# Supplementary Material for: Soundscape Perception Indices (SPI)

Andrew Mitchell[a,*], Francesco Aletta[a]

[a] *University College London, Institute for Environmental Design and Engineering, Central House, 14 Upper Woburn Place, London, WC1H 0NN*

**Table of contents**



## 1. Supplementary Material

### *1.1. Multi-objective Optimization to Derive an SPI target*

To set up the optimisation task, we first need to express the parameter space and any constraints. Since our goal is to identify an optimised soundscape target distribution, the parameters we will search over are:

- $\xi = (\xi_x, \xi_y)$, $-1 \leq \xi \leq 1$
- $\Omega = \begin{pmatrix} var(x) & cov(x,y) \\ cov(y,x) & var(y) \end{pmatrix}$
  - $0 \leq var() \leq 1$
  - $-1 \leq cov() \leq 1$
  - $\Omega$ must be symmetric and positive definite
- $\alpha = (\alpha_x, \alpha_y)$, $-5 \leq \alpha \leq 5$
- $-1 \leq x, y \leq 1$ In `pymoo`, each objective function is supposed to be minimized. Therefore, we need to convert both SPI and r() to minimize problems.
- min $-r(ranks_{quality}, ranks_{target})$

*Corresponding author
*Email addresses:* `andrew.mitchell.18@ucl.ac.uk` (Andrew Mitchell), `f.aletta@ucl.ac.uk` (Francesco Aletta)

- min $-mean(SPI_{target}(X_i))$

The final objective function is:

- $f_1 = -r(ranks_{quality}, ranks_{target})$
- $f_2 = -mean(SPI_{target}(X_i))$

So our variables to optimize are:

- $-1 \leq \xi_x \leq 1$
- $-1 \leq \xi_y \leq 1$
- $0 \leq var(x) \leq 1$
- $0 \leq var(y) \leq 1$
- $-1 \leq cov(x, y) \leq 1$
- $-5 \leq \alpha_x \leq 5$
- $-5 \leq \alpha_y \leq 5$

Constraint: - $\Omega$ must be symmetric and positive definite - `np.linalg.eigvals(omega) > 0`

We then define the objective functions based on the two goals given above. For each step in the algorithm with a given trial set of parameters, a target distribution will be produced, the SPI for each test location assessed according to the protocol described in Section~**??**, and the resulting set of SPI scores and ranking will be scored using the objective functions. Goal (1) is assessed by calculating the Spearman rank correlation between the *a priori* ranking and the SPI ranking:

```
import warnings
from pathlib import Path

import numpy as np
import pandas as pd
import soundscapy as sspy
from soundscapy.surveys.survey_utils import LANGUAGE_ANGLES, PAQ_IDS

import optimize_target as ot
from MultiSkewNorm import MultiSkewNorm

warnings.filterwarnings("ignore")
```

```
# Load latest ISD dataset
# data = sspy.isd.load_zenodo()
# Load latest ISD dataset

data = sspy.isd.load()
data, excl_data = sspy.isd.validate(data)
data = data.query("Language != 'cmn'")

# Exclude RegentsParkJapan outliers
# excl_id = list(data.query(
    # "LocationID == 'RegentsParkJapan'"
    # ).query("ISOEventful > 0.72 | ISOEventful < -0.5").index)
```

```
# Excluded RegentsParkFields outliers
# excl_id = excl_id + list(data.query(
    # "LocationID == 'RegentsParkFields' and ISOPleasant < 0").index) # Helicopters
excl_id = [652, 706, 548, 550, 551, 553, 569, 580, 609, 618, 623, 636, 643]
data.drop(excl_id, inplace=True)
data
```

