SoftwareX 15 (2021) 100761

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

impulseest: A Python package for non-parametric impulse response estimation with input–output data

Luan Vinícius Fiorio [a,*], Chrystian Lenon Remes [b], Yales Rômulo de Novaes [a]

[a] Santa Catarina State University, R. Paulo Malschitzki, 200 - Zona Industrial Norte, Joinville SC, 89219-710, Brazil
[b] Federal University of Rio Grande do Sul, Av. Osvaldo Aranha, 103 - Centro Histórico, Porto Alegre RS, 90035-190, Brazil

ARTICLE INFO

Article history:
Received 24 March 2021
Received in revised form 8 June 2021
Accepted 29 June 2021

Keywords:
Impulse response
Estimation
Regularization

ABSTRACT

This paper presents the *impulseest* Python package, used for estimating the impulse response of a system relying solely on input and output data. This package can provide estimates in a non-parametric fashion either with regularization techniques. For the regularized estimates, *impulseest* function uses the Empirical Bayes method. On the other hand, the non-regularized case is solved through the least squares algorithm. This function is tested considering an experimental situation, several dynamic processes and also through Monte Carlo simulations. The obtained results are analyzed mainly in terms of the Mean Square Error (MSE), besides other quantities. Through those results, it is shown that the *impulseest* function with regularization using the proposed regularization kernels leads to low MSE for all tested cases.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00056
Code Ocean compute capsule	*codeocean.com/capsule/*7181297
Legal Code License	MIT
Code versioning system used	Git
Software code languages, tools, and services used	Python3
Compilation requirements, operating environments & dependencies	NumPy, SciPy
If available Link to developer documentation/manual	*pypi.org/project/impulseest*
Support email for questions	luan.lvf@edu.udesc.br

1. Motivation and significance

The impulse response of dynamic systems can be used in various applications within different science subjects, such as electric power systems [1], computer networks [2], applied acoustics [3,4], instrumentation and measurement [5], communication and signal processing [6], and circuit fault detection [7]. It is also applicable in the representation of antennas response [8], physiological signal processing [9], and in the estimation of $\mathcal{H}_1$, $\mathcal{H}_2$ and $\mathcal{H}_\infty$ system norms [10].

The estimation of the transfer function or the impulse response of a system is of paramount importance for describing its properties [11], and has been implemented for different programming languages, such as MATLAB® [12,13] and R [14]. However, despite the popularity of Python language for both scientific and technical purposes [15,16], which shows an increase of system identification related software releases in the last years [8,17], there is a lack of methods and algorithms for estimating the impulse response of a system from data in the Python context. In this sense, this work aims to contribute by developing a data-driven impulse response estimation tool using Python language.

The estimation of the impulse response of a system based on input–output data follows system identification theory [18–20], and can be done in a parametric based approach [11,21] and also as a non-parametric type of estimation [18]. When estimating the impulse response with the Least Squares (LS) algorithm in a non-parametric approach, which is commonly achieved by the use of high-order FIR models [22], the variance of the estimation increases linearly with the order of the model [22]. This leads to a large Mean Squared Error (MSE) and may lead the LS algorithm

* Corresponding author.
E-mail address: luan.lvf@edu.udesc.br (Luan Vinícius Fiorio).

https://doi.org/10.1016/j.softx.2021.100761

to diverge [23]. A solution to this problem is to use regularization along with the standard least squares algorithm [24].

Considering the given context, a non-parametric impulse response estimation package relying solely on input–output data was developed in Python and is detailed in this paper. The implementation is done according to the Bayesian perspective presented in [25] and a package is made available in the PyPI indexes.

1.1. Theoretical background

The transfer function of a discrete-time Linear Invariant-Time system can be described as

$$G_0(z) = \sum_{i=1}^{\infty} g_i^0 z^{-i}, \tag{1}$$

where g_i^0, $i \in \mathbb{N}$, is the impulse response of the system and z is the time-shift operator. As known from linear systems and signal theory [26], the output of the system can be obtained, then, by

$$y(k) = G_0(z)u(k) + v(k), \tag{2}$$

in which $u(k)$ is the input signal and $v(k)$ is an additive noise. The ideal impulse response (1) can be truncated to a finite number, resulting in a Finite Impulse Response (FIR) model. This way, least squares method can be used to estimate the impulse response [27]. But a problem may arise in this situation: the variance of the estimate increases linearly with the FIR model order, i.e., the final terms, that should go to zero for stable systems, will be poorly estimated and result in several nonzero values. To counteract this issue, one must use some form of regularization, aiming to obtain a sparse solution, i.e., a solution that presents more zero estimated parameters.

