Supplementary Information

Newton meets Ockham: parameter estimation and model selection of NMR data with NMR-EsPy

Simon G. Hulse, Mohammadali Foroozandeh*

Chemistry Research Laboratory, University of Oxford, 12 Mansfield Road, Oxford, OX1 3TA, U.K.

* mohammadali.foroozandeh@chem.ox.ac.uk

Contents

A Preliminary Information

We consider a one-dimensional FID $\boldsymbol{y} \in \mathbb{C}^N$, for which it is assumed that:

$$y_n = \underbrace{\sum_{m=1}^{M} a_m \exp(\mathrm{i}\phi_m) \exp\left[\left(2\pi f_m - \eta_m\right) n \Delta t\right]}_{x_n(\boldsymbol{\theta})} + w_n \quad \forall n \in \{0, 1, \cdots, N-1\}, \tag{S1a}$$

$$\boldsymbol{\theta} = \begin{bmatrix} a_1, & a_2, & \cdots, & a_M, & \phi_1, & \cdots, & \phi_M, & f_1, & \cdots, & \eta_1, & \cdots, & \eta_M \end{bmatrix}^{\mathrm{T}} \tag{S1b}$$

B An Outline of the Matrix Pencil Method

The Matrix Pencil Method computes an estimate of the poles $\{z_m\}$, $m \in \{1, 2, \ldots, M\}$ associated with $\boldsymbol{y}$. Consider the noiseless Hankel matrix $\boldsymbol{X}$:

$$\boldsymbol{X} = \begin{pmatrix} x_0 & x_1 & \cdots & x_L \\ x_1 & x_2 & \cdots & x_{L+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N-L-1} & x_{N-L} & \cdots & x_{N-1} \end{pmatrix} \tag{S2}$$

Two matrices $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ can be formed by the removal of the last and first column of $\boldsymbol{X}$, respectively.

$$\boldsymbol{X}_1 = \begin{pmatrix} x_0 & x_1 & \cdots & x_{L-1} \\ x_1 & x_2 & \cdots & x_L \\ \vdots & \vdots & \ddots & \vdots \\ x_{N-L-1} & x_{N-L} & \cdots & x_{N-2} \end{pmatrix}, \tag{S3a}$$

$$\boldsymbol{X}_2 = \begin{pmatrix} x_1 & x_2 & \cdots & x_L \\ x_2 & x_3 & \cdots & x_{L+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N-L} & x_{N-L+1} & \cdots & x_{N-1} \end{pmatrix}. \tag{S3b}$$

These can be deconstructed into

$$\boldsymbol{X}_1 = \boldsymbol{Z}_{\mathrm{L}} \boldsymbol{A} \boldsymbol{Z}_{\mathrm{R}}, \tag{S4a}$$

$$\boldsymbol{X}_2 = \boldsymbol{Z}_{\mathrm{L}} \boldsymbol{A} \boldsymbol{Z}_{\mathrm{D}} \boldsymbol{Z}_{\mathrm{R}}, \tag{S4b}$$

where

$$\boldsymbol{Z}_{\mathrm{L}} = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ z_1 & z_2 & \cdots & z_M \\ \vdots & \vdots & \ddots & \vdots \\ z_1^{N-L-1} & z_2^{N-L-1} & \ldots & z_M^{N-L-1} \end{pmatrix}, \tag{S5a}$$

$$\boldsymbol{Z}_{\mathrm{R}} = \begin{pmatrix} 1 & z_1 & \cdots & z_1^{L-1} \\ 1 & z_2 & \cdots & z_2^{L-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & z_M & \cdots & z_M^{L-1} \end{pmatrix}, \tag{S5b}$$

$$\boldsymbol{Z}_{\mathrm{D}} = \mathrm{diag}(z_1, z_2, \cdots, z_M), \tag{S5c}$$

$$\boldsymbol{A} = \mathrm{diag}(\alpha_1, \alpha_2, \cdots, \alpha_M). \tag{S5d}$$

Consequently, the matrix pencil $\boldsymbol{X}_2 - \lambda \boldsymbol{X}_1$ can be expressed as

$$\boldsymbol{X}_2 - \lambda \boldsymbol{X}_1 = \boldsymbol{Z}_{\mathrm{L}} \boldsymbol{A} (\boldsymbol{Z}_{\mathrm{D}} - \lambda \boldsymbol{I}_M) \boldsymbol{Z}_{\mathrm{R}}, \quad \lambda \in \mathbb{C}, \tag{S6}$$

where $\boldsymbol{I}_M$ is the $M \times M$ identity matrix. Providing the condition $M \leq L \leq N - M$ is satisfied, $\mathrm{rank}(\boldsymbol{X}_2 - \lambda \boldsymbol{X}_1) = M$. When $\lambda = z_m \; \forall m \in \{1, 2, \cdots, M\}$, $\left[\boldsymbol{Z}_{\mathrm{D}} - \lambda \boldsymbol{I}_M\right]_{mm} = 0$. Therefore, the matrix pencil's rank is reduced by 1, making the signal poles the rank-reducing values of $\boldsymbol{X}_2 - \lambda \boldsymbol{X}_1$. Equivalently, the poles can be found by determining the eigenvalues of $\boldsymbol{X}_1^{+} \boldsymbol{X}_2$.

