now describe. I make a distinction here between an optimisation *routine*, as well as the *settings* used to run these routines. Routines consist of a series of predefined variables, such as the parameter(s) to be optimised: however, these may be optimised in different *ways*, which is where the settings come in. When discussing individual applications in § 3.4, I will make repeated reference to these components of an optimisation.

3.2.1 Routines

subsec:poise__routines

An *optimisation routine*, as defined in POISE, consists of the following components:

1. *Name*

   This is an identifier used to refer to the entire routine, which is arbitrary, but should ideally be descriptive.

2. *Parameters*

   The parameters to be optimised. These are given as strings and directly correspond to TopSpin parameter names, for example, `P1` for a pulse width.

3. *Initial guess* (one per parameter)

   The point at which the optimisation is started. Naturally, this should represent the user's best guess at where the optimum lies. It is generally sensible to choose the unoptimised, 'default' values for these.

4. *Lower* and *upper bounds* (one each per parameter)

   Most parameters have a 'chemically sensible' range, or alternatively, instrumental limits may sometimes restrict the range of parameter values which can be explored.

5. *Tolerances* (one per parameter)

   This loosely corresponds to the level of accuracy required for the optimisation. It is pointless setting this to be too small (i.e. requesting an overly accurate optimum), as the value of the cost function at two points too close together will likely differ only by noise. Conversely, setting this to be too larger may yield an inaccurate result. This makes it sound as if there is little room for error, but in practice getting the order of magnitude correct is usually enough (and the desired accuracy is also often reasonably clear from the context);

6. *AU programme*

   The AU programme defined here is used to acquire and process the spectrum. The user may leave this empty, in which case POISE automatically detects the dimensionality of the experiment and performs standard processing steps (Fourier transformation, window multiplication, phase correction, and baseline correction). However, this allows for almost infinite customisation of the actual spectral measurement: for example, the AU programme may call other scripts in TopSpin which create shaped pulses.

7. *Cost function*

   As before, this measures the 'badness' of the spectrum thus recorded, and as before, the optimisation seeks to minimise this value. The cost function is written in Python 3: this design decision is considered later in § 3.2.3. Several cost functions which cover 'typical' optimisation scenarios, such as maximising or minimising some signal intensity, come pre-installed with POISE, meaning that users do not necessarily need to write their own cost function if they are not familiar with Python.

POISE allows users to create new routines interactively through a series of dialog boxes. Alternatively, routines themselves can be created on-the-fly using the `poise −create` command: this is useful when some components are not known beforehand, such as if the optimum from a different optimisation is to be used as the initial point in a new one. However, this is limited to single-parameter routines.

After being created, routines are stored in the human-readable JSON format: they can therefore be modified using any text editor. Examples of these JSON files are presented in subsequent sections.

3.2.2 Optimisation settings and algorithms

subsec:poise__settings

Once the user has defined a routine, it can then be run from the TopSpin command line using the command `poise ROUTINE_NAME`. However, the routine itself merely controls what parameters are being optimised: it does not specify what experiment is to be run (i.e. the pulse programme), nor any of the other parameters in the experiment. These must be set by the user, and can most conveniently be stored in a TopSpin parameter set which can simply be loaded before starting the optimisation. This flexibility means that the same *type* of optimisation may be applied to different pulse sequences without having to create individual routines for each: for example, an experiment to optimise the NOE mixing time (as described in § 3.4.3) can be run with different versions of the NOESY sequence depending on what is most appropriate. Likewise, parameters such as the number of scans can be adjusted in order to run optimisations on samples with different concentrations.

Once the experiment parameters have been set up, there are a few more options which control how the optimisation is carried out:

- the `−maxfev` option allows the user to control the maximum number of function evaluations, or in other words, the maximum number of experiments run. If the optimisation has not converged after acquiring this many spectra, the best result so far is simply returned. This effectively allows the time spent on optimisation to be capped.
- the `−quiet` option silences all output from the optimiser (the best parameters found are stored in the dataset itself after the optimisation ends, and can therefore be retrieved). This is useful when a POISE optimisation is to be run under automation.
- the `−separate` option allows each function evaluation to be run in a new experiment number, so that the optimisation trajectory can be analysed after its conclusion.
- perhaps most importantly, the `−algorithm` option allows the user to choose one of three optimisation algorithms: the Nelder–Mead (NM) method,[28] the multidirectional search (MDS) method,[29,30] and the Py-BOBYQA trust region method.[31,32] All of these are derivative-free algorithms in that they do not use the value of ∇f at any point: as described in § 2.4.1, gradients cannot be accurately estimated when there is noise in the cost function.[*] I will now describe these algorithms in slightly greater detail.

The NM method is a highly popular derivative-free optimisation algorithm, which maintains a set of points $\{y_1, y_2, \cdots, y_{n+1}\}$ during the optimisation, where n is the number of parameters being

[*]Nocedal and Wright[33] give an upper bound on the finite difference gradient (as compared to the true gradient) of $\eta(x;\varepsilon)/\varepsilon + O(\varepsilon^2)$, where x is the point at which the gradient is being measured, ε is the step size used for the finite difference calculation, and $\eta(x;\varepsilon)$ is the noise in the region $[x - \varepsilon, x + \varepsilon]$. If ε is small, the first term (the error due to noise) is large, and if ε is large, the second term (the error due to the finite difference approximation) is large.

optimised. The convex hull of these points, Y, is the smallest possible set of points containing all the y_k such that

$$\forall x_1, x_2 \in Y, \forall \alpha \in [0,1], \alpha x_1 + (1-\alpha)x_2 \in Y, \tag{3.1}$$ {eq:convex_hull}

and is called a *simplex*. To provide an analogy for $n = 2$, the convex hull is the shape obtained by stretching a rubber band around three pins placed at y_1, y_2, y_3. If this convex hull is nonempty—or equivalently, if the n vectors $y_k - y_1$ $(2 \leq k \leq n+1)$ are linearly independent—then the simplex is called *nonsingular*. (In the $n = 2$ case, the convex hull would be empty if the three points were collinear.)

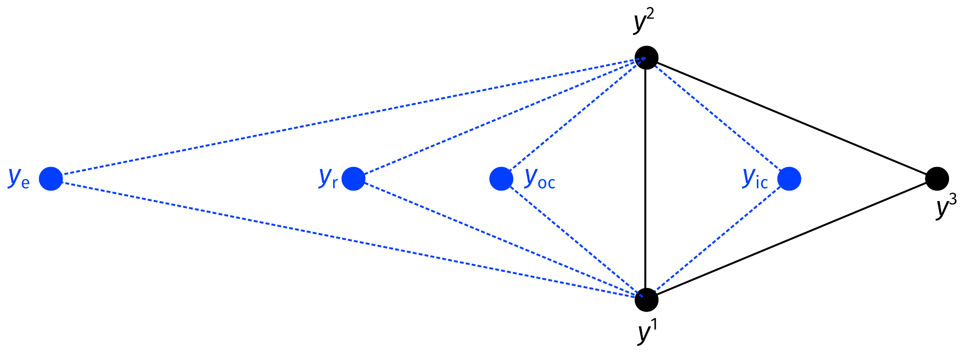

fig:neldermead

Figure 3.2: Diagram showing various points evaluated in one iteration of the Nelder–Mead algorithm (for an optimisation of two parameters). The solid black lines indicate the current simplex, which is assumed to be ordered such that y_1 is the best point (has the lowest cost function value) and y_3 the worst. The blue dots indicate the trial points which the algorithm attempts to replace y_3 with, and are further discussed in the text. Blue dashed lines indicate the simplex which would result if the corresponding trial point is accepted.

The NM algorithm is in fact quite intuitive to understand. The optimisation begins by measuring the cost function f at every point of the simplex, and sorting the points in ascending order of cost function values (i.e. from best to worst), such that $f(y_1) \leq f(y_2) \leq \cdots \leq f(y_{n+1})$. The centroid of the simplex is defined by the best n points,

$$\bar{y} = \sum_{i=1}^{n} y_i. \tag{3.2}$$ {eq:simplex_centroid}

On each iteration of the NM algorithm, we attempt to replace the worst point y_{n+1} with a better point (fig. 3.2). The search for the new point is performed in several steps: first, the worst point is *reflected* about the centroid of the simplex to obtain a new point:

$$y_\mathrm{r} = \bar{y} - (y_{n+1} - \bar{y}). \tag{3.3}$$ {eq:nm_reflect}

The value of the cost function is evaluated at this point, and is critical in determining how the algorithm proceeds. If this reflected point falls in the middle of the pack, such that $f(y_1) \leq f(y_\mathrm{r}) < f(y_n)$, this represents a 'modest' improvement in the cost function: we simply replace the worst point with this and continue to the next iteration.

