py-rDCM, a Python Implementation of Regression Dynamic Causal Modeling

Xaver Funk

September 16, 2021

1 Introduction

Since its conception in the early 1990s [1, 2], functional magnetic resonance imaging (fMRI) has developed into a hallmark technique in the cognitive and systemic neurosciences [3]. The method has been employed extensively during experimental manipulations and behavioural tasks in order to identify task-related neuronal activity, as well as in the absence of any task or stimulus to investigate brain activity at rest (i.e. the resting-state) [4, 5]. At the core of fMRI lies the blood oxygen level dependent signal (BOLD-signal), that reflects a transient rise in oxygenated hemoglobin, commonly assumed to be linked to neural activity by neurovascular coupling [6, 7]. Combined with a comparably high spatial resolution this fuelled efforts into brain mapping, establishing functional segregation as a core organizational principle. In more recent years however, the focus has shifted towards functional integration, with efforts to assess the interaction or connectivity between distinct brain regions [8]. Broadly, such connectivity measures can be categorized into two approaches, termed functional and effective connectivity [9]. While the former mostly infers connectivity through correlations between BOLD-signals across regions, the latter aims to additionally provide insight into the directionality of a given connection, leading to its synonym "directed connectivity". Effective connectivity approaches therefore aim to infer a causal structure underlying a network of brain regions and their corresponding signals [9]. Often, these approaches are model-based, with the most prominent example being dynamic causal modeling (DCM) [10]. DCM has been employed to infer effective connectivity measures across a variety of tasks and was recently adapted to resting-state fMRI in the form of spectral DCM [11]. However, due to its high computational cost, a realm that so far has largely escaped DCM

approaches is whole-brain effective connectivity. Several contemporary efforts aim to extend a DCM-like approach to this domain, for example by constraining the prior covariance matrix of a spectral DCM with principal component analysis (PCA) [12]. Another such effort is regression DCM (rDCM), a novel technique that achieves an unparalleled computational efficiency by reformulating DCM as a Bayesian linear regression in the frequency domain [13].

Regardless of being in an early development stage, initial results suggest that rDCM is applicable to both large-scale networks and the resting state [14, 15]. In an era where the potential for clinical applicability of resting-state measures is assumed to be high, but not ripe yet [16, 17], there is an urgent need for more informative analysis techniques [18]. In this regard rDCM does provide hope, firstly by virtue of providing effective connectivity estimates that are presumably more informative than functional connectivity and secondly by having performed favorably when compared to functional connectivity in terms of test-retest reliability [19]. Therefore, rDCM might possibly be better suited as a computational assay to uncover pathophysiological mechanisms [20]. Though initial results are promising, rDCM is still lacking some major conceptual and practical features that ought to be developed in the future, such as accounting for region-wise endogenous fluctuations or variability in hemodynamics across regions and subjects (see section 3). Additionally, rDCM is currently implemented as part of the TAPAS toolbox (http://www.translationalneuromodeling.org/tapas) in MATLAB [21]. Though in itself an open-source toolbox, being dependent on the proprietary software MATLAB cuts TAPAS off from recent developments in community-driven neuroscience [22] that center around open science practices and mainly rely on python and its associated data-science stack for analyzing brain imaging data. Driven by collaborative and inclusive open-science practices, this community has been developing several frameworks and libraries for neuroimaging data in python, most notably nipype [23], nilearn [24], and nibabel (https://doi.org/10.5281/zenodo.4295521), as well as containerized command-line tools (BIDS-Apps) that rely on a standardized specification for neuroimaging datasets (BIDS) [25]. For example, fMRIprep [26] builds upon nipype's capability of interfacing with different established neuroimaging analysis tools to provide a preprocessing pipeline that leverages each software package's advantages. Tapping into this active community and their ecosystem could benefit rDCM threefold: First, it would become accessible to a wider range of users. Second, other researchers and developers could contribute more easily to the further development of rDCM, potentially addressing the aforementioned limitations. Third, it would benefit greatly from a more seamless integration into the community's available frameworks and analysis tools. In order for this to happen, a crucial

first step is to liberate rDCM from its current implementation's constraints imposed by the proprietary software MATLAB and translate it to python. Not only would this allow the wider open-neuroscience community to use and contribute to rDCM, but also provide further avenues of integration with the various data science, data visualization and machine learning tools that are readily available through open-source python libraries. This work will therefore focus on a python implementation of rDCM, making it more accessible and easier to use and customize.

2 Theory

2.1 DCM

Dynamic causal modeling (DCM) is a modeling technique that has been devised for and extensively applied to neuroimaging data, especially BOLD fMRI time series [10]. In this context, DCM aims to infer effective (i.e. directed) connectivity between regions in brain networks [9]. At its core, DCM provides a deterministic model of how observed BOLD time series arise from hidden states across interacting brain regions through connectivity patterns between them and experimental inputs (direct stimulation or behavioural task). The canonical DCM equation describes the rate of change of a (hidden) neuronal state vector x with a set of experimental inputs u as follows:

$$\frac{dx}{dt} = Ax + \sum_{i=1}^{K} u_i B^{(i)} x + Cu \tag{1}$$

With connectivity patterns being summarized in matrices A, B and C. Matrix A contains connectivity parameters between regions, commonly referenced as "fixed" or "endogenous" connections. B contains modulatory influences of any experimental input u_i on the parameters in A. Lastly, C contains direct influences of such an experimental input on the neuronal signal, which are usually referenced as "exogenous" connections. This equation can possibly be extended with a term comprising a matrix D that accounts for nonlinear effects inside a system of interacting brain regions, i.e. how the connection of two regions depends on activity in others [27].

