Nesterov

December 1, 2017

1 Nesterov Method

```
In [1]: from sys import path
        path.append('../src')
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.neighbors import KernelDensity
        from dimReduct.SpectralMethods import DiffusionMap
        from dimReduct.Nesterov import cNesterov
        from sklearn.neighbors import KernelDensity
        from scipy.io import savemat
        from scipy.spatial.distance import pdist
        from scipy.stats import truncnorm
```

```
In [2]: % matplotlib inline
```

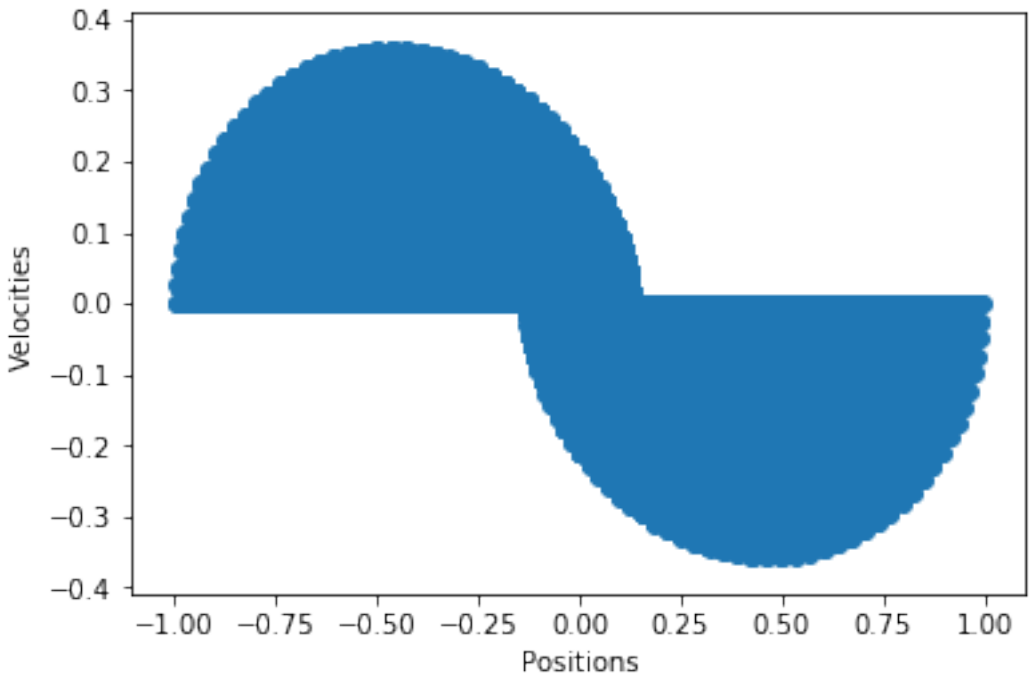
2 Simulation

Choose y = 0.5 * x ** 2 as the function to be minimized.

```
In [3]: minpos, maxpos, points = -1, 1, 1000
        tsteps = np.linspace(0, 20, 200)
        f = lambda x: 0.5 * x ** 2
        #f = lambda x: -1 * (x ** 2 * (x - 4) * (x + 7))
        gradf = lambda x: x
        #gradf = lambda x: -1 * (4 * x ** 3 + 9 * x ** 2 - 56 * x)
        positions, velocities = cNesterov(truncnorm.rvs(minpos, maxpos, size=points), f, gradf
```

Plot the orbits of the continuous Nesterov system for times (0, 20) with a truncated normal distribution of points over the interval (-1, 1).

```
In [4]: fig, ax = plt.subplots()
        ax.scatter(positions, velocities)
        ax.set_xlabel('Positions'); ax.set_ylabel('Velocities');
```



3 KDE

Perform kernel density estimation on the Nesterov orbits to get a density that can be used to check if the orbits obey the heat equation.

```
In [5]: density = np.empty((positions.shape[0], positions.shape[1]))
        samples = np.linspace(minpos, maxpos, points)
        for i in range(positions.shape[1]):
            kde = KernelDensity(bandwidth=(200 * (max(positions[:, i]) - min(positions[:, i]))
            kde.fit(positions[:, i][:, np.newaxis])
            density[:, i] = np.exp(kde.score_samples(samples[:, np.newaxis]))
        #     delta = pdist(positions[:, i][:, np.newaxis]).min()
        #     samples[:, i] = np.hstack((np.linspace(minpos, min(positions[:, i]) - 21 * delta,
        #                                np.array([min(positions[:, i]) - (20 - j) * delta for
        #                                np.sort(positions[:, i]),
        #                                np.array([max(positions[:, i]) + (j + 1) * delta for j
        #                                np.linspace(max(positions[:, i]) + 21 * delta, maxpos,
        #     samples[:, i].sort()
```

4 Numerical Derivatives

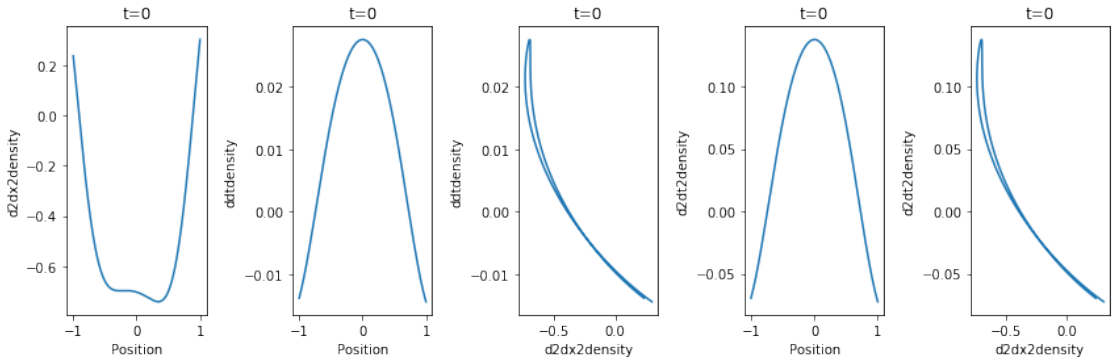
Compute the numerical derivatives needed to check if either the heat or wave equation is satisfied by using central differences.

```
In [6]: d2dx2density = np.empty((tsteps.shape[0] - 2, density.shape[0] - 2))
        ddtdensity = np.empty((tsteps.shape[0] - 2, density.shape[0] - 2))
        d2dt2density = np.empty((tsteps.shape[0] - 2, density.shape[0] - 2))
        for i in range(tsteps.shape[0] - 2):
            for j in range(density.shape[0] - 2):
                d2dx2density[i, j] = (density[j + 2, i + 1] - 2 * density[j + 1, i + 1] + densi
                ddtdensity[i, j] = (density[j + 1, i + 2] - density[j + 1, i]) / (tsteps[i + 1
                d2dt2density[i, j] = (density[j + 1, i + 2] - 2 * density[j + 1, i + 1] + densi
```

4.1 Numerical Derivatives at One Time

Plot the results of the numerical derivatives at all positions for a fixed time.

```
In [7]: fig, ax = plt.subplots(1, 5, figsize=(12, 4))
        time = 0
        ax[0].plot(samples[1:999], d2dx2density[time, :]); ax[0].set_xlabel('Position'); ax[0]
        ax[1].plot(samples[1:999], ddtdensity[time, :]); ax[1].set_xlabel('Position'); ax[1].se
        ax[2].plot(d2dx2density[0, :], ddtdensity[time, :]); ax[2].set_xlabel('d2dx2density');
        ax[3].plot(samples[1:999], d2dt2density[time, :]); ax[3].set_xlabel('Position'); ax[3]
        ax[4].plot(d2dx2density[time, :], d2dt2density[time, :]); ax[4].set_xlabel('d2dx2densit
        fig.tight_layout()
```



Save the data to export to matlab for movie generation.

```
In [8]: savemat('NesterovData', {'positions' : positions, 'velocities' : velocities, 'densitySa
                                 'd2dx2density' : d2dx2density, 'ddtdensity' : ddtdensity, 'd2dt
```