CMDA 3606 · MATHEMATICAL MODELING II

Problem Set 7 · Solutions

Posted 14 October 2022. Due at 11:59pm on Thursday, 20 October 2022.

Please submit your problem set by uploading to Canvas one PDF file that includes all elements of your solution. Be sure to include your Python code/Jupyter notebook(s).

Basic guidelines: Students may discuss the problems on this assignment, but each student must submit his or her individual write-up and code. (In particular, you *must write up your own individual Python code.*) Students may consult the class notes and other online resources, but *the use of solutions from previous classes is forbidden and will be regarded as a violation of the Honor Code.*

1. [25 points: 3 points each for (a) and (b); 6 points each for (d) and (e); 7 points for (c)]

   Recall that for a square matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, the linear system $\mathbf{Ac} = \mathbf{f}$ has a unique solution $\mathbf{c} \in \mathbb{R}^n$ for any $\mathbf{f} \in \mathbb{R}^n$ provided $\mathcal{N}(\mathbf{A}) = \{\mathbf{0}\}$, where $\mathcal{N}(\mathbf{A})$ denotes the null space of $\mathbf{A}$.

   On the other hand, if there is a nonzero vector $\mathbf{z} \in \mathcal{N}(\mathbf{A})$, then there are two possibilities:

   - If $\mathbf{f} \notin \mathcal{R}(\mathbf{A})$, then there is *no solution* $\mathbf{c}$ to the linear system $\mathbf{Ac} = \mathbf{f}$.
   - If $\mathbf{f} \in \mathcal{R}(\mathbf{A})$, then there are *infinitely many solutions* to the linear system $\mathbf{Ac} = \mathbf{f}$. In particular, $\mathbf{c}_+ = \mathbf{A}^+\mathbf{f} \in \mathcal{R}(\mathbf{A}^T)$ satisfies $\mathbf{Ac}_+ = \mathbf{f}$, as does $\mