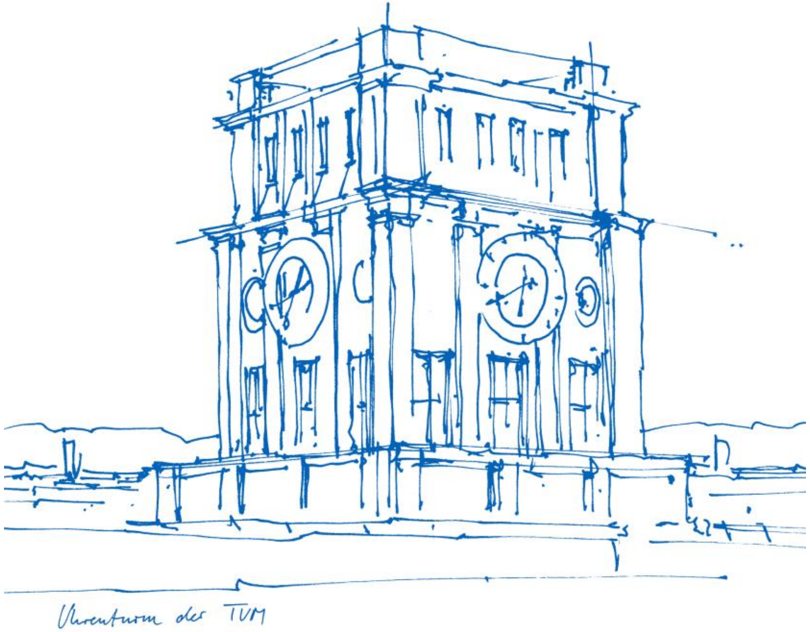
Laboratory for Product Development and Lightweight Design
TUM School of Engineering and Design
Technical University of Munich

A Tutorial on Using the SSO Toolbox

Eduardo Rodrigues Della Noce

Garching, 02.09.2024



General Content

1. Defining a System Response Function
2. Finding an Optimal Box-shaped Solution Space (and visualization – 3D plot; selective design space projection)
3. Using Surrogate Modeling (with Active Learning)
4. Stacking Evaluators (Multi-fidelity simulation, Solution-Compensation Spaces)
5. Interlude: Finding an Optimal Design
6. Interlude: Using Python Bottom-up Mappings
7. Interlude: Performing Batch Analysis
8. Interlude: Requirement Spaces
9. Using Component Solution Space Optimization – Simple Example
10. Using Component Solution Space Optimization – Advanced Example

Getting Started

Code is available on both GitLab and the OneDrive

GitLab Instructions:

- Clone the tutorial project into your PC in a folder of your choosing (Project: Tutorial SSO Toolbox)
  - `git clone https://gitlab.lrz.de/lpl-tum/sso-toolbox-lpl/tutorial-sso-toolbox.git`

OneDrive Instructions:

- Copy the tutorial folder, and create a copy of the sso-toolbox inside of that

Open this folder in MATLAB (or VScode) and run ‘setup_sso_toolbox.m’

- `run('sso-toolbox/setup_sso_toolbox')`

1. Defining a System Response Function

File: `tutorial_01_euclidean_distance_3d`

A system response function must look like this:

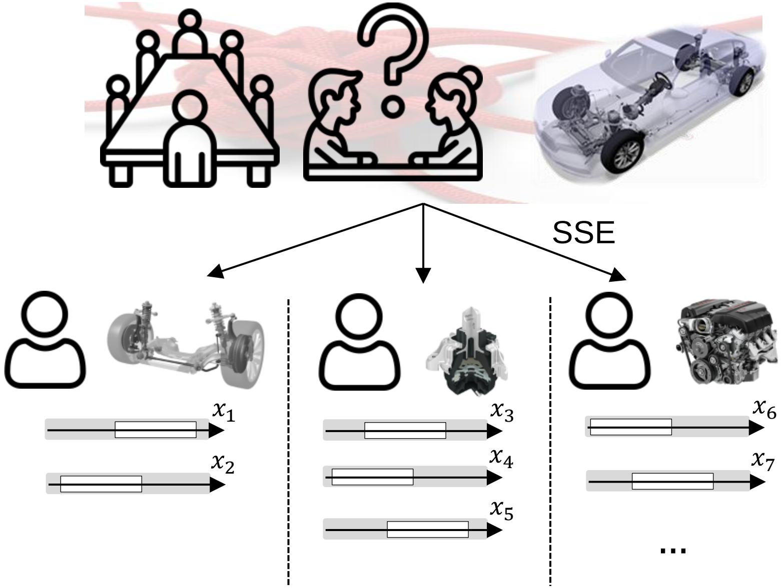
`[performanceMeasure,physicalFeasibilityMeasure,systemOutput] = f(designSample,systemParameter)`

- **`designSample`** : array with all design sample points; each column is a design variable, each row is a different point
- **`performanceMeasure`** : system response (in terms of performance); each column is a performance measure, each row is a different point
- `systemParameter` : constant system parameter (doesn't change with the samples), when applicable
- `physicalFeasibilityMeasure` : physical feasibility of the designs, when applicable
- `systemOutput` : any extra information, when applicable

Task:

Write a function that computes the distances of all given sample points to a center (constant parameter)

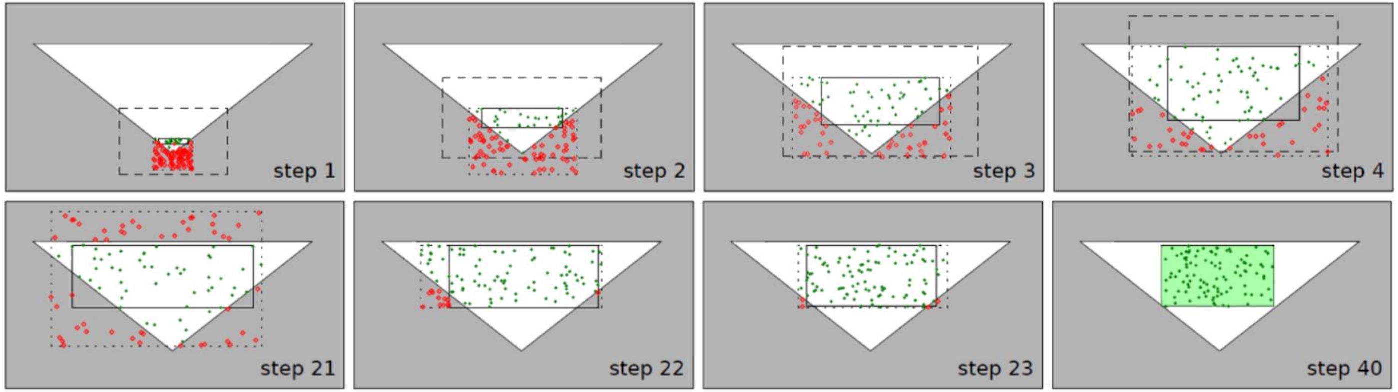
2. Finding an Optimal Box-shaped Solution Space



2. Finding an Optimal Box-shaped Solution Space

Phase I:
Exploration

Phase II:
Consolidation



Classical optimization on Ω_{ds}. Identify one good design $\mathbf{x_0}$.
The first candidate box Ω is constructed at $\mathbf{x_0}$ with zero volume.
Phase I. While $\mu(\Omega)$ is changing:
Modification Step B: Extend candidate box.
Compute Monte Carlo sample in Ω.
Modification Step A: Remove bad sample designs.
Compute Monte Carlo sample in Ω.
Phase II. While $m/N < a_c$:
Modification Step A. Remove bad sample designs.
Compute Monte Carlo sample in Ω.

2. Finding an Optimal Box-shaped Solution Space

File: `tutorial_02_sso_box_sphere`

First, the problem must be setup correctly. For that, a Design Evaluator is needed.

- First setup a Bottom-up Mapping (`BottomUpMappingFunction`).
- Then, use it to create a design evaluator (`DesignEvaluatorBottomUpMapping`).
- Finally, call the box SSO function (`sso_box_stochastic`)

Tip: for more information on functions/classes, you can always use “help <NAME>” or “doc <NAME>”

<u>Task 1:</u>

Run the box-shaped solution space optimization function and obtain the optimal solution space with the following:

- System response: `tutorial_01_euclidean_distance_3d`
- Design Space: $-6 \leq x_1, x_2, x_3 \leq 6$
- Performance Limits: $distance \leq 5$ (center: $[0,0,0]$)
- Initial Design: $[x_1, x_2, x_3] = [3,0,0]$

2. Finding an Optimal Box-shaped Solution Space

File: `tutorial_02_sso_box_sphere`

Visualization tools:

- Boxes in 2D and 3D can be easily plotted using the functions: `plot_design_box_2d`, `plot_design_box_3d`
- Selective Design Space Projection: `plot_selective_design_space_projection`