On the other hand, if the reflected point is better than all the other points (i.e. $f(y_\mathrm{r}) < f(y_1)$), then we ambitiously attempt to *expand* the simplex even further in that direction:

$$y_\mathrm{e} = \bar{y} - 2(y_{n+1} - \bar{y}). \tag{3.4}$$ {eq:nm_expand}

Of course, there is no guarantee that this is necessarily better than y_r; therefore, we choose whichever point of y_r or y_e had a lower value of f, and replace the worst point with this and continue to the next iteration.

If the reflected point is an improvement on the worst point but is no better than the remaining points, in that $f(y_n) \leq f(y_\mathrm{r}) < f(y_{n+1})$, then the algorithm performs an *outside contraction*, which resembles a half-hearted reflection:

$$y_\mathrm{oc} = \bar{y} - (1/2)(y_{n+1} - \bar{y}). \tag{3.5}$$ {eq:nm_outside_contrac

Conversely, if the reflected point is even worse than the worst point ($f(y_{n+1}) \leq f(y_\mathrm{r})$), then this suggests that that search direction is very poor: we thus perform an *inside contraction*, which uses a point halfway between the worst point and the centroid:

$$y_\mathrm{oc} = \bar{y} + (1/2)(y_{n+1} - \bar{y}). \tag{3.6}$$ {eq:nm_inside_contract

If either of these contracted points are any better than y_r, then we replace the worst point in the simplex and continue to the next iteration; otherwise, we conclude that no search direction was good, and simply shrink the simplex towards the current best point by replacing each point y_k with $(y_k + y_1)/2$. In practice, these 'last-resort' shrink steps occur very rarely.

Convergence... self implementation instead of using scipy

In the preceding discussion, we noted that the simplex Y was nonsingular if the n vectors $y_k - y_1$ were linearly independent. Equivalently, the matrix M formed by concatenating these vectors

$$M = \left(y_2 - y_1, y_3 - y_1, \cdots, y_{n+1} - y_1\right) \tag{3.7}$$ {eq:simplex_matrix}

must be nonsingular, i.e. have a nonzero determinant. We can quantify how 'close' the simplex Y is to being singular, using the l^2 condition number of the matrix M, which in this context is usually referred to as the *simplex condition*:

$$\kappa(Y) = \|M\|\|M^{-1}\|, \tag{3.8}$$ {eq:simplex_condition}

where $\|M\|$ is the matrix norm induced by the Euclidean norm,

$$\|M\| = \max_{x\neq 0} \frac{\|Mx\|}{\|x\|}. \tag{3.9}$$ {eq:matrix_norm}

A singular simplex Y of course does not have a well-defined condition, since M^{-1} does not exist. However, the larger the condition of a simplex is, the closer it is to being singular. Very loosely speaking, a long and thin simplex has a large condition number, and would be singular if its width were to go to zero.

The simplex updates made in the process of the NM algorithm mean that the simplex condition changes throughout the course of the optimisation. If the simplex condition gets too large, it is possible that the optimisation will stall at a nonstationary point, since the search directions of the simplex are severely limited. The MDS algorithm was proposed partially for the purpose of avoiding this ill-conditioning.* The MDS method is also simplex-based, but instead of reflecting a single worst point y_{n+1} about the other points, it reflects all of the n worst points $\{y_2, y_3, \cdots, y_{n+1}\}$ about the best point y_1. This means that the shape of the simplex, and thus its condition number, is always preserved; from this, it could be concluded that at least one of the search directions was bounded away from being orthogonal to the gradient.[29] In simpler terms, at least one of the search directions is 'good enough'.