The standard least squares problem is formulated as

$$\min_{\theta} \quad \|Y_N - \Phi_N^T\theta\|_2^2, \tag{3}$$

where Y_N are the N output regressors, such that $Y_N = [y(n+1)\, y(n+2) \ldots y(N)]^T$, in which n is the number of impulse response coefficients to be estimated,[1] Φ_N are the N input regressors, given by $\Phi_N = [\phi(n+1)\, \phi(n+2) \ldots \phi(N)]$, where $\phi(k) = [u(k-1)\, u(k-2) \ldots u(k-n)]^T$, and θ are the n truncated impulse response coefficients. Those coefficients are the LS solution of (3), given as:

$$\hat{\theta} = [\hat{g}_1\ \hat{g}_2 \ldots \hat{g}_n]^T = (\Phi_N\Phi_N^T)^{-1}\Phi_N Y_N. \tag{4}$$

Since the solution (4) is non-regularized, it has increased variance and non-zero high-order terms in the estimated impulse response. To counteract this issue, regularization can be included in the optimization problem, which is commonly represented by the weighted 2-norm of the vector θ, resulting in the regularization term $\theta^T D\theta$, where D is the weighting matrix, also known as the regularization matrix. In this sense, the inclusion of this regularization term in (3), leads to the new optimization problem:

$$\min_{\theta} \sum_{k=n+1}^{N} (y(k) - \phi^T(k)\theta)^2 + \theta^T D\theta. \tag{5}$$

Notice that this regularized problem tries to find a solution that minimizes both the MSE of the estimates and the 2-norm of the parameter vector, which should result in a sparser solution when compared to the non-regularized one. However, such optimization problem introduces a new unknown, represented by the regularization matrix D, and that should be estimated alongside with θ. Differently of the initial non-regularized problem (3), the regularized solution cannot be explicitly determined.

An efficient approach for finding a solution to (5) is through the Bayesian perspective [27,28]. The idea is to consider the estimation parameter a random variable and given the observations, search for some *a posteriori* distribution. At first, it is considered a parameter θ with zero mean and covariance matrix P_n, such that $\theta \sim \mathcal{N}(0, P_n)$, and a white noise with σ^2 variance, described by $v(k) \sim \mathcal{N}(0, \sigma^2)$. From those considerations, an *a posteriori* distribution of θ given the vector Y_N can be obtained as follows [27]:

$$\theta|Y_n \sim \mathcal{N}(\hat{\theta}_N^{apost}, P_N^{apost}) \tag{6a}$$

$$\hat{\theta}_N^{apost} = ((\sigma^2(\Phi_N\Phi_N^T)^{-1})^{-1} + P_n^{-1})^{-1}(\sigma^2(\Phi_N\Phi_N^T)^{-1})^{-1}\hat{\theta}_N^{LS} \tag{6b}$$

$$P_N^{apost} = ((\sigma^2(\Phi_N\Phi_N^T)^{-1})^{-1} + P_n^{-1})^{-1} \tag{6c}$$

in which *apost* means *a posteriori*, the 'hat' marker indicates an estimation and $\hat{\theta}_N^{LS}$ is the solution of the non-regularized least squares case (4). The *a posteriori* estimate $\hat{\theta}_N^{apost}$ will be equal to the regularized estimate if [27]

$$D = \sigma^2 P_n^{-1}. \tag{7}$$

In a few words, the solution of (5) depends on the covariance matrix P_n and on the noise variance σ^2, besides the LS solution $\hat{\theta}_{LS}$, and they should be estimated jointly with θ.

In order to estimate the covariance matrix P_n, it can be written in a parametrized manner, according to some known prior covariance matrices obtained through the Bayesian approach — also known as kernels. The three kernels for the *impulseest* package, here chosen through the results provided in [25], are the diagonal/correlated (DC) kernel P_{DC}, the diagonal (DI) kernel P_{DI} and the tuned correlated (TC) kernel P_{TC}, respectively given by:

$$P_{DC}(k,j) = c\rho^{|k-j|}\lambda^{(k+j)/2}; \tag{8}$$

$$P_{DI}(k,j) = \begin{cases} c\lambda^k, & if\ k = j; \\ 0, & else; \end{cases} \tag{9}$$

$$P_{TC}(k,j) = c\ min(\lambda^j, \lambda^k). \tag{10}$$

An array of hyper-parameters $\alpha = [c\ \lambda\ \rho\ \sigma]$ is denoted for the prior distribution – in the case of DC kernel or $\alpha = [c\ \lambda\ \sigma]$ in the case of DI or TC kernels – which can be estimated through the maximum likelihood approach, according to the *empirical Bayes* method[2] [28], applied in

$$Y_N \sim \mathcal{N}(0, \sigma^2 I_{N-n} + \Phi_N^T P_n(\alpha)\Phi_N), \tag{11}$$

resulting in

$$\hat{\alpha} = \min_{\alpha}\ Y_N^T \Sigma(\alpha)^{-1} Y_N + \log\det\Sigma(\alpha) \tag{12}$$

with

$$\Sigma(\alpha) = \sigma^2 I_{N-n} + \Phi_N P_n(\alpha)\Phi_N^T. \tag{13}$$

2. Software description

The *impulseest* function provides a non-parametric estimation of the impulse response using only input–output data of a process. The estimation can be regularized or not. In the sections below, the package *impulseest* is detailed.

2.1. Software architecture

The package is structured in two modules: *impulseest.py* and *creation.py*, which are described in the following subsections. The main module is *impulseest.py*, which contains the main function. The *creation* module initializes arrays and matrices.

[1] The first n outputs of the data set are not used to allow $\phi(k)$ to be formed.

[2] Similar approaches can be achieved with other Bayesian methods as well [29,30].

