This README file is divided into two parts which explain the code files and how they can be used to generate the figures:

I - PARAMETERS USED TO GENERATE EACH FIGURE
II - CODE FILES AND DEPENDENCIES
III - Notes on randomization

Part I contains the exact plotter files and parameters used to generate each figure/table. To recreate the figures/tables, simply paste the parameters given into the “””PARAMETERS””” section of the plotter file.

The code files are described in **Part II**, and are divided into the following two categories:
- *Plotter files* are run to generate the tables and figures. Parameters to get different plots are placed at the top of each plotter file, and their values can be adjusted. After choosing the appropriate parameters, run the whole .py file to get the plots.
- *Helper files* contain functions that are used by the plotter files (or higher-level helper files), and need not be touched.

Part III contains details on the randomization used in the code, should the user want to understand the randomization processes used in the code or to change the random seed.

All code run using the following installations:
```
Python 3.9.13
NumPy 1.21.5
SciPy 1.9.1
```

I - PARAMETERS USED TO GENERATE EACH FIGURE

FIGURE 1 (ENTROPY)
entropyplot.py and *bregmanplot.py* in the entropy_viz folder.

Parameters for bregmanplot.py (coordinates of reference point in Figure 1):
```
ref = np.array([0.3*0.7, 0.4*0.7, 0.5])
```

FIGURE 2 (SCMWU REGRET)
SCMWU_plot.py in the OSCO folder.

```
"""Run parameters"""
dims = [
    (0, [], [2, 3, 4, 5, 6]),
    (5, [], [5]),
    (0, [3], [5]),
    (3, [2,3], [2,3])
]  # Dimension (3-tuple): (m, n, d), where
    # m int: dimension of R^m direct-sum component;
    # n list;
    # d list.
Ts = [
    10**5
] # Time Horizon (Integer)
doubling = True # Set to TRUE to use doubling trick, FALSE for fixed stepsize (Bool)
```

```
"""Multiplicity parameters"""
many_SCMWU = 100 # Number of SCMWU trajectories (Integer >= 1)
runs = 1 # Number of runs, i.e., number of plots.

"""Randomization parameters"""
seed = 1234
```

FIGURE 3 (SCMWU-BALL REGRET PLOTS)
SCMWUball_plot.py in the OSCO folder.

```
"""Run parameters"""
ds = [
    2, 3, 5, 10
] # list of dimensions of ball to learn over.
Ts = [
    10**4
] # Time Horizon (Integer)
doubling = True # Set to TRUE to use doubling trick, FALSE for fixed stepsize (Bool)
OGD_compare = False # Set to TRUE if want to plot OGD (Bool)
plot_bound = True # (Bool)

"""Multiplicity parameters"""
many_SCMWU = 100 # Number of SCMWU trajectories per plot (Integer >= 1)
runs = 1 # Number of runs, i.e., number of plots.

"""Randomization parameters"""
seed = 1234
```

FIGURE 4 (SCMWU-BALL VS OGD)
SCMWUball_plot.py in the OSCO folder.

```
"""Run parameters"""
ds = [
    10
] # list of dimensions of ball to learn over.
Ts = [
    10**4
] # Time Horizon (Integer)
doubling = False # Set to TRUE to use doubling trick, FALSE for fixed stepsize (Bool)

"""Multiplicity parameters"""
many_SCMWU = 1 # Number of SCMWU trajectories per plot (Integer >= 1)
runs = 10 # Number of runs, i.e., number of plots.

"""Randomization parameters"""
seed = 1234
```

TABLES 1 & 2 (SVM GAME PERFORMANCE)
SVM_game_tabulate.py in the OSCO folder.

```
# Dataset parameters
n = 10**3 # Number of data points per run
ds = [
    2, 3,
    5, 10
]# Dimension of data points
margin = 0.1 # Linear classification margin (guaranteed for dataset)

# Run parameters
compute_time_horizon = False #If False, use T. If True, use margin_error to compute
time_needed.
if compute_time_horizon:
    margin_error = 0.05 # Margin to guarantee
    T = SVM_game.time_needed(margin_error, n)
else:
    Ts = [
    10**2, 10**3
    ] # Time Horizon (Integer)

# Number of runs
runs = 10**2

# Randomization/seed parameters
randseed = 1234 # (1234 was used for data, unless otherwise stated)
```

FIGURES 5 - 7 (SVM GAME VISUALIZATION)
SVM_game_run.py in the OSCO folder.
(Set time horizon T below to 10**3, 10**4, and 10**5 for Figures 5, 6, and 7 respectively.)

```
# Dataset parameters
n = 10**3 # Number of data points per run
d = 2 # Dimension of data points
margin = 0.1 # Linear classification margin (guaranteed for dataset)

# Run parameters
compute_time_horizon = False #If False, use T. If True, use margin_error to compute
time_needed.
if compute_time_horizon:
    margin_error = 0.05 # Margin to guarantee
    T = SVMgame.time_needed(margin_error, n)
else:
    T = 10**3 # Time Horizon (Integer)
print("T =", T)

# Number of runs
runs = 10**1

# Randomization/seed parameters
randseed = 1234 # (1234 was used for data, unless otherwise stated)
```

II - CODE FILES AND DEPENDENCIES

entropy_viz folder: Plotter files
entropyplot.py and *bregmanplot.py* contain the code for plotting the entropy and Bregman divergence over the trace-one slice of the 2-dimensional second-order cone (Figure 1). Parameters for *bregmanplot.py* (choosing the reference point for the Bregman divergence) are at the top of the file, and can be adjusted. Ruby the whole .py file to get the plots.
- Uses *SOC.py* for EJA operations relevant to the SOC.

entropy_viz folder: Helper files
SOC.py contains EJA operations relevant to the SOC.
* Used in both *entropyplot.py* and *bregmanplot.py*.

OSCO folder: Plotter files
SCMWU_plot.py contains the code for plotting regret of SCMWU for the task of online linear optimization over the trace-1 slice of a symmetric cone (Figure 2). Parameters to get the different plots are at the top of the file, and can be adjusted. After choosing the appropriate parameters, run the whole .py file to get the plots.
- Uses *SCMWU.py* for SCMWU algorithm.
- Uses *SC_losses.py* for loss vector generation.

SCMWUball_plot.py contains the code for plotting regret of SCMWU_ball for the task of online linear optimization over the unit ball, with possible comparison to OGD (Figures 3 & 4). Parameters to get the different plots are at the top of the file, and can be adjusted. After choosing the appropriate parameters, run the whole .py file to get the plots.
- Uses *SCMWU_ball.py* for SCMWU_ball algorithm.
- Uses *SC_losses.py* for loss vector generation.

SVM_game_tabulate.py contains code for generating tables of the average and worst-case margins achieved by running the SCMWU against MWU in the SVM game (Tables 1 & 2). Parameters to get the different plots are at the top of the file, and can be adjusted. After choosing the appropriate parameters, run the whole .py file to get the plots.
- Uses *SVMgame.py* to run SVM game.
- Uses *LinClass_datasets.py* for dataset generation.

SVM_game_plot.py contains code for plotting visualizations of the classifiers achieved by running the SCMWU against MWU in the SVM game for data points of dimension 2 (Figures 5 - 7). Parameters to get the different plots are at the top of the file, and can be adjusted. After choosing the appropriate parameters, run the whole .py file to get the plots.
- Uses *SVMgame.py* to run SVM game.
- Uses *LinClass_datasets.py* for dataset generation.

OSCO folder: Helper files
Symmetric cone operations
SC.py contains EJA operations over symmetric cones with direct-sum structure.
* Used in *SCMWU.py*.
- Uses *SC_NO.py*, *SC_PSD.py*, and *SC_SOC.py* for operations over the primitive symmetric

cones.

SC_NO.py contains EJA operations over the nonnegative orthant.
* Used in *SC.py*.

SC_PSD.py contains EJA operations over the PSD cone.
* Used in *SC.py*.

SC_SOC.py contains EJA operations over the second-order cone.
* Used in *SC.py*.

SCMWU algorithm
SCMWU.py contains the SCMWU algorithm.
* Used in *SCMWU_plot.py*.
- Uses *SC.py* for EJA operations over symmetric cones.

SCMWU-ball algorithm
SCMWUball.py contains the SCMWU-ball algorithm.
* Used in *SCMWUball_plot.py, SVMgame.py*.
- Uses *helpers.py* for vector (Euclidean) length function.

SVM game
SVMgame.py contains the code for running the SVM game.
* Used in *SVM_game_tabulate.py, SVM_game_plot.py*.
- Uses *SCMWU.py* for SCMWU algorithm (learning over ball).
- Uses *MWU.py* for MWU algorithm (learning over probability simplex).
- Uses *helpers.py* for vector length and normalization functions.

MWU.py contains the MWU (Multiplicative Weights Update) algorithm for learning over the probability simplex.
* Used in *SVMgame.py*.
- Uses *helpers.py* for an exponential function that avoids numerical error when taking in very negative exponents.

Loss and Dataset generation
SC_losses.py generates the losses for online optimization over symmetric cones and the ball.
* Used in *SCMWU_plot.py, SCMWUball_plot.py*. (Figures 2-4.)

LinClass_datasets.py generates the datasets for the SVM game.
* Used in *SVM_game_tabulate.py, SVM_game_plot.py*. (Tables 1-2, Figures 5-7.)

Others
helpers.py contains miscellaneous helper functions, including an exponential function that avoids numerical error when taking in very negative exponents.
* Used in *SCMWUball.py, SVMgame.py*, and *MWU.py*.

III - Notes on randomization

As the algorithms implemented are all deterministic, all the randomization happening in the

code occurs in the randomization of losses. All of these are confined to the *SC_losses.py* helper file, and pseudocode can be found in the comments there. (distr='ball' is used in all function calls.)

The random seed has been set to the value 1234 in all plotter files to be able to recreate the same plots. To change the seed, simply find the seed in the """PARAMETERS""" section of the file and change it.