Chapter 4
ForceFinder: A Python Package for Inverse Source Estimation

Steven Carter

Abstract Inverse source estimation (ISE) is a common technique in structural dynamics, with applications in transfer path analysis (TPA), multiple-input / multiple-output (MIMO) vibration control, and force reconstruction. Unfortunately, ISE is an ill-posed process, which makes it sensitive to measurement noise and modelling errors. Many techniques have been proposed to mitigate the issues related to the ill-posed nature of the problem. However, it can be difficult for the practitioner to implement and use the techniques, limiting their adoption. Simple bookkeeping mistakes can also lead to significant errors in the estimated sources. ForceFinder was created to mitigate these problems and make ISE easier to perform. It does this with a SourcePathReceiver object that automatically performs the bookkeeping without intervention from the user. Several advanced ISE techniques have also been implemented as methods for the SourcePathReceiver object, which gives non-expert practitioners access to the techniques. ForceFinder also has functions to benchmark and compare the different ISE techniques on a simple academic problem. It is hoped that ForceFinder can become a common framework for ISE problems, making it easier for researchers and practitioners to collaborate and accelerate the adoption of advanced ISE techniques.

Keywords TPA · Force reconstruction · MIMO control · Open source · Python

Introduction

Inverse source estimation (ISE) is a common technique in structural dynamics, with applications in force reconstruction [1], transfer path analysis (TPA) [2], and multiple-input / multiple-output (MIMO) vibration control [3]. Unfortunately, ISE is known to be an ill-posed process that is sensitive to measurement and modelling errors [4–6]. Many techniques have been developed to improve the robustness of ISE to these errors [7–17] but very few of them have been implemented in commercial software, limiting their adoption. Further, most of the proposed techniques are algorithmically complex, which makes it difficult for the practitioner to implement and use them. The implementation difficulties are further compounded by the strict bookkeeping requirements in ISE. Consequently, it is difficult for the practitioner to use advanced ISE methods in "production" work, where the software must work "out of the box", as compared to research work where in-situ code modifications are acceptable.

These implementation details may stifle advancements in ISE, since researchers and practitioners may not be able to use, benchmark, and compare all the available ISE methods. The ForceFinder Python package [18] was created to mitigate these issues and make ISE easier to perform, this package can be found at https://github.com/sandialabs/forcefinder. The project has two main goals:

1. Advance the state of the art in ISE via frequency response function (FRF) matrix inversion by creating a common framework to implement and review different ISE methods.
2. Make it easier for non-expert practitioners to use advanced ISE methods through the creation of a simple object-oriented framework that can be used with a "low code" approach.

This paper will briefly introduce the ForceFinder ISE theory and framework, describe the ISE methods that have been implemented in the package (at the time of writing), describe some benchmarks that are included with the package, and show a simple example of using the package.

Steven Carter
Sandia National Laboratories, P.O. Box 5800 – MS0557, Albuquerque, NM, 87185
e-mail: spcarte@sandia.gov

© The Author(s), under exclusive license to River Publishers 2026

Kristin B. Zimmerman, PhD (eds.), *Dynamic Substructuring & Transfer Path Analysis, 2026, Vol. 4*, Conference Proceedings of the Society for Experimental Mechanics Series, https://doi.org/10.13052/rp-9788743814153

ISE Framework in ForceFinder

ForceFinder introduces an object oriented framework for ISE problems that is based on the SourcePathReceiver (SPR) object, which leverages the objects in the SDynPy [19] Python package. The SPR object holds all the fundamental information for an ISE problem that is based on FRF matrix inversion. The fundamental parameters that create a SPR object are (some parameters are required while others are not):

- FRFs – required for object initialization
- Responses – required for object initialization
- Forces – not required to initialize the object, but will be stored in the object once they are estimated
- Response and reference (force) transformations – not required for object initialization, but may be used for a virtual point transformation (VPT) [20, 21], etc.

These parameters are passed to the object initializer/constructor as SDynPy objects, which allows ForceFinder to automatically organize the data. The data is stored as NumPy arrays [22] in the object, which reduces overhead and makes it easy to perform calculations. As a result, the data is prepared for fundamental ISE operations upon object creation, meaning that the practitioner does not need to consider any bookkeeping when performing an ISE problem. Further, the practitioner does not need to consider the application of transformations to the FRFs and responses since they are automatically applied to the data in the ISE method call. Note that the individual pieces of data (FRFs, responses, etc.) can be recalled as SDynPy objects through SPR object attributes. For example, the FRFs can be recalled by typing spr_object.frfs, where the FRFs would be output as a SDynPy TransferFunctionArray.

The automatic bookkeeping also simplifies the process of implementing a new ISE method. This is because the practitioner simply needs to provide code that takes FRF and response arrays as arguments, which are then used to estimate the forces. No consideration needs to be made in the ISE code for any pre or post processing, such as transformations or transient data constant overlap and add (COLA) processing. The pre and post processing (that is consistent to all ISE methods) is applied in so-called inverse_processing decorator functions, which are applied to every ISE method in the package.

Additional organization methods exist in the SPR object to enable sample splitting for so-called “training” and “validation” response degrees of freedom (DOFs). The sample splitting is applied to both the responses and FRFs. The training response DOFs are used in the force estimation, whereas the validation DOFs are held out and only used after the ISE for optional quality evaluations. The training and validation DOFs are concatenated to create a superset of “target” response DOFs, which are response DOFs in the FRFs that have accompanying response data. Note that accompanying response data is not required for all the response DOFs in the FRFs, meaning that responses can be predicted at locations where measured data is unavailable. The difference between the training and validation data is intuited by the initializer/constructor function in one of two ways:

1. The user can supply the target and training response data as separate SDynPy objects. The function will determine the validation DOFs based on the DOFs that are not in the intersection between the target and training data.
2. The user can supply a SDynPy CoordinateArray of training response DOFs and the function will perform the sample splitting accordingly. The function will determine the validation DOFs based on the DOFs that are not in the intersection between the target and training CoordinateArrays.

Note that the target and training data (for either the FRFs or responses) does not need to have the same ordinate, for cases where the data has been processed differently. Further, the user is not required to explicitly provide both training and target response data/DOFs. The initializer/constructor function will assume that the target and training DOFs are the same (i.e., there are not any validation DOFs) if the training response DOFs are not explicitly defined per the above methods.

