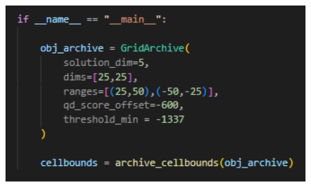
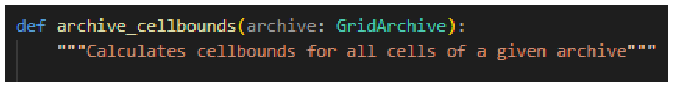
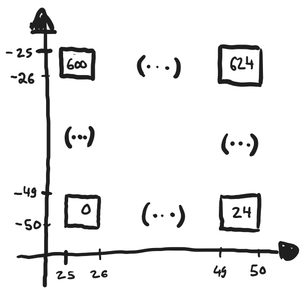
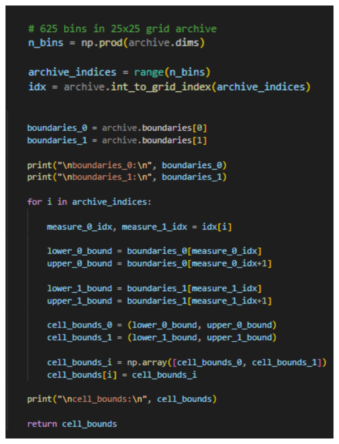


```python
if __name__ == "__main__":

    obj_archive = GridArchive(
        solution_dim=5,
        dims=[25,25],
        ranges=[(25,50),(-50,-25)],
        qd_score_offset=-600,
        threshold_min = -1337
    )

    cellbounds = archive_cellbounds(obj_archive)
```

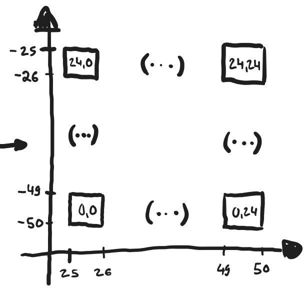


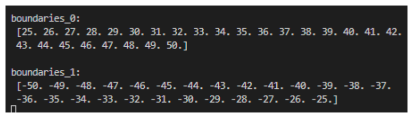
```python
def archive_cellbounds(archive: GridArchive):
    """Calculates cellbounds for all cells of a given archive"""
```



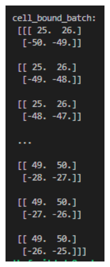


```python
# 625 bins in 25x25 grid archive
n_bins = np.prod(archive.dims)

archive_indices = range(n_bins)
idx = archive.int_to_grid_index(archive_indices)


boundaries_0 = archive.boundaries[0]
boundaries_1 = archive.boundaries[1]

print("\nboundaries_0:\n", boundaries_0)
print("\nboundaries_1:\n", boundaries_1)

for i in archive_indices:

    measure_0_idx, measure_1_idx = idx[i]

    lower_0_bound = boundaries_0[measure_0_idx]
    upper_0_bound = boundaries_0[measure_0_idx+1]

    lower_1_bound = boundaries_1[measure_1_idx]
    upper_1_bound = boundaries_1[measure_1_idx+1]

    cell_bounds_0 = (lower_0_bound, upper_0_bound)
    cell_bounds_1 = (lower_1_bound, upper_1_bound)

    cell_bounds_i = np.array([cell_bounds_0, cell_bounds_1])
    cell_bounds[i] = cell_bounds_i

print("\ncell_bounds:\n", cell_bounds)

return cell_bounds
```





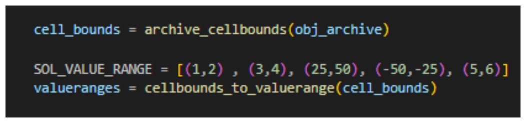
```
boundaries_0:
 [25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36. 37. 38. 39. 40. 41. 42.
 43. 44. 45. 46. 47. 48. 49. 50.]

boundaries_1:
 [-50. -49. -48. -47. -46. -45. -44. -43. -42. -41. -40. -39. -38. -37.
 -36. -35. -34. -33. -32. -31. -30. -29. -28. -27. -26. -25.]
```



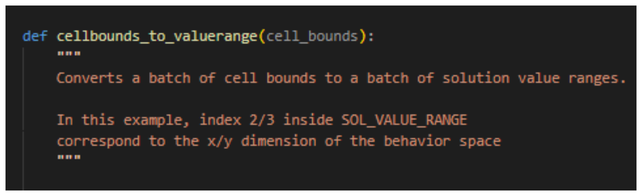
```
cell_bound_batch:
 [[[ 25.  26.]
  [-50. -49.]]

 [[ 25.  26.]
  [-49. -48.]]

 [[ 25.  26.]
  [-48. -47.]]

 ...

 [[ 49.  50.]
  [-28. -27.]]

 [[ 49.  50.]
  [-27. -26.]]

 [[ 49.  50.]
  [-26. -25.]]]
```