These parameters are then approximated by minimizing the free energy in order to maximize model evidence given some data under a Variational Bayesian Laplace (VBL) framework (see [10] for details). In order to transform the hidden states x into a signal resembling the measured BOLD time series, the so-called Balloon model

is employed as a hemodynamic model describing the neurovascular coupling between neuronal activity and blood oxygenation [6, 7].

2.2 Regression DCM

In its essence, rDCM is a reformulation of classical DCM as a Bayesian linear regression in the frequency domain, allowing for rapid and large-scale inference on directed connectivity parameters [13]. This is accomplished by a number of modifications to the initial DCM framework. First, only endogenous (A-matrix) and exogenous (C-matrix) connections are considered, leaving out modulatory (B-matrix) and nonlinear effects (D-matrix). The resulting simplified state and observation equations are then transferred to the frequency domain by means of Fourier transformation. Further, the balloon model is replaced with a linear hemodynamic response function (HRF), removing nonlinearities in neurovascular coupling. Finally, a mean-field approximation is applied across regions, effectively rendering connectivity parameters targeting different regions independent of each other.

Together, these simplifications and transformations allow for the following general linear regression model (GLM) of the data y, given parameters θ, noise precision τ and design matrix X (for the complete derivation, see [13]):

$$\begin{aligned} p(Y|\theta,\tau,X) &= \prod_{r=1}^{R} N(Y_r; X\theta_r, \tau_r^{-1} I_{N\times N}) \\ Y_r &= (e^{2\pi i \frac{m}{N}} - 1)\frac{\hat{y_r}}{T} \\ X &= [\hat{y_1}, \hat{y_2}, \ldots, \hat{y_R}, \hat{h}\hat{u}_1, \hat{h}\hat{u}_2, \ldots, \hat{h}\hat{u}_K] \\ \theta &= [a_{r,1}, a_{r,2}, \ldots, a_{r,R}, c_{r,1}, c_{r,2}, \ldots, c_{r,K}] \end{aligned} \tag{2}$$

where Y_r is the derivative of the BOLD signal in region r and the parameters θ contain vectorized matrices A and C, i.e. the endogenous and exogenous connectivity parameters. The design matrix X contains as regressors for the GLM the signal of each region connected to region r, as well as a convolution of the inputs u entering region r with the HRF h, both in the frequency domain as indicated by the hat ($\hat{}$) symbol. Comparing this to equation 1, it is important to note that while canonical DCM is dealing with hidden sates x representing the actual neural signal that later gets transformed into BOLD signals using the balloon model, rDCM aims to directly explain the observed BOLD signal in a region as a linear mixture of the signals in connected regions $\hat{y}$ plus $\hat{h}\hat{u}$ contained in X. The seperation between hidden and

observed states is therefore given up. Additionally, through the use of a fixed HRF to map neuronal to BOLD signals, $\hat{y}$ now represents a convolution of x with a fixed hemodynamic response kernel in the Fourier domain. In practice, $\hat{y_r}$ corresponds to the observed, fourier-transformed signals in each region.

Lastly, for the full generative model Gaussian and Gamma priors are placed on the connectivity parameters θ and noise precision τ, respectively.

$$p(\theta_r) = \mathcal{N}(\theta_r; \mu_0^r, \Sigma_0^r) = (2\pi)^{-\frac{D_r}{2}} |\Sigma_0^r|^{-\frac{1}{2}} exp^{-\frac{1}{2}(\theta_r - \mu_0^r)^\top \Sigma_0^{r\,-1}(\theta_r - \mu_0^r)} \tag{3}$$
$$p(\tau_r) = Gamma(\tau_r; \alpha_0, \beta_0) = \frac{\beta_0^{\alpha_0}}{\Gamma(\alpha_0)} \tau_r^{\alpha_0 - 1} exp^{-\beta_0 \tau_r}$$

With Σ_0 and μ_0 as the covariance and mean of the Gaussian distribution on θ and α_0 and β_0 as the shape and rate parameters of the Gamma distribution on τ and Γ denoting the Gamma function. The Gaussian and Gamma act as conjugate priors, greatly simplifying the posterior. Under a VBL framework this allows for the definition of the following highly efficient iterative scheme for the sufficient statistics to approximate the posterior, owing to the aforementioned simplifications (the interested reader is referred to [13] for details on its derivation):

$$\Sigma_{\theta|y} = \Big(\frac{\alpha_{\tau|y}}{\beta_{\tau|y}} X^\top X + \Sigma_0^{-1}\Big)^{-1} \tag{4}$$
$$\mu_{\theta|y} = \Sigma_{\theta|y}\Big(\frac{\alpha_{\tau|y}}{\beta_{\tau|y}} X^\top Y + \Sigma_0^{-1}\mu_0\Big)^{-1} \tag{5}$$
$$\alpha_{\tau|y} = \alpha_0 + \frac{N}{2} \tag{6}$$
$$\beta_{\tau|y} = \beta_0 + \frac{1}{2}(Y - X_{\mu_{\theta|y}})^\top (Y - X)_{\mu_{\theta|y}}) + \frac{1}{2} tr(X^\top X \Sigma_{\theta|y}) \tag{7}$$

Here, $\Sigma_{\theta|y}$ and $\mu_{\theta|y}$ denote the posterior covariance and mean of the Gaussian distribution on θ and $\alpha_{\tau|y}$ and $\beta_{\tau|y}$ the posterior shape and rate parameters of the Gamma distribution on τ.

3 Limitations of rDCM

3.1 Conceptual Limitations

Due to rDCM being in an early development stage, there are still a number of conceptual limitations inherent to as mentioned in the original publications [13, 15, 14].