Table 1
Hyper-parameters' bounds.

Parameter	Lower bound	Upper bound
c	10^{-8}	none (∞)
λ	0.7 (DI, TC) | 0.72 (DC)	1
σ	0	none (∞)
ρ (DC only)	-0.99	0.99

2.1.1. Creation.py

This module contains functions that create arrays and matrices that are not called during the minimization procedure, so there is no need to define them inside the *impulseest* function. Those functions called by the main *impulseest* function during the initialization of the variables are:

- **create_alpha**: returns the initial alpha (hyper-parameter) array according to the chosen regularization method;
- **create_bounds**: returns the bounds of each hyper-parameter to be considered in the minimization procedure, according to the chosen regularization method;
- **create_Phi**: returns the input regressor matrix;
- **create_Y**: returns the output regressor matrix.

The bounds that guarantee a low condition number for the P_n matrix, for each regularization kernel, are given in Table 1 and defined in [27]. The lower bound of c is included to avoid singular value decomposition divergence in the minimization procedure using SciPy's minimization function.

The idea of using a separated module to initialize some variables is a choice of organization. One of the main objectives was to keep the code as clean and clear as possible to ease future modifications by its users. To allow average – and even beginner – users to make modifications as desired, the code was implemented in procedural programming, avoiding the need of background knowledge in object-oriented programming.

2.1.2. Impulseest.py

This is the main module of the package, which has two functions:

- ***impulseest***: the main function of the package, it estimates the impulse response of a system in a non-parametric fashion based only in input–output data. Its details and implementation are described in the next subsection;
- **argument_check**: a function that checks if the arguments given by the user are valid.

2.2. Software functionalities

The package proposes the non-parametric estimation of the impulse response of a system relying solely on input–output data. This estimation can be regularized or not, as chosen by the user. The main function of this package is *impulseest* and have five possible arguments:

- u [numpy array]: input signal (size $N \times 1$);
- y [numpy array]: output signal (size $N \times 1$);
- n [integer]: number of impulse response estimates (default is $n = 100$);
- RegularizationKernel [string]: regularization method — 'none', 'DC', 'DI' or 'TC' (default is 'none');
- MinimizationMethod [string]: bound-constrained optimization method used to minimize the cost function — 'L-BFGS-B', 'Powell' or 'TNC' (default is 'L-BFGS-B').

The *impulseest* function begins by reshaping the input (u) and output (y) data arrays to an $N \times 1$ shape. All the input arguments are then checked in the *argument_check* function, raising an exception if something is not correct. The initialization of the global variables (arrays, matrices) is done through the functions in the creation module.

The first local function defined inside *impulseest* is *Prior*. This function writes the P_n matrix of (7) according to the chosen regularization kernel, as presented in (8), (9) and (10). If no kernel is chosen, *Prior* returns *None*. It has α as an argument, which is updated during the minimization procedure.

```
def Prior(alpha):
for k in range(n):
for j in range(n):
if(RegularizationKernel=='DC'):
P[k,j] = alpha[0]*(alpha[2]**abs(k j))*(alpha[1]**((k+j)/2))
elif(RegularizationKernel=='DI'):
if(k==j):
P[k,j] = alpha[0]*(alpha[1]**k)
else:
P[k,j] = 0
elif(RegularizationKernel=='TC'):
P[k,j] = alpha[0]*min(alpha[1]**j,alpha[1]**k)
else:
None
return P
```

The second local function, named *cost_func*, is the cost function of the minimization procedure. An efficient implementation of the least squares algorithm, in terms of computational complexity, can be based on two main approaches: the Cholesky factorization or the QR factorization [31][3]. According to [31], solving least squares with QR factorization is more precise than with the Cholesky factorization if — in this case - P_n is ill-conditioned. As presented in the literature, P_n can be very ill-conditioned when estimating the impulse response of a process [27]. Therefore, the QR factorization approach is chosen to be implemented, according to [25], as described below.

The thin QR factorization

$$[\Phi_N^T \; Y_N] = Q_d[R_{d1} \; R_{d2}] \tag{14}$$

is precomputed, where Q_d is an $N \times (n+1)$ matrix, R_{d1} is an $(n+1)\times n$ matrix and R_{d2} is an $(n+1)\times 1$ array. Then, the function *cost_func* is defined and has the following calculation steps:

1. compute the Cholesky factorization L of $P_n(\alpha)$;
2. compute $R_{d1}L$;
3. compute the QR factorization

$$\begin{bmatrix} R_{d1}L & R_{d2} \\ \sigma I_n & 0 \end{bmatrix} = Q_c R_c; \tag{15}$$

4. compute the cost

$$\frac{r^2}{\sigma^2} + (N-n)\log\sigma^2 + 2\log|R_1|, \tag{16}$$

which solves the problem presented in (12).

```
#precomputation
aux0 = qr(hstack((transpose(Phi),Y)),mode='r')
Rd1 = aux0[0:n+1,0:n]
Rd2 = aux0[0:n+1,n]
Rd2 = Rd2.reshape(len(Rd2),1)

#cost function
def cost_func(alpha):
L = cholesky(Prior(alpha))
Rd1L = Rd1 @ L
to_qr = bmat([[Rd1L,Rd2],[alpha[len(alpha) 1]*I,zeros((n,1))
    ]])
```

[3] Solving regularized least squares in a computational efficient way is still a subject of research nowadays [32,33].

```
R = qr(to_qr,mode='r')
R1 = R[0:n,0:n]
r = R[n,n]
cost = (r**2)/(alpha[len(alpha) 1]**2) + (N n)*log(alpha[len
    (alpha) 1]**2)
 + 2*sum(log(abs(diag(R1))))
return cost
```

The minimization procedure uses the *scipy.optimize.minimize* function and is implemented as follows:

```
A = minimize(cost_func, alpha_init, method=
    MinimizationMethod, bounds=bnds)
alpha = A.x
L = cholesky(Prior(alpha))
Rd1L = Rd1 @ L
to_qr = bmat([[Rd1L,Rd2],[alpha[len(alpha) 1]*I,zeros((n,1))
    ]])
R = qr(to_qr,mode='r')
R1 = R[0:n,0:n]
R2 = R[0:n,n]
ir = L @ pinv(R1) @ R2
ir = ir.reshape(len(ir),1)
```

where *alpha_init* is the initial alpha generated by *create_alpha*, *bnds* is the array with bounds generated by *create_bounds* and the method is chosen by the user, according to the documentation of the SciPy's module [34]. The QR factorization of (15) is recalculated for the new alpha and then the impulse response is obtained, which is reshaped to $n \times 1$ and returned to the user.

3. Illustrative examples

In this section, the *impulseest* function is illustrated through examples. At first, an experimental case is shown to illustrate the impulse response identification for a real-world case. Then, a test is executed considering a grid of values that form a whole set of transfer functions, which is denoted here by test grid. The objective is to investigate the behavior of the function for different cases. In the sequence, three Monte Carlo simulations are done for each regularization kernel, where a statistical analysis is realized to evaluate the robustness of the methods to deal with data corrupted by noise. All obtained results are depicted in through box plots and analyzed both qualitatively and quantitatively.

3.1. Experimental case

In order to illustrate the proposed package, an experimental setup of a boost converter operating in continuous conduction mode, used to increase the voltage of a photovoltaic array, is considered. The boost converter is controlled by a Digital Signal Processor (DSP) with a stabilizing controller. Fig. 1 shows the boost converter circuit fed by a three-phase diode rectifier and the diagram of the control scheme, which parameters are presented in Table 2. The output voltage signal used for the impulse response estimation was acquired by the DSP.

The labels from Fig. 1 that are not defined in Table 2 are: v_r, v_s, v_t — the voltage of each phase of the AC input to the rectifier; $i_L(t)$ — the inductor current; Q_1 — the MOSFET transistor (switch); Q_2 — the output diode; $v_o(t)$ — the instantaneous output voltage; R_o - the load that results in P_o; B1 and B2 — buffers of the DSP; LPF — a low pass filter with cut-off frequency ω_{LPF}; ADC — analog to digital converter; K_{AD} — feedback gain; $y(k)$ - digital output voltage signal after K_{AD} compensation; controller — the controller structure applied digitally; $u(k)$ — the output of the controller; PWM — pulse-width modulation block, which transforms the input signal into a pulse-width modulated signal; and $\tilde{d}(t)$ — duty cycle signal that is applied to the gate-driver of the MOSFET Q_1.

Table 2
Parameters of the boost converter used as example.

Description	Parameter	Value
Output power	P_o	400 W
Input voltage	v_{in}	65 V∼85 V
Nominal output voltage	Y_o	310 V
Nominal duty cycle	U_o	0.72 pu
Switching frequency	f_s	50 kHz
Sampling frequency	f_a	50 kHz
Output filter inductance	L_c	2.15 mH
Output filter capacitance	C_c	2.2 μF
Input capacitance	C_{in}	22.6 mF
Cut-off frequency LPF	ω_{LPF}	25 kHz

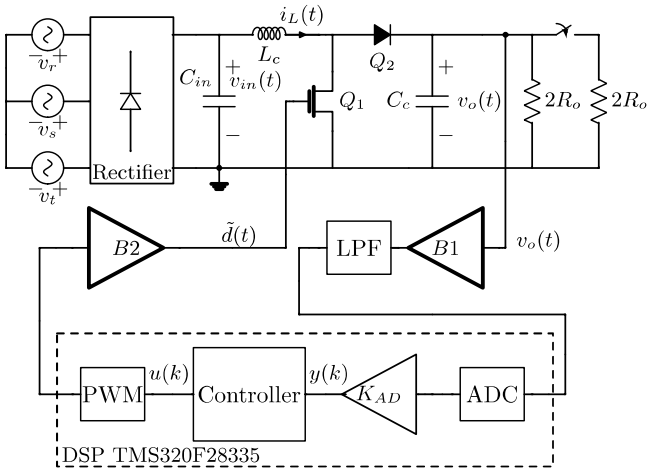


Fig. 1. Rectifier, boost converter and the control scheme used for this example.