| | LocationID | SessionID | GroupID | RecordID | start_time | end_time | latit |
|---|---|---|---|---|---|---|---|
| 0 | CarloV | CarloV2 | 2CV12 | 1434 | 2019-05-16 18:46:00 | 2019-05-16 18:56:00 | 37.1 |
| 1 | CarloV | CarloV2 | 2CV12 | 1435 | 2019-05-16 18:46:00 | 2019-05-16 18:56:00 | 37.1 |
| 2 | CarloV | CarloV2 | 2CV13 | 1430 | 2019-05-16 19:02:00 | 2019-05-16 19:12:00 | 37.1 |
| 3 | CarloV | CarloV2 | 2CV13 | 1431 | 2019-05-16 19:02:00 | 2019-05-16 19:12:00 | 37.1 |
| 4 | CarloV | CarloV2 | 2CV13 | 1432 | 2019-05-16 19:02:00 | 2019-05-16 19:12:00 | 37.1 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 1693 | Noorderplantsoen | Noorderplantsoen1 | NP161 | 61 | 2020-03-11 12:42:00 | 2020-03-11 12:55:00 | NaN |
| 1694 | Noorderplantsoen | Noorderplantsoen1 | NP162 | 63 | 2020-03-11 12:39:00 | 2020-03-11 13:00:00 | NaN |
| 1695 | Noorderplantsoen | Noorderplantsoen1 | NP162 | 62 | 2020-03-11 12:54:00 | 2020-03-11 12:58:00 | NaN |
| 1696 | Noorderplantsoen | Noorderplantsoen1 | NP162 | 64 | 2020-03-11 12:56:00 | 2020-03-11 12:59:00 | NaN |
| 1697 | Noorderplantsoen | Noorderplantsoen1 | NP163 | 70 | 2020-03-11 23:08:00 | 2020-03-11 23:18:00 | NaN |

### *1.2. Calculate ISOPleasant and ISOEventful coordinates*

Here we use the adjusted angles from Aletta et al. (2024) for each language included.

```
for i, row in data.iterrows():
    lang = row["Language"]
    angles = LANGUAGE_ANGLES[lang]
    iso_pl, iso_ev = (
        sspy.surveys.processing._adj_iso_pl(row[PAQ_IDS], angles, scale=4),
        sspy.surveys.processing._adj_iso_ev(row[PAQ_IDS], angles, scale=4),
    )
    data.loc[i, "ISOPleasant"] = iso_pl
    data.loc[i, "ISOEventful"] = iso_ev
```

```
# Separate out parks and non-parks

parks = [
    "RegentsParkFields",
    "RegentsParkJapan",
    "Noorderplantsoen",
    "StPaulsCross",
    "MiradorSanNicolas",
    "RussellSq",
    "Noorderplantsoen",
    "MonumentoGaribaldi",
    "CampoPrincipe",
]

not_parks = [
```

```
    "MarchmontGarden",
    "PancrasLock",
    "TateModern",
    "PlazaBibRambla",
    "SanMarco",
    "StPaulsRow",
    "CarloV",
    "CamdenTown",
    "EustonTap",
    "TorringtonSq",
]

park_data = data.query("LocationID in @parks")
not_park_data = data.query("LocationID in @not_parks")

rank_on = "sss01"
```

```
# Creating a somewhat arbitrary ranking of parks
park_quality = pd.DataFrame(
    park_data.groupby("LocationID")[rank_on].mean().sort_values(ascending=False)
)
park_quality["Rank"] = range(1, len(park_quality) + 1)
park_quality
```

| LocationID | sss01 | Rank |
|---|---|---|
| RegentsParkJapan | 4.617978 | 1 |
| RegentsParkFields | 4.467290 | 2 |
| CampoPrincipe | 4.345455 | 3 |
| MonumentoGaribaldi | 4.156250 | 4 |
| RussellSq | 4.020548 | 5 |
| MiradorSanNicolas | 3.964286 | 6 |
| StPaulsCross | 3.803030 | 7 |
| Noorderplantsoen | 2.412371 | 8 |

### *1.3. `pymoo` Multi-objective Optimization*

Defining the optimization problem:

- max $r(ranks_{quality}, ranks_{target})$
- max $mean(SPI_{target}(X_i))$

where $r$ is the rank correlation coefficient, $ranks_{quality}$ and $ranks_{target}$ are the ranks of the quality and target values, and $SPI_{target}(X_i)$ is the SPI for a given target on the data for the $i$-th location. Therefore we are trying to achieve the best correlation between the desired ranking and the ranking produced by $SPI_{target}$ *and* to achieve the highest mean $SPI_{target}$.

