

Optimising NMR Spectroscopy through Method and Software Development

Jonathan Yong

University of Oxford

Contents

Abstract	v
Acknowledgements	vi
Preface	vii
List of figures	xi
List of tables	xiii
List of code listings	xiv
1 NMR theory	1
1.1 Quantum mechanics	2
1.2 The rotating frame	5
1.3 Density operators	8
1.4 Pulse sequences	11
1.4.1 1D pulse-acquire	11
1.4.2 INEPT and product operators	15
1.4.3 2D NMR: general principles	19
1.4.4 The States HSQC experiment	23
1.4.5 The echo-antiecho HSQC: gradients and coherence selection	24
1.5 References	31
2 Pure shift NMR	34
2.1 Theoretical background	35
2.2 Pure shift in practice	39
2.2.1 Acquisition modes	39
2.2.2 Pure shift elements	41
2.2.3 PSYCHE in detail	43

2.3	PSYCHE with a variable number of saltires	47
2.4	Direct optimisation of PSYCHE waveform	50
2.4.1	Techniques for pure shift optimisations	51
2.4.2	Flip angle optimisation	55
2.4.3	Waveform parameterisation and optimisation	57
2.5	Time-reversal method	61
2.6	‘Discrete PSYCHE’	65
2.6.1	Speeding up dPSYCHE simulations	66
2.6.2	Optimisations and experimental evaluation	70
2.7	Ultrafast PSYCHE-iDOSY	77
2.8	References	83
3	POISE	93
3.1	Introduction	94
3.2	Technical overview	96
3.2.1	Routines	96
3.2.2	Optimisation settings	98
3.2.3	Optimisation algorithms	98
3.2.4	Implementation details	104
3.3	What POISE is not	106
3.4	Applications	108
3.4.1	Pulse width calibration	109
3.4.2	Ernst angle optimisation	114
3.4.3	Inversion–recovery	117
3.4.4	NOE mixing time	119
3.4.5	ASAP-HSQC excitation delay	122
3.4.6	Ultrafast NMR	125
3.4.7	HMBC low-pass J-filter	127
3.4.8	PSYCHE pure shift NMR	127
3.4.9	Solvent suppression	127
3.4.10	Diffusion NMR	127
3.5	POISE for ESR	127
3.6	References	127
4	NOAH	130
4.1	Introduction	131
4.2	Sensitivity analysis of NOAH supersequences	131
4.3	GENESIS: automated pulse programme creation	131

4.4	Discussion of individual modules	132
4.4.1	Sensitivity-enhanced HSQC	132
4.4.2	HSQC-TOCSY	132
4.4.3	HSQC-COSY	132
4.4.4	2DJ and PSYCHE	132
4.4.5	DQF-COSY	132
4.4.6	HMQC	132
4.4.7	HMBC	132
4.4.8	ADEQUATE	133
4.5	Solvent suppression in NOAH	133
4.6	NOAH with short recovery ;delays (???)	133
4.7	Parallel and generalised NOAH supersequences	133
4.8	References	133
A	Other work	135
A.1	NMR plotting in Python	135
A.2	Citation management	136
A.3	Group website and pulse programming tutorials	136
A.4	References	136

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](https://doi.org/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 ESR⁶ spectroscopies, the only direct parallel in NMR which I have found is that of the eDUMBO pulses for heteronuclear^{7,8} and homonuclear dipolar⁹ 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*. Each *function evaluation* (FE), i.e. each measurement of the cost function, corresponds to the acquisition of an NMR spectrum which may take seconds to minutes. In most cases, it is probably easier to run NMR optimisations in a theoretical manner, which can be much faster and also circumvents the effect of noise. Examples of such optimisations include the design of shaped pulses through optimal control theory,^{10–16} by simple parameterisation,^{17–22} or the optimisation of entire pulse sequences^{23–27} (this is essentially what I did with the dPSYCHE experiment in § 2.6). (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.²⁸)

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=QT CeSCRZs4I>).

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

In this section, I first cover the general principles underlying, and the implementation of, POISE. The basic operation of POISE is summarised in fig. 3.1, which is essentially a generalised version of the pure shift optimisations carried out in § 2.4.



./figures/poise/flowchart.png

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

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 § 3.2.4. 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

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 § 3.4.4) 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 FEs, 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 FE 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. I now describe these algorithms in greater detail.

3.2.3 Optimisation algorithms

As was briefly mentioned in § 2.4.1, derivative-based optimisation algorithms cannot be used for experimental optimisations. To be more precise, when analytical gradients are not available, derivative-based algorithms calculate gradients using a finite difference approximation:

$$\nabla f(x) \approx \frac{f(x + \varepsilon) - f(x - \varepsilon)}{2\varepsilon}, \quad (3.1) \quad \text{\small {eq:finite_difference_}}$$

where ε is the step size used for the finite difference calculation. In Nocedal and Wright,²⁹ it is shown that the error in this finite difference gradient (as compared to the true gradient) has an upper bound of

$$\frac{\eta(x; \varepsilon)}{\varepsilon} + O(\varepsilon^2), \quad (3.2) \quad \text{\texttt{\{eq:finite_difference\}}}$$

where $\eta(x; \varepsilon)$ 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 (which is the error due to the finite difference approximation) is large. This means that finite difference gradients, and any algorithms which use them, are entirely unreliable in the presence of (sufficient) noise. Instead, POISE provides a choice of three derivative-free optimisation algorithms: the Nelder–Mead (NM) method,³⁰ the multidirectional search (MDS) method,^{31,32} and the Py-BOBYQA trust-region method.^{33,34}

Nelder–Mead

The NM method is a highly popular derivative-free optimisation algorithm, which maintains a set of points $\{y_1, y_2, \dots, y_{n+1}\}$ during the optimisation, where n is the number of parameters being optimised. The convex hull of these points, Y , is the smallest possible set of points containing all the y_k such that

$$\forall x_1, x_2 \in Y, \forall \alpha \in [0, 1], \alpha x_1 + (1 - \alpha)x_2 \in Y, \quad (3.3) \quad \text{\texttt{\{eq:convex_hull\}}}$$

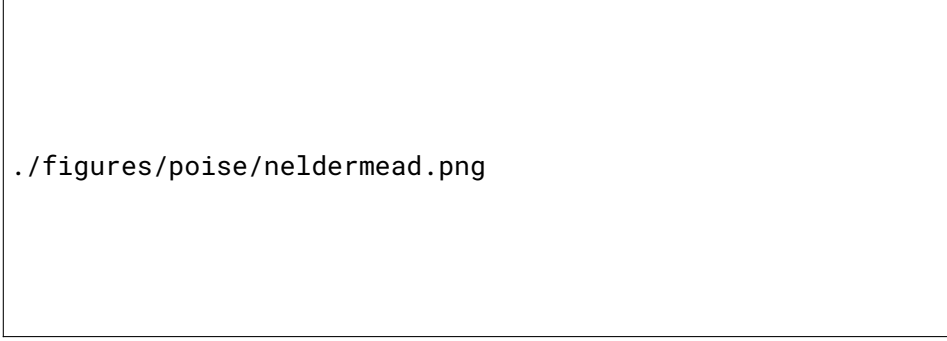
and is called a *simplex*. To provide an analogy for $n = 2$, the convex hull is the shape obtained by stretching a rubber band around three pins placed at y_1, y_2, y_3 . If this convex hull is nonempty—or equivalently, if the n vectors $y_k - y_1$ ($2 \leq k \leq n + 1$) are linearly independent—then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

The NM algorithm is in fact quite intuitive to understand. The initial simplex is first constructed using the supplied initial point: POISE specifically uses the method of Spendley et al.³⁵ The optimisation itself 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 $f(y_1) \leq f(y_2) \leq \dots \leq f(y_{n+1})$. The centroid of the simplex is defined by the best n points,

$$\bar{y} = \sum_{i=1}^n y_i. \quad (3.4) \quad \text{\texttt{\{eq:simplex_centroid\}}}$$

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:

$$y_r = \bar{y} - (y_{n+1} - \bar{y}). \quad (3.5) \quad \text{\texttt{\{eq:nm_reflect\}}}$$



./figures/poise/neldermead.png

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 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 $f(y_1) \leq f(y_r) < f(y_n)$, 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. $f(y_r) < f(y_1)$), then we ambitiously attempt to *expand* the simplex even further in that direction:

$$y_e = \bar{y} - 2(y_{n+1} - \bar{y}). \quad (3.6) \quad \{\text{eq:nm_expand}\}$$

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 $f(y_n) \leq f(y_r) < f(y_{n+1})$, then the algorithm performs an *outside contraction*, which resembles a half-hearted reflection:

$$y_{oc} = \bar{y} - (1/2)(y_{n+1} - \bar{y}). \quad (3.7) \quad \{\text{eq:nm_outside_contrac}\}$$

Conversely, if the reflected point is even worse than the worst point ($f(y_{n+1}) \leq f(y_r)$), 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:

$$y_{oc} = \bar{y} + (1/2)(y_{n+1} - \bar{y}). \quad (3.8) \quad \{\text{eq:nm_inside_contract}\}$$

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.

Finally, convergence is signalled when for each dimension of the optimisation, the width of the simplex is smaller than the chosen optimisation tolerance.* For a multiple-parameter optimisation, this can potentially mean that extra accuracy is obtained in one of the parameters (because the simplex may have shrunk along that dimension more quickly). However, it does guarantee that *at least* the specified level of accuracy in every dimension is achieved.

Multidirectional search

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

$$M = (y_2 - y_1, y_3 - y_1, \dots, y_{n+1} - y_1) \quad (3.9) \quad \text{\small \{eq:simplex_matrix\}}$$

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*:

$$\kappa(Y) = \|M\| \|M^{-1}\|, \quad (3.10) \quad \text{\small \{eq:simplex_condition\}}$$

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

$$\|M\| = \max_{x \neq 0} \frac{\|Mx\|}{\|x\|}. \quad (3.11) \quad \text{\small \{eq:matrix_norm\}}$$

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. This is good for achieving decreases in the cost function, since the simplex shape *adapts* to the cost function being optimised. However, if the simplex condition gets too large, it is possible that the optimisation will stall at a nonstationary

*The implementation of the NM algorithm in the `scipy` library only accepts a single value for the ‘tolerance’, which is then used in all dimensions. This is designed to be used by scaling the parameters beforehand such that the tolerance in each dimension is equal (and in fact, POISE was later updated to do so). However, during initial development I chose instead to re-implement the NM algorithm with a convergence check which allowed for different tolerances to be specified in each dimension.

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, and uses similar reflection/expansion/contraction steps as NM. However, instead of (e.g.) reflecting a single worst point y_{n+1} about the other points, it reflects all of the n worst points $\{y_2, y_3, \dots, y_{n+1}\}$ about the best point y_1 . This means that the shape of the simplex, and thus its condition number, is always preserved, which provides it with much better convergence properties.^{31,36†}

The increased reliability of the MDS algorithm over the NM algorithm was demonstrated on a variety of example optimisation problems: even in the very simple case where the cost function was simply the norm of a vector,

$$f(y) = \|y\|, \quad (3.12) \quad \text{\small \{eq:norm_cf\}}$$

it was shown that the NM algorithm stalled when the dimension of the problem, n , was sufficiently large. The value of n needed to precipitate this failure depended on the problem being solved, and generally ranged from 8 to 40. On the other hand, the MDS method proved to be robust under the same conditions, eventually converging to the optimum—although in the cases where NM *did* work, the MDS method generally required more FEs.

It was this improved robustness of the MDS algorithm which prompted Goodwin et al.⁶ to use it in their (experimental) optimisation of ESR pulse shapes, and for me to later include it in POISE. In the ESR work, the number of pulse points being optimised was 11 or 21, which fell into the regime where the MDS method would likely have better convergence properties than NM. However, optimisations of this scale are feasible in ESR only because of the rapid relaxation and thus short experiment repetition times. In NMR, each experiment takes a substantially longer time, and even optimisations with $n > 2$ become rather time-consuming due to the number of FEs required (the largest n explored in the present work is 4). As will be shown later, we found that the NM and MDS methods were equally reliable in our optimisations, with NM generally being faster.

Py-BOBYQA

Unlike the NM and MDS algorithm, Py-BOBYQA is not simplex-based, but is a trust-region algorithm.^{33,34} The fundamental idea behind a (derivative-free) trust-region method is to sample the cost function at a set of points Y , and construct a model m through interpolation, which

^{*}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.

[†]Specifically, it can be concluded that at least one of the search directions was bounded away from being orthogonal to the gradient; or in simpler (and less precise) terms, at least one of the search directions is close enough to a direction in which the cost function f decreases.

matches the cost function at these points:

$$\forall y \in Y, m(y) = f(y). \quad (3.13) \quad \text{\small \{eq:trust_region_model\}}$$

The model at iteration k is labelled m_k . Most trust region methods use a quadratic model, and Py-BOBYQA is no exception. This can be expressed as:

$$m_k(x_k + p) = c + g^T p + p^T G p, \quad (3.14) \quad \text{\small \{eq:trust_region_quadr\}}$$

where G is a symmetric matrix and x_k is the centre of the model at iteration k (x_0 being the user-specified initial point). For this model to be fully determined, the set Y must therefore contain $(n + 1)(n + 2)/2$ points in total.*

The algorithm maintains a *trust region radius* Δ_k at each iteration, which is a measure of how reliable the model is. The initial trust region radius, Δ_0 , can be arbitrarily chosen: in the case of POISE, I elected to set Δ_0 to be 10 times the desired tolerance. The model m_k is then used to calculate the next step s_k , which is obtained by minimising m_k over all points within a radius of Δ_k from the centre x_k (the *trust region subproblem*):

$$s_k = \arg \min_{\|s\| \leq \Delta_k} m_k(x_k + s). \quad (3.15) \quad \text{\small \{eq:trust_region_subpr\}}$$

Since m_k is noiseless, this can be done with almost any algorithm: Py-BOBYQA uses a conjugate gradient method. The (true) cost function is then evaluated at the trial point $x_k + s_k$, and compared against the value predicted by the model. If the ratio of ‘actual improvement’ to ‘predicted improvement’ is large enough, i.e.

$$r_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)} \geq \eta \quad (3.16) \quad \text{\small \{eq:trust_region_thres\}}$$

for some threshold value η , then the step s_k is accepted and x_{k+1} is set to $x_k + s_k$, replacing the worst point in Y . Additionally, the trust region radius Δ_k may be increased so that the next step(s) can be more ambitious. Conversely, if $r_k < \eta$, then there are one of two possibilities: either the model is poorly conditioned (in that the points in Y are very unevenly distributed), in which case one of the points is replaced and the model recalculated; or the model is sufficiently well-conditioned, in which case the step is rejected, and Δ_k is decreased.

Py-BOBYQA goes beyond a standard derivative-free trust-region algorithm in further limiting the rate at which the radius Δ_k can change (amongst others). Separately from Δ_k , Py-BOBYQA also maintains a lower bound on the trust region radius ρ_k , and on unsuccessful iterations Δ_k

*In a derivative-based trust region method, g and G are determined using information from the gradient and/or Hessian.

is not allowed to decrease further than ρ_k . This prevents Δ_k from decreasing too quickly until the algorithm is certain that Y is sufficiently well-conditioned.³⁷ Another critical feature of Py-BOBYQA is the implementation of multiple restarts, which endows it with greater robustness towards noise and also allows it to escape local minima.^{34,38} However, the multiple-restarts feature in Py-BOBYQA was disabled in POISE as this often led to overly long optimisations.*

Crucially, Py-BOBYQA differs from the simplex-based methods in that *it cares about the actual value of the cost function*. In the NM and MDS methods, only the relative ordering of the points in the simplex matters; it makes no difference to the algorithm whether the worst point has a cost function value of 10 or 1000. However, in Py-BOBYQA, the value of f is used in constructing the model, and thus directly influences the optimisation trajectory. Although this is beneficial in cases where the underlying cost function is relatively well-behaved (this *probably* means cases where the cost function is well described by a quadratic model[†]), and is reflected in faster convergence rates, it can be problematic for some cost functions. Py-BOBYQA is set as the default optimiser in POISE, but the user is strongly recommended to try the NM method as a first step when troubleshooting failed optimisations.

3.2.4 Implementation details

In this subsection, I discuss some behind-the-scenes details about how POISE is implemented and several design choices. This information is relevant for anybody looking to improve or otherwise modify the POISE codebase.

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).

POISE is available on the Python Package Index (PyPI), so can be installed using a single

*Most mathematics papers on optimisation have no qualms in using hundreds or even thousands of FEs, and it is this context in which Py-BOBYQA outperforms other algorithms. Unfortunately for me, POISE works in an *extremely* restrictive regime where even 50 FEs would be considered very expensive.

[†]Of course, because of Taylor’s theorem, every non-noisy cost function can be locally described by a quadratic model within a sufficiently small region. However, for meaningful progress to be made with noisy cost functions, the model must be built over a large enough region such that noise becomes less relevant.

[‡]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.

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 experiments: this can easily happen if, for example, a user opens a new dataset in TopSpin. To

*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.

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.) There is one other quirk of TopSpin surrounding data acquisition: it is possible to start the acquisition from a Python script (such as the frontend `poise.py` script), but it is not possible to block execution of the Python script while acquisition is running. Thus, it is not possible to trigger acquisition and wait until it is done before sending a signal to the backend.* The workaround is to call an AU programme containing acquisition commands, which (somehow) blocks the Python script.

3.3 What POISE is not

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

$$\arg \min_{\mathbf{x}} f(\mathbf{x}). \quad (3.17) \quad \text{\small \{eq:poise_argmin\}}$$

*The TopSpin Python documentation claims that this *can* be accomplished using, for example `XCMD("zg", wait=WAIT_TILL_DONE)`. However, none of the suggestions in the documentation worked as intended.

```

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.

1st:poise_backendpid

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. For example, we see this in § 3.4.1: the TopSpin `pulsecal` routine for pulse width calibration can be much faster than POISE, because it only needs to perform one experiment to obtain an answer.

A related point is that on each FE, 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 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^{39,40} and undersampling in multidimensional NMR.^{41–43}

^{*}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.

Secondly, *POISE is not a global optimiser*. The optimisation algorithms provided within POISE are not designed to search for global minima (except for Py-BOBYQA, but as described in § 3.2.3, I disabled the multiple restarts option responsible for this). In challenging optimisation cases where multiple local minima exist, it is not generally possible to predict which local minimum the algorithm will converge to. What *can* be guaranteed is that if the initial point is not already an optimum, then the optimisation will always provide a decrease in the cost function: in other words, it will always lead to an improvement in the spectrum (insofar as the cost function accurately represents the quality of the spectrum).

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 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

In this section, I cover a number of scenarios in which POISE can be used. These are generally ordered from simple to complex, and progressively show how the features in POISE can be used to customise optimisation procedures.

All POISE optimisations run in this chapter were performed five times to check for potential reproducibility issues. Due to noise in the cost function, these optimisations are not deterministic, and the optima obtained typically span a range. Where possible, this range is quoted in all the results shown in this chapter.

*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.

3.4.1 Pulse width calibration

The first of these applications is the calibration of a 90° ^1H pulse, which is applicable to virtually every NMR experiment. Essentially, we seek to determine τ_p for which

$$\tau_p \omega_1 = \frac{\pi}{2}, \quad (3.18) \quad \{\text{eq:pw90}\}$$

where the RF amplitude ω_1 is not known *a priori* (it is only indirectly controlled via the power level). This pulse width is conventionally specified as the P1 parameter in TopSpin.

Optimisation setup

In theory, performing a pulse-acquire spectrum with a perfect 180° or 360° pulse would yield no detectable (i.e. transverse) magnetisation, i.e. a *null*. Generally, the 360° null is preferred as it minimises effects due to radiation damping, and also allows a smaller recovery delay to be used. We can use POISE to search for this by acquiring the spectrum, performing a magnitude-mode calculation, and using the intensity of the resulting spectrum as a cost function:


$$f_{\text{minabsint}} = \sum_i |S_i|, \quad (3.19) \quad \{\text{eq:minabsint}\}$$

where (reusing notation from § 2.4.1) S is the spectrum under consideration represented as a complex-valued vector, and the i -th point of the spectrum $S_i = \sqrt{S_{\text{re},i}^2 + S_{\text{im},i}^2}$. The label `minabsint` makes reference to the fact that this cost function drives the optimisation to *minimise* the *absolute intensity* of the spectrum. An implementation of this is shown in listing 3.3 (for all other cost functions in this chapter, the reader is directed to the POISE source code for their implementations).

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

Listing 3.3: The implementation of the `minabsint` cost function in POISE.

To check whether this cost function is sensible, I manually acquired a series of spectra with increasing pulse widths and calculated $f_{\text{minabsint}}$ for all of these (fig. 3.3). In this thesis, I will refer to this process as a *reference grid search*. It should be noted that reference grid searches are a time-consuming procedure, and an end-user of POISE generally does *not* need to do this: I only do it here to provide some insight into the nature of the optimisation. In any case, it is



./figures/poise/p1_scan.png

fig:p1_scan

Figure 3.3: Reference grid search, showing how the minabsint cost function varies with the pulse width P1. Data code: 6F-200826.

clear that there is a well-defined minimum, located in this case at 48.3 μs ; we expect that POISE routines will converge to this point.

To complete the description of the POISE optimisation, it remains to define the rest of the optimisation routine. I chose the initial point to be four times the `proso1` value for P1: this represents our ‘best guess’ and is derived from prior calibration of the pulse width on a standard sample. The tolerance is set to 0.2 μs , which corresponds to an accuracy of 0.05 μs for the 90° pulse width itself. The lower and upper bounds are set to be 8 μs away from the initial point, representing a ‘sensible’ region within which we expect the null to lie (this may need to be adjusted for samples with high ionic strength, but for typical organic samples this is more than enough). A standard `poise_1d` acquisition AU programme is used, which simply acquires the spectrum and performs Fourier transformation, phase correction, and baseline correction. The routine in JSON format is shown in the caption to table 3.1. Finally, in order to reduce the time taken for the optimisation, several tricks are used: each FE is run using no dummy scans and only one scan, the acquisition time is set to just 1.1 s (high resolution is not required for a reliable cost function value), and the recovery delay D1 is set to 0. In practice, there is an extra gap of ~ 5 s between successive FEs due to spectrometer initialisation, so an extra recovery delay is not needed.

The competition

The performance of POISE can be compared against two ‘competitors’ in this area. The traditional method of determining the 90° pulse width is to measure a pulse width array (colloquially, ‘to array the pulse width’).⁴⁴ This entails measuring a series of pulse–acquire spectra, over which τ_p is evenly incremented: in optimisation parlance this would be called a *grid search*. This leads

to a sinusoidal pattern in the peak intensities, from which the 360° null can be directly read off. An example of this is shown in fig. 3.4, where the 360° null at $\tau_p \approx 48 \mu\text{s}$ is visible (it is never a *perfect* null because of off-resonance effects and ω_1 , i.e. B_1 , inhomogeneity).

./figures/poise/p1_scan_spectra.png

Figure 3.4: An example of a pulse width array, where the variation of the residual water peak is monitored with changes in pulse width. These spectra were acquired manually; the TopSpin `popt` command would yield essentially identical results. *Data code:* 6F-200826.

TopSpin provides a built-in mechanism for measuring a pulse width array using the `popt` command. The pulse-acquire spectrum is measured for pulse widths between a lower and upper bound, and the user specifies a spectral region of interest which `popt` uses to determine the null in spectrum intensity.* While this usually yields highly accurate results, the acquisition of so many spectra is relatively time-consuming and arguably unnecessary if the only purpose is to determine the null.

A more rapid method for pulse calibration is the nutation experiment of Wu and Otting,⁴⁵ which allows the 90° pulse width to be determined in a single-scan experiment. In this experiment, an RF pulse with a given power level, corresponding to an (unknown) amplitude of ω'_1 , is applied during acquisition.[†] Assuming that the pulse is applied along the x -axis, this leads to the following product operators during the acquisition period:

$$I_z \xrightarrow{\omega'_1 t_2} I_z \cos(\omega'_1 t_2) - I_y \sin(\omega'_1 t_2) \quad (3.20) \quad \{\text{eq:pulsecal_operators}\}$$

*In fact, `popt` uses the notion of a cost function as well, in that it determines the point where the cost function is minimised. In this case, I set it to use the MAGMIN cost function, which seeks to minimise the intensity of the magnitude-mode spectrum; this is essentially identical to the `minabsint` cost function which was used for the POISE optimisations except that it only applies to the spectral region of interest.

[†]To be precise, it is applied for a proportion d of the dwell time between acquisition of successive points in the FID; d is called the *duty cycle* and must be accounted for when calculating the pulse width using this method.

and an FID of

$$s(t_2) = -\sin(\omega'_1 t_2) = -\frac{1}{2i}(\exp(i\omega'_1 t_2) - \exp(-i\omega'_1 t_2)), \quad (3.21) \quad \text{(eq:pulsecal_signal)}$$

which when Fourier transformed yields an antiphase doublet where the two peaks are separated by the frequency $2\omega'_1$. Measuring the separation between the two peaks directly yields the unknown amplitude ω'_1 , from which $\tau'_p = \pi/(2\omega'_1)$ can be calculated. Typically, the RF amplitude ω'_1 is rather smaller than the amplitude ω_1 which we would like to apply the hard pulse at, and thus $\tau'_p > \tau_p$. However, this can be adjusted for using the power levels applied (which are known to the user).

Although the nutation experiment can be performed extremely quickly using the TopSpin `pulsecal` command, often only requiring a few seconds, the pulse widths calculated are generally slightly shorter compared to the value obtained from a 360° null. This is a known effect which arises because the separation is calculated from the top of the peaks, which correspond to the most homogeneous region of the B_1 profile.⁴⁵ On the other hand, the 360° null (as measured through `popt` or POISE, for example) measures a signal which is averaged over the entire B_1 profile.

Entry	Method	Optimum found (μs)	FEs	Time taken (s)
1	<code>popt</code>	48.40	41	299
2	<code>pulsecal</code>	46.64	–	37
3	POISE (NM)	48.38	10	76–79
4	POISE (MDS)	48.38	10	77–80
5	POISE (BOBYQA)	48.29–48.41	6–7	46–54

tbl:pulsecal_48

Table 3.1: Comparison of methods for 360° pulse width determination. `popt` grid searches were run between values of $40\mu\text{s}$ and $56\mu\text{s}$, with a linear increment of $0.4\mu\text{s}$ (which, through interpolation, provides a precision of approximately $0.2\mu\text{s}$ in the result, matching the tolerance used for POISE). `pulsecal` was run as normal and the reported pulse width multiplied by 4 to obtain the 360° pulse width for comparison. POISE optimisations were run according to the routine in table 3.1. *Data code:* 6F-200826.

Optimisation results

Compared to these two existing methods, we expect POISE to be faster than the `popt` grid search, and also more accurate than the nutation experiment in `pulsecal`. This is borne out in practice (table 3.1). `popt` yields an optimum of $48.4\mu\text{s}$, which is closely matched by POISE. However, POISE locates this optimum using far fewer FEs because its algorithms are more efficient than a simple grid search. While `pulsecal` is even faster than POISE, it underestimates the 90° pulse width by about 4%. In this particular case, POISE is the only option which strikes a useful balance between speed and accuracy. These results also provide the first evidence that Py-BOBYQA is

generally faster than the simplex-based methods: this observation is faithfully reproduced in the other optimisations in this chapter.

Different initial points

One question we might reasonably ask is how robust POISE is towards poor initial guesses. In the case of the pulse width calibration, the answer is: *very* robust. Tables 3.2 and 3.3 summarise the results obtained with an initial guess of 43 μs and 53 μs respectively. There is slightly decreased performance in that a few more FEs are required for convergence, but the accuracy of the result is unchanged.

Entry	Method	Optimum found (μs)	FEs	Time taken (s)
1	POISE (NM)	48.38	14	109–114
2	POISE (MDS)	48.38	14	108–112
3	POISE (BOBYQA)	48.27–48.33	9	70

tbl:poisecal_43

Table 3.2: Pulse width optimisations with an initial point of 43 μs . The POISE routine is the same as in table 3.1, except with "init":[43.0]. *Data code:* 6F-200826.

Entry	Method	Optimum found (μs)	FEs	Time taken (s)
1	POISE (NM)	48.38	14	110–114
2	POISE (MDS)	48.25–48.38	16	123–126
3	POISE (BOBYQA)	48.26–48.33	9	69–70

tbl:poisecal_53

Table 3.3: Pulse width optimisations with an initial point of 53 μs . The POISE routine is the same as in table 3.1, except with "init":[53.0]. *Data code:* 6F-200826.

It is tempting to use this example to draw the conclusion that the initial point does not matter in POISE optimisations. However, this is only really true for a simple optimisation like this. Looking again at the reference grid search in fig. 3.3, it is clear that there is no other possible minimum that the optimiser could converge to. Furthermore, the noise in the cost function is almost indiscernible. These represent the *ideal* conditions for an experimental optimisation to work, and it is not surprising that extremely good performance is obtained with POISE. Some of the subsequent examples include more difficult or more noisy cost functions. We will see that POISE does indeed have *some* tolerance towards poor initial points, even in the presence of noise (after all, this is the entire purpose of using derivative-free algorithms). However, for very challenging optimisations it is very likely that the optimisation will ultimately converge to a local minimum near the initial point.

3.4.2 Ernst angle optimisation

Often, in a simple 1D pulse–acquire spectrum it is not hugely important to know the exact 90° pulse width: instead, it is more valuable to optimise the sensitivity per unit time of the spectrum.

Optimisation setup

./figures/pp/poise/zg_repeated.png

Figure 3.5: Steady-state pulse–acquire experiment. The excitation flip angle is θ , and the repetition time between experiments is τ_r .

Before launching straight into how this may be obtained through optimisation, it is instructive to first consider which parameters are worth optimising. For a pulse–acquire experiment (fig. 3.5), the repetition time is the sum of the acquisition time AQ plus the recovery delay D1; the flip angle θ is controlled via the pulse width P1. We assume that the experiment has been repeated enough times to reach a *steady state*, that is, the amount of z-magnetisation prior to the excitation pulse (point ①) is a constant $M_{z,ss}$. Application of the excitation pulse leads to a signal scaling as $M_{z,ss} \sin \theta$, and residual (unexcited) longitudinal magnetisation of $M_{z,ss} \cos \theta$ at point ②. After the repetition time τ_r (point ③), it can be shown using the Bloch equations⁴⁶ that the z-magnetisation recovers to

$$M_{z,0}(1 - c) + cM_{z,ss} \cos \theta, \quad (3.22) \quad \{\text{eq:z_magn_ernst1}\}$$

where $c = \exp(-\tau_r/T_1)$ and $M_{z,0}$ is the initial, equilibrium z-magnetisation (before the experiment begins). Since the experiment has reached a steady state, points ① and ③ are equivalent: thus, we have that

$$M_{z,0}(1 - c) + cM_{z,ss} \cos \theta = M_{z,ss}, \quad (3.23) \quad \{\text{eq:z_magn_ernst2}\}$$

which can be rearranged to give

$$\frac{M_{z,ss}}{M_{z,0}} = \frac{1 - c}{1 - c \cos \theta}. \quad (3.24) \quad \{\text{eq:z_magn_ernst3}\}$$

The signal amplitude s therefore scales as

$$s = \frac{(1 - c)}{1 - c \cos \theta} \cdot \sin \theta, \quad (3.25) \quad \{\text{eq:z_magn_ernst4}\}$$

and is maximised when $ds/d\theta = 0$, the solution of which is the celebrated *Ernst angle*:⁴⁷

$$\theta_E = \arccos c = \arccos \left[\exp \left(-\frac{\tau_r}{T_1} \right) \right]. \quad (3.26) \quad \text{{eq:ernst_angle}}$$

In general, T_1 and hence θ_E varies across the different spins in a given sample, so some degree of compromise is required in order to maximise sensitivity for all peaks.

Naively, we may then consider fixing τ_r and optimising P1 to locate the Ernst angle (or to be precise, the pulse width which corresponds to the Ernst angle, since that is the only quantity we really care about). This is generally true. However, we can go one step further, because τ_r itself is comprised of two parameters, and the sensitivity *per unit time* may be affected by varying τ_r . Since the signal scales as $1/\tau_r$ (a shorter τ_r means more repetitions per unit time) but the noise scales only as $\sqrt{1/\tau_r}$, the sensitivity per unit time is

$$S = \frac{(1 - c) \sin \theta}{(1 - c \cos \theta) \sqrt{\tau_r}}. \quad (3.27) \quad \text{{eq:z_magn_ernst5}}$$

Assuming that θ is always set to the respective Ernst angle for different τ_r , it can be shown that the best sensitivity per unit time is attained when $\tau_r \rightarrow 0$.^{48,49} Of course, this limit is not physically possible: τ_r comprises the acquisition time which must be nonzero. However, it does imply that AQ should be kept as short as possible, and D1 set to zero, as shown in fig. 3.6.

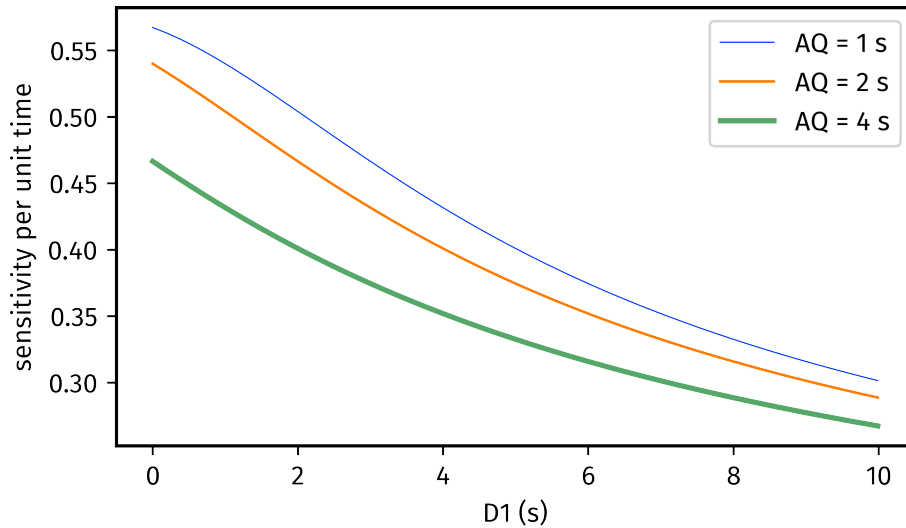


Figure 3.6: Sensitivity per unit time as a function of AQ and D1, assuming that an Ernst angle excitation pulse is used. T_1 was set to 1.5 s.

Knowing this, we can then set up a meaningful optimisation routine. We seek to optimise the pulse width P1 such that the intensity of the real part of the spectrum is maximised (corresponding to a `maxrealint` cost function). In practice, I took the extra step of calibrating the 90° pulse width (as per the previous section) and modifying the pulse programme such that the flip angle could be

specified as the parameter `CNST20`: this is not generally necessary and is only useful for evaluating the results, as will be shown later. As per the above analysis, I set `AQ` and `D1` to be 1.2 s and 0 respectively. The number of scans (`NS`) can be set to 1, but unlike in the pulse width calibration (§ 3.4.1), we must use enough dummy scans to ensure that a steady-state signal intensity is recorded: in practice I set `DS=4`. This means that each FE, and thus the overall optimisation, requires a slightly longer time than the pulse width calibrations previously shown.

Optimisation results

The Ernst angle optimisation was run with two different spectral regions of interest: firstly, on all the aromatic and olefinic peaks in the sample of ferulic acid (table 3.4), and secondly, on only the peak at 6.79 ppm (table 3.5). (This spectral region can be selected using the TopSpin `dp1` command, which stores the bounds using the parameters `F1P` and `F2P`: all built-in cost functions respect these two parameters.) Generally, optimisations could be completed in under two minutes. The optima found for these two optimisations are different: this is because the former searches for a compromise Ernst angle which balances T_1 of all peaks within the region, and the latter optimises only for one T_1 .

Entry	Algorithm	Optimum found (°)	FEs	Time taken (s)
1	NM	67.5–73.1	9–13	91–132
2	MDS	67.5–73.1	9	90–92
3	BOBYQA	70.1–70.7	7	70–71

Table 3.4: Ernst angle optimisation, performed on all aromatic and olefinic peaks in ferulic acid (between 6 and 8 ppm). The POISE routine used here is: `{"name": "ernst", "pars": ["cnst20"], "lb": [10.0], "ub": [90.0], "init": [30.0], "tol": [3.0], "cf": "maxrealint", "au": "poise_1d"}`. *Data code:* 5F-210619.

Entry	Algorithm	Optimum found (°)	FEs	Time taken (s)
1	NM	60.0–67.5	9–11	91–111
2	MDS	65.6–67.5	11	110–111
3	BOBYQA	60.0–65.2	6–7	59–71

Table 3.5: Ernst angle optimisations on the peak at 6.79 ppm in ferulic acid. The POISE routine is the same as in table 3.4, but the spectral region under optimisation was set to be 6.71–6.87 ppm. The theoretical optimum, as given in table 3.6, is 61.6°. *Data code:* 5F-210619.

To determine the accuracy of the optima found, rather than performing a reference grid search as in § 3.4.1, I measured T_1 of each of these peaks using a typical gradient-enhanced inversion–recovery experiment, and calculated the theoretical Ernst angles from this (table 3.6). The first

optimisation, which should yield essentially a weighted average of the five Ernst angles, appears at first glance to be biased towards larger values. However, this can be rationalised by the fact that a flip angle larger than θ_E is less detrimental to sensitivity compared to one that is smaller (this can be seen by plotting eq. (3.25)). On the other hand, the second optimisation yields an accurate value for the relevant peak (number 4 in table 3.6), at least to within the specified tolerance of 3°.

Peak	¹ H chemical shift (ppm)	T_1 (s)	θ_E (°)	$T_1 \ln 2$ (s)
1	7.49	1.750	59.8	1.213
2	7.27	0.977	73.0	0.677
3	7.08	1.279	67.0	0.887
4	6.79	1.615	61.6	1.119
5	6.36	1.415	64.6	0.981

tbl:ernst_invrec

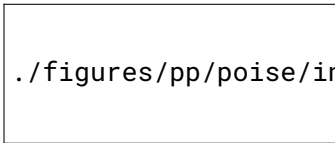
Table 3.6: T_1 and corresponding Ernst angles for each peak in ferulic acid, calculated for a repetition time of 1.20 s. The values of $T_1 \ln 2$ are also provided here, in anticipation of the inversion–recovery experiments performed in § 3.4.3. Data code: 5F-210619.

Finally, it is worth considering whether this optimisation is truly worth it. ¹H pulse–acquire spectra already have a very high intrinsic sensitivity, and in the two minutes taken to optimise the flip angle, one could easily just acquire (around) 64 more scans, at which point knowledge of the Ernst angle would cease to be useful. It *may* be more useful for nuclei which have lower sensitivity, such as ¹³C: I did not evaluate this possibility. However, it must be borne in mind that a low-sensitivity experiment will also require more scans per FE, which leads to a corresponding increase in the optimisation time.

In my estimation, a more useful application of this optimisation routine would be to use it to determine an average T_1 value for a group of peaks. This could then be used to inform the choice of recovery delay for multidimensional experiments^{50,51} or quantitative NMR experiments.^{52,53} It is, however, possible to more directly obtain T_1 values from an inversion–recovery experiment, which I describe next.

3.4.3 Inversion–recovery

Optimisation setup



./figures/pp/poise/invrec.png

fig:poise_invrec

Figure 3.7: Inversion–recovery pulse sequence.

If what we really want to measure is T_1 for a particular peak (or a set of peaks), a more direct way is to perform an inversion–recovery experiment (fig. 3.7). This can either be recorded in a 2D form where the τ delay is incremented and the resulting intensities fit to an exponential curve, or in an iterative fashion by searching for the null in spectral intensity, which occurs at $\tau = T_1 \ln 2$: this latter option is particularly suited to optimisation. The delay τ was specified as the parameter D27, and the `zerorealint` cost function used here is just the absolute value of the integral over the region of interest: an ideal null would have a cost function value of 0. Table 3.6 provides the theoretical values of $T_1 \ln 2$, which were calculated using the more accurate 2D fitting procedure.

A slight drawback of using this method, compared to the Ernst angle optimisations in the previous section, is that the recovery delay must be sufficiently long to allow for complete relaxation between FEs: in this case, I used a D1 of 5 s. On the other hand, this also means that dummy scans are no longer needed: I therefore set DS=0 and NS=1.

Optimisation results

Just as in the Ernst angle optimisations, these were performed twice, once on the entire 6–8 ppm region and once on just a single peak (note that this time, the chosen peak is at 7.08 ppm), or peak 3 in table 3.6). The results are shown in tables 3.7 and 3.8. The former correctly yields an ‘averaged’ value over the five peaks, and the latter closely matches the theoretical value for the peak in question.

Entry	Algorithm	Optimum found (s)	FEs	Time taken (s)
1	NM	0.938–0.969	14–16	204–235
2	MDS	0.956–0.975	16	233–235
3	BOBYQA	0.953–0.971	9–11	130–160

Table 3.7: Inversion–recovery optimisations on all aromatic and olefinic peaks in ferulic acid (between 6 and 8 ppm. The POISE routine used here is: {"name": "invrec", "pars": ["d27"], "lb": [0.35], "ub": [1.75], "init": [0.6], "tol": [0.01], "cf": "zerorealint", "au": "poise_1d"}. Data code: 5F-210619.

The only downside of these optimisations would then be the time required, which is on the order of 2–4 minutes. Although this is less time than required for a full 2D inversion–recovery experiment, POISE has the drawback that an optimisation can only be run on one peak at a time. Thus, if the aim is to determine T_1 for all peaks, then the 2D experiment may well end up being faster. On top of that, there are many other ways to measure T_1 which are faster than a full 2D inversion–recovery experiment and almost certainly also faster than POISE.^{54–58} However, no explicit comparisons were performed in this work.

Entry	Algorithm	Optimum found (s)	FEs	Time taken (s)
1	NM	0.863–0.875	14	202–205
2	MDS	0.863–0.869	14	203–204
3	BOBYQA	0.862–0.873	9–10	128–145

Table 3.8: Inversion–recovery optimisations on the peak at 7.08 ppm in ferulic acid. The POISE routine is the same as in table 3.4, but the spectral region under optimisation was set to be 7.02–7.15 ppm. The theoretical optimum (from table 3.6) is 0.887 s. *Data code:* 5F-210619.

3.4.4 NOE mixing time

In all of the optimisations done previously, the cost functions used are relatively simple, simply seeking to maximise or minimise some intensity. In fact, the Bruker `popt` interface does come with a number of cost functions itself, which can be used for a grid search-based optimisations: so, in principle, all the previous examples could have been done with `popt` (albeit with a much longer time). However, POISE goes beyond this in that it allows users to define their own cost functions. In this section, we exploit this customisability to devise a more complicated cost function for optimising mixing times in NOE experiments.

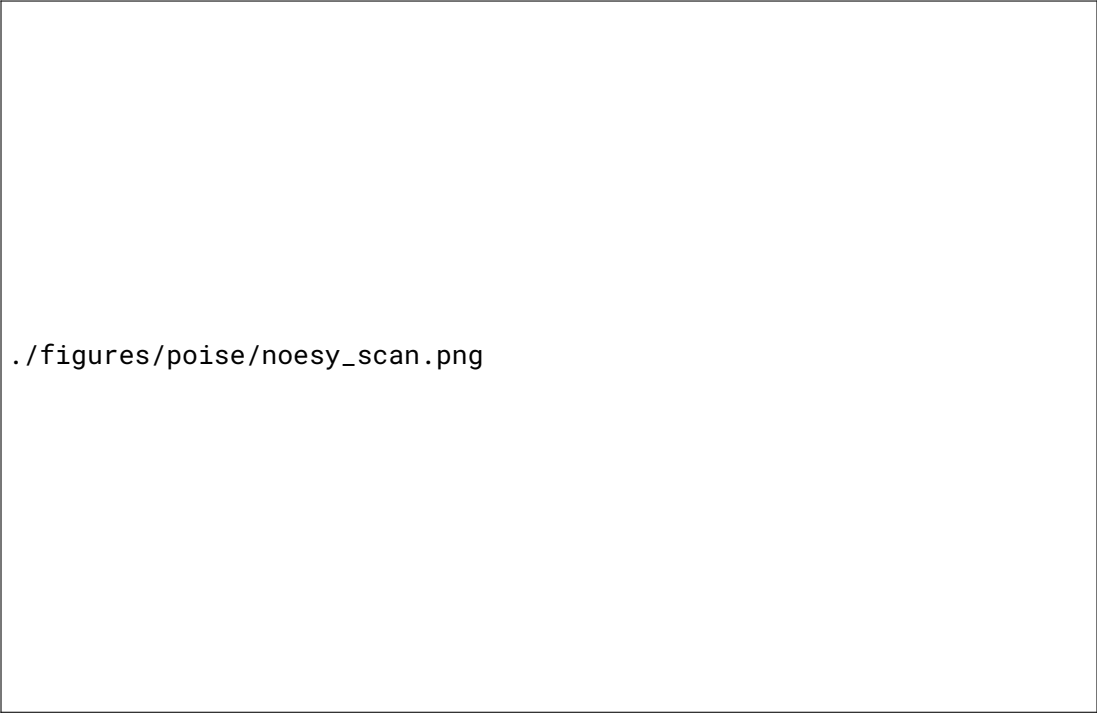
Optimisation setup

The ideal NOE mixing time for a given compound depends on the rates of various relaxation processes: too short a mixing time does not allow for sufficient buildup of the NOE, but too long a mixing time leads to loss of signal through relaxation. In this section, I do not deal with this theoretically: the optimisation process is merely used to find the empirically best value (for the sample under study).

In the chosen sample, 3-fluorophenylboronic acid, there are four crosspeaks of interest in the 2D NOESY spectrum. I first performed a reference grid search to determine where the crosspeak intensities were maximised (fig. 3.8). Generally, a broad minimum between 2.5 and 4 s is observed: any result within this range should be considered as ‘correct.’* Although this may at first glance seem imprecise, it merely reflects the underlying physical characteristics of the sample under study: it is not the job of an optimisation process to ‘discover’ extra precision where there is none to be found. We also see here the first example of a cost function where the noise is significant: this provides a good test of the derivative-free algorithms used in POISE.

While the 2D reference grid search offers first-hand insight into what our target optimum should

*There is a complicating factor in that the use of such long mixing times also leads to a noticeable increase in the experiment duration. As such, it is not necessarily the case that an optimised mixing time yields a greater sensitivity *per unit time*. In the optimisations which follow, I have neglected this issue. However, it could probably be somewhat accounted for by modifying the form of the cost function.



./figures/poise/noesy_scan.png


Figure 3.8: Reference grid search of 2D NOE crosspeak intensities. The individual crosspeak intensities are shown in dashed lines; the solid black line is the average of these four, which represents the quantity we seek to optimise. *Data code:* 7B-200725.

be, it is unwise to run an optimisation using the full 2D experiment, simply because of the time required for each FE. A more sensible method is to use a selective 1D NOESY experiment, where the mixing time is represented by the parameter D8. Although this is much faster, it does come with two caveats:

1. The frequency for selective irradiation must be first chosen, likely after acquisition of a 1D ^1H spectrum. Thus, the optimisation does require some *a priori* knowledge of the system being studied.
2. The crosspeak intensities in the 1D NOESY *must* be sufficiently representative of those in the full 2D NOESY.

The cost function used (noe_1d) must thus pick out only the crosspeaks from the 1D NOESY spectrum. It does this by detecting the frequency used for the selective irradiation (which is given by the SPOFFS2 parameter), excising a region of ca. 100 Hz around the irradiation frequency, and integrating the remainder of the spectrum. In order to account for the fact that the NOE crosspeaks may be either positive or negative (depending not only on the molecular weight, but also how the spectrum is phased), the absolute value of the integral is taken, and the negative of this is used as the cost function (since we seek to maximise the intensity). Note that if a different 1D NOESY pulse programme is used with different parameter definitions, then the cost function

must be adjusted accordingly.



./figures/pp/poise/noe1d.png

fig: noe1d_pulseq

Figure 3.9: Selective 1D NOESY pulse sequence used for POISE optimisations. Phase cycling was performed using $\phi_1 = \phi_{\text{rec}} = (x, -x)$. The exact gradient amplitudes used are not hugely important, except for the z-filter gradient g_4 , which should be calibrated as per the protocol in Thrippleton et al.⁵⁹

In the event, I used a modified 1D NOESY pulse programme (fig. 3.9), with two extra inversion pulses during the mixing time: these minimise artefacts arising from relaxation during the mixing time, which can be especially problematic for long mixing times of several seconds. (In principle, these artefacts have equal positive and negative components and thus should not contribute to the cost function except in terms of noise; however, it is always a good idea to minimise the noise in the cost function as much as possible.) The initial value for the optimisation was set to 0.5 s, which is a reasonable compromise value for most ‘small’ organic molecules.

Optimisation results

Entry	Algorithm	Optimum found (s)	FEs	Time taken (s)
1	NM	3.25–3.88	16–18	268–312
2	MDS	3.63–3.75	16–18	269–305
3	BOBYQA	3.38–3.80	6–10	88–162

tbl:poise_noe_3fpba

Table 3.9: NOE mixing time optimisations on a sample of 3-fluorophenylboronic acid. The POISE routine used here is: `{"name": "1dnoe", "pars": ["d8"], "lb": [0.2], "ub": [6.0], "init": [0.5], "tol": [0.1], "cf": "noe_1d", "au": "poise_1d"}`.
Data code: 7B-200721.

The results of this optimisation are shown in table 3.9. In all cases, the optimisations converged to the correct region within 2.5 minutes (for BOBYQA) and 5 minutes (for the simplex-based algorithms). The resulting 2D NOESY spectra, with the initial and optimised mixing times of 0.5 s and 3.5 s respectively, are shown in fig. 3.10, where the improvement in crosspeak sensitivity is clearly visible.

It should be mentioned here that each FE was run using one dummy scan and two scans. This was made possible due to the high SNR afforded by a concentrated sample (120 mM), as well as a cryogenic probe. For more dilute samples where SNR is insufficient, the POISE optimisation will require more scans per FE, and consequently will take longer. However, it can be argued that the

./figures/poise/noesy_spec.png

Figure 3.10: 2D NOESY spectra of 3-fluorophenylboronic acid, obtained (a) before and (b) after optimising the mixing time on the 1D NOESY sequence in fig. 3.9. Both spectra are plotted with the same contour levels. Data code: 7B-200725.

benefit reaped from the optimisation is also correspondingly larger, since the final (optimised) 2D NOESY which is run will also take a longer time.

A different sample

To more clearly illustrate the *sample-specific* nature of POISE optimisations, it is also useful to run the same optimisation on a different sample: in this case, the decapeptide gramicidin S. For this rather larger compound, we would expect the optimisation to converge instead to a shorter mixing time, and this is validated in practice (table 3.10). The peak at 4.76 ppm was used for selective irradiation: this is the H^α proton of the ornithine residue.

Entry	Algorithm	Optimum found (s)	FEs	Time taken (s)
1	NM	0.63–0.78	9–14	253–384
2	MDS	0.44–0.75	9–11	254–309
3	BOBYQA	0.58–0.77	5–8	136–223

Table 3.10: NOE mixing time optimisations on a sample of gramicidin S. The POISE routine used here is identical to before. Data code: 7G-210815.

3.4.5 ASAP-HSQC excitation delay

The next example of the ASAP-HSQC experiment^{60–63} continues to illustrate how custom cost functions in POISE can be used to carry out a variety of optimisations. In the ASAP-HSQC experiment (fig. 3.11), a HSQC spectrum is recorded using only ^{13}C -bound proton magnetisation, and the ^{12}C -bound (‘bulk’) proton magnetisation is returned to the equilibrium +z axis (i.e. the I_z state). In this way, instead of having a conventional recovery delay, the ^{13}C - ^1H

magnetisation can be directly replenished using isotropic mixing, which causes transfer of z -magnetisation from bulk protons. The elision of the recovery delay from the sequence thus leads to significantly shorter experiment durations: in one of the more extreme examples, a HSQC spectrum could be recorded within 7 seconds. This separation of different ‘magnetisation pools’ is conceptually very similar to that in NOAH experiments, and in anticipation of that, I use the notation $^1\text{H}^{\text{C}}$ and $^1\text{H}^{\text{!C}}$ to represent magnetisation belonging to protons coupled and not coupled to ^{13}C respectively.

./figures/pp/poise/asaphsqc.png

Figure 3.11: ASAP-HSQC pulse sequence. Phase cycling is performed using $\phi_1 = (x, -x)$, $\phi_2 = (x, x, -x, -x)$, and $\phi_{\text{rec}} = (x, -x, x, -x)$. Gradient amplitudes are as follows: $g_1 = 33\%$, $g_2 = 43\%$, and for echo–antiecho selection, $(g_3, g_4) = (59.9\%, 80\%)$ and $(63.9\%, 80\%)$ respectively. The delay Δ is set to $1/(4 \cdot ^1J_{\text{CH}})$; the delay Δ_{E} was optimised as described in the text. The BIBOP, BEBOP, and BURBOP optimal control pulses^{11,13,14} were used for ^1H 180° pulses and all ^{13}C pulses; for more details, refer to the original paper by Luy and coworkers.⁶⁰

By modifying the length of the delay Δ_{E} in the initial INEPT block, it is possible to adjust the proportion of the $^1\text{H}^{\text{C}}$ magnetisation excited during the sequence: the remainder is stored along z . This means that the signal is decreased by a factor of $\sin \theta_{\text{eff}}$ (where θ_{eff} is an effective flip angle), but means that on the next scan or increment, there is a larger pool of $^1\text{H}^{\text{C}}$ magnetisation to start from. The combination of these factors means that there is an optimum Δ_{E} which yields the greatest steady-state signal. (Notice that this is entirely analogous to the Ernst angle previously discussed in § 3.4.2.) The delay Δ_{E} is related to the effective flip angle, θ_{eff} , through

$$\theta_{\text{eff}} = 2\pi J \Delta_{\text{E}}. \quad (3.28) \quad \{\text{eq:asaphsqc_ernst_ang}\}$$

where J is here short for $^1J_{\text{CH}}$. Equivalently, we can define an ‘effective J-coupling’, J_{eff} for which the excitation is optimised:

$$\Delta_{\text{E}} = \frac{1}{4J_{\text{eff}}} \iff J_{\text{eff}} = \frac{\pi J}{2\theta_{\text{eff}}}. \quad (3.29) \quad \{\text{eq:asaphsqc_j_eff}\}$$

Optimisation setup

Given the analysis above, it is extraordinarily easy to set up the optimisation: the parameter being optimised is J_{eff} , and the initial point chosen is simply $^1J_{\text{CH}}$ (to be precise, a ‘compromise’ $^1J_{\text{CH}}$

value of 150 Hz). Although generally it is not a good idea to use a 2D experiment as part of the FE, in this specific case it is acceptable because the ASAP-HSQC has such a short running time. In this case, we simply take the projection of the 2D ASAP-HSQC onto the F_2 axis, integrate it, and take the negative to obtain the cost function (negative because we are trying to maximise the intensity).

A reference grid search was performed so that I could later verify the optimisation results (fig. 3.12). The exact point where the crosspeak intensities are maximised is not obvious: generally, the signal increases up until $J_{\text{eff}} \approx 230$ Hz, after which it plateaus off. In a similar fashion to the NOE optimisations (§ 3.4.4), we may therefore consider any value above this to be ‘correct’.

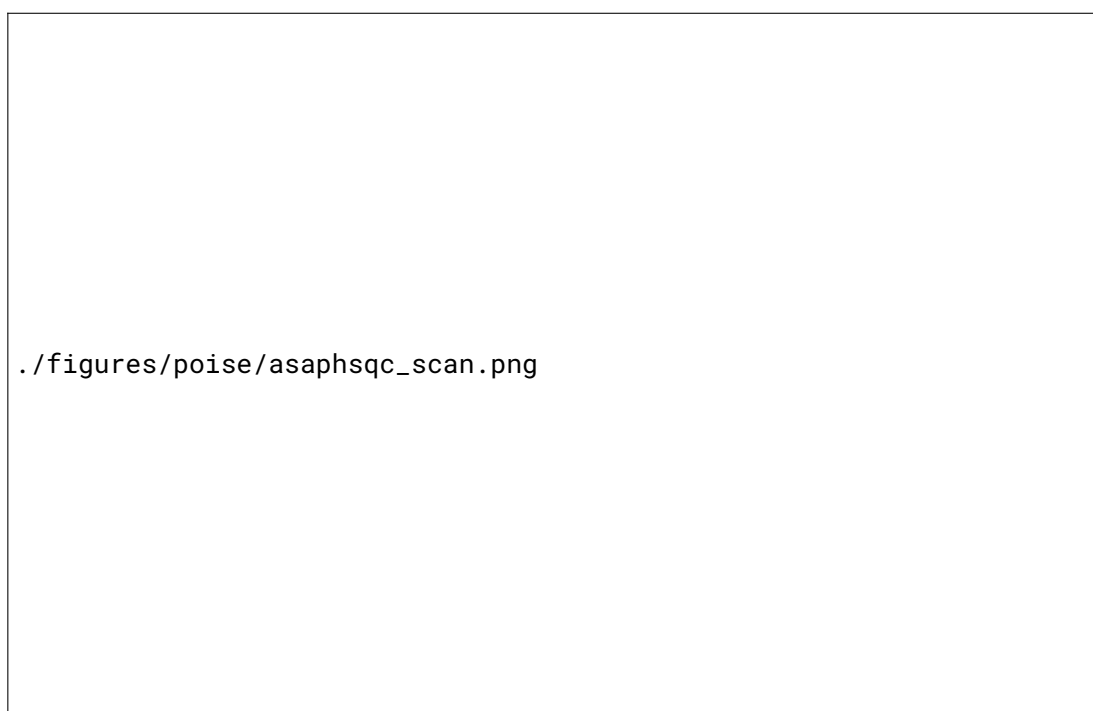


fig:asaphsqc_scan

Figure 3.12: Reference grid search showing how the ASAP-HSQC signal intensity for the 3-fluorophenylboronic acid sample varies with J_{eff} . Data code: 7B-200722.

To speed up the optimisation, optimisations were run using a ‘low-quality’ ASAP-HSQC spectrum with only 32 t_1 increments. A recovery delay of 0.1 s was used.

Optimisation results

The results from the POISE optimisations are collated in table 3.11. As can be seen, even though the FE involves acquisition of a 2D spectrum, the optimisation itself completes in just 2–3 minutes, correctly converging to J_{eff} values in the region of around 240 Hz. When re-evaluated on a ‘full’ ASAP-HSQC experiment with 64 t_1 increments (fig. 3.13a),* the optimisation yielded an 18–27%

*The ^{13}C resonances of interest in this compound fall within a very small window, so relatively few t_1 increments are needed.

Entry	Algorithm	Optimum found (Hz)	FEs	Time taken (s)
1	NM	250.0–256.3	8–9	169–195
2	MDS	237.5–243.8	8	171–179
3	BOBYQA	229.8–245.6	4–7	114–157

Table 3.11: ASAP-HSQC INEPT delay optimisations. The POISE routine used was: {"name": "asaphsqc", "pars": ["cnst3"], "lb": [120.0], "ub": [280.0], "init": [150.0], "tol": [10.0], "cf": "asaphsqc", "au": "poise_2d"}. *Data code:* 7B-200722.

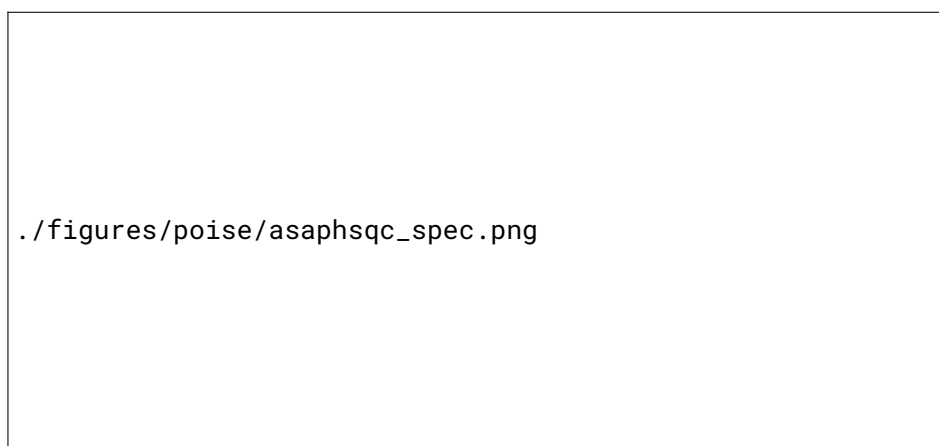


Figure 3.13: (a) ASAP-HSQC spectrum of the 3-fluorophenylboronic acid sample. (b) Projections of the ASAP-HSQC spectra before (grey dotted line, using $J_{\text{eff}} = 150$ Hz) and after (green dotted line, using $J_{\text{eff}} = 245$ Hz) optimisation of the INEPT delay. Sensitivity increases for each peak are indicated. *Data code:* 7B-200722.

improvement in the sensitivity of the ASAP-HSQC spectrum, as shown in fig. 3.13b.

3.4.6 Ultrafast NMR

The final example of a single-parameter POISE optimisation is also the most complicated: it pertains to the EPSI acquisition technique, which was previously introduced in § 2.7. EPSI acquisition allows signals from different slices of the sample to be simultaneously detected in a single FID, and (in liquid-state NMR) has most famously been used in the context of *ultrafast* 2D experiments,^{64–69} where the t_1 evolution is spatially encoded and read out using the EPSI technique.

During an EPSI acquisition period, alternating gradients of equal magnitude but opposite sign are applied, as shown in FIG – zgepsi and tocsyepsi. Each data point collected therefore depends not only on t_2 (the acquisition time domain) but also on k , a wavenumber which measures the

extent of gradient dephasing caused by the acquisition gradients:

$$k = \int_0^{t_2} \gamma G(t) dt. \quad (3.30) \quad \text{{eq:epsi_k_space}}$$

This value of k increases during positive gradients and decreases during negative gradients, leading to a zigzag pattern in k -space, all while t_2 is still increasing (FIG – data points, correct and incorrect) The *prephasing gradient* immediately before acquisition has the same duration as the EPSI gradients, but has half the amplitude, meaning that k begins from a negative value, specifically, $k_{\max}/2$, where k_{\max} is the change in k caused by one complete positive EPSI gradient. As shown in previous reviews,⁶⁶ the spatial profile $f(z)$ can be obtained by Fourier transformation of the k domain. Alternatively, since t_1 is proportional to z and the indirect-dimension frequency F_1 is itself obtained through Fourier transformation of the t_1 -domain, the F_1 and k domains are in fact directly proportional: no Fourier transform is required along this axis to obtain the indirect-dimension frequencies.

This rapid alternating of gradients is very demanding on spectrometers, and it is often the case that the positive and negative EPSI gradients—although *nominally* specified with the same amplitude—are not perfectly balanced. This causes a ‘drift’ in the k -domain over time, as illustrated in FIG. In this section, POISE is used to perform an *instrument-specific* optimisation in order to correct for this effect.

change axis order of k, t_2 — Frydman always uses a weird depiction for whatever reason...

Optimisation setup

To measure the drift in k -space, it is easiest to use a pulse sequence for which there is no spatial encoding; the pulse–EPSI experiment (FIG) is perfectly suited for this. In the pulse programme, I multiplied the amplitudes of the negative gradients by a factor α , represented as CNST16 in TopSpin: the objective of POISE is therefore to determine the optimum value for this which minimises the drift. Calculating this drift requires fairly substantial processing, which is most easily done inside the cost function using numpy functions (rather than, for example, a TopSpin AU script). This consists of the following steps:

1. reading in the 1D FID and reshaping it into a 2D (k, t_2) matrix;
2. discarding data obtained during negative gradients;
3. determining, for each point in t_2 , the value of k for which the maximum signal is found;
4. performing linear regression on these points and obtaining the slope of k against t_2 , which directly represents the drift in k -space.

3.4.7 HMBC low-pass J-filter

Artefacts.

3.4.8 PSYCHE pure shift NMR

J-refocusing

3.4.9 Solvent suppression

Mouse

3.4.10 Diffusion NMR

Automated DOSY

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

3.5 POISE for ESR

Need JB to write this section

3.6 References

- (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](https://doi.org/10.1021/acs.analchem.1c01767).
- (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](https://doi.org/10.1016/S0009-2614(97)01081-6).
- (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](https://doi.org/10.1006/jmre.1999.1824).
- (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](https://doi.org/10.1515/zna-2000-1-213).
- (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](https://doi.org/10.1016/j.jmr.2020.106858).
- (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](https://doi.org/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](https://doi.org/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](https://doi.org/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 ^1H solid-state NMR spectroscopy. *Chem. Phys. Lett.* **2010**, 498, 214–220, DOI: [10.1016/j.cplett.2010.08.038](https://doi.org/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](https://doi.org/10.1016/s1090-7807(03)00153-8).
- Kobzar2004JMR (11) Kobzar, K.; Skinner, T. E.; Khaneja, N.; Glaser, S. J.; Luy, B. Exploring the limits of broadband excitation and inversion pulses. *J. Magn. Reson.* **2004**, 170, 236–243, DOI: [10.1016/j.jmr.2004.06.017](https://doi.org/10.1016/j.jmr.2004.06.017).
- Khaneja2005JMR (12) 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](https://doi.org/10.1016/j.jmr.2004.11.004).
- Kobzar2008JMR (13) 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](https://doi.org/10.1016/j.jmr.2008.05.023).
- Kobzar2012JMR (14) 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](https://doi.org/10.1016/j.jmr.2012.09.013).
- Schilling2014ACIE (15) Schilling, F.; Warner, L. R.; Gershenson, 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](https://doi.org/10.1002/anie.201400178).
- Glaser2015EPJD (16) 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](https://doi.org/10.1140/epjd/e2015-60464-1).
- Geen1989JMR (17) 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](https://doi.org/10.1016/0022-2364(89)90254-0).

- Emsley1990CPL (18) 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](https://doi.org/10.1016/0009-2614(90)87025-m).
- Geen1991JMR (19) Geen, H.; Freeman, R. Band-selective radiofrequency pulses. *J. Magn. Reson.* **1991**, *93*, 93–141, DOI: [10.1016/0022-2364\(91\)90034-q](https://doi.org/10.1016/0022-2364(91)90034-q).
- Nuzillard1994JMRSa (20) 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](https://doi.org/10.1006/jmra.1994.1056).
- Kupce1995JMRSa (21) Kupce, E.; Freeman, R. Band-Selective Correlation Spectroscopy. *J. Magn. Reson., Ser. A* **1995**, *112*, 134–137, DOI: [10.1006/jmra.1995.1023](https://doi.org/10.1006/jmra.1995.1023).
- Kupce1995JMRSb (22) 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/jmr.1995.1049](https://doi.org/10.1006/jmr.1995.1049).
- 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](https://doi.org/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](https://doi.org/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](https://doi.org/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 B₁-field inhomogeneities (COB3). *J. Magn. Reson.* **2014**, *247*, 111–117, DOI: [10.1016/j.jmr.2014.07.010](https://doi.org/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](https://doi.org/10.1016/j.jmr.2019.106641).