More specifically, rDCM currently fails to account for endogenous neuronal fluctuations that drive neural activity, which are especially important in the absence of exogenous input. Omitting these also leads to a loss of the separation between state and observation levels that are typical for classical DCMs. Furthermore, neither modulatory influences on coupling parameters between regions (B-matrix) nor nonlinear effects (D-matrix) are presently included in the model and present major challenges for future development. Additionally, rDCM doesn't currently account for individual and regional differences in the hemodynamic response, which could be solved by replacing the fixed HRF with a more flexible hemodynamic model. Lastly, resulting from its linear formulation and the stationarity assumption of the Fourier transform, rDCM is limited to stationary measures of effective connectivity. Considering that functional integration is subject to change on relatively short timescales [28], introducing dynamics to the effective connectivity estimates obtained by rDCM presents itself as a compelling opportunity for further development. In sum, there are several possible avenues to extend rDCM whose development could benefit from collaborative efforts.

3.2 Practical Limitations

There are a number of practical limitations of rDCM in its current form that mostly stem from its conserving compatibility with the neuroimaging data analysis toolbox SPM (https://www.fil.ion.ucl.ac.uk/spm/) and being implemented in MATLAB.

Firstly, similar to SPM, the DCM object is implemented as a MATLAB *structure array* (struct) carrying all necessary information, such as data, connectivity parameters and estimation options in different *fields*. During model inversion, this struct is then being passed around as input to different functions, possibly alongside other struct arrays encoding optional and additional arguments or containing outputs. These functions then perform some computations or modifications on either one or several inputs and return the modified struct arrays (see section 4 for details). This style of implementation is arguably rather convoluted and confusing, possibly hurting both the user's experience and learning curve. A more straightforward object-oriented programming (OOP) style could thus substantially simplify the user's experience by allowing for a simpler application programming interface (API) that is easier to learn and apply.

Additionally, MATLAB as a developing environment provides the user with a range of practical limitations. Firstly, being a proprietary software, MATLAB users face a substantial paywall to even get started. Secondly, MATLAB is a closed-source program, both limiting customizability and hindering a complete understanding of

the underlying algorithms and procedures used. Due to this, there is a gap between tools implemented in MATLAB and the currently prospering ecosystem of open-source development for neuroimaging applications native to python. As mentioned in the introduction, tapping into this ecosystem would benefit rDCM threefold: It would be available to more researchers, invite more developers to contribute to solving some of rDCM's conceptual limitations (see 3.1) and integrate more seamlessly into the growing sphere of python-based tools for neuroimaging. Taken together, rDCM could greatly benefit from not only a more straightforward execution of OOP, but also the community and ecosystem realized through python.

4 Implementation

In this section, the original implementation and functionality of rDCM in MATLAB (mat-rDCM) as well as in the python version (py-rDCM) will be detailed and their differences highlighted. All functions in this section will be referenced by their names, omitting the preceding `tapas_rdcm_`.

4.1 mat-rDCM

In order to maintain compatibility with SPM and conventional DCMs, the model is implemented as a MATLAB struct that stores all relevant information in specific fields. This information includes connectivity parameters, data, experimental inputs, estimation options, etc. Through the wrapper function `estimate`, this struct gets passed to different functions that use the information stored in its fields to perform model inversion and synthetic signal generation (see chapters 4.1.1 and 4.1.2). Additionally, `options`, `args` and `output` structs are supplied to some of these functions in order to encode additional parameters and flags or store outputs. These additional struct arrays may also be changed by a given function. How this works in more detail has been documented in both the code itself, as well as in an accompanying manual and will be briefly summarized below.

4.1.1 Core Loop

At the core of mat-rDCM lies the function `estimate`, a wrapper function that calls `set_options`, `create_regressors`, `ridge` and `compute_statistics`. As its name suggests, `set_options` creates the `options` struct that contains different settings for the analysis and also generates the fixed HRF stored in field `options.h` through a call to `get_convolution_bm`. While most default settings are unlikely to be changed

from their default values, optional settings can be passed to `set_options` via the optional argument `input_options`, which is then parsed for specific fields in order to overwrite the defaults. Most importantly, `type` indicates whether the DCM contains empirical or synthetic data, `SNR` specifies the signal-to-noise ratio for simulations and `y_dt` the repetition time (TR) of the data.

In `create_regressors`, the design matrix X containing a set of regressors and data Y are created from a BOLD signal. This includes their transformation to the frequency domain by means of fast Fourier transformation, convolution of the inputs U with the canonical HRF in `options.h`, high-pass filtering, removing noninformative frequencies from X and Y and differentiating Y. Taking X and Y as inputs, `ridge` first generates priors on connectivity parameters θ and noise precision τ through a call to `get_prior`. Then it performs model inversion, i.e. learning the parameters on θ and τ by implementing the variational Bayesian update equations 4 to 7.

Through a call to `store_parameters`, outputs are returned as a struct that contains the posterior estimates of parameters on θ and τ, the region-wise and total free energy and additional fields for saving the MAP (maximum-a-posteriori) estimates on connectivity parameters (`output.Ep`) and priors returned by `get_prior` (`output.priors`). Lastly, `compute_statistics` computes mean squared errors and sign errors (in case a ground truth is available), adding these to the output struct as `output.statistics`. Additionally, it calls `compute_signal`, itself calling `generate` to simulate region-wise BOLD signals using the estimated connectivity parameters (see 4.1.2). During this, two additional fields, `output.signal` and `output.residuals` are added to the output struct that contain the source signal with and without noise (in case it was generated), the estimated signal plus their derivatives and mean squared errors between them, respectively.

