8 Plato Input Deck

8.1 Syntax .............................................. 29
8.2 Objectives .......................................... 30
  8.2.1 Type ........................................... 31
  8.2.2 Code ........................................... 31
  8.2.3 Weight ......................................... 31
  8.2.4 Name ........................................... 31
  8.2.5 Stress Limit .................................... 31
  8.2.6 Mass to Stress Constraint Ratio ................. 32
  8.2.7 Initial Mass Weight Factor ....................... 32
  8.2.8 Control Stagnation Tolerance ..................... 32
  8.2.9 Penalty Param Expansion Factor ................. 32
  8.2.10 Boundary Conditions ......................... 33
  8.2.11 Loads .......................................... 33
  8.2.12 Load Case Weights ............................. 33
  8.2.13 Output For Plotting ......................... 33
  8.2.14 Number of Processors ......................... 33
  8.2.15 Multi-Load Case .............................. 34
  8.2.16 Distribute Objective .......................... 34
  8.2.17 Analysis solver tolerance ...................... 36
8.3 Constraints ........................................ 36
  8.3.1 Type ........................................... 36
  8.3.2 Volume Fraction ................................ 36
  8.3.3 Volume Absolute ................................ 36
8.4 Boundary Conditions .............................. 37
8.5 Loads .............................................. 37
  8.5.1 Traction ....................................... 38
  8.5.2 Pressure ....................................... 38
  8.5.3 Acceleration .................................. 38
  8.5.4 Force ......................................... 38
8.6 Optimization Parameters ......................... 38
  8.6.1 Optimization Algorithm ......................... 39
  8.6.2 Number Processors ............................. 39
  8.6.3 Discretization Method ......................... 40
  8.6.4 Filter Radius Scale ........................... 40
  8.6.5 Filter Radius Absolute ......................... 40
  8.6.6 Output Frequency .............................. 40
8.6.7  Fixed Blocks .................................................. 40
8.6.8  Initial Density Value ......................................... 41
8.6.9  Max Iterations ................................................ 41
8.6.10 Restart Iteration ............................................. 41
8.6.11 Initial Guess Filename ...................................... 41
8.6.12 Initial Guess Field Name .................................. 41
8.6.13 Prune Mesh .................................................. 42
8.6.14 Number Buffer Layers ...................................... 42
8.6.15 Number Prune and Refine Processors ..................... 42
8.6.16 Number Refines ............................................... 42
8.6.17 Kelley-Sachs Outer Stationarity Tolerance ............... 43
8.6.18 Kelley-Sachs Outer Stagnation Tolerance .................. 43
8.6.19 Kelley-Sachs Outer Actual Reduction Tolerance ........ 43
8.6.20 Kelley-Sachs Max Trust Region Iterations ............... 43
8.6.21 Problem Update Frequency .................................. 43
8.6.22 GCMMA Max Inner Iterations .............................. 44
8.6.23 GCMMA Inner KKT Tolerance ............................... 44
8.6.24 GCMMA Inner Control Stagnation Tolerance ............. 44
8.6.25 GCMMA Outer KKT Tolerance ............................... 44
8.6.26 GCMMA Outer Control Stagnation Tolerance ............. 44
8.6.27 GCMMA Outer Objective Stagnation Tolerance .......... 44
8.6.28 GCMMA Outer Stationarity Tolerance ..................... 44
8.6.29 GCMMA initial moving asymptotes scale factor ......... 44
8.6.30 Output Method ............................................... 44
8.7  Mesh ............................................................. 45
8.7.1  Name ......................................................... 45
8.8  Materials ......................................................... 45
8.8.1  Penalty Exponent ........................................... 45
8.8.2  Youngs Modulus ............................................. 46
8.8.3  Poissons Ratio .............................................. 46
8.8.4  Density ........................................................ 46
8.9  Blocks .......................................................... 46
8.9.1  Material ....................................................... 47
8.9.2  Element Type ................................................ 47
8.10 Code Paths ....................................................... 47
8.10.1 Sierra-SD .................................................... 48
8.10.2 PlatoMain ..................................................... 48
8.10.3 Prune and Refine ............................................ 48
8.11 Uncertainties .................................................... 48
8.11.1 Type .......................................................... 49
8.11.2 Axis .......................................................... 49
1 Release Notes

Plato 2.1 includes a new stress constrained mass minimization objective. **THIS NEW FEATURE IS A BETA CAPABILITY MEANING YOU SHOULD USE IT AT YOUR OWN RISK.** This feature will allow the user to specify a stress constraint above which she does not want the vonmises stress to exceed at any point in the model. Plato will attempt to remove as much mass as possible while still maintaining the stress constraint. This capability is being release as a beta feature because it is still partially under development. A production-ready capability is expected to be released in the next Plato release–Fall of 2019.
2 Introduction

Plato is a design environment that uses Topology Optimization (TO) to create “Generative Designs”—designs that are optimized to meet user-specified functional requirements. It does this by iteratively running Finite Element Analysis to simulate the required physics and then optimally placing material to meet the desired objective. The objective could be things like maximizing stiffness, maximizing heat flow, or minimizing stress. The user specifies the “function” and Plato determines the “form”. Plato includes a powerful graphical user-interface that allows the user to set up the problem and then actively monitor and guide the evolving design.

Plato leverages existing tools where possible. It is built on the Sandia Analysis Workbench (SAW)—a plug-in environment for setting up and running finite element analysis problems. This environment provides many of the capabilities required for running topology optimization. Plato also leverages the CUBIT geometry and meshing toolkit. The CUBIT functionality is included as a component in the Plato environment. See https://cubit.sandia.gov for more information about CUBIT.
3 Installation and Running Plato

These instructions are specific to Sandia users. If you are not a Sandia user and need help configuring your installation of Plato please contact the Plato team at plato3D-help@sandia.gov.

3.1 Running on the Sandia CEE LAN

Plato is installed on the Sandia SRN CEE LAN. Simply type “/projects/plato/plato” to launch Plato on this LAN. The default installation of Plato uses Sierra/SD as its finite element engine so you will need to have permission to run the Sierra physics codes. If you do not already have access to Sierra physics codes see the section below on signing up for Sierra access. To launch topology optimization runs on the HPC platforms you will need an HPC account. If you do not already have an HPC account see the section below on obtaining an HPC account.

3.1.1 Signing Up For Sierra Access

To sign up for access to run Sierra physics codes go into Webcars and sign up for “SIERRA Analysts Code Access”. This will probably require manager approval and may take overnight to get into the system.

3.1.2 Obtaining an HPC Account

To sign up for an HPC account first go into Webcars and sign up for “SRN Capacity Clusters” or “SCN Capacity Clusters” depending on your needs. This will probably require manager approval and may take overnight to get into the system. Next, go to wc-tool.sandia.gov/wctool and submit a request for a WCID. You will use your WCID when launching topology optimization runs on the HPC platforms. If necessary, work with your manager to obtain the correct input for your WCID application.

3.2 Running Your Own Local Installation of Plato at Sandia

If you are not running on a LAN where Plato is already installed you will need to download your own version of Plato and install it on your local machine. To download Plato go to www.sandia.gov/plato3d and click on the ”Downloads” link. Once you have downloaded the install package follow the instructions below for your platform.
3.2.1 Windows

If you downloaded the .zip file unzip it (it is recommended that you choose an area to unzip the file with no spaces in the path and in a location with a fairly short path name). If you downloaded the self-extracting .exe file launch it to unzip the package. After you unzip the package the contents of the top level PLATO directory will look something like that shown below. To launch Plato double-click on the “PLATO.bat” file (it might just show up as “PLATO”). Note that there is a “PLATO.exe” executable down in the “data” directory but launching this file will not work correctly.

![Windows Unzipped Directory]

3.2.2 Mac

After downloading the Mac install package unzip the file if necessary. After you unzip the file you should be able to launch it directly to start PLATO.

3.2.3 Linux

After downloading the Linux install package unzip the file if necessary. After you unzip the file the contents of the top level PLATO directory will look something like that shown below. To launch Plato double-click on the “PLATO.sh” file. Note that there is a “PLATO” executable down in the “data” directory but launching this file will not work correctly. If prompted as to whether you want to run PLATO or display its contents choose “Run”.

![Linux Unzipped Directory]
3.3 Installing and Running Outside of Sandia

If you plan to load both Plato and Sierra on a local machine please see the instructions in Local Installations. If you plan to load Sierra on a remote cluster please contact the Plato team at plato3D-help@sandia.gov.
4 Tutorial Problems

Various tutorials can be found at the tutorials section of the Plato website. Start with the one called "Getting Started". You can also try running some canned example problems as described in the next section.
5 Canned Example Problems

Plato has some example problems completely setup that you can load and run. This section will describe the process for loading these examples.

