The objective of this lab is for you to program in Matlab several nonparametric methods for density estimation, classification and regression, apply them to some datasets and observe their behavior. The TA will first demonstrate the results of the algorithms on a toy dataset, and then you will program them, replicate those results, and further explore the datasets with the algorithms. You can use the textbook, lecture notes and your own notes.

I Datasets

Construct your own toy datasets to visualize the result easily. Take the input instances $\{\mathbf{x}_n\}_{n=1}^N$ in $\mathbb{R}$ and the labels $\{y_n\}_{n=1}^N$ in $\{1,\dots,K\}$ (classification) or $\mathbb{R}$ (regression). You can also take $\mathbf{x} \in \mathbb{R}^2$ and use surface or contour plots.

II Implementing and using nonparametric methods

Implement the following methods:

- *Histogram* with origin $x_0 \in \mathbb{R}$ and bin width $h > 0$, for density estimation. Plot the resulting density estimate $p(x)$ using a bar chart. Consider the following questions:
  - How does the histogram change if you change x_0? How does it change if you change h?
- *Gaussian kernel density estimate* with bin width $h > 0$, for:
  - Density estimation: plot the resulting density estimate $p(x)$ as a continuous curve in $\mathbb{R}$.
  - Classification: plot the resulting posterior distribution estimate $p(k|x)$ for each class $k = 1,\dots,K$ as a continuous curve in $\mathbb{R}$, using a different color for each class. Plot the data points colored according to the predicted label $\arg\max_{k=1,\dots,K} p(k|x)$.
  - Regression: plot the resulting regression function $g(x)$ as a continuous curve in $\mathbb{R}$.

  Consider the following questions:
  - How does the result change if you change h? How does the estimated density $p(x)$ and the regression function $g(x)$ behave for $h \to 0$ and for $h \to \infty$?
  - How well does the estimated density $p(x)$ or regression function $g(x)$ approximate the true one?
  - How does the regression function $g(x)$ behave near discontinuities in the true function $f(x)$, or in regions $x \in \mathbb{R}$ that have no data points?

Further things to try:

- Repeat for other 1D datasets having different distribution, noise levels, etc.
- Use kernels other than the Gaussian, e.g. uniform or k-nearest-neighbor.

The following Matlab functions will be useful (among others): `hist bar randn rand find linspace scatter`.

Practical advice:

- Many operations in machine learning algorithms involve vectors and matrices. Try to program these using vector and matrix operations in Matlab ("vectorized code") rather than loops, because 1) the code will be shorter and more readable (closer to the pseudocode), 2) it will be faster (because Matlab is an interpreted language), and 3) you will save effort and avoid bugs. Example: a matrix-vector product `y = A*x` instead of a double loop $y_i = \sum_{j=1}^n a_{ij}x_j$ for $i = 1,\dots,n$.
- Machine learning algorithms can have a high time or space complexity, so to get a result in a few seconds you may need to run them on small datasets. You can do this by selecting a random sample of a given dataset.
- Machine learning algorithms often are randomized. Likewise, toy datasets are usually generated randomly. To make sure you can generate the exact dataset multiple times and run an algorithm and get the same result every time, fix the seed of the pseudorandom number generator. In Matlab: `rng(1778);` where 1778 is the seed. You can also save a toy dataset for later use.
- Matlab tips:
  - To suppress extra line feeds: `format compact`.
  - To get more decimals: `format long`.
  - To compare two matrices or vectors (by finding the largest difference): `max(abs(A(:)-B(:)))`.
  - To avoid distorted plots: `daspect([1 1 1])`.
  - To plot grayscale images with values in $[0,1]$: `colormap(gray(256)); imagesc(I,[0 1]);`
    To plot images with negative and positive values: `colormap(parula(256)); imagesc(I);`