Basic ISE Theory in ForceFinder

SPR objects have been created for three main response types:

1. LinearSourcePathReceiver – this SPR object type is intended for responses that are formatted as linear spectra, which are supplied as a SDynPy SpectrumArray

2. PowerSourcePathReceiver – this SPR object type is intended for responses that are formatted as cross power spectral densities (CPSDs), which are supplied as a SDynPy PowerSpectralDensityArray
3. TransientSourcePathReceiver – this SPR object type is intended for responses that are formatted as time traces, which are supplied as a SDynPy TimeHistoryArray

All the SPR object types perform ISE with FRF matrix inversion in the frequency domain. However, the exact implementation is dependent on the response type. The LinearSourcePathReceiver uses a normal inverse problem. As an example, the standard pseudo-inverse method for linear spectra is shown in (1).

$$\{F\} = [H]^{+} \{X\} \tag{1}$$

In this equation, $\{F\}$ is a vector of force spectra, $[H]$ is a matrix of FRFs, $\{X\}$ is a vector of response spectra, and $[\cdots]^{+}$ indicates a pseudo-inverse of the given quantity. Note that all these functions are 3D arrays with a frequency dependency, but this notation was excluded for brevity. The PowerSourcePathReceiver uses an inverse problem that has been adapted for CPSDs. As an example, the standard pseudo-inverse method for CPSDs is shown in (2).

$$[G_{ff}] = [H]^{+} [G_{xx}] [H]^{+^*} \tag{2}$$

In this equation, $[G_{ff}]$ is a square matrix of force CPSDs, $[G_{xx}]$ is a square matrix of response CPSDs, and $[\cdots]^{*}$ indicates the conjugate transpose of the given quantity. Note that a vector of power spectral densities (PSDs) can be supplied to the PowerSourcePathReceiver object instead of a matrix of CPSDs. However, a so-called "buzz CPSD" array [3] must be supplied when initializing the object since the inverse processing function needs the responses to be organized as a square matrix.

The TransientSourcePathReceiver uses the same inverse problem format that the LinearSourcePathReceiver uses, except additional pre and post processing is required to handle the deconvolution aspects for the time domain force estimation. The pre and post processing is similar to the COLA procedure that is described in [23] and is applied to the data in the associated inverse_processing decorator function. An abridged description of the process is:

1. The full response time trace is zero padded to avoid any spoilage from the windowing in the COLA procedure.
2. The response time trace (from step one) is split into overlapped and windowed segments. The practitioner can set the segment block time, overlap, and window. The default block time is set to match the frequency resolution of the FRFs. The default window is a Tukey with an alpha parameter of 0.5, which uses a 25% overlap for a COLA condition. An example of this segmentation is shown in Figure 1.
3. The segmented time responses (from step two) are zero padded to avoid any convolution wraparound errors [24, 25].
4. The windowed, segmented, and zero padded time responses (from step three) are converted to the frequency domain with a discrete Fourier transform (DFT).
5. The FRFs are (sinc) interpolated to match the frequency domain responses from step four.
6. The frequency domain response (from step four) and interpolated FRFs (from step five) are used to estimate the forces with a frequency domain ISE method. The same methods as the LinearSourcePathReceiver can be used, an example of the standard pseudo-inverse method is shown in (1).
7. The frequency domain forces (for each time segment) are converted to the time domain using an inverse DFT.
8. The segmented time domain forces are recompiled into a single time trace for the whole time.

This COLA procedure is used instead of pure deconvolution [26], adaptive filtering [27], or modal methods [28, 29] for the following reasons:

1. The COLA procedure uses the same force estimation methods as linear spectra, which has been widely researched. Further, this problem form is the same as the supervised learning approach for machine learning [30], where there has been a dramatic amount of research on inverse problems.
2. Pure deconvolution can be extremely computationally inefficient, especially for long time responses.
3. The COLA procedure directly uses FRFs, which eliminates the need for additional assumptions and data processing to develop a modal description of the system (when the FRF data is based on experiments).

