Reference Set Generator (RSG) Manual

Angel E. Rodriguez-Fernandez, Hao Wang, and Oliver Schütze

https://github.com/aerfangel/RSG/

2025-June-04 — Version 1

Contents

Chapter 1

About this Manual

This document provides a brief and practical guide on how to use the Reference Set Generator (RSG) and configure its parameters.

1.1 How to Cite

For more information on the methodology behind RSG, see: Rodriguez-Fernandez, A.E.; Wang, H.; Schütze, O. Reference Set Generator: A Method for Pareto Front Approximation and Reference Set Generation. Mathematics 2025, 13, 1626. https://doi.org/10.3390/math13101626 [1].

1.2 Source Code

The RSG code is available at: `https://github.com/aerfangel/RSG/`.

1.3 Contact

Report bugs, errors, or comments to `angel.rodriguez@cinvestav.mx`.

Chapter 2

Using RSG and Adjusting its Parameters

2.1 Running RSG

The Reference Set Generator (RSG) computes a reference set (Z) of size N from a starting set P_y. The specific procedure differs for the cases $k = 2$ and $k > 2$ objectives. However, in both cases, a filling step is performed to obtain a set I_y containing N_f points. RSG can be called in MATLAB as follows:

$$[Z, I_y] = \texttt{RSG}(P_y, N_f, N, \texttt{clean\_method}, \texttt{threshold}, \texttt{epsInterval}, \texttt{eps\_def}, \texttt{minptsInterval}, \texttt{trimming}, \texttt{endpoints}, \texttt{subsel});$$

where

- Z is the result of RSG;
- I_y is the filled set;
- P_y is the starting approximation of the Pareto front (PF);
- N is the number of desired points in Z;
- N_f is the number of desired points for the filling;
- `clean_method` (for problems with $k > 2$ objectives) is the triangle cleaning method. It can be `'long'`, `'cond'`, `'area'`, or `'off'`.
  - `'long'` (recommended) cleans the triangulation using the length of the largest side of each triangle;
  - `'cond'` uses the condition number of the matrix formed by the triangle's vertices;
  - `'area'` uses the area of the triangles;
  - `'off'` skips the cleaning step.

  For bi-objective problems, set `clean_method` to an empty array (`clean_method = []`). For more details, see Section 2.3;
- `threshold` (for problems with $k > 2$ objectives) is the cleaning threshold value. Triangles with a property value greater than `threshold` will be removed. For bi-objective problems, set `threshold` as an empty array (`threshold = []`). See Section 2.3 for more details;
- `epsInterval` is an array $[a, b]$, where a and b are the initial and final radii used by DBSCAN for grid search. (A discussion on how to select these parameters is presented in Section 2.2);
- `eps_def` is the step size for `epsInterval`. DBSCAN will try radii in the set $\{a, a + \texttt{eps\_def}, a + 2 \cdot \texttt{eps\_def}, \ldots, b\}$ (see Section 2.2);
- `minptsInterval` is an array $[a, b]$ specifying the initial and final values for the minimum number of points parameter in DBSCAN;
- `trimming` is a boolean variable: 1 reduces the filled set I_y to Z, and 0 only computes the filled set;
- `endpoints` is a boolean variable: 1 includes the endpoints in Z, and 0 does not;
- `subsel` is a string that selects the reduction method used to go from I_y to Z. Options are `'means'`, `'medoids'`, or `'spectral'`.

An example run of RSG for some functions can be found in Section 3.

2.2 Component Detection Parameters (`epsInterval`, `eps_def`, `minptsInterval`)

The variables `epsInterval`, `eps_def`, and `minptsInterval` control the component detection step. A grid search will be performed for this purpose.

If P_y consists of one single component, we recommend setting: `epsInterval` $= [\max(\max(P_y)) - \min(\min(P_y)), \max(\max(P_y)) - \min(\min(P_y))]$, `eps_def` $= 1$, and `minptsInterval` $= [3, 3]$.

If P_y consists of several components (or if no prior information is available about P_y), we recommend the following settings:

- **For bi-objective problems:** `minptsInterval` $= [2, 3]$, `epsInterval` $= [0.1\bar{d}, 0.15\bar{d}]$, `eps_def` $= 0.1\bar{d}$;
- **Otherwise:** `minptsInterval` $= [3, 4]$, `epsInterval` $= [0.19\bar{d}, 0.23\bar{d}]$, `eps_def` $= 0.1\bar{d}$;

where $\bar{d}$ is the average pairwise distance between all points in P_y:

$$\bar{d} = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \|p_i - p_j\|,$$

with n being the number of points in P_y, and p_i, p_j the individual points.

2.3 Cleaning Parameters (`clean_method`, `threshold`)

The variable `clean_method` and its associated `threshold` are parameters for the triangle cleaning procedure. This step only applies to problems with $k > 2$ objectives. For bi-objective problems, set both `clean_method` and `threshold` as empty arrays.

For problems with $k > 3$ objectives, RSG includes a hidden variable called `debug`, which must be set to `true`. To activate debug mode and determine appropriate values for `clean_method` and `threshold`, we recommend following the steps described below. The function `CONV3-4` from [1] will be used to illustrate this process. The data P_y will be available on GitHub (in the subfolder `https://github.com/aerfangel/RSG/tree/main/Data/Py/CONV3-4.mat`) so that users can reproduce these steps.

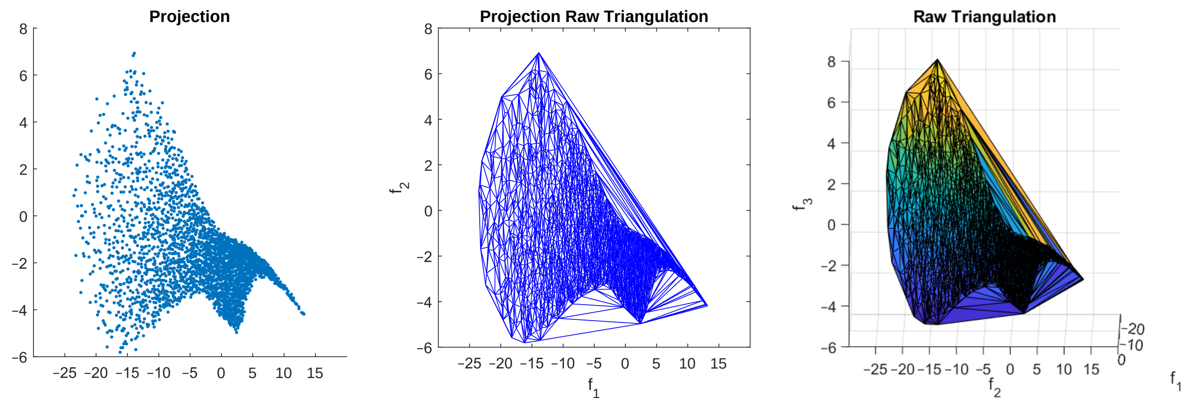
1. Include a breakpoint at the line that contains `if strcmp(clean_method,'cond')` (line 761).
2. When running RSG, add an additional `true` input at the end:

   `RSG`$(P_y, N_f, N,$ `clean_method`, `threshold`, `epsInterval`, `eps_def`, `minptsInterval`, `trimming`, `endpoints`, `subsel`, `true`$)$

   This activates debug mode and shows useful plots to help determine the appropriate value for `threshold`.
3. Set `clean_method = 'long'` and an initially high value for `threshold`, e.g., `threshold = 10000`.
4. Run RSG using the suggested values for the cleaning. For the CONV3-4 example, the command looks like:

   $$[Z, I_y] = \texttt{RSG}(P_y, 10000, 300, \texttt{'long'}, 10000, [1, 1], 0.1, [3, 3], \texttt{true}, \texttt{false}, \texttt{'means'}, \texttt{true})$$

5. Execution will pause at the breakpoint (line 761) and display three plots:



(a) Figure 1 (b) Figure 2 (c) Figure 3

Figure 2.1:

These plots show: (1) the projection of the original data, (2) the triangulation in the projected space, and (3) the triangulation in the original space. By comparing Figures 2.1a and 2.1b, we can observe that the triangulation covers areas not occupied by original data points. Filling such triangles would add points that do not belong to the PF.

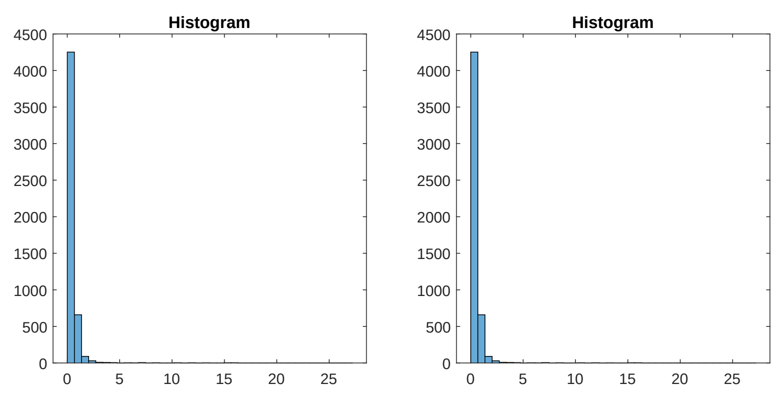
6. Execution remains paused. To clean triangles that span invalid regions, use the cleaning command at line 769 (if `clean_method = 'long'`):

```
DT = rmtriangle(DT_raw, Ry(:,1:end), @tri_long, threshold, comp_name, basename, debug, save_plots, debug); % longitud
```

However, **do not continue** the run. Instead, copy this line and paste it into the MATLAB Command Window:

```
Command Window
K>> DT = rmtriangle( DT_raw, Ry(:,1:end), @tri_long, threshold, comp_name, basename, debug, save_plots, debug ); % longitud
```

This will open two histograms:



(a) Figure 4 (b) Figure 5

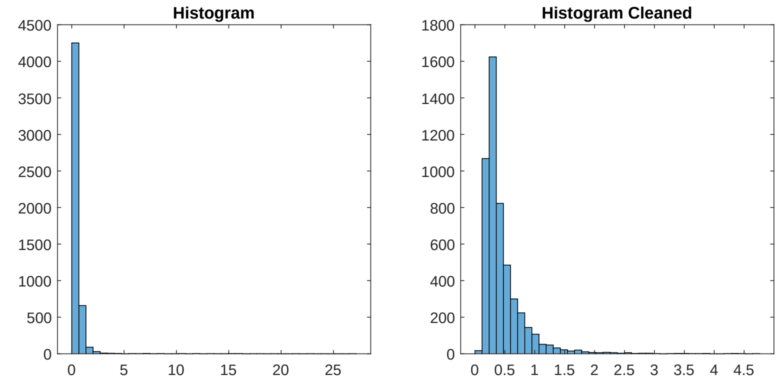
Figure 2.2:

These histograms show the triangle property values (e.g., longest side length) before and after cleaning. Since `threshold = 10000` is very high, no triangles are removed, so both plots are identical.

7. From Figure 2.2a, we estimate a better value for `threshold`. Most triangle lengths are below 5. Set `threshold = 5` in the Command Window and re-run the cleaning line (line 769):

```
Command Window
K>> DT = rmtriangle( DT_raw, Ry(:,1:end), @tri_long, threshold, comp_name, basename, debug, save_plots, debug ); % longitud
K>> threshold = 5;
K>> DT = rmtriangle( DT_raw, Ry(:,1:end), @tri_long, threshold, comp_name, basename, debug, save_plots, debug ); % longitud
```

This will again open two histograms:



(a) Figure 6 (b) Figure 7

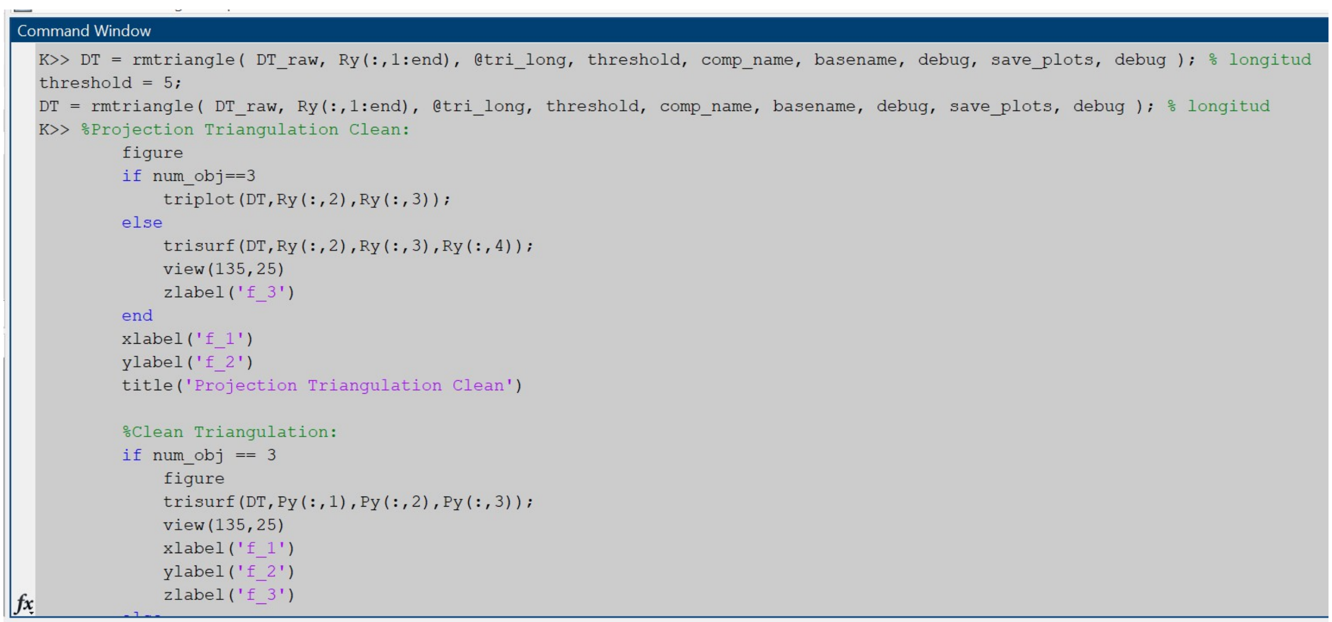
Figure 2.3:

8. We now observe an effect from cleaning. To visualize the cleaned triangulation, copy and run the block of code from line 773 to 820, starting with:

```
% Projection Triangulation Clean:
```

and ending with:

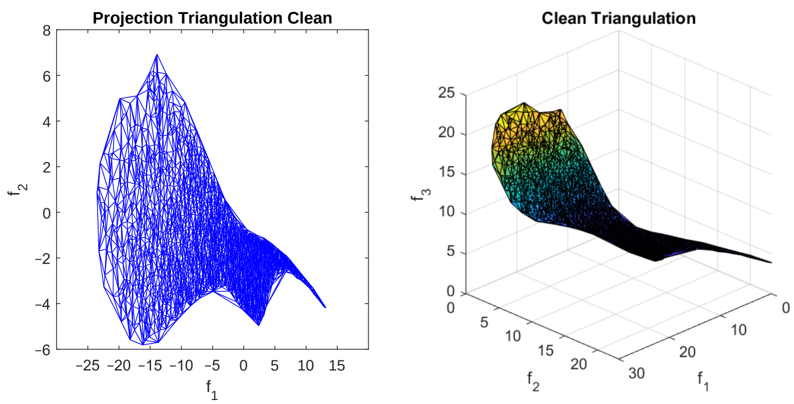
```
title('Clean Triangulation')
```



```
Command Window
K>> DT = rmtriangle( DT_raw, Ry(:,1:end), @tri_long, threshold, comp_name, basename, debug, save_plots, debug ); % longitud
threshold = 5;
DT = rmtriangle( DT_raw, Ry(:,1:end), @tri_long, threshold, comp_name, basename, debug, save_plots, debug ); % longitud
K>> %Projection Triangulation Clean:
        figure
        if num_obj==3
            triplot(DT,Ry(:,2),Ry(:,3));
        else
            trisurf(DT,Ry(:,2),Ry(:,3),Ry(:,4));
            view(135,25)
            zlabel('f_3')
        end
        xlabel('f_1')
        ylabel('f_2')
        title('Projection Triangulation Clean')

        %Clean Triangulation:
        if num_obj == 3
            figure
            trisurf(DT,Py(:,1),Py(:,2),Py(:,3));
            view(135,25)
            xlabel('f_1')
            ylabel('f_2')
            zlabel('f_3')
fx
```

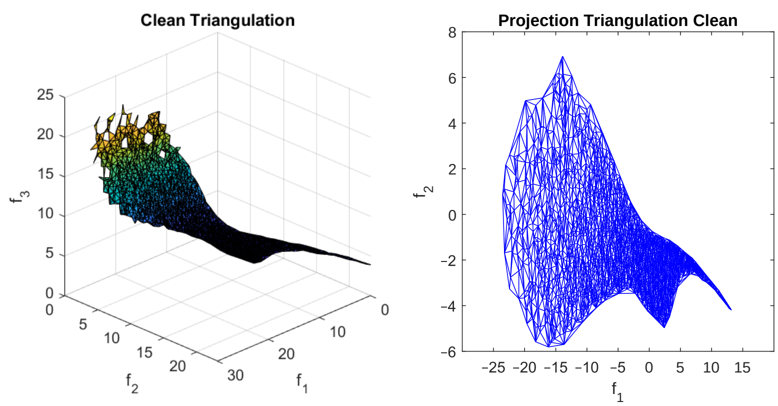
This will open two plots:



(a) Figure 8 (b) Figure 9

Figure 2.4:

9. From Figure 2.4a, we see that some irrelevant triangles remain. Using the cleaned histogram (Figure 2.3b), we refine `threshold`. Setting it too low (e.g., `threshold = 1.5`) might remove valid triangles. Repeat steps 7 and 8 with `threshold = 1.5` to visualize the effect:

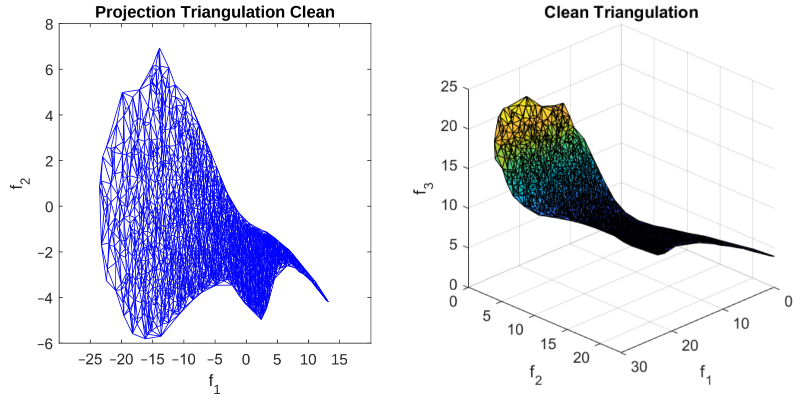


(a) Figure 10 (b) Figure 11

Figure 2.5:

We observe that this threshold is too restrictive.

10. By iteratively adjusting `threshold`, we find that `threshold = 3.75` gives a good result:



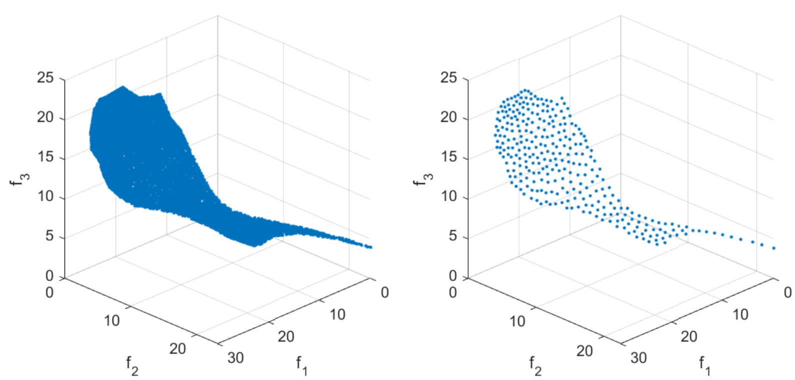
(a) Figure 12 (b) Figure 13

Figure 2.6:

11. Finally, stop the debug run. Then re-run RSG (without the debug flag) using the value of `threshold` found during the previous steps:

$$[Z, I_y] = \texttt{RSG}(P_y, 10000, 300, \texttt{'long'}, 3.75, [1,1], 0.1, [3,3], \texttt{true}, \texttt{false}, \texttt{'means'})$$

 where Z and I_y look as follows:



(a) I_y ($N_f = 10{,}000$) (b) Z ($N = 300$)

Figure 2.7:

2.3.1 Cleaning Multiple Components

For disconnected problems with $k > 2$ objectives, the main challenge is that each independent component may require a different value of `threshold` for triangle cleaning. To address this, we recommend the following steps:

1. Detect components *before* running RSG.

2. Follow the steps in Section 2.3 to determine the appropriate value of `threshold` for each component. It is important to set `trimming=false` (only for this step) to save time, since subset selection will be performed after merging all the filled components.