For noisy data, since the data matrix $\boldsymbol{Y}$ is likely to be of full rank, a pre-filtration is necessary, driven by the result of model order selection. $\boldsymbol{Y}$ can be expressed in terms of its SVD:

$$\boldsymbol{Y} = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}^{\dagger}, \tag{S7}$$

where the columns of $\boldsymbol{U} \in \mathbb{C}^{(N-L)\times(N-L)}$ and $\boldsymbol{V} \in \mathbb{C}^{(L+1)\times(L+1)}$ contain the left and right singular vectors of $\boldsymbol{Y}$, respectively. $\boldsymbol{U}$ corresponds to the complete set of eigenvectors $\{\boldsymbol{u}\}$ of $\boldsymbol{Y}\boldsymbol{Y}^{\dagger}$, whilst $\boldsymbol{V}$ corresponds to the complete set of eigenvectors $\{\boldsymbol{v}\}$ of $\boldsymbol{Y}^{\dagger}\boldsymbol{Y}$. $\boldsymbol{\Sigma} \in \mathbb{R}^{(N-L)\times(L+1)}$ is a rectangular diagonal matrix, containing the singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_M \geq \cdots \geq \sigma_R$ of $\boldsymbol{Y}$, where $R = \min(N - L, L + 1)$. The singular values are equivalent to the square roots of the eigenvalues of $\boldsymbol{Y}^{\dagger}\boldsymbol{Y}$ as well as $\boldsymbol{Y}\boldsymbol{Y}^{\dagger}$.

For a data-set with model order M, it is necessary to reduce the rank of $\boldsymbol{Y}$ from R to M, whilst ensuring it resembles the original matrix as closely as possible. The Eckart-Young-Mirsky theorem[1] states that the matrix $\widetilde{\boldsymbol{Y}}$, a low-rank approximation of matrix $\boldsymbol{Y}$, can be expressed as a truncated SVD of $\boldsymbol{Y}$:

$$\widetilde{\boldsymbol{Y}} = \sum_{m=1}^{M} \sigma_m \boldsymbol{u}_m \boldsymbol{v}_m^{\dagger} = \widetilde{\boldsymbol{U}} \widetilde{\boldsymbol{\Sigma}} \widetilde{\boldsymbol{V}}^{\dagger}. \tag{S8}$$

This matrix minimises $\|\boldsymbol{Y} - \widetilde{\boldsymbol{Y}}\|_{\mathrm{F}}^2$, where $\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm. Here $\boldsymbol{u}_m$ and $\boldsymbol{v}_m$ correspond to the m^{th} left and right singular vectors of $\boldsymbol{Y}$, respectively, $\widetilde{\boldsymbol{U}} = \left[\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_M\right]$, $\widetilde{\boldsymbol{V}} = \left[\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_M\right]$, and

$\widetilde{\boldsymbol{\Sigma}} = \mathrm{diag}(\sigma_1, \sigma_2, \cdots, \sigma_M)$.

The poles of $\boldsymbol{y}$ are obtained by computation of the eigenvalues of $\widetilde{\boldsymbol{Y}}_1^{+}\widetilde{\boldsymbol{Y}}_2$, where $\widetilde{\boldsymbol{Y}}_1$ and $\widetilde{\boldsymbol{Y}}_2$ are constructed from $\widetilde{\boldsymbol{Y}}$ by the removal of the last and first column respectively. Alternatively, the same result can be achieved by computing the M non-zero eigenvalues of $\widetilde{\boldsymbol{V}}_1^{+}\widetilde{\boldsymbol{V}}_2$, where

$$\widetilde{\boldsymbol{V}}_1 = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_{M-1} \end{bmatrix}, \tag{S9a}$$

$$\widetilde{\boldsymbol{V}}_2 = \begin{bmatrix} \boldsymbol{v}_2 & \boldsymbol{v}_3 & \cdots & \boldsymbol{v}_{M} \end{bmatrix}. \tag{S9b}$$

C Derivative Definitions

The first and second partial derivatives of the model vector elements $x_n(\boldsymbol{\theta})$ and the phase variance $\mathrm{Var}(\boldsymbol{\phi})$ are required for the computation of the gradient and Hessian of the fidelity $\mathcal{F}^{\phi}(\boldsymbol{\theta})$, given by:

$$\left[\nabla \mathcal{F}^{\phi}\right]_i = -2\Re\left[\left(\boldsymbol{y} - \boldsymbol{x}\right)^{\dagger} \frac{\partial \boldsymbol{x}}{\partial \theta_i}\right] + \frac{1}{\pi}\frac{\partial\, \mathrm{Var}(\boldsymbol{\phi})}{\partial \theta_i}, \tag{S10a}$$

$$\left[\nabla^2 \mathcal{F}^{\phi}\right]_{ij} = -2\Re\left[\left(\boldsymbol{y} - \boldsymbol{x}\right)^{\dagger} \frac{\partial^2 \boldsymbol{x}}{\partial \theta_i \partial \theta_j} - \frac{\partial \boldsymbol{x}^{\dagger}}{\partial \theta_i}\frac{\partial \boldsymbol{x}}{\partial \theta_j}\right] + \frac{1}{\pi}\frac{\partial^2\, \mathrm{Var}(\boldsymbol{\phi})}{\partial \theta_i \partial \theta_j}, \tag{S10b}$$

$$\forall\, i, j \in \{1, 2, \cdots, 4M\}.$$

C.1 Derivatives of Model

$$\frac{\partial x_n}{\partial a_m} = \frac{x_n}{a_m} \tag{S11a}$$

$$\frac{\partial x_n}{\partial \phi_m} = \mathrm{i} x_n \tag{S11b}$$

$$\frac{\partial x_n}{\partial f_m} = 2\pi \mathrm{i} \tau_n x_n \tag{S11c}$$

$$\frac{\partial x_n}{\partial \eta_m} = -\tau_n x_n \tag{S11d}$$

$$\frac{\partial^2 x_n}{\partial a_m^2} = 0 \tag{S11e}$$

$$\frac{\partial^2 x_n}{\partial a_m \partial \phi_m} = \frac{\mathrm{i} x_n}{a_m} \tag{S11f}$$

$$\frac{\partial^2 x_n}{\partial a_m \partial f_m} = \frac{2\pi \mathrm{i} \tau_n x_n}{a_m} \tag{S11g}$$