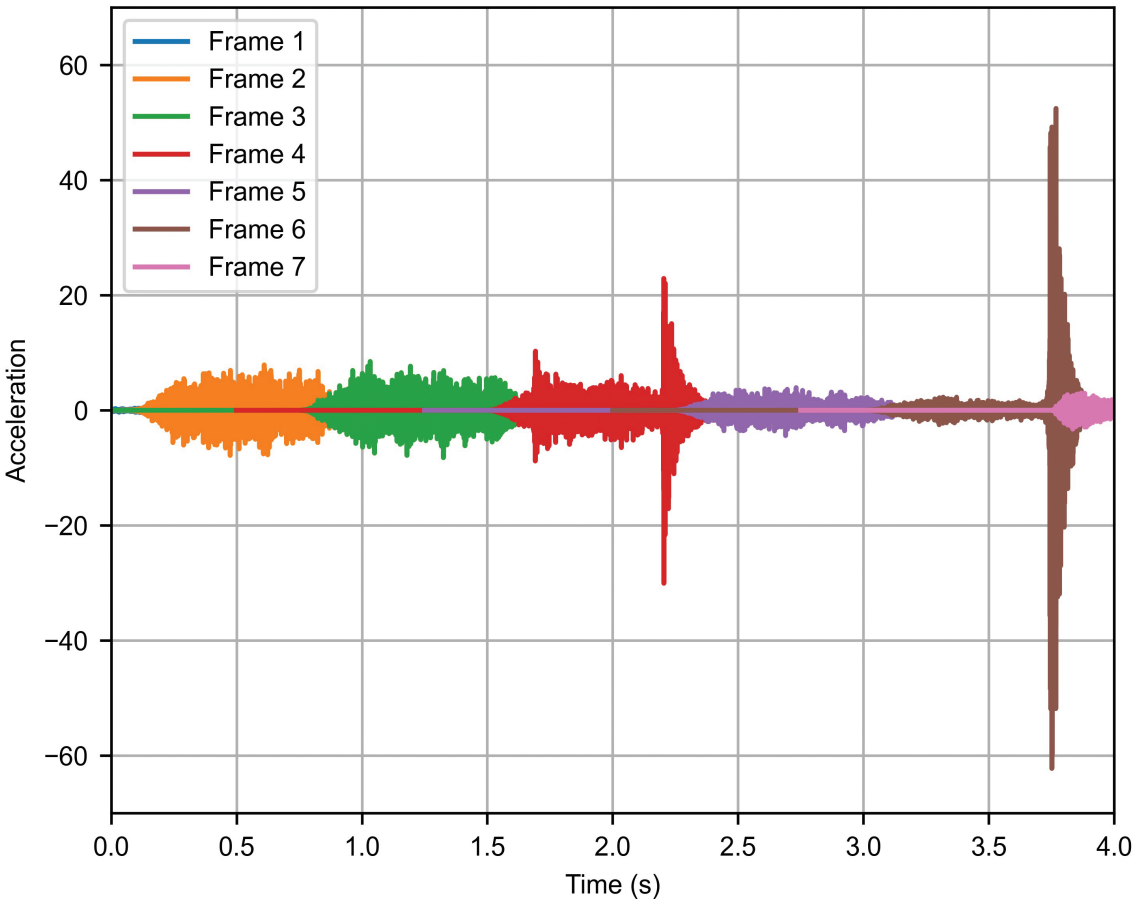


Fig. 1 Example signal segmentation for COLA procedure

It is important to note that while the force estimation is being performed in the frequency domain, it is still a deconvolution process where the FRFs can be viewed as finite impulse response (FIR) filters. As such, the practitioner must be careful to avoid issues related to non-causal filtering. The most significant of these issues are apparent non-causalities in the FRFs [31, 32]. There are not any techniques in ForceFinder to mitigate these apparent non-causalities, but there are methods in SDynPy that can be used to pre-process the FRFs before SPR object initialization.

Transformations in ForceFinder

Transformations have been implemented in ForceFinder for two main purposes:

- VPTs, like what is standard in TPA and substructuring [20, 21]
- DOF weightings [33] and normalization

The transformations are referred to as `response_transformation` and `reference_transformation`, depending on if they are applied to the response or reference DOFs (for the FRFs, responses, and forces). The transformations must be supplied as SDynPy Matrix objects, where the transformed quantities are on the rows of the matrix and the physical quantities are on the columns of the matrix (i.e., the transformation should be formatted to convert physical quantities to transformed quantities). Like everything else, the transformations will be automatically organized to match the DOFs in the SPR object. The practitioner can choose whether to apply transformations to the ISE problem by setting the `use_transformation` kwarg to true or false when estimating sources. The default argument is true, meaning that the transformations will be applied to

the ISE problem. Note that the SPR object defaults to an identity transformation matrix if one is not supplied. As such, the practitioner has the option to supply only a response transformation, only a reference transformation, or no transformations.

The derivation for the transformations in ForceFinder is shown below to clearly describe the implementation. The most basic transformations are applied to time traces or linear spectra of the responses and forces, as shown in (3) and (4), respectively.

$$\left\{\hat{X}\right\} = \left[T_{response}\right]\{X\} \tag{3}$$

$$\left\{\hat{F}\right\} = \left[T_{reference}\right]\{F\} \tag{4}$$

In these equations, $\widehat{\cdots}$ indicates a transformed quantity, $[T_{response}]$ is the transformation that is applied to the response DOFs, and $[T_{reference}]$ is the transformation that is applied to the reference (force) DOFs. The transformations are adapted to CPSDs, as shown in (5). For brevity, this derivation is only shown for responses. However, the same method is used for both force and response CPSDs.

$$\begin{gathered}
\left[\widehat{G_{xx}}\right] = \left\{\hat{X}\right\}\left\{\hat{X}\right\}^{*} \\
\downarrow \\
\left[\widehat{G_{xx}}\right] = \left(\left[T_{response}\right]\{X\}\right)\left(\left[T_{response}\right]\{X\}\right)^{*} \\
\downarrow \\
\left[\widehat{G_{xx}}\right] = \left[T_{response}\right]\{X\}\{X\}^{*}\left[T_{response}\right]^{*} \\
\downarrow \\
\left[\widehat{G_{xx}}\right] = \left[T_{response}\right]\left[G_{xx}\right]\left[T_{response}\right]^{*}
\end{gathered} \tag{5}$$

Inserting the FRF equation of motion into (3) shows how the response transformation is applied to the response DOFs in the FRF matrix, as shown in (6).

$$\begin{gathered}
\left\{\hat{X}\right\} = \left[T_{response}\right]\{X\} = \left[T_{response}\right]\left[H\right]\{F\} \\
\downarrow \\
\left\{\hat{X}\right\} = \left[\hat{H}\right]\{F\},\ where\ \left[\hat{H}\right] = \left[T_{response}\right]\left[H\right]
\end{gathered} \tag{6}$$

The reference transformation is applied to the FRFs in a similar manner, except the transformed forces are applied to the FRF equation of motion, as shown in (7).

$$\begin{gathered}
\left\{\hat{F}\right\} = \left[T_{reference}\right]\{F\} \\
\downarrow \\
\{F\} = \left[T_{reference}\right]^{+}\left\{\hat{F}\right\} \\
\downarrow \\
\{X\} = \left[H\right]\{F\} = \left[H\right]\left[T_{reference}\right]^{+}\left\{\hat{F}\right\} \\
\downarrow \\
\{X\} = \left[\hat{H}\right]\left\{\hat{F}\right\},\ where\ \left[\hat{H}\right] = \left[H\right]\left[T_{reference}\right]^{+}
\end{gathered} \tag{7}$$

The reference and response transformations are simultaneously applied to the FRFs by combining (6) and (7), as shown in (8).

$$\left[\hat{H}\right] = \left[T_{response}\right]\left[H\right]\left[T_{reference}\right]^{+} \tag{8}$$

As previously mentioned, DOF weightings can also be applied to the reference and response transformations. This is done by supplying a vector of weights to the apply_response_weighting or apply_reference_weighting methods for the SPR object. The weights are applied to the transformation matrix through the method that is shown in (9).

$$\left[T^{weighted}\right] = diag\left(\{W_{transformed}\}\right)\left[T^{original}\right]diag\left(\{W_{untransformed}\}\right) \tag{9}$$

In this equation, $\left[T^{weighted}\right]$ is the transformation matrix with the weightings applied, $diag\left(\cdots\right)$ is the operator that converts a vector to a diagonal matrix, $\{W_{transformed}\}$ is a vector of weightings that are applied to the transformed DOFs, $\left[T^{original}\right]$ is the unweighted transformation matrix, and $\{W_{untransformed}\}$ is a vector of weightings that is applied to the untransformed DOFs. Note that the practitioner can choose to optionally supply weightings for the either the transformed DOFs, untransformed DOFs, or both.

