A control theoretic approach to evaluate and inform ecological momentary interventions

Instructions for code usage

May 22, 2024

If you find the repository and/or corresponding paper helpful for your own research, please cite our work.

1 Introduction

In the main article, we employ concepts and measures from network control theory (NCT) to formally quantify and analyze proximal intervention effects on an individual's mental state as measured by time series of ecological momentary assessment (EMA) variables. This GitHub repository contains the complete code used for performing the analyses mentioned, and an additional tutorial notebooks. For instructional purposes, we use random synthetic time series resembling the original data. To reproduce the figures and statistical tests, the original data is available on reasonable request to the corresponding author.

2 Usage guide

2.1 Setting up the environment

The code requires only standard python packages, as specified in the `requirements.txt` file. We recommend that you set up a python environment specifically for this code. If you're using Windows, this is done from the Power Shell by

```
python -m venv .venv
.venv/Scripts/activate.ps1
pip install -r requirements.txt
```

2.2 File overview

- `tutorial.ipynb`

  Jupyter notebook that guides the user through the analyses performed in the paper, step by step, using a randomly generated synthetic EMA time series. You can easily replace the synthetic data with your own.

- `ctrl/discrete_optimal_control.py`

  Module containing all control theoretic methods.

- `ctrl/control_strategies.py`

  Module containing input selection strategies.

- `ctrl/utils.py`

  Module containing additional resources for data loading, plotting, model fitting, and statistical testing.

- `figure<x>.ipynb`

  Notebook that produces figure ¡x¿ from the paper. Each notebook contains a global variable `TEST`. If `TEST==False`, the original analyses are performed and the plots are drawn as they appear in the paper, provided the original data files are placed in the directory `data`. Figure panels are saved to `figures/figure<x><panel>.png`. If `TEST==True`, analyses are carried out with randomly generated synthetic data, and figures are not saved.

- `custom_rcparams_paper.py`

  Plotting parameters.

- `model_distribution.py`

  Contains parameters of the distribution from which a model is drawn to create synthetic data.

2.3 General remarks regarding the code

- For all time series variables, the first dimension is assumed to be time.
- A is assumed to be the system matrix of a discrete-time linear system. Hence, it is an $n \times n$ matrix, where n is the number of system variables.
- B is assumed to be the input matrix of a discrete-time linear system. Hence, it is an $n \times m$ matrix, where m is the number of input variables.
- The discrete-time linear system, i.e. vector auto-regressive model of order 1 (VAR(1)) without bias term, is

$$\hat{x}_{t+1} = Ax_t + Bu_t$$

2.4 Methods in `ctrl/discrete_optimal_control.py`

- `seminorm(M, x)`

  Calculates $x^\top Mx$ for a matrix $M \in \mathbb{R}^{n \times n}$ and a vector $x \in \mathbb{R}^n$. If x is a $k \times n$ matrix, each row of x is treated as a separate vector.

- `step(A, B, x, u=None)`

  Given a system matrix A, an input matrix B, a state vector $x \in \mathbb{R}^n$ and an input vector $u \in \mathbb{R}^m$, calculate the prediction of a VAR(1)-model without bias:

$$\hat{x_{t+1}} = Ax_t + Bu_t$$

  If x is a $T \times n$ matrix and u is a $T \times m$ matrix, each row will be treated as a separate time step. If u is left empty, it is set to a vector of 0.

- `evolve(A, B, T, x0, u=None)`

  Propagate vector $x_0 \in \mathbb{R}^n$ T times forward with `step`.

- `ctrb(A, B, T=None)`

  Calculate the T-step controllability matrix

$$C_T = \begin{bmatrix} B & AB & AAB & \dots & A^{T-1}B \end{bmatrix}$$

  If T is left empty, it is set to n, the number of system variables.

- `ctrb_gramian(A, B, T)`

  Calculate the t-step controllability Gramian

$$W_T = \sum_{t=0}^{T-1} A^t BB^\top (A^\top)^t.$$

  For $T = \infty$, this is the solution to the discrete $(A, BB^\top)$-Lyapunov equation.

- `controllable(A, B, T=None)`

  Check if the system is controllable in T time steps, by asserting that the controllability matrix has full rank.

- `average_ctrb(A, T=np.inf, B=None, per_column_of_B=True)`

  Calculates the average controllability (AC), the trace of the controllability Gramian,

  $$\mathrm{AC}(A, B, T) = \mathrm{tr} W_T.$$

  If B is left empty, it is set to the $n \times n$ identity matrix to allow for inputs on every single system variable. If `per_column_of_B` is true, the AC is calculated for every column of B separately, resulting in an m-sized vector of ACs.