$$\frac{\partial^2 x_n}{\partial a_m \partial \eta_m} = \frac{-\tau_n x_n}{a_m} \tag{S11h}$$

$$\frac{\partial^2 x_n}{\partial \phi_m^2} = -x_n \tag{S11i}$$

$$\frac{\partial^2 x_n}{\partial \phi_m \partial f_m} = -2\pi \tau_n x_n \tag{S11j}$$

$$\frac{\partial^2 x_n}{\partial \phi_m \partial \eta_m} = -\mathrm{i} \tau_n x_n \tag{S11k}$$

$$\frac{\partial^2 x_n}{\partial f_m^2} = -4\pi^2 \tau_n^2 x_n \tag{S11l}$$

$$\frac{\partial^2 x_n}{\partial f_m \partial \eta_m} = -2\pi \mathrm{i} \tau_n^2 x_n \tag{S11m}$$

$$\frac{\partial^2 x_n}{\partial \eta_m^2} = \tau_n^2 x_n \tag{S11n}$$

$$\frac{\partial^2 x_n}{\partial \theta_i \partial \theta_j} = 0 \quad \text{otherwise} \tag{S11o}$$

$$\frac{\partial^2 x_n}{\partial \theta_i \partial \theta_j} = \frac{\partial^2 x_n}{\partial \theta_j \partial \theta_i} \tag{S11p}$$

Note that Eq. S11p illustrates the symmetric structure of the second derivatives, and Eq. S11o implies that if θ_i and θ_j are parameters of different oscillators, the second derivative will be zero. These properties greatly reduce the computational cost of generating $\nabla^2 \mathcal{F}^{\phi}(\boldsymbol{\theta})$, since instead of $16NM^2$ derivative calculations, only $10NM$ (or $9NM$ considering Eq. S11e) are required.

C.2 Derivatives of Phase Variance

$$\frac{\partial \operatorname{Var}(\boldsymbol{\phi})}{\partial \theta_i} = \begin{cases} \frac{2}{M}\left(\theta_i - \mu^{\phi}\right) & M < i \leq 2M \\ 0 & \text{otherwise} \end{cases} \tag{S12a}$$

$$\frac{\partial^2 \operatorname{Var}(\boldsymbol{\phi})}{\partial \theta_i \partial \theta_j} = \begin{cases} \frac{2M-2}{M^2} & i = j \text{ and } M < i \leq 2M \\ -\frac{2}{M^2} & i \neq j \text{ and } M < i, j \leq 2M \\ 0 & \text{otherwise} \end{cases} \tag{S12b}$$

D Derivation of Standard Errors

The well know relationship between the covariance matrix, $\boldsymbol{C}(\boldsymbol{\theta})$, and the observed Fisher Information matrix, $\boldsymbol{I}(\boldsymbol{\theta})$, of the estimated parameter vector, $\hat{\boldsymbol{\theta}}$ (henceforth written without the hat, though this is implied in this section), can be written as[2, Chapter 8]

$$\boldsymbol{C}(\boldsymbol{\theta}) = \boldsymbol{I}(\boldsymbol{\theta})^{-1} \tag{S13}$$

where the elements of the observed Fisher Information matrix,

$$[\boldsymbol{I}(\boldsymbol{\theta})]_{ij} = -\frac{\partial^2 \ell(\boldsymbol{\theta}|\boldsymbol{y})}{\partial\theta_i \partial\theta_j}, \tag{S14}$$

provide information about the curvature of the log-likelihood function:

$$\ell(\boldsymbol{\theta}|\boldsymbol{y}) = -N\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\|\boldsymbol{y} - \boldsymbol{x}(\boldsymbol{\theta})\|_2^2. \tag{S15}$$

As such, the Fisher Information matrix provides a metric for ascertaining how accurately one can estimate a given parameter. Individual elements in the Fisher Information matrix are therefore given by

$$[\boldsymbol{I}(\boldsymbol{\theta})]_{ij} = -\frac{1}{\sigma^2}\Re\left[\left(\boldsymbol{y} - \boldsymbol{x}(\boldsymbol{\theta})\right)^\dagger \frac{\partial^2 \boldsymbol{x}(\boldsymbol{\theta})}{\partial\theta_i\partial\theta_j} - \frac{\partial \boldsymbol{x}(\boldsymbol{\theta})}{\partial\theta_i}^\dagger \frac{\partial \boldsymbol{x}(\boldsymbol{\theta})}{\partial\theta_j}\right], \tag{S16}$$

which very closely resemble the expression for the Hessian of the fidelity, excluding the phase variance penalty (Eq. S10)

$$[\boldsymbol{I}(\boldsymbol{\theta})]_{ij} = \frac{1}{2\sigma^2}\left[\nabla^2\mathcal{F}(\boldsymbol{\theta})\right]_{ij}. \tag{S17}$$

The relationship between the standard error and the covariance matrix is

$$\mathrm{SE}(\boldsymbol{\theta}) = \sqrt{\mathrm{diag}(\boldsymbol{C}(\boldsymbol{\theta}))} \tag{S18}$$

and therefore the standard error can be written as

$$\mathrm{SE}(\boldsymbol{\theta}) = \sqrt{2\sigma^2\,\mathrm{diag}\left(\left[\nabla^2\mathcal{F}(\boldsymbol{\theta})\right]^{-1}\right)}. \tag{S19}$$

Considering that the mean and variance of the noise, $\boldsymbol{w}$, are 0 and $2\sigma^2$, respectively, we can write

$$2\sigma^2 = \frac{1}{N-1}\sum_{n=0}^{N-1}|w_n|^2 = \frac{1}{N-1}\|\boldsymbol{y} - \boldsymbol{x}(\boldsymbol{\theta})\|_2^2. \tag{S20}$$