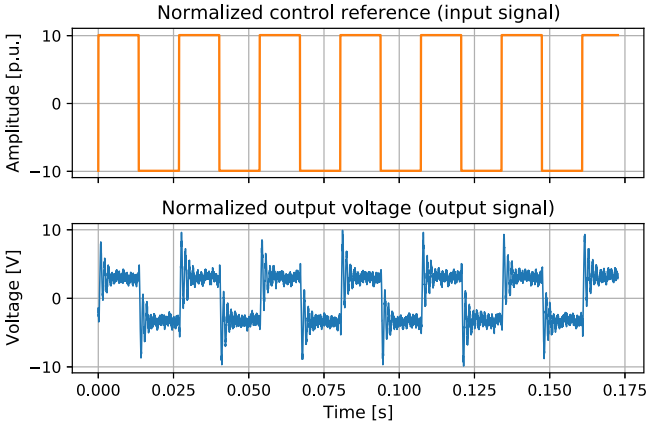


Fig. 2. Normalized control reference signal (input) and normalized output voltage (output) of the boost converter used for this example.

The input (excitation) signal used for this experiment is a square wave with 10 V of amplitude and 37 Hz of frequency. The square wave signal is sufficiently rich to estimate the impulse response and simple enough to be generated in a DSP. This signal is used as the controller's reference. The output signal is the output voltage of the boost converter. As it can be seen from Fig. 2, there is a considerable amount of noise in the output signal. Both signals were normalized with amplitudes from −10 to 10 V for this estimation procedure.

The impulse response of the output voltage by the control reference plant of the boost converter used in this experiment is estimated through *impulseest*, using the DC kernel and also with no regularization kernel. The IR's are shown in Fig. 3. The

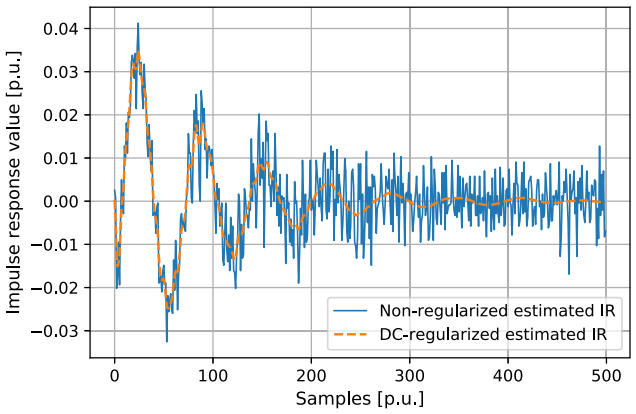


Fig. 3. DC-regularized and non-regularized impulse response estimation of a boost converter's output voltage by control reference plant.

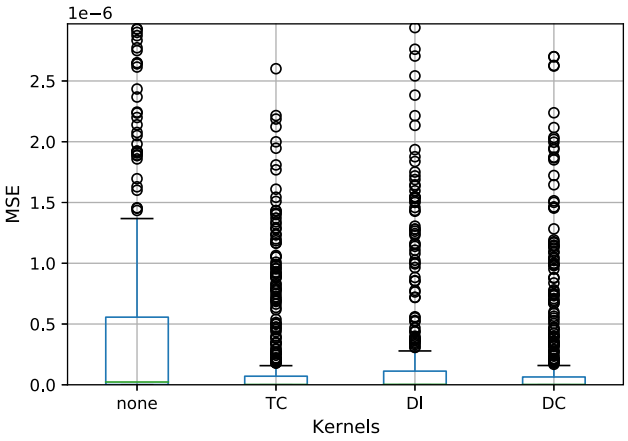


Fig. 4. Box plot of all the MSEs obtained during the test for all kernels.

non-regularized case suffers from the increase of variance in the impulse estimates as the order increases, which generates a highly noisy impulse response. At the other hand, the regularization kernel compensates the variance increase, resulting in a less noisy impulse response.

3.2. Test grid

A test grid was defined by using a second-order discrete-time plant with variable parameter values and step time of 0.001 s. The model was chosen as a second-order system once many processes can be modeled as such [35,36]. The transfer function of the plant to be tested with *impulseest* is

$$G_{test}(z) = \frac{p_1 + p_2}{\lambda} \frac{(z-\lambda)}{(z-p_1)(z-p_2)}, \tag{17}$$

where p_1, p_2 and λ are defined as follows:

- λ, the zero of the plant, varies from ≈0.22 to ≈3.16, embracing both minimum and nonminimum-phase systems. It is worth to mentioning that the case where the zero is exactly 1 is not included for simplicity;
- the poles p_x, $x = 1, 2$ are complex-conjugate poles, written as $p_x = ae^{\pm j \cdot b}$, where $a \in (0.27,\ 0.978)$ and $b \in (0.006,\ 0.82)$. Therefore, all possible poles are inside the unit circle and all considered systems are stable.

Table 3 shows the considered values for a, b and λ.

The input signal of the test grid is a PRBS and follows the same characteristics described in Section 3.1. The output signals are obtained by simulating the response of each plant to the input signal using the SciPy's *dlsim* function. A randomly generated Gaussian white noise is added to both signals before each of the impulse response's estimation.