4.1.2 Signal Generation

rDCM includes functionality to generate neuronal signals and BOLD time series, which lends itself to two use cases. First, synthetic data can be created from ground truth to later test the model's capability to recover the underlying parameters. Second, time series created on estimated model parameters can be compared to empirical data to assess model fit. In mat-rDCM, this functionality is implemented as a numerical integrator in `dcm_euler_integration.c` that is compiled to a MEX function capable of interfacing with MATLAB through its native C Matrix API. The integrator takes the matrices A, B, C and D, plus a list of hemodynamic parameters as input and calculates time series of the neuronal activity (`x_out`), vasodilatory signal (`s_out`), blood flow (`f_out`), blood volume (`v_out`) and deoxyhemoglobin content

(`q_out`) according to the classical DCM forward model. The resulting quantities can be used to calculate the BOLD response y by either convolving `x_out` with the canonical HRF or according to:

$$y_{r,i} = V_0\big(k_1(1 - q_{r,i}) + k_2(1 - \frac{q_{r,i}}{v_{r,i}}) + k_3(1 - v_{r,i})\big) \tag{8}$$

Where $y_{r,i}$, $q_{r,i}$ and $v_{r,i}$ are the BOLD signal, deoxyhemoglobin content and blood volume at timestep i in region r. V_0 is the resting venous blood volume and coefficients $k1$, $k2$ and $k3$ are dimensionless quantities dependent on experimental and physiological parameters. This is implemented in the function `euler_gen` that is called by either `get_convolution` for generating a simple HRF or `generate` for generating a synthetic BOLD signal.

4.2 py-rDCM

This section concerns rDCM as implemented in python (py-rDCM) and submitted along with this report. It is crucial to clarify that py-rDCM does not include sparse rDCM as detailed in [14].The conversion to python enabled and motivated a few changes in programming style and functionality that will be detailed in the following sections.

4.3 Differences

4.3.1 Object-oriented Programming

In py-rDCM, the need to maintain compatibility with SPM is not an issue anymore. This allows a more straightforward implementation of the core model under the object-oriented programming framework, that is natively supported by python's `class` constructor. More precisely, the core model is implemented as a `class` called `DCM`, following mat-rDCM's nomenclature. It follows that all functions taking the DCM struct as input in mat-rDCM, are implemented as methods acting on the `DCM` class in py-rDCM. Further, experimental data, inputs, priors and optional parameters are configured as attributes of this class. Similarly, results are saved as attributes as well, see section 4.3.3. While the inputs `U` and data `Y` are supported through their own `class` objects, priors, options, and the results are implemented as `Bunch` objects as provided by the python library scikit-learn [29]. `Bunch` objects are essentially an extension of dictionaries making values accessible by both key and attribute, allowing for maximal syntactical flexibility.

Taken together, this allows for easier usage and provides a simple API that is similar to the canonical way of how ML models are used in the python community.

4.3.2 Signal Generation

For the python implementation, the function `dcm_euler_integration.c` was stripped of data types and functions specific to the C Matrix API and implemented as a dynamic link library that is called from python through `euler_gen` via a wrapper provided in `wrap_euler.py`. While debugging, it was discovered that some of the default settings from the original call to the integrator were superfluous. More specifically, the last two entries of the list of parameters passed to the integrator were set to 1, thereby causing the program to loop over the B and D matrices that are currently not supported for rDCM. Nevertheless, B and D matrices are provided in the rDCM tutorial containing only 0s, thus not exerting any effect on the resulting time series. Setting the aforementioned two parameters to 0 instructs the integrator to ignore these empty matrices, resulting in a 5-fold decrease in time for signal generation duing the tutorial in py-rDCM as compared to mat-rDCM. Other performance tests on this most computationally costly section of the code indicate approximately similar performance. More specifically, the time needed for generating a simple HRF through `get_convolution` for one region and one input were below a millisecond for both versions of rDCM.

4.3.3 Further Improvements and Differences

Several improvements were made possible by virtue of built-in python functionality or its arguably more concise syntax that is either not supported by MATLAB or was not made use of in mat-rDCM. For example, default arguments are not supported in MATLAB, yet quite concise in python and were included in py-rDCM where appropriate. Additionally, optional argument handling is greatly simplified by python's `**kwargs` argument (e.g. in `set_options`). Furthermore, some custom functions included in mat-rDCM that were written specifically for their use in SPM or the TAPAS toolbox were replaced with equivalents from diligently maintained, well-documented and highly optimized python libraries, further reducing the py-rDCM codebase. In this sense, the function `spm_logdet` that computes the logarithm of the determinant of a matrix has been replaced by `numpy.linalg.slogdet`. Likewise, the function `ep2par` simply vectorizes a matrix, which is easily achieved using numpy's `array.flatten`. In a similar vein, `empty_par` is not needed in py-rDCM, due to the aforementioned `Bunch` object taking over its use case. Furthermore, various deprecated features such as the `padding` option have been removed, while others were

deemed superfluous during translation to python, for example code snippets used to compare rDCM to conventional DCM for past publications.

Another minor difference concerns the results being stored as object attributes in py-rDCM, since the output struct has been removed. The posterior estimates of the parameters on τ and θ, along with MAP estimates on connectivity parameters A, C and influences of confound regressors and the region-wise and total free energy are stored in `DCM.posterior`. `DCM.signal` holds the source signal with and without noise (in case it was generated), the estimated signal and their derivatives, while `DCM.residuals` holds mean squared errors between them. Lastly, `DCM.statistics` contains the outputs of `compute_statistics`, i.e. the mean squared errors and sign errors on connectivity parameters, in case a ground truth is available.