Common performance metrics can also be automatically plotted: `postprocess_sso_box_stochastic` → `plot_sso_box_stochastic_metrics`

Task 2:

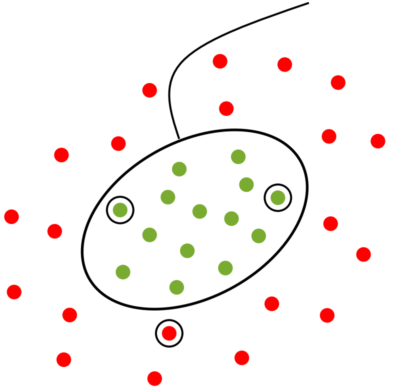
Visualize the solution space box in 3D, and then also use selective design space projection

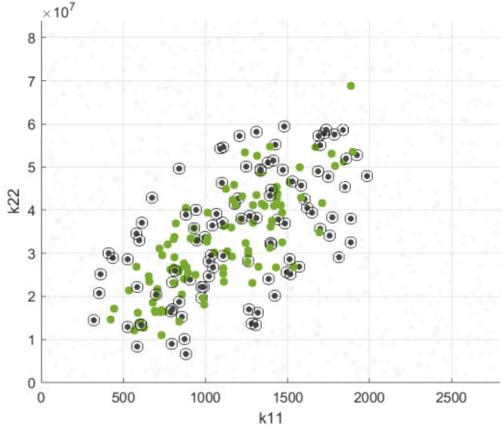
Task 3:

Plot the algorithm performance metrics for this solution

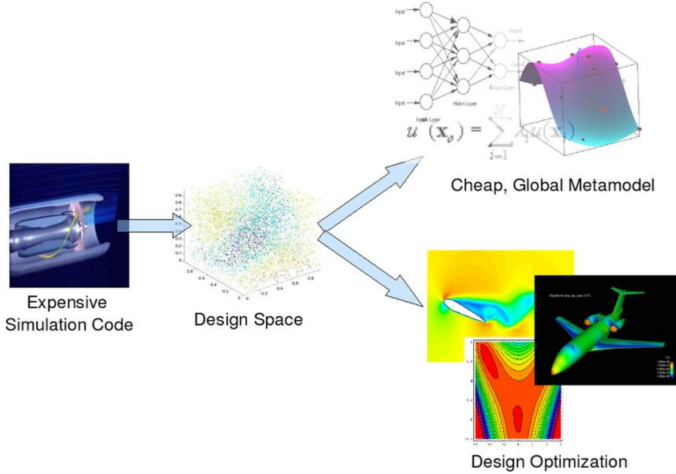
3. Using Surrogate Modeling (with Active Learning)

Decision boundary can be used to select data points during the sampling





From: Lukas Krischer, DokSem #7



Source: https://www.researchgate.net/profile/Tom-Dhaene/publication/265152921/figure/fig1/AS:335678228451328@1457043334234/Surrogate-modeling-versus-Design-Optimization.png

3. Using Surrogate Modeling (with Active Learning)

File: `tutorial_03_surrogate_modeling`

Special class of surrogate modeling for top-down systems design: `DesginFastForwardBase`
Active learning method: `active_learning_model_training`
Visualization of performance metrics: `postprocess_active_learning_model_training` → `plot_active_learning_model_training_metrics`

Task 1:

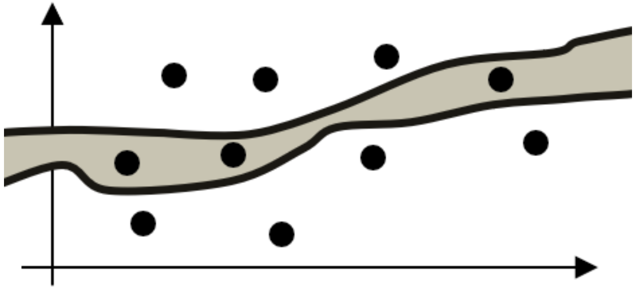
Train a surrogate model for the following problem:

- System response: `tutorial_01_euclidean_distance_3d`
- Design Space: $-6 \leq x_1, x_2, x_3 \leq 6$
- Performance Limits: $2 \leq distance \leq 5$ (center: $[0,0,0]$)

Task 2:

Visualize how the algorithm performance metrics evolved at each iteration.