3. Using the computed `threshold` values, run RSG on each component and store the resulting filled set as I_y^i, with $i = 1, \ldots, n_c$, where n_c is the number of components.

4. Merge the sets into $I_y = \cup_{i=1}^{n_c} I_y^i$, and apply k-means (or any other desired selection method) on I_y with N clusters to obtain Z as the set of centroids found by k-means.

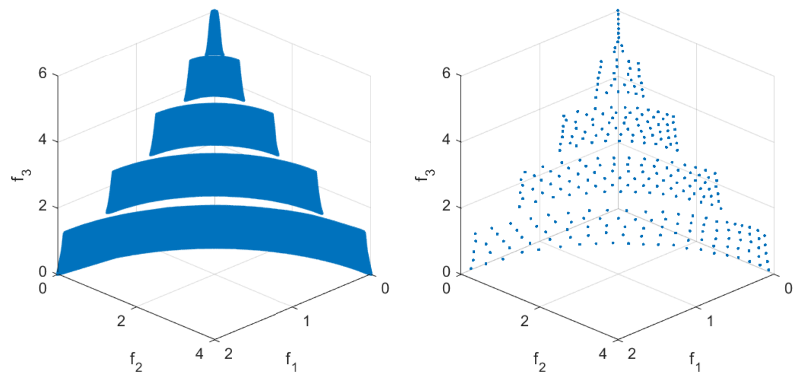
To reproduce the results of [1] for the WFG2 and DTLZ7 problems, we have included scripts that perform steps 3 and 4 (steps 1 and 2 were carried out manually beforehand). These scripts are available at the following GitHub paths:

https://github.com/aerfangel/RSG/tree/main/Data/Clean_Multiple_Components/run_RSG_load_components_WFG2.m
https://github.com/aerfangel/RSG/tree/main/Data/Clean_Multiple_Components/run_RSG_load_components_DTLZ7.m

To run these scripts, the reader must also download the additional data files accessed by the scripts, as well as the source code for RSG. All required files, including the scripts and dependencies, are available at:

https://github.com/aerfangel/RSG/tree/main/Data/Clean_Multiple_Components/

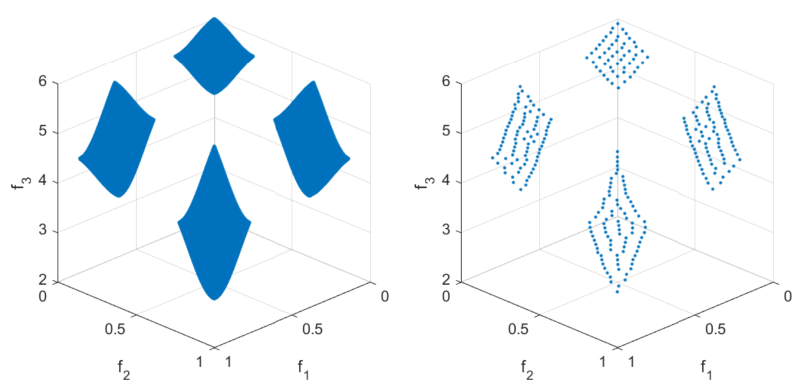
These scripts compute both Z and I_y, which for WFG2 are illustrated below:



(a) I_y (b) Z

Figure 2.8: Filled set I_y and RSG reference set Z for the WFG2 problem.

And for DTLZ7:



(a) I_y (b) Z

Figure 2.9: Filled set I_y and RSG reference set Z for the DTLZ7 problem.

Chapter 3

Examples

In this section, we present some examples of how to call RSG for specific problems. The parameters naturally depend on the set P_y, so we have included these sets in the GitHub repository to allow users to reproduce the results of [1].

3.1 $k = 2$ Objectives

3.1.1 ZDT1

To reproduce the results for problem ZDT1, use P_y from the following GitHub path:

`https://github.com/aerfangel/RSG/tree/main/Data/Py/ZDT1.mat`

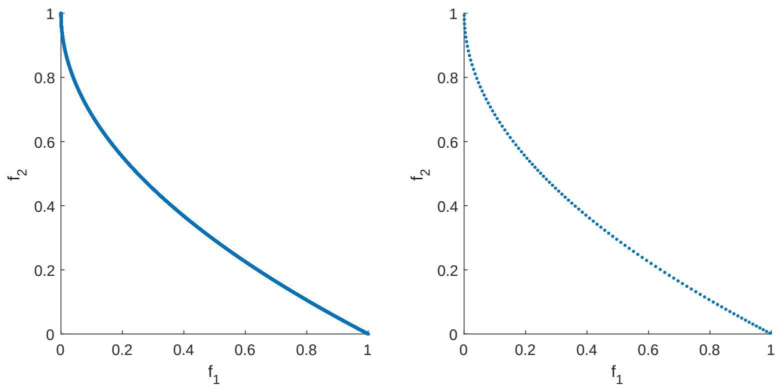
Then, call RSG as follows:

$[Z, I_y]$=RSG(P_y, 10000, 100, `[]`, `[]`, `[1.5,1.5]`, `0.1`, `[3,3]`, `true`, `false`, `'means'`);

Hereby, the chosen input parameter values are as follows (see also Section 2.1):

- The first parameter refers to the starting set P_y, obtained from `ZDT1.mat`;
- The second parameter, $N_f = 10000$, corresponds to the size of the filling;
- The third parameter, $N = 100$, indicates the size of the obtained reference set Z;
- The fourth parameter `clean_method=[]` and the fifth parameter `threshold=[]` are left as empty vectors, as they only apply to problems with $k > 2$ objectives;
- The sixth parameter, `eps_Interval = [1.5, 1.5]`, is set with identical initial and final values so that the component detection grid search only performs one run;
- The seventh parameter is set as `eps_def = 0.1`;
- The eighth parameter, `minptsInterval = [3, 3]`, is similarly fixed for a single grid search run;
- The ninth parameter, `trimming = true`, enables reduction;
- The tenth parameter, `endpoints = false`, omits the inclusion of endpoints in Z;
- The eleventh parameter, `subsel = 'means'`, specifies k-means clustering for the reduction.

