|  |  |
| --- | --- |
| **Procedural approach** | **Object-Oriented interface** |
| **from** SALib**.**test\_functions **import** Sobol\_G  **from** SALib**.**analyze **import** morris  **from** SALib**.**sample**.**morris **import** sample  problem **=** **{**  'num\_vars'**:** 8**,**  'names'**:** **[**'x1'**,** 'x2'**,** 'x3'**,** 'x4'**,**  'x5'**,** 'x6'**,** 'x7'**,** 'x8'**],**  'groups'**:** **None,**  'bounds'**:** **[[**0.0**,** 1.0**],]** **\*** 8  **}**  X **=** sample**(**problem**,** N**=**1000**,**  num\_levels**=**4**)**  Y **=** Sobol\_G**.**evaluate**(**X**)**  Si **=** morris**.**analyze**(**  problem**,** X**,** Y**,**  num\_levels**=**4**,**  num\_resamples**=**100  **)**  ax = Si**.**plot**()**  Si\_df **=** Si**.**to\_df**()** | **from** SALib**.**test\_functions **import** Sobol\_G  **from** SALib **import** ProblemSpec  sp **=** ProblemSpec**({**  'names'**:** **[**'x1'**,** 'x2'**,** 'x3'**,** 'x4'**,**  'x5'**,** 'x6'**,** 'x7'**,** 'x8'**],**  'groups'**:** **None,**  'bounds'**:** **[[**0.0**,** 1.0**],]** **\*** 8**,**  'outputs'**:** **[**'Y'**]**  **})**  **(**sp**.**sample\_morris**(**1000**,** num\_levels**=**4**)**  **.**evaluate**(**Sobol\_G**.**evaluate**)**  **.**analyze\_morris**(**num\_levels**=**4**,**  num\_resamples**=**100**))**  **print(**sp**)**  ax = sp**.**plot**()**  Si\_df **=** sp**.**to\_df**()** |
| 1. Note that it is not necessary to directly import the sampling and analysis methods if using the object-oriented interface compared to procedural approach 2. Instantiation of an SALib Problem specification compared to the original dictionary-based specification for the Sobol’ G-function. Note that `num\_vars` is now optional, and inferred from `names` if not provided 3. The trifecta of sampling, model evaluation, and analysis with the Morris method. The default values for `num\_levels` (for the Morris analysis) and `num\_resamples` (for bootstrapping) are shown.  Model evaluation and analysis can be parallelized with the OO interface by adding an `nprocs` argument to the `evaluate` and `analyze` methods. 4. Print results and associated information 5. Produces an indicative plot. The returned object is a matplotlib axes object which can be further modified 6. Both OO and procedural approaches support conversion to a Pandas DataFrame for further analysis. | |

Print output

Samples:

8 parameters: ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8']

9000 evaluations

Outputs:

1 outputs: ['Y']

9000 evaluations

Analysis:

mu mu\_star sigma mu\_star\_conf

x1 0.022084 2.612547 2.737028 0.052250

x2 0.047827 1.441302 1.634405 0.048126

x3 -0.034698 0.596377 0.691939 0.017550

x4 -0.028873 0.328173 0.384837 0.013199

x5 -0.001124 0.033406 0.039273 0.001205

x6 -0.001552 0.033400 0.039238 0.001293

x7 -0.001546 0.034216 0.039986 0.001227

x8 0.001518 0.034003 0.039842 0.001253

Plot output

Chart, bar chart

Description automatically generated

Plot customization

|  |
| --- |
| **import** matplotlib**.**pyplot **as** plt  **from** SALib**.**plotting**.**bar **import** plot **as** barplot  fig**,** ax1 **=** plt**.**subplots**(**1**,**1**,** figsize**=(**12**,**6**))**  ax1 **=** barplot**(**Si**.**to\_df**(),** ax**=**ax1**)**  ax1**.**set\_yscale**(**'log'**)**  ax1**.**set\_xlabel**(**"Parameters"**)**  ax1**.**set\_ylabel**(**"EE"**)** |