Finally, we arrive at the following expression for the standard error of the parameter estimate:

$$\mathrm{SE}(\boldsymbol{\theta}) = \sqrt{\frac{\mathcal{F}(\boldsymbol{\theta})\,\mathrm{diag}\Big(\big[\nabla^2\mathcal{F}(\boldsymbol{\theta})\big]^{-1}\Big)}{N-1}}. \tag{S21}$$

E Details of Spectral Truncation for Frequency Filtering

In order to reduce the computational cost of our estimation technique, it is typically necessary to apply frequency filtration to data. The following approach achieves this (see the main text for details):

$$\tilde{\boldsymbol{y}} = \mathrm{IFT}\Big\{\underbrace{\mathrm{FT}\big\{\boldsymbol{y}_{\mathrm{ve}}\big\}\odot \boldsymbol{g} + \boldsymbol{w}}_{\tilde{\boldsymbol{s}}_{\mathrm{ve}}}\Big\}[:N-1], \tag{S22}$$

where $\boldsymbol{y}_{\mathrm{ve}} \in \mathbb{C}^{2N-1}$ denotes a virtual echo signal constructed from $\boldsymbol{y}$, $\boldsymbol{g}$ is a super-Gaussian function which acts as a band-pass filter, and $\boldsymbol{w}$ is a synthetic noise vector to maintain the SNR present in the signal.

After filtering a specified spectral region by use of a band-pass filter, it is beneficial to discard regions far away from the spectral segment of interest, as those contain no meaningful information. Subsequently, an IFT of such trimmed spectrum, $\tilde{\boldsymbol{s}}_{\mathrm{trm}}$, results in a filtered FID, $\tilde{\boldsymbol{y}}_{\mathrm{trm}}$. There are important considerations however:

(i) By discarding points in the frequency-domain signal, the associated sweep width and transmitter offset both need to be re-defined to their filtered analogues, $\tilde{f}_{\mathrm{sw}}$, and $\tilde{f}_{\mathrm{off}}$,

(ii) The signal must be re-scaled, the requirement of which can be recognised through the definition of IFT

$$y_n = \frac{1}{N}\sum_{k=0}^{N-1} s_k \exp\left(\frac{2\mathrm{i}\pi kn}{N}\right). \tag{S23}$$

To account for the above, the resulting time-domain signal is subsequently obtained via

$$\tilde{\boldsymbol{y}}_{\mathrm{trm}} = \frac{r-l}{2N-1}\,\mathrm{IFT}\big\{\tilde{\boldsymbol{s}}_{\mathrm{trm}}\big\}\left[0:\left\lfloor\frac{r-l}{2}\right\rfloor\right], \tag{S24a}$$

$$\tilde{\boldsymbol{s}}_{\mathrm{trm}} = \tilde{\boldsymbol{s}}_{\mathrm{ve}}[l:r+1], \tag{S24b}$$

$$l = \begin{cases} c - \left\lfloor\frac{b\gamma}{2}\right\rfloor & \text{if} \geq 0 \\ 0 & \text{otherwise} \end{cases}, \tag{S24c}$$

$$r = \begin{cases} c + \left\lceil \frac{b\gamma}{2} \right\rceil & \text{if} \leq 2N - 1 \\ 2N - 2 & \text{otherwise} \end{cases}, \tag{S24d}$$

where c denotes the central index of the super-Gaussian, and (l/r) denotes the index of the (leftmost/rightmost) point to retain within the spectral data. These can also be used to derive the corrected sweep width and transmitter offset:

$$\tilde{f}_{\text{sw}} = \frac{r - l - 1}{2N - 2} f_{\text{sw}}, \tag{S25a}$$

$$\tilde{f}_{\text{off}} = f_{\text{off}} + \frac{f_{\text{sw}}}{2}\left(1 - \frac{r + l}{N - 1}\right). \tag{S25b}$$

F Assignments for Multiplet Structures in Regions Considered

Cyclosporin A			
Region (ppm)	Nuclei	Multiplet Structure(s)	Comments
5.54–5.42	H^a, H^d	dd, dd	
5.287–5.185	H^g	ddd	Two of the ^3J values are so similar, there is an apparent dt structure.
Andrographolide			
Region (ppm)	Nucleus	Multiplet Structure	Comments
1.43–1.29	H^c	dddd	Similar values for ^2J coupling H^b and ^3J ax-ax couplings to H^d and H^a give rise to an apparent quartet. Weak ^3J ax-eq coupling to H^e gives rise to narrow doublets. Leads to an apparent dq structure.
2.38–2.28	H^e	ddd	Strong ^2J coupling to H^d. Weaker ^3J eq-eq and ax-eq couplings to H^b and H^c, respectively.
6.665–6.59	H^i	dt	^3J couplings to H^g and H^h. Small ^4J coupling to H^f.

G Parameter Tables

G.1 Cyclosporin: 5.54–5.42 ppm

m	a_m	ϕ_m (rad)	f_m (Hz)	f_m (ppm)	η_m (s^{-1})
1	256.35 ± 2.93	0.043 337 ± 7.85×10^{-3}	2719.8 ± 2.90×10^{-3}	5.4396 ± 5.80×10^{-6}	9.2571 ± 9.2571
2	269.26 ± 2.77	0.021 353 ± 0.0102	2725.3 ± 7.75×10^{-3}	5.4506 ± 1.55×10^{-5}	9.2599 ± 9.2599
3	258.32 ± 5.87	0.014 426 ± 2.91×10^{-3}	2728.4 ± 0.0115	5.4568 ± 2.30×10^{-5}	9.0054 ± 9.0054
4	232.82 ± 3.96	-9.4864×10^{-3} ± 0.0139	2733.8 ± 0.0197	5.4676 ± 3.94×10^{-5}	8.8395 ± 8.8395
5	242.42 ± 2.55	0.019 378 ± 0.0212	2743.1 ± 0.0238	5.4862 ± 4.76×10^{-5}	7.5556 ± 7.5556
6	260.19 ± 4.27	−0.012 428 ± 9.96×10^{-3}	2748.6 ± 0.0123	5.4971 ± 2.45×10^{-5}	7.6242 ± 7.6242
7	255.87 ± 2.84	−0.014 27 ± 0.0104	2753.5 ± 0.0123	5.507 ± 2.46×10^{-5}	7.5974 ± 7.5974
8	251.75 ± 2.26	−0.036 223 ± 7.27×10^{-3}	2758.9 ± 0.011	5.5178 ± 2.20×10^{-5}	7.947 ± 7.947
9	16.328 ± 1.69	-6.4546×10^{-4} ± 7.83×10^{-3}	2766.4 ± 0.168	5.5327 ± 3.36×10^{-4}	8.8827 ± 8.8827