- Sakellariou2000CPL (28) 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](https://doi.org/10.1016/S0009-2614(00)00127-5).
- Nocedal2006 (29) Nocedal, J.; Wright, S. J., *Numerical Optimization*, 2nd ed.; Springer: New York, 2006.
- Nelder1965TCJ (30) 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](https://doi.org/10.1093/comjnl/7.4.308).
- Torczon1989 (31) Torczon, V. J. Multidirectional search: A direct search algorithm for parallel machines, Ph.D. Rice University, 1989.
- DennisJr1991SIAMJ (32) Dennis Jr., J. E.; Torczon, V. Direct Search Methods on Parallel Machines. *SIAM J. Optim.* **1991**, *1*, 448–474, DOI: [10.1137/0801027](https://doi.org/10.1137/0801027).
- Powell12009Proc (33) Powell, M. J. D. *The BOBYQA algorithm for bound constrained optimization without derivatives*; tech. rep. DAMTP 2009/NA06; University of Cambridge, 2009.

- Cartis2019ACMTMS (34) 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](https://doi.org/10.1145/3338517).
- Spendley1962T (35) Spendley, W.; Hext, G. R.; Himsworth, F. R. Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation. *Technometrics* **1962**, *4*, 441–461, DOI: [10.1080/00401706.1962.10490033](https://doi.org/10.1080/00401706.1962.10490033).
- Torczon1991SIAMJO (36) Torczon, V. On the Convergence of the Multidirectional Search Algorithm. *SIAM J. Optim.* **1991**, *1*, 123–145, DOI: [10.1137/0801010](https://doi.org/10.1137/0801010).
- Powell12003MP (37) Powell, M. J. D. On trust region methods for unconstrained minimization without derivatives. *Mathematical Programming* **2003**, *97*, 605–623, DOI: [10.1007/s10107-003-0430-6](https://doi.org/10.1007/s10107-003-0430-6).
- Cartis2022O (38) Cartis, C.; Roberts, L.; Sheridan-Methven, O. Escaping local minima with local derivative-free methods: a numerical investigation. *Optimization* **2022**, *71*, 2343–2373, DOI: [10.1080/02331934.2021.1883015](https://doi.org/10.1080/02331934.2021.1883015).
- Song2018JMR (39) 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](https://doi.org/10.1016/j.jmr.2018.02.009).
- Tang2019SR (40) 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](https://doi.org/10.1038/s41598-019-53929-1).
- Eghbalnia2005JACS (41) 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](https://doi.org/10.1021/ja052120i).
- Hansen2016ACIE (42) 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](https://doi.org/10.1002/anie.201608048).
- BrukerSmartDriveNMR (43) 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).
- Keifer1999CMR (44) Keifer, P. A. 90° pulse width calibrations: How to read a pulse width array. *Concepts Magn. Reson.* **1999**, *11*, 165–180, DOI: [10.1002/\(SICI\)1099-0534\(1999\)11:3<165::AID-CMR4>3.0.CO;2-D](https://doi.org/10.1002/(SICI)1099-0534(1999)11:3<165::AID-CMR4>3.0.CO;2-D).
- Wu2005JMR (45) Wu, P. S. C.; Otting, G. Rapid pulse length determination in high-resolution NMR. *J. Magn. Reson.* **2005**, *176*, 115–119, DOI: [10.1016/j.jmr.2005.05.018](https://doi.org/10.1016/j.jmr.2005.05.018).
- Bloch1946PR (46) Bloch, F. Nuclear Induction. *Phys. Rev.* **1946**, *70*, 460–474, DOI: [10.1103/physrev.70.460](https://doi.org/10.1103/physrev.70.460).
- Ernst1966RSI (47) Ernst, R. R.; Anderson, W. A. Application of Fourier Transform Spectroscopy to Magnetic Resonance. *Rev. Sci. Instrum.* **1966**, *37*, 93–102, DOI: [10.1063/1.1719961](https://doi.org/10.1063/1.1719961).

- Waugh1970JMS (48) Waugh, J. S. Sensitivity in Fourier transform NMR spectroscopy of slowly relaxing systems. *J. Mol. Spectrosc.* **1970**, 35, 298–305, DOI: [10.1016/0022-2852\(70\)90205-5](https://doi.org/10.1016/0022-2852(70)90205-5).
- Traficante1992CMR (49) Traficante, D. D. Optimum tip angle and relaxation delay for quantitative analysis. *Concepts Magn. Reson.* **1992**, 4, 153–160, DOI: [10.1002/cmr.1820040204](https://doi.org/10.1002/cmr.1820040204).
- Reynolds2002JNP (50) Reynolds, W. F.; Enríquez, R. G. Choosing the Best Pulse Sequences, Acquisition Parameters, Postacquisition Processing Strategies, and Probes for Natural Product Structure Elucidation by NMR Spectroscopy. *J. Nat. Prod.* **2002**, 65, 221–244, DOI: [10.1021/np010444o](https://doi.org/10.1021/np010444o).
- Burns2021MRC (51) Burns, D. C.; Reynolds, W. F. Minimizing the risk of deducing wrong natural product structures from NMR data. *Magn. Reson. Chem.* **2021**, 59, 500–533, DOI: [10.1002/mrc.4933](https://doi.org/10.1002/mrc.4933).
- Pauli2005JNP (52) Pauli, G. F.; Jaki, B. U.; Lankin, D. C. Quantitative ^1H NMR: Development and Potential of a Method for Natural Products Analysis. *J. Nat. Prod.* **2005**, 68, 133–149, DOI: [10.1021/np0497301](https://doi.org/10.1021/np0497301).
- Giraudeau2014MRC (53) Giraudeau, P. Quantitative 2D liquid-state NMR. *Magn. Reson. Chem.* **2014**, 52, 259–272, DOI: [10.1002/mrc.4068](https://doi.org/10.1002/mrc.4068).
- Christensen1974JPC (54) Christensen, K. A.; Grant, D. M.; Schulman, E. M.; Walling, C. Optimal Determination of Relaxation Times of Fourier Transform Nuclear Magnetic Resonance. Determination of Spin–Lattice Relaxation Times in Chemically Polarized Species. *J. Phys. Chem.* **1974**, 78, 1971–1977, DOI: [10.1021/j100612a022](https://doi.org/10.1021/j100612a022).
- Homer1985JMR (55) Homer, J.; Beevers, M. S. Driven-equilibrium single-pulse observation of T_1 relaxation. A reevaluation of a rapid “new” method for determining NMR spin-lattice relaxation times. *J. Magn. Reson.* **1985**, 63, 287–297, DOI: [10.1016/0022-2364\(85\)90318-x](https://doi.org/10.1016/0022-2364(85)90318-x).
- Loening2003JMR (56) Loening, N. M.; Thrippleton, M. J.; Keeler, J.; Griffin, R. G. Single-scan longitudinal relaxation measurements in high-resolution NMR spectroscopy. *J. Magn. Reson.* **2003**, 164, 321–328, DOI: [10.1016/s1090-7807\(03\)00186-1](https://doi.org/10.1016/s1090-7807(03)00186-1).
- Smith2013CPC (57) Smith, P. E. S.; Donovan, K. J.; Szekely, O.; Baias, M.; Frydman, L. Ultrafast NMR T_1 Relaxation Measurements: Probing Molecular Properties in Real Time. *ChemPhysChem* **2013**, 14, 3138–3145, DOI: [10.1002/cphc.201300436](https://doi.org/10.1002/cphc.201300436).
- Wei2021JOC (58) Wei, R.; Dickson, C. L.; Uhrín, D.; Lloyd-Jones, G. C. Rapid Estimation of T_1 for Quantitative NMR. *J. Org. Chem.* **2021**, 86, 9023–9029, DOI: [10.1021/acs.joc.1c01007](https://doi.org/10.1021/acs.joc.1c01007).
- Thrippleton2003ACIE (59) Thrippleton, M. J.; Keeler, J. Elimination of Zero-Quantum Interference in Two-Dimensional NMR Spectra. *Angew. Chem., Int. Ed.* **2003**, 42, 3938–3941, DOI: [10.1002/anie.200351947](https://doi.org/10.1002/anie.200351947).
- Sunninghausen2014JACS (60) Schulze-Sünninghausen, D.; Becker, J.; Luy, B. Rapid Heteronuclear Single Quantum Correlation NMR Spectra at Natural Abundance. *J. Am. Chem. Soc.* **2014**, 136, 1242–1245, DOI: [10.1021/ja411588d](https://doi.org/10.1021/ja411588d).

- zeSunninghausen2017JMR (61) Schulze-Sünninghausen, D.; Becker, J.; Koos, M. R. M.; Luy, B. Improvements, extensions, and practical aspects of rapid ASAP-HSQC and ALSOFAST-HSQC pulse sequences for studying small molecules at natural abundance. *J. Magn. Reson.* **2017**, *281*, 151–161, DOI: [10.1016/j.jmr.2017.05.012](https://doi.org/10.1016/j.jmr.2017.05.012).
- Koos2019JMR (62) Koos, M. R. M.; Luy, B. Polarization recovery during ASAP and SOFAST/ALSOFAST-type experiments. *J. Magn. Reson.* **2019**, *300*, 61–75, DOI: [10.1016/j.jmr.2018.12.014](https://doi.org/10.1016/j.jmr.2018.12.014).
- Becker2019JMR (63) Becker, J.; Koos, M. R. M.; Schulze-Sünninghausen, D.; Luy, B. ASAP-HSQC-TOCSY for fast spin system identification and extraction of long-range couplings. *J. Magn. Reson.* **2019**, *300*, 76–83, DOI: [10.1016/j.jmr.2018.12.021](https://doi.org/10.1016/j.jmr.2018.12.021).
- Frydman2002PNASUSA (64) Frydman, L.; Scherf, T.; Lupulescu, A. The acquisition of multidimensional NMR spectra within a single scan. *Proc. Natl. Acad. Sci. U. S. A.* **2002**, *99*, 15858–15862, DOI: [10.1073/pnas.252644399](https://doi.org/10.1073/pnas.252644399).
- Pelupessy2003JACS (65) Pelupessy, P. Adiabatic Single Scan Two-Dimensional NMR Spectroscopy. *J. Am. Chem. Soc.* **2003**, *125*, 12345–12350, DOI: [10.1021/ja034958g](https://doi.org/10.1021/ja034958g).
- Frydman2003JACS (66) Frydman, L.; Lupulescu, A.; Scherf, T. Principles and Features of Single-Scan Two-Dimensional NMR Spectroscopy. *J. Am. Chem. Soc.* **2003**, *125*, 9204–9217, DOI: [10.1021/ja030055b](https://doi.org/10.1021/ja030055b).
- Tal2010PNMRS (67) Tal, A.; Frydman, L. Single-scan multidimensional magnetic resonance. *Prog. Nucl. Magn. Reson. Spectrosc.* **2010**, *57*, 241–292, DOI: [10.1016/j.pnmrs.2010.04.001](https://doi.org/10.1016/j.pnmrs.2010.04.001).
- Giraudeau2014ARAC (68) Giraudeau, P.; Frydman, L. Ultrafast 2D NMR: An Emerging Tool in Analytical Spectroscopy. *Annu. Rev. Anal. Chem.* **2014**, *7*, 129–161, DOI: [10.1146/annurev-anchem-071213-020208](https://doi.org/10.1146/annurev-anchem-071213-020208).
- Gouilleux2018ARNMRS (69) Gouilleux, B.; Rouger, L.; Giraudeau, P. Ultrafast 2D NMR: Methods and Applications. *Annu. Rep. NMR Spectrosc.* **2018**, 75–144, DOI: [10.1016/bs.arnmr.2017.08.003](https://doi.org/10.1016/bs.arnmr.2017.08.003).

refsection:4

refsection:5