Lastly, there are some minor differences between the specific matrix operations under python and matlab that have been discovered during translation and might be intriguing for the interested reader. Most curiously, transposing a complex-valued matrix with the `'` operator in MATLAB returns the conjugate transpose by default, but in numpy requires `array.conjugate()` before applying `.T`. Another peculiar detail concerns the standard deviation of an array: Replicating the output from MATLAB's `std(array)`, requires the optional argument `ddof=1` in `numpy.std(array)` which defines the delta degrees of freedom used in the calculation.

Taken together, various large-scale and small optimizations to the original mat-rDCM code provide py-rDCM with a simpler API and more concise style of implementation.

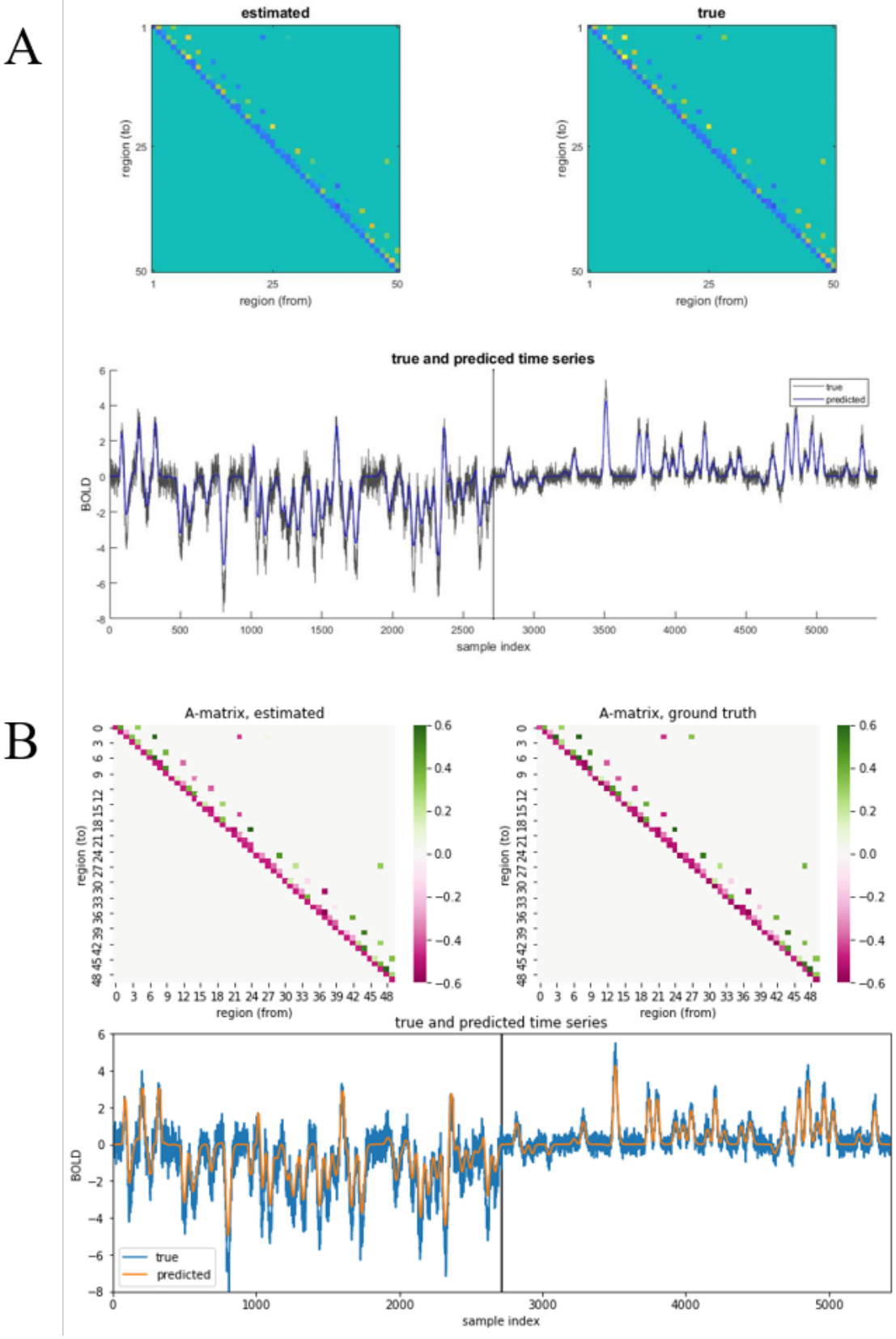

Figure 1: Tutorial outputs of (A) mat-rDCM and (B) py-rDCM

5 Validation

5.1 Replicating the mat-rDCM Tutorial

Inside the original mat-rDCM code, a tutorial has been provided that includes a synthetic large-scale DCM detailed in [30] that has been used to establish face validity for sparse rDCM in [14]. It comprises 50 regions that are connected in a sparse manner (see figure 1 A, top-right). This tutorial has been adapted here to show that py-rDCM is functionally equivalent to mat-rDCM. To this end, the code has been adapted in a way that it performs "regular", not sparse rDCM model inversion. During the tutorial, rDCM first generates synthetic data for the provided ground truth, then estimates connectivity parameters from them and generates a signal based on the estimated parameters. Ground truth is then compared to the estimates graphically (see figure 1 A). It is important to note here that this should not be considered a reliable validation of rDCM's capabilities per se, since it uses extremely informative priors, a benign signal-to-noise ratio and generous TR, and data are tailored for rDCM, since they are generated based on A and C matrices. Rather, this tutorial is used merely for the purpose of validating the py-rDCM implementation in the sense that its results are equivalent to mat-rDCM. To this end, the tutorial has been implemented using py-rDCM. As detailed in figure 1 B, py-rDCM replicates faithfully both signal generation and parameter estimation for this tutorial. The code reproducing the graphs in figure 1 A using py-rDCM is provided along with this submission (see X).

5.2 Testing py-rDCM on real data

In order to establish py-rDCM's applicability to real data, it has been employed on a single-subject resting-state BOLD fMRI run and compared to canonical functional connectivity as assessed by Pearson correlation. Resting-state data has been chosen since this will be py-rDCM's primary use case during the upcoming months.

Resting-state fMRI data was acquired from the first subject of a freely available online dataset [31, 32] and preprocessed using fMRIprep [26] with default settings. Subsequently, BOLD signals were extracted for 39 regions according to a brain atlas derived from multi-subject dictionary learning [33] and functional connectivity (Pearson correlation) was computed with `nilearn.connectome.ConnectivityMeasure` and setting `kind="correlation"`, according to [34]. Additionally, connectivity parameters (A-matrix) were calculated with py-rDCM with fully connected priors on A, as well as settings for resting state, i.e. an empty input matrix U and `filtu=0`.

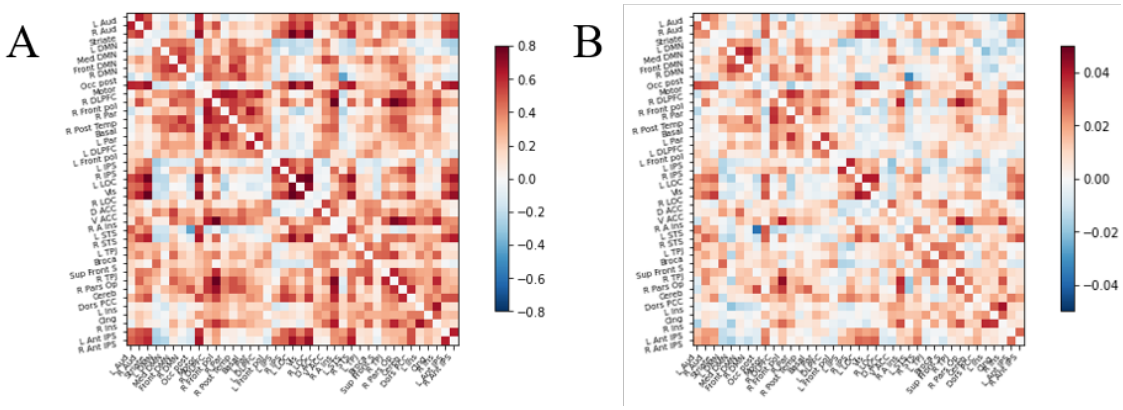

Figure 2: Resting-state connectivity derived from (A) functional connectivity and (B) py-rDCM

As shown in figure 2, there is substantial overlap between the two approaches and both make sense in terms of functional networks. For example, all nodes related to the default-mode network are highly connected. It is also reassuring that the range of rDCM coupling parameters is approximately the same as for the ones reported in [15]. Note however, that the reported connectivity matrices in figure 2 should not be taken as a definitive treatment of functional connectivity, since no possible confounds have been regressed out of the signal. Moreover, the attentive reader might notice that the connectivity parameters as calculated by rDCM seem too symmetric to plausibly resemble the promised *directed* connectivity. Additionally, some parameters (e.g. between left and right auditory cortices) are biologically implausible, yet obviously result from their statistical dependencies and the model structure. Whether these issues could be mediated by informed structural priors on A, or whether rDCM should rather be considered a Bayesian version of functional connectivity are topics to be explored in the future.

6 Future Developments

In the future, several additional improvements can be made, adding to py-rDCM's functionality and practicality. First of all, compatibility of rDCM with the most common operating systems has to be ensured to secure widespread availability. Currently, py-rDCM's signal generation functionality is only supported on Linux machines. Providing support on Windows and Mac OS would entail compiling the shared c library that performs signal integration against their respective runtimes. An alternative

way would be to containerize py-rDCM with the help of tools such as docker or singularity. This would not only circumvent the operating system compatibility issue, but also ensure that all necessary dependencies are maintained.

With respect to functionality, any solutions to the conceptual limitations mentioned in 3.1 are to be considered as possible extensions and may hereafter be tackled by anyone interested in contributing to the project. Besides, adding the option of sparsity constraints (see [14]) to py-rDCM represents another obvious next step in development. Apart from that, several additional tools are conceivable that would represent valuable additions to py-rDCM. As such, an automatic tool for region parcellation of brain images would be especially useful, allowing users to specify the desired parcellation for whole-brain effective connectivity as an optional argument to `estimate`. Moreover, a plotting tool using state-of-the-art visualization libraries represents another valuable avenue of future development. Both of these will likely be developed in the course of py-rDCM's first applications to real-world datasets during the following months. Lastly, considering the relatively small amount of testing that the code has so far undergone, py-rDCM most probably still contains lots of bugs that need to be discovered and fixed. Along these lines of code quality, an adequate amount of input checking and related warnings are also desirable for a more pleasurable user experience.

7 Conclusion

In conclusion, py-rDCM provides a free, open-source and collaborative alternative for researchers interested in using or contributing to regression DCM. Being liberated from MATLAB's closed-source and proprietary constraints, py-rDCM is fully customizable, extendable and able to tap into the currently growing open-neuroscience community that is forming itself around open and free software, data and research practices. Additionally, several small improvements facilitate the user's experience by enabling a more straightforward implementation and interface. In the future, py-rDCM can be further improved and extended in a collaborative manner to tackle its inherent limitations or enhance its practicability.