$ranks_{quality}$ is pre-defined. $ranks_{target}$ is calculated by sorting the target values and assigning ranks to them. $SPI_{target}$ is calculated for each location and target.

`target_success(target, pre_ranks, data)`

`target` = `MultiSkewNorm`($\xi$, $\Omega$, $\alpha$) parameters

- $\xi = (\xi_x, \xi_y)$, $-1 \leq \xi \leq 1$
- $\Omega = \begin{pmatrix} var(x) & cov(x,y) \\ cov(y,x) & var(y) \end{pmatrix}$
  - $0 \leq var() \leq 1$
  - $-1 \leq cov() \leq 1$
  - $\Omega$ must be symmetric and positive definite
- $\alpha = (\alpha_x, \alpha_y)$, $-5 \leq \alpha \leq 5$
- $-1 \leq x, y \leq 1$ In `pymoo`, each objective function is supposed to be minimized. Therefore, we need to convert both SPI and r() to minimize problems.
- min $-r(ranks_{quality}, ranks_{target})$
- min $-mean(SPI_{target}(X_i))$

The final objective function is:

- $f_1 = -r(ranks_{quality}, ranks_{target})$
- $f_2 = -mean(SPI_{target}(X_i))$

So our variables to optimize are:

- $-1 \leq \xi_x \leq 1$
- $-1 \leq \xi_y \leq 1$
- $0 \leq var(x) \leq 1$
- $0 \leq var(y) \leq 1$
- $-1 \leq cov(x,y) \leq 1$
- $-5 \leq \alpha_x \leq 5$
- $-5 \leq \alpha_y \leq 5$

Constraint: - $\Omega$ must be symmetric and positive definite - `np.linalg.eigvals(omega) > 0`

### *1.3.1. Problem Definition*

```
import pathos
from pymoo.core.callback import Callback
from pymoo.core.problem import ElementwiseProblem, StarmapParallelization
from pymoo.visualization.scatter import Scatter
from pyrecorder.recorder import Recorder
from pyrecorder.writers.streamer import Streamer
from pyrecorder.writers.video import Video
from pymoo.decomposition.asf import ASF
```

```
class MyProblem(ElementwiseProblem):
    def __init__(self, data, ranking, **kwargs):
        super().__init__(
            n_var=7,
            n_obj=2,
            n_constr=0,
            xl=np.array([-1, -1, 0, 0, -1, -50, -50]),
            xu=np.array([1, 1, 0.5, 0.5, 1, 50, 50]),
            n_eq_constr=1,
            elementwise_evaluation=True,
            **kwargs,
        )

        self.data = data
        self.ranking = ranking

    def _evaluate(self, X, out, *args, **kwargs):
        h = 1 - int(
            np.all(np.linalg.eigvals(np.array([[X[2], X[4]], [X[4], X[3]]])) > 0)
        )
        out["H"] = h
        if h != 0:
            out["F"] = np.column_stack([0, 0])
            return
        else:
            tgt = MultiSkewNorm()
            tgt.define_dp(
                np.array([X[0], X[1]]),
                np.array([[X[2], X[4]], [X[4], X[3]]]),
                np.array([X[5], X[6]]),
            )
            tgt.sample()
            r, wspi, spi_ranks, target = ot.target_success(tgt, self.ranking, self.data)

            f1 = -r[0]
            f2 = -wspi / 100

            out["F"] = np.column_stack([f1, f2])


class VideoCallback(Callback):
    def __init__(self) -> None:
        super().__init__()
        self.rec = Recorder(Streamer(sleep=0.1))

    def notify(self, algorithm):
        sc = Scatter(
            title="Gen %s" % algorithm.n_gen,
            labels=["spearman", "WSPI"],
        )
        sc.add(algorithm.pop.get("F"))
```

```
        sc.do()
        self.rec.record()
```

```
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.operators.crossover.sbx import SBX
from pymoo.operators.mutation.pm import PM
from pymoo.operators.sampling.rnd import FloatRandomSampling
from pymoo.optimize import minimize
from pymoo.termination.default import DefaultMultiObjectiveTermination

algorithm = NSGA2(
    pop_size=150,
    n_offsprings=100,
    sampling=FloatRandomSampling(),
    crossover=SBX(),
    mutation=PM(),
    eliminate_duplicates=True,
    # callback=VideoCallback()
)

termination = DefaultMultiObjectiveTermination(n_max_gen=100)
```

### *1.3.2. Park Quality optimization*

```
# initialize the thread pool and create the runner
mp = pathos.helpers.mp
n_process = 12
pool = mp.Pool(n_process)
runner = StarmapParallelization(pool.starmap)

park_problem = MyProblem(
    data=park_data, ranking=park_quality.sort_index()["Rank"], elementwise_runner=runner
)

park_res = minimize(
    park_problem, algorithm, termination, seed=42, save_history=True, verbose=True
)

pool.close()

park_F = park_res.F
park_X = park_res.X
```

```
==========================================================================================
n_gen  |  n_eval  | n_nds  |     cv_min    |     cv_avg    |      eps      |   indicator
==========================================================================================
     1 |      150 |      2 |  0.000000E+00 |  0.7599240000 |             - |             -
     2 |      250 |      3 |  0.000000E+00 |  0.4999500000 |  0.2727272727 |         ideal
     3 |      350 |      3 |  0.000000E+00 |  0.0199980000 |  0.2142857143 |         ideal
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 450 | 4 | 0.000000E+00 | 0.000000E+00 | 0.1620792395 | ideal |
| 5 | 550 | 7 | 0.000000E+00 | 0.000000E+00 | 0.0558552677 | f |
| 6 | 650 | 4 | 0.000000E+00 | 0.000000E+00 | 0.8461538462 | nadir |
| 7 | 750 | 5 | 0.000000E+00 | 0.000000E+00 | 0.0222136817 | f |
| 8 | 850 | 6 | 0.000000E+00 | 0.000000E+00 | 0.1101638372 | ideal |
| 9 | 950 | 6 | 0.000000E+00 | 0.000000E+00 | 0.0657188723 | f |
| 10 | 1050 | 5 | 0.000000E+00 | 0.000000E+00 | 1.0047062320 | nadir |
| 11 | 1150 | 6 | 0.000000E+00 | 0.000000E+00 | 0.0691690492 | ideal |
| 12 | 1250 | 8 | 0.000000E+00 | 0.000000E+00 | 0.0503174951 | ideal |
| 13 | 1350 | 7 | 0.000000E+00 | 0.000000E+00 | 0.0127793462 | f |
| 14 | 1450 | 9 | 0.000000E+00 | 0.000000E+00 | 0.0424372038 | ideal |
| 15 | 1550 | 8 | 0.000000E+00 | 0.000000E+00 | 0.0292206309 | f |
| 16 | 1650 | 9 | 0.000000E+00 | 0.000000E+00 | 0.0785192627 | ideal |
| 17 | 1750 | 9 | 0.000000E+00 | 0.000000E+00 | 0.000000E+00 | f |
| 18 | 1850 | 9 | 0.000000E+00 | 0.000000E+00 | 0.0185516738 | f |
| 19 | 1950 | 11 | 0.000000E+00 | 0.000000E+00 | 0.0370370370 | ideal |
| 20 | 2050 | 12 | 0.000000E+00 | 0.000000E+00 | 0.0083471711 | f |
| 21 | 2150 | 12 | 0.000000E+00 | 0.000000E+00 | 0.0033142727 | f |
| 22 | 2250 | 15 | 0.000000E+00 | 0.000000E+00 | 0.0093007525 | ideal |
| 23 | 2350 | 13 | 0.000000E+00 | 0.000000E+00 | 0.0034666441 | f |
| 24 | 2450 | 12 | 0.000000E+00 | 0.000000E+00 | 0.0098196917 | f |
| 25 | 2550 | 12 | 0.000000E+00 | 0.000000E+00 | 0.000000E+00 | f |
| 26 | 2650 | 13 | 0.000000E+00 | 0.000000E+00 | 0.0034300107 | f |
| 27 | 2750 | 14 | 0.000000E+00 | 0.000000E+00 | 0.0053483509 | f |
| 28 | 2850 | 14 | 0.000000E+00 | 0.000000E+00 | 0.000000E+00 | f |
| 29 | 2950 | 14 | 0.000000E+00 | 0.000000E+00 | 0.000000E+00 | f |
| 30 | 3050 | 14 | 0.000000E+00 | 0.000000E+00 | 0.000000E+00 | f |
| 31 | 3150 | 14 | 0.000000E+00 | 0.000000E+00 | 0.0006784315 | f |
| 32 | 3250 | 12 | 0.000000E+00 | 0.000000E+00 | 0.0082062343 | f |
| 33 | 3350 | 12 | 0.000000E+00 | 0.000000E+00 | 0.0021983597 | f |
| 34 | 3450 | 14 | 0.000000E+00 | 0.000000E+00 | 0.0154559077 | ideal |
| 35 | 3550 | 14 | 0.000000E+00 | 0.000000E+00 | 0.0008284906 | f |
| 36 | 3650 | 13 | 0.000000E+00 | 0.000000E+00 | 0.0065038186 | f |
| 37 | 3750 | 13 | 0.000000E+00 | 0.000000E+00 | 0.0047267260 | f |
| 38 | 3850 | 13 | 0.000000E+00 | 0.000000E+00 | 0.000000E+00 | f |
| 39 | 3950 | 13 | 0.000000E+00 | 0.000000E+00 | 0.0011035360 | f |
| 40 | 4050 | 13 | 0.000000E+00 | 0.000000E+00 | 0.0016008743 | f |
| 41 | 4150 | 12 | 0.000000E+00 | 0.000000E+00 | 0.0167253281 | ideal |
| 42 | 4250 | 13 | 0.000000E+00 | 0.000000E+00 | 0.0052208080 | f |
| 43 | 4350 | 13 | 0.000000E+00 | 0.000000E+00 | 0.000000E+00 | f |
| 44 | 4450 | 12 | 0.000000E+00 | 0.000000E+00 | 0.0030262140 | f |
| 45 | 4550 | 12 | 0.000000E+00 | 0.000000E+00 | 0.000000E+00 | f |
| 46 | 4650 | 13 | 0.000000E+00 | 0.000000E+00 | 0.0029423569 | f |
| 47 | 4750 | 13 | 0.000000E+00 | 0.000000E+00 | 0.0001362881 | f |
| 48 | 4850 | 11 | 0.000000E+00 | 0.000000E+00 | 0.0168434981 | f |
| 49 | 4950 | 12 | 0.000000E+00 | 0.000000E+00 | 0.0065122582 | f |
| 50 | 5050 | 11 | 0.000000E+00 | 0.000000E+00 | 0.0044925500 | f |
| 51 | 5150 | 11 | 0.000000E+00 | 0.000000E+00 | 0.0071445969 | ideal |
| 52 | 5250 | 12 | 0.000000E+00 | 0.000000E+00 | 0.0074595724 | f |
| 53 | 5350 | 12 | 0.000000E+00 | 0.000000E+00 | 0.0026178835 | f |
| 54 | 5450 | 12 | 0.000000E+00 | 0.000000E+00 | 0.0016927330 | f |
| 55 | 5550 | 12 | 0.000000E+00 | 0.000000E+00 | 0.0016927330 | f |

```
    56 |      5650 |      12 |  0.000000E+00 |  0.000000E+00 |  0.0016927330 |             f
    57 |      5750 |      12 |  0.000000E+00 |  0.000000E+00 |  0.0016927330 |             f
    58 |      5850 |      12 |  0.000000E+00 |  0.000000E+00 |  0.0016927330 |             f
    59 |      5950 |      12 |  0.000000E+00 |  0.000000E+00 |  0.0016927330 |             f
    60 |      6050 |      12 |  0.000000E+00 |  0.000000E+00 |  0.0030163061 |             f
    61 |      6150 |      12 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    62 |      6250 |      12 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    63 |      6350 |      12 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    64 |      6450 |      12 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    65 |      6550 |      12 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    66 |      6650 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0035306991 |             f
    67 |      6750 |      13 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    68 |      6850 |      13 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    69 |      6950 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0000145448 |             f
    70 |      7050 |      14 |  0.000000E+00 |  0.000000E+00 |  0.0026667316 |             f
    71 |      7150 |      14 |  0.000000E+00 |  0.000000E+00 |  0.0019718538 |             f
    72 |      7250 |      14 |  0.000000E+00 |  0.000000E+00 |  0.0027648400 |             f
    73 |      7350 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0054222167 |             f
    74 |      7450 |      13 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    75 |      7550 |      12 |  0.000000E+00 |  0.000000E+00 |  0.0060101024 |             f
    76 |      7650 |      12 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    77 |      7750 |      12 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    78 |      7850 |      12 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    79 |      7950 |      12 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    80 |      8050 |      11 |  0.000000E+00 |  0.000000E+00 |  0.0010485449 |             f
    81 |      8150 |      11 |  0.000000E+00 |  0.000000E+00 |  0.0010485449 |             f
    82 |      8250 |      11 |  0.000000E+00 |  0.000000E+00 |  0.0010485449 |             f
    83 |      8350 |      11 |  0.000000E+00 |  0.000000E+00 |  0.0010485449 |             f
    84 |      8450 |      11 |  0.000000E+00 |  0.000000E+00 |  0.0010485449 |             f
    85 |      8550 |      11 |  0.000000E+00 |  0.000000E+00 |  0.0010485449 |             f
    86 |      8650 |      11 |  0.000000E+00 |  0.000000E+00 |  0.0010485449 |             f
    87 |      8750 |      11 |  0.000000E+00 |  0.000000E+00 |  0.0010485449 |             f
    88 |      8850 |      12 |  0.000000E+00 |  0.000000E+00 |  0.0021136652 |             f
    89 |      8950 |      12 |  0.000000E+00 |  0.000000E+00 |  0.0021136652 |             f
    90 |      9050 |      12 |  0.000000E+00 |  0.000000E+00 |  0.0021136652 |             f
    91 |      9150 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0047031047 |             f
    92 |      9250 |      13 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    93 |      9350 |      13 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    94 |      9450 |      13 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00 |             f
    95 |      9550 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0005236113 |             f
    96 |      9650 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0007334714 |             f
    97 |      9750 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0007334714 |             f
    98 |      9850 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0007604831 |             f
    99 |      9950 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0007604831 |             f
   100 |     10050 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0007604831 |             f
```