3.2.3 Implementation details

:poise__implementation

Having covered the user-facing details of POISE (which are also covered in substantial detail in the user documentation), I now spend some time discussing how POISE is implemented and several design choices.

Firstly, POISE is written in Python 3, and since TopSpin does not have a Python 3 interface,* this means that POISE is not entirely self-contained within TopSpin: in particular, an external installation of Python 3 is required, which may be a slight inconvenience. This choice was necessary because it would have been too time-consuming to implement numerical optimisation algorithms using the existing C or Python 2 APIs in TopSpin (notably, the Python 2 API uses the Jython implementation of the language, which is incompatible with `numpy`). An indirect benefit of this is that, since the 'cost' of installing Python 3 is already paid, we can also allow users to define their own cost functions using libraries such as `numpy` and `scipy` (without these it is very awkward to perform any kind of data processing).

*The main reason was in fact to better exploit computer parallelism, but it was also noticed that the MDS method proved to be generally more robust than NM.

*Version 4.1.4 of TopSpin now comes with a Python 3 API; however, this was introduced too late for the work in this chapter.

POISE is available on the Python Package Index (PyPI), so can be installed using a single command, `pip install nmrpoise`. Like all other Python packages, POISE is first installed to the Python `site-packages` directory. If the `nmrpoise` package is imported from a Python 3 script, then this code is read. This may be required on occasion, as the `nmrpoise` package provides a few functions to analyse optimisation logs created by POISE. We might refer to this code as the 'library' component of `nmrpoise`.

This, however, is irrelevant for actually *running* optimisations. When POISE is installed, on top of the default installation to `site-packages`, it automatically searches for TopSpin installations in either `C:\` (Windows), or `/opt/` (Unix/Linux). (If necessary, a non-standard TopSpin installation location can be specified using the `$TS` environment variable.) The installation then creates:

- a *frontend script* at `$TS/exp/stan/nmr/py/user/poise.py`, which allows POISE to be invoked by simply typing `poise` in the TopSpin command line and is responsible for controlling data acquisition; as well as
- a *backend directory* at `$TS/exp/stan/nmr/py/user/poise_backend`, within which all of the POISE data and logic is stored. For example, routines can be found in the `routines` subdirectory, and cost functions in the `costfunctions.py` and `costfunctions_user.py` files.

All optimisations are run using the code *only* in the backend directory, and not anything in Python's `site-packages` folder. This is because the frontend script must know how to launch the backend (i.e. where to find the files), and it is simply easiest to predefine this location.*

Having files in two different places does mean that some form of communication between the two must be established. In POISE, this is accomplished through the use of anonymous pipes, one for each direction of communication (listing 3.1). In this way, the backend can signal to the frontend what values of parameters should be evaluated; the frontend can then begin data acquisition, and signal to the backend when this is complete so that the cost function can be calculated. Although this setup works perfectly fine when left to run untouched, a frustrating number of 'tricks' are required to keep these synchronised if either the frontend or the backend are terminated unexpectedly, or if acquisition is prematurely stopped by the user (which usually suggests that they wish to stop the optimisation). This includes the backend creating a file with its process ID every time it is called and deleting it upon exit (listing 3.2), meaning that the frontend can locate any stray backend processes which were not appropriately terminated.

Finally, the frontend must also be careful not to overwrite data by triggering acquisition of other

*In fact, it is possible to dynamically determine the `site-packages` installation location at runtime, meaning that the entire backend does not need to be copied to TopSpin directories. However, that would mean the cost functions would be buried inside the `site-packages` directory, which can be difficult to find.

```
try:
    # Launch backend
    backend = subprocess.Popen([p_python3, "-u", p_backend],
                               stdin=subprocess.PIPE,
                               stdout=subprocess.PIPE)

    # Pass information from frontend to backend
    for item in [args.algorithm, routine_id, p_spectrum, args.maxfev]:
        print >>backend.stdin, item
    backend.stdin.flush()

    while True:
        # Receive information from backend
        line = backend.stdout.readline()
```

Listing 3.1: Excerpt from the POISE frontend script, illustrating the two-way communication between frontend and backend.

st:poise_communication

experiments: this can easily happen if, for example, a user opens a new dataset in TopSpin. To ensure that this is the case, the frontend *always* brings the optimisation dataset to the foreground immediately before acquisition is started. This has a slight drawback in that it can be difficult to view other spectra in TopSpin while an optimisation is proceeding. (Note, however, that this is out of my control: TopSpin does not give me any documented way of running an acquisition AU programme on a background dataset.)