- `modal_ctrb(A)`

  Calculates the modal controllability (MC) for system matrix A,

  $$\mathrm{MC}(A) = \sum_{k=1}^{n} (1 - |\lambda_k|^2) w_k^2$$

  where λ_k is the k-th eigenvalue of A, and w_k the corresponding eigenvector.

- `cumulative_impulse_response(A, b, T)`

  Calculates the T-step cumulative impulse response (CIR) for system matrix A and perturbation vector $b \in \mathbb{R}^n$

  $$\mathrm{CIR}_T(A, b) = \sum_{t=0}^{T-1} A^t b.$$

  Note that, if you have a discrete-time linear system, $b = Bu$ for some input vector $u \in \mathbb{R}^m$.

- `kalman_gain(A, B, Q, R)`

  Calculates the Kalman gain matrix $G \in \mathbb{R}^{m \times n}$ used for the Linear Quadratic Regulator (LQR) control law. For a discrete-time linear system, $u_t = -Gx_t$ is the optimal control input, in that it minimizes the quadratic loss function $L = x_T^\top Q x_T \sum_{t=1}^{T-1} x_t^\top Q x_t + u_t^\top R u_t$. Hence, it drives the state x_t towards 0, i.e. it stabilizes the system.

  Matrix $Q \in \mathbb{R}^{n \times n}$ must be symmetric and positive semidefinite. It defines the importance of convergence speed of x_t towards 0 in the optimization. Matrix $R \in \mathbb{R}^{m \times m}$ must be symmetric and positive definite. It defines the importance of control energy in the optimization.

- `closed_loop_optimal_control(A, B, Q, R, X)`

  Using `kalman_gain`, calculates the LQR optimal control input for a time series $X \in \mathbb{R}^{T \times n}$ given the model matrix A, input matrix B, convergence speed weight matrix Q and control energy weight matrix R.

- `feedback_feedforward_gain(A, B, Q, R)`

  Calculates the Kalman gain matrix $G \in \mathbb{R}^{m \times n}$ and feedforward gain matrix $F \in \mathbb{R}^{m \times n}$ used for the Linear Quadratic Tracker control law. In contrast to the LQR, the control input $u_t = -Gx_t + Fv_{t+1}$ minimizes the loss function $L = (x_T - r_T)^\top Q (x_T - r_T) \sum_{t=1}^{T-1} (x_T - r_t)^\top Q (x_T - r_t) + u_t^\top R u_t$. Hence, it drives the system state x_t closer to a reference trajectory r_t. v_{t+1} is called the adjoint and must be calculated beforehand. It encodes information from the reference trajectory.

- `tracking_optimal_control(A, B, Q, R, X, Yref, save_memory=False)`

  Calculates the tracker optimal control input for a time series $X \in \mathbb{R}^{T \times n}$ designed to drive it towards reference trajectory $Yref \in \mathbb{R}^{T \times n}$. Internally, it calculates the adjoint v_t and uses `feedback_feedforward_gain`. If `save_memory` is true, the adjoint won't be saved but iteratively re-computed for each step, which slightly reduces exactness but lowers memory consumption.

2.5 Methods in `ctrl/control_strategies.py`

- `optimal_control_strategy(data, inputs, target_state, admissible_inputs, rho, online=True)`

  For a `data` and `inputs` time series, returns a suggested control input according to the optimal control strategy. First, it fits a discrete-time linear model to the time series. Using this model, it calculates the tracking optimal control which brings the system state closer to the `target_state`, then selects the row of `admissible_inputs` closest to the optimal control. The optimal control is calculated with $Q = I$, the identity, and $R = \rho I$. If `online` is true, only the last selected input is returned, assuming that this will be administered immediately. Else, the complete selected input sequence is returned.

- `brute_force_strategy(data, inputs, target_state, admissible_inputs, time_horizon, rho, online=True)`

  For a `data` and `inputs` time series, returns a suggested control input according to the brute force strategy. First, it fits a discrete-time linear model to the time series. Using this model, it iterates through all sequences of `admissible_inputs` of length `time_horizon` and predicts their outcome. It selects the input which brings the system state closer to the `target_state` over the complete time horizon. If `online` is true, only the last selected input is returned, assuming that this will be administered immediately. Else, the complete selected input sequence is returned.

- `max_ac_strategy(data, inputs, admissible_inputs, online=True)`

  For a `data` and `inputs` time series, returns a suggested control input according to the max AC strategy. First, it fits a discrete-time linear model to the time series. Using this model, it calculates the average controllability (AC) of all `admissible_inputs`. It selects the input with highest AC. If `online` is true, only the last selected input is returned, assuming that this will be administered immediately. Else, the complete selected input sequence is returned.