0, [0,0]

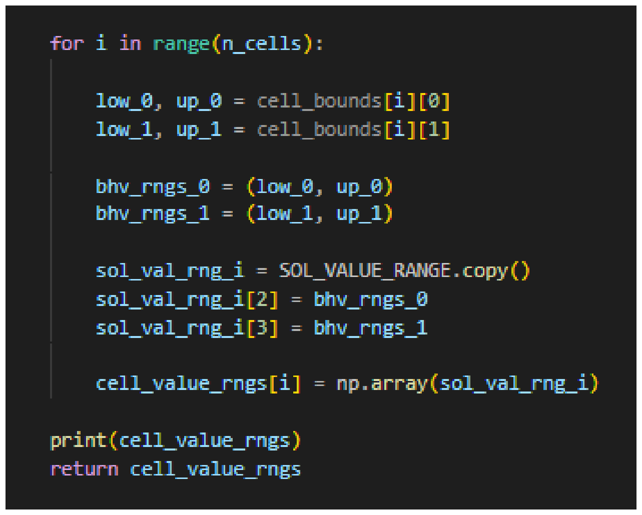
624, [24,24]



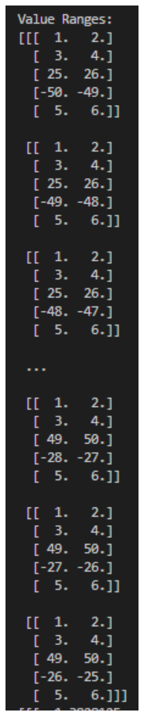
```python
cell_bounds = archive_cellbounds(obj_archive)

SOL_VALUE_RANGE = [(1,2) , (3,4), (25,50), (-50,-25), (5,6)]
valueranges = cellbounds_to_valuerange(cell_bounds)
```



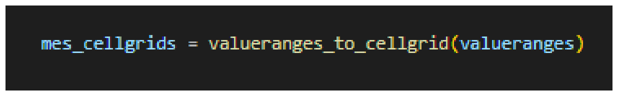
```python
def cellbounds_to_valuerange(cell_bounds):
    """
    Converts a batch of cell bounds to a batch of solution value ranges.

    In this example, index 2/3 inside SOL_VALUE_RANGE
    correspond to the x/y dimension of the behavior space
    """
```



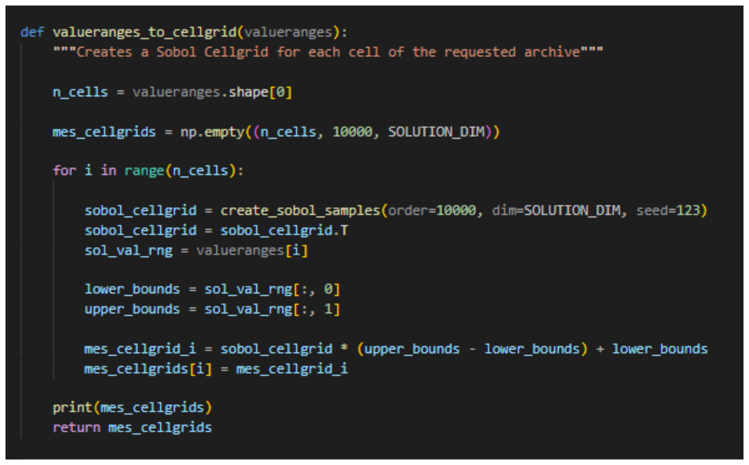
```python
for i in range(n_cells):

    low_0, up_0 = cell_bounds[i][0]
    low_1, up_1 = cell_bounds[i][1]

    bhv_rngs_0 = (low_0, up_0)
    bhv_rngs_1 = (low_1, up_1)

    sol_val_rng_i = SOL_VALUE_RANGE.copy()
    sol_val_rng_i[2] = bhv_rngs_0
    sol_val_rng_i[3] = bhv_rngs_1

    cell_value_rngs[i] = np.array(sol_val_rng_i)

print(cell_value_rngs)
return cell_value_rngs
```