3.3 What POISE is not

sec:poise__notpoise

Before moving on to cover applications of POISE, I want to make a note about several limitations of the approach chosen.

Firstly, *POISE is not specialised.* While generality is a strength in that POISE can be applied to a diverse range of NMR experiments, it can also be a weakness. POISE *always* follows the framework in fig. 3.1: in particular, it simply seeks to find the optimum $\mathbf{x}^*$, defined by

$$\arg\min_{\mathbf{x}} f(\mathbf{x}). \tag{3.10}$$

{eq:poise_argmin}

This rigidity in the underlying logic means that it is very conceivable that in specific instances, specialised optimisation routines which use customised strategies for data acquisition and analysis *can* outperform POISE in terms of speed and/or accuracy.

A related point is that on each function evaluation, the only bits of information retained are the parameters $\mathbf{x}$ and the value of the cost function $f(\mathbf{x})$. The spectral data itself is not stored

```
from contextlib import contextmanager

@contextmanager
def pidfile():
    # Create a file with the PID
    pid = os.getpid()
    pid_fname = Path(__file__).parent / f".pid{pid}"
    pid_fname.touch()
    # Run the code in the 'with' block
    try:
        yield
    # Delete the file after the 'with' block is exited
    finally:
        if pid_fname.exists():
            pid_fname.unlink()

if __name__ == "__main__":
    with pidfile() as _:
        main()
```

Listing 3.2: Simplified excerpt from POISE backend script, showing a context manager used to keep track of backend process IDs. The context manager ensures that when the script is started, a file with the process ID is created; and when the script exits, this file is deleted. The ‘main()‘ function carries out the actual optimisation.

lst:poise_backendpid

anywhere:[*] thus, it is not possible to perform (for example) an ‘optimisation’ which collects scans until a certain SNR is reached, or one which collects t_1 increments of a 2D spectrum and performs non-uniform sampling (NUS) processing until the signal to artefact ratio is sufficiently high. In particular, I want to distinguish POISE from other types of ‘optimisations’ reported in the literature, which typically *accumulate* data points until a given confidence level is reached (e.g. through a model-fitting procedure). Such procedures have been performed before in the contexts of (for example) relaxation measurements[34,35] and undersampling in multidimensional NMR.[36–38]

Secondly, *POISE is not a processing software.* Although POISE can in theory be used for optimisation of processing parameters (for example, by changing the ‘acquisition’ AU programme to perform only processing tasks), this usage is not considered here. Such optimisations can be more efficiently carried out on a personal computer, without requiring access to a spectrometer.

Finally, *POISE is not a panacea.* It should be noted that there is always an inherent tradeoff against the time required for the optimisation itself. For example, it makes little sense to spend several

[*]In principle, it *could* be. There is nothing stopping me from implementing something to store previous spectra; it was just not the original motivation behind POISE.

minutes optimising the sensitivity of a pulse–acquire experiment: the time could simply be used to improve the SNR by collecting more scans. There is also the critical—though undeniably subjective—question of whether the optimisation is *worth it*: even if better results can be obtained in relatively short times, does this provide a substantial benefit over a 'compromise' value in a default parameter set?* I do not profess to have a definitive answer to this, and I leave the reader to form their own conclusions in the specific contexts where they may consider using POISE. In any case, for practical use, it is imperative to make sure that the optimisation is either fast, or solves a problem which cannot simply be tackled through signal averaging in the same amount of time. It is my hope that this is (broadly) true of the examples shown.

3.4 Applications

ec:poise__applications

3.4.1 Pulse width calibration

subsec:poise__pulsecal

`popt` vs `pulsecal`

3.4.2 Ernst angle optimisation

subsec:poise__ernst

Maths

3.4.3 NOE mixing time

subsec:poise__noe

Stuff

3.4.4 ASAP-HSQC excitation delay

subsec:poise__asaphsqc

INEPT

3.4.5 HMBC low-pass J-filter

subsec:poise__hmbc

Artefacts.