4.1 Stacking Evaluators – Multifidelity Evaluation



White region: low- and high-fidelity models agree

Gray region: low- and high-fidelity models disagree

4.1 Stacking Evaluators – Multifidelity Evaluation

File: `tutorial_04_1_multifidelity`

Transforming a `DesignFastForwardBase` object to a `DesignEvaluatorBase` object: `DesignEvaluatorFastForward`
Estimating region of uncertainty: `design_fast_forward_find_uncertainty_score`

Task 1:
Train a surrogate model for the following problem:

- System response: `tutorial_01_euclidean_distance_3d`
- Design Space: $-6 \leq x_1, x_2, x_3 \leq 6$
- Performance Limits: $distance \leq 5$ (center: $[0,0,0]$)
- Maximum number of iterations: 1
- Number of samples to be evaluated per iteration: 20

Task 2:
Estimate the region of uncertainty.

4.1 Stacking Evaluators – Multifidelity Evaluation

File: `tutorial_04_1_multifidelity`

Setting up a multi-fidelity evaluator: `DesignEvaluatorMultiFidelity`

Task 3:

Solve a box SSO problem using the Multifidelity evaluator with the following additional considerations:

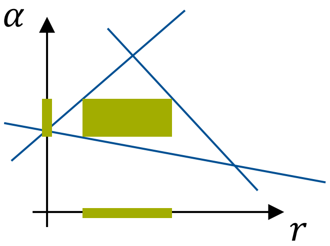
- Initial Design: $[x_1, x_2, x_3] = [3{,}0{,}0]$

4.2 Stacking Evaluators – Solution-compensation Spaces

Solution Spaces:

$$\min_{[\alpha_l,\alpha_u,r_l,r_u]} -\mu(\Omega)$$

$$\text{s.t. } g(r,\alpha) \leq 0$$



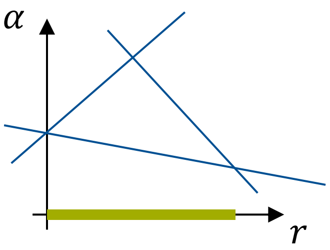
Solution Compensation Spaces:

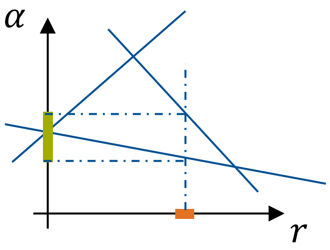
$$\min_{[r_l,r_u]} -\mu(\Omega_\mathrm{a})$$

$$\text{s.t. } \forall x_a \in \Omega_a \; \exists \; x_b \in \Omega_b$$

early decision variables (limited controllability, PFD): sensor characteristics, $x_a = r$

late decision variables (controllable): sensor positionings, $x_b = \alpha$





From: Nicola Barthelmes, DokSem #5

4.2 Stacking Evaluators – Solution-compensation Spaces

File: `tutorial_04_2_compensation`

Setting up an evaluator for solution-compensation spaces: `DesignEvaluatorCompensation`

Task:

Solve a box SCSO problem with the following considerations:

- System response: `tutorial_01_euclidean_distance_3d`
- Design Space: $-6 \leq x_1, x_2, x_3 \leq 6$
- Performance Limits: $distance \leq 5$ (center: $[0,0,0]$)
- Initial Design: $[x_1, x_2, x_3] = [3,0,0]$
- A-space: $[x_1, x_2]$; B-space: x_3

5. Interlude: Finding an Optimal Design

File: `tutorial_05_point_based_optimal_design`

Special optimization functions:

- `design_optimize_quantities_of_interest`
- `design_optimize_performance_score`

Task:

Find optimal designs for the 2D car crash problem: one with the best acceleration, another with the best score.

6. Interlude: Using Python Bottom-up Mappings

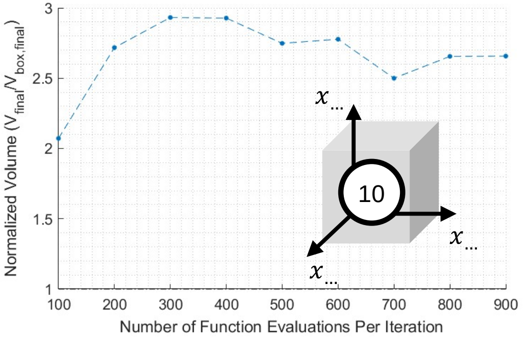
File: `tutorial_06_python_bottom_up_mapping`