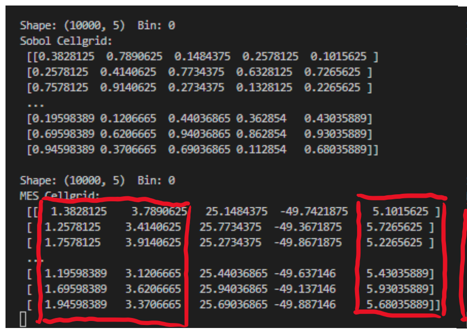
```
Value Ranges:
[[[  1.   2.]
  [  3.   4.]
  [ 25.  26.]
  [-50. -49.]
  [  5.   6.]]

 [[  1.   2.]
  [  3.   4.]
  [ 25.  26.]
  [-49. -48.]
  [  5.   6.]]

 [[  1.   2.]
  [  3.   4.]
  [ 25.  26.]
  [-48. -47.]
  [  5.   6.]]

 ...

 [[  1.   2.]
  [  3.   4.]
  [ 49.  50.]
  [-28. -27.]
  [  5.   6.]]

 [[  1.   2.]
  [  3.   4.]
  [ 49.  50.]
  [-27. -26.]
  [  5.   6.]]

 [[  1.   2.]
  [  3.   4.]
  [ 49.  50.]
  [-26. -25.]
  [  5.   6.]]]
```



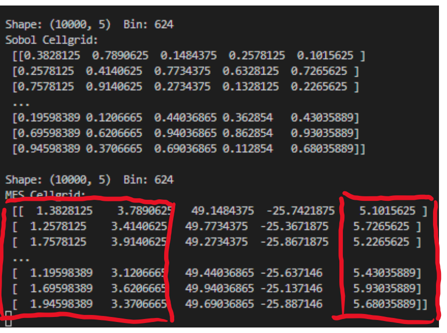
```python
mes_cellgrids = valueranges_to_cellgrid(valueranges)
```



```python
def valueranges_to_cellgrid(valueranges):
    """Creates a Sobol Cellgrid for each cell of the requested archive"""

    n_cells = valueranges.shape[0]

    mes_cellgrids = np.empty((n_cells, 10000, SOLUTION_DIM))

    for i in range(n_cells):

        sobol_cellgrid = create_sobol_samples(order=10000, dim=SOLUTION_DIM, seed=123)
        sobol_cellgrid = sobol_cellgrid.T
        sol_val_rng = valueranges[i]

        lower_bounds = sol_val_rng[:, 0]
        upper_bounds = sol_val_rng[:, 1]

        mes_cellgrid_i = sobol_cellgrid * (upper_bounds - lower_bounds) + lower_bounds
        mes_cellgrids[i] = mes_cellgrid_i

    print(mes_cellgrids)
    return mes_cellgrids
```



```
Shape: (10000, 5)  Bin: 0
Sobol Cellgrid:
 [[0.3828125  0.7890625  0.1484375  0.2578125  0.1015625 ]
 [0.2578125  0.4140625  0.7734375  0.6328125  0.7265625 ]
 [0.7578125  0.9140625  0.2734375  0.1328125  0.2265625 ]
 ...
 [0.19598389 0.1206665  0.44036865 0.362854   0.43035889]
 [0.69598389 0.6206665  0.94036865 0.862854   0.93035889]
 [0.94598389 0.3706665  0.69036865 0.112854   0.68035889]]

Shape: (10000, 5)  Bin: 0
MES Cellgrid:
 [[  1.3828125    3.7890625   25.1484375  -49.7421875    5.1015625 ]
 [  1.2578125    3.4140625   25.7734375  -49.3671875    5.7265625 ]
 [  1.7578125    3.9140625   25.2734375  -49.8671875    5.2265625 ]
 ...
 [  1.19598389   3.1206665   25.44036865 -49.637146     5.43035889]
 [  1.69598389   3.6206665   25.94036865 -49.137146     5.93035889]
 [  1.94598389   3.3706665   25.69036865 -49.887146     5.68035889]]
```



```
Shape: (10000, 5)  Bin: 624
Sobol Cellgrid:
 [[0.3828125  0.7890625  0.1484375  0.2578125  0.1015625 ]
 [0.2578125  0.4140625  0.7734375  0.6328125  0.7265625 ]
 [0.7578125  0.9140625  0.2734375  0.1328125  0.2265625 ]
 ...
 [0.19598389 0.1206665  0.44036865 0.362854   0.43035889]
 [0.69598389 0.6206665  0.94036865 0.862854   0.93035889]
 [0.94598389 0.3706665  0.69036865 0.112854   0.68035889]]

Shape: (10000, 5)  Bin: 624
MES Cellgrid:
 [[  1.3828125    3.7890625   49.1484375  -25.7421875    5.1015625 ]
 [  1.2578125    3.4140625   49.7734375  -25.3671875    5.7265625 ]
 [  1.7578125    3.9140625   49.2734375  -25.8671875    5.2265625 ]
 ...
 [  1.19598389   3.1206665   49.44036865 -25.637146     5.43035889]
 [  1.69598389   3.6206665   49.94036865 -25.137146     5.93035889]
 [  1.94598389   3.3706665   49.69036865 -25.887146     5.68035889]]
```