3.4.6 Inversion–recovery

subsec:poise__invrec

Stuff

*Of course, even though it is nowadays fashionable for authors to imply that their publications possess *great impact*, a similar argument can be applied to *many* scientific discoveries. To use an example from the next chapter, is it really necessary to acquire NOAH spectra when one can just acquire the standalone 2D experiments? I have seen arguments on both sides—some people simply do not need the speedups provided and do not want to spend the time to set up or troubleshoot new experiments.

3.4.7 Ultrafast NMR

subsec:poise__epsi

EPSI gradient imbalance

3.4.8 PSYCHE pure shift NMR

subsec:poise__psyche

J-refocusing

3.4.9 Solvent suppression

subsec:poise__solvsupp

Mouse

3.4.10 Diffusion NMR

ubsec:poise__diffusion

Automated DOSY

Probably a good idea to revisit (and understand) Iain's comments.

3.5 POISE for ESR

sec:poise__esrpoise

Need JB to write this section

3.6 References

Yong2021AC (1) Yong, J. R. J.; Foroozandeh, M. On-the-Fly, Sample-Tailored Optimization of NMR Experiments. *Anal. Chem.* **2021**, *93*, 10735–10739, DOI: 10.1021/acs.analchem.1c01767.

Bardeen1997CPL (2) Bardeen, C. J.; Yakovlev, V. V.; Wilson, K. R.; Carpenter, S. D.; Weber, P. M.; Warren, W. S. Feedback quantum control of molecular electronic population transfer. *Chem. Phys. Lett.* **1997**, *280*, 151–158, DOI: 10.1016/S0009-2614(97)01081-6.

Schiano1999JMR (3) Schiano, J. L.; Routhier, T.; Blauch, A. J.; Ginsberg, M. D. Feedback Optimization of Pulse Width in the SORC Sequence. *J. Magn. Reson.* **1999**, *140*, 84–90, DOI: 10.1006/jmre.1999.1824.

Schiano2000ZNA (4) Schiano, J. L.; Blauch, A. J.; Ginsberg, M. D. Optimization of NQR Pulse Parameters using Feedback Control. *Zeitschrift für Naturforschung A* **2000**, *55*, 67–73, DOI: 10.1515/zna-2000-1-213.

Monea2020JMR (5) Monea, C. Optimization of NQR excitation sequences using black-box techniques. *J. Magn. Reson.* **2020**, *321*, 106858, DOI: 10.1016/j.jmr.2020.106858.

Goodwin2018JMR (6) Goodwin, D. L.; Myers, W. K.; Timmel, C. R.; Kuprov, I. Feedback control optimisation of ESR experiments. *J. Magn. Reson.* **2018**, *297*, 9–16, DOI: 10.1016/j.jmr.2018.09.009.

DePaepe2003CPL (7) De Paëpe, G.; Hodgkinson, P.; Emsley, L. Improved heteronuclear decoupling schemes for solid-state magic angle spinning NMR by direct spectral optimization. *Chem. Phys. Lett.* **2003**, *376*, 259–267, DOI: 10.1016/S0009-2614(03)00966-7.

Elena2004CPL (8) Elena, B.; de Paëpe, G.; Emsley, L. Direct spectral optimisation of proton–proton homonuclear dipolar decoupling in solid-state NMR. *Chem. Phys. Lett.* **2004**, *398*, 532–538, DOI: 10.1016/j.cplett.2004.09.122.

Salager2010CPL (9) Salager, E.; Dumez, J.-N.; Stein, R. S.; Steuernagel, S.; Lesage, A.; Elena-Herrmann, B.; Emsley, L. Homonuclear dipolar decoupling with very large scaling factors for high-resolution ultrafast magic angle spinning ^{1}H solid-state NMR spectroscopy. *Chem. Phys. Lett.* **2010**, *498*, 214–220, DOI: 10.1016/j.cplett.2010.08.038.