1. Create a new model by choosing “File->New->Model” from the menus.

2. Choose “Tutorial Examples” for the category and click “Next”.

![Screenshot of choosing new model category](image)
3. Choose one of the tutorial example models and click “Next”.
Accept the default model name and location and click “Finish”.

At this point Plato will create a new model containing the example problem and you can modify the default parameters or choose your own and run the optimization as usual. Example results for the Crank, Hanger, and RoundTable tutorial examples respectively are shown below for reference.

Crank, Hanger, and RoundTable tutorial example results.
6 The Overall Process

Designing with topology optimization is fundamentally different from traditional CAD design. In traditional design an engineer or designer will typically mock up a design in a CAD system using his or her knowledge and experience related to the design space. The result is a CAD model specifying the “form” of the design with the hope that it will meet the “function” required of the design. The function is then often assessed using experimentation or computational analysis. When designing with topology optimization this traditional design paradigm is inverted. Instead of defining the form of the design the engineer or designer specifies the required function and lets the topology optimization come up with the form that will meet that function based on computational analysis. This section describes the main steps in designing with topology optimization using Plato.

6.1 Problem Setup

Problem setup consists of defining the design envelope or allowable space in which the optimized design can reside, the functional requirements that need to be met by the design, and the desired objective to optimize.

6.1.1 Design Envelope Definition

The design domain or envelope is the allowable space that the optimized design can occupy. This could be as simple as a cubic bounding box or much more complicated if the component being designed is part of a complex assembly of components. Based on your requirements you will need to define what this allowable space looks like and generate a solid model to represent it. This can be done within Plato using the solid modeling capabilities there or in another CAD package if it is easier. If you will be generating this solid model external to Plato you should save it in the STEP format so that it can be imported into Plato. If you will be generating the solid model for the design domain using the solid modeling capabilities in Plato please see the CUBIT user manual [1] for complete documentation on these capabilities.

6.1.2 Mesh Generation

The input for the topology optimization consists of a finite element mesh of the design domain defined in 6.1.1 and an input deck with parameters defining the functional requirements of the design and other parameters to guide the optimization. Therefore, the next step is to generate a finite element mesh of the design domain solid model. A mesh can either be created outside of Plato and then imported into Plato or it can be generated in Plato using the CUBIT mesh generation capabilities. For detailed documentation on using the CUBIT mesh generation capabilities incorporated in Plato please see the CUBIT user manual [1] and CUBIT tutorials [2].
6.1.3 Boundary Conditions and Loading

Design functional requirements are communicated to Plato via boundary conditions and loads. For example, in the lantern bracket problem in the “Getting Started” tutorial (see tutorials) we need to communicate to Plato that the bracket had to be bolted to a wall and support the load of a hanging lantern. To do this we specify a fixed boundary condition where the design would be attached to the wall and applied a vertical load where the lantern would hang.

6.1.4 Units and Magnitudes

It is up to the user to maintain consistent units when setting up a problem. For example, material properties and loads should have consistent units. For compliance minimization (stiffness maximization) the position and direction of a load are critical for determining the “stiffest” configuration of material—not the magnitude of the load. However, if there are multiple loads the relative magnitudes of the loads are important. Even though absolute magnitudes don’t play a role in determining optimally “stiff” structures it is important to get rough magnitudes of loads correct with regards to units so that the problem is well-posed. This will help the stability of the optimization algorithm.

6.1.5 Output

Plato can generate output containing variables for the design topology, displacements at the nodes, and VonMises stresses on the elements. These will show up in the output exodus result files as variable names “OptimalTopology”, "Disp", and "VonMises" respectively. The OptimalTopology field is a nodal field which contains the values of the design variables in the range of [0, 1] with 0 signifying void material and 1 for solid material. Upon completion, isosurfaces are extracted using the OptimalTopology field at a value of 0.5 (currently not modifiable by user) at the number of iterations requested by the user. The OptimalTopology field must be present for extraction of the isosurfaces. VonMises is an element field containing the values of the Von Mises stress for each element. The Disp field outputs displacements and rotations in the x, y and z directions.

6.2 Topology Optimization Setup

The user sets up objectives and constraints that define the functional requirements of the design. For example, the user may choose to define an objective to have the design be as stiff as possible with a volume fraction constraint. The user also has some control over the optimization algorithm through parameters in the input deck. For example, the user can specify the maximum number of iterations and the smoothing filter radius. The various topology optimization parameters are described in detail in section 8.
6.3  Job Submission

Once all of the input for the topology optimization problem is defined the job is submitted. The job can be submitted either to run on the local machine or on a remote cluster or HPC platform. Plato provides a very nice interface for specifying where and how the job will be run. The user can specify the machine, number of processors, requested time, etc. and then launch the job with a single click. Valuable information such as current machine load is shown graphically in the “Machines” view and helps in determining the best place to run the job. Multiple jobs can be launched and running simultaneously in the Plato environment.

6.3.1  Auto Mesh Prune and Refine

Plato provides a capability called mesh pruning and refining that is used to help reduce computational cost while providing higher resolution of design features. This feature allows the user to specify how many levels of mesh refinement he wants and how much of the “unused” mesh from a previous run he wants to prune away. Thus, the pruning aspect of this feature only makes sense when doing a restart from a previous iteration where a design exists. Example results and intermediate pruned and refined meshes are shown in the diagram below. Note that this prune feature is almost always employed to produce smooth designs ready to directly print on an additive manufacturing (AM) machine.
The mesh pruning and refining happens at the beginning of a restart of a prior optimiza-
tion, before the next topology optimization process is launched. Because pruning relies on a previous result a job submission using pruning must be launched as a restart job. To do this you must specify a “restart iteration” in the input deck/topology optimization parameters, indicating which iteration of the prior optimization to use as the restart.

Because the volume of the starting mesh will change when using a pruned mesh the target “volume fraction” parameter in the input deck would also need to be updated to reflect the fact that the input mesh was pruned. Instead of calculating an updated “volume fraction” value you can simply use the “volume absolute” input deck parameter instead. “volume absolute” is the absolute target volume that you want the resulting optimized design to have. Using this parameter avoids the need to constantly be updating the target volume fraction. The absolute target volume can be calculated by multiplying the volume of your original design domain by “volume fraction”. If you don’t know the original volume, use CUBIT to calculate this, using the “list volume” command.

The prune and refine parameters are in the “optimization parameters” section of the input deck. “prune mesh true” specifies that pruning should take place. The “number buffer layers” parameter is only applicable when pruning and specifies how many layers of elements outside of the elements intersected by the design will be retained in the pruned mesh. The “restart iteration” parameter you specify in the input deck will tell Plato which design iteration from the previous run should be used to guide the pruning. All elements that either lie in the design or intersect the boundary of the design will be kept by default and then the user-specified number of buffer layers outside of these elements will also be retained. “Number refines” specifies how many levels of global refinement should be applied to the mesh (pruned or not) before launching the topology optimization run. Refinement is independent of pruning and thus you can specify to simply globally refine the input mesh without doing any pruning. Furthermore, refinement is independent of restarting and can be specified regardless of whether or not the run is a restart run.

6.4 Job Monitoring

Plato provides some nice tools for monitoring the topology optimization process as it progresses. For jobs being run on queued systems the “Job Status” view lets the user know when the job gets through the queue and actually starts running. This view can also be used to kill running jobs. Once a job starts running the user will see snapshots of the evolving design in the graphics window. The user can change how often these results are shown depending on how large the problem is and how long each iteration takes. Each of these snapshots represent an actual design iteration that can be viewed and manipulated within Plato and then exported for 3D printing if desired. At any time the user can also plot the optimization objective value vs. iteration to see how well the process is converging to a solution.
6.5 Export for Manufacture

At any point during or after the optimization run the user can export any of the design iterations that were generated to an STL file for 3D printing.
7 Graphical User Interface

This section will describe the different aspects of Plato’s graphical user interface.

7.1 Main Layout

Plato has a full-featured graphical user interface (GUI) to provide a powerful, user-friendly environment for doing topology optimization-based design. The image above depicts a typical Plato layout with many of the GUI features present. In the sections that follow the various features of the Plato GUI will be discussed in more detail.

7.2 Menus

As with most modern GUls Plato has a set of top level menus for accessing many of the administrative tasks related to managing projects, models, preferences, etc.

7.3 Views

Most of the features in the Plato GUI are in the form of a separate window or view. Each of the views can be dragged outside of the main Plato application or “undocked” as a
separate window. To re-dock a window back into the Plato application simply click on the tab just under the window title bar and drag that tab back into Plato. To open views that have been closed or did not show up in the default layout when Plato started you can go to “Window-&gt;Show View” in the main menus and choose the view you want to show. In the sections that follow, most of the commonly used views will be shown undocked and each will be described in more detail. Other views are accessible by going to “Window-&gt;Show View”.