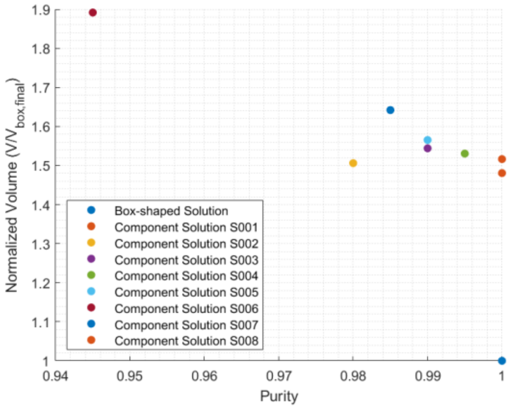
Setting up a bottom-up mapping that uses Python code for its system response: `BottomUpMappingPython`

Task:

- Find the optimal box for the 2D car crash problem using the MATLAB function `car_crash_2d`
- Find the optimal box for the 2D car crash problem using the Python function `car_crash_2d_python.py`

7. Interlude: Performing Batch Analysis





ID	ReferenceID	Number Samples Per Iteration	Growth Rate	Tolerance Purity Consolidation
S001	-	200	0.2	1
S002	S001	100	0.2	1
S003	S001	300	0.2	1
S004	S001	200	0.1	1
S005	S001	200	0.3	1
S006	S001	200	0.2	0.85
S007	S001	200	0.2	1
S008	S001	200	0.2	1

7. Interlude: Performing Batch Analysis

File: `tutorial_07_batch_analysis`

Reading a table: `batch_analysis_read_table`

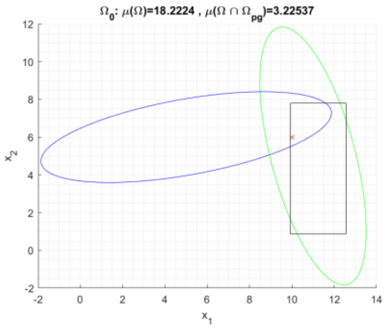
Batch SSO analysis: `batch_sso_stochastic_analysis`

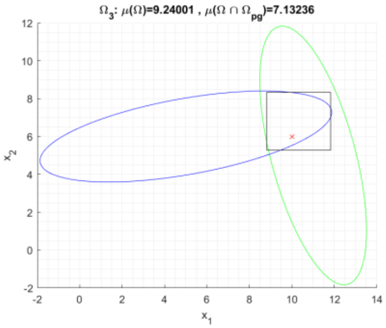
Task:

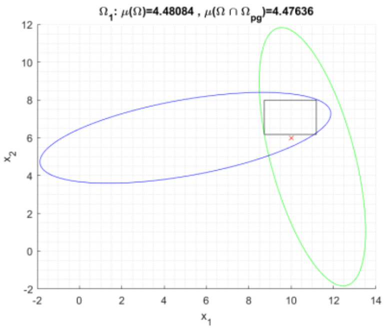
Perform a batch analysis for the following problem:

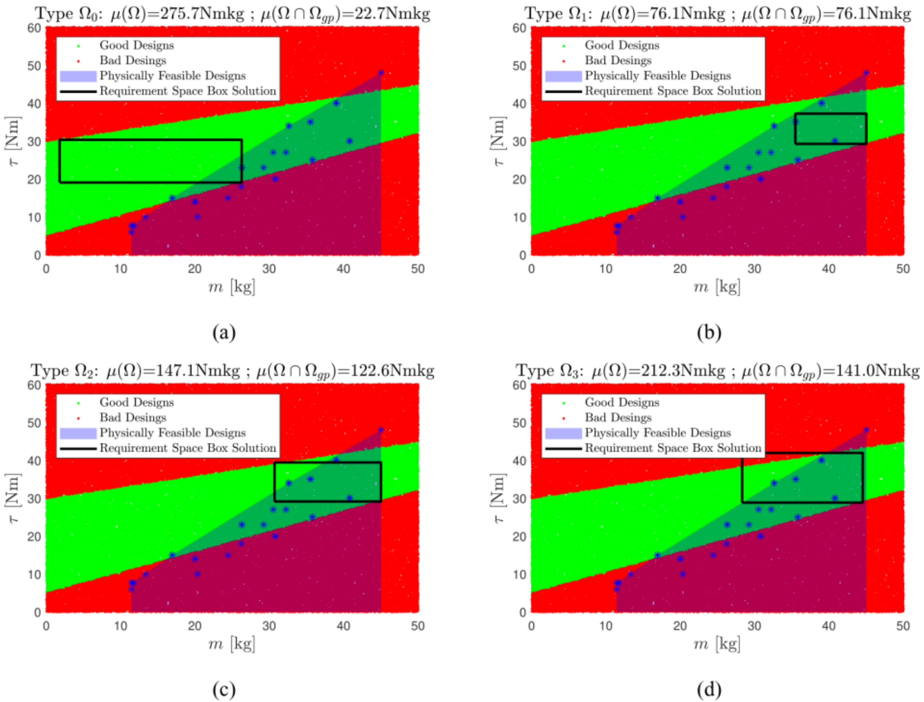
- System response: `tutorial_01_euclidean_distance_3d`
- Design Space: $-6 \leq x_1, x_2, x_3 \leq 6$
- Performance Limits: $distance \leq 5$ (center: $[0,0,0]$)
- Initial Design: $[x_1, x_2, x_3] = [3,0,0]$
- Batch options: 'BatchTestHollowSphere.xlsx'