3.3. Statistical analysis

The result of the grid test are the box plots presented in Fig. 4, with values in Table 4, which has been obtained through 1024 estimations for each regularized kernel and the non-regularized case, varying a, b and λ with equally spaced steps, comparing the first 200 terms of the estimated against the model-based impulse response. As can be seen, the regularized estimations resulted in lower MSE than the non-regularized ones for most cases. The smaller interquartile range is observed in the TC kernel, whilst between the regularized cases, the DI kernel has the largest one.

For each box plot, there are around 200 outliers out of the 1024 samples. That happens because the system under test is changing its parameters and for each resulting system, a different type of regularization may achieve better results. At the same time, at each test, there is a different noise realization, what makes the outliers even more present. In summary, both the plant and noise have been changed together.

Table 3
Values of a, b and λ.

a	b	λ
0.272251	0.006738	0.223130
0.473223	0.013376	0.325814
0.598901	0.026554	0.475753
0.697120	0.052715	0.694693
0.779969	0.104649	1.014388
0.852692	0.207748	1.481207
0.918117	0.412419	2.162854
0.977970	0.818731	3.158193

Table 4
Quantitative box plot results for the grid test MSEs.

Metrics	(none)	TC	DI	DC
mean ($\times 10^{-3}$)	20.0	2.12	25.7	1.37
σ ($\times 10^{-2}$)	26.3	3.53	40.4	1.98
min ($\times 10^{-12}$)	36.9	7.40	10.7	16.9
max	6.16	1.01	10.1	0.50

3.3.1. Monte Carlo

A Monte Carlo experiment, in this subject, is to estimate the impulse response of a fixed discrete-time plant, with a time step of 0.001 s, a determined number of times, with only different noise realizations at each execution. Three Monte Carlo experiments have been considered, one for each of the following discrete-time systems:

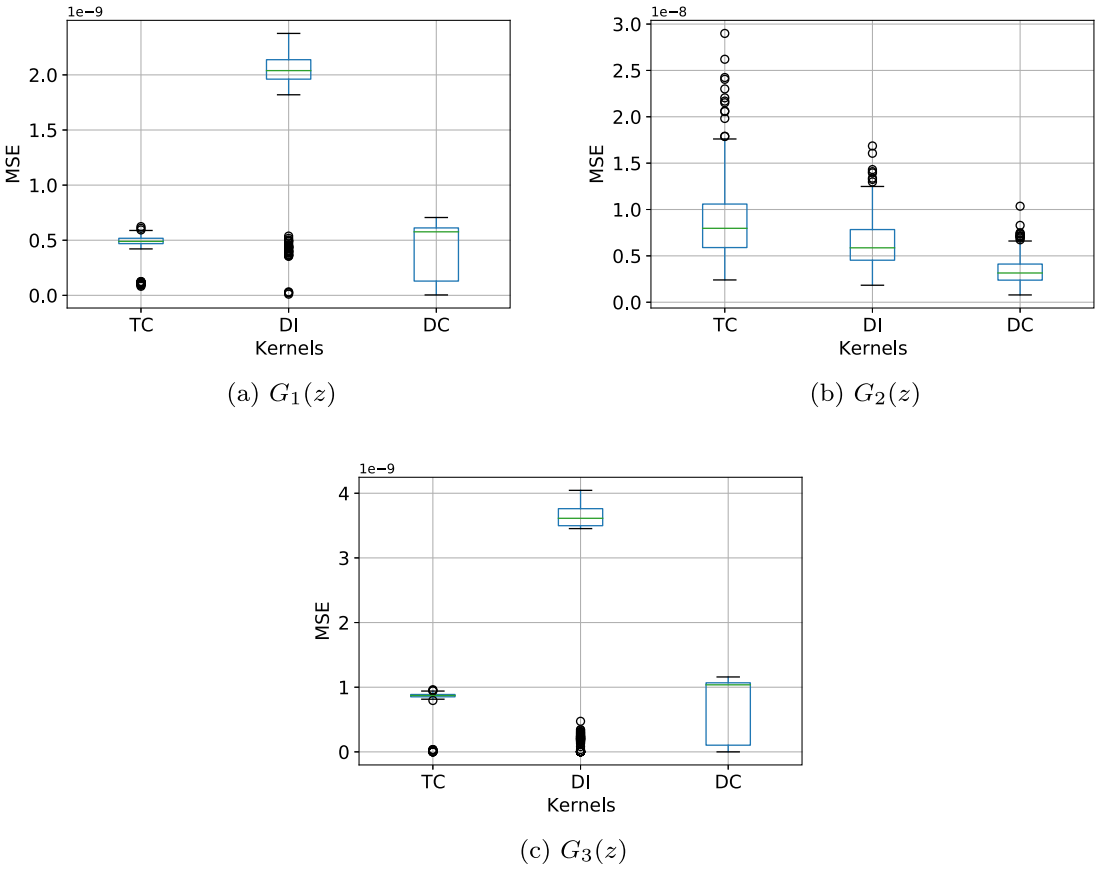
$$G_1(z) = -0.25\frac{(z-1.2)}{(z-0.95)}; \tag{18}$$

$$G_2(z) = 0.5\frac{(z-0.8)}{(z-0.9-0.3j)(z-0.9+0.3j)}; \tag{19}$$

$$G_3(z) = -0.075\frac{(z-1.2)}{(z-0.7)(z-0.95)}. \tag{20}$$

These structures of models are common to appear when processes are modeled and approximated to a first or second-order model [35].

