SANDIA REPORT
SAND2022-8704
Printed June 29, 2022

Sandia National Laboratories

# Inverse Methods – Users Manual – 5.8

Sierra Inverse Methods Development Team:
Volkan Akcelik, Wilkins Aquino, Andrew Kurzawski, Cam McCormick, Clay Sanders, Chandler Smith, Ben Treweek, and Tim Walsh

Latest Software Release:
5.8-Release 2022-06-27

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.



Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods


## ABSTRACT

The inverse methods team provides a set of tools for solving inverse problems in structural dynamics and thermal physics, and also sensor placement optimization via Optimal Experimental Design (OED). These methods are used for designing experiments, model calibration, and verification/validation analysis of weapons systems. This document provides a user's guide to the input for the three apps that are supported for these methods. Details of input specifications, output options, and optimization parameters are included.

This page intentionally left blank.

# CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

# 1. Inverse Methods in Sierra/SD

## 1.1. *Inverse Solution Methods in* Sierra/SD

**Sierra/SD** supports a wide variety of different analyses or solution methods. Input consists of an **Exodus** mesh file and a text input file. Solution methods are specified in the text input file in the solution section. For details on using **Sierra/SD**, including analysis types and solution methods not related to inverse problems, the reader is directed to the **Sierra/SD** User's Manual [12].

The **Solution** section of the input file defines the type of physics to simulate. Analysis types relevant to inverse problems are shown in Table 1-1.

**Table 1-1.** – Inverse Solution Types.

| Solution Type | Description |
|---|---|
| eigen-inverse | Inverse solution to find material properties to produce given eigen solution |
| ModalFrf-inverse | Inverse solution to find load or power spectral density (PSD) to produce given modal frf |
| modaltransient-inverse | Inverse solution to find load to produce given modal transient |
| directfrf-inverse | Inverse solution to find load or material properties to produce given frequency response |
| transient-inverse | Inverse solution to find load or material properties to produce given transient solution |

## 1.2. *DirectFRF-Inverse Solution Case*

| Parameter | Type | Default | Description |
|---|---|---|---|

**Table 1-2.** – DirectFRF-Inverse Solution Case Parameters.

The `directfrf-inverse` solution method is used to solve an inverse problem for a direct frequency response analysis. As in a forward solution, most of the parameters of an inverse frequency response method are found in other sections[1]. The user provides complex displacements and/or pressures at a set of nodes in the model, and the solution to the inverse problem is a set of loads, materials, etc. that best correspond with the user's input.

[1]The forward solution supports a Padé expansion. This is not supported for inverse methods.

The forward problem is defined in Eq. (1.1)

$$\left( \underbrace{K + i\omega C - \omega^2 M}_{\equiv A(\omega)} \right) \overline{u} = \overline{f}(\omega) \tag{1.1}$$

where $\overline{u}$ is the Fourier transform of the response $u$, and $\overline{f}$ is the Fourier transform of the applied force.The inverse equation is identical, but must be solved with optimization subject to regularization because measurements are available only at a subset of the analysis degrees of freedom.

The basic requirements for a `directfrf-inverse` simulation are as follows:

**Optimization:** Control over the optimization problem is specified in the **optimization** block. See Sec. 1.7.1 for further details.

**Inverse-Problem:** The **inverse-problem** block provides the connection to the measurement data. It is also where `design_variable` is specified (e.g., load, material, etc.).

**Truth Table:** The truth table `data_truth_table` from the **inverse-problem** block is a list of the indices of the global node numbers (a.k.a. target nodes) where displacements or acoustic pressures are measured. See Sec. 1.7.3 for file format details.

**Data File:** Experimentally determined "target" displacements are read from `real_data_file` and `imaginary_data_file` specified in the **inverse-problem** block. See Sec. 1.7.3 for file format details.

**Frequency:** The frequencies at which the problem is solved are specified in the **frequency** block.

**Sierra/SD** uses the Rapid Optimization Library (ROL) for solving optimization problems. During the optimization solution ROL writes an output file, *ROL_Messages.txt* that contains convergence information. Section 1.7.8 contains a discussion of the output file that is written by ROL.

### 1.2.1. Load Identification

```
solution
  directfrf-inverse
end
inverse-problem
  design_variable = load
  data_truth_table = ttable.txt
  real_data_file = dataReal.txt
  imaginary_data_file = dataImag.txt
end
```

```
optimization
%  optimization_package = ROL_lib
  ROLmethod = trustregion
  TRstep = truncatedcg
  Max_iter_Krylov = 20
  Use_FD_hessvec = false
  Use_inexact_hessvec = false
  opt_tolerance = 1e-12
  opt_iterations = 2
end
loads
  sideset 301
    inverse_load_type = spatially_constant
    pressure=10
    function = 1
end
```

**Input 1.1. Direct frequency response load identification example input**

Specifying `design_variable = load` applies inverse methods to determine sideset loads which best correspond with the measured displacements and/or acoustic pressures provided by the user. The material and model parameters do not change during the solution. For structures, the loads are pressures or tractions[2], and for acoustics, the loads are acoustic accelerations. Note that for structures, inversion is based on the signed magnitudes of the tractions; the direction of each traction is fixed.

An example input deck is given in input 1.1. In addition to the input blocks discussed in the beginning of this section, there are several others specific to `design_variable = load`:

**Loads:** See Sec. 1.7.6 for a description of the inverse parameters in the **loads** block for load identification problems.

**Frequency:** For unknown loads, the frequencies at which the problem is solved are independent. That is, a separate load identification is performed at each frequency.

Pressure or traction load identification through *modal* frequency response is also supported for structures (see Sec. 1.4.1). Because modal solutions are significantly cheaper than direct solutions, one approach might be to begin the inverse optimization with `modalfrf-inverse`, then use the output as the initial guess for a follow-up `directfrf-inverse` case. Section 1.7.7 contains a discussion of the current limitations with inverse load methods.

[2]Moments and point forces are not currently supported.

### 1.2.2. Material Identification

Specifying `design_variable = material` with the `directfrf-inverse` method applies inverse methods to determine material parameters when provided with both loads and structural displacements and/or acoustic pressures in a given finite element model[3]. The load parameters do not change during the solution. As in the previous section, the forward problem is defined in Eq. (1.1), and the inverse equation is identical but must be solved with optimization subject to regularization because measurements are available only at a subset of the analysis degrees of freedom. The solution provides the material parameters for elements in the model that are specified to have unknown materials.

In addition to the input blocks discussed at the beginning of this section, there are several others specific to `design_variable = material`:

**Block:** See Sec. 1.7.4 for a description of the **block** specifications for material inverse problems.

**Material:** See Sec. 1.7.5 for a description of the provides **material** specifications for material inverse problems.

**Frequency:** For unknown materials, the same set of material properties apply for every frequency in the simulation, except in the case of frequency-dependent material properties.

Viscoelastic material identification is also supported using measured homogenized complex bulk and shear moduli. This capability is limited to structural-only problems where all material blocks are isotropic viscoelastic.

### 1.2.3. Multi-Experiment Material Identification

In the same manner as `design_variable = material` described in the previous section, `design_variable = multi_material` may be used to apply inverse methods in the frequency domain to determine material parameters. Here, multiple inverse problems are combined. For instance, if two different load and displacement conditions result in two separate responses for the same set of material properties, this method will use both responses to determine a single set of material properties.

Most parameters for a multi-experiment inverse frequency method are similar to those for a single-experiment inverse frequency method. The differences occur in the following sections:

**Loads:** The **loads** block must be empty for this solution case; anything the user specifies here will be overwritten by what they specify in the **load** block.

**Load:** Each **load** block provides a separate set of loads for each experiment individually.

[3] As the system matrices (and consequently the modes) change at every inverse iteration, `design_variable` cannot be set to `material` for `modalfrf-inverse` problems.

**Inverse-Problem:** In addition to the parameters discussed earlier, the **inverse-problem** section must include values for `nresponses` for the number of experiments and `loadID` to specify a list of loads, one for each experiment.

### 1.2.4. Circuit Parameter Identification For Piezoelectric Modeling

In piezoelectric modeling with electric circuits, the circuit parameters are real constants, and can be any combination of resistance, capacitance and inductance values. Specifiying `inverse_material_type = homogeneous` in a circuit block input can be used to identify these constants. This capability is currently only supported for the `directfrf-inverse` solution case. User must also specify upper and lower bounds for each circuit parameter used in a given circuit block. For example, input 1.2 inverts for three circuit parameters defined in Block 1. The keyword `inverse_material_type = homogenous` declares that circuit parameters in this block are treated as inverse parameters. The upper and lower bounds for each parameters are specificed with keywords `capacitance_bounds`, `resisance_bounds` and `inductance_bounds`. The upper and lower constants are user specified real values.

If needed, user can also identify circuit parameters concurrently with material model as decribed in 1.2.2.

```
BLOCK 1
 electrical_circuit
 inverse_material_type = homogeneous
 capacitance = 1e-9
 resistance = 50
 inductance = 1e-6
 capacitance_bounds = 1e-12 1e-6
 resistance_bounds = 1 100
 inductance_bounds = 1e-9 1e-3
END
```

**Input 1.2. Directfrf circuit parameter identification example input**

### *1.3. Eigen-Inverse Solution Case*

| Parameter | Type | Default | Description |
|---|---|---|---|
| **nmodes** | *Integer* | 10 | Number of modes to extract. |
| **shift** | *Real* | -1.0e6 | Shift to apply to matrix system to allow solving singular systems. |
| **untilfreq** | *Real* | Inf | Target frequency to reach. |
| **ModalFilter** | *string* | none | Modal filter to define modes to retain. |
| **modalAdjoint-Solver** | *gdsw*\| *camp*\| *both* | camp | Select solver for the inverse problem (eigenvector material identification only |

**Table 1-3.** – Eigen-Inverse Solution Case Parameters.

The `eigen-inverse` solution method is used to solve an inverse problem for an eigen analysis. In this solution method, only material identification is currently supported. Specifying `design_variable = material` applies inverse methods in the modal domain to determine material properties on a block or element when provided with modal frequencies and mode shapes. The user specifies some of the lowest modes of the structure, and optionaly the mode shapes of the structure at locations in the model.

The standard parameters for modal analysis also apply here. Modal filters may not be applied in `eigen-inverse` analysis. The analysis requires input both for measurement data and for control of various optimization parameters. See the following sections for details:

**Optimization:** Control over the optimization problem is specified in the **optimization** block. See Sec. 1.7.1 for further details.

**Inverse-Problem:** The **inverse-problem** block provides the connection to the measurement data. Of particular importance are the parameters `modal_data_file` and `modal_weight_table`, which are described further in Sec. 1.7.3. Also necessary is the parameter `design_variable = material`.

**Sierra/SD** uses the Rapid Optimization Library (ROL) for solving optimization problems. During the optimization solution ROL writes an output file, *ROL_Messages.txt* that contains convergence information. Section 1.7.8 contains a discussion of the output file that is written by ROL.

### 1.3.1. Eigenvalue Material Identification

```
solution
  eigen-materialid
    nmodes=12
    shift=-1e5
end
inverse-problem
  design_variable =  material
  modal_data_file =  modal_data.txt
  modal_truth_table =  modal_truth.txt
end
optimization
  optimization_package = ROL_lib
  ROLmethod = trustregion
  TRstep = secant
  scaleDesignVars = yes
  Max_iter_Krylov = 50
  Use_FD_hessvec = false
  Use_inexact_hessvec = true
  opt_tolerance = 1e-10
  opt_iterations = 30
end
```

**Input 1.3. Eigen material identification example input**

In the case of eigenvalue optimization, only the modal frequencies are included in the objective function. An example input is shown in input 1.3. The theory for this problem is available in [3]. The objective function for the eigen value problem is given as:

$$J(\boldsymbol{\lambda}, \boldsymbol{u}, \boldsymbol{p}) = \sum_{i=1}^{i=m} \left[ \frac{\beta_i}{2} \left( \frac{\lambda_i - \lambda_{mi}}{\lambda_{mi}} \right)^2 \right] + \mathcal{R}(\boldsymbol{p}). \tag{1.2}$$

Where $m$ is the number of modes, $\beta_i$ is zero or one, $\lambda_i$ is the computed eigenvalue, $\lambda_{mi}$ is the measured eigenvalue, and $\mathcal{R}(\boldsymbol{p})$ is the regularization term.

### 1.3.2. Eigenvector Material Identification

In this case, both the eigenvalues (modal frequencies) and eigenvectors (mode shapes) are matched in the inverse solution. A detailed description of the theory and implementation details of this solution case is given in [3]. To use this capability, it is necessary to specify the keywords `data_file` and `data_truth_table` for the eigenvector data and eigenvector truth table, respectively, which specify the modal shape amplitude data and truth table

information. The latter allows one to differentiate between a tri-axial and uni-axial accelerometer.

The objective function extends Eq. (1.2) by adding the eigenvector term.

$$J(\boldsymbol{\lambda}, \boldsymbol{u}, \boldsymbol{p}) = \sum_{i=1}^{N} \left[ \frac{\beta_i}{2} \left( \frac{\lambda_i - \lambda_{mi}}{\lambda_{mi}} \right)^2 + \frac{\kappa_i}{2} \frac{\|\boldsymbol{u}_i - \boldsymbol{u}_{mi}\|^2_{\boldsymbol{Q}}}{\|\boldsymbol{u}_{mi}\|^2_{\boldsymbol{Q}}} \right] + \mathcal{R}(\boldsymbol{p}) \,, \tag{1.3}$$

where $\boldsymbol{u}_i$ is the computed eigenvector, $\boldsymbol{u}_{mi}$ is the measured eigenvector, and $\boldsymbol{Q}$ is the observation matrix obtained from the truth table.

The issues an implementation must handle include repeated modes, crossing modes, the singular adjoint linear system, and eigenvector scaling.

**1.3.2.1. Repeated Modes** When the computed or measured data contain two repeated eigenvalues, the associated eigenvectors are not determanistic and steps are taken to orthogonalize these modes with respect to the measured data. The measured data must be sufficient enough for this orthogonalization for the inverse problem to converge. Note that repeated modes can occur at iterations in the inverse process even when no repeated modes are present in the measured data.

**1.3.2.2. Mode Swapping/Crossing** If the computed eigenvalues of a structure are ordered from smallest to largest, the ordering of mode shapes will typically change as the material parameters are varied. This also causes 10non-differentiability in the objective function, which causes difficulties in gradient-based optimization. Mode tracking refers to maintaining a correspondence of eigenpairs (eigenvalue and eigenvectors) between an original and an updated system throughout changes in the eigenproblem. Measured data is often incomplete, having only a few measured data points (physical accelerometer locations) on a model with millions of degrees of freedom. An incorrect mode swap results in a discontinuity in the slope of the objective function.

A mode tracking algorithm is used to minimize eigenvector misfit at each optimization step.

**1.3.2.3. Singular Solve** The Adjoint Solution is singular due to the fact that the eigenvector $u_i$ is in the kernel of the coefficient matrix. In order for a solution to exist, the right hand side must be orthogonal to $u_i$ . Additionally, if rigid body modes ($\lambda = 0$) or repeated mode are present, components of the corresponding eigenvectors must also be removed from the right hand side before the solve. Even when this is done, however, the resulting system of equations is singular and a Helmholtz (indefinite) problem, which presents significant computational cost and robustness challenges for iterative linear solvers.

The `modalAdjointSolver = camp` (default for eigenvector inversion) option enables a new solver that uses a modal superposition of the previously computed eigenvectors to solve

this system of equations. When using the `camp` solver, it is recommended to request more modes than contained in the measured data, and use the truth table file to remove these modes from the optimization part of the solution.

The *CAMP* block is used to define the parameters for the `camp` solver.

```
CAMP
   SC_OPTION 0
END
```

**Input 1.4. CAMP solver block**

Note that it is recommended to set the **sc_option** parameter to 0 for the `camp` solver.

**1.3.2.4. Computed Eigenvector Scaling** An important consideration in eigenvector optimization is that the mode shapes computed in Sierra/SD are by default *mass normalized*. Measured modal shape amplitudes, on the other hand, could present with very different scalings, since any eigenvector can be scaled by an arbitrary scale factor and will still be a valid eigenvector. Thus, the eigen-inverse solution method includes an automatic re-scaling of the computed mode shapes in the optimization so that they have the same norm as the measured mode shapes. This re-normalization allows them to be properly differenced in the objective function. We note that this internal re-scaling requires no user intervention.

If the norms of the measured eigenvectors differ substantially from the norms of the eigenvectors computed in Sierra/SD, then the re-scaling described in the previous paragraph is necessary to correctly determine the next iteration of the design variables. The scaled computed eigenvector $\tilde{\boldsymbol{u}}_i$ can be written as

$$\tilde{\boldsymbol{u}}_i = \alpha_i \boldsymbol{u}_i \,, \tag{1.4}$$

where $\alpha_i = |\boldsymbol{u}_{mi}|/|\boldsymbol{u}_i|$ such that the norm of $\tilde{\boldsymbol{u}}_i$ is identical to the norm of eigenvector $\boldsymbol{u}_{mi}$. With this change, the eigenvector term in Eq. (1.3) becomes

$$J(\boldsymbol{u}) = \sum_{i=1}^{N} \frac{\kappa_i}{2} \frac{\|\alpha \boldsymbol{u}_i - \boldsymbol{u}_{mi}\|^2_{\boldsymbol{Q}}}{\|\boldsymbol{u}_{mi}\|^2_{\boldsymbol{Q}}} \,, \tag{1.5}$$

A corresponding change to the gradient $J_{\boldsymbol{u}}$ is also required, but this change is not discussed here.

### 1.4. ModalFRF-Inverse Solution Case

| Parameter | Type | Default | Description |
|---|---|---|---|
| **nmodes** | *Integer* | 10 | Number of modes to extract. |
| **shift** | *Real* | -1.0e6 | Shift to apply to matrix system to allow solving singular systems. |
| **untilfreq** | *Real* | Inf | Target frequency to reach. |
| **ModalFilter** | *string* | none | Modal filter to define modes to retain. |
| **lfcutoff** | *Real* | -Inf | Exclude any modes below this frequency from the modal computation. Often used to exclude rigid body modes. |

**Table 1-4.** – ModalFRF-Inverse Solution Case Parameters.

The `modalfrf-inverse` solution method is used to solve an inverse problem for a modal frequency response analysis. The modal FRF method is similar to the direct FRF method, except the user must specify the number of modes `nmodes`. As in a forward solution, most of the parameters in an inverse modal frequency response analysis are found in other sections, and as in a `directfrf-inverse` problem, the user provides complex displacements and/or acoustic pressures at a set of nodes in the model.

The forward problem is defined in equation (1.1). The inverse equation is identical, but must be solved with optimization subject to regularization because measurements are available only at a subset of the analysis degrees of freedom.

The basic requirements for a `modalfrf-inverse` simulation are as follows:

**Optimization:** Control over the optimization problem is specified in the **optimization** block. See Sec. 1.7.1 for further details.

**Inverse-Problem:** The **inverse-problem** block provides the connection to the measurement data. It is also where `design_variable` is specified (e.g., load, material, etc.).

**Truth Table:** The truth table `data_truth_table` from the **inverse-problem** block is a list of the indices of the global node numbers (a.k.a. target nodes) where displacements or acoustic pressures are measured. See Sec. 1.7.3 for file format details.

**Data File:** Experimentally determined "target" displacements are read from `real_data_file` and `imaginary_data_file` specified in the **inverse-problem** block. See Sec. 1.7.3 for file format details.

**Frequency:** The frequencies at which the problem is solved are specified in the **frequency** block.

**Sierra/SD** uses the Rapid Optimization Library (ROL) for solving optimization problems. During the optimization solution ROL writes an output file, *ROL_Messages.txt* that contains convergence information. Section 1.7.8 contains a discussion of the output file that is written by ROL.

### 1.4.1. Load Identification

```
solution
  modalfrf-inverse
  nmodes 100
end
inverse-problem
  design_variable = load
  data_truth_table = ttable.txt
  real_data_file = data.txt
  imaginary_data_file = data_im.txt
end
optimization
  ROLmethod = trustregion
  TRstep = truncatedcg
  Max_iter_Krylov = 50
  Use_FD_hessvec = false
  Use_inexact_hessvec = false
  opt_tolerance = 1e-10
  opt_iterations = 2
end
loads
  sideset 6
    inverse_load_type = spatially_constant
    pressure=1
    function = 1
  sideset 6
    inverse_load_type = spatially_constant
    ipressure=1
    function = 2
end
function 1
  type linear
```

```
  data 1 3
  data 2 4
end
function 2
  type linear
  data 1 5
  data 2 6
end
```

**Input 1.5. Modal frequency response load identification example input**

Specifying `design_variable = load` applies inverse methods to determine sideset loads which best correspond with the measured displacements and/or acoustic pressures provided by the user. The material and model parameters do not change during the solution. For structures, the loads are pressures or tractions[4], and for acoustics, the loads are acoustic accelerations. Note that for structures, inversion is based on the signed magnitudes of the tractions; the direction of each traction is fixed.

An example input deck is given in input 1.5. In addition to the input blocks discussed in the beginning of this section, there are several others specific to `design_variable = load`:

**Loads:** See Sec. 1.7.6 for a description of the inverse parameters in the **loads** block for load identification problems.

**Frequency:** For unknown loads, the frequencies at which the problem is solved are independent. That is, a separate load identification is performed at each frequency.

Section 1.7.7 contains a discussion of the current limitations with inverse load methods.

### 1.4.2. PSD Load Identification

```
solution
  modalfrf-inverse
  nmodes 100
end
inverse-problem
  design_variable = psd_load
  data_truth_table = ttable.txt
  data_file = data.txt
end
optimization
  ROLmethod = trustregion
  TRstep = truncatedcg
```

[4]Moments and point forces are not currently supported.

```
  Max_iter_Krylov = 50
  Use_FD_hessvec = false
  Use_inexact_hessvec = false
  opt_tolerance = 1e-10
  opt_iterations = 2
end
loads
  sideset 6
    inverse_load_type = spatially_constant
    pressure=1
    function = 1
  sideset 6
    inverse_load_type = spatially_constant
    ipressure=1
    function = 2
end
function 1
  type linear
  data 1 3
  data 2 4
end
function 2
  type linear
  data 1 5
  data 2 6
end
```

**Input 1.6. Modal frequency response PSD load identification example input**

Specifying `design_variable = psd_load` applies inverse methods to determine a *load PSD* (power spectral density) as an output when provided with the PSD of acoustic pressures or structural displacements and a finite element model. The input is similar to modal FRF load identification, with two exceptions. First, the design variables must be defined such that there are independent real and imaginary parts of the force, traction, or pressue. Thus, there should be twice the number of design variables as the dimension of load PSD, corresponding to real and imaginary parts of the load. These design variables must be entered into the input file in the order of the load index. Furthermore, for each load index, the design variable for the real part should be immediately followed by the design variable for the imaginary part. Second, a different data file format is required, although the truth table format is identical. See Sec. 1.7.3 for further details.

An example input deck is given in input 1.6. In addition to the input blocks discussed in the beginning of this section, there are several others specific to `design_variable = psd_load`:

**Data File:** Experimentally determined "target" response PSDs are read from the `psd_data_file` described in Sec. 1.7.3.

**Loads:** See Sec. 1.7.6 for a description of the inverse parameters in the **loads** block for load identification problems. **loads** block.

**Frequency:** For unknown loads, the frequencies at which the problem is solved are independent. That is, a separate load identification is performed at each frequency.

## *1.5. ModalTransient-Inverse Solution Case*

| Parameter | Type | Default | Description |
|---|---|---|---|
| **nmodes** | *Integer* | 10 | Number of modes to extract. |
| **shift** | *Real* | -1.0e6 | Shift to apply to matrix system to allow solving singular systems. |
| **untilfreq** | *Real* | Inf | Target frequency to reach. |
| **ModalFilter** | *string* | none | Modal filter to define modes to retain. |
| **lfcutoff** | *Real* | -Inf | Exclude any modes below this frequency from the modal computation. Often used to exclude rigid body modes. |
| **time_step** | *Real* | | Time step size. |
| **nsteps** | *Integer* | | Number of time steps to take. |
| **start_time** | *Real* | 0.0 | Solution case start time. |
| **nskip** | *Integer* | 1 | Results output frequency. |
| **rho** | *Real* | 1 | Select time integrator. |
| **load** | *Integer* | | Load to apply during solution case. |
| **write_files** | *all*\| *none*\| *output*\| *history* | all | Controls which result files are written during this solution. |

**Table 1-5.** – ModalTransient-Inverse Solution Case Parameters.

```
solution
  modaltransient-inverse
  nsteps = 100
  time_step = 1e-3
```

```
  nskip = 1
  nmodes = 100
end
inverse-problem
  design_variable = load
  data_truth_table = ttable.txt
  data_file = dataReal.txt
  tikhonovParameter 1.0e-5
end
optimization
%  optimization_package = ROL_lib
  ROLmethod = trustregion
  TRstep = truncatedcg
  Max_iter_Krylov = 20
  Use_FD_hessvec = false
  Use_inexact_hessvec = false
  opt_tolerance = 1e-12
  opt_iterations = 2
end
loads
  sideset 301
    inverse_load_type = spatially_constant
    pressure=10
    function = 1
end
```

**Input 1.7. Modal transient load identification example input**

The `modaltransient-inverse` solution method is used to solve an inverse problem for a modal transient analysis. In this solution method, only load identification is supported. Specifying `design_variable = load` applies inverse methods to determine sideset loads with best correspond with measured displacements and/or acoustic pressures provided by the user in the modal time domain. This capability differs from load identification in a `transient-inverse` problem (Sec. 1.6.1) only in that modal superposition is used to reduce computation time. As with forward analysis, the modaltransient solution will converge to the direct solution as the number of modes increases. See the SierraSD verification manual for an example of this convergence[13].

The parameters for load identification in a direct `transient-inverse` problem also apply in the `modaltransient-inverse` case. The latter also requires the parameter `nmodes`, the number of eigen modes calculated in the forward solve, as well as any additional parameters needed for the eigen solution case. The eigen modes need only be calculated once, and then can be re-used for each inverse iteration. Note that the Tikhonov parameter can be used to mollify instability in the early time history.

An example is shown in input 1.7. The following input blocks are needed for `modaltransient-inverse` with `design_variable = load`:

**Optimization:** Control over the optimization problem is specified in the **optimization** block. See Sec. 1.7.1 for further details.

**Inverse-Problem:** The **inverse-problem** block provides the connection to the measurement data. It is also where `design_variable = load` must be specified.

**Truth Table:** The truth table (`data_truth_table` from the **inverse-problem** block is a list of the indices of the global node numbers (a.k.a. target nodes) where displacements or acoustic pressures are measured. See Sec. 1.7.3 for file format details.

**Data File:** Experimentally determined "target" displacements are read from `data_file` specified in the **inverse-problem** block. See Sec. 1.7.3 for file format details.

**Loads:** See Sec. 1.7.6 for a description of the inverse parameters in the **loads** block for load identification problems.

Section 1.7.7 contains a discussion of the current limitations with inverse load methods.

**Sierra/SD** uses the Rapid Optimization Library (ROL) for solving optimization problems. During the optimization solution ROL writes an output file, *ROL_Messages.txt* that contains convergence information. Section 1.7.8 contains a discussion of the output file that is written by ROL.

## *1.6. Transient-Inverse Solution Case*

| Parameter | Type | Default | Description |
|---|---|---|---|
| **time_step** | *Real* | | Time step size. |
| **nsteps** | *Integer* | | Number of time steps to take. |
| **start_time** | *Real* | 0.0 | Solution case start time. |
| **nskip** | *Integer* | 1 | Results output frequency. |
| **rho** | *Real* | 1 | Select time integrator. |
| **load** | *Integer* | | Load to apply during solution case. |
| **write_files** | *all*\| *none*\| *output*\| *history* | all | Controls which result files are written during this solution. |

**Table 1-6.** – Transient-Inverse Solution Case Parameters.

The `transient-inverse` solution method is used to solve in inverse problem for a time domain analysis. With a few exceptions, the parameters for the forward `transient` solution case apply to this solution method as well. The user provides a time series of displacements and/or pressures at a set of nodes in the model, and the solution to the inverse problem is a set of loads, materials, etc. that best correspond with the user's input.

The basic requirements for a `transient-inverse` simulation are as follows:

**Optimization:** Control over the optimization problem is specified in the **optimization** block. See Sec. 1.7.1 for further details.

**Inverse-Problem:** The **inverse-problem** block provides the connection to the measurement data. It is also where `design_variable` is specified (e.g., load, material, etc.).

**Truth Table:** The `data_truth_table` from the **inverse-problem** block is a list of the indices of the global node numbers (a.k.a. target nodes) where displacements or acoustic pressures are measured. See Sec. 1.7.3 for file format details.

**Data File:** Experimentally determined "target" displacements are read from `data_file` specified in the **inverse-problem** block. See Sec. 1.7.3 for file format details.

**Sierra/SD** uses the Rapid Optimization Library (ROL) for solving optimization problems. During the optimization solution ROL writes an output file, *ROL_Messages.txt* that contains convergence information. Section 1.7.8 contains a discussion of the output file that is written by ROL.

### 1.6.1. Load Identification

```
solution
  transient-inverse
  nsteps = 100
  time_step = 1e-3
  nskip = 1
end
inverse-problem
  design_variable = load
  data_truth_table = ttable.txt
  data_file = dataReal.txt
end
optimization
%  optimization_package = ROL_lib
  ROLmethod = trustregion
  TRstep = truncatedcg
  Max_iter_Krylov = 20
  Use_FD_hessvec = false
  Use_inexact_hessvec = false
```

```
  opt_tolerance = 1e-12
  opt_iterations = 2
end
loads
  sideset 301
    inverse_load_type = spatially_constant
    pressure=10
    function = 1
end
```

**Input 1.8. Transient Load Identification Example**

Specifying `design_variable = load` applies inverse methods to determine sideset loads which best correspond with the measured displacements and/or acoustic pressures provided by the user. The material and model parameters do not change during the solution. For structures, the loads are pressures or tractions[5], and for acoustics, the loads are acoustic accelerations. Note that for structures, inversion is based on the signed magnitudes of the tractions; the direction of each traction is fixed.

An example input deck is given in input 1.8. In addition to the input blocks discussed in the beginning of this section, there is another that is specific to `design_variable = load`:

**Loads:** See Sec. 1.7.6 for a description of the inverse parameters in the **loads** block for load identification problems.

Section 1.7.7 contains a discussion of the current limitations with inverse load methods.

### 1.6.2. Material Identification

Specifying `design_variable = material` with the `transient-inverse` method applies inverse methods to determine material parameters when provided with both loads and structural displacements and/or acoustic pressures in a given finite element model[6] The load parameters do not change during the solution, which provides the material parameters for elements in the model that are specified to have unknown materials.

In addition to the input blocks discussed in the beginning of this section, there are several others specific to `design_variable = material`:

**Block:** See Sec. 1.7.4 for a description of the **block** specifications for material inverse problems.

**Material:** See Sec. 1.7.5 for a description of the provides **material** specifications for material inverse problems.

[5] Moments and point forces are not currently supported.

[6] As the system matrices (and consequently the modes) change at every inverse iteration, `design_variable` cannot be set to **material** for `modaltransient-inverse` problems.

## 1.7. *Inverse Options in* Sierra/SD

Inverse problems optimize parameters to reproduce experimental results. Inverse methods include transient and direct frequency response load identification, direct frequency response material identification, and material identification from eigenvalues. The methods are based on solving optimization problems, with the goal to minimize the norm of the difference between measured and predicted data. More detail is provided in the references found in the theory notes. Inverse methods for identifying an unknown material (1.3, 1.2.2) or an unknown load (1.2.1, 1.6.1) require solution block input. There may be additional input required, such as the specification of test data results 1.7.3 and the specification of the parameters in the Optimization 1.7.1 and Inverse Problem 1.7.2 sections. Input 1.9 illustrates a *partial* input for a "directfrf-materialid" problem. Highlighted portions of the input are outlined below.

**Sierra/SD** uses the Rapid Optimization Library (ROL) as an optimization engine. Portions of the ROL documentation can be found on the Trilinos website.[7]

```
solution
  directfrf-inverse
end
optimization
  optimization_package = ROL_lib
  ROLmethod = trustregion
  TRstep = secant
  opt_tolerance = 1e-10
end
inverse-problem
  design_variable = material
  data_truth_table = ttable.txt
  real_data_file =  data.txt
  imaginary_data_file =  data_im.txt
end
block 1
  inverse_material_type=homogeneous
  material 1
end
block 2
  inverse_material_type=known
  material 2
end
material 1
  isotropic
```

[7] https://trilinos.org/packages/rol

```
    density 10
    G 1
    K 1
end
material 2
  isotropic
    density 1
    G 2
    K 2
end
```

**Input 1.9. Sample "directfrf-inverse" input for material identification. Portions of the input that are specific to inverse methods are emphasized.**

### 1.7.1. Optimization

The **optimization** section provides options to control the optimization strategy as part of an inverse method such as material identification. Parameters for the optimization section are listed in Table 1-7, and an example is shown in input 1.11.

**Sierra/SD** uses the Rapid Optimization Library (ROL) [9], which is a Trilinos [7] package for large-scale optimization. ROL is particularly well suited for the solution of optimal design, optimal control and inverse problems in large-scale engineering applications. The currently supported methods are trust region and line search, and the corresponding parameters for these methods are listed in Tables 1-7, 1-8 and 1-9. These are only a subset of the parameters available in ROL. We note that in tables 1-8 and 1-9, the abbreviations *tr* and *ls* stand for *trust region* and *line search*, respectively. We use these abbreviations to keep the parameter names succinct.

We note that for *source inversion* problems that involve **directfrf-inverse** , **transient-inverse** or **modaltransient-inverse** solution cases, we recommend using Krylov-based methods such as *line search* with the *newton-krylov* option, or *trust region* with *truncatedcg* option. An example for that case is given in 1.10.

```
optimization
  ROLmethod = trustregion
  TRstep = truncatedcg
  Max_iter_Krylov = 50
  opt_tolerance = 1e-8
  opt_iterations = 5
  scaleDesignVars = yes | no
END
```

### Input 1.10. Optimization Section Example for Source Inversion

In the case of *material inversion* problems, the best algorithm is problem-dependent, and may require some experimentation to arrive at the optimal parameters. We note that for material inverse problems, the parameter **scaleDesignVars** has been shown to significantly help convergence. This parameter defaults to `no`, but can be set to `yes` to facilitate convergence.

**Sierra/SD** currently uses ROL methods for unconstrained and bound-constrained optimization with line searches and trust regions (ROLmethod, boundConstraints, Table 1-7). To provide the context for the parameter tables, we review some notation and provide references to optimization textbooks.

Suppose $\mathcal{X}$ is a Hilbert space of functions mapping $\Xi$ to $\mathbb{R}$. For example, $\Xi \subset \mathbb{R}^n$ and $\mathcal{X} = L^2(\Xi)$ or $\Xi = \{1,\dots,n\}$ and $\mathcal{X} = \mathbb{R}^n$. We assume that the objective function $f : \mathcal{X} \to \mathbb{R}$ is twice continuously Fréchet differentiable and that the bound constraints $a, b \in \mathcal{X}$ are given with $a \le b$ almost everywhere in $\Xi$. We focus on methods for solving unconstrained and bound-constrained optimization problems,

$$\underset{x}{\text{minimize}} \quad f(x) \qquad \text{and} \qquad \underset{x}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad a \le x \le b\,,$$

respectively. The methods implemented in ROL utilize derivative information and two strategies for guaranteed (global) convergence from remote starting points, line searches and trust regions. We use the notation $\nabla f(x)$ to the note the gradient of $f$ at $x$ and $\nabla^2 f(x)$ to denote the Hessian of $f$ at $x$.

**Line-search methods**. Let $x_k$ be the $k$-th optimization iterate. For unconstrained problems, line search methods compute an update to $x_k$ in the form of

$$x_{k+1} = x_k + \alpha_k s_k\,,$$

where $s_k$ is a descent vector, and $\alpha_k > 0$ is a scalar. The vector $s_k$ can be computed using a variety of methods, including steepest descent, nonlinear conjugate gradients, quasi-Newton (secant) methods and Newton-Krylov methods, see [11, Ch. 3, 5, 8 and 6, resp.] and [6, Ch. 6.3]. Table 1-8 lists the parameters corresponding to the choice of the method to compute the descent vector $s_k$ (LSstep, Table 1-8). To compute the scalar $\alpha_k$, a line search approximately minimizes the one-dimensional function $\phi_k(\alpha) \coloneqq f(x_k + \alpha s_k)$, i.e., it approximately solves the optimization problem

$$\underset{\alpha}{\text{minimize}} \quad \phi_k(\alpha) \coloneqq f(x_k + \alpha s_k)\,.$$

In general, the approximate minimizer $\alpha_k$ must satisfy sufficient decrease and curvature conditions to guarantee global convergence [11, Ch. 3]. **Sierra/SD** uses the *cubic interpolation* line search from ROL, which includes a backtracking procedure that satisfies the Armijo sufficient decrease condition

$$\phi_k(\alpha_k) \le \phi_k(0) + c_1 \alpha_k \phi_k'(0) \qquad \Longleftrightarrow \qquad f(x_k + \alpha_k s_k) \le f(x_k) + c_1 \alpha_k \langle \nabla f(x_k), s_k \rangle_{\mathcal{X}}\,,$$

where $0 < c_1 < 1$, and does not require a curvature condition. An initial guess for the line-search parameter can be specified if steepest descent or nonlinear conjugate gradient methods are used for the computation of the descent vector $s_k$ (initial_LS_Par, Table 1-8). For bound constrained problems, the line search is a *projected* search. That is, the line search approximately minimizes the one-dimensional objective function

$$\phi_k(\alpha) = f(P_{[a,b]}(x_k + \alpha s_k)),$$

where $P_{[a,b]}$ denotes the projection onto the upper and lower bounds. Such line-search algorithms result in projected gradient, projected quasi-Newton and projected Newton algorithms (see below).

**Trust-region methods**. For unconstrained problems, given the $k$-th iterate $x_k$ trust-region methods compute the trial step $s_k$ by approximately solving the trust-region subproblem

$$\underset{s}{\text{minimize}} \quad \frac{1}{2}\langle B_k s, s\rangle_{\mathcal{X}} + \langle g_k, s\rangle_{\mathcal{X}} \qquad \text{subject to} \qquad \|s\|_{\mathcal{X}} \leq \Delta_k,$$

where $B_k \in L(\mathcal{X},\mathcal{X})$ is an approximation of $\nabla^2 f(x_k)$, $g_k$ approximates $\nabla f(x_k)$, and $\Delta_k > 0$ is the trust-region radius. The approximate minimizer $s_k$ must satisfy the fraction of Cauchy decrease condition

$$-\frac{1}{2}\langle B_k s, s\rangle_{\mathcal{X}} - \langle g_k, s\rangle_{\mathcal{X}} \geq \kappa_0 \|g_k\|_{\mathcal{X}} \min\left\{\Delta_k, \frac{\|g_k\|_{\mathcal{X}}}{1 + \|B_k\|_{L(\mathcal{X},\mathcal{X})}}\right\}$$

for some $\kappa_0 > 0$ independent of $k$. ROL implements several trust-region methods, including Cauchy point, dogleg, double dogleg, and truncated conjugate gradient methods, see [11, Ch. 4] and [6, Ch. 6.4]. Table 1-9 lists the parameters corresponding to the choice of the method (TRstep, Table 1-9). Additionally, the user can specify the initial trust-region radius, $\Delta_0$ (initial_TR_Radius, Table 1-9). For bound constrained problems, ROL employs projected gradient, projected secant, and projected Newton-type methods. These methods prune variables based on the binding set (see below) and run standard trust-region subproblem solvers on the remaining variables. To ensure sufficient decrease, ROL then performs a modified projected line search.

**Bound Constraints.** The bound constraint methods in ROL require the active set of an iterate $x_k$,

$$\mathcal{A}_k = \{\,\xi \in \Xi : x_k(\xi) = a(\xi)\,\} \cap \{\,\xi \in \Xi : x_k(\xi) = b(\xi)\,\}.$$

The active set is the subset of $\Xi$ corresponding to points in which $x_k$ is equal to the upper or lower bound. The complement of the active set (called the inactive set), $\mathcal{I}_k = \mathcal{A}_k^c = \Xi \setminus \mathcal{A}_k$, is the subset of $\Xi$ corresponding to points in which $x_k$ is strictly between $a$ and $b$. Given $\mathcal{A}_k$ and the gradient $g_k = \nabla J(x_k)$, we define the binding set as

$$\mathcal{B}_k = \{\,\xi \in \Xi : x_k(\xi) = a(\xi),\ -g_k(\xi) < 0\,\} \cap \{\,\xi \in \Xi : x_k(\xi) = b(\xi),\ -g_k(\xi) > 0\,\}.$$

The binding set contains the values of $\xi \in \Xi$ such that if $x_k(\xi)$ is equal to either the upper and lower bound, then $(x_k - g_k)(\xi)$ will violate bound. For both projected line-search and

trust-region methods, the step is computed by fixing the variables in the active set and only optimizing over the inactive variables. That is,

$$s_k|_{\mathcal{A}_k} = 0 \quad \text{and} \quad s_k|_{\mathcal{I}_k} \text{ are free.}$$

Considering active, inactive and binding variables results in poor overall performance. To circumvent this behavior, ROL employs $\epsilon$ variations of this set. Namely, if $\epsilon > 0$, then

$$\mathcal{A}_k^\epsilon = \{\xi \in \Xi : x_k(\xi) \leq a(\xi) + \epsilon\} \cap \{\xi \in \Xi : x_k(\xi) \geq b(\xi) - \epsilon\}$$
$$\mathcal{B}_k^\epsilon = \{\xi \in \Xi : x_k(\xi) \leq a(\xi) + \epsilon,\ -g_k(\xi) < 0\} \cap \{\xi \in \Xi : x_k(\xi) \geq b(\xi) - \epsilon,\ -g_k(\xi) > 0\}.$$

The $\epsilon$ inactive set is similarly defined as the complement of the $\epsilon$ active set. ROL dynamically controls $\epsilon$ so that as an algorithm approaches the optimal solution, $\epsilon$ decreases to zero. [8, 2, 10, 4, 5].

**Krylov methods**. Both line-search and trust-region methods may involve iterative methods of the Krylov type in the step computation. If such methods are requested, the stopping conditions for these sub-solvers can be defined through the parameters Absolute_Krylov_tol, Relative_Krylov_tol and Max_iter_Krylov described in Table 1-7. Scenarios requiring Krylov methods are triggered, for instance, if TruncatedCG is selected for a trust-region algorithm or if newtonkrylov is selected for a line-search algorithm.

| Parameter | type | default | Description |
|---|---|---|---|
| ROLmethod | select | required | LineSearch, TrustRegion |
| boundConstraints | Yes/No | Yes | Bound constraints on design variables |
| scaleDesignVars | Yes/No | no | Controls scaling of design variables by initial guess to make them nondimensional |
| opt_tolerance | real | 1e-10 | Gradient tolerance |
| opt_iterations | int | 20 | Maximum iterations; an iteration may include one solve of the Newton system |
| objective_tolerance | real | 1e-20 | Objective function tolerance |
| Absolute_Krylov_tol | real | 1e-6 | Krylov absolute tolerance |
| Relative_Krylov_tol | real | 1e-3 | Krylov relative tolerance |
| Max_iter_Krylov | int | 20 | Krylov iteration limit |

**Table 1-7.** – Optimization Section Parameters

| Parameter | type | default | Description |
|---|---|---|---|
| LSstep | select | secant | nonlinearcg, steepest, secant, newtonkrylov, newton (require useTransferMatrix=on 1-10) |
| initial_LS_Par | real | 1.0 | only applicable for nonlinearcg and steepest. |

**Table 1-8.** – Optimization Section Parameters for Line Search

| Parameter | type | default | Description |
|---|---|---|---|
| TRstep | select | Cauchypoint | TruncatedCG, CauchyPoint, Dogleg, DoubleDogleg |
| initial_TR_Radius | real | -1 | initial radius; if -1, ROL computes an initial value |

**Table 1-9.** – Optimization Section Parameters for Trust Region

```
optimization
  ROLmethod = trustregion
  TRstep = truncatedcg
  Max_iter_Krylov = 50
  opt_tolerance = 1e-8
  opt_iterations = 5
END
```

**Input 1.11. Optimization Section Example**

### 1.7.2. Inverse-Problem

The **inverse-problem** block of the input deck connects externally defined data describing the test. The format for the files is described in the Inverse Data Files section (1.7.3). This section also controls parameters of the optimization (such as regularization). Options for the **inverse-problem** section are included in Table 1-10.

#### 1.7.2.1. Regularization Parameters

**gradientSurfaceRegParameter** a penalty term, penalizing jumps in load input parameters between neighboring patches. Increasing the regularization parameter decreases checker-boarding on multi-patch outputs.

**tikhonovParameter** a penalty term penalizing large values of the design variables. A larger term forces smaller design variables.

**MECE_penalty** a penalty term, penalizes the size of the misfit error between the measured and predicted data. Unlike least squares methods, MECE methods to not strive to exactly reproduce the measured data. Increasing the penalty term decreases the misfit error.

| **Inverse Problem Parameters** | **Load Id** | | **Material Id** | |
|---|---|---|---|---|
| | **Transient** | **FRF** | **FRF** | **Eigen** |
| *General Parameters* | | | | |
| design_variable | R | R | R | R |
| data_file | R | NA | NA | NA |
| data_truth_table | R | R | R | NA |
| data_weight_table | NA | NA | NA | - |
| real_data_file | NA | R | R | NA |
| imaginary_data_file | NA | R | R | NA |
| modal_data_file | NA | NA | NA | R |
| modal_weight_table | NA | NA | NA | R |
| data_type | - | - | - | - |
| useTransferMatrix | NA | - | - | NA |
| *Regularization Parameters* | | | | |
| gradientSurfaceRegParameter | - | - | NA | NA |
| tikhonovParameter | - | - | - | - |
| gradientTikhonovRegularizationParameter | NA | NA | - | - |
| MECE_penalty | NA | NA | - | NA |
| *Multi-Experiment Parameters* | | | | |
| nresponses | NA | NA | NA | NA |
| loadID | NA | NA | - | NA |

**Table 1-10.** – Inverse-Problem parameters. Items marked with "-" are optional. Items marked 'R' are required, and items marked NA are not applicable.

#### 1.7.2.2. Multi-Experiment Parameters

**nresponses** the number of experients in a multi-experiment inverse problem. If not specified by the user, this parameter defaults to 1. This parameter must match the number of data files listed for 'real_data_file' and 'imaginary_data_file'.

**loadID** list of load identifiers, each corresponding to a separate experiment in a multi-experiment inverse problem. The length of this list must match the value specified for 'nresponses'.

**1.7.2.3. Transfer Matrix Option** The transfer matrix is a linear map between the design space and the state space for linear inverse problems such as source inversion. When the **useTransferMatrix** is set to true, the transfer matrix is internally computed and stored, and explicitly used to compute the gradient and the full hessian matrix. This option may signficantly decrease computational time at the expense of in core memory for select optimization problems with some or all of the following characteristics: 1. small number of design variables, 2. linear inverse problems with uncertainty, and 3. PSD source inversion problems (future release capability). In addition to increased computational performance, this option will allow the user to solve linear inverse problems using the full newton method (see Table 1-7).

Currently, the useTransferMatrix is only applicable to solution type **inverse-directfrf** problems, and must be flagged with `beta` when executing **Sierra/SD**.

### 1.7.3. Inverse Data Files

The interface for measured data involves several data files. The specific files needed depends on the solution method, as summarized in Table 1-10. This section describes the data files and data files formats. Input 1.12 provides an example of a frequency domain inverse problem.

```
inverse-problem
  design_variable = load | material
  data_truth_table = truthTable.txt
  real_data_file = dataReal.txt
  imaginary_data_file = dataImag.txt
  data_type = disp | accel | moduli | voltage
end
```

**Input 1.12. The inverse problem section for direct frequency response sets the names of the files that contain user specified data.**

In the case of a structural-only problem, the `data_type` can be set to either `disp` or `accel`, indicating that the incoming data is in the form of displacements or accelerations, respectively. In addition to displacement and acceleration data, the `data_type` can also be set to `moduli` for viscoelastic material identification problems. For acoustics-only problems, `data_type` is not applicable as the incoming data always corresponds to acoustic pressure. For models incorporating piezoelectric materials, setting `data_type` to `voltage` indicates that the experimental data is in the form of voltages. The parameter `data_type` defaults to `disp` if not specified.

#### Data Truth Table

The `data_truth_table` file contains the **global** node numbers (a.k.a. target nodes) where the experimental data measurements are given. The first line in the file contains the number of points where measurements are given, and the remaining lines contain the global node numbers where the experimental data is specified. If `data_type` is set to `moduli`, the `data_truth_table` file contains the node numbers where the displacements are computed.

Figure 1-1 provides a simple example of the truth table format for an acoustics-only problem, Figure 1-2 provides an example for a structural-only problem, and Figure 1-3 provides an example for a structural acoustics problem. Note that for acoustics-only, the file consists of the list of nodes where the acoustic measurements were taken. For voltage based problem, the file looks identical to an acoustic-only problem, where the listed nodes represent the locations where voltage measurements were taken. For structural-only problems, each line in the truth table contains 4 columns that indicate the node numbers where measurements were taken, and then 3 columns of binary (0/1) input that indicate which dofs (i.e. $x$, $y$, and $z$) are active in the optimization. For structural acoustics problems, Fig. 1-3 shows that each line contains 5 columns, with the first column again indicating the node number, and the remaining four columns contain either 1 or 0, which allows to turn on/off the $x$, $y$, $z$, and *pressure* degrees of freedom in the optimization.

The data truth table identifies the location of measurement data. For example, if the experimental data is collected at three microphones, which correspond to nodes 10, 120, and 3004, then the `data_truth_table` file is as follows.

```
3
10
120
3004
```

Thus there are a total of 4 lines in the file, even though the first line specifies three nodes for the measurement data. The global node numbers correspond to the global ID in the exodus input to **Sierra/SD**. The order of the nodes in this list corresponds to the order of the data in the corresponding real or imaginary data files. Other than that, there is no restriction on the ordering of the node numbers (i.e. they do not need to be in ascending or descending order).

**Figure 1-1.** – Sample Data Truth Table Input for Acoustic Problem

```
4
25  1  1  1
96  1  1  1
13  1  1  1
17  1  1  1
```

**Figure 1-2.** – Example Data Truth Table for Structures. There are 4 nodes, with numbers 25, 96, 13, 17. The second through fourth columns of 1, 1, 1 mean that all structural degrees of freedom are active, the fifth column is acoustic and implied to be zero.

```
4
5  0  1  0  0
6  0  1  0  0
7  0  0  0  1
8  0  0  0  1
```

**Figure 1-3.** – Example data truth table for Structural Acoustics. There are 4 nodes, with numbers 5 to 8. Nodes 5 and 6 are structural where only the $y$ component of displacement is measured. Nodes 7 and 8 are acoustic, where only the pressure is measured.

### Real Data File

For inverse problems in the frequency domain, the `real_data_file` contains the real component of the measurement data at each frequency, corresponding to the nodes that are specified in the `data_truth_table` file. For a multi-experiment inverse problem, several files must be specified.

An example of the `real_data_file` for an acoustics-only problem is shown in Figure 1-4. The first line of the file contains the number of nodes where measurement data is provided, followed by the number of frequencies of data. Starting on the second line, the real part of the data at the first node is given for all frequencies. In particularly, starting on the second line the data corresponds to the first node in the truth table list, not the node with the lowest ID number. Similarly, subsequent lines contain the real part of the data, at all frequencies, for the remaining nodes. Note that, since this an acoustics-only example, only one line of data is needed for each node in the truth table. The format for the data files describing voltages exactly matches the format of the acoustics-only data file.

We build on the small example given in Figure 1-1 that has measurements at nodes 10, 120, and 3004, and consider the case where there are 2 frequencies in the data set. The `real_data_file` file for an acoustics-only problem could look as follows

| | |
|---|---|
| 3 | 2 |
| 1.1 | 2.4 |
| 0.7 | 3.3 |
| 2.1 | 1.4 |

The actual values in the above table were chosen arbitrarily, but observe that there is one "header row" followed by 3 data rows. There are 2 columns, corresponding to the two frequencies of the measured data. The units of the measurements must correspond to appropriate units in the analysis. One row of data is required for each DOF in the truth table. Data is required even if the value in the truth table is zero. If the truth table value for a DOF is zero, the data is not used in the analysis.

**Figure 1-4.** – Sample Real Data File Input for an acoustics-only problem

An example of the `real_data_file` for a structural-only problem is shown in Figure 1-5. Starting on the second line, the real part of the data at the first node is given for all frequencies. In particularly the first node is the node first in the truth table list, not the node with the lowest ID number. The $x$ displacements for all nodes in the truth table is listed first, followed by the $y$ displacements for all nodes, followed by the $z$ displacements for all nodes.

For viscoelastic material identification problems in the frequency domain with `data_type` set to `moduli`, the `real_data_file` contains the real part of the shear and bulk modulus. An example of the `real_data_file` is shown in Fig. 1-6. The first line of the file is always

We build on the small example given in Figure 1-2 that has measurements at nodes 25, 96, 13, and 17, and consider the case where there is only 1 frequency in the data set. The `real_data_file` file for a structural-only problem could look as follows

```
12          1
1.1 // x-displacement for node 25
0.7 // x-displacement for node 96
2.1 // x-displacement for node 13
1.1 // x-displacement for node 17
0.7 // y-displacement for node 25
2.1 // y-displacement for node 96
1.1 // y-displacement for node 13
0.7 // y-displacement for node 17
2.1 // z-displacement for node 25
1.1 // z-displacement for node 96
0.7 // z-displacement for node 13
2.1 // z-displacement for node 17
```

The actual values in the above table were chosen arbitrarily, but observe that there is one "header row" followed by 12 data rows. There is only one column, corresponding to the single frequency of the measured data. The units of the measurements must correspond to appropriate units in the analysis. For structures these are units of displacement. Rows $2-5$ correspond to the $x$ components of the displacements at the nodes from the truth table, rows $6-9$ correspond to the $y$ components of displacement, and rows $10-13$ correspond to the $z$ components. Note that the $x$ components of displacements for all nodes are listed first, followed by the $y$ components for all nodes, followed by the $z$ components for all nodes. One row of data is required for each DOF in the truth table. Data is required even if the value in the truth table is zero. If the truth table value for a DOF is zero, the data is not used in the analysis.

**Figure 1-5.** – Sample Real Data File Input for a structural-only problem

2, followed by the number of frequencies of data. Currently, this feature only supports one frequency of data, hence the first row should read 2 1. The second and third lines are respectively the shear modulus and bulk modulus. There is only one column, corresponding to a single frequency of measured data.

The `real_data_file` file for a viscoelastic material identification problem with `data_type = moduli` could look as follows

```
2          1
5.7 // Shear modulus
3.1 // Bulk modulus
```

The actual values in the above table were chosen arbitrarily, but observe that there is one "header row" followed by 2 data rows.

**Figure 1-6.** – Sample Real Data File Input for a data type moduli problem

The frequencies of the measured data are specified in the **frequency** section. The frequencies given by **frequency** section must correspond to the frequencies where the experimental data was measured. These frequencies can be either uniformly or non-uniformly spaced, as specified in the **frequency** section.

#### Imaginary Data File

The `imaginary_data_file` has the exact same format as the `real_data_file` except that it contains the imaginary part of the data rather than the real part.

#### PSD Data File

In the context of PSD inversion, instead of specifying the real and imaginary data files, a single `psd_data_file` is needed, which contains the complex (Hermitian) PSD matrices for all the frequencies. The first two numbers in the `psd_data_file` must be the number of frequencies and the number of (measured) response degrees of freedom. The remainder of the file contains the PSD matrices, one matrix each for each frequency. The format of the PSD matrix should be in the natural row-oriented format, with each line containing a row of the matrix. The complex values should be entered in *(real part, imaginary part)* format, with space between the numbers. Sierra-SD automatically checks if each of the PSD matrices is Hermitian and positive definite, as each PSD matrix should be.

#### Data File

Transient time history data is stored in the `data_file`. The format is identical to the `real_data_file` except for the addition of the time step as the third column in the first

row. Each column of data now corresponds to a time step of analysis. An example is shown in Fig. 1-7. No interpolation is performed, and the measured data must exactly match the time steps of the analysis. As in the `real_data_file`, the rows of data are grouped as all of the $x$ components of displacements at the measured nodes, followed by all of the $y$ displacements, followed by the $z$ displacements.

We note that for time-domain inverse problems, the data in the `data_file` must be padded with zeros at the beginning, since the sensor data is typically started at time $t = 0$. The maximum time-of-flight from the input loads to the sensors can be estimated easily (an upper bound is fine). Then, dividing the maximum time-of-flight by the time step gives the number of zeros to be added to the beginning of each time history. Without these zeros, the forward problem would not be able to come up with loads that match the sensor data in the early time response, due to the finite wave propagation speed.

```
6    4    0.1
0.0  0.0  2.1  2.3
0.0  0.0  2.3  3.5
0.0  0.0  3.6  4.1
0.0  0.0  1.5  1.8
0.0  0.0  0.9  1.4
0.0  0.0  3.4  9.5
```

**Figure 1-7.** – Sample Transient Data File Input for a structural-only problem

### Modal Data File

The `modal_data_file` contains measured modal results for inversion with the `eigen` solution. The first line of the file is the number of eigenvalues. Each subsequent line contains only the eigen frequency of the measured mode. It is important that the simulation modes are in the same order as the test modes.

### Modal Weight Table

Not all computed eigenvalues may correspond to a test (or measured) mode. Further, even those modes identified may be more or less important. The `modal_weight_table` contains the weights applied to each computed mode.

The first line contains the number of weights, which must match the number of modes computed in the analysis, and each subsequent line contains the weight for the corresponding computed mode. All weights must be non-negative, and computed modes with no corresponding measured data should have zero weight.

#### Data Weight Table

In the case of eigenvector inversion (described in Sec. 1.3.2), the `data_weight_table` contains weights applied to each computed eigenvector error. This parameter allows for independent weighting of the eigenvalues and eigenvectors, but is not required; by default, it is the same as the `modal_weight_table`.

As above, the first line contains the number of weights, which must match the number of modes computed in the analysis, and each subsequent line contains the weight for the corresponding computed mode. All weights must be non-negative, and computed modes with no corresponding measured data should have zero weight.

### 1.7.4. Block section for Material Identification

For material inverse problems, the **block** section provides an additional option to control the optimization strategy. In particular, blocks can be specified as `known`, `homogeneous`, or `heterogeneous`, as shown in Table 1-11, and an example is shown in input 1.13.

The default behavior for the **inverse_material_type** keyword is `known`, which implies that the material parameters given in the corresponding material are fixed, and are not modified in the optimization process. In the case where all blocks are known, there is no need to solve the material inverse problem at all.

The remaining two options for the **inverse_material_type** keyword are:

- `homogeneous`. In this case, the material parameters are unknown and will be optimized during the inversion process, but are treated as constant over the entire block. This option is best used in the case when material properties are unknown in a block, but not expected to vary much over the block.
- `heterogeneous`. In this case, the material parameters are also unknown, will be optimized in the inversion process, but each element in the block will be given its own material properties. This is typically referred to as *spatially varying material properties*.

In input 1.13 Block 1 is `heterogeneous`, Block 2 is `homogeneous`, and Block 3 is `known`.

| Parameter | type | Description |
|---|---|---|
| inverse_material_type | string | block inverse type |

**Table 1-11.** – Block Section Parameters for Material Inversion

```
block 1
  material 1
  inverse_material_type heterogeneous
```

```
END
block 2
  material 2
  inverse_material_type homogeneous
END
block 3
  material 3
  inverse_material_type known
END
```

**Input 1.13. Block Section Example for Material Inversion**

### 1.7.5. Material section for Material Identification

For material inverse problems, the **material** section provides additional options to control the optimization strategy. Currently, for 3D elements, only `isotropic`, `isotropic_viscoelastic`, `orthotropic`, and `acoustic` material types are supported for material inverse problems.

We note that the parameters `G_bounds` and `K_bounds` only apply to elastic materials, whereas `Greal_bounds`, `Kreal_bounds`, `Gim_bounds`, and `Kim_bounds` only apply for viscoelastic materials. The parameters `Eij_bounds`, `Gij_bounds` and `Aij_bounds` apply only to orthotropic elastic materials, and the parameter `c0_bounds` applies only to acoustic materials.

For elastic, orthotropic elastic, and acoustic materials, the set of unknown material parameters may be customized by explicitly specifying the parameter `num_material_parameters` and a list of texttts following `material_parameters`. These texttts may include `bulk` and/or `shear` for elastic materials, `orthotropic` for orthotropic elastic materials, `sound_speed` for acoustic materials, and `rho` for all of the above.

For elastic materials, specification of `E_bounds` and `Nu_bounds` is permitted instead of specifying bounds for $G$ and $K$. When specifying bounds for the elastic modulus and Poisson ratio, the material parameters must also be input as $E$ and $nu$, and inversion must be requested over both elastic parameters.

For acoustic materials, the parameter `impedance_match` may also be specified. This has the effect of optimizing for sound speed $c_0$ and density $\rho_0$ under the condition where impedance $Z = \rho_0 c_0$ is constant.

**Orthotropic Material Inversion:** Unlike isotropic inversion, orthotropic inversion requires special care with respect to inadvertent material instabilities, i.e., during the inversion iterations, the elasticity tensor may not satisfy positive definiteness, potentially leading to the failure of even the forward solution. To avoid this, we parametrize the material tensor using the standard normal moduli $E_{ii}$, shear moduli $G_{ij}$, and special dimensionless parameters $A_{ij} = E_{ij}/\sqrt{E_{ii}E_{jj}}$. This is referred to as *Alpha* parametrization.

| Parameter | type | Description |
|---|---|---|
| G_bounds | real real | lower and upper bounds on shear modulus |
| K_bounds | real real | lower and upper bounds on bulk modulus |
| E_bounds | real real | lower and upper bounds on Young's modulus |
| Nu_bounds | real real | lower and upper bounds on Poisson's ratio |
| Greal_bounds | real real | lower and upper bounds on real part of shear modulus |
| Kreal_bounds | real real | lower and upper bounds on real part of bulk modulus |
| Gimag_bounds | real real | lower and upper bounds on imag part of shear modulus |
| Kimag_bounds | real real | lower and upper bounds on imag part of bulk modulus |
| Eij_bounds | 6 reals | lower and upper bounds on the three normal moduli |
| Gij_bounds | 6 reals | lower and upper bounds on the three shear moduli |
| Aij_bounds | 6 reals | lower and upper bounds on the three $\alpha$ parameters |
| density_bounds | real real | lower and upper bounds on density |
| c0_bounds | real real | lower and upper bounds on sound speed |
| impedance_match | real | value of $\rho_0 c_0$ to match for acoustic materials |
| num_material_parameters | int | (optional) number of material parameters |
| material_parameters | list of strings | (optional) `bulk`, `shear`, `rho`, `orthotropic`, and `sound_speed` |

**Table 1-12.** – Material Section Parameters for Material Inversion

Such parametrization lends to easier imposition of material stability constraint which is done through a single inequality constraint and several bound constraints. Further details will be provided in an upcoming SAND report. Notwithstanding the theoretical details, the user is asked to pay special attention to the definition of $A_{ij}$, when applying bound constraints on these parameters. Bound constraints can be specified as as shown in Table 1-12, and an example is shown in input 1.16. These parameters constrain the optimizer to work in a restricted space of possible design variable values, and prevent convergence to physically unrealistic values of the parameters. An alternative to Alpha parametrization is *Cholesky* parametrization, where the modulus matrix is parametrized through Cholesky factorization, avoiding the need of inequality constraints. This can be utilized by setting the flag `alphaparametrization` to `no`.

**Transverse Isotropic Material Inversion** is performed essentially through orthotropic inversion, by setting the flag `transverselyisotropic` to `yes`, followed by the plane of isotropy represented by a two-digit number, i.e., 12, 23, or 13, where 1,2,3 represent the three axes of material symmetry consistent with the coordinate system within the element block. Input 1.16 contains an example for transversely isotropic inversion.

Additional material parameters that can be optimized include `joint2g` elements and block proportional stiffness damping `blkbeta`. Since these parameters live in the **block** section rather than in the **material** section, their bound constraints are specified in the former. However, we specify their input syntax here since they are material parameters. An example of input syntax for `joint2g` and `blkbeta` inputs for material optimization are given in Table 1-13 and input 1.14.

The `joint2g` elements involve 6 parameters that can be optimized. The

`joint_truth_table` specifies which of these parameters will be included in the optimization solution. For example, in input 1.14, only the first (i.e. *x-component*) of the `joint2g` parameters will be optimized. The remaining parameters will stay fixed in the optimization, including those that are specified as *NULL*. In the case of `blkbeta`, the optimal *blkbeta* for the specified block will be computed and written to the result file.

| Parameter | type | Description |
|---|---|---|
| joint_elastic_bounds | real real | bounds on joint2g spring stiffnesses |
| joint_damper_bounds | real real | bounds on joint2g dashpot (damper) parameters |
| blkbeta_bounds | real real | bounds on block stiffness proportional damping |

**Table 1-13.** – Block Section Parameters for Material Inversion

```
block 50
  joint2g
  kx = elastic 2.474e5
  ky = elastic 2.e8
  kz = elastic 2.e8
  krx = NULL
  kry = NULL
  krz = NULL
  joint_elastic_bounds = 2.0e5  1e9
  joint_truth_table = yes no no no no no
end

block 51
  inverse_material_type = homogeneous
  material 1
  blkbeta 2.0e-4
end
```

**Input 1.14. Block Section Example for Material Inversion**

Initial guesses for the material parameters are also given in the **material** block. In the case of elastic materials, the initial guess is specified as the values for $G$ and $K$ that are input in the **material** block. In the example shown in input 1.15, the shear and bulk moduli are given initial guesses of 100 and 200, respectively.

For viscoelastic materials, the real and imaginary components of $G$ and $K$ are frequency-dependent. Thus the initial guesses are specified in functions that define the values of these parameters as a function of frequency. As shown in input 1.15, in this example functions $1-4$ specify the initial guesses for the real and imaginary components of $G$ and $K$.

```
material 1
  isotropic
  G_bounds 0 1e4
  K_bounds 0 1e4
  G 100.0
  K 200.0
  density 10.0
end
material 2
  isotropic_viscoelastic_complex
    Greal_bounds 0 1e4
    Kreal_bounds 0 1e4
    Gim_bounds 0 1e2
    Kim_bounds 0 1e2
  Greal = function 1
  Gim = function 2
  Kreal = function 3
  Kim = function 4
  density = 10.0
end
material 3
  isotropic
  E_bounds 1e-10 1e40
  Nu_bounds -0.9999 0.49999
  E 1.0e6
  Nu 0.25
  density 10.0
end
end
```

**Input 1.15. Material Section Example for Material Inversion**

```
material 4  // Full orthotropy
  density = 1.0
  orthotropic
  Cij = 4.0 1.0 2.0
  5.0 3.0
  6.0
  2.0
  1.0
  3.0

  Eii_bounds = 0.01 20.0 0.01 20.0 0.01 20.0
```

```
  Gij_bounds = 0.01 10.0 0.01 10.0 0.01 10.0
  Aij_bounds = 0.0 0.707 0.0 0.707 0.0 0.707
end

material 5 //Transeverse isotropy
  density = 1.0
  orthotropic
  alphaparametrization  no
  inequalityconstraints no
  transverselyisotropic yes 23
  Cij = 5  1  1
    16  6
  16
  5
  1
  1
  Eii_bounds = 1 100 1 10 1 100
  Gij_bounds = 0.1 5 0.1 5 0.1 5
end
```

**Input 1.16. Material Section Example for Orthotropic Material Inversion**

The `directfrf-inverse` method supports viscoelastic material identification using homogenized moduli data by setting `data_type` to `moduli` in the **inverse-problem** block (see section 1.7.3). This feature is only available for the `inverse_material_type` `homogenous` option. In addition, all material blocks, including known materials, must be specified as `isotropic_viscoelastic_complex`.

Figure 1-14 shows which material parameters can be optimized in the different solution procedures. In general, parameters associated with damping can only be optimized in **directfrf** solutions, and those associated with stiffness can be optimized in either **directfrf** of **eigen** .

**Damage Identification:** Damage identification is a design variable implemented for `directfrf-inverse` problems as a special case of elastic material identification. In damage identification, a damage phase field variable is used to interpolate between damaged (weak) and full-strength material, ideally converging to near-binary values to indicate presence of material damage. Damage ID is activated by `design variable = damage` in the **inverse-problem** block.

Damage identification employs techniques originated for topology optimization problems but may used for a variety of applications, including identification of weakened material regions, determination of contact area in a thin layer of elements at an interface, or two-phase material design. The elastic damage model uses a Solid Isotropic Material with Penalization (SIMP) model [1] that interpolates the isotropic elastic and mass density

| Parameter | directfrf-inverse | eigen-inverse |
|---|---|---|
| K | yes | yes |
| G | yes | yes |
| E | yes | yes |
| Nu | yes | yes |
| K_real | yes | no |
| K_imag | yes | no |
| G_real | yes | no |
| G_real | yes | no |
| Cij | yes | no |
| c0 | yes | no |
| density | yes | no |
| joint2g elastic | yes | yes |
| joint2g damper | yes | no |
| blkbeta | yes | no |

**Table 1-14.** – Table of Supported Material Parameters for Inverse Methods

properties in the unknown material between two phases using the scalar phase field variable $\beta \in [0,1]$, expressed as

$$G(\beta) = G_0 + (G_u - G_l)\beta^p \tag{1.6}$$
$$K(\beta) = K_0 + (K_u - K_l)\beta^p \tag{1.7}$$
$$\rho(\beta) = \rho_0 + (\rho_u - \rho_l)\beta^q, \tag{1.8}$$

where $\{G_u, K_u, \rho_u\}$ and $\{G_l, K_l, \rho_l\}$ are the upper and lower bounds for the bulk modulus, shear modulus, and density, respectively. Definition of different powers $p \geq q \geq 1$ for the elastic and mass density components renders elastic properties relatively weaker, for a given mass density, thus disincentivizing intermediate density values. Powers $p$ and $q$ are specified with `penalizationElasticity` and `penalizationMass`, respectively.

Bounds of the material interpolation are specified by setting limits for shear modulus `G_bounds`, bulk modulus `K_bounds`, and mass density `density_bounds` in the **material** block. Meanwhile, the initial damage phase field value is controlled using the `G`, with respect to `G_bounds`, as

$$\beta^{(0)} = \frac{(G - G_l)}{(G_u - G_l)}. \tag{1.9}$$

Initial `density` and `K` values must still be given, though are not used to determine the initial phase field value. The solution for $\beta$ is not available for output, but rather the material parameters are output as evaluated within the penalized elastic and mass density models using the phase field solution.

Damage identification is enabled for both homogeneous and hetereogeneous unknown material blocks. In the homogeneous unknown material block case, only the penalization

powers for the mass density and elastic properties must be specified. In heterogeneous unknown material blocks, additional filtering and projection operations are employed to control solution length scale and to encourage binary quality. Kernel filtering prevents development of mesh dependent or checkerboard (i.e. alternating 0-1 phase field density) patterns. In this strategy, a weighted kernel is convolved with the density field, producing a locally-averaged filtered field. The filter kernel radius is defined in the **inverse-problem** block as `filter_radius` (default = 0.1), which should be set no smaller than the minimum distance between element centroids in the unknown material block.

A Heaviside-approximating function is then used to map filtered values closer to 0 or 1 bounds and recover a more-binary valued field. The smooth Heavisde function is defined

$$\tilde{\beta}(\beta) := \frac{\tanh(\zeta\eta) + \tanh(\zeta(\beta - \eta))}{\tanh(\zeta\eta) + \tanh(\zeta(1 - \eta))}. \tag{1.10}$$

Here, the slope of the smooth Heaviside $\zeta > 1$ is specified by `smooth_heaviside_slope`, while its inflection threshold $\eta \in (0, 1)$ is specified by `smooth_heaviside_threshold`. Typically, modest values for `smooth_heaviside_slope` (5-10) can produce high contrast fields without creating an excessively severe projection, which can impede optimization convergence.

Below, we summarize the necessary parameters in the **material** and **inverse-problem** sections for damage identification problems.

| **Parameter** | **type** | **Description** |
|---|---|---|
| G | real | initial value of shear modulus (determines initial phase field value) |
| K | real | initial value of bulk modulus |
| density | real | initial value for mass density |
| G_bounds | real real | lower and upper bounds on shear modulus interpolation |
| K_bounds | real real | lower and upper bounds on bulk modulus interpolation |
| density_bounds | real real | lower and upper bounds on mass density interpolation |

**Table 1-15.** – Parameters in **material** section for Damage Identification

| **Parameter** | **type** | **Description** |
|---|---|---|
| design_variable | string | damage |
| penalizationElasticity | integer | elasticity penalty exponent (default = 3) |
| penalizationMass | integer | elasticity penalty exponent (default = 1) |
| filter_radius | real | filter kernel radius (default = 0.1) |
| smooth_heaviside_slope | real | smooth heaviside projection slope (default = 1) |
| smooth_heaviside_threshold | real | smooth heaviside inflection threshold (default = 0.5) |

**Table 1-16.** – Parameters in **inverse-problem** section for Damage Identification

### 1.7.6. Loads section for Load Identification

For inverse load problems, the **loads** section provides additional options to control the optimization strategy. Inverse loads are currently supported for acoustics and structures. Inverse acoustic loads are only supported for sidesets. Inverse structural loads are supported on both sidesets and nodesets. On sidesets, both pressures and traction loads may be optimized in the inverse problem. For nodesets, both forces and moments can be optimized, the latter only being applicable in the case when the nodeset in question has rotational degrees of freedom (e.g. a concentrated mass). Table 1-17 summarizes the load options which apply to inverse methods. The `inverse_load_type` options are detailed in Table 1-18.

The default behavior for the `inverse_load_type` keyword is `known`, which implies that the loads on that sideset or nodeset are fixed, and are not modified in the optimization process. In the case where all blocks were known, there would be no need to solve the inverse loads problem.

In input 1.17 sideset 1 is `known`, sideset 2 is a real-valued, unknown acoustic load that is `spatially_constant`, and sideset 3 is an imaginary-valued, unknown acoustic load that is `spatially_constant`. We note that in the case of a transient problem, the **loads** block in an inverse loads problem would look the same except that there would be no imaginary loads in that case.

For acoustic problems currently inverse acoustic loads are limited to the `acoustic_accel` option. The `acoustic_vel` keyword is not supported for acoustic loads.

Input 1.18 shows a similar example for a structural pressure, traction and force load case. This example contains a known pressure load, and unknown pressure, traction, and force loads. Note that in the case of traction and force loads, the direction of the traction load (in this case 111) is fixed, and only the function amplitudes are calculated in the inverse problem.

In input 1.18, functions $1-4$ contain the initial guesses for the load amplitudes for sidesets $1-3$ and forces/moments on nodeset 4. These load amplitudes are then refined during the optimization process. The resulting loads are written to a text file called *force_function_data.txt*, which could then be included in a subsequent forward or inverse loads case in a restart analysis.

| Parameter | type | Description |
|---|---|---|
| inverse_load_type | string | load inverse type |

**Table 1-17.** – Loads Section Parameters for Force Inversion

```
loads
  sideset 1
    acoustic_accel = 1
    function = 1
  sideset 2
```

| Parameter | Description |
|---|---|
| spatially_constant | Load amplitude is unknown and will be optimized during the inversion process, but is treated as constant over the entire sideset or nodeset. |
| spatially_variable | Load amplitude is unknown and will be optimized during the inversion process. Each dof on the sideset or nodeset is optimized. |
| known | default. No optimization performed. |

**Table 1-18.** – Inverse Load Type Options

```
    acoustic_accel = 1
    function = 2
    inverse_load_type = spatially_constant
  sideset 3
    iacoustic_accel = 1
    function = 3
    inverse_load_type = spatially_constant
end
```

**Input 1.17. Loads Section Example for Acoustic Force Inversion**

```
loads
  sideset 1
    pressure = 1
    function = 1
  sideset 2
    pressure = 1
    function = 2
    inverse_load_type = spatially_constant
  sideset 3
    traction = 1 1 1
    function = 3
    inverse_load_type = spatially_constant
  nodeset 4
    force = 0 1 1
    function = 4
    inverse_load_type = spatially_constant
  nodeset 4
    moment = 0 1 1
    function = 5
    inverse_load_type = spatially_constant
end
```

**Input 1.18. Loads Section Example for Structural Force Inversion**

### 1.7.7. Limitations for Inverse Load Problems

**Limitations:** There are a number of limitations which apply to transient load identification. These include the following.

1. Structural, acoustic and structural-acoustic domains may be addressed.
2. Only the simple Newmark integrator should be used, i.e. do not use generalized alpha integration and do not use the `rho` keyword.
3. Pressure is always applied along the surface normal, and tractions are applied along the direction specified in the **loads** block. Inverse methods do not support follower pressures.
4. Load identification applies to `acoustic_accel` loadings on acoustic sidesets, pressures/tractions on structural sidesets, and forces and moments on structural nodesets.
5. In force identification problems, a pressure, traction or force may be applied on a shell. The measured displacement fields in the truth table for shells can be applied to nodes with rotational DOFs; however, only displacement DOFs can be specified in the data files. Specifying rotational DOFs as measured data at nodes is not supported.

### 1.7.8. ROL Output for Inverse Problems

**Sierra/SD** uses the Rapid Optimization Library (ROL), which is part of Trilinos [7] for solving optimization problems. During the optimization process, ROL writes out a text file called *ROL_Messages.txt* that contains information about the convergence of the optimization solution. It is important to examine this file to assure that the solution is adequately converged.

An example of a *ROL_Messages.txt* is given in Figure 1-8. The first 2 lines show the optimization method that was used by ROL, and the following lines contain convergence information. Each line corresponds to a single optimization iteration. The first column shows the iteration number under the *iter* heading. The second column shows the value of objective function at that iteration, under the *value* heading. The third column shows the absolute norm of the gradient (i.e. the derivative of the objective function with respect to the optimization variables). In Figure 1-8 we only show the first three columns of output as these will typically be of most interest to the user, but the remaining columns contain information about step size, number of function evaluations, etc... (Denis, Drew, any input here would be great).

For a typical **Sierra/SD** user, the first three columns in the *ROL_Messages.txt* file will typically be of the most important to pay attention to. As the goal is to minimize the objective function, a substantial decrease in the second column should be observed. Also, the desired minimum of the optimization corresponds to a zero gradient, and thus the third column should be observed to be as close to zero as possible.

Newton-Krylov with Cubic Interpolation Linesearch satisfying Null Curvature Condition Krylov Type Conjugate Gradients

| iter | value | gnorm |
|---|---|---|
| 0 | 5.171112e-03 | 7.337197e-03 |
| 1 | 1.160506e-10 | 1.245979e-06 |
| 2 | 1.453148e-17 | 4.667996e-10 |

**Figure 1-8.** – Example of *ROL_Messages.txt* file for Inverse Problem Solution

## *1.8. Example Inverse Problems*

Inverse problems are class of problems where some portion of the solution to an analysis is known, but the inputs to the problem are not. Inverse methods solve an optimization problem where the inputs are optimized to match the solution. The current types of input problems supported are Load ID (transient or FRF) and Material ID (Eigen and FRF).

## *1.9. Experimental Data*

For inverse problems, experimental data is typically gathered in a lab. In acoustics, microphones are used to measure acoustic pressure For the transient case, these are measured over a period of time. For the FRF case, these are measured over a series of frequencies. Introducing additional measurements at new data points generally improves the fidelity of the computed solution. On the other hand, the computational difficulty of solving the inverse problem increases too. For this demonstration, synthetic data is generated by solving a forward acoustic problem.

## *1.10. Inverse Problems - Load-ID*

### 1.10.1. Experimental Model

The experimental model is shown in Figure 1-9. The *Football Model* is an ellipsoidal acoustic mesh, with a cylindrical hole in the middle. 70 sidesets are placed around the exterior of the football, allowing for different loading on each sideset.

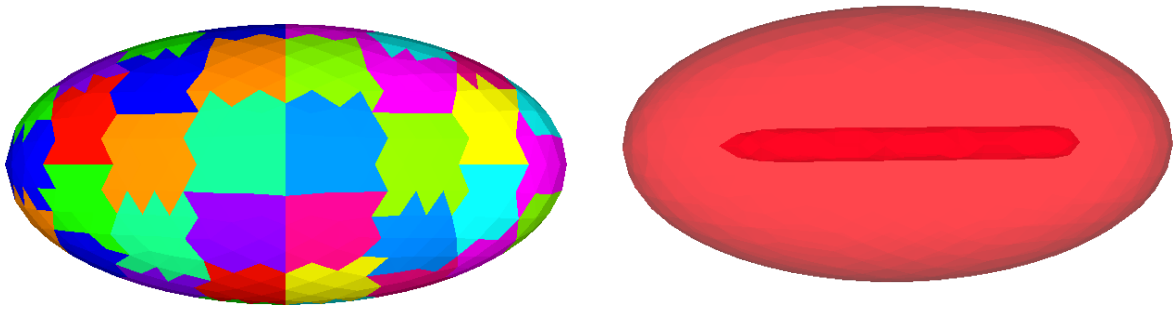

**Figure 1-9.** – Inverse Football Problem Geometry. On the left, the sideset definitions on the surface. On the right, the interior of the problem.

### 1.10.2. Forward Problem

To generate the experimental data, the model is solved with the solution method `direct-frf`. A set of human generated loads is used to generate “experimental" pressures at a select set of nodes. Any set of loads can be used. The Matlab function `inputDataProcDynFreqAcoustic_Exo.m` is used to generate the experimental data files: `ttable.txt`, `dataReal.txt`, and `dataImag.txt`. This Matlab function requires the results from the forward run, and a nodeset containing the nodes of interest. In practice, this data is generated experimentally, with the measured acoustic pressures being inserted in dataReal.txt and dataImag.txt.

```
solution
    directfrf
end

loads
    sideset 4
    acoustic_accel = 1.0
    scale = 2
    function = 10001
    inverse_load_type = spatially_constant \\ ignored
end
```

Inverse keywords are ignored when running a forward problem.

### 1.10.3. Inverse Problem with known loads

Next, the experimental model is solved with solution method `directfrf-inverse`. and `design_variable = load`. For the first run of the inverse problem, the synthetic loads were left in place, as the “initial guess". The inverse problem converges on the first iteration, as the initial guess is the exact solution to the inverse problem. This is an easy way to make sure the input file are correct. The relative tolerance is shown in the first column of ROL_Messages.txt, and the absolute tolerance is shown in the second column of

ROL_Messages.txt. If the exact loading is used as the initial guess, the relative error norm should be on the order of machine precision.

```
solution
   directfrf-inverse
end

optimization
   check_grad = no
   optimization_package = ROL_lib
   LSstep = Newton-krylov // recommended
   LS_curvature_condition = null
   max_iter_Krylov = 50 // tolerance on gradient
   opt_tolerance = 1e-8 // of objective function
   // with respect to parameters
   objective_tolerance = 1e-4 // tolerance on
   // objective function value
   opt_iterations = 50 // before stopping
end

inverse-problem
   design_variable = load
   data_truth_table = ttable.txt
   real_data_file = dataReal.txt
   imaginary_data_file = dataImag.txt
end
```

#### 1.10.4. Inverse Problem with unknown loads

Next, the synthetic loads are removed, and the initial guess for the loading is set to be 0 at all time steps. The inverse problem converged in four iterations, with an objective tolerance of $10^{-4}$. The objective norm is a relative measure, and any objective norm of $10^{-6}$ or smaller is considered more than sufficient. Alternatively **opt_tolerance** can be used to set the absolute tolerance. Recommended values for **opt_tolerance** are problem dependent.

#### 1.10.5. Verification

Finally, the loading output from the inverse run, `force_function_data.txt`, is used to run the forward problem again. This file is designed so that it can replace the function file with no changes. The problem is verified by checking the pressures at the selected nodes against the initial run. Though the loading may not be exactly the same between the initial forward run and the verification forward run, the inverse problem has been solved successfully, as the objective function has been solved to the selected tolerance. To generate loading closer to the initial loading, more nodal data can be added or tolerances can be tightened.

## *1.11. Inverse Problems - Material-ID*

### 1.11.1. Experimental Model

The experimental model is a solid assembly of two steel blocks joined by a region of viscoelastic foam material. Figure 1-10 shows the geometry of the test model.

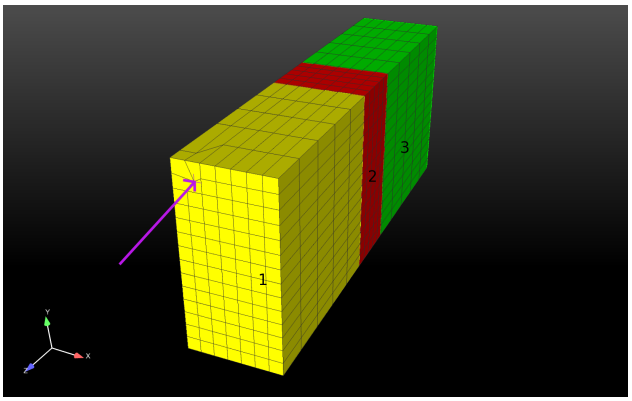


**Figure 1-10.** – Foam block model with finite element mesh and force location

As shown in Figure 1-10, the model assembly consists of two equally-sized steel blocks, depicted in yellow and green, joined by a region of viscoelastic foam material, shown in red. The model was discretized with a finite element mesh of Hex-8 elements. A periodic point load with a frequency of 500 Hz was applied to the yellow block, also as shown in the figure. It was desired to calculate the frequency-dependent viscoelastic material properties of the foam block, including complex values for the bulk ($K$) and shear ($G$) moduli.

### 1.11.2. Inverse Problem input format

The relevant sections of the input used for this example are shown below, followed by some notes about each section.

```
solution
  directfrf-inverse
end
inverse-problem
  design_variable = material
  data_truth_table = ttable.txt
  real_data_file =  data.txt
  imaginary_data_file =  data_im.txt
end
optimization
  optimization_package = ROL_lib
  ROLmethod = trustregion
  TRstep = secant
```

```
  opt_tolerance = 1e-13
  opt_iterations = 100
end

...

block 1
  inverse_material_type = homogeneous
  material 4
  hex8f
end

...

material 4
  isotropic_viscoelastic_complex
  Greal_bounds  -1000 100000
  Kreal_bounds  -1000 100000
  Gim_bounds -1000000 1000000
  Kim_bounds -1000000 1000000

  Greal = function 2
  Gim   = function 3
  Kreal = function 4
  Kim   = function 5
  density=0.010804
end
```

- **solution section:** defines the type of solution (inverse DirectFRF).
- **inverse-problem section:** specifies the design variable (material) and connects externally defined data describing the test.
  - The **data truth table file** contains the global node numbers where the experimental data measurements are given. For example, the input below gives the number of nodal locations (1) in the first line, followed by the single node id (212) showing all structural dof active (1 1 1), and an inactive acoustic dof (assumed 5th column = 0).

    ```
    1
    212    1 1 1
    ```

  - The `real_data_file` and `imaginary_data_file` contain the real and imaginary parts of the measurement data at each frequency. For example, the input below (from a real data file) gives the number of nodes (3) and frequencies (2), followed by the data at each node. Each frequency requires a separate column of data.

```
3  2
-4.385640897908e-02     -3.985611576838e-02
2.898761003889e-02     3.478175302036e-03
-1.167004279970e-01     -8.319040683879e-02
```

- **optimization section:** provides options to control the optimization strategy. See the users manual for more information on available optimization options.
- **block section:** provides an information on the material of the block. Options include,

  **known** - The material parameters of the block will not be varied in the inverse solution.

  **homogeneous** - Material properties are uniform within the block, and are varied to arrive at the best fit for the data.

  **heterogeneous** - Material properties vary element by element within the block, and are varied to arrive at the best fit.
- **material section:** provides additional options to control the optimization strategy.

### 1.11.3. Running the Inverse Problem

Next, the experimental model is solved as indicated above. A good choice for the first run of the inverse problem is to use the actual material data used as the "initial guess". This causes our problem to converge much faster than with a general guess, and is a good verification step for problems where the material data is known *a priori*. Next, initial guesses for the material data is set to be something other than the actual value to represent a typical initial guess. Convergence data can be found in the file `ROL_Messages.txt`. The objective norm is a relative measure, and any objective norm of $10^{-6}$ or smaller is sufficient. Alternatively **opt_tolerance** can be used to set the absolute tolerance.

### 1.11.4. Verification

For the problem presented here, the following material data is obtained from running the inverse problem (taken from the `name_0.rslt` file):

```
...

Block 1 Viscoelastic Material Properties
  Real Part of K:    40000.002012
  Imaginary Part of K:    -0.005544
  Real Part of G:    15999.999313
  Imaginary Part of G:    5000.000812
```

```
...
```

This gives us values that we can then use as part of an input for a forward problem, and see if we obtain the same values given in the input data from the truth table and data files. Though the displacements may not be exactly the same between the initial forward run used to generate the inverse data files and the verification forward run, the inverse problem has been solved successfully, as the objective function has been solved to the selected tolerance. To generate more exact results, more nodal data can be added or tolerances can be tightened.

# 2. Inverse Methods with InverseAria

## *2.1. Introduction*

Inverse Aria enables solving inverse problems with SIERRA/Aria using adjoint-based gradients through an interface to the Rapid Optimization Library (ROL). The major advantage of calculating gradients with adjoints comes in the form of computational savings as the design space grows in size. Only one forward solve and one adjoint solve are required to compute the gradient of the reduced objective function with respect to the design variable vector regardless of the number of design variables. Contrast this with finite difference gradients where N+1 forward solves are required for N design variables.

## *2.2. Outline*

- Overview of inverse heat transfer problems
- Inverse capabilities
- How to build and run
- Inverse problems with example inputs
  - Thermal conductivity inversion
  - Boundary heat flux inversion
  - Arrhenius source term inversion (with FD gradients)

### 2.2.1. Beta Capabilities and Limitations

Inverse Aria is a still in early development and should be treated as a beta feature. There are three over-arching problem types that Inverse Aria targets with each in various states of development. The three classes of target problems are material property, boundary condition, and heat source inversion. For the first two years of development (FY20 and FY21), Inverse Aria has been limited to solving linear conduction problems. Typically, thermal engineering problems of interest contain non-linear effects such as temperature dependent material properties, chemical decomposition, and thermal radiation among others. Beginning in FY22, development will shift from adding new inverse design variables to focusing on supporting non-linear heat conduction physics.

Thermal conductivity was the first target material property to be developed. Currently, thermal conductivity inversion is possible in Inverse Aria for basic (non-porous) materials with a limited number of boundary conditions. The user can invert for conductivity on a block by block or element by element basis (see Sec. 2.4).

Heat flux to a surface was the first target boundary condition inversion problem. Both steady and transient heat flux inversion are available in Inverse Aria as described in Sectioins 2.5 and 2.6.

Many real problems involve at least one reacting material. Examples include pyrolysis of organic materials, ablation of thermal protection systems, and thermal runaway of batteries. These reactions can add or remove heat from the system and are represented in Aria as volumetric heat source/sink terms. Typically, these reactions can be model by Arrhenius forms. To achieve wider ranging applicability, Inverse Aria must be able to solve problems with reacting source terms, either inverting directly for the source term model parameters or inverting for other parameters (material or boundary conditions) in problems where reacting materials are present. As a first step towards enabling this functionality, inversion for the Arrhenius activation energy and frequency factor with finite-difference gradients has been enabled in Inverse Aria. Development of this feature serves to put the building blocks in place for solving this class for problems while further research is conducted on deriving and implementing the adjoint solve.

Computation of the objective function requires experimental data and will typically benefit from some form of regularization. Inclusion of experimental data is currently limited to time histories of temperatures at user specified node locations.

A finial limitation is that problem size is currently constrained by available memory. To execute an adjoint solve, all state variables at every node and time step must be saved in memory. This constraint will be alleviated in future releases by using checkpointing schemes such as Wang et al. [14].

### 2.2.2. Getting Started with Inverse Aria

SIERRA developer access is required to build Inverse Aria, and these permissions can be acquired in WebCARS. More information on setting up the code repository can be found at Aria New Developer Resources.

To build Inverse Aria, first load the developer module.

```
$ module load sierra-devel
```

Then compile with the following command (e.g. for the gcc release build)

```
$ bake InverseAria -e release
```

Inverse Aria can then be run with

```
$ <path-to-bin>/inverse_aria -i <input_file> -opt <ROL_inputs>
```

where the path to the code bin is the location of the Inverse Aria executable, the input file is the target Aria input deck, and ROL inputs is an xml file containing inputs to ROL. Among the inputs contained in the xml file is the location of the experimental data file, optimization algorithm inputs, and termination criteria. (need documentation for list of ROL inputs).

Several commonalities exist in the input file requirements for the different inverse problems that use adjoint based gradients. We note that the Arrhenius reaction parameter inversion capability does not yet use adjoint based gradients, so the following input requirements do not apply.

First, an adjoint source term must always be present on all blocks.

```
Source for energy on all_blocks = Optimization value = 0
```

The "Optimization" keyword must also be specified on all initial conditions as follows

```
IC for Temperature on all_blocks = Optimization value = 300
```

A subset of boundary conditions are supported by Inverse Aria. These boundary conditions are no flux (default), Dirichlet, flux, and generalized natural convection. Usage of boundary conditions outside of these will result in unexpected behavior in the adjoint solve. The optimization keyword is not required for specifying boundary conditions.

## *2.3. Inverse Problems*

The chapter covers the current capabilities of Inverse Aria with examples of the required inputs. The problems covered are thermal conductivity inversion, steady boundary heat flux inversion, and Arrhenius source term inversion. The input decks and meshes for the examples can be found at docs/InverseOpt/InverseAria /Examples.

### 2.4. Thermal Conductivity

An example use case for thermal conductivity inversion can arise in situations where two materials are joined together by some sort of adhesive and the quality of the bond is unknown. A simplified example is shown in Figure 2-11, where two cylinders are joined by a thin material of unknown thermal conductivity. The domain is modeled as 2D axi-symmetric with a heat flux applied to the outside of the outer cylinder and temperature measurements are only available at this outer surface. In this example, synthetic data at each surface node were generated from an Aria simulation with specified thermal conductivities in the joining layer.

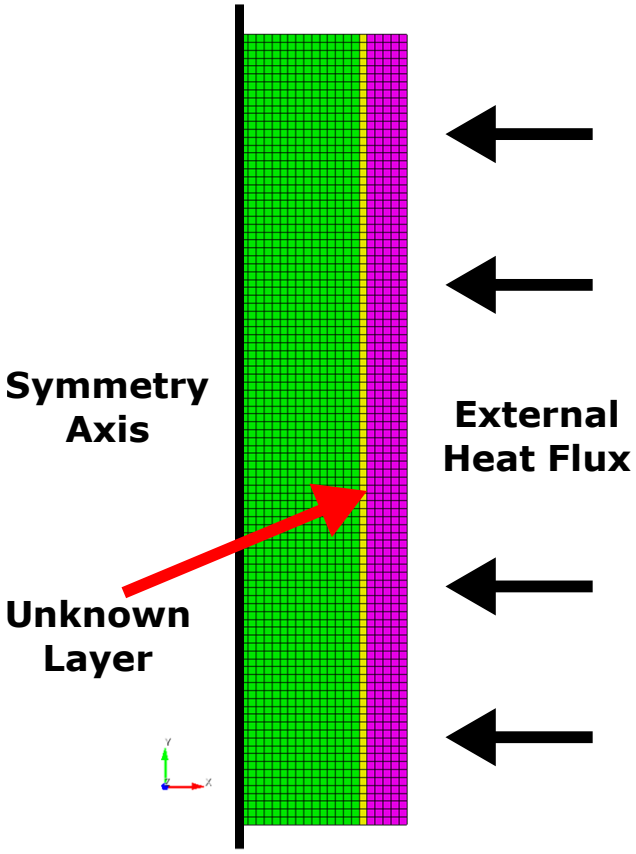


**Figure 2-11.** – Domain of the example thermal conductivity inverse problem.

There are two options in the material specification when inverting for thermal conductivity. Choosing homogeneous will result in inverting for a single thermal conductivity value for the entire material, whereas using heterogeneous will invert for a thermal conductivity value for every element with that material. the syntax for a homogeneous thermal conductivity inversion problem is

```
Begin Aria Material MatA
    Density              = Constant rho = 2702.
    Specific Heat        = Constant cp  = 903.
    Thermal Conductivity = Optimization type = homogeneous k = 300
```

```
    Heat Conduction      = Basic
End   Aria Material MatA
```

and a heterogeneous problem is defined by

```
Begin Aria Material MatA
    Density              = Constant rho = 2702.
    Specific Heat        = Constant cp  = 903.
    Thermal Conductivity = Optimization type = heterogeneous k = 300
    Heat Conduction      = Basic
End   Aria Material MatA
```

The user has the option to provide lower and upper bounds for the design variable on the thermal conductivity line with the keywords `lower_bound` and `upper_bound` and values separated by equals signs.

A simple example can be found at
Examples/Thermal_Conductivity/Homogeneous_1_Block. The key files are:

- `layered_2D_inv.i` - Inverse Aria input deck
- `inverseInput.xml` - ROL input parameters
- `layered_2D_data.txt` - Synthetic temperature data
- `layered_2D.i` - Aria input deck for generating synthetic data
- `layered_2D.jou/g` - Mesh journal file and genesis file

In this example, a 500 kW/m$^2$ heat flux is applied to the outside of the cylinder, and the thermal conductivity of the joining layer is treated as homogeneous. The inverse solution is found quickly for this simple problem as shown by the objective function and gradient norm in Figure 2-12.

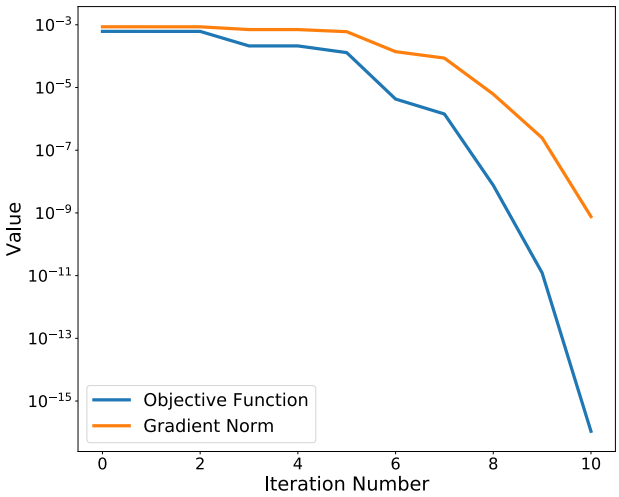


**Figure 2-12.** – Objective function and gradient norm at each iteration of the optimizer.

## *2.5. Steady Boundary Heat Flux*

To invert for a steady heat flux on a surface, the "Optimization_Design" keyword is used.

```
BC Flux for Energy on surface_2 = Optimization_Design value = -1e6
```

The example problem is composed of three materials with different material properties and a steady flux on the right half of the top surface as shown in Figure 2-13 (left). Material A is a conductive material, and materials B and C represent different internal layers with varying material properties. In real experiments, data may only be available on one surface of the test article, and for this example synthetic data are generated at the nodes along the bottom surface.

For the inverse problem we imagine a scenario where the heat flux along the entire top surface is unknown, and we search for the fluxes on the left and right halves of the top surface (i.e. two design variables). In the example input deck (Examples/Heat_Flux/Steady/layered_2D_inv.i), we set an initial guess of 5 kW/m$^2$ on both halves and the optimizer quickly finds the solution (Fig. 2-13 right) of 30 kW/m$^2$ on the top right half of the domain. The values of the heat fluxes at the top surface can be viewed in the heartbeat file: `layered_2D_flux_inv.txt`.

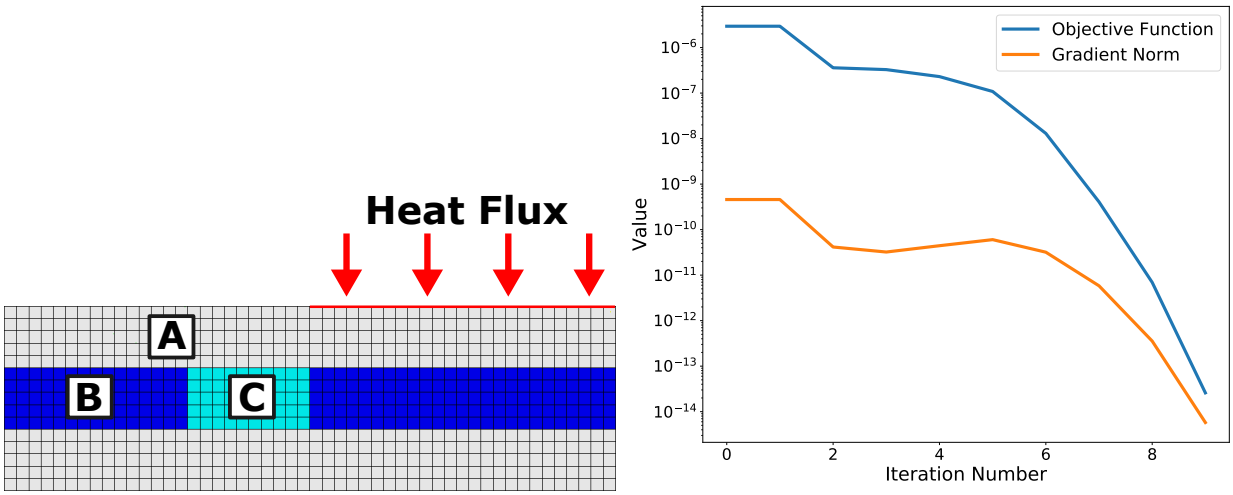


**Figure 2-13.** – Domain of the example heat flux inverse problem (left) and residuals for the inverse problem (right).

## *2.6. Transient Boundary Heat Flux*

Reconstructing an unknown transient heat flux can be accomplished in a variety of ways that all require the specification of a functional dependence of the heat flux on time. In the present formulation, the heat flux is modeled as a piecewise linear function between a set of user-specified times. Two options are available for specifying these times: constant time intervals and arbitrary time points.

To use constant time intervals, the "num_times" keyword is added to the boundary condition specification.

```
BC Flux for Energy on surface_2 = Optimization_Design \$
    value = -1e6 num_times = 2
```

In this example input line, the total simulation time is divided in to two equal intervals with three design variables (unknown heat fluxes) located at time 0, one half the simulation time, and the final simulation time. For example, if the total simulation time is 300 seconds and the user selects "num_times" = 2, the transient heat flux will be reconstructed at 0 s, 150 s, and 300 s with piecewise linear functions between each of these points. The initial guess for the optimization problem is a constant heat flux specified with the "value" keyword.

If the user desires more control over the specific time points and initial guess for the heat flux, they can use a tabular user function. This input syntax will be familiar to current Aria users, as it is commonly used to specify time varying boundary conditions or temperature dependent material properties. The syntax for the boundary condition line is as follows

```
BC Flux for Energy on surface_2 = Optimization_Design_User_Function \$
    NAME = input_flux X = time
```

This requires the specification of a user function in the Sierra domain of the input file. The name of this function must match the name in the boundary condition line ("input_flux" in this case).

```
BEGIN DEFINITION FOR FUNCTION input_flux
    type is piecewise linear
    begin values
        # t(s),     W/m2
        0           -2.0e4
        200         -1.0e5
        300         -5.0e4
    end values
END DEFINITION FOR FUNCTION
```

In the function above, the optimizer will invert for the heat flux at times 0 s, 200 s, and 300 s using the values provided in the table as the initial guess.

An example transient heat flux inverse problem was created based on the steady heat flux simulation in Section 2.5. This example uses the same geometry, heat flux location, and temperature measurement locations on the bottom of the domain as the steady heat flux example. Synthetic temperature data is generated using a heat flux profile that begins at 50 kW/m$^2$ and remains constant for 60 seconds (Fig. 2-14 left). The heat flux then decreases linearly to 0 kW/m$^2$ over the next 40 seconds and remains at zero until the end of the simulation time.

The time interval for the inverse solution of the heat flux was set to 20 seconds, and the initial guess was set at 10 kW/m$^2$ for 100 seconds decreasing to 0 kW/m$^2$ at 120 seconds as seen in Figure 2-14 left. It is important to note that the time intervals are informed by the heat conduction time between the heated surface and the measurement locations. This heat conduction time is modeled as $t = \delta^2/\alpha$, where $\delta$ is the distance between the heat must travel and $\alpha$ is the thermal diffusivity of the material. For the thermally "closest" thermocouple to the surface in this geometry, the heat conduction time is approximately 9 seconds. This is effectively a lower limit on the fidelity of the transient reconstruction as there is insufficient information to resolve smaller times.

Figure 2-14 right shows the objective function and norm of the gradient versus optimization iteration. Note that after a large initial decrease in the objective function and gradient, several iterations are required to reach the "true" synthetic solution due to the loss of thermal information as heat diffuses from the heated surface to the temperature measurement locations.

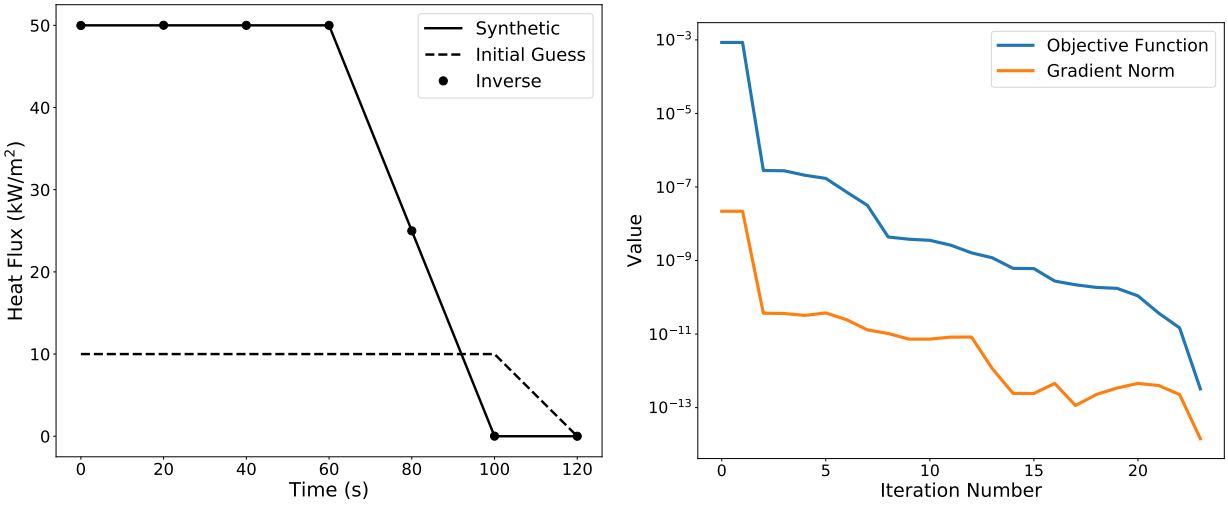


**Figure 2-14.** – Transient heat flux with inverse solution (left) and residuals for the inverse problem (right).

## 2.7. *Arrhenius Source Terms with Finite Differences*

The reaction source term inversion feature differs from the other inverse problems in that it uses finite differences and not adjoint solves to compute gradients of the objective function. This requires less modification of the input file for inverse problems, but the inverse solution suffers from the lack of adjoint-based gradients. This feature has been primarily implemented as a stepping-stone towards adjoint solves with reaction source terms in Inverse Aria.

The Arrhenius form of a reaction is given as

$$k = Ae^{-E_a/RT} \tag{2.1}$$

where the rate constant ($k$) is a function of the pre-exponential factor ($A$), activation energy ($E_a$), ideal gas constant ($R$), and temperature ($T$). Typically, calibration of reaction models involves searching for an $A$ and $E_a$ that fit experimental calorimetry data. Other parameters such as the heat of reaction and concentration function are also candidates for inversion.

Inverse Aria currently supports inversion for $A$ and $E_a$ for an arbitrary number of reactions using the following syntax in the General Chemistry block.

```
Begin Optimization Reaction_Name
    Optimize = A log_transform = true
    Optimize = Ea lower_bound = 1 upper_bound = 1e10
End
```

In this example syntax, `Reaction_Name` must match the name of the reaction block with unknown parameters. Initial guesses are set in the reaction block and the rate function must be Arrhenius. The user can specify $A$ and/or $E_a$ as design variables with the `Optimize =` command. Providing lower and upper bounds on the potential values for the design variable are optional. Log transforming the design variable before sending it to ROL is also optional, with the default being false.

Unlike problems that use adjoint-based gradients, solving a problem with finite differences does not require modification of the boundary condition specification or the addition of an adjoint source term in the input file. Only the optimization block must be provided by the user in the General Chemistry block of the Aria input file. Additionally, the ROL input file must have the following line to enable finite difference gradients

```
<Parameter name="Use FD Gradient" type="bool" value="true" />
```

The user is directed to the regression test suite (InverseAria_rtest) for examples with one and two reactions (inverseProblems/one_rxn_one_param and inverseProblems/two_rxn_two_param).

# 3. Optimal Experimental Design

## 3.1. Input Deck

The input deck is an XML formated file *.xml*. Default input decks are provided with the user's manual. The input deck is used to specify the optimality criteria, the transfer matrix, and the parameters of the optimization algorithm. This section explains the input deck parameters required to execute an optimal experiment design (OED) solve.

The *xml* file opens with an $< ParameterList\ name = Inputs >$ parameter list with a corresponding $< /ParameterList >$ to close out the parameter list at the end of the file. Between this parameter list the user provides the parameters and options needed to execute the OED.

```
ParameterList name="Inputs"
  User provided parameters
ParameterList
```

## 3.2. OED

In the OED parameter list, the user provides the optimality criterion they wish to minimize and optional parameters specific to each optimality type. For basic usage, the user only needs to provide the *Optimality Type*. All other parameters are advanced options.

```
Input Deck: OED
ParameterList name="OED"
  Parameter name="Optimality Type" type="string" value="D"
ParameterList
```

### 3.2.1. Optimality Type

The solution of the OED depends on the optimality criterion selected. This is specified with the *Optimality Type* parameter. The user can choose between A, C, D, I and R. D-optimality is disabled for multi-observation cases (*Number of Models* is greater than one). Table 3-19 provides a brief description for each criterion. Note that $\mathbf{C} \in \Re^{mxm}$ is the covariance matrix of the design parameters. The theory documentation describes each optimality criterion in greater detail.

**Table 3-19.** – Optimality Criteria

| Criterion | $\Psi(C(p))$ | Description |
|---|---|---|
| **A** | $Tr(C)$ | Average estimation variance ( MSE) |
| **C** | $v^T C v$ | Variance of $v \in \Re^m$ times the estimator |
| **D** | $det(C)$ | Volume of the uncertainty ellipsoid |
| **I** | $\mathbf{E}[h^T C h]$ | Average prediction variance ( MSPE) |
| **R** | $AVaR_\beta[h^T C h]$ | Tail average prediction variance for $\beta \in [0,\ 1]$ |

### 3.2.2. Initial Design

The user can specify the initial design weights. Typically upon first initialization the weights should be uniform, however in the case that the user wants to warm start the algorithm at a set of specified weights then there is an option to provide an initial design guess.

- **Uniform Initial Guess**: If true, then the initial weights at each sensor are equal. If false, then an initial design guess must be provided.
- **Initial Design Guess**: A user specified text file containing the initial weights on each sensor. This file will have the same format as a resulting optimal design text file. See section 3.4 for a description of the results file format. This option can be used to restart an optimization problem if the previous optimization algorithm terminated early. Note that the sum of the weights should equal one.

```
Input Deck: Initial Design
ParameterList name="OED"
  Parameter name="Uniform Initial Guess" type="bool" value="true"
  Parameter name="Initial Design Guess" type="string" value="initial.txt"
ParameterList
```

## 3.3. Linear Model

Currently, optimal experiment design assumes a linear regression model. In other words, there exists a matrix $H(q) \in \Re^{nxm}$ that maps the design parameters of interest $\theta(q) \in \Re^m$ to the measured response data $y(q) \in \Re^n$, where $m$ represents the number of design parameters and $n$ represents the number of measurement locations. We refer to the linear operator $H$ as a transfer matrix. The modal matrix used for modal expansion, the frequency response function for frequency domain control problems, or the convolution matrix for time domain control problems are a few common examples of the transfer matrix.

The simplest case is where each row of the transfer matrix corresponds to a single candidate sensor and correspondingly a sensor weight, such as the case for the modal matrix. The more complex case is when there are multiple observations of the data at a sinlge sensor location. For example, in the frequency domain the number of observations is equal to the number of frequency lines.

The transfer matrix is provided in the Linear Model parameter list. To minimize the liklihood that a transfer matrix is specified incorretly, we provide the user three options for parsing the transfer matrix based on the domain type:

- General: Any transfer matrix can be specified using the general framework
- Frequency Domain: Specific to the frequency domain where the response at each frequency line is independent
- Time Domain: Specific to the time domain where the response at each time step is dependent

The following sections describes each domain type and the associated input deck.

### 3.3.1. General Framework

The user stores the transfer matrix as a two dimensional matrix on a *.txt* file, which will then be passed to the oed executable. The full user specified transfer matrix includes all observations. In the general framework, the user provides a transfer matrix text file, a transfer matrix dimension file, and specifies the number of models. The transfer matrix is divided by its rows by the number of models where each model is a transfer matrix associated with a single observation.

```
Input Deck: General Domain
ParameterList name="Linear Model"
  Parameter name="Domain Type" type="string" value="General"
  Parameter name="Number of Models" type="int" value="2"
  Parameter name="Transfer Matrix" type="string" value="transferMatrix.txt"
  Parameter name="Transfer Matrix Dimension" type="string"
    value="tmDimFile.txt"
```

```
ParameterList
```

Further details on each parameter:

- Transfer Matrix: User provided transfer matrix text (.txt) file that contains the full transfer matrix.
- Transfer Matrix Dimension: User provided transfer matrix dimension text (.txt) file. The first and second entry are respectively the total number of rows and columns of the full transfer matrix.
- Number of Models: Total number of possible measurements at a sensor. For a frequency domain problem, this value is the total number of frequencies, and in the time domain, the total number of time steps. The user should check that the total number of rows of the transfer matrix divided by the specified Number of Models is equal to the number of possible sensors.

For example, consider a frequency domain problem with real valued (no-imaginary part) transfer matrices. Let $n_0$ be the number of frequency lines. The full transfer matrix will have $n * n_o$ rows and $m * n_o$ columns. The format of the transfer matrix is as follows:

$$\mathbf{H} = \begin{pmatrix} H(\omega_1) & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & H(\omega_2) & 0 & \cdot & \cdot & 0 \\ \cdot & 0 & \cdot & & & \cdot \\ \cdot & \cdot & & \cdot & & \cdot \\ \cdot & \cdot & & & \cdot & 0 \\ 0 & 0 & \cdot & \cdot & 0 & H(\omega_{n_o}) \end{pmatrix}$$

where $H(\omega_i)$ is the frequency response function (FRF) at frequency $\omega_i$. Again $H(\omega_i)$ has $n$ rows for the $n$ sensor locations, and $m$ columns for the number of design parameters associated with the $i^{th}$ frequency. The rows and columns of $H(\omega_i)$ must be ordered consistently for each frequency, and the number of sensors and number of parameters must be equal for all frequencies. Note that the full transfer matrix $\mathbf{H}$ is block diagonal since the measured response at $\omega_i$ is independent of parameters of $\omega_j$ for $i \neq j$. This format becomes more complex if $H(\omega_i)$ is complexed value. In that case, $H(\omega_i)$ contains both the real and imaginary parts in the following block form.

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}_R & -\mathbf{H}_I \\ \mathbf{H}_I & \mathbf{H}_R \end{pmatrix}$$

where $\mathbf{H}_R$ is the full transfer matrix containing the real part, and $\mathbf{H}_I$ contains the imaginary part.

### 3.3.2. Frequency Domain

To avoid the parsing confusion for a complex-valued frequency domain transfer matrix, we provide a simpler parsing interface where the user only needs to specify the real part of the transfer matrix and the imaginary part as two separate files in the following format:

$$\mathbf{H} = \begin{pmatrix} H(\omega_1) \\ H(\omega_2) \\ . \\ . \\ . \\ H(\omega_{n_o}) \end{pmatrix}$$

Instead of providing a transfer matrix dimension file, the dimensions of the transfer matrix are provided in the Linear Model parameter list as shown below. If there is no imaginary component then set the *Complex Valued* parameter to false.

```
Input Deck: Frequency Domain
ParameterList name="Linear Model"
  Parameter name="Domain Type" type="string" value="Frequency"
  Parameter name="Number of Parameters" type="int" value="3"
  Parameter name="Number of Outputs" type="int" value="10"
    ParameterList name="Frequency Domain"
      Parameter name="Complex Valued" type="bool" value="true"
      Parameter name="Number of Frequencies" type="int" value="2"
      Parameter name="Real Frequency Response Function"
        type="string" value="real.txt"
      Parameter name="Imaginary Frequency Response Function"
        type="string" value="imag.txt"
    ParameterList
ParameterList
```

### 3.3.3. Time Domain

The third parsing option is for time domain type problems. When the transfer matrix is a function of time, the user specifies the impulse response for each parameter-output combination. The Domain Type option should be set to Time. By doing so, the code internally converts the impulse responses into a convolution matrix which takes a Hankel matrix form. The number of models will be equal to the total number of time steps. The time domain transfer matrix takes the following form:

$$\mathbf{H} = \begin{pmatrix} H(t_1) \\ H(t_2) \\ . \\ . \\ . \\ H(t_{n_o}) \end{pmatrix}$$

where $H(t_i) \in \Re^{nxm}$ is the impulse response at the $i^{th}$ time step between the $m$ parameters and $n$ outputs (the measurement locations). For example, the first row of $H(t_i)$ is the response at $t = i$ at outputs $1...m$ due to a delta function at the first parameter.

```
Input Deck: Time Domain
ParameterList name="Linear Model"
  Parameter name="Domain Type" type="string" value="Time"
  Parameter name="Number of Models"  type="int" value="2"
  Parameter name="Transfer Matrix" type="string" value="matrix.txt"
  Parameter name="Transfer Matrix Dimension"
    type="string" value="dimFile.txt"
ParameterList
```

## *3.4. Executing OED and Results*

### 3.4.1. OED executable

To execute an OED run use "mpirun -np **numProcs** oed_inverse -opt **inputXML**.xml", where **numProcs** is the number of processors for parallel runs, and **inputXML**.xml is the xml file which contains the user specified options.

### 3.4.2. Parallel Runs

Internally, the OED algorithm splits the number of processors between algebraic operations for design variables (sensor weights) and the stochastic objective function evaluations. This split is not done automatically. The user specifies which fraction of processors are devoted to algebraic operations as follows:

```
Input Deck: Processors
ParameterList name="Problem"
  ParameterList name="Design"
    Parameter name="Number of Design Processors" type="int" value="4"
  ParameterList
ParameterList
```

For example, if *Number of Design Processors* is set to 5, and there are 60 processors, then 12 processors are devoted to the objective function and each of the 12 has 5 processors devoted to algebraic operations. We recommend that *Number of Design Processors* rougly be 10 percent of the total number of processors. However, processor decomposition which results in optimal performance will depend on the specific problem.

### 3.4.3. Results

After the optimization algorithm converges 3 (4 for I optimality) text files are outputted to the current directory. *I_optimal_design_0.txt* contains the solution to the OED on the first processor. The first column corresponds to the sensor label. This label is ordered consistently with the order in which sensors appear in the transfer matrix. Columns 2 and 3 can be ignored. Column 4 represents the probability measure of that sensor location. For a fully converged solution where the optimization constraints are met, this value is between 0 and 1. The sum of all probabilities equals 1. The probability measures can be interpretted as the precentage that the sensor should be experimentally sampled. The candidate sensors with weights that are non-zero should be selected for the experiment design.

If the oed is ran in parallel, then **numProcs** *I_optimal_design* files are printed. The algorithm now automatically concatenates these files into a file called *I_optimal_design_final*.

## 3.5. Greedy Algorithm

The default behavior of the oed executable is to call the Rapid Optimization Library's OED convex optimization algorithm. This algorithm may not be suitable if the user needs to specify a total sensor budget. In order to enforce a total sensor budget, we provided a greedy based algorithm to minimizing the optimality criterion. This is a heuristic approach which does not gurantee the global minimum solution. However, we the optimality criterion that are available are submodular which means that the quality of the solution is bounded.

The greedy algorithm is only available for D and A optimality. In order to run the greedy algorithm, the user provides the following option:

```
Input Deck: Greedy
ParameterList name="Inputs"
  Parameter name="Use Greedy Method" type="bool" value="true"
ParameterList
```

The total sensor budget is specified in the greedy parameter list as follows:

```
Input Deck: Greedy
ParameterList name="Greedy"
  Parameter name="Budget" type="int" value="5"
ParameterList
```

### 3.6. Baseline senors

For both greedy and ROL OED algorithms, a set of sensors of which must be included in the optimal set can be specified by the user. In terms of the algorithm, the transfer matrix components associated with this sensor set is always included in the covariance calculation. The algorithm optimizes only over the remaining set of sensors. To enable this feature, the xml file will need a text file specifying a list of indices corresponding to the rows of the transfer matrix that are included in the baseline. This file must begin with the total number of baseline sensors followed by the indices. These indices are written for C++ so 0 indicates the first sensor. The list is provided to the xml file via the following parameter sublist:

```
Input Deck: Baseline Sensors
ParameterList name="OED"
  Parameter name="Use Baseline Sensors" type="bool" value="true"
  Parameter name="Baseline Sensors File" type="string" value="baseline.txt"
ParameterList
```

The following example baseline sensor text file includes 3 total sensors: the fourth (3), first (0), and seventh (6) sensor.

```
Example Baseline Sensor File
 3 //Total number of sensors
 3
 0
 6
```

### 3.7. Tutorial Scripts

Within the user guide sub-directory is the folder *sample_executable* which contains a number of example input decks. Once the transfer matrix is built using the provided matlab script, the oed is ready to run using the provided XML files.

# BIBLIOGRAPHY


[1] Martin P Bendsøe. "Optimal shape design as a material distribution problem". In: *Structural optimization* 1.4 (1989), pp. 193–202 (cit. on p. 38).

[2] D. P. Bertsekas. "Projected Newton Methods for Optimization Problems with Simple Constraints". In: *SIAM J. Control and Optimization* 20 (1982), pp. 221–246 (cit. on p. 23).

[3] Gregory Bunting et al. "Novel Strategies for Modal-Based Structural Material Identification". In: *Journal of Mechanical Systems and Signal Processing* 149.107295 (2021) (cit. on p. 7).

[4] J. V. Burke, J. J. Moré, and G. Toraldo. "Convergence properties of trust region methods for linear and convex constraints". In: *Math. Programming* 47 (1990), pp. 305–336 (cit. on p. 23).

[5] P. H. Calamai and J. J. Moré. "Projected gradient methods for linearly constrained problems". In: *Math. Programming* 39 (1987), pp. 93–116 (cit. on p. 23).

[6] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Nonlinear Equations and Unconstrained Optimization*. Philadelphia: SIAM, 1996 (cit. on pp. 21, 22).

[7] M. Heroux et al. "An overview of the Trilinos project". In: *ACM Trans. on Math. Software* 31.3 (2005), pp. 397–423. ISSN: 0098-3500 (cit. on pp. 20, 43).

[8] C. T. Kelley and E. W. Sachs. "A Trust Region Method for Parabolic Boundary Control Problems". In: *SIAM J. Optimization* 9 (1999), pp. 1082–1099 (cit. on p. 23).

[9] D. P. Kouri et al. *Rapid Optimization Library*. 2014. URL: https://trilinos.org/packages/rol (cit. on p. 20).

[10] C.-J. Lin and J. J. Moré. "Newton's method for large bound-constrained optimization problems". In: *SIAM J. Optim.* 9.4 (1999), pp. 1100–1127 (cit. on p. 23).

[11] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2006 (cit. on pp. 21, 22).

[12] S D Team. *SD – User's Manual*. Tech. rep. SAND2021-12052. PO Box 5800, Albuquerque, NM 87185-5800: Sandia National Laboratories, 2021 (cit. on p. 1).

[13] S D Team. *Sierra Structural Dynamics Verification*. Tech. rep. SAND2021-11330. Sandia National Laboratories, 2021 (cit. on p. 15).

[14] Qiqi Wang, Parviz Moin, and Gianluca Iaccarino. "Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation". In: *SIAM Journal on Scientific Computing* 31.4 (2009), pp. 2549–2567 (cit. on p. 51).

# INDEX

This page intentionally left blank.

## DISTRIBUTION

**Hardcopy—Internal**

| Number of Copies | Name | Org. | Mailstop |
|---|---|---|---|
| 1 | T. F. Walsh | 1543 | 0897 |

**Email—Internal** [redacted]

| Name | Org. | Sandia Email Address |
|---|---|---|
| Technical Library | 1911 | sanddocs@sandia.gov |

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.