Assignment 8

1. **KDE in various dimensions.**

   Kernel density estimation means estimating a probability density from a collection of samples and an estimation kernel. To be specific (though this is not necessary) choose the basic smoothing kernel to be

   $$\phi(x) = \frac{1}{(2\pi)^{\frac{d}{d}}} e^{-\frac{1}{2}\|x\|_2^2} \; .$$

   Suppose $f(x)$ is a smooth probability density in d dimensions and that X_j for $j = 1, \cdots, n$ is a collection of independent samples of f. The density estimate is

   $$\widehat{f}(x) = \frac{1}{n} \sum_{j=1}^{n} \phi_\epsilon(x - X_j) \; .$$

   The *lengthscale* ϵ smoother is

   $$\phi_\epsilon(x) = \frac{1}{\epsilon^d} \phi(\tfrac{1}{\epsilon} x) \; .$$

   The value of density estimate $\widehat{f}(x)$ is random because it depends on the random samples X_j. Denote the expected value of the density estimator by

   $$\overline{f}(x) = \mathrm{E}\Big[\widetilde{f}(x) \Big] \; .$$

   (a) Show that if f is smooth, then the bias of the KDE satisfies

   $$\overline{f}(x) - f(x) = O(\epsilon^2) \; .$$

   (b) Show that the variance of the KDE satisfies (for small ϵ)

   $$\mathrm{var}(\widetilde{f}(x)) \approx \frac{C}{\epsilon^p \, n} \; .$$

   Find the exponent p, which depends on n.

   (c) The total error of any estimator is the sum of the bias and the statistical error. Suppose that the statistical error is equal to the standard deviation of the estimator. Suppose that $\epsilon = n^{-\alpha}$ and find the exponent α that minimizes the total error (has the best power of n). What is that power of n? What does that say about the accuracy of KDE estimation if d is not small?

2. **KDE experiment.**

   Use the Brownian bridge code form Assignment 7, but change notation so the number of components is L and the Brownian bridge components are U_j for $1 \leq j \leq L$. The PDF for $U = (U_1, \cdots, U_L)$ is the same

$$p(u) = \frac{1}{Z} e^{-\frac{1}{2}\left(u_1^2 + \cdots + u_L^2 \right)}$$

   Create n independent samples of the pair (X, Y), where

$$X = \sum_{j=1}^{L} U_j$$
$$Y = \max_{1 \leq j \leq L} U_j$$

   We want to estimate and visualize the PDF $f(x, y)$ using kernel density estimation. Hints:

   - Make a color contour plot of the estimated density $\widehat{f}(x, y)$. Do this by evaluating $\widehat{f}$ on a uniform grid to create an array of values to pass to the plot routine.
   - Rescale x and y before doing the density estimate. That means, choose length scales λ_x and λ_y and replace X_j by X_j/λ_x and Y_j by Y_j/λ_y. One way would be to use the standard deviation of the samples X_j and Y_j. Another way would be to use the range of values, such as $\max X_j - \min X_j$.
   - If you want to understand the importance of scaling, try $L = 100$ without scaling. The range of X will be different from the range of Y, so isotropic smoothing with the isotropic Gaussian will not work well.
   - Experiment with various n (number of samples) and ϵ (smoothing length) relations to see which work well.
   - You can debug you code by taking (X, Y) to be from a bivariate Gaussian with $\sigma_X^2 \neq \sigma_Y^2$ and $\sigma_{XY} = 0$ (the covariance). The contour lines should be elliptical. If you scale using standard deviation, the contour lines should be circular.
   - As always, you will lose points unless you make some comments on the results. What did you learn from this?

3. **The Runge phenomenon.**

   Write a function that takes as input n interpolation points and values $x = (x_1 \cdots, x_n)$ and values $f_1, \cdots, x_n)$ and returns a tuple of two objects, the first being an array of the polynomial coefficients $a = (a_0, \cdots, a_{n-1})$ with $f(x) = a_0 + a_1 x + \cdots a_{n-1} x^{n-1}$ having $f(x_j) = f_j$ for $j = 1, \cdots, n$.

The second item in the tuple should be the condition number of the Vandermonde matrix

$$V = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & & x_2^{n-1} \\ & \vdots & & & \vdots \\ 1 & x_n & x_n^2 & & x_n^{n-1} \end{pmatrix} .$$

The function should form V and use the linear solver in `numpy` to find a. You can get the condition number using the SVD of V or using a `numpy` routine. The `numpy` routine either uses the SVD or some heuristic that can fail.

Try your interpolation function on the target "Runge" function

$$r(x) = \frac{1}{1 + 25x^2} .$$

Take the x_k to be uniformly spaced with $x_1 = -1$ and $x_n = 1$. The interpolation values should be $f_k = r(x_k)$. Make one plot with r and three well chosen f curves. For moderate n the fit is OK. For very large n the fit shows the *Runge phenomenon*. The phenomenon is visible but less severe for intermediate n.

Repeat the experiment with similar looking target functions such as (Look at many functions, not necessarily the ones listed here. Hand in one or two that you think are interesting and different. If your Runge code is completely automated, all you have to do is type in another function r and push `enter`.)

$$r(x) = \cos(\pi x) + 1$$

$$r(x) = e^{-\frac{x^2}{2\sigma^2}} \qquad \text{(play with } \sigma\text{)}$$

$$r(x) = \frac{1}{\left(1 + \frac{1}{y^2}x^2\right)^p} \qquad \text{(play with } y \text{ and } p\text{)}$$

The last example has poles at $\pm iy$. The Runge example has poles at $\pm i\frac{1}{5}$, which are close to the real axis.

Make a plot of the condition number $\kappa(V)$ as a function of n. This is, of course, the same for all the examples. The difference is that this one uses many n values rather than just three carefully chosen values. Choose axes (linear, semi-log, log-log) to try to see how V depends on n when it gets large.

(*Extra credit, if you have time and intertest*) Repeat some of these experiments using the Chebychev points. For those, take n angles θ_j uniformly spaced in $[-\pi, \pi]$ and take $x_k = \cos(\theta_k)$. Comment on the differences in the results.

4. **Spline approximation.**

   Do numerical experiments to determine the order of accuracy of cubic b-spline interpolation. Start with a function $r(x)$, evaluate r at n equally spaced points in $[0, 1]$ with separation Δx. Call the spline interpolant $f(x)$. Estimate the maximum error by testing many points, also uniformly spaced but with a much finer spacing:

$$M(\Delta x) = \max_{0 \le x \le 1} |f(x) - r(x)| \approx \max_{k} |f(kh) - r(kh)| \ .$$

   It should suffice to take $h = .1\Delta x$, but maybe try some smaller values to make sure. A smaller value may make the accuracy results cleaner. Make a plot or a table of M as a function of Δx or n and determine the order of accuracy. Use a smooth trial function r (or a few trial functions to see whether the results are stable) that is non-trivial. For example, don't use a low order polynomial. Possible choose one that's harder to interpolate than $\sin(x)$.

   Use the `interpolate` package of `scipy` to find spline interpolants. Make sure to ask for cubic b-splines. Note that the routine (which you should use) `make_splrep` is a constructor that returns an object that can be used to evaluate the spline function.