7.3.1 Graphics Window View

The graphics window is where you do all of the graphical visualization. This is where you see the domain envelope, any meshes you generate, design results coming back from the
optimization, etc. Like most CAD systems you can pan, zoom, and rotate the view in the graphics window by dragging the mouse while holding down different mouse buttons. The graphics window is the same one used in the CUBIT mesh generation software package and has many powerful capabilities for interacting with the various models you will generate. For more details about these capabilities see the CUBIT user manual [1].

7.3.2 Model Navigator View

The model navigator view is where all of your models are displayed. Each model is represented as a tree and you can explore the different parts of the model by expanding its various branches. Each model has a “Geometry/Mesh” node that contains the domain envelope solid model, domain envelope mesh, and optimized design results. Each model also has a node containing all of the finite element analysis data and topology optimization data. This node is called “plato”. After a topology optimization job has been submitted a “Simulation Job” node in the tree will appear which contains the contents of the local and remote directories where the optimization was run.
7.3.3 Console View

The console view allows for text input and output in various modes. One function of the console view is to show text output from running processes. For example, when a job is submitted the console window displays text output about what is happening at each stage of the job submission. Monitoring this output is a good way to make sure things are progressing as expected or to identify errors or problems during the process. Another purpose of the console view is to provide a place to enter commands when using the CUBIT geometry and meshing component in Plato. These different functions or modes of the console view are managed by actually having multiple consoles. There is one console for displaying output and a different console for entering CUBIT commands. You can toggle through the different consoles at any time by clicking on the icon that looks like a computer monitor in the toolbar in the top right portion of the console view.
7.3.4 Settings View

The settings view is a general purpose view where many different kinds of data are displayed and edited depending on what is selected elsewhere in the Plato GUI. One of the most common uses of the settings view is to view and edit the data associated with whatever is selected in the model navigator view. For example, in the image above topology optimization parameters are being displayed. This is a result of the user selecting the “Topology-Optimization” node in a model in the model navigator view. Often, parameters and data in the settings view can be edited by simply clicking on it and entering new values. Look to the setting view for various kinds of information. If the settings view ever seems to “disappear” make sure that it isn’t just behind another tab in the window where it is docked.
7.3.5 CUBIT Command Panel View

The CUBIT command panel view is where you can find all of the GUI panels related to performing geometry and meshing commands using the CUBIT component in Plato. For
help on learning more about the CUBIT command panel see the CUBIT tutorial [2] on the CUBIT GUI.

7.3.6 Input Deck Editor View

Plato contains a very nice input deck editor. The input deck is the file that controls the simulation. This input deck editor view is where the user can view and edit the input deck parameters defined in the model navigator tree. This editor is syntax aware so that it can flag errors in the input deck and provide context sensitive suggestions. The input deck editor is completely coupled with the model navigator and settings views so that any edits made to parameters in the input deck editor are immediately reflected in the model navigator and settings and graphics views and vice versa.
7.3.7 Machines View

The machines view displays information about the available HPC machines that you can submit jobs to. It displays the current load on the machine and provides functionality for estimating how long it will take your job to get through the queue on a given machine. This information is very valuable in determining where to run jobs to minimize wait times.

7.3.8 Job Status View

The job status view displays information about all of the jobs that have previously run or are currently running. This view provides one way for you to monitor the status or your currently running jobs. You can click on jobs to control them.
8 Plato Input Deck

The Plato input deck defines an optimization problem which can be solved on a collection of codes. The input deck provides a unifying abstraction from the input formats of the underlying codes. The deck structure is divided into blocks (Objectives, Constraints, Loads, etc.) with keywords and sub-blocks within each block.

8.1 Syntax

We will document how variables can be set in the Plato input deck to specify a design problem to be solved. Variables will have formats like: “weight {value}”. A “value” is a
floating point number, and for example, you could specify this weight as “weight 4.20”. The format “number processors {integer}” needs an integer such as “number processors 9”. The format “type {string}” needs a string such as “type volume”. The format “prune mesh {Boolean}” needs a Boolean value such as “prune mesh true” or “prune mesh false”. The format “load ids {integer} {...}” needs a sequence of at least one integer. The “{...}” specifies a repeating of the last argument type an arbitrary number of times. So to specify “load ids {integer} {...}”, one could provide “load ids 1 5 2 7”. Similarly to specify “load case weights {value} {...}”, one might set “load case weights 0.1 0.2 0.3 0.4”. Figure 1 shows a simple Plato input deck example.

Comments within an input deck can be specified as lines beginning with the “//”. For that line, all symbols after the “//” will be ignored by the input deck parsing.

8.2 Objectives

Objectives are used to define what you want the optimizer to do. Each Plato run must have at least one objective specified. The optimizer will attempt to improve the optimality of the objective(s) while also trying to honor any specified constraints.

Each Objective block begins and ends with the tokens “begin objective” and “end objective”. Within an Objective block, tokens can be specified in any order. The following is an Objective block definition example:

```
begin objective
  type maximize stiffness
  load ids 1 2
  load case weights .5 .5
  boundary condition ids 256
  code sierra_sd
  number processors 90
  weight 1
  multi load case true
end objective
```

Multiple objectives can be specified for a given Plato run. If multiple objectives are specified, Plato will attempt to improve the optimality of all objectives. At each iteration each objective is evaluated and in a multi-objective run the objective values are aggregated into a single value. At each iteration the sensitivity of the objective with respect to the design variables is also calculated so that adjustments to the design can be made prior to the next iteration. See the “weight” section below, section 8.2.3, to see how objectives can be prioritized in a multi-objective run.
8.2.1 Type

Each objective has a type in the format: “type {string}”. An example objective is “maximize stiffness”. Objectives that are available to you will depend on the physics modules you have access to. Table 1 shows the currently supported objective types and the associated codes.

Table 1: Supported objective types and associated physics modules.

<table>
<thead>
<tr>
<th>Objective Type</th>
<th>Physics Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximize stiffness</td>
<td>Sierra-SD</td>
</tr>
<tr>
<td>stress constrained mass minimization</td>
<td>Sierra-SD</td>
</tr>
</tbody>
</table>

When using the stress constrained mass minimization objective you need to use the KSBC optimization algorithm (see section 8.6.1).

8.2.2 Code

The code which will perform this objective calculation is specified in the format: “code {string}”. Although numerous code (i.e. physics) and objectives are supported in beta versions of Plato, these have yet to be elevated to the released version. There is only one analysis code available in this release: Sierra-SD.

8.2.3 Weight

Each objective has a weight associated with it in the format: “weight {value}”. At each iteration the calculated objective value is multiplied by its weight. In a problem with multiple objectives the weights are used to give more emphasis or priority to one objective over another. The “weight” parameter can be any positive value and is defaulted to 1.0.

8.2.4 Name

A name can be assigned to this objective in the format: “name {string}”. This name will be used for files and identifiers for this objective. Each objective name should be unique to that objective. If a name is not assigned, an arbitrary name will be assigned by Plato.

8.2.5 Stress Limit

⚠️ BETA FEATURE!

When using the stress constrained mass minimization objective you specify the vonmises
stress limit as follows: “stress limit \{value\}”. Plato will attempt to make sure that the vonmises stress stays below this limit at every point in the design.

8.2.6 Mass to Stress Constraint Ratio

⚠️ BETA FEATURE!
When using the stress constrained mass minimization objective you can specify this ratio value to try to control how large the mass minimization portion of the objective is compared to the stress constraint portion of the objective. Plato will typically set this parameter automatically but the user is allowed to override it. The syntax is: “scmm mass to stress constraint ratio \{value\}”.

8.2.7 Initial Mass Weight Factor

⚠️ BETA FEATURE!
When using the stress constrained mass minimization objective you can specify how much Plato will try to remove mass in the beginning. This parameter is defaulted to 0 which means that Plato will only focus on reducing stress in the design in the beginning. If you know that your initial density values will result in the whole design having stress less than the stress constraint you will need to set this parameter to something greater than 0 (0.01, for example) so that the objective gradients won’t be 0. The syntax is: “scmm initial mass weight factor \{value\}”.

8.2.8 Control Stagnation Tolerance

⚠️ BETA FEATURE!
When using the stress constrained mass minimization objective you can specify the control (density values) stagnation tolerance which will determine how often the stress constrained mass minimization algorithm will update its penalty parameters on the mass and stress constraint portions of the objective. The larger the tolerance the more often the algorithm will update its parameters. A larger tolerance will often lead to a more aggressive approach at meeting the stress constraint. The syntax is: “scmm control stagnation tolerance \{value\}”.