G.2 Cyclosporin: 5.285–5.18 ppm

m	a_m	ϕ_m (rad)	f_m (Hz)	f_m (ppm)	η_m (s^{-1})
1	123.53 ± 1.14	0.027 584 ± 0.0385	2601.9 ± 0.0445	5.2038 ± 8.91×10^{-5}	7.3367 ± 7.3367
2	256.07 ± 3.85	4.7964×10^{-3} ± 9.64×10^{-3}	2609.3 ± 7.04×10^{-3}	5.2186 ± 1.41×10^{-5}	6.7065 ± 6.7065
3	124.64 ± 4.21	5.8777×10^{-3} ± 9.69×10^{-3}	2611.6 ± 0.0224	5.2232 ± 4.49×10^{-5}	7.4814 ± 7.4814
4	148.18 ± 7.23	-2.6056×10^{-3} ± 0.0231	2616.7 ± 0.0205	5.2333 ± 4.10×10^{-5}	8.3593 ± 8.3593
5	234.39 ± 4.47	-7.9273×10^{-3} ± 0.0189	2619.0 ± 0.0104	5.238 ± 2.08×10^{-5}	6.3265 ± 6.3265

6	5.5912 ± 2.18	$-1.4543 \times 10^{-5} \pm 0.0169$	2621.9 ± 0.13	$5.2437 \pm 2.59 \times 10^{-4}$	5.6014 ± 5.6014
7	170.31 ± 1.99	$-0.015\,855 \pm 9.07 \times 10^{-3}$	2626.2 ± 0.012	$5.2523 \pm 2.40 \times 10^{-5}$	8.6777 ± 8.6777
8	5.6255 ± 1.12	$-5.7523 \times 10^{-4} \pm 9.59 \times 10^{-3}$	2629.9 ± 0.117	$5.2598 \pm 2.34 \times 10^{-4}$	5.5072 ± 5.5072
9	36.273 ± 1.03	$-9.0214 \times 10^{-3} \pm 7.05 \times 10^{-3}$	2636.5 ± 0.0243	$5.2729 \pm 4.87 \times 10^{-5}$	6.9703 ± 6.9703

G.3 Andrographolide: 6.665–6.59 ppm

m	a_m	ϕ_m (rad)	f_m (Hz)	f_m (ppm)	η_m (s^{-1})
1	74.043 ± 1.5	$0.037\,753 \pm 5.84 \times 10^{-3}$	$3307.3 \pm 5.47 \times 10^{-3}$	$6.6146 \pm 1.09 \times 10^{-5}$	4.838 ± 4.838
2	125.61 ± 1.69	$0.036\,018 \pm 0.0157$	$3309.0 \pm 6.63 \times 10^{-3}$	$6.618 \pm 1.33 \times 10^{-5}$	6.6188 ± 6.6188
3	188.08 ± 2.4	$0.022\,865 \pm 8.52 \times 10^{-3}$	$3314.1 \pm 4.85 \times 10^{-3}$	$6.6281 \pm 9.70 \times 10^{-6}$	5.0766 ± 5.0766
4	234.14 ± 0.265	$0.014\,801 \pm 5.44 \times 10^{-3}$	$3315.8 \pm 3.85 \times 10^{-3}$	$6.6317 \pm 7.69 \times 10^{-6}$	5.8585 ± 5.8585
5	118.06 ± 1.55	$4.6183 \times 10^{-3} \pm 3.93 \times 10^{-3}$	$3320.9 \pm 7.22 \times 10^{-3}$	$6.6419 \pm 1.44 \times 10^{-5}$	6.5361 ± 6.5361
6	90.834 ± 1.4	$5.1973 \times 10^{-3} \pm 4.36 \times 10^{-3}$	$3322.7 \pm 6.43 \times 10^{-3}$	$6.6454 \pm 1.29 \times 10^{-5}$	5.6551 ± 5.6551

G.4 Andrographolide: 2.38–2.28 ppm

m	a_m	ϕ_m (rad)	f_m (Hz)	f_m (ppm)	η_m (s^{-1})
1	100.13 ± 1.64	$0.022\,898 \pm 5.08 \times 10^{-3}$	$1153.3 \pm 9.23 \times 10^{-3}$	$2.3066 \pm 1.85 \times 10^{-5}$	7.6436 ± 7.6436
2	140.11 ± 3.42	$0.014\,134 \pm 6.14 \times 10^{-3}$	1155.6 ± 0.011	$2.3112 \pm 2.19 \times 10^{-5}$	8.0485 ± 8.0485
3	137.28 ± 3.38	$8.0611 \times 10^{-3} \pm 5.22 \times 10^{-3}$	$1157.4 \pm 9.78 \times 10^{-3}$	$2.3149 \pm 1.96 \times 10^{-5}$	7.9874 ± 7.9874