Skinner2003JMR (10) Skinner, T. E.; Reiss, T. O.; Luy, B.; Khaneja, N.; Glaser, S. J. Application of optimal control theory to the design of broadband excitation pulses for high-resolution NMR. *J. Magn. Reson.* **2003**, *163*, 8–15, DOI: 10.1016/s1090-7807(03)00153-8.

Khaneja2005JMR (11) Khaneja, N.; Reiss, T.; Kehlet, C.; Schulte-Herbrüggen, T.; Glaser, S. J. Optimal control of coupled spin dynamics: design of NMR pulse sequences by gradient ascent algorithms. *J. Magn. Reson.* **2005**, *172*, 296–305, DOI: 10.1016/j.jmr.2004.11.004.

Kobzar2008JMR (12) Kobzar, K.; Skinner, T. E.; Khaneja, N.; Glaser, S. J.; Luy, B. Exploring the limits of broadband excitation and inversion: II. Rf-power optimized pulses. *J. Magn. Reson.* **2008**, *194*, 58–66, DOI: 10.1016/j.jmr.2008.05.023.

Kobzar2012JMR (13) Kobzar, K.; Ehni, S.; Skinner, T. E.; Glaser, S. J.; Luy, B. Exploring the limits of broadband 90° and 180° universal rotation pulses. *J. Magn. Reson.* **2012**, *225*, 142–160, DOI: 10.1016/j.jmr.2012.09.013.

Schilling2014ACIE (14) Schilling, F.; Warner, L. R.; Gershenzon, N. I.; Skinner, T. E.; Sattler, M.; Glaser, S. J. Next-Generation Heteronuclear Decoupling for High-Field Biomolecular NMR Spectroscopy. *Angew. Chem., Int. Ed.* **2014**, *53*, 4475–4479, DOI: 10.1002/anie.201400178.

Glaser2015EPJD (15) Glaser, S. J.; Boscain, U.; Calarco, T.; Koch, C. P.; Köckenberger, W.; Kosloff, R.; Kuprov, I.; Luy, B.; Schirmer, S.; Schulte-Herbrüggen, T.; Sugny, D.; Wilhelm, F. K. Training Schrödinger's cat: quantum optimal control. *Eur. Phys. J. D* **2015**, *69*, No. 279, DOI: 10.1140/epjd/e2015-60464-1.

Geen1989JMR (16) Geen, H.; Wimperis, S.; Freeman, R. Band-selective pulses without phase distortion. A simulated annealing approach. *J. Magn. Reson.* **1989**, *85*, 620–627, DOI: 10.1016/0022-2364(89)90254-0.

Emsley1990CPL (17) Emsley, L.; Bodenhausen, G. Gaussian pulse cascades: New analytical functions for rectangular selective inversion and in-phase excitation in NMR. *Chem. Phys. Lett.* **1990**, *165*, 469–476, DOI: 10.1016/0009-2614(90)87025-m.

Geen1991JMR (18) Geen, H.; Freeman, R. Band-selective radiofrequency pulses. *J. Magn. Reson.* **1991**, *93*, 93–141, DOI: 10.1016/0022-2364(91)90034-q.

Nuzillard1994JMRSA (19) Nuzillard, J. M.; Freeman, R. Band-Selective Pulses Designed to Accommodate Relaxation. *J. Magn. Reson., Ser. A* **1994**, *107*, 113–118, DOI: 10.1006/jmra.1994.1056.

Kupce1995JMRSA (20) Kupce, E.; Freeman, R. Band-Selective Correlation Spectroscopy. *J. Magn. Reson., Ser. A* **1995**, *112*, 134–137, DOI: 10.1006/jmra.1995.1023.

Kupce1995JMRSB (21) Kupce, E.; Boyd, J.; Campbell, I. D. Short Selective Pulses for Biochemical Applications. *J. Magn. Reson., Ser. B* **1995**, *106*, 300–303, DOI: 10.1006/jmrb.1995.1049.

Sakellariou2000CPL (22) Sakellariou, D.; Lesage, A.; Hodgkinson, P.; Emsley, L. Homonuclear dipolar decoupling in solid-state NMR using continuous phase modulation. *Chem. Phys. Lett.* **2000**, *319*, 253–260, DOI: 10.1016/s0009-2614(00)00127-5.