8.2.9 Penalty Param Expansion Factor

⚠️ BETA FEATURE!
When using the stress constrained mass minimization objective you can specify how aggressively you want the penalty parameter on the stress constraint portion of the objective to grow. The syntax is: “scmm penalty param expansion factor \{value\}”.

32
8.2.10  Boundary Conditions

A sequence of boundary conditions can be assigned to this objective in the format: “boundary condition ids \{integer\} {...}”. If the boundary condition keyword is included, at least one boundary condition is expected. The boundary conditions identifier numbers are specified in the Boundary Conditions block of the input deck.

8.2.11  Loads

A sequence of loading conditions can be assigned to this objective in the format: “load ids \{integer\} {...}”. The loading identifier numbers are specified in the Loads block of the input deck.

8.2.12  Load Case Weights

A sequence of loading weights can be assigned to this objective in the format: “load case weights \{value\} {...}”. The length of these weights should match the length of the “load ids” sequence.

These weights are only used if Sierra-SD multi-load is enabled, and in that situation, these weights represent the relative importance of the associated load id in the sequence. Larger weight represents increased relative importance of the objective with the loading condition. It is typical in optimization for weightings to sum to 1.0; however, we do not strictly enforce this.

8.2.13  Output For Plotting

For this objective, specify quantities of interest for output plotting by: “output for plotting \{string\} {...}”. The strings can include: “dispx”, “dispy”, “dispz”, and “vonmises”. This output plotting is currently only implemented in the Sierra-SD code.

8.2.14  Number of Processors

The number of processors to use for the computation of this objective is specified: “number processors \{integer\}”. Processors are dedicated to specific tasks within a full Plato problem. The number of processors specified here will compute only this objective, and hence, other portions of solving the optimization problem will use other dedicated processors. This number of processors is not the total number of processors that this full Plato problem will use.
8.2.15 Multi-Load Case

Interpreting multiple load identifiers as multi-load or single-load is specified in the format: “multi load case {Boolean}”. If “multi load case true” is set, each load specified in the Loads sequence for this objective will be applied and solved individually. If “multi load case false” is set, all loads specified in the Loads sequence for this objective will be applied and solved simultaneously, as part of a single load case. Currently only Sierra-SD supports multi-load case enabled.

8.2.16 Distribute Objective

Objectives with multiple load cases can be computed sequentially on a single group of processors or in a distributed manner across several groups of processors. We will use the term performer as a group of processors within an objective on a common physics code, set of loading conditions, and boundary conditions. To specify a distributed computation mode for multiple load cases, use the format: “distribute objective {string} {...}”. There are two possible formats of strings to be used with these tokens.

The default is “distribute objective none”, and in this case, each load case will be computed sequentially on a single group of processors (a single performer).

The second format is specifying “distribute objective at most {integer} processors”, and in that case, possibly multiple groups of processors (multiple performers) will be created behind-the-scenes to efficiently compute the load cases in a distributed mode.

As an example, here are eight load cases on a single objective.

```
begin objective
    type maximize stiffness
    load ids 1 2 3 4 5 6 7 8
    boundary condition ids 1
    code sierra_sd
    number processors 5
    multi load case true
    distribute objective none
end objective
```

In this case a single performer with 5 processors will compute all eight load cases. The line “distribute objective none” is the default behavior if a distribute is not specified. We will show alternative scenarios for this eight load problem in the following.

Alternative scenario 1: if the line “distribute objective none” is replaced with “distribute
objective at most 10 processors”, then two performers are created with 5 processors each. Each performer will compute four of the total eight load cases. Two performers at 5 processors each will create exactly 10 processors for this distributed objective.

Distributed objectives provide a succinct input file representation. The scenario above using “distribute objective at most 10 processors” is equivalent to the following two objectives:

```
begin objective
  type maximize stiffness
  load ids 1 3 5 7
  boundary condition ids 1
  code sierra_sd
  number processors 5
  multi load case true
end objective
begin objective
  type maximize stiffness
  load ids 2 4 6 8
  boundary condition ids 1
  code sierra_sd
  number processors 5
  multi load case true
end objective
```

Alternative scenario 2: if the line “distribute objective none” is replaced with “distribute objective at most 22 processors”, then four performers are created with 5 processors each. Each performer will compute two of the total eight load cases. Four performers at 5 processors each will create 20 of the at most 22 processors for this distributed objective. Notice that specifying “at most” 20, 21, 22, 23, or 24 processors will produce the same processor usage outcome and only 20 processors will be used.

Alternative scenario 3: if the line “distribute objective none” is replaced with “distribute objective at most 50 processors”, then eight performers are created with 5 processors each. Each performer will compute one of the total eight load cases. Eight performers at 5 processors each will create 40 of the at most 50 processors for this distributed objective. Notice that specifying “at most” 40 or more processors will produce the same processor usage outcome and only 40 processors will be used.
8.2.17 Analysis solver tolerance

The solver tolerance parameter for the analysis code is specified in the format: “analysis solver tolerance {value}”.

8.3 Constraints

In this section, we show how to specify constraints on the optimization problems. Each Constraint block begins and ends with the tokens “begin constraint” and “end constraint”. The following is a typical Constraint block definition:

```plaintext
begin constraint
  type volume
  volume fraction 0.40
end constraint
```

Plato input decks can contain an arbitrary number of Constraint blocks, but most problems will have exactly one Constraint. The following tokens can be specified in any order within the constrains block.

8.3.1 Type

Each constraint must have a type specified in the format: “type {string}”. The only supported constraint type is volume in the released version of Plato. Other constraint types such as “max stress” and “displacement” are planned.

8.3.2 Volume Fraction

For the “volume” type constraint, a volume fraction is specified by: “volume fraction {value}”. When using this input parameter, the optimization problem is constrained to use at most the volume specified here. The input mesh represents the full design domain for the optimization problem, and this volume fraction specifies the allowable fraction (between 0.0 and 1.0) of the full design domain. Either a “volume fraction” or a “volume absolute” should be specified for a “volume” type constraint.

8.3.3 Volume Absolute

For the “volume” type constraint, an absolute volume is specified by: “volume absolute {value}”. When using this input parameter, the optimization problem is constrained to use at most the volume specified here. The input mesh represents the full design domain for the optimization problem, and typically, this absolute volume will be set to some value less than the volume of the full design domain. Either a “volume fraction” or a “volume absolute” should be specified for a “volume” type constraint. Use of “volume absolute”
is recommended for problems using prune and refine. This is because an absolute volume
does not change from pruning but a fraction does change.

8.4 Boundary Conditions

In this section, we will show the format for defining boundary conditions for the physics
problems. The Boundary Conditions block begins and ends with the tokens “begin bound-
ary conditions” and “end boundary conditions”. The following is an example of a boundary
condition block definition in the input deck:

begin boundary conditions
    fixed displacement sideset 1 bc id 256
end boundary conditions

Every Plato input deck should contain at most one Boundary Conditions block. Within
the block, each line will be devoted to a single boundary condition.

A boundary condition will be specified in following format:

- “fixed {physical_quantity} {mesh_attribute_type} {integer} bc id {integer}”, OR
- “fixed {physical_quantity} {mesh_attribute_type} {integer} {dimension_name} bc id
  {integer}”.

Where {physical_quantity} is either displacement or temperature, {mesh_attribute_type}
is either nodeset or sideset followed by the {integer} of its identifier in the mesh file,
and {dimension_name} is either x, y, or z. If a dimension name is not specified, all three
dimensions are fixed. Each boundary condition is given an identifier by the integer following
its “bc id”. Each “bc id” should match one of the numbers listed after the “boundary
condition ids” token in the Objective block. The reader is referred to section 8.2 for more
information on the Objective block and the “boundary condition ids” token.

8.5 Loads

In this section, we will show the format for specifying loading for the physics problems. The
Loads block begins and ends with the tokens “begin loads” and “end loads”. Every Plato
input deck should contain one Loads block. Within the block, each line will be devoted to
a single load. The following is an example of a Loads block definition in the input deck:

begin loads
    acceleration 0 -1 0 load id 1
    traction sideset 2 value -1 0 0 load id 2
end loads
Each single load is finished with the three tokens “load id \{integer\}”; this identifier index will be referenced in other blocks such as the Objective block. Loads can be grouped by specifying them with the same load identifier index, and in this case, all loads with the shared identifier index will be applied simultaneously. If a load is used, its “load id” should be listed after a “load ids” token in the Objective block. The reader is referred to section 8.2 for more information on the Objective block and the “load ids” token.

### 8.5.1 Traction

A traction may be applied to a sideset by the following: “traction sideset \{integer\} value \{value\} \{value\} \{value\} load id \{integer\}”. Following the “sideset” keyword, an integer specifies which sideset in the mesh file. Three real values follow the “value” keyword for the traction value in the x, y, and z directions.