The transformations are applied to the ISE problem in the associated inverse_processing decorator function for all the SPR object types with the following process:

1. Apply the response transformation to the target responses using (3) or (5), depending on the response type.
2. Apply the reference and response transformations to the FRFs using (8).
3. Estimate the transformed forces using the selected ISE method.
4. Convert the transformed forces to physical forces by inverting the transformation equation as shown in (10) for linear spectra and time traces or (11) for CPSDs.

$$\{F\} = [T_{reference}]^{+}\left\{\hat{F}\right\} \tag{10}$$

$$[G_{ff}] = [T_{reference}]^{+}\left[\widehat{G_{ff}}\right][T_{reference}]^{+^{*}} \tag{11}$$

Note that the SPR object always stores the FRFs, responses, and forces as physical quantities, but that the transformed quantity can be called by the associated "transformed" class attribute. For example, the training response with the transformation applied can be called via spr_object.transformed_training_response. Also note that transformations are only defined in ForceFinder for the "training" quantities of an SPR object.

ISE Methods in ForceFinder

Several ISE methods have been implemented in ForceFinder, including methods with and without automatic hyperparameter tuning. The ISE methods with manual hyperparameter tuning include: standard pseudo-inverse, truncated singular value decomposition (TSVD) [7, 8], and Tikhonov regularization [9]. The manual methods have been implemented for all the SPR object types and leverage the existing FRF matrix inversion code from SDynPy.

The ISE methods with automatic hyperparameter tuning include: TSVD, Tikhonov regularization, and the elastic net [15, 34]. Note that not all the ISE methods have been implemented for all the SPR object types. Different hyperparameter tuning methods have been implemented for the different ISE methods. The implemented tuning methods for the different ISE methods are detailed in Table 1.

Table 1 Hyperparameter tuning methods for the different ISE methods

Tuning Method	TSVD	Tikhonov Regularization	Elastic Net
L-curve [11, 35]	Yes	Yes	No
Cross Validation [36, 37]	No	Yes	No
Information Criterion [38]	No	No	Yes

Note that ForceFinder is under active development, so additional methods may be available at the time of reading. Additionally, the technical details for the tuning methods out of the scope of this paper and will not be described here. However, references are included in the docstrings for many of the ISE methods in ForceFinder, so the interested party can find technical details on the associated method. Lastly, it should be noted that the automatic tuning methods will typically perform the tuning for every frequency line and every time segment in the ISE problem unless otherwise noted.

Benchmarks in ForceFinder

Standardized and high-quality benchmarks are an important tool when developing and evaluating ISE methods. The value of benchmarks can be seen in a variety of engineering fields, including vehicle crash safety testing, vehicle efficiency ratings,

benchmarks for different large language models, etc. Some benchmarks have been implemented in ForceFinder to evaluate and compare the different ISE methods that have been implemented in the package, to help establish some standardized benchmarks for ISE.

At the time of writing, only two benchmarks have been implemented and they have only been applied to the LinearSourcePathReceiver object type. The goal for these benchmarks is to estimate a set of pseudo-forces acting on one version of a system (referred to as system A), then the estimated forces are used to predict the vibration response on a different version of the system (referred to as system B). Both benchmarks are based on finite element (FE) models of beams, where the data has been corrupted in one of two ways (depending on the benchmark):

- Modeling errors (for the mass, stiffness, and damping) are applied to the FRFs that are used to estimate the pseudo-forces
- Response noise (random noise and quantization error) is applied to the responses that are used to estimate the pseudo-forces.

Sketches of the A and B beam systems are shown in Figure 2 and Figure 3, respectively. These systems are composed of four components:

1. Source beam – this is the brown (bottom) line and dots in Figure 2 and Figure 3
2. Receiver beam – this is the turquoise (top) line and dots in Figure 2 and Figure 3
3. Ground springs – these are the yellow springs that connect the source beam to the ground, note that these are torsion and translating springs even though the torsion spring is not shown in the sketch
4. Connector springs – these are the red springs that connect the source and receiver springs, note that these are torsion and translating springs even though the torsion spring is not shown in the sketch

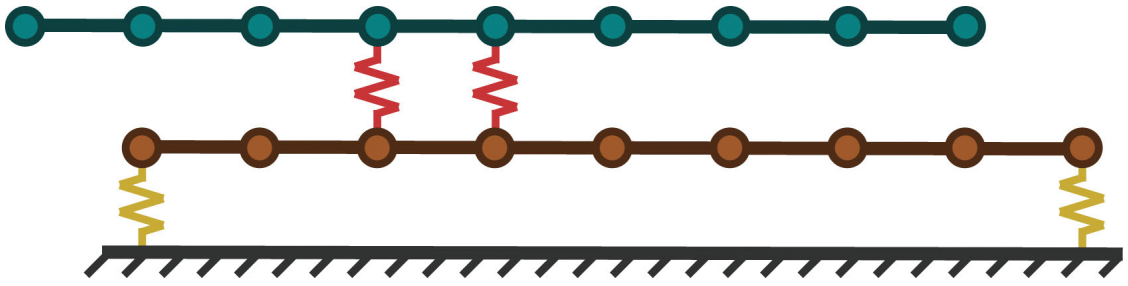


Fig. 2 Benchmark system A

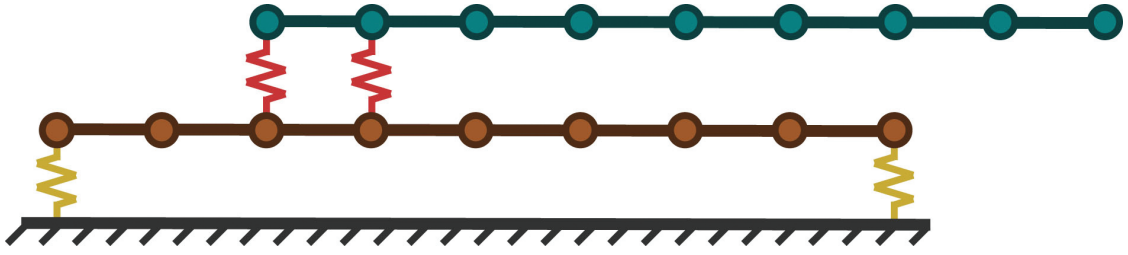


Fig. 3 Benchmark system B

For compactness and portability, the beam models, FRFs, and response data are generated when the benchmarks are performed. As such, no additional data is necessary outside of the benchmark scripts that are delivered with the package. These benchmarks are documented in a Jupyter notebook that is delivered with the package (in the benchmark folder of the root directory for the package). The general process for the benchmarks is as follows. Note that only major details will be listed here. The interested reader is directed to the documentation in the package for more technical details.

1. Target response will be generated for both system A and B, where the same forces are applied to all the DOFs on the source beam for both systems.