2.6 Methods in `ctrl/utils.py`

- `load_data`

  Loads the EMIcompass dataset from a hard-coded location. Must be adjusted by the user.

- `generate_dataset(data, inputs, data_labels, input_labels)`

  Creates a dataset from a $T \times n$ state time series `data`, a $T \times m$ input time series `inputs`, and a list of labels for data and inputs, respectively. The dataset will be structured equivalently to the EMIcompass dataset, such that it can be used by the notebooks in this repository.

- `generate_random_dataset(N, T, seed)`

  Creates a random dataset with N subjects and time series length T for the purpose of testing the code. For details on how the data is generated, see 4.

- `stable_ridge_regression(X, Inp, intercept=False, accepted_eigval_threshold=1, max_regularization=10.5)`

  Given a data time series and an input time series, estimates the system matrix A and input matrix B using regularized least squares. The regularization factor $\lambda \geq 0$ is chosen to be the smallest value such that $|\lambda_{\max}| < \Lambda$. $\lambda_{\max}$ is the eigenvalue of A which has maximum absolute value, while Λ is the `accepted_eigval_threshold`. By setting $\Lambda = 1$, the system becomes stable. λ cannot be greater than `max_regularization`. Returns A, B, and λ. If `intercept` is set to true, the model can have an intercept term, which is returned additionally.

- `cohens_d(x, y, paired=False, correct=False)`

  Calculates the effect size (Cohen's d) between samples x and y. If `paired` is true, the samples are assumed to be paired and must have the same size. If `correct` is true, calculates Hedge's g, a version of d corrected for sample size:

$$g = d \cdot \left(1 - \frac{3}{4(n_x + n_y) - 9}\right)$$

  where n_x and n_y are the respective sample sizes. See the original paper for details.

- `trace(M)`

  The trace of square matrix M.

- `round_to_vector(M, Z, p_norm=2)`

  Round each row of $M \in \mathbb{R}^{k\times l}$, to the row of $Z \in \mathbb{R}^{k\times l}$ which is nearest in terms of the p-norm. That is, for each row m,

  $$m_{\text{rounded}} = Z_i, \quad i = \operatorname*{argmin}_j \|m - Z_{j,:}\|_p.$$

  Used in the optimal control strategy.

3 Notes on the tutorial

The `tutorial.ipynb` performs all NCT analyses mentioned in the paper on a VAR(1) model fitted to a synthetic time series: controllability and impulse response measures, as well as optimal control strategies. Please note that in the article, these analyses are carried out on the VAR(1) model of each subject and centrality measures and/or distributions are reported. For the sake of simplicity, the tutorial only shows the analysis of a single model. Hence, the way results are presented differ slightly from the article.

4 Notes on the synthetic data

The method `utils.generate_random_trajectory(T, seed)` creates synthetic EMA time series in the following way:

1. A multivariate Gaussian distribution was estimated over the A and B matrices of the VAR(1) models used in the study. The estimated mean and covariance are stored in the file `model_distribution.py`. The original data set comprised 15 EMA variables and 4 EMI, thus the synthetic data will have the same size.
2. A VAR(1) model is created using $A \in \mathbb{R}^{15\times 15}$ and $B \in \mathbb{R}^{15\times 4}$ matrices drawn at random from this distribution.
3. An initial condition x_1 is drawn at random from a uniform distribution over $\{-3, -2, -1, 0, 1, 2, 3\}^{15}$.
4. A sequence of input vectors $U = u_1, \ldots, u_{T-1}$ is drawn. Every second input is an indicator vector from the set

   $$\{[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]\}$$

   representing the occurrence of 1 out of 4 EMI. The others are 0. This roughly reflects EMI distributions, where uniformly drawn EMI are administered every second time an EMA is carried out.
5. A sequence $X = x_1, \ldots, x_T$ is created by iterating the model

   $$Ax_t + Bu_t$$

   from the initial condition $t = 1$ to $t = T$, where T is the first argument to the method. The sequence is rounded to the nearest integer vectors in $\{-3, -2, -1, 0, 1, 2, 3\}^{15}$ to mimic Likert scale EMA data.
6. The method returns X, U.

This way, the synthetic data very roughly reflect the properties of the empirical data, which are time series of Likert scales and indicator variables, and even may elicit similar results in the analyses. To obtain the same results again when drawing random synthetic data, specify the `seed` argument to `utils.generate_random_trajectory`.

5 Questions welcome

Please address any questions or comments to the corresponding author Janik Fechtelpeter.