References

[1] J W Belliveau, D N Kennedy, Jr, R C McKinstry, B R Buchbinder, R M Weisskoff, M S Cohen, J M Vevea, T J Brady, and B R Rosen. Functional

mapping of the human visual cortex by magnetic resonance imaging. *Science*, 254(5032):716–719, November 1991.

[2] K K Kwong, J W Belliveau, D A Chesler, I E Goldberg, R M Weisskoff, B P Poncelet, D N Kennedy, B E Hoppel, M S Cohen, and R Turner. Dynamic magnetic resonance imaging of human brain activity during primary sensory stimulation. *Proc. Natl. Acad. Sci. U. S. A.*, 89(12):5675–5679, June 1992.

[3] Bruce R Rosen and Robert L Savoy. fMRI at 20: Has it changed the world? *Neuroimage*, 62(2):1316–1324, August 2012.

[4] Marcus E Raichle, Ann Mary MacLeod, Abraham Z Snyder, William J Powers, Debra A Gusnard, and Gordon L Shulman. A default mode of brain function. *Proceedings of the National Academy of Sciences*, 98(2):676–682, 2001.

[5] Michael D Fox and Marcus E Raichle. Spontaneous fluctuations in brain activity observed with functional magnetic resonance imaging. *Nat. Rev. Neurosci.*, 8(9):700–711, September 2007.

[6] Richard B. Buxton, Eric C. Wong, and Lawrence R. Frank. Dynamics of blood flow and oxygenation changes during brain activation: The balloon model. *Magnetic Resonance in Medicine*, 39(6):855–864, 1998.

[7] Richard B Buxton, Kâmil Uludağ, David J Dubowitz, and Thomas T Liu. Modeling the hemodynamic response to brain activation. *Neuroimage*, 23 Suppl 1:S220–33, 2004.

[8] H Lv, Z Wang, E Tong, L M Williams, G Zaharchuk, M Zeineh, A N Goldstein-Piekarski, T M Ball, C Liao, and M Wintermark. Resting-State functional MRI: Everything that nonexperts have always wanted to know. *AJNR Am. J. Neuroradiol.*, 39(8):1390–1399, August 2018.

[9] Karl J Friston. Functional and effective connectivity: a review. *Brain Connect.*, 1(1):13–36, 2011.

[10] K J Friston, L Harrison, and W Penny. Dynamic causal modelling. *Neuroimage*, 19(4):1273–1302, August 2003.

[11] Karl J Friston, Joshua Kahan, Bharat Biswal, and Adeel Razi. A DCM for resting state fMRI. *Neuroimage*, 94:396–407, July 2014.

[12] Adeel Razi, Mohamed L Seghier, Yuan Zhou, Peter McColgan, Peter Zeidman, Hae-Jeong Park, Olaf Sporns, Geraint Rees, and Karl J Friston. Large-scale DCMs for resting-state fMRI. *Netw Neurosci*, 1(3):222–241, October 2017.

[13] Stefan Frässle, Ekaterina I Lomakina, Adeel Razi, Karl J Friston, Joachim M Buhmann, and Klaas E Stephan. Regression DCM for fMRI. *Neuroimage*, 155:406–421, July 2017.

[14] Stefan Frässle, Ekaterina I Lomakina, Lars Kasper, Zina M Manjaly, Alex Leff, Klaas P Pruessmann, Joachim M Buhmann, and Klaas E Stephan. A generative model of whole-brain effective connectivity. *Neuroimage*, 179:505–529, October 2018.

[15] Stefan Frässle, Samuel J Harrison, Jakob Heinzle, Brett A Clementz, Carol A Tamminga, John A Sweeney, Elliot S Gershon, Matcheri S Keshavan, Godfrey D Pearlson, Albert Powers, and Klaas E Stephan. Regression dynamic causal modeling for resting-state fMRI. *Hum. Brain Mapp.*, (hbm.25357), February 2021.

[16] Karsten Specht. Current challenges in translational and clinical fMRI and future directions. *Front. Psychiatry*, 10:924, 2019.

[17] Edgar Canario, Donna Chen, and Bharat Biswal. A review of resting-state fMRI and its use to examine psychiatric disorders. *psychoradiology*, 1(1):42–53, May 2021.

[18] Andrew T Reid, Drew B Headley, Ravi D Mill, Ruben Sanchez-Romero, Lucina Q Uddin, Daniele Marinazzo, Daniel J Lurie, Pedro A Valdés-Sosa, Stephen José Hanson, Bharat B Biswal, Vince Calhoun, Russell A Poldrack, and Michael W Cole. Advancing functional connectivity research from association to causation. *Nat. Neurosci.*, 22(11):1751–1760, November 2019.

[19] Stefan Frässle and Klaas E Stephan. Test-retest reliability of regression dynamic causal modeling. June 2021.

[20] Stefan Frässle, Yu Yao, Dario Schöbi, Eduardo A Aponte, Jakob Heinzle, and Klaas E Stephan. Generative models for clinical applications in computational psychiatry. *Wiley Interdiscip. Rev. Cogn. Sci.*, 9(3):e1460, May 2018.

[21] MATLAB. *9.7.0.1190202 (R2021a)*. The MathWorks Inc., Natick, Massachusetts, 2021.

[22] Rémi Gau, Stephanie Noble, Katja Heuer, Katherine L Bottenhorn, Isil P Bilgin, Yu-Fang Yang, Julia M Huntenburg, Johanna MM Bayer, Richard AI Bethlehem, Shawn A Rhoads, et al. Brainhack: Developing a culture of open, inclusive, community-driven neuroscience. *Neuron*, 109(11):1769–1775, 2021.