Shaka1985JMR (23) Shaka, A. J.; Barker, P. B.; Freeman, R. Computer-optimized decoupling scheme for wideband applications and low-level operation. *J. Magn. Reson.* **1985**, *64*, 547–552, DOI: 10.1016/0022-2364(85)90122-2.

Freeman1987JMR (24) Freeman, R.; Xili, W. Design of magnetic resonance experiments by genetic evolution. *J. Magn. Reson.* **1987**, *75*, 184–189, DOI: 10.1016/0022-2364(87)90331-3.

Bechmann2013JMR (25) Bechmann, M.; Clark, J.; Sebald, A. Genetic algorithms and solid state NMR pulse sequences. *J. Magn. Reson.* **2013**, *228*, 66–75, DOI: 10.1016/j.jmr.2012.12.015.

Ehni2014JMR (26) Ehni, S.; Luy, B. Robust INEPT and refocused INEPT transfer with compensation of a wide range of couplings, offsets, and B 1 -field inhomogeneities (COB3). *J. Magn. Reson.* **2014**, *247*, 111–117, DOI: 10.1016/j.jmr.2014.07.010.

Lapin2020JMR (27) Lapin, J.; Nevzorov, A. A. De novo NMR pulse sequence design using Monte-Carlo optimization techniques. *J. Magn. Reson.* **2020**, *310*, 106641, DOI: 10.1016/j.jmr.2019.106641.

Nelder1965TCJ (28) Nelder, J. A.; Mead, R. A Simplex Method for Function Minimization. *The Computer Journal* **1965**, *7*, 308–313, DOI: 10.1093/comjnl/7.4.308.

Torczon1989 (29) Torczon, V. J. Multidirectional search: A direct search algorithm for parallel machines, Ph.D. Rice University, 1989.

DennisJr1991SIAMJO (30) Dennis Jr., J. E.; Torczon, V. Direct Search Methods on Parallel Machines. *SIAM J. Optim.* **1991**, *1*, 448–474, DOI: 10.1137/0801027.

Powell2009Proc (31) Powell, M. J. D. *The BOBYQA algorithm for bound constrained optimization without derivatives*; tech. rep. DAMTP 2009/NA06; University of Cambridge, 2009.

Cartis2019ACMTMS (32) Cartis, C.; Fiala, J.; Marteau, B.; Roberts, L. Improving the Flexibility and Robustness of Model-based Derivative-free Optimization Solvers. *ACM Trans. Math. Softw.* **2019**, *45*, 1–41, DOI: 10.1145/3338517.

Nocedal2006 (33) Nocedal, J.; Wright, S. J., *Numerical Optimization*, 2nd ed.; Springer: New York, 2006.

Song2018JMR (34) Song, Y.-Q.; Tang, Y.; Hürlimann, M. D.; Cory, D. G. Real-time optimization of nuclear magnetic resonance experiments. *J. Magn. Reson.* **2018**, *289*, 72–78, DOI: 10.1016/j.jmr.2018.02.009.

Tang2019SR (35) Tang, Y.; Song, Y.-Q. Realtime optimization of multidimensional NMR spectroscopy on embedded sensing devices. *Sci. Rep.* **2019**, *9*, DOI: 10.1038/s41598-019-53929-1.

Eghbalnia2005JACS (36) Eghbalnia, H. R.; Bahrami, A.; Tonelli, M.; Hallenga, K.; Markley, J. L. High-Resolution Iterative Frequency Identification for NMR as a General Strategy for Multidimensional Data Collection. *J. Am. Chem. Soc.* **2005**, *127*, 12528–12536, DOI: 10.1021/ja052120i.

Hansen2016ACIE (37) Hansen, A. L.; Brüschweiler, R. Absolute Minimal Sampling in High-Dimensional NMR Spectroscopy. *Angew. Chem. Int. Ed.* **2016**, *55*, 14169–14172, DOI: 10.1002/anie.201608048.

BrukerSmartDriveNMR (38) Corporation, B. Advanced Acquisition Software Application | NMR Software https://www.bruker.com/en/products-and-solutions/mr/nmr-software/smartdrivenmr.html (accessed 22/08/2022).

refsection:4

refsection:5