4	102.6 ± 1.5	$-3.0507 \times 10^{-3} \pm 8.08 \times 10^{-3}$	$1159.8 \pm 8.93 \times 10^{-3}$	$2.3196 \pm 1.79 \times 10^{-5}$	6.7816 ± 6.7816
5	89.536 ± 1.37	0.012 617 ± 0.0107	$1166.0 \pm 3.22 \times 10^{-3}$	$2.332 \pm 6.44 \times 10^{-6}$	6.4465 ± 6.4465
6	128.16 ± 3.45	$-9.0239 \times 10^{-4} \pm 7.65 \times 10^{-3}$	1168.5 ± 0.0129	$2.337 \pm 2.57 \times 10^{-5}$	7.8568 ± 7.8568
7	115.2 ± 3.71	$-4.4391 \times 10^{-3} \pm 7.04 \times 10^{-3}$	1170.3 ± 0.0136	$2.3405 \pm 2.73 \times 10^{-5}$	7.8965 ± 7.8965
8	86.502 ± 1.6	$-9.7376 \times 10^{-3} \pm 5.24 \times 10^{-3}$	1172.6 ± 0.011	$2.3452 \pm 2.19 \times 10^{-5}$	7.4506 ± 7.4506

G.5 Andrographolide: 1.43–1.29 ppm

m	a_m	ϕ_m (rad)	f_m (Hz)	f_m (ppm)	η_m (s^{-1})
1	58.754 ± 0.104	$7.8367 \times 10^{-3} \pm 5.43 \times 10^{-3}$	$657.11 \pm 5.74 \times 10^{-3}$	$1.3142 \pm 1.15 \times 10^{-5}$	8.4133 ± 8.4133
2	65.548 ± 0.265	$6.3650 \times 10^{-3} \pm 6.94 \times 10^{-3}$	$661.26 \pm 2.14 \times 10^{-3}$	$1.3225 \pm 4.28 \times 10^{-6}$	8.0351 ± 8.0351
3	172.1 ± 0.403	$9.8997 \times 10^{-4} \pm 4.30 \times 10^{-3}$	$670.1 \pm 5.33 \times 10^{-3}$	$1.3402 \pm 1.07 \times 10^{-5}$	8.2409 ± 8.2409
4	169.62 ± 0.723	$-8.4000 \times 10^{-3} \pm 3.29 \times 10^{-3}$	$674.29 \pm 4.47 \times 10^{-3}$	$1.3486 \pm 8.93 \times 10^{-6}$	7.8737 ± 7.8737
5	163.35 ± 0.688	$7.4533 \times 10^{-4} \pm 4.18 \times 10^{-3}$	$683.12 \pm 5.32 \times 10^{-3}$	$1.3662 \pm 1.06 \times 10^{-5}$	8.1844 ± 8.1844
6	158.33 ± 0.649	$-0.018\,976 \pm 3.99 \times 10^{-3}$	$687.29 \pm 5.29 \times 10^{-3}$	$1.3746 \pm 1.06 \times 10^{-5}$	8.3117 ± 8.3117
7	52.743 ± 0.491	$-8.4536 \times 10^{-3} \pm 6.29 \times 10^{-3}$	696.14 ± 0.0101	$1.3923 \pm 2.01 \times 10^{-5}$	7.8436 ± 7.8436
8	46.004 ± 0.466	$-0.018\,095 \pm 3.78 \times 10^{-3}$	$700.26 \pm 9.84 \times 10^{-3}$	$1.4005 \pm 1.97 \times 10^{-5}$	7.8435 ± 7.8435

H Python Implementations

Here, code snippets are provided which give an outline of how both the Matrix Pencil Method and Numerical Optimisation are performed using NMR-EsPy. It should be noted that these snippets are trimmed-down versions of the code in NMR-EsPy, to focus on the numerical computation itself.

Simple examples of how these functions can be implemented are provided at the end of certain code blocks, underneath `if __name__ == "__main__":` statements.

H.1 Constructing a Synthetic FID

The following code block defines the function `make_fid`, which enables synthetic FIDs to be generated, with Gaussian white noise.

```python
import numpy as np
import numpy.random as nrandom


def make_fid(
    theta: np.ndarray, N: int, sw: float, offset: float, snr: float
) -> np.ndarray:
    """Construct a 1D FID.

    Parameters
    ----------
    theta
        Parameter array, of form
        ``np.array([a1, ..., aM, ϕ1, ..., ϕM, f1, ..., fM, η1, ..., ηM])``

    N
        Number of points the FID comprises.

    sw
        Sweep width.

    offset
        Transmitter offset.

    snr
        Signal-to-noise ratio (dB).

    Returns
```

```
    -------
    fid
        Synthetic FID.
    """
    assert theta.size % 4 == 0
    M = theta.size // 4
    a = theta[:M]
    phi = theta[M : 2 * M]
    f = theta[2 * M : 3 * M] - offset
    eta = theta[3 * M :]
    timepoints = np.linspace(0, (N - 1) / sw, N)
    Z = np.exp(np.outer(timepoints, (2j * np.pi * f - eta)))
    alpha = a * np.exp(1j * phi)
    noiseless_fid = Z @ alpha
    return noiseless_fid + make_noise(noiseless_fid, snr)


def make_noise(fid: np.ndarray, snr: float) -> np.ndarray:
    """Generate an array of white Gaussian complex noise with a given SNR.

    Parameters
    ----------
    fid
        FID to add noise to.

    snr
        Signal-to-nosie ratio (dB).

    Returns
    -------
    noise
        Noise vector.
    """
    N = fid.size
    std = np.std(np.abs(fid)) / (10 ** (snr / 20))
    # Make several instance of noise, and determine how close their SNR
    # is to the desired value.
    instances = []
    discrepancies = []
    for _ in range(20):
        instance = nrandom.normal(loc=0, scale=std, size=N)
        instances.append(instance)
```

```
        discrepancies.append(np.std(np.abs(instance)) - std)

    # Extract the indices of the two noise instances with the closest
    # SNR to the desired value.
    first, second, *_ = np.argpartition(discrepancies, 1)

    return instances[first] + 1j * instances[second]
```