[23] Krzysztof J. Gorgolewski, Oscar Esteban, Christopher Burns, Erik Ziegler, Basile Pinsard, Cindee Madison, Michael Waskom, David Gage Ellis, Dav Clark, Michael Dayan, Alexandre Manhães-Savio, Michael Philipp Notter, Hans Johnson, Blake E Dewey, Yaroslav O. Halchenko, Carlo Hamalainen, Anisha Keshavan, Daniel Clark, Julia M. Huntenburg, Michael Hanke, B. Nolan Nichols, Demian Wassermann, Arman Eshaghi, Christopher Markiewicz, Gael Varoquaux, Benjamin Acland, Jessica Forbes, Ariel Rokem, Xiang-Zhen Kong, Alexandre Gramfort, Jens Kleesiek, Alexander Schaefer, Sharad Sikka, Martin Felipe Perez-Guevara, Tristan Glatard, Shariq Iqbal, Siqi Liu, David Welch, Paul Sharp, Joshua Warner, Erik Kastman, Leonie Lampe, L. Nathan Perkins, R. Cameron Craddock, René Küttner, Dmytro Bielievtsov, Daniel Geisler, Stephan Gerhard, Franziskus Liem, Janosch Linkersdörfer, Daniel S. Margulies, Sami Kristian Andberg, Jörg Stadler, Christopher John Steele, William Broderick, Gavin Cooper, Andrew Floren, Lijie Huang, Ivan Gonzalez, Daniel McNamee, Dimitri Papadopoulos Orfanos, John Pellman, William Triplett, and Satrajit Ghosh. Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in Python. 0.12.0-rc1, April 2016.

[24] Alexandre Abraham, Fabian Pedregosa, Michael Eickenberg, Philippe Gervais, Andreas Mueller, Jean Kossaifi, Alexandre Gramfort, Bertrand Thirion, and Gael Varoquaux. Machine learning for neuroimaging with scikit-learn. *Frontiers in Neuroinformatics*, 8:14, 2014.

[25] Krzysztof J Gorgolewski, Tibor Auer, Vince D Calhoun, R Cameron Craddock, Samir Das, Eugene P Duff, Guillaume Flandin, Satrajit S Ghosh, Tristan Glatard, Yaroslav O Halchenko, Daniel A Handwerker, Michael Hanke, David Keator, Xiangrui Li, Zachary Michael, Camille Maumet, B Nolan Nichols, Thomas E Nichols, John Pellman, Jean-Baptiste Poline, Ariel Rokem, Gunnar Schaefer, Vanessa Sochat, William Triplett, Jessica A Turner, Gaël Varoquaux, and Russell A Poldrack. The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. *Scientific Data*, 3(1):1–9, June 2016.

[26] Oscar Esteban, Christopher J Markiewicz, Ross W Blair, Craig A Moodie, A Ilkay Isik, Asier Erramuzpe, James D Kent, Mathias Goncalves, Elizabeth

DuPre, Madeleine Snyder, Hiroyuki Oya, Satrajit S Ghosh, Jessey Wright, Joke Durnez, Russell A Poldrack, and Krzysztof J Gorgolewski. fMRIPrep: a robust preprocessing pipeline for functional MRI. *Nat. Methods*, 16(1):111–116, January 2019.

[27] Klaas Enno Stephan, Lars Kasper, Lee M Harrison, Jean Daunizeau, Hanneke EM den Ouden, Michael Breakspear, and Karl J Friston. Nonlinear dynamic causal models for fmri. *Neuroimage*, 42(2):649–662, 2008.

[28] Klaas E Stephan, Sandra Iglesias, Jakob Heinzle, and Andreea O Diaconescu. Translational perspectives for computational neuroimaging. *Neuron*, 87(4):716–732, August 2015.

[29] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[30] Stephen M Smith, Diego Vidaurre, Christian F Beckmann, Matthew F Glasser, Mark Jenkinson, Karla L Miller, Thomas E Nichols, Emma C Robinson, Gholamreza Salimi-Khorshidi, Mark W Woolrich, Deanna M Barch, Kamil Uğurbil, and David C Van Essen. Functional connectomics from resting-state fMRI. *Trends Cogn. Sci.*, 17(12):666–682, December 2013.

[31] Yana Panikratova, Alexander Tomyshev, Ekaterina Pechenkova, and Roza Vlasova. "fmri: resting state and arithmetic task", 2021.

[32] Yana Panikratova, Olga Dobrushina, Alexander Tomyshev, Tatiana Akhutina, Ekaterina Pechenkova, Valentin Sinitsyn, and Roza Vlasova. Context-dependency in the cognitive bias task and resting-state functional connectivity of the dorsolateral prefrontal cortex. *J. Int. Neuropsychol. Soc.*, 26(8):749–762, September 2020.

[33] Gael Varoquaux, Alexandre Gramfort, Fabian Pedregosa, Vincent Michel, and Bertrand Thirion. Multi-subject dictionary learning to segment an atlas of brain spontaneous activity. In Gábor Székely and Horst K. Hahn, editors, *Information Processing in Medical Imaging*, pages 562–573, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[34] Gaël Varoquaux, Flore Baronnet, Andreas Kleinschmidt, Pierre Fillard, and Bertrand Thirion. Detection of brain functional-connectivity difference in post-stroke patients using group-level covariance modeling. In Tianzi Jiang, Nassir

Navab, Josien P. W. Pluim, and Max A. Viergever, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2010*, pages 200–208, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.