### 8.5.2 Pressure

A pressure is applied to a sideset by the following: “pressure sideset \{integer\} value \{value\}\{value\} load id \{integer\}”. Following the sideset keyword, an integer specifies which sideset in the mesh file. A single real value follows the “value” keyword for the pressure value.

### 8.5.3 Acceleration

An acceleration is applied to an entire body by: “acceleration \{value\} \{value\} \{value\} load id \{integer\}”. Three real values follow the “acceleration” keyword for the accelerations in the x, y, and z directions.

### 8.5.4 Force

A force is applied by the following format: “force \{mesh_attribute_type\} \{integer\} value \{value\} \{value\} \{value\} load id \{integer\}”. The \{mesh_attribute_type\} is either sideset or nodeset, and following this keyword, an integer specifies which of this set in the mesh file. Three real values follow the “value” keyword for the force values in the x, y, and z directions.

### 8.6 Optimization Parameters

In this section, we will show the format for controlling the parameters of the optimizer. In addition to the optimizer, several other features such as the filter and pruning are present in this block. This block begins and ends with the tokens “begin optimization parameters” and “end optimization parameters”. The following is an example of an Optimization Parameters block definition in the input deck:
begin optimization parameters
  number processors 1
  filter radius scale 1.4
  max iterations 25
  output frequency 0
  algorithm oc
  discretization density
  initial density value 0.40
  fixed blocks 1
end optimization parameters

Every Plato input deck should contain exactly one Optimization Parameters block. Within this block, tokens can be specified in any order.

8.6.1 Optimization Algorithm

Specify which optimization algorithm to use in the format: “Algorithm {string}”. The available algorithms are “oc” (Optimality Criteria method), “gcmma” (Globally Convergent Method of Moving Asymptotes), and “mma” (Method of Moving Asymptotes). “ksbc” (Kelly-Sachs Trust Region Method with Bound Constraints (⚠️ BETA FEATURE!)),

For “maximize stiffness” objectives, the OC method works fairly well. For more complicated objectives currently only available in beta versions, we recommend MMA or GCMMA. OC, GCMMA and MMA require a Constraint to be defined in the input deck.

8.6.2 Number Processors

Specify the number of processors assigned to the optimizer by: “number processors {integer}”.

Processors will be dedicated to specific tasks within a full Plato problem. This number of processors is not the total number of processors that this full Plato problem will use. Other processors will compute objective and constraint information, but the number of processors specified here will advance the optimization problem. These optimization processors aggregate and modify the current solution to the optimization problem.

In general, the physics code computations take significantly more time than the optimizer; typically, the physics code computations take 85% to 98% of the overall runtime. Because of this relationship, it is recommended to place most processors on the objective calculations, and leave only 1 to 5 processors for the optimizer.
8.6.3 Discretization Method

Specify the discretization method in the syntax: “discretization {string}”. The only supported discretization method is “density” in the released version. The “levelset” discretization method is under development. Many other portions of the Plato Input deck depend upon this token, and so, switching between discretization methods is not as simple as changing this token.

8.6.4 Filter Radius Scale

Specify a scale value for the filter radius in the following format: “filter radius scale {value}”. The filter scale specifies the radius of influence that the filter will have. In this case the filter scale is a value that will be multiplied by the average edge length in the mesh to get a filter radius. The default value is 3.5. It is typical to run problems with smaller filter radii in the 1.5 to 2.0 range.

In general, the larger the filter scale the larger the feature size in the design results. Running a density discretization with a very small filter radius scale, say 0.1, creates problems with so-called “checkerboarding” and jagged designs. If your design looks very jagged, increase the filter radius. However, running density with too large of a filter radius, say 5.0, can also create problems with too much intermediate material. If your design is not connecting up in the iso-volume, decrease the filter radius; also, consider increasing the volume constraint budget.

8.6.5 Filter Radius Absolute

Specify an absolute filter radius in the following format: “filter radius absolute {value}”. This parameter specifies an absolute value for the filter radius used in kernel smoothing. This parameter is used as an alternative to the filter scale. If both a filter scale and filter absolute are specified, the absolute will be used in Plato.

8.6.6 Output Frequency

Specify frequency for outputting by: “output frequency {integer}”. Specifies how often design results will be output and displayed in Plato during the optimization. A value of 5 means that every 5 iterations of the optimization a result will be output and displayed. A value of 0 means that output and display of the results will be turned off.

8.6.7 Fixed Blocks

Specify the blocks of the geometry file that are fixed in optimization by: “fixed blocks {integer} {...}”. These blocks are fixed with respect to optimization. This is different from having fixed displacements in the physics problem. We recommend users to fix geometry
blocks that you want to be perfectly preserved in the design optimization; this often occurs around bolt holes or required components in an assembly. Do not include the fixed blocks keyword if no blocks are fixed optimization blocks.

8.6.8 Initial Density Value

For the density discretization, the initial density is specified by: “initial density value {value}”. It is suggested that this value match the “volume fraction” value from the Constraints block; this suggestion is typical because it satisfies the volume constraint. If restarting, this initial density value is not used.

8.6.9 Max Iterations

Specify the maximum iterations of the optimizer by the syntax: “max iterations {integer}”. After this number of iterations, the optimizer will always stop iterating; that is, regardless of whether the optimizer believes it is still making progress. The optimizer could stop before this specified number of iterations if other convergence criteria are met.

The number of iterations needed is dependent upon the problem. For “maximize stiffness” objectives, 25 to 50 iterations is typically sufficient. Other objectives are often more difficult for the optimizer and require 100 to 200 iterations.

8.6.10 Restart Iteration

To specify an iteration of the optimization to restart the problem from, use the format: “restart iteration {integer}”. Specifying this iteration greater than 0 will cause a restart to be attempted. The use of “restart iteration”, “initial guess filename”, and “initial guess field name” will often occur together to specify a restart.

8.6.11 Initial Guess Filename

For users doing a restart and not using our GUI, a filename to grab an initial guess from must be specified by: “initial guess filename {string}”. The use of “restart iteration”, “initial guess filename”, and “initial guess field name” will often occur together to specify a restart.

8.6.12 Initial Guess Field Name

For users doing a restart and not using our GUI, a field in the restart file to grab an initial guess from must be specified by: “initial guess field name {string}”. If your initial guess filename is a “platomain.exo” from a previous run, this field name will be “Topology” or “OptimizationDOFs” by default. The use of “restart iteration”, “initial guess filename”, and “initial guess field name” will often occur together to specify a restart.
8.6.13 Prune Mesh

To enable or disable pruning the mesh, specify in the format: “prune mesh {Boolean}”. Setting this value to “true” will require the specification of a “restart iteration”. The mesh file specified in the Mesh block is pruned by Initial Guess Field on the Initial Guess Filename. This allows a process of repeated pruning to occur where pruning occurs each time on the same original background mesh. We recommend pruning again if after a prune the design solution meets the boundary of the pruned mesh within the background mesh; in this case, the design after pruning will have stair-steps on its boundary.

If you are pruning, we recommend using “volume absolute” rather than “volume fraction”. This volume calculation occurs within the pruned mesh, and so the physical volume allowed by a volume fraction constraint will change significantly on a pruned mesh versus the original background mesh.

8.6.14 Number Buffer Layers

Specify the number of buffer layers for pruning in the format: “number buffer layers {integer}”. Buffer layers are grown in all directions from the design of a previous run. Buffer layers are counted in the mesh after refinement occurs. Typically, 2 to 4 layers are enough to allow the topology to change some throughout. Specifying these buffers layers without “prune mesh true” will not cause a prune to occur.

8.6.15 Number Prune and Refine Processors

Specify the number of processors for the prune and refine executable by: “number prune and refine processors {integer}”. Processors are dedicated to specific tasks within a full Plato problem. This number of processors is not the total number of processors that this full Plato problem will use. These processors will transfer a previous solution from one mesh to another for a restart, perform mesh pruning, and/or perform mesh refinement depending on the input deck. The prune and refine process is generally fairly quick, and a few processors, say 5, is generally sufficient.

8.6.16 Number Refines

Specify the number of refine iterations in format: “number refines {integer}”. Omitting these keywords or setting its value to 0 will produce no refinement. Typically, 1 or 2 levels of refinement is enough to produce a significantly more fine result.

Refinement is recursive. A 1 level refine will take a problem with $N$ Hex elements to $8N$ Hex elements. A 2 level refine will take a problem with $N$ Hex elements to $64N$ Hex elements.
Refinement is typically considered with pruning. When using both prune and refine, only the pruned mesh receives the refinement, and so the physics codes do not see the total problem size increase of a global refinement.

If you are using refinement, we recommend using “filter radius absolute” rather than “filter radius scale”. The physical distance of the radius will change with refinements if this radius is specified as a scale.