For all cases, input and output signals have been corrupted by independent additive noises. The applied noise signals have the same attributes as detailed in Section 3.1. For each system



(a) $G_1(z)$ (b) $G_2(z)$ (c) $G_3(z)$

Fig. 5. Box plot of the Monte Carlo MSEs for plants $G_1(z)$, $G_2(z)$ and $G_3(z)$.

Table 5
Quantitative box plot results (MSE) for plant $G_1(z)$.

Metrics	TC	DI	DC
mean ($\times 10^{-10}$)	4.73	18.7	4.44
σ ($\times 10^{-11}$)	9.85	55.4	24.3
min ($\times 10^{-12}$)	82.1	10.6	3.64
max ($\times 10^{-10}$)	6.23	23.8	7.06

Table 6
Quantitative box plot results (MSE) for plant $G_2(z)$.

Metrics	TC	DI	DC
mean ($\times 10^{-9}$)	8.86	6.38	3.43
std ($\times 10^{-9}$)	4.32	2.50	1.44
min ($\times 10^{-10}$)	24.0	18.3	7.86
max ($\times 10^{-8}$)	2.90	1.69	1.04

Table 7
Quantitative box plot results (MSE) for plant $G_3(z)$.

Metrics	TC	DI	DC
mean ($\times 10^{-10}$)	8.44	30.4	7.52
std ($\times 10^{-10}$)	1.57	13.6	4.61
min ($\times 10^{-13}$)	3.44	13.3	1.62
max ($\times 10^{-10}$)	9.61	40.4	1.16

it has been considered 300 realizations, for each one of the regularization kernels.

Fig. 5(a) shows the box plot for all three regularization kernels at the estimation of the impulse response of plant $G_1(z)$. TC has the smaller interquartile range and DC the largest, but DC achieved a lower mean than DI. There are no outliers for the DC kernel, whilst there are a few for DI and TC. The quantitative results for this system are presented in Table 5.

The second plant $G_2(z)$ has an oscillatory behavior caused by the complex-conjugate poles. As can be seen in Fig. 5(b) and from Table 6, all three regularization kernels have outliers, with best results being achieved by the DC kernel in terms of mean of the box plots and size of the interquartile range.

The third case, with plant $G_3(z)$, presented in Fig. 5(c), shows a similar behavior to that of plant $G_1(z)$. Again, TC kernel has the smallest interquartile range and DI the higher MSE values. The quantitative results of $G_3(z)$ are presented in Table 7.

4. Impact

The intention of the *impulseest* package is to impact researchers in the academic and industrial scenarios, who have an application to the impulse response of some system, based only in input–output data. The *impulseest* function/package was only available in MATLAB® [12] and R [14], therefore, the need of a Python function to estimate IR was felt, as it is an open-source, widely used language in various subjects of science and one of the most in-demand and popular [15,16] high-level object-oriented programming language nowadays.

5. Conclusions

The *impulseest* package is a Python package for the non-parametric estimation of the impulse response of a system based only in input–output data. The package takes a Bayesian approach to the problem by using the Empirical Bayes method, with an QR-matrix approach to the implementation in order to lower the computational complexity. The user can choose between

a regularized estimation – with three available regularization kernels – or not regularized. The architecture of the software is clean and simple, enabling most of the average Python users to fork the GitHub repository and make modifications as desired. The main function of the package, *impulseest*, obtained low MSE in estimating the impulse response of all the pants that were tested, mainly if regularization was used. The *impulseest* package is distributed under an MIT license and it is open-source.

The main results show that the use of regularized estimates of impulse responses using DC and TC kernels provided more consistent results, once they lead to smaller means and standard deviations of the MSE, along with a reduced number of outliers, when compared to the other cases. As a future work, one could study and compare the use of another Bayesian methods in the estimation of hyper-parameters in order to achieve a more efficient implementation; regarding application, one can highlight the use of *impulseest* for data-driven estimation of system norms for robust control design.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and partly by the Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina (FAPESC) - Grant number 8209/2021.