This will output Z and I_y, which should look like the following:



(a) I_y (b) Z

Figure 3.1: Filled set I_y and RSG reference Z for the ZDT1 problem and starting set P_y.

3.1.2 ZDT3

To reproduce the results for problem ZDT3, use P_y from the following GitHub path:

`https://github.com/aerfangel/RSG/tree/main/Data/Py/ZDT3.mat`

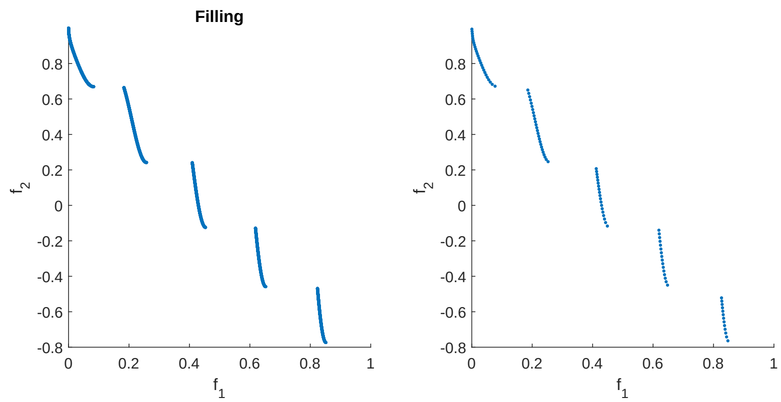
Then, call RSG as follows:

$[Z, I_y]$=`RSG`$(P_y, 10000, 100,$ `[]`, `[]`, `[0.1,0.1]`, `0.1`, `[3,3]`, `true`, `false`, `'means'`$)$;

Hereby, the chosen input parameter values are as follows (see also Section 2.1):

- The first parameter refers to the starting set P_y, obtained from `ZDT3.mat`;
- The second parameter, $N_f = 10000$, corresponds to the size of the filling;
- The third parameter, $N = 100$, indicates the size of the obtained reference set Z;
- The fourth parameter `clean_method=[]` and the fifth parameter `threshold=[]` are left as empty vectors, as they only apply to problems with $k > 2$ objectives;
- The sixth parameter, `eps_Interval = [0.1, 0.1]`, is set with identical initial and final values so that the component detection grid search only performs one run;
- The seventh parameter is set as `eps_def = 0.1`;
- The eighth parameter, `minptsInterval = [3, 3]`, is similarly fixed for a single grid search run;
- The ninth parameter, `trimming = true`, enables reduction;
- The tenth parameter, `endpoints = false`, omits the inclusion of endpoints in Z;
- The eleventh parameter, `subsel = 'means'`, specifies k-means clustering for the reduction.

This will output Z and I_y, which should look like the following:



(a) I_y (b) Z

Figure 3.2: Filled set I_y and RSG reference Z for the ZDT3 problem and starting set P_y.

3.2 $k > 2$ Objectives

3.2.1 CONV3

To reproduce the results for problem CONV3, use P_y from the following GitHub path:

`https://github.com/aerfangel/RSG/tree/main/Data/Py/CONV3.mat`

Then, call RSG as follows:

$[Z, I_y]$=`RSG`(P_y, 10000, 300, `'long'`, `2.5`, `[0.15,0.15]`, `0.1`, `[3,3]`, `true`, `false`, `'means'`);

Hereby, the chosen input parameter values are as follows (see also Section 2.1):

- The first parameter refers to the starting set P_y, obtained from `CONV3.mat`;
- The second parameter, $N_f = 10000$, corresponds to the size of the filling;
- The third parameter, $N = 300$, indicates the size of the obtained reference set Z;
- The fourth parameter, `clean_method = 'long'`, specifies that triangulation cleaning is based on the longest side of a triangle;
- The fifth parameter, `threshold = 2.5`, indicates that triangles with a longest side greater than `2.5` will be removed;
- The sixth parameter, `eps_Interval = [0.15, 0.15]`, is set with identical initial and final values so that the component detection grid search only performs one run;
- The seventh parameter is set as `eps_def = 0.1`;
- The eighth parameter, `minptsInterval = [3, 3]`, is similarly fixed for a single grid search run;
- The ninth parameter, `trimming = true`, enables reduction;
- The tenth parameter, `endpoints = false`, omits the inclusion of endpoints in Z;
- The eleventh parameter, `subsel = 'means'`, specifies k-means clustering for the reduction.

This will output Z and I_y, which should look like the following:

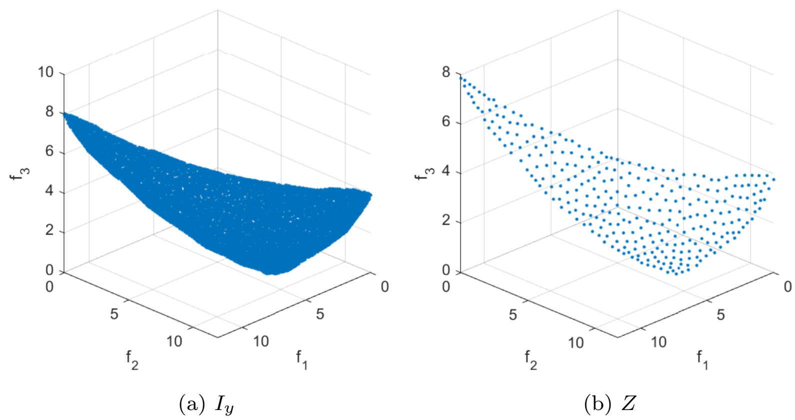


Figure 3.3: Filled set I_y and RSG reference Z for the CONV3 problem and starting set P_y.

3.2.2 CONV3-4

To reproduce the results for problem CONV3-4, use P_y from the following GitHub path:

`https://github.com/aerfangel/RSG/tree/main/Data/Py/CONV3-4.mat`

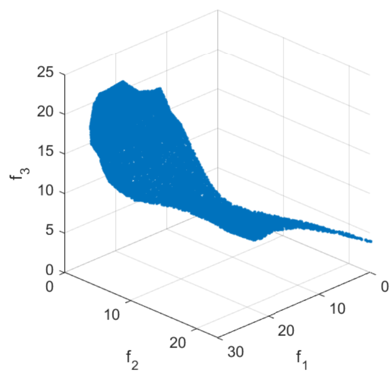
Then, call RSG as follows:

[Z, I_y]=`RSG`(P_y, 10000, 300, `'long'`, `3.75`, `[0.15,0.15]`, `0.1`, `[3,3]`, `true`, `false`, `'means'`);

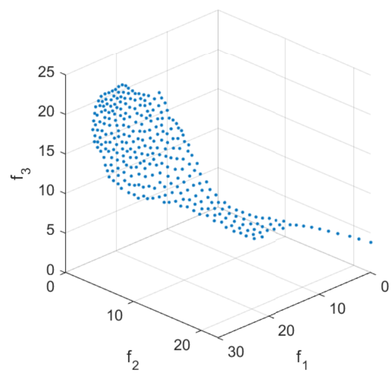
Hereby, the chosen input parameter values are as follows (see also Section 2.1):

- The first parameter refers to the starting set P_y, obtained from `CONV3-4.mat`;
- The second parameter, $N_f = 10000$, corresponds to the size of the filling;
- The third parameter, $N = 300$, indicates the size of the obtained reference set Z;
- The fourth parameter, `clean_method = 'long'`, specifies that triangulation cleaning is based on the longest side of a triangle;
- The fifth parameter, `threshold = 3.75`, indicates that triangles with a longest side greater than `3.75` will be removed;
- The sixth parameter, `eps_Interval = [0.15, 0.15]`, is set with identical initial and final values so that the component detection grid search only performs one run;
- The seventh parameter is set as `eps_def = 0.1`;
- The eighth parameter, `minptsInterval = [3, 3]`, is similarly fixed for a single grid search run;
- The ninth parameter, `trimming = true`, enables reduction;
- The tenth parameter, `endpoints = false`, omits the inclusion of endpoints in Z;
- The eleventh parameter, `subsel = 'means'`, specifies k-means clustering for the reduction.

This will output Z and I_y, which should look like the following:



(a) I_y



(b) Z

Figure 3.4: Filled set I_y and RSG reference Z for the CONV3-4 problem and starting set P_y.

3.2.3 CONV4-2F

To reproduce the results for problem CONV4-2F, use P_y from the following GitHub path:

`https://github.com/aerfangel/RSG/tree/main/Data/Py/CONV4-2F.mat`

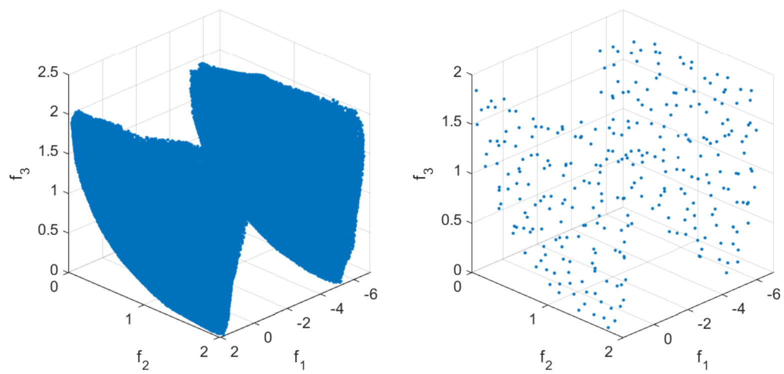
Then, call RSG as follows:

[Z, I_y]=RSG(P_y, 100000, 300, 'long', 1, [0.15,0.15], 0.1, [3,3], true, false, 'means');

Hereby, the chosen input parameter values are as follows (see also Section 2.1):

- The first parameter refers to the starting set P_y, obtained from `CONV4-2F.mat`;
- The second parameter, $N_f = 100000$, corresponds to the size of the filling;
- The third parameter, $N = 300$, indicates the size of the obtained reference set Z;
- The fourth parameter, `clean_method = 'long'`, specifies that triangulation cleaning is based on the longest side of a triangle;
- The fifth parameter, `threshold = 1`, indicates that triangles with a longest side greater than `1` will be removed;
- The sixth parameter, `eps_Interval = [0.15, 0.15]`, is set with identical initial and final values so that the component detection grid search only performs one run;
- The seventh parameter is set as `eps_def = 0.1`;
- The eighth parameter, `minptsInterval = [3, 3]`, is similarly fixed for a single grid search run;
- The ninth parameter, `trimming = true`, enables reduction;
- The tenth parameter, `endpoints = false`, omits the inclusion of endpoints in Z;
- The eleventh parameter, `subsel = 'means'`, specifies k-means clustering for the reduction.

This will output Z and I_y, which should look like the following:



(a) I_y

(b) Z

Figure 3.5: Filled set I_y and RSG reference Z for the CONV4-2F problem and starting set P_y.

3.2.4 DTLZ1

To reproduce the results for problem DTLZ1, use P_y from the following GitHub path:

`https://github.com/aerfangel/RSG/tree/main/Data/Py/DTLZ1.mat`

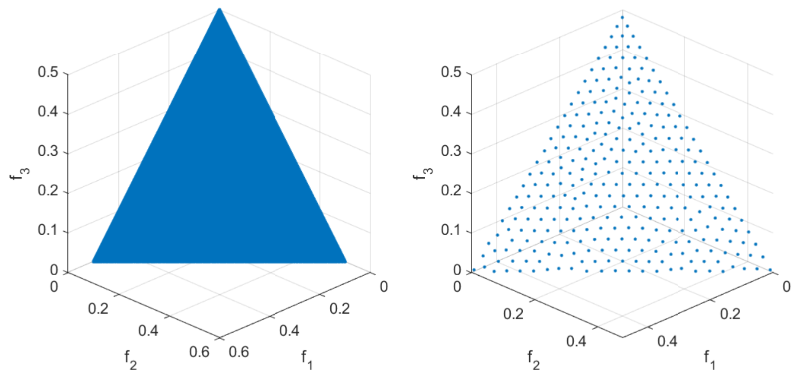
Then, call RSG as follows:

`[`Z, I_y`]=RSG(`P_y, 1000000, 300, `'off'`, `'off'`, `[1.5,1.5]`, `0.1`, `[3,3]`, `true`, `false`, `'means'`);

Hereby, the chosen input parameter values are as follows (see also Section 2.1):

- The first parameter refers to the starting set P_y, obtained from `DTLZ1.mat`;
- The second parameter, $N_f = 1000000$, corresponds to the size of the filling;
- The third parameter, $N = 300$, indicates the size of the obtained reference set Z;
- The fourth parameter, `clean_method = 'off'`, specifies that triangulation cleaning is omitted;
- The fifth parameter, `threshold = 'off'`, indicates that triangulation cleaning is omitted;
- The sixth parameter, `eps_Interval = [1.5, 1.5]`, is set with identical initial and final values so that the component detection grid search only performs one run;
- The seventh parameter is set as `eps_def = 0.1`;
- The eighth parameter, `minptsInterval = [3, 3]`, is similarly fixed for a single grid search run;
- The ninth parameter, `trimming = true`, enables reduction;
- The tenth parameter, `endpoints = false`, omits the inclusion of endpoints in Z;
- The eleventh parameter, `subsel = 'means'`, specifies k-means clustering for the reduction.

This will output Z and I_y, which should look like the following:



(a) I_y (b) Z

Figure 3.6: Filled set I_y and RSG reference Z for the DTLZ1 problem and starting set P_y.

3.2.5 DTLZ2

To reproduce the results for problem DTLZ2, use P_y from the following GitHub path:

`https://github.com/aerfangel/RSG/tree/main/Data/Py/DTLZ2.mat`

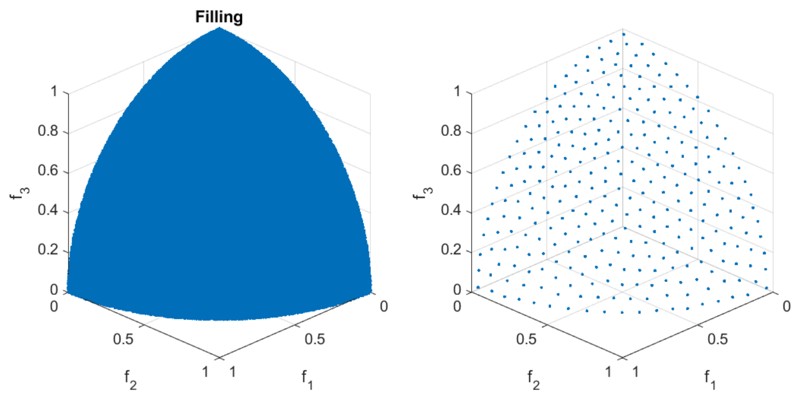
Then, call RSG as follows:

$[Z, I_y]$=RSG(P_y, 1000000, 300, 'long', 0.15, [0.25,0.25], 0.1, [3,3], true, false, 'means');

Hereby, the chosen input parameter values are as follows (see also Section 2.1):

- The first parameter refers to the starting set P_y, obtained from `DTLZ2.mat`;
- The second parameter, $N_f = 1000000$, corresponds to the size of the filling;
- The third parameter, $N = 300$, indicates the size of the obtained reference set Z;
- The fourth parameter, `clean_method = 'long'`, specifies that triangulation cleaning is based on the longest side of a triangle;
- The fifth parameter, `threshold = 0.15`, indicates that triangles with a longest side greater than `0.15` will be removed;
- The sixth parameter, `eps_Interval = [0.25, 0.25]`, is set with identical initial and final values so that the component detection grid search only performs one run;
- The seventh parameter is set as `eps_def = 0.1`;
- The eighth parameter, `minptsInterval = [3, 3]`, is similarly fixed for a single grid search run;
- The ninth parameter, `trimming = true`, enables reduction;
- The tenth parameter, `endpoints = false`, omits the inclusion of endpoints in Z;
- The eleventh parameter, `subsel = 'means'`, specifies k-means clustering for the reduction.

This will output Z and I_y, which should look like the following:



(a) I_y (b) Z

Figure 3.7: Filled set I_y and RSG reference Z for the DTLZ2 problem and starting set P_y.

3.2.6 CDTLZ2

To reproduce the results for problem CDTLZ2, use P_y from the following GitHub path:

`https://github.com/aerfangel/RSG/tree/main/Data/Py/CDTLZ2.mat`

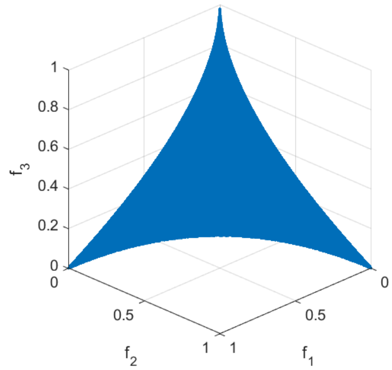
Then, call RSG as follows:

$$[Z, I_y]\texttt{=RSG}(P_y, 1000000, 300, \texttt{'long'}, \texttt{0.064}, \texttt{[1.5,1.5]}, \texttt{0.1}, \texttt{[3,3]}, \texttt{true}, \texttt{false}, \texttt{'means'});$$

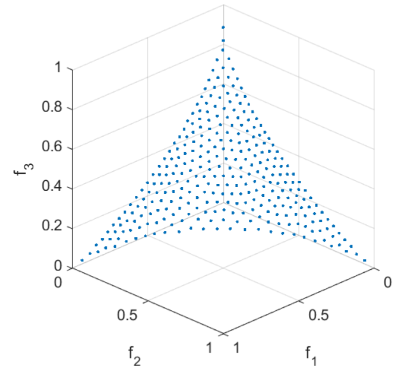
Hereby, the chosen input parameter values are as follows (see also Section 2.1):

- The first parameter refers to the starting set P_y, obtained from `CDTLZ2.mat`;
- The second parameter, $N_f = 1000000$, corresponds to the size of the filling;
- The third parameter, $N = 300$, indicates the size of the obtained reference set Z;
- The fourth parameter, `clean_method = 'long'`, specifies that triangulation cleaning is based on the longest side of a triangle;
- The fifth parameter, `threshold = 0.064`, indicates that triangles with a longest side greater than `0.064` will be removed;
- The sixth parameter, `eps_Interval = [1.5, 1.5]`, is set with identical initial and final values so that the component detection grid search only performs one run;
- The seventh parameter is set as `eps_def = 0.1`;
- The eighth parameter, `minptsInterval = [3, 3]`, is similarly fixed for a single grid search run;
- The ninth parameter, `trimming = true`, enables reduction;
- The tenth parameter, `endpoints = false`, omits the inclusion of endpoints in Z;
- The eleventh parameter, `subsel = 'means'`, specifies k-means clustering for the reduction.

This will output Z and I_y, which should look like the following:



(a) I_y



(b) Z

Figure 3.8: Filled set I_y and RSG reference Z for the CDTLZ2 problem and starting set P_y.

3.2.7 C2-DTLZ2

To reproduce the results for problem C2-DTLZ2, use P_y from the following GitHub path:

`https://github.com/aerfangel/RSG/tree/main/Data/Py/C2-DTLZ2.mat`

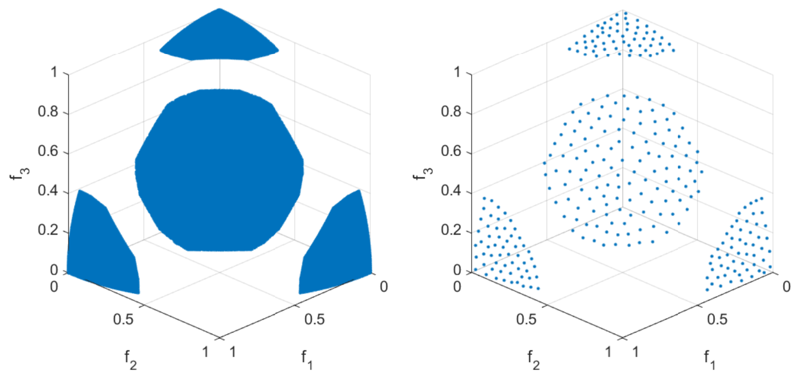
Then, call RSG as follows:

[Z, I_y]=RSG(P_y, 200000, 300, 'off', 'off', [1.5,1.5], 0.1, [3,3], true, false, 'means');

Hereby, the chosen input parameter values are as follows (see also Section 2.1):

- The first parameter refers to the starting set P_y, obtained from `C2-DTLZ2.mat`;
- The second parameter, $N_f = 200000$, corresponds to the size of the filling;
- The third parameter, $N = 300$, indicates the size of the obtained reference set Z;
- The fourth parameter, `clean_method = 'off'`, specifies that triangulation cleaning is omitted;
- The fifth parameter, `threshold = 'off'`, indicates that triangulation cleaning is omitted;
- The sixth parameter, `eps_Interval = [1.5, 1.5]`, is set with identical initial and final values so that the component detection grid search only performs one run;
- The seventh parameter is set as `eps_def = 0.1`;
- The eighth parameter, `minptsInterval = [3, 3]`, is similarly fixed for a single grid search run;
- The ninth parameter, `trimming = true`, enables reduction;
- The tenth parameter, `endpoints = false`, omits the inclusion of endpoints in Z;
- The eleventh parameter, `subsel = 'means'`, specifies k-means clustering for the reduction.

This will output Z and I_y, which should look like the following:



(a) I_y (b) Z

Figure 3.9: Filled set I_y and RSG reference Z for the C2-DTLZ2 problem and starting set P_y.

3.2.8 IDTLZ1

To reproduce the results for problem IDTLZ1, use P_y from the following GitHub path:

`https://github.com/aerfangel/RSG/tree/main/Data/Py/IDTLZ1.mat`

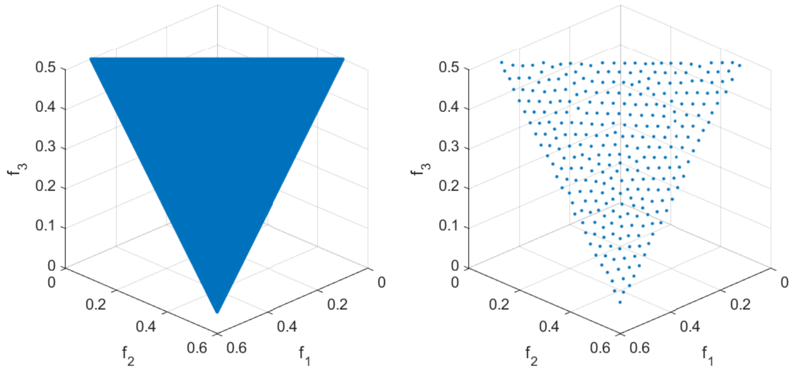
Then, call RSG as follows:

[Z, I_y]=RSG(P_y, 500000, 300, 'long', 0.1, [1.5,1.5], 0.1, [3,3], true, false, 'means');

Hereby, the chosen input parameter values are as follows (see also Section 2.1):

- The first parameter refers to the starting set P_y, obtained from `IDTLZ1.mat`;
- The second parameter, $N_f = 500000$, corresponds to the size of the filling;
- The third parameter, $N = 300$, indicates the size of the obtained reference set Z;
- The fourth parameter, `clean_method = 'long'`, specifies that triangulation cleaning is based on the longest side of a triangle;
- The fifth parameter, `threshold = 0.1`, indicates that triangles with a longest side greater than `0.1` will be removed;
- The sixth parameter, `eps_Interval = [1.5, 1.5]`, is set with identical initial and final values so that the component detection grid search only performs one run;
- The seventh parameter is set as `eps_def = 0.1`;
- The eighth parameter, `minptsInterval = [3, 3]`, is similarly fixed for a single grid search run;
- The ninth parameter, `trimming = true`, enables reduction;
- The tenth parameter, `endpoints = false`, omits the inclusion of endpoints in Z;
- The eleventh parameter, `subsel = 'means'`, specifies k-means clustering for the reduction.

This will output Z and I_y, which should look like the following:



(a) I_y (b) Z

Figure 3.10: Filled set I_y and RSG reference Z for the IDTLZ1 problem and starting set P_y.

3.2.9 IDTLZ2

To reproduce the results for problem IDTLZ2, use P_y from the following GitHub path:

`https://github.com/aerfangel/RSG/tree/main/Data/Py/IDTLZ2.mat`

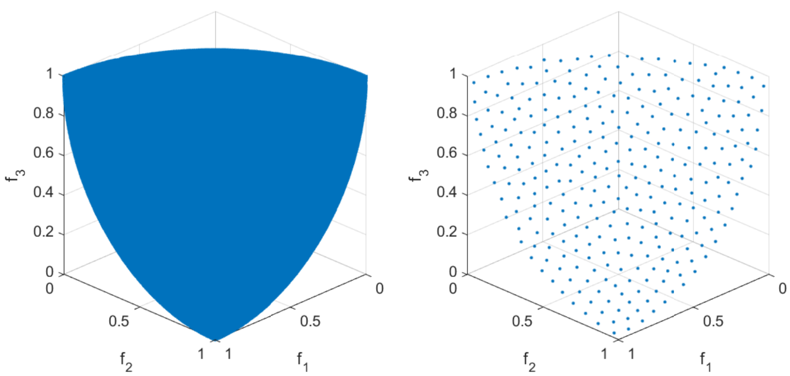
Then, call RSG as follows:

[Z, I_y]=RSG(P_y, 500000, 300, 'long', 10, [1.5,1.5], 0.1, [3,3], true, false, 'means');

Hereby, the chosen input parameter values are as follows (see also Section 2.1):

- The first parameter refers to the starting set P_y, obtained from `IDTLZ2.mat`;
- The second parameter, $N_f = 500000$, corresponds to the size of the filling;
- The third parameter, $N = 300$, indicates the size of the obtained reference set Z;
- The fourth parameter, `clean_method = 'long'`, specifies that triangulation cleaning is based on the longest side of a triangle;
- The fifth parameter, `threshold = 10`, indicates that triangles with a longest side greater than `10` will be removed;
- The sixth parameter, `eps_Interval = [1.5, 1.5]`, is set with identical initial and final values so that the component detection grid search only performs one run;
- The seventh parameter is set as `eps_def = 0.1`;
- The eighth parameter, `minptsInterval = [3, 3]`, is similarly fixed for a single grid search run;
- The ninth parameter, `trimming = true`, enables reduction;
- The tenth parameter, `endpoints = false`, omits the inclusion of endpoints in Z;
- The eleventh parameter, `subsel = 'means'`, specifies k-means clustering for the reduction.

This will output Z and I_y, which should look like the following:



(a) I_y (b) Z

Figure 3.11: Filled set I_y and RSG reference Z for the IDTLZ2 problem and starting set P_y.

Bibliography

[1] Angel E. Rodriguez-Fernandez, Hao Wang, and Oliver Schütze. Reference Set Generator: A method for Pareto front approximation and reference set generation. *Mathematics*, 13(10), 2025.