Sheet 2: Maximum Likelihood Estimation

In this exercise sheet, we will look at various properties of maximum-likelihood estimation, and how to find maximum-likelihood parameters.

ML vs. James Stein Estimator (15 P)

Let $X_1, \ldots, X_n \in \mathbb{R}^d$ be independent draws from a multivariate Gaussian distribution with mean vector μ and covariance matrix $\Sigma = \sigma^2 I$. It can be shown that the maximum-likelihood estimator of the mean parameter μ is the empirical mean given by:

$$\hat{\mu}_{\mathsf{ML}} = \frac{1}{N} \sum_{i=1}^{N} X_i$$

It was once believed that the maximum-likelihood estimator was the most accurate possible (i.e. the one with the smallest Euclidean distance from the true mean). However, it was later demonstrated that the following estimator

$$\hat{\mu}_{JS} = \Big(1 - \frac{(d-2)\cdot\sigma^2}{n \cdot \|\mu_{\mathsf{ML}}\|^2}\Big)\hat{\mu}_{\mathsf{ML}}$$

(a shrinked version of the maximum-likelihood estimator towards the origin) has actually a smaller distance from the true mean when $d \geq 3$. This however assumes knowledge of the variance of the distribution for which the mean is estimated. This estimator is called the James-Stein estimator. While the proof is a bit involved, this fact can be easily demonstrated empirically through simulation. This is the object of this exercise.

The code below draws ten 50-dimensional points from a normal distribution with mean vector $\mu = (1, \ldots, 1)$ and covariance $\Sigma = I$.

```
In [1]: def getdata(seed):

            n = 10               # data points
            d = 50               # dimensionality of data
            m = numpy.ones([d])  # true mean
            s = 1.0              # true standard deviation

            rstate = numpy.random.mtrand.RandomState(seed)
            X = rstate.normal(0,1,[n,d])*s+m

            return X,m,s
```

The following function computes the maximum likelihood estimator from a sample of the data assumed to be generated by a Gaussian distribution:

```
In [2]: def ML(X):
            return X.mean(axis=0)
```

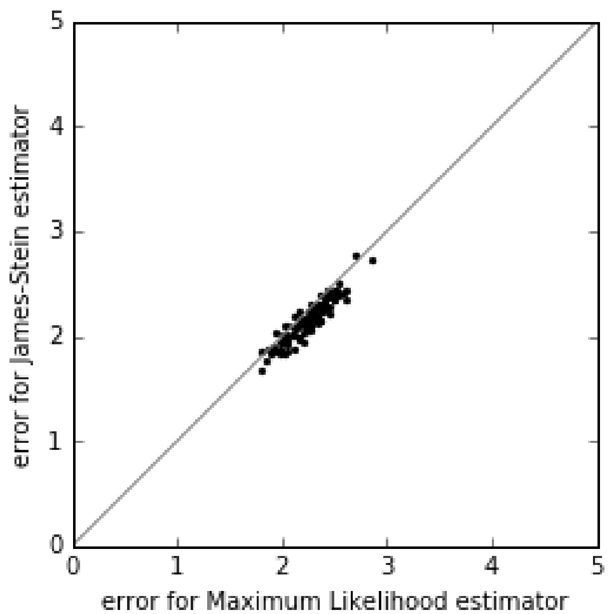
- **Based on the ML estimator function, write a function that receives as input the data $(X_i)_{i=1}^n$ and the (known) variance σ^2 of the generating distribution, and computes the James-Stein estimator**

```
In [3]: def JS(X,s):
            # REPLACE BY YOUR CODE
            import solution
            m_JS = solution.JS(X,s)
            ###
            return m_JS
```

We would like to compute the error of the maximum likelihood estimator and the James-Stein estimator for 100 different samples (where each sample consists of 10 draws generated by the function `getdata` with a different random seed). Here, for reproducibility, we use seeds from 0 to 99. The error should be measured as the Euclidean distance between the true mean vector and the estimated mean vector.

- **Compute the maximum-likelihood and James-Stein estimations.**
- **Measure the error of these estimations.**
- **Build a scatter plot comparing these errors for different samples.**

```
In [4]: %matplotlib inline
        ### REPLACE BY YOUR CODE
        import solution
        solution.compare_ML_JS()
        ###
```



Parameters of a mixture of exponentials (15 P)

We consider the following "mixture of exponentials" distribution supported on $\mathbb{R}^+$, that we use to generate data, but whose parameters α and β are unknown.

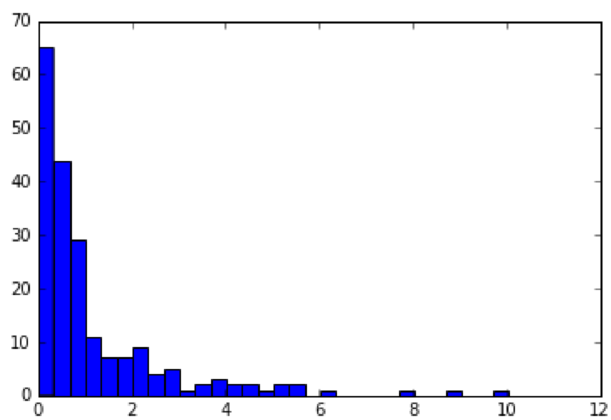
$$p(x;\alpha,\beta) = 0.5 \cdot \left[\alpha e^{-\alpha x} + \beta e^{-\beta x}\right]$$

A dataset $\mathcal{D} = x_1, \ldots, x_N$ with $N = 200$ has been generated from that distribution. It is given below and plotted as a histogram.

```
In [5]: D=[ 0.74,  0.20,  0.56,  0.05,  0.67,  0.41,  0.74,  4.63,  0.59,  0.39,
            0.71,  0.17,  5.34,  0.33,  0.01,  1.11,  0.60,  0.41,  0.65,  1.97,
            0.19,  0.80,  0.04,  0.48,  0.54,  0.59,  0.31,  1.40,  0.63,  0.38,
            0.36,  0.02,  0.68,  0.72,  0.84,  0.30,  0.01,  1.37,  0.89,  0.10,
            0.21,  0.68,  0.14,  0.10,  0.11,  0.01,  0.09,  0.50,  0.34,  0.30,
            1.22, 10.05,  0.19,  0.04,  0.13,  1.53,  2.28,  1.76,  0.03,  0.31,
```

```
    0.37,  0.50,  0.05,  0.30,  0.53,  0.63,  4.20,  0.86,  0.29,  1.98,
    1.27,  0.35,  0.43,  0.35,  0.75,  0.25,  1.15,  1.65,  0.82,  0.37,
    2.55,  2.75,  3.06,  0.97,  2.65,  8.97,  0.04,  2.98,  0.36,  0.01,
    0.85,  0.90,  0.09,  0.01,  0.82,  2.30,  2.09,  0.29,  0.16,  2.12,
    5.28,  0.27,  0.15,  1.02,  0.51,  0.02,  1.72,  1.35,  0.51,  0.27,
    1.05,  2.24,  3.93,  0.62,  3.38,  0.56,  0.49,  2.84,  0.27,  0.12,
    3.99,  0.16,  0.09,  3.61,  0.54,  0.08,  0.31,  1.38,  0.63,  0.61,
    0.21,  0.13,  2.28,  2.61,  4.60,  0.02,  0.34,  0.15,  0.07,  2.44,
    0.86,  0.73,  2.01,  0.26,  0.72,  1.56,  0.09,  0.97,  0.24,  0.92,
    1.05,  0.71,  1.28,  3.79,  1.32,  0.17,  0.39,  2.82,  0.12,  2.06,
    2.04,  0.00,  1.94,  0.27,  0.91,  0.36,  0.92,  5.69,  0.33,  0.69,
    1.00,  2.19,  0.01,  0.08,  1.16,  0.31,  0.83,  0.41,  1.27,  0.08,
    4.69,  0.65,  0.43,  0.10,  2.92,  0.06,  6.21,  0.90,  0.00,  0.52,
    0.65,  0.26,  1.94,  0.37,  0.50,  5.66,  4.24,  0.40,  0.39,  7.89]
```

```
%matplotlib inline
from matplotlib import pyplot as plt
plt.hist(D,bins=30)
plt.show()
```



For this dataset, the log-likelihood function is given by

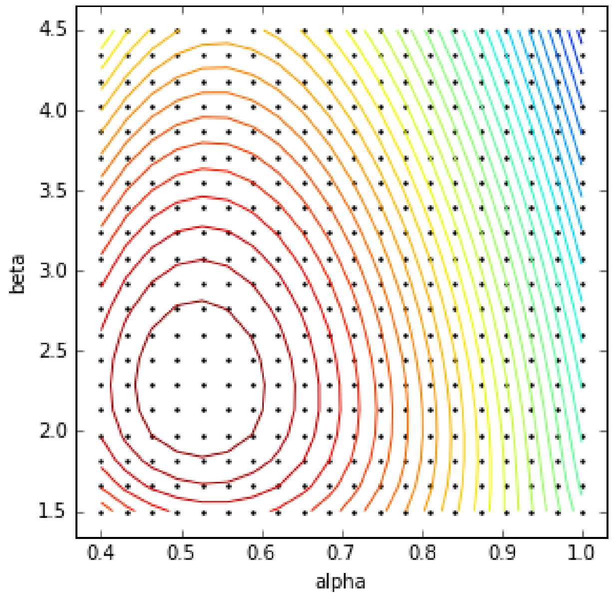
$$\ell(\alpha, \beta) = \log \prod_{i=1}^{N} p(x_i; \alpha, \beta) = \sum_{i=1}^{N} \log(e^{-\alpha x_i} + \beta e^{-\beta x_i}) - \log(2)$$

Unfortunately, it is difficult to extract the parameters α, β analytically by solving directly the equation $\nabla \ell = 0$. Instead, we will analyze the function over a grid of parameters α, β. We know a priori that parameters α and β are in the intervals $[0.4, 1.0]$ and $[1.5, 4.5]$ respectively.

- **Build a grid on this limited domain and evaluate log-likelihood at each point of the grid.**
- **Plot the log-likelihood function as a contour plot, and superpose the grid to it.**

Highest log-likelihood values (i.e. most probable parameters) should appear in red, and lowest values should be plotted in blue. Two adjacent lines of the contour plot should represent a log-likelihood difference of 1.0. In your code, favor numpy array operations over Python loops.

```
In [6]: ### REPLACE BY YOUR CODE
        import solution
        solution.s2a(D)
        ###
```



Gradient-Based Optimization (10 P)

As an alternative to computing the log-likelihood for a whole grid, we would like to find the optimal parameters α, β by gradient-based optimization. The partial derivatives of the log-likelihood function are given by:

$$\frac{\partial \ell(\alpha,\beta)}{\partial \alpha} = \sum_{i=1}^{N} \frac{e^{-\alpha x_i}(1-\alpha x_i)}{\alpha e^{-\alpha x_i} + \beta e^{-\beta x_i}}$$

$$\frac{\partial \ell(\alpha,\beta)}{\partial \beta} = \sum_{i=1}^{N} \frac{e^{-\beta x_i}(1-\beta x_i)}{\alpha e^{-\alpha x_i} + \beta e^{-\beta x_i}}$$

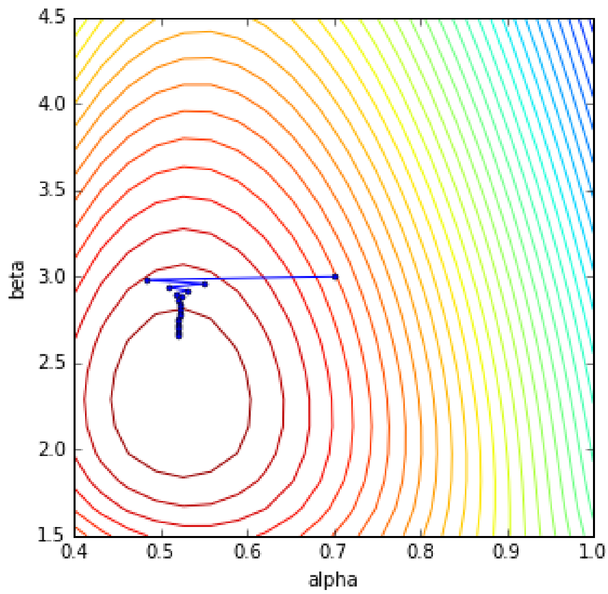
A gradient ascent step of the log-likelihood function takes the form

$$\begin{pmatrix}\alpha \\ \beta\end{pmatrix} \leftarrow \begin{pmatrix}\alpha \\ \beta\end{pmatrix} + \gamma \nabla_{\alpha,\beta} \ell(\alpha,\beta)$$

where γ is a learning rate to be defined. We start with initial parameters $\alpha = 0.7$ and $\beta = 3.0$.

- **Implement the gradient ascent procedure.**
- **Run the gradient ascent with parameter $\gamma = 0.005$.**
- **Plot the trajectory of the gradient ascent in superposition to the contour plot of the previous exercise.**

```
In [7]: ### REPLACE BY YOUR CODE
        import solution
        solution.s2b(D)
        ###
```



As it can be seen, the optimization procedure does not converge in reasonable time and seems to oscillate.

- **Explain the problem(s) with this approach. Propose a simple improvement of the optimization technique and apply it.**

[REPLACE BY YOUR EXPLANATION + PROPOSITION]

In [8]:
```
### REPLACE BY YOUR CODE
import solution
solution.s2c(D)
###
```