References

[1] Ferreira Caetano CE, Saldanha Paulino JO, Fonseca Barbosa C, do Couto Boaventura W. Earthing system characterization from its measured impulse response. Electr Power Syst Res 2020;187:106517.
[2] Sciacca EC, Galluccio L. Impulse response analysis of an ultrasonic human body channel. Comput Netw 2020;171:107149.
[3] Brinkmann F, Aspöck L, Ackermann D, Opdam R, Vorländer M, Weinzierl S. A benchmark for room acoustical simulation. Concept and database. Appl Acoust 2021;176.
[4] Lim H, Imran M, Jeon JY. A new approach for acoustic visualization using directional impulse response in room acoustics. Build Environ 2016;98:150–7.
[5] Kang T, Kim S, Oh K-I, Hwang J-H, Lee J, Park H, et al. Evaluation of human body characteristics for electric signal transmission based on measured body impulse response. IEEE Trans Instrum Meas 2020;69(9):6399–411.
[6] Wu Yanqun, Zhang Wen, Lin Yonggang, Xu Guojun, Zhang Bingbing. Geoacoustic inversion based on matched impulse response processing for moving source. In: 2020 IEEE 3rd international conference on information communication and signal processing (ICICSP). 2020. p. 113–116.
[7] Parai M, Srimani S, Ghosh K, Rahaman H. Analog circuit fault detection by impulse response-based signature analysis. Circuits Systems Signal Process 2020;39(9):4281–96.
[8] Semper S, D M. eadf: Representation of far-field antenna responses in Python. SoftwareX 2020;12:100583.
[9] Bizzego A, Battisti A, Gabrieli G, Esposito G, Furlanello C. pyphysio: A physiological signal processing library for data science approaches in physiology. SoftwareX 2019;10:100287.
[10] Gonçalves da Silva GR, Bazanella AS, Campestrini L. One-shot data-driven controller certification. ISA Trans 2020;99:361–73.
[11] Ljung L. On the estimation of transfer functions. In: 7th IFAC/IFORS symposium on identification and system parameter estimation, York, UK, 3-7 July. IFAC Proc Vol 1985;18(5):1653–7.
[12] Matlab. impulseest: Nonparametric impulse response estimation. United States of America: MathWorks, Inc.; 2020.
[13] Matlab. System identification toolbox. United States of America: MathWorks, Inc.; 2021.
[14] Yerramilli S, Tangirala A. impulseest: estimate impulse response coefficients. Vienna, Austria: R Foundation for Statistical Computing; 2017.
[15] Rosalie C. The 10 most popular programming languages, according to the microsoft-owned github; 2019. https://www.businessinsider.com/most-popular-programming-languages-github-2019-11. [Accessed 8 August 2021].
[16] Brian E. The 10 most popular programming languages to learn in 2021; 2021. https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/. [Accessed 8 August 2021].
[17] Boeira E, Eckhard D. pyvrft: A python package for the virtual reference feedback tuning, a direct data-driven control method. SoftwareX 2020;11:100383.
[18] Ljung L. System identification: Theory for the user. USA: Prentice-Hall, Inc.; 1986.
[19] Tangirala AK. Principles of system identification: Theory and practice. CRC Press, Taylor & Francis Group; 2015.
[20] Kailath T, Sayed AH, Hassibi B. Linear estimation. first ed. Prentice Hall; 2000.
[21] Killich A, Rake H. Impulse response estimation via parametric methods. In: 9th IFAC/IFORS Symposium on identification and system parameter estimation 1991, Budapest, Hungary, 8-12 July 1991. IFAC Proc Vol 1992;25(15):331–6.
[22] Chen T, Zhao Y, Ljung L. Impulse response estimation with binary measurements: A regularized FIR model approach. In: 16th IFAC symposium on system identification. IFAC Proc Vol 2012;45(16):113–8.
[23] Akçay H, Khargonekar PP. The least squares algorithm, parametric system identification and bounded noise. Automatica 1993;29(6):1535–40.
[24] Gubarev VF, Panova NV. Properties of the regularized least-squares method in transfer function identification. In: 14th IFAC world congress 1999, Beijing, Chia, 5-9 July. IFAC Proc Vol 1999;32(2):4201–6.
[25] Chen T, Ljung L. Implementation of algorithms for tuning parameters in regularized least squares problems in system identification. Automatica 2013;49(7):2213–20.
[26] Lathi BP. Linear systems and signals. second ed. USA: Oxford University Press, Inc.; 2009.
[27] Chen T, Ohlsson H, Ljung L. On thse estimation of transfer functions, regularizations and Gaussian processes—Revisited. Automatica 2012;48(8):1525–35.
[28] Carlin BP, Louis TA. Bayes and empirical bayes methods for data analysis. Stat Comput 1997;7(2):153–4.
[29] Tichý O, Šmídl V. Non-parametric Bayesian models of response function in dynamic image sequences. Comput Vis Image Underst 2016;151:90–100.
[30] Ganesan AL, Rigby M, Zammit-Mangion A, Manning AJ, Prinn RG, Fraser PJ, et al. Characterization of uncertainties in atmospheric trace gas inversions using hierarchical Bayesian methods. Atmos Chem Phys 2014;14(8):3855–64, https://acp.copernicus.org/articles/14/3855/2014/.
[31] Golub GH, Van Loan CF. Matrix computations. third ed. USA: Johns Hopkins University Press; 1996.
[32] Shen Y, Ypma TJ. Solving separable nonlinear least squares problems using the QR factorization. J Comput Appl Math 2019;345:48–58.
[33] Teng Y, Qi S, Han F, Yao Y, Fan F, Lyu Q, et al. A framework for least squares nonnegative matrix factorizations with Tikhonov regularization. Neurocomputing 2020;387:78–90.
[34] The SciPy Community. SciPy Optimization. 2020.
[35] Nise NS. Control systems engineering. third ed. USA: John Wiley & Sons, Inc.; 2000.
[36] Skögestad S, Postlethwaite I. Multivariable feedback control: Analysis and design. Hoboken, NJ, USA: John Wiley & Sons, Inc.; 2005.