H.2 Matrix Pencil Method with Minimum Description Length

```
import numpy as np
import numpy.linalg as nlinalg
import scipy.linalg as slinalg


def matrix_pencil(fid: np.ndarray, sw: float, offset: float) -> np.ndarray:
    """Perform the Matrix Pencil Method with model order selection.

    Parameters
    ----------
    fid
        The FID to analyse.

    sw
        Sweep width.

    offset
        Transmitter offset (carrier frequency).

    Returns
    -------
    result
        Array of size ``4 * M`` where ``M`` is the number of oscillators determined
        by the MDL.
    """
    # Determine FID size and pencil parameter
    N = fid.size
    L = N // 3

    # Normalise the FID
```

```
    norm = nlinalg.norm(fid)
    normed_fid = fid / norm

    # Singular Value Decomposition of Hankel data matrix
    Y = slinalg.hankel(normed_fid[: N - L], normed_fid[N - L - 1 :])
    _, sigma, Vh = nlinalg.svd(Y)
    V = Vh.T

    # Minimum Description Length
    M = np.argmin(
        [
            -N * np.einsum("i->", np.log(sigma[k:L]))
            + N * (L - k) * np.log(np.einsum("i->", sigma[k:L]) / (L - k))
            + k * np.log(N) * (2 * L - k) / 2
            for k in range(L)
        ]
    )

    # Extract signal poles and complex amplitudes
    Vm = V[:, :M]
    V1 = Vm[:-1]
    V2 = Vm[1:]
    poles = nlinalg.eig(V2 @ nlinalg.pinv(V1))[0][:M]
    alpha = nlinalg.pinv(np.power.outer(poles, np.arange(N))).T @ normed_fid

    # Construct 4M-length vector of parameter estimates
    result = np.zeros((4 * M,))
    # amplitudes
    result[:M] = np.abs(alpha)
    # phases
    result[M : 2 * M] = np.arctan2(np.imag(alpha), np.real(alpha))
    # frequencies
    result[2 * M : 3 * M] = (sw / (2 * np.pi)) * np.imag(np.log(poles)) + offset
    # damping factors
    result[3 * M :] = -sw * np.real(np.log(poles))
    return result

if __name__ == "__main__":
    # Two parameter signal with 2 oscillators
    theta = np.array([1.0, 2.0, 0.0, 0.0, 200.0, -300.0, 5.0, 6.0])
    N = 2048
    sw= 5000.0
```

```
    offset = 0.0
    snr = 30.0
    fid = make_fid(theta, N, sw, offset, snr)
    # Run the matrix pencil method
    result = matrix_pencil(fid, sw, offset)
```