```python
weights = np.array([0.5, 0.5])
decomp = ASF()

with Recorder(Video("park_nsga2.mp4")) as rec:
    for entry in park_res.history:
```

```
        # Get the approximated ideal and nadir points
        approx_ideal = park_res.F.min(axis=0)
        approx_nadir = park_res.F.max(axis=0)

        # Normalize the obtained front
        nF = (entry.pop.get("F") - approx_ideal) / (approx_nadir - approx_ideal)
        park_I = decomp(nF, weights).argmin()
        sc = Scatter(title="Generation: %s" % entry.n_gen)
        sc.add(entry.pop.get("F"))
        sc.add(entry.pop.get("F")[park_I], color="red", s=30)
        sc.do()
        rec.record()


with Recorder(Video("park_nsga2_sspy.mp4")) as rec:
    for entry in park_res.history:
        # Get the approximated ideal and nadir points
        approx_ideal = park_res.F.min(axis=0)
        approx_nadir = park_res.F.max(axis=0)

        # Normalize the obtained front
        nF = (entry.pop.get("F") - approx_ideal) / (approx_nadir - approx_ideal)
        park_I = decomp(nF, weights).argmin()
        park_X = entry.pop.get("X")[park_I]
        park_tgt = MultiSkewNorm()
        park_tgt.define_dp(
            np.array([park_X[0], park_X[1]]),
            np.array([[park_X[2], park_X[4]], [park_X[4], park_X[3]]]),
            np.array([park_X[5], park_X[6]]),
        )
        park_tgt.sample()
        ss = sspy.plotting.density_plot(
            data=pd.DataFrame({
                "ISOPleasant": park_tgt.sample_data[:,0],
                "ISOEventful": park_tgt.sample_data[:,1],
            }),
            # x=park_tgt.sample_data[:,0],
            # y=park_tgt.sample_data[:,1],
            title="Generation: %s" % entry.n_gen,
        )
        rec.record()
```