2. The data for the force estimate is corrupted, as described above.
3. Pseudo-forces are estimated on system A at the DOFs where the connector springs are attached to the source beam. Two quantities of interest (QoIs) are evaluated at this point:
    a. The accuracy of the reconstructed response (computed by applying the pseudo-forces to system A) compared to the training response (note that the corrupted data is used for this comparison)
    b. The force amplitudes
4. The pseudo-forces from system A are applied to system B to compute a predicted response. A single QoI is evaluated at this point:
    a. The accuracy of the predicted response compared to the target response that was computed in step 1

Two metrics are used to evaluate the response QoIs, since it is difficult to show the reconstructed/predicted response for every DOF in the benchmarks. The first metric is simply the response RMS level error in dB that is averaged over all the response DOFs in the benchmark. The second metric is slightly more complicated and attempts to show the spectral errors for all DOFs in a single curve. It is called the "RMS ASD error" and is computed with the following process:

1. The reconstructed/predicted spectra and truth spectra are converted to PSDs.
2. The narrowband PSDs are converted to 1/3 octave bands. This conversion is used to make the plots less cluttered.
3. The dB PSD error is computed for all the response DOFs in the benchmark.
4. A single error spectrum is created by computing the RMS of the PSD error over all the response DOFs in the benchmark.

The force QoI is evaluated simply by comparing the RMS level for each force DOF. The RMS ASD error and force QoI plots for the response noise benchmark are shown in Figures 4-6 below. Note that discussion about the benchmark QoIs was intentionally omitted (here and in the documentation that is included in the package) to eliminate any bias that is inherent to analysis of the data.

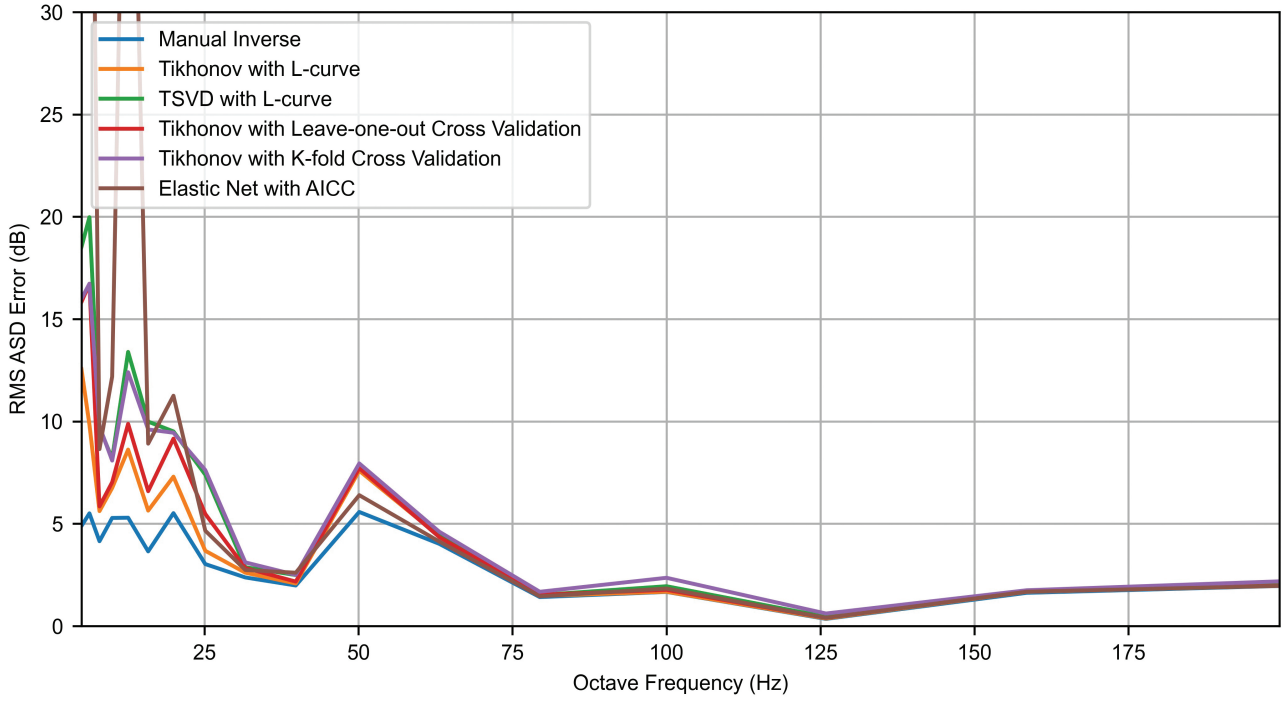


Fig. 4 RMS ASD error benchmark response QoI for system A

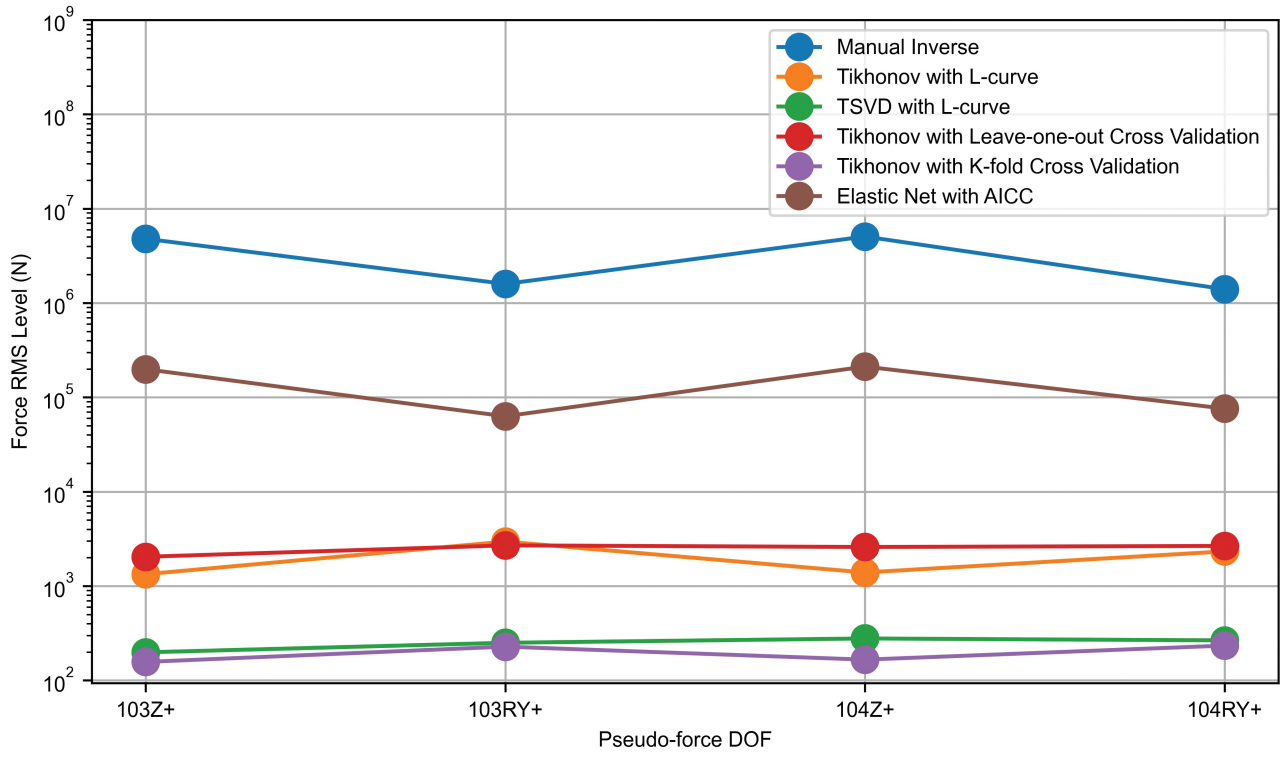


Fig. 5 Benchmark force RMS level QoI

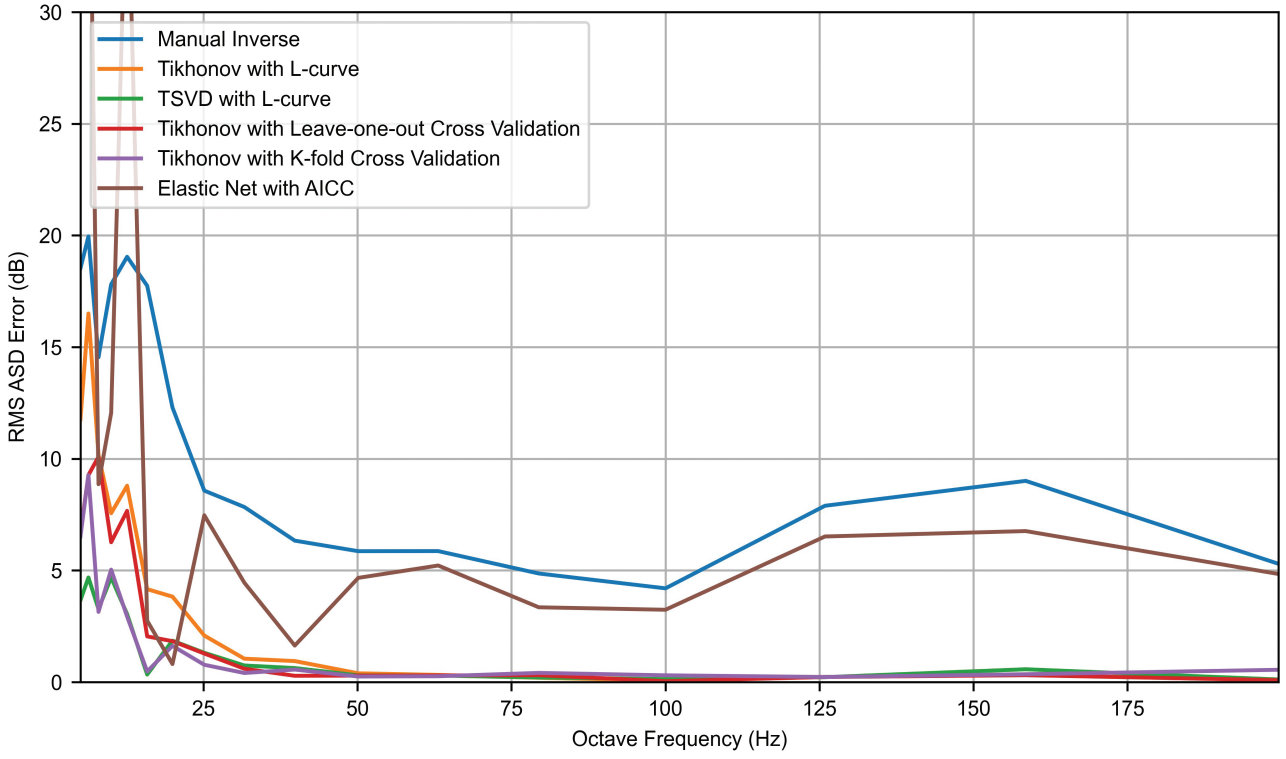


Fig. 6 RMS ASD error benchmark response QoIs for system B

It is recognized that no single benchmark can capture all the complexities that are experienced in ISE problems. As such, the ultimate goal is to have a suite of benchmarks in ForceFinder, where different benchmarks evaluate different issues that are experienced in ISE. However, it will take significant collaboration and community effort to develop these benchmarks. It is hoped that the existing benchmarks in ForceFinder demonstrate a framework that can be used to kick-off such collaborative efforts.

Basics of Using Forcefinder

This section will show some of the basic commands that are used in ForceFinder for completeness of the document. However, examples are currently being developed and will be posted to the code repository (as Jupyter notebooks) once they are available. As such, it is suggested that the reader use those examples to learn how to use the package. For brevity the commands will only be shown for the LinearSourcePathReceiver. Most of these commands, except for the error metrics, are applicable to any SPR object type (linear, power, or transient). The most basic commands are to import the package and create an SPR object, as shown below.

```
1  import forcefinder as ff
2
3  spr_object = ff.LinearSourcePathReceiver(frfs = system_frfs,
4                                           target_response = test_response,
5                                           training_response_coordinate = training_dofs)
```

In this code, system_frfs is a SDynPy TransferFunctionArray, test_response is a SDynPy SpectrumArray, and training_dofs is a SDynPy CoordinateArray. This code supplied the training response DOFs, which means that ForceFinder will automatically split the test_response data into the validation and training responses, as necessary. Next, it might be desirable to review the condition number and singular values of the FRF matrix to understand the conditioning of the ISE problem. Note that this code leverages method chaining in Python where the training_frf property of the SPR object is a SDynPy TransferFunctionArray. As such, the plotting commands are SDynPy methods, which are called by a SDynPy object, which is an attribute of the ForceFinder object.

```
6  spr_oject.training_frfs.plot_cond_num()
7  spr_object.training_frfs.plot_singular_values()
```

The ISE is done with a single line command, as shown below.

```
8  spr_object.auto_tikhonov_by_l_curve(use_transformation = False)
```

As is obvious in the name, the ISE method that is shown above uses Tikhonov regularization where the regularization parameter is automatically selected with an L-curve technique. The use_transformation kwarg is shown simply to demonstrate its use, the practitioner does not need to supply this parameter unless they wish to change its value from the default (which is true). Note that the ISE is done “in place”, which means that the sources are estimated and saved into the SPR object without needing to create a new object. The forces can be assigned to a SDynPy SpectrumArray via the “force” attribute of the SPR object.

With the sources estimated, the reconstructed response (the response computed from the estimated sources) should be compared to the supplied target response to ensure that the sources are capturing the dynamics of the system. This comparison can be done many ways, with the most obvious being a DOF-DOF spectrum comparison, as shown below.

```
9  import sdynpy as sdpy
10
11 sdpy.GUIPlot(spr_object.target_response, spr_object.reconstructed_target_response)
```

The package also includes some summary metrics that turn the comparison across many DOFs into a single spectrum.

A sample summary metric comparison is shown below for the RMS ASD error that was previously discussed for the benchmark.

```
12 rms_asd_error = spr_object.rms_asd_error(channel_set = 'training')
13 rms_ads_error.plot()
```

In this code, the summary metric is computed for the training response DOFs with the transformation applied. This summary metric could also be computed for the validation and target response DOF by changing the selected option for the channel_set kwarg. Also note that the summary metrics are returned as a SDynPy PowerSpectralDensityArray with a single entry for the one metric. This means that method chaining could be used to plot the metric in one line of code. However, two lines were used above so the metric could be saved to a new variable.

Lastly, it is sometimes valuable to have multiple copies of the of the SPR object so different ISE methods (or method options) can be evaluated. To do this, a copy of the SPR object must be explicitly made, as shown below. Note that the copy method must be used to obtain a copy of the object. Otherwise, much of the data in the objects will share the same memory and changes to one SPR object will result in changes to the other SPR object (because of the pass-by-reference mechanism in Python).

```
14 spr_object_copy = spr_object.copy()
```

Rattlesnake Integration

ForceFinder has also been integrated into a random vibration control law for the Rattlesnake vibration controller [39, 40]. Note that this control law is included as part of the ForceFinder package rather than the Rattlesnake package. As a result, the practitioner can use all the advanced inverse methods from ForceFinder in a MIMO vibration control problem. This integration also means that the practitioner can use the same code to perform offline predictions as the online control problem, which was challenging to do previously. It should be noted that the Rattlesnake integration is being actively improved, with a GUI control law is currently under development, so it will not be discussed in more detail here.

Conclusions and Future Work

ForceFinder introduces a framework that dramatically simplifies the ISE process by automatically organizing and storing all the data that is related to the ISE process in a single source path receiver object. As a result, many functions that may have previously taken several lines of code were simplified to single line method calls. An example of this simplification was shown above, where an SPR object was created, the FRFs were evaluated, sources were estimated, and response reconstruction accuracy was evaluated in thirteen lines of code.

Some benchmarks have also been implemented into the ForceFinder package to provide a standardized way to evaluate different ISE methods. The current benchmarks were primarily implemented to demonstrate what a benchmarking process might look like. It is hoped that members of the structural dynamics community will collaborate to develop high quality benchmarks that could be implemented in the package to better evaluate the different ISE methods.

Note that ForceFinder is being actively developed with work underway to add new ISE methods, benchmarks, example problems, quality assurance testing, and workflow improvements. Any community support for the development of ForceFinder is welcomed. This support could be in many forms, from code development to simply using the package and reporting bugs or workflow pain points. It is hoped that ForceFinder will provide a catalyst for improved collaboration and advance the state of the art in the ISE community.

Acknowledgments The author would like to thank the advice and contributions from many people. Most importantly, Thomas Brown, Dan Rohe, Keaton Coletti, and Ryan Schultz.

This article has been authored by an employee of National Technology & Engineering Solutions of Sandia, LLC under Contract No. DE-NA0003525 with the U.S. Department of Energy (DOE). The employee owns all right, title and interest in and to the article and is solely responsible for its contents. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United

States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or allow others to do so, for United States Government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan https://www.energy.gov/downloads/doe-public-access-plan.

References

1. J. Sanchez and H. Benaroya, "Review of force reconstruction techniques," Journal of Sound and Vibration, vol. 333, no. 14, pp. 2999-3018, 2014/07/07/ 2014, doi: https://doi.org/10.1016/j.jsv.2014.02.025.
2. M. V. van der Seijs, D. de Klerk, and D. J. Rixen, "General framework for transfer path analysis: History, theory and classification of techniques," Mechanical Systems and Signal Processing, vol. 68-69, pp. 217-244, 2016/02/01/ 2016, doi: https://doi.org/10.1016/j.ymssp.2015.08.004.
3. P. Daborn, "Smarter dynamic testing of critical structures," PhD dissertation, Aerospace Department, University of Bristol, 2014.
4. S. Carter and B. Owens, "Errors using six degree-of-freedom force estimates to translate environments system-to-system," in ISMA, Leuven, BE, 2022.
5. S. Carter and B. Owens, "Using Component Based TPA to Translate Vibration Environments Between Versions of the Round Robin Structure with FRFs Derived from Analytical Models," in Proceedings of IMAC XLI, the 41st International Modal Analysis Conference, Austin, 2023.
6. S. Carter, "A look at overfitting in source estimation problems," in Proceeding of IMAC XLIII, the 43rd International Modal Analysis Conference, Orlando, 2025.
7. D. Otte, "Development and evaluation of singular value analysis methodologies for studying multivariate noise and vibration problems," Department of Mechanical Engineering, Katholieke Universiteit Leuven, 1994.
8. A. N. Thite and D. J. Thompson, "The quantification of structure-borne transmission paths by inverse methods. Part 1: Improved singular value rejection methods," Journal of Sound and Vibration, vol. 264, no. 2, pp. 411-431, 2003/07/03/ 2003, doi: https://doi.org/10.1016/S0022-460X(02)01202-6.
9. A. N. Thite and D. J. Thompson, "The quantification of structure-borne transmission paths by inverse methods. Part 2: Use of regularization techniques," Journal of Sound and Vibration, vol. 264, no. 2, pp. 433-451, 2003/07/03/ 2003, doi: https://doi.org/10.1016/S0022-460X(02)01203-8.
10. H. G. Choi, A. N. Thite, and D. J. Thompson, "A threshold for the use of Tikhonov regularization in inverse force determination," Applied Acoustics, vol. 67, no. 7, pp. 700-719, 2006/07/01/ 2006, doi: https://doi.org/10.1016/j.apacoust.2005.11.003.
11. H. G. Choi, A. N. Thite, and D. J. Thompson, "Comparison of methods for parameter selection in Tikhonov regularization with application to inverse force determination," Journal of Sound and Vibration, vol. 304, no. 3, pp. 894-917, 2007/07/24/ 2007, doi: https://doi.org/10.1016/j.jsv.2007.03.040.
12. M. Aucejo and O. De Smet, "Bayesian source identification using local priors," Mechanical Systems and Signal Processing, vol. 66-67, pp. 120-136, 2016/01/01/ 2016, doi: https://doi.org/10.1016/j.ymssp.2015.05.004.
13. M. Aucejo and O. De Smet, "On a full Bayesian inference for force reconstruction problems," Mechanical Systems and Signal Processing, vol. 104, pp. 36-59, 2018/05/01/ 2018, doi: https://doi.org/10.1016/j.ymssp.2017.10.023.
14. M. Aucejo and O. De Smet, "An optimal Bayesian regularization for force reconstruction problems," Mechanical Systems and Signal Processing, vol. 126, pp. 98-115, 2019/07/01/ 2019, doi: https://doi.org/10.1016/j.ymssp.2019.02.021.
15. S. Carter, "Thoughts on Using Sparse Inverse Solutions in Transfer Path Analysis," in Proceedings of IMAC XLII, the 42nd International Modal Analysis Conference, Orlando, 2024.
16. P. E. Bofinger, J. Boelens, S. W. B. Klaassen, and D. de Klerk, "X-DoF: Automatic Degree-of-Freedom Subset Selection for Inverse Blocked Force Characterization," Journal of Vibration and Acoustics, vol. 147, no. 1, 2024, doi: 10.1115/1.4067081.
17. R. Schultz and P. Avitabile, "Shape-constrained Input Estimation for Efficient Multi-shaker Vibration Testing," Experimental Techniques, vol. 44, no. 4, pp. 409-423, 2020/08/01 2020, doi: 10.1007/s40799-020-00361-0.
18. S. Carter. "ForceFinder." https://github.com/sandialabs/forcefinder (accessed 2025).
19. D. Rohe, "SDynPy: A Structural Dynamics Python Library," in Proceedings of IMAC XLI, the 41st International Modal Analysis Conference, Austin, 2023.
20. M. Van der Seijs, D. van den Bosch, D. Rixen, and D. Klerk, "An improved methodology for the virtual point transformation of measured frequency response functions in dynamic substructuring," in Proceedings of the 4th International Conference on Computational Methods in Structural Dynamics and Earthquake Engineering, Kos Island, 2013, pp. 4334-4347, doi: 10.7712/120113.4816.C1539.
21. M. Haeussler and D. Rixen, "Optimal transformation of frequency response functions on interface deformation modes," in Proceedings of IMAC XXXV, the 35th International Modal Analysis Conference, Los Angeles 2017.
22. C. R. Harris et al., "Array programming with NumPy," Nature, vol. 585, no. 7825, pp. 357-362, 2020/09/01 2020, doi: 10.1038/s41586-020-2649-2.
23. R. Schultz and S. Carter, "A MIMO Time Waveform Replication Control Implementation," in Proceedings of IMAC XLI, the 41st International Modal Analysis Conference, Austin, 2023.
24. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, Numerical Recipes in FORTRAN 77: Volume 1, Volume 1 of Fortran Numerical Recipes: The Art of Scientific Computing. Cambridge: Cambridge University Press, 1992.
25. R. J. Thornhill and C. C. Smith, "Time Aliasing: A Digital Data Processing Phenomenon," Journal of Dynamic Systems, Measurement, and Control, vol. 105, no. 4, pp. 232-237, 1983, doi: 10.1115/1.3140664.
26. D. C. Kammer, "Input Force Reconstruction Using a Time Domain Technique," Journal of Vibration and Acoustics, vol. 120, no. 4, pp. 868-874, 1998, doi: 10.1115/1.2893913.
27. M. Sturm, A. Moorhouse, T. H. Alber, and F. Li, "Force reconstruction using an adaptive algorithm in time domain," in ISMA, Leuven, BE, 2012.

28. T. Carne, V. Bateman, and R. Mayes, "Force Reconstruction using a Sum of Weighted Accelerations Technique," in The 10th International Modal Analysis Conference, 1992.
29. P. Logan, P. Avitabile, and J. Dodson, "Reconstruction of External Forces Beyond Measured Points Using a Modal Filtering Decomposition Approach," Experimental Techniques, vol. 44, 08/06 2019, doi: 10.1007/s40799-019-00340-0.
30. T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition ed. New York: Springer New York, 2017.
31. M. Zvonkin, "Methods for checking and enforcing physical quality of linear electrical network models," Master of Science in Electrical Engineering, Electrical Engineering, MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY, 2015.
32. K. Coletti, S. Carter, and R. Schultz, "FRF Error Mitigation Methods for Time Waveform Replication," in the 43rd International Modal Analysis Conference, Orlando, FL, 2025.
33. R. Schultz, "Demonstration of Output Weighting in MIMO Control," Cham, 2024: Springer Nature Switzerland, in Dynamic Environments Testing, Volume 7, pp. 141-149.
34. T. Hastie, R. Tibshirani, and M. Wainwright, Statistical Learning with Sparsity: The Lasso and Generalizations (Monographs on Statistics and Applied Probability 143). Boca Raton, FL: Chapman & Hall/CRC, 2015.
35. P. C. Hansen and D. P. O'Leary, "The Use of the L-Curve in the Regularization of Discrete Ill-Posed Problems," SIAM Journal on Scientific Computing, vol. 14, no. 6, pp. 1487-1503, 1993, doi: 10.1137/0914086.
36. D. M. Allen, "The Relationship between Variable Selection and Data Agumentation and a Method for Prediction," Technometrics, vol. 16, no. 1, pp. 125-127, 1974, doi: 10.2307/1267500.
37. J. Shao, "Linear Model Selection by Cross-Validation," Journal of the American Statistical Association, vol. 88, no. 422, pp. 486-494, 1993, doi: 10.2307/2290328.
38. C. M. Hurvich and C.-L. Tsai, "Regression and time series model selection in small samples," Biometrika, vol. 76, no. 2, pp. 297-307, 1989, doi: 10.1093/biomet/76.2.297.
39. D. Rohe and R. Schultz, "Rattlesnake: An open-source multi-axis and combined evironments vibration controller," in Proceedings of IMAC-XL, the 40th International Modal Analysis Conference, Orlando, 2022.
40. D. Rohe, R. Schultz, and N. Hunter, "Rattlesnake User's Manual," Sandia National Laboratories, 2021.