8. Interlude: Requirement Spaces









8. Interlude: Requirement Spaces

File: `tutorial_08_requirement_spaces`

Important option:

- `'RequirementSpacesType’` (in `sso_stochastic_options`)
- `'PhysicalFeasibilityUpperLimit'` (in `DesignEvaluatorBottomUpMapping`)

Task:

Solve the following requirement spaces problem:

- System response: `two_ellipses_requirement_space`
- Design Space: $-2 \leq x_1 \leq 14\,; -2 \leq x_2 \leq 12$
- Performance Limit: $\text{ellipsenorm} \leq 1$
- Physical Feasibility Limit: $\text{ellipsenorm} \leq 1$
- Initial Design: $[x_1, x_2\,] = [10{,}6]$
- Number of evaluations per iteration: 200
- Apply leanness condition only at the end

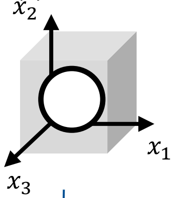
9. Component Solution Space Optimization – Simple Example

Example: Sphere

Components:

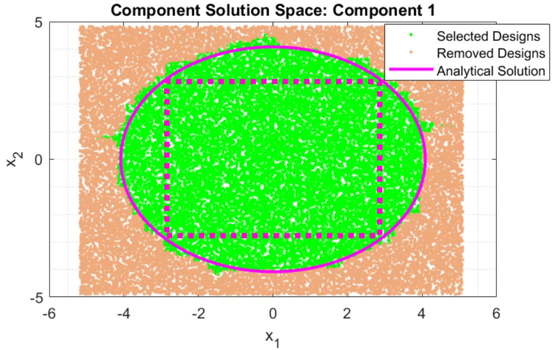
$\boldsymbol{x}_{(1)} = [x_1, x_2]$

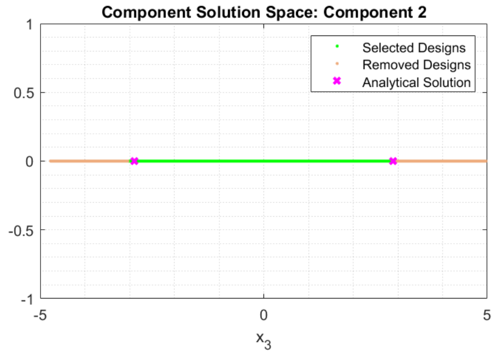
$\boldsymbol{x}_{(2)} = [x_3]$



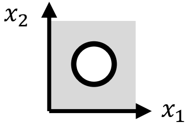
Stochastic Algorithm

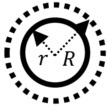
Corner Box Removal





Analytical Solution





$r = R\sqrt{\frac{2}{3}}$

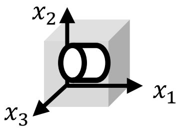
$l = \frac{2R}{\sqrt{3}}$

Solution Box Area:

$$A_b = l_b^2 = \left(\frac{2R}{\sqrt{3}}\right)^2 = R^2\,\frac{4}{3}$$

Component Solution Space Area:

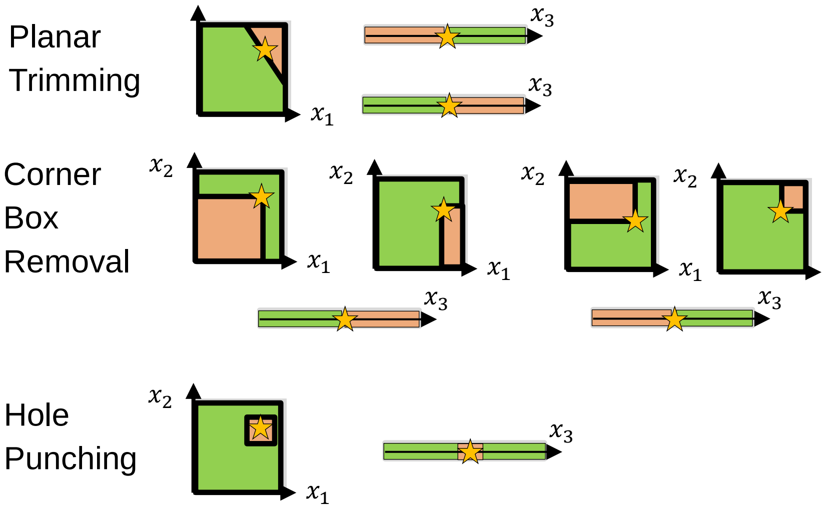
$$A_c = \pi r_c^2 = \pi\left(R\sqrt{\frac{2}{3}}\right)^2 = R^2\,\frac{2\pi}{3}$$

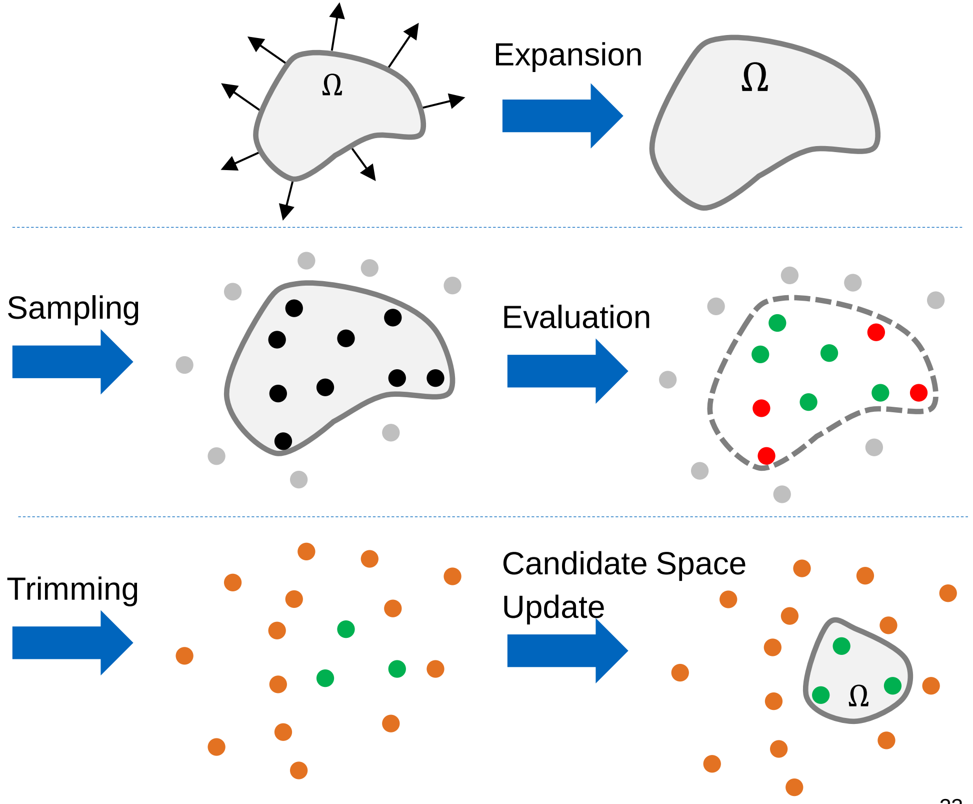


<u>~57% increase in area</u>

9. Component Solution Space Optimization – Simple Example

- Candidate space operations:
  - Expand (given a growth rate g)
  - Identify x_{sample} as inside or outside
  - Be updated given samples x_{in}, x_{out}