H.3 Non-linear Programming

```
from typing import Tuple
import numpy as np
import numpy.linalg as nlinalg
from scipy import optimize


def numerical_optimisation(
    fid: np.ndarray,
    theta0: np.ndarray,
    sw: float,
    offset: float,
    max_iterations: int = 200,
) -> Tuple[np.ndarray, np.ndarray]:
    """Use numerical optimisation to estimate FID parameters.

    Parameters
    ----------
    fid
        FID to estimate.

    theta0
        Initial parameter guess.

    sw
        Sweep width.

    offset
         Transmitter offset.

    max_iterations
        The maximum number of iterations permitted before the optimiser terminates
        and returns the parameter array.
```

```
    """
    assert theta0.size % 4 == 0  # Parameter array should be multiple of 4
    M = theta0.size // 4  # Number of oscillators
    N = fid.size
    timepoints = np.linspace(0, (N - 1) / sw, N)
    norm = nlinalg.norm(fid)
    normed_fid = fid / norm

    theta0[:M] /= norm  # Normalise the amplitudes
    theta0[2 * M : 3 * M] -= offset  # Shift the frequencies to center about 0

    opt_args = (normed_fid, timepoints, M)
    theta = optimize.minimize(
        fun=fidelity,
        x0=theta0,
        args=opt_args,
        method="trust-constr",
        jac=gradient,
        hess=hessian,
        options={"maxiter": max_iterations},
    )["x"]

    theta[:M] *= norm  # Re-scale the amplitudes
    theta[2 * M : 3 * M] += offset  # Re-correct the frequencies
    return theta


def fidelity(theta: np.ndarray, *args) -> float:
    """The cost function, ℱ(θ)."""
    y, timepoints, M = args
    Z = signal_pole_matrix(theta, M, timepoints)
    alpha = complex_amplitudes(theta, M)
    x = Z @ alpha
    residual = y - x
    return np.real(np.vdot(residual, residual))


def gradient(theta: np.ndarray, *args) -> np.ndarray:
    """The gradient, ∇ℱ(θ)."""
    y, timepoints, M = args
    N = y.size
    Z = signal_pole_matrix(theta, M, timepoints)
```

```
    alpha = complex_amplitudes(theta, M)
    # N x M array comprising M N-length vectors.
    # Each vector is the model produced by a single oscillator.
    x_per_osc = Z * alpha

    # First derivatives array
    d1 = np.zeros((N, 4 * M), dtype="complex")
    d1[:, :M] = amp_deriv(x_per_osc, theta, M)  # a
    d1[:, M : 2 * M] = phase_deriv(x_per_osc)  # φ
    d1[:, 2 * M : 3 * M] = freq_deriv(x_per_osc, timepoints)  # f
    d1[:, 3 * M :] = damp_deriv(x_per_osc, timepoints)  # η

    x = np.einsum("ij->i", x_per_osc)  # Sum all oscs to get model vector
    residual = y - x
    return -2 * np.real(d1.conj().T @ residual)


def hessian(theta: np.array, *args) -> np.ndarray:
    """The hessian, ∇²ℱ(θ)."""
    y, timepoints, M = args
    N = y.size
    Z = signal_pole_matrix(theta, M, timepoints)
    alpha = complex_amplitudes(theta, M)
    x_per_osc = Z * alpha

    # First derivatives
    d1 = np.zeros((N, 4 * M), dtype="complex")
    d1[:, :M] = amp_deriv(x_per_osc, theta, M)  # a
    d1[:, M : 2 * M] = phase_deriv(x_per_osc)  # φ
    d1[:, 2 * M : 3 * M] = freq_deriv(x_per_osc, timepoints)  # f
    d1[:, 3 * M :] = damp_deriv(x_per_osc, timepoints)  # η

    # Second derivatives
    d2 = np.zeros((N, 10 * M), dtype="complex")
    # aa: d2[:, :M] is trivially zero
    d2[:, M : 2 * M] = phase_deriv(d1[:, M : 2 * M])  # φφ
    d2[:, 2 * M : 3 * M] = freq_deriv(d1[:, 2 * M : 3 * M], timepoints)  # ff
    d2[:, 3 * M : 4 * M] = damp_deriv(d1[:, 3 * M :], timepoints)  # ηη
    d2[:, 4 * M : 5 * M] = phase_deriv(d1[:, :M])  # aφ
    d2[:, 5 * M : 6 * M] = phase_deriv(d1[:, 2 * M : 3 * M])  # φf
    d2[:, 6 * M : 7 * M] = damp_deriv(d1[:, 2 * M : 3 * M], timepoints)  # fη
    d2[:, 7 * M : 8 * M] = amp_deriv(d1[:, 2 * M : 3 * M], theta, M)  # af
```

```
    d2[:, 8 * M : 9 * M] = phase_deriv(d1[:, 3 * M :])  # φη
    d2[:, 9 * M :] = amp_deriv(d1[:, 3 * M :], theta, M)  # aη

    x = np.einsum("ij->i", x_per_osc)  # Sum all oscs to get model vector
    residual = y - x
    # See eqn S8b.
    # These are the 10 * M elements in the Hessian where the term
    # involving the second derivative is not trivially zero
    diagonals = -2 * np.real(np.einsum("ji,j->i", d2.conj(), residual))
    diagonal_idx = _diagonal_indices(M)

    hessian = np.zeros((4 * M, 4 * M))
    hessian[diagonal_idx] = diagonals

    main_diagonal = _displaced_diag_indices(4 * M, 0)
    hessian[main_diagonal] /= 2
    hessian += hessian.T
    hessian += 2 * np.real(np.einsum("ki,kj->ij", d1.conj(), d1))
    return hessian


def signal_pole_matrix(theta: np.ndarray, M: int, timepoints: np.ndarray) ->
    ↪ np.ndarray:
    return np.exp(
        np.outer(timepoints, 2j * np.pi * theta[2 * M : 3 * M] - theta[3 * M :])
    )


def complex_amplitudes(theta: np.ndarray, M: int) -> np.ndarray:
    return theta[:M] * np.exp(1j * theta[M : 2 * M])


def amp_deriv(array: np.ndarray, theta: np.ndarray, M: int) -> np.ndarray:
    return array / theta[:M]


def phase_deriv(array: np.ndarray) -> np.ndarray:
    return 1j * array


def freq_deriv(array: np.ndarray, timepoints: np.ndarray) -> np.ndarray:
    return np.einsum("ij,i->ij", array, 2j * np.pi * timepoints)
```

```


def damp_deriv(array: np.ndarray, timepoints: np.ndarray) -> np.ndarray:
    return np.einsum("ij,i->ij", array, -timepoints)


def _diagonal_indices(M: int) -> Tuple[np.ndarray, np.ndarray]:
    """Find Hessian locations with non-zero model second partial derivative."""
    idx0 = []
    idx1 = []
    for i in range(4):
        to_add = _displaced_diag_indices(4 * M, i * M)
        idx0.extend(to_add[0])
        idx1.extend(to_add[1])

    return idx0, idx1


def _displaced_diag_indices(n: int, k: int = 0):
    """Return the indices of an array's kth diagonal."""
    rows, cols = np.diag_indices(n)
    if k < 0:
        return rows[-k:], cols[:k]
    elif k > 0:
        return rows[:-k], cols[k:]
    else:
        return rows, cols


if __name__ == "__main__":
    # True parameter array (2 oscillators)
    theta = np.array([1.0, 2.0, 0.0, 0.0, 200.0, -300.0, 5.0, 6.0])
    # Initial guess array
    theta0 = np.array([0.9, 2.2, 0.1, -0.1, 202.0, -297.0, 5.3, 5.7])
    N = 2048
    sw = 5000.0
    offset = 0.0
    snr = 30.0
    # Make synthetic FID using the true parameter array
    fid = make_fid(theta, N, sw, offset, snr)
    result = numerical_optimisation(fid, theta0, sw, offset)
```

I NMR-EsPy Links

NMR-EsPy is an open source Python package. We welcome bug reports and feature requests. Please either submit an issue or pull request on the GitHub repository: https://github.com/foroozandehgroup/NMR-EsPy, or send an email to simon.hulse@chem.ox.ac.uk. The most recent version of the documentation exists in both HTML and PDF forms, and can be accessed through the following links:

- PDF: https://github.com/foroozandehgroup/NMR-EsPy/raw/gh-pages/nmr-espy.pdf
- HTML: https://foroozandehgroup.github.io/NMR-EsPy/

References

[1] Carl Eckart and Gale Young. "The approximation of one matrix by another of lower rank". eng. In: *Psychometrika* 1.3 (1936), pp. 211–218.

[2] Y. Pawitan. *In All Likelihood: Statistical Modelling and Inference Using Likelihood*. Oxford science publications. OUP Oxford, 2001.