```
import matplotlib.pyplot as plt

park_X = park_res.X[park_I]
park_tgt = MultiSkewNorm()
park_tgt.define_dp(
    np.array([park_X[0], park_X[1]]),
    np.array([[park_X[2], park_X[4]], [park_X[4], park_X[3]]]),
    np.array([park_X[5], park_X[6]]),
)
```

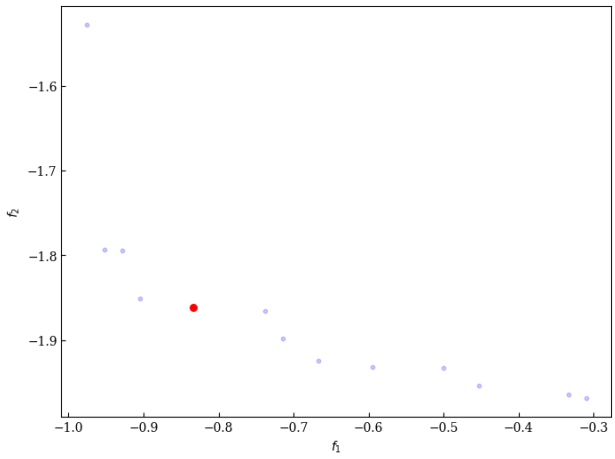
```
from pymoo.decomposition.asf import ASF

# Get the approximated ideal and nadir points
approx_ideal = park_res.F.min(axis=0)
approx_nadir = park_res.F.max(axis=0)

# Normalize the obtained front
nF = (park_res.F - approx_ideal) / (approx_nadir - approx_ideal)

weights = np.array([0.48, 0.52])
decomp = ASF()

park_I = decomp(nF, weights).argmin()
# print("Best regarding decomposition: Point %s - %s" % (park_I, park_res.F[park_I]))
```

Figure 1

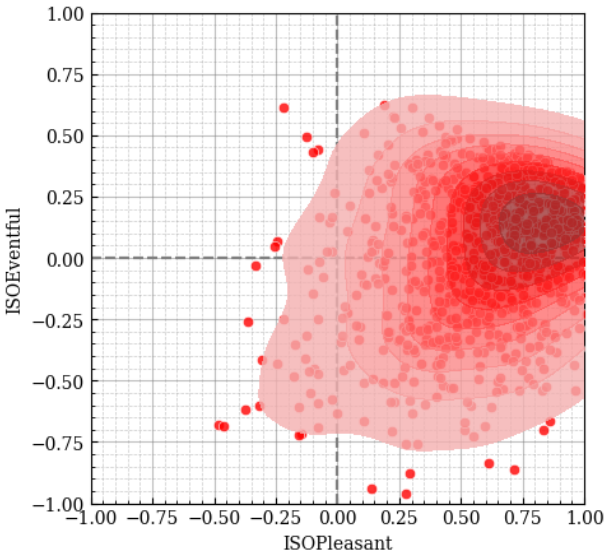
```
park_tgt.sample()

# print(park_tgt.summary())

plot = Scatter()
plot.add(park_res.F, color="blue", alpha=0.2, s=10)
plot.add(park_res.F[park_I], color="red", s=30)
plot.do()
# plot.apply(lambda ax: ax.arrow(0, 0, 0.5, 0.5, color='black',
#                                head_width=0.01, head_length=0.01, alpha=0.4))
plot.show()
plt.show()



# park_tgt.sspy_plot()
df = pd.DataFrame(park_tgt.sample_data, columns=["ISOPleasant", "ISOEventful"])
sspy.plotting.density_plot(
    df, color='red', title=None
)
plt.show()
```

```
# print(park_tgt.summary())
```

## References




(a) Multi-objective optimization Pareto front. The selected solution is indicated in red.




(b) SCM distribution of the derived target distribution.

Figure 2: NSGA-II optimization to learn the MSN parameters which produce the Park ranking.