9. Component Solution Space Optimization – Simple Example

File: `tutorial_09_component_solution_space_simple`

Candidate Space Definition: `CandidateSpaceBase`

Trimming Methods: `component_trimming_method_<NAME>`

Component SSO function: `sso_component_stochastic`

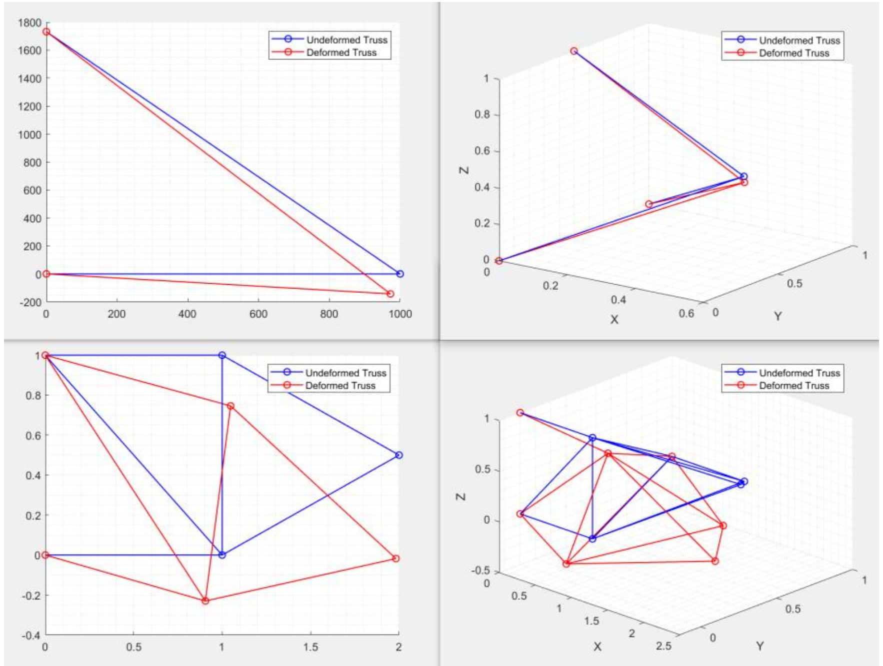
Performance metrics: `postprocess_sso_component_stochastic` → `plot_sso_component_stochastic_metrics`

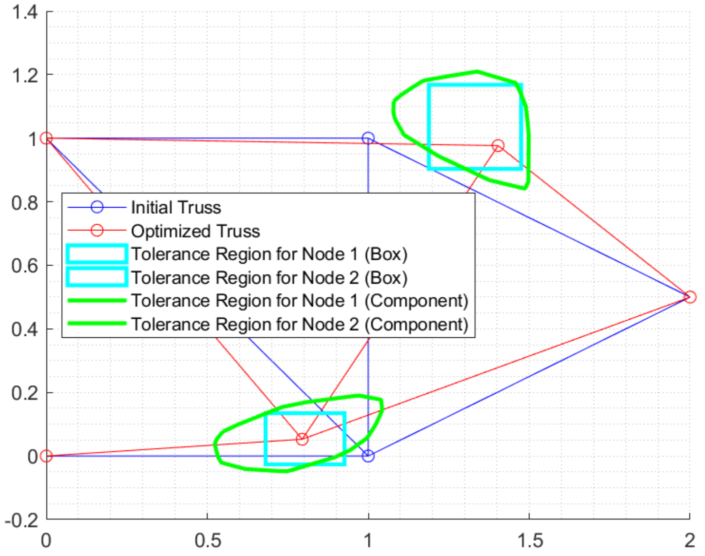
Task:

Solve the following requirement spaces problem:

- System response: `tutorial_01_euclidean_distance_3d`
- Design Space: $-6 \leq x_1, x_2, x_3 \leq 6$
- Performance Limits: $distance \leq 5$ (center: $[0,0,0]$)
- Initial Design: $[x_1, x_2, x_3] = [3,0,0]$
- Number of samples per iteration: 300

10. Component Solution Space Optimization – Advanced





10. Component Solution Space Optimization – Advanced

File: `tutorial_10_component_solution_space_advanced`

Options to use:

- Growth rate: 0.1
- Number of exploration/consolidation iterations: 30
- Number of evaluations per iteration: 300

Tasks:

- Setup the bottom-up mapping
- Find the optimum design with minimum displacement
- Find the optimum solution space box starting from the optimum design and having as performance limit a 10% tolerance from the optimum displacement
- Find the optimum component solution space under the same conditions
- Visualize both box and component solution space in the same plot to compare
- Plot performance metrics for each

Laboratory for Product Development and Lightweight Design
TUM School of Engineering and Design
Technical University of Munich

Thank you for your attention!

Eduardo R. Della Noce, M.Sc.
eduardo.noce@tum.de
+49 89 289 15157

LPL

Research funded by the DFG
(German Research Foundation)
DFG Deutsche Forschungsgemeinschaft
Project Number: 454149634