8.6.17 Kelley-Sachs Outer Stationarity Tolerance

⚠️ BETA FEATURE!
This advanced parameter for the Kelley-Sachs algorithm (KSBC and KSAL) specifying the convergence tolerance for the stationarity criteria is given by: “ks outer stationarity tolerance {value}”.

8.6.18 Kelley-Sachs Outer Stagnation Tolerance

⚠️ BETA FEATURE!
This advanced parameter for the Kelley-Sachs algorithm (KSBC and KSAL) specifying the convergence tolerance for the stagnation criteria is given by: “ks outer stagnation tolerance {value}”.

8.6.19 Kelley-Sachs Outer Actual Reduction Tolerance

⚠️ BETA FEATURE!
An advanced parameter for the KSAL and KSBC algorithms is given by: “ks outer actual reduction tolerance {value}”.

8.6.20 Kelley-Sachs Max Trust Region Iterations

⚠️ BETA FEATURE!
An advanced parameter for the KSAL and KSBC algorithms is given by: “ks max trust region iterations {integer}”.

8.6.21 Problem Update Frequency

⚠️ BETA FEATURE!
When using the stress constrained mass minimization objective this parameter controls how often the algorithm will attempt to update the penalty scalars on the mass minimization and stress constraint portions of the objective. The syntax is: “problem update frequency {integer}”.

43
8.6.22 **GCMMA Max Inner Iterations**
An advanced parameter for the GCMMA algorithms is given by: “gcmma max inner iterations {integer}”.

8.6.23 **GCMMA Inner KKT Tolerance**
An advanced parameter for the GCMMA algorithms is given by: “gcmma inner kkt tolerance {value}”.

8.6.24 **GCMMA Inner Control Stagnation Tolerance**
An advanced parameter for the GCMMA algorithms is given by: “gcmma inner control stagnation tolerance {value}”.

8.6.25 **GCMMA Outer KKT Tolerance**
An advanced parameter for the GCMMA algorithms is given by: “gcmma outer kkt tolerance {value}”.

8.6.26 **GCMMA Outer Control Stagnation Tolerance**
An advanced parameter for the GCMMA algorithms is given by: “gcmma outer control stagnation tolerance {value}”.

8.6.27 **GCMMA Outer Objective Stagnation Tolerance**
An advanced parameter for the GCMMA algorithms is given by: “gcmma outer objective stagnation tolerance {value}”.

8.6.28 **GCMMA Outer Stationarity Tolerance**
An advanced parameter for the GCMMA algorithms is given by: “gcmma outer stationarity tolerance {value}”.

8.6.29 **GCMMA initial moving asymptotes scale factor**
An advanced parameter for the GCMMA algorithms is given by: “gcmma initial moving asymptotes scale factor {value}”.

8.6.30 **Output Method**
An advanced parameter for selecting data output methods is given by: “output method {string}”. The default method is “epu”.

44
8.7 Mesh

In this section, we will show the format for specifying the mesh. This block begins and ends with the tokens “begin mesh” and “end mesh”. The following is an example of a Mesh block definition in the input deck:

```
begine mesh
  name my_mesh_file.gen
end mesh
```

Every Plato input deck should contain exactly one Mesh block. Within this block, tokens can be specified in any order.

8.7.1 Name

Mesh filename strings are specified in the format: “name {string}”. The string should state a relative path to the undecomposed mesh file. The mesh is must be in the Exodus format.

8.8 Materials

In this section, we will show the format for specifying materials for the physics problems. These blocks begin and end with the tokens “begin material {integer}” and “end material”. The integer following the “begin material” specifies the identification index of this material. All identifiers should be unique within an input deck. The material identifiers will be referenced in the Blocks block. The following is an example of a material definition in the input deck:

```
begin material 1
  poissons ratio .3
  youngs modulus 1e9
  density 8000
end material
```

Every Plato input deck should contain at least one Material block. Within this block, tokens can be specified in any order. In Sierra-SD, all materials are automatically treated as isotropic.

8.8.1 Penalty Exponent

For density discretizations, the solid isotropic material with penalization (SIMP) penalty parameter can be specified by the syntax: “penalty exponent {value}”. The SIMP model is a power-law relation to penalize the stiffness of elements to reflect whether the element is part of the design in a differentiable manner.
The default value is 3.0. Penalties greater than or equal to 1.0 are reasonable. A larger penalty coefficient tends to reduce the occurrence of intermediate densities (that is, push density values towards either 0.0 or 1.0). For continuations on the penalty parameter, specify a sequence of exponents; specifying a sequence rather than a single value is only supported in Sierra-SD.

8.8.2 Youngs Modulus

To specify the numerical Youngs Modulus of this material, use the format: “youngs modulus {value}”. In the Sierra-SD code, this value is the “E” parameter of this material.

8.8.3 Poissons Ratio

To specify the numerical Poissons Ratio of this material, use the format: “poissons ratio {value}”. In the Sierra-SD code, this value is the “nu” parameter of this material.

8.8.4 Density

To specify the numerical density of this material, use the format: “density {value}”. This density value is only used for the Sierra-SD code, and the value is directly passed to the “density” keyword of this material within the Sierra-SD input. When using the objective for maximizing stiffness it isn’t necessary to specify a material density, however, when using the stress constrained mass minimization objective (⚠️ BETA FEATURE!) you must specify a density in units consistent with the rest of the problem.

8.9 Blocks

In this section, we will show the format for specifying geometry blocks for the physics problems. Geometry blocks are identified portions of the mesh geometry file assigned unique identifiers. Geometry blocks begin and end with the tokens “begin block {integer}” and “end block”. The integer following the “begin block” specifies the identification index of this geometry block. All identifiers should be unique within an input deck. The geometry block identifiers reference block identifiers in the mesh file. The following is an example of a geometry block definition in the input deck:

begin block 1
  material 1
    element type Hex8
end block

Every Plato input deck should contain at least one geometry block. Within these geometry blocks, tokens can be specified in any order.
8.9.1 Material

A material is assigned to this block in the format: “material \{integer\}”. This integer identifier should match an identifier to a Materials block in the Plato Input Deck.

8.9.2 Element Type

The finite element type of the block is specified by: “element type \{string\}”. Currently, this type is ignored unless it is an “rbar” in Sierra-SD. The type “rbar” is passed through to Sierra-SD in that case. Plato supports using the Sierra-SD RBAR element type for applying a load at a single location and distributing that load through rigid beam elements to other nodes. This loading approach is often used when representing a component by its center of mass and then distributing loading on this component to its interfaces with other components. **Blocks consisting of RBAR elements must be specified as “fixed” blocks in the optimization since they don’t have material associated with them.** A tutorial example problem using RBAR elements (called “Rbar”) is included in Plato.

For cases where the element type is ignored, the physics codes decide the type of finite element to use based on their defaults for that geometry element type. For example, the Sierra-SD code will take hexahedral geometry elements as “Hex8b” finite elements.

8.10 Code Paths

In this section, we will show the format for specifying file system paths to code executables. Typically, the user will not need to specify code paths once Plato has been installed. However, if the user desires, this part of the input deck will allow him to override which code executables are used in the Plato run. This block begins and ends with the tokens “begin paths” and “end paths”. The following is an example of a Paths block definition in the input deck:

```plaintext
begin paths
    code sierra_sd my_path_to_sierra_sd
    code PlatoMain my_path_to_plato_engine
    code prune_and_refine my_path_to_prune_and_refine
end paths
```

The Paths block is optional and is typically used in advanced scenarios where custom Sierra-SD, plato engine, and prune_and_refine programs need to be used. Within this block, tokens can be specified in any order.

File system paths can be absolute paths or relative paths. Solving a problem posed in a Plato Input Deck often requires the invocation of multiple code executables with specific arguments in a specific order. Specifying these paths allows for the automatic orchestration
of these code executables.

It is not necessary to specify a path to a code executable that is not needed by the other blocks in this Plato Input Deck. For the physics code Sierra-SD the path is only needed if that code is used in an Objective block. The path to PlatoMain is always needed. The path to the Prune and Refine code is only needed if prune, refine, or restart actions were requested in the optimizer block of the input deck.

8.10.1 Sierra-SD

Specify the path for the Sierra-SD code in the format: “code sierra_sd {string}”.

8.10.2 PlatoMain

Specify the path for the PlatoMain code in the format: “code platomain {string}”.

8.10.3 Prune and Refine

Specify the path for the Prune and Refine code in the format: “code prune_and_refine {string}”.

8.11 Uncertainties

In this section, we will show the format for specifying uncertainties in the problem posed by an input deck. This block begins and ends with the tokens “begin uncertainty” and “end uncertainty”. Plato input decks can contain an arbitrary number of Uncertainty blocks, but for deterministic problems, no Uncertainty block is needed. Within this block, tokens can be specified in any order. The following is an example of an Uncertainty block definition in the input deck:

```
begin uncertainty
  type angle variation
  axis X
  load 1
  distribution beta
  mean 0
  upper bound 90
  lower bound -90
  standard deviation 45
  num samples 7
end uncertainty
```
8.11.1 Type
The type of uncertainty specified by this specific Uncertainty block is set by the format: “type {string}”. Currently, the only type of uncertainty implemented is “angle variation”. An angle variation is an uncertainty in the vector orientation of a load, and for this uncertainty, the distribution values specify rotations about an axis in degrees.

8.11.2 Axis
For uncertainties that need the specification of an axis, the axis is specified in the format: “axis {string}” where the string is either “x”, “y”, or “z”. Angle variations require an axis. This input specifies the coordinate axis for the rotation.

8.11.3 Load
For uncertainties that need the specification of a load, a load identifier is specified in the format: “load {integer}” where this integer matches the identifier from the Loads block of the input deck.

8.11.4 Distribution
The type of distribution for this uncertainty is specified by: “distribution {string}”. Currently, the only distribution implemented is “beta”. The beta distributions requires specification of “mean”, “lower”, “upper”, and “standard deviation” parameters.

8.11.5 Mean
The mean of the distribution is set by: “mean {value}”.

8.11.6 Lower Bound
The absolute lower bound of the distribution is set by: “lower bound {value}”.

8.11.7 Upper Bound
The absolute upper bound of the distribution is set by: “upper bound {value}”.

8.11.8 Standard Deviation
The standard deviation of the distribution is set by: “standard deviation {value}”.

49
8.11.9 Number of Samples

The number of samples to make of the distribution is set by: “num samples \{integer\}”. The number of samples needed to resolve the uncertainty of the distribution depends on many factors specific to each problem. However as a general rule-of-thumb, four or five samples are a good start, and ten to twenty samples often provide statistical and geometric convergence.
9 Data Representations

During the topology optimization process various models are generated and various data representations are used. This section will describe the various models and data representations that you will see as you progress through the design process in Plato. A graphical depiction of the process and models is show in Figure 2.

9.1 Design Domain Solid Model

Ultimately, the design domain needs to be represented as a finite element mesh for the topology optimization run. If such a mesh already exists it can be imported into Plato directly. Otherwise, a solid model of the design domain can be imported into Plato and then meshed using the geometry and meshing capabilities in Plato. Solid models can also be created directly in Plato if desired and then meshed. The best format for solid models being imported into Plato is the .sat ACIS format or STEP.

9.2 Design Domain Mesh

A finite element mesh of the design domain is required as input to the topology optimization run. The design domain mesh can be generated in Plato from the design domain solid model or in another package and then imported into Plato. The best format if importing the mesh is the Exodus format. It will typically have a .exo, .e, .gen, or .g file extension.

9.3 Optimized Design Results

As the optimization process advances snapshots of the evolving design will be loaded and displayed in Plato. These design iterations are in the Exodus mesh file format and are simply a set of water-tight mesh triangles representing the boundary of the optimized design. Note that this is only a triangle mesh and does not contain volume mesh elements on the interior.

9.4 Design Exported to STL Format

Any one of the design results loaded into Plato during the optimization process can be exported to an STL file for 3D printing by simply right-clicking on the design in the Model Tree and choosing “Generate STL”.

51
Figure 2: Models encountered during the topology optimization process in Plato.
10 Plato Diagnostic File

The Plato diagnostic file is a table of statistics describing the progress of the optimization algorithm at every iteration. The statistics depend on the optimizer selected by the user. The statistics are always written to a text file which users can track throughout the optimization iterations.

10.1 Optimality Criteria Algorithm

This section presents the format used to present the statistics associated with the Optimality Criteria algorithm. These statistics are collected at every iteration and saved in an output file named plato_optimality_criteria_diagnostics.txt. The following table lists the headings found in plato_optimality_criteria_diagnostics.txt.

Table 2: Headings found in plato_optimality_criteria_diagnostics.txt.

<table>
<thead>
<tr>
<th>Heading</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iter</td>
<td>Outer optimization iteration.</td>
</tr>
<tr>
<td>F-count</td>
<td>Number of function evaluations.</td>
</tr>
<tr>
<td>F(X)</td>
<td>Objective function value.</td>
</tr>
<tr>
<td>Norm(F')</td>
<td>Norm of the objective function gradient.</td>
</tr>
<tr>
<td>H_i(X)</td>
<td>Inequality constraint value, where i = 1,..., the total number of constraints.</td>
</tr>
<tr>
<td>abs(dX)</td>
<td>Control stagnation metric. The control stagnation metric measures the infinity norm of the misfit between two subsequent vectors of design variables (i.e. controls).</td>
</tr>
<tr>
<td>abs(dF)</td>
<td>Objective function stagnation metric. The objective function stagnation metric measures the misfit between two subsequent objective function evaluations.</td>
</tr>
</tbody>
</table>

10.2 Conservative Convex Separable Approximation Algorithms

This section presents the format used to present the statistics associated with the family of Conservative Convex Separable Approximation (CCSA) algorithms implemented in Plato. The family of CCSA algorithms implemented in Plato include the Method of Moving Asymptotes (MMA) and the Globally Convergent Method of Moving Asymptotes (GCMMA). The statistics associated with the CCSA algorithms are collected at every iteration and saved in an output file named plato_ccsa_algorithm_diagnostics.txt. The following table lists the headings found in plato_ccsa_algorithm_diagnostics.txt.

Table: plato: srom output file - srom moments
Table 3: Headings found in `plato_ccsa_algorithm_diagnostics.txt`.

<table>
<thead>
<tr>
<th>Heading</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iter</td>
<td>Outer optimization iteration.</td>
</tr>
<tr>
<td>F-count</td>
<td>Number of function evaluations.</td>
</tr>
<tr>
<td>F(X)</td>
<td>Objective function value.</td>
</tr>
<tr>
<td>Norm(F')</td>
<td>Norm of the objective function gradient.</td>
</tr>
<tr>
<td>H_i(X)</td>
<td>Inequality constraint value, where ( i = 1, \ldots, ) the total number of constraints.</td>
</tr>
<tr>
<td>KKT</td>
<td>Inexactness in Karush-Kuhn-Tucker (KKT) conditions. The KKT metric measures the error in the KKT conditions.</td>
</tr>
<tr>
<td>Norm(S)</td>
<td>Norm of the descent direction.</td>
</tr>
<tr>
<td>abs(dX)</td>
<td>Control stagnation metric. The control stagnation metric measures the infinity norm of the misfit between two subsequent vectors of design variables (i.e. controls).</td>
</tr>
<tr>
<td>abs(dF)</td>
<td>Objective function stagnation metric. The objective function stagnation metric measures the misfit between two subsequent objective function evaluations.</td>
</tr>
</tbody>
</table>
11 Detailed Examples

In this section, we will show examples of topology optimization problems. With each example, we hope to share suggestions about how to use Plato. Problems can take minutes to days to accurately formulate and solve; by presenting many examples, we hope to jumpstart your usage of Plato.

11.1 “ThreeRings”: Various Filter Radii

Our first detailed example shows the affects of various filter radius values. We will use the “ThreeRings” example which has files (“ThreeRings.cubitJournal.txt” and “ThreeRings_platoInput.txt”). Cubit processes this journal file to produce the hex mesh in Figure 3.

The mesh is the design domain, and within the Plato input file, we specify that at most 25% of the design domain be used. We also specify that the three cylindrical bolt holes are fixed with respect to optimization. The part experiences a force on the two adjacent bolt holes in the direction of the single bolt hole, and the single bolt hole has a fixed displacement boundary condition. We show the optimized designs for a parameter study of “filter radius scale” between 1.0 and 4.5 in Figure 4.

The filter radius is a user-specified parameter and reflects a choice of desired smoothness and minimum member size. Larger filter radii will in general produce smoother designs and larger components. In Figure 4, we see the most well connected designs are for scales $\leq 1.2$ and 2.8. For the largest filter radii considered, the design fully disconnects.

Plato uses a density method formulation, and the shown final designs are iso-volumes at 0.5 of the nodal density field. In the continuous optimization problem, connections in the design can be made with density values $< 0.5$, but in the final iso-volume, these connections are discarded.

Notice that the meshed design domain had a volume of 2.707 and the user-specified the volume budget of 25% of the design domain. This would suggest that a final design of volume of at most $0.677 = 2.707 \times 0.25$ is expected. For the optimization problem posed (minimize compliance with an upper bound volume constraint), the optimizer will use all of the upper bound volume constraint because placing density anywhere in the domain decreases the compliance. Notice that in Table 4 the final design volume is close to 0.67 for small radii, but for large radii, the final design volume is much less than 0.67. Again, this is because major portions of the design were made with topology values $< 0.5$ which were discarded in the iso-volume extraction.
Figure 4: ThreeRings example at various “filter radius scale” values.
Figure 4 | Filter Radius Scale | Final Design Volume
--- | --- | ---
(a) | 1.0 | 0.678
(b) | 1.1 | 0.677
(c) | 1.2 | 0.676
(d) | 1.3 | 0.672
(e) | 1.6 | 0.662
(f) | 1.9 | 0.651
(g) | 2.2 | 0.639
(h) | 2.5 | 0.623
(i) | 2.8 | 0.606
(j) | 3.1 | 0.589
(k) | 3.4 | 0.569
(l) | 3.7 | 0.547
(m) | 4.0 | 0.517
(n) | 4.3 | 0.490

Table 4: Volumes of the final designs.

11.2 “UncertainQuarterCylinder”: Uncertain Loading

We will quantify the benefit of designing with uncertain loading for a specific example. We will use the “UncertainQuarterCylinder” example which has files (“UncertainQuarterCylinder_cubitJournal.txt” and “UncertainQuarterCylinder_platoInput.txt”). Cubit processes this journal file to produce the hex mesh in Figure 5.

The design domain for this problem is a quarter portion of a cylindrical disc. The innermost and outermost portions of the cylinder are fixed to optimization. The innermost portion has a single bolt hole, and the outermost portion has five bolt holes with a 20 degree separation between each. The expected value of a load is applied as a traction on the innermost single bolt hole towards the middle of the five outermost bolt holes. This vector direction of the expected value of the load is at 45 degrees of a 0 to 90 degree sweep that traces the outermost cylinder. We will discuss this traction load being uncertain over various subsets of this 90 degree sweep.

Our analysis will be on a compliance minimization with upper bound volume constraint problems. We develop two hypotheses:

**H1** For a load at the expected value, we expect the design optimized for only the expected value to be more stiff than a design optimized for the distribution.
For a load sufficiently off the expected value, we expect a design optimized for the
distribution to be more stiff than the design optimized for the expected value.

Because we are solving a volume constrained optimization problem, material is placed in
locations that most decrease the compliance of the continuous optimization problem. This
creates an inherent trade-off that material is either placed to most decrease the compliance
of the expected value or placed to most decrease the compliance for the full distribution.
This trade-off is reflected in our two hypotheses.

Table 5 shows the results of running several topology optimization problems for differ-
ent loading distributions. Physical quantities of interest like displacements are computed
within a topology optimization run with a density method. However, it is more accurate
to produce a conformal mesh of the final design and compute those physical quantities of
interest on the conformal mesh. Because of our optimization problem, we are interested
in the compliance on the conformal meshes.

In our hypotheses, we used the phrases “design optimized for the expected value” and
“design optimized for the distribution”. To be clear, the first row of table 5 with standard
deviation 0.005 is a “design optimized for the expected value”; this standard deviation in
the angle is sufficiently small to approximate a delta function. The next three rows of
table 5 are “designs optimized for the distribution” with various ranges of the distribution
angle bounds.

In table 6, we are evaluating the compliance of the four designs for a load at the expected
value. We have evidence that hypothesis H1 is true by the higher compliances for strictly
positive distribution ranges versus the zero distribution range. Recall the zero distribution
range is the “design optimized for the expected value” and the strictly positive distribution
ranges are “designs optimized for the distribution”. Higher compliance values correspond
to less stiff designs.

In addition, we have found that distributions with larger ranges tend to decrease the
compliance at the expected value more. This is evidenced by the trend 1.05E-8 < 1.12E-
8 < 1.21E-8 < 2.05E-8. Specific numeric values will be problem specific, but from this
problem we finalize the claim: for a load at the expected value, the design optimized for
only the expected value can be twice as stiff as the design optimized for the distribution.

In table 7, we are evaluating the compliance of the four designs for loads off the expected
value. Specifically, we choose the off expected value point \( \mu + \sigma \) where \( \mu \) is the mean
of the distribution (45 degrees) and \( \sigma \) is the standard deviation. We have evidence that
hypothesis H2 is true by the greater than 1 compliance ratios. Recall that the “with
Expected” column of this table correspond to the “design optimized for the expected
value” and the “with Off” column of this table correspond to the “designs optimized for
the distribution”. The compliance ratios are the fractional increase in compliance for the
“expected” over “off”. Specific numeric values will be problem specific, but from this
problem we finalize the claim: for a load sufficiently off the expected value, the design
optimized for the distribution can be six times more stiff than the design optimized for

58
<table>
<thead>
<tr>
<th>Distribution</th>
<th>Surface Mesh of Design</th>
<th>Sculpt Mesh of Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper bound 45.01</td>
<td><img src="image1" alt="Surface Mesh" /></td>
<td><img src="image2" alt="Sculpt Mesh" /></td>
</tr>
<tr>
<td>lower bound 44.99</td>
<td><img src="image3" alt="Surface Mesh" /></td>
<td><img src="image4" alt="Sculpt Mesh" /></td>
</tr>
<tr>
<td>standard deviation 0.005</td>
<td><img src="image5" alt="Surface Mesh" /></td>
<td><img src="image6" alt="Sculpt Mesh" /></td>
</tr>
<tr>
<td>upper bound 56.25</td>
<td><img src="image7" alt="Surface Mesh" /></td>
<td><img src="image8" alt="Sculpt Mesh" /></td>
</tr>
<tr>
<td>lower bound 33.75</td>
<td><img src="image9" alt="Surface Mesh" /></td>
<td><img src="image10" alt="Sculpt Mesh" /></td>
</tr>
<tr>
<td>standard deviation 5.625</td>
<td><img src="image11" alt="Surface Mesh" /></td>
<td><img src="image12" alt="Sculpt Mesh" /></td>
</tr>
<tr>
<td>upper bound 67.5</td>
<td><img src="image13" alt="Surface Mesh" /></td>
<td><img src="image14" alt="Sculpt Mesh" /></td>
</tr>
<tr>
<td>lower bound 22.5</td>
<td><img src="image15" alt="Surface Mesh" /></td>
<td><img src="image16" alt="Sculpt Mesh" /></td>
</tr>
<tr>
<td>standard deviation 11.25</td>
<td><img src="image17" alt="Surface Mesh" /></td>
<td><img src="image18" alt="Sculpt Mesh" /></td>
</tr>
<tr>
<td>upper bound 90</td>
<td><img src="image19" alt="Surface Mesh" /></td>
<td><img src="image20" alt="Sculpt Mesh" /></td>
</tr>
<tr>
<td>lower bound 0</td>
<td><img src="image21" alt="Surface Mesh" /></td>
<td><img src="image22" alt="Sculpt Mesh" /></td>
</tr>
<tr>
<td>standard deviation 22.5</td>
<td><img src="image23" alt="Surface Mesh" /></td>
<td><img src="image24" alt="Sculpt Mesh" /></td>
</tr>
</tbody>
</table>

Table 5: Designs for Various Distributions.
<table>
<thead>
<tr>
<th>Distribution Range</th>
<th>Displacement X</th>
<th>Displacement Y</th>
<th>Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7.73E-09</td>
<td>7.10E-09</td>
<td>1.05E-08</td>
</tr>
<tr>
<td>22.5</td>
<td>7.95E-09</td>
<td>7.89E-09</td>
<td>1.12E-08</td>
</tr>
<tr>
<td>45</td>
<td>8.58E-09</td>
<td>8.56E-09</td>
<td>1.21E-08</td>
</tr>
<tr>
<td>90</td>
<td>1.46E-08</td>
<td>1.44E-08</td>
<td>2.05E-08</td>
</tr>
</tbody>
</table>

Table 6: Physical Quantities with load at expected value.

<table>
<thead>
<tr>
<th>$\mu + \sigma$ angle</th>
<th>Force X</th>
<th>Force Y</th>
<th>Compliance with Expected</th>
<th>Compliance with Off</th>
<th>Compliance Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>67.5</td>
<td>0.383</td>
<td>0.924</td>
<td>1.55E-07</td>
<td>2.56E-08</td>
<td>6.040</td>
</tr>
<tr>
<td>56.25</td>
<td>0.556</td>
<td>0.831</td>
<td>4.78E-08</td>
<td>1.50E-08</td>
<td>3.184</td>
</tr>
<tr>
<td>50.625</td>
<td>0.634</td>
<td>0.773</td>
<td>1.93E-08</td>
<td>1.17E-08</td>
<td>1.645</td>
</tr>
</tbody>
</table>

Table 7: Physical Quantities with loads off the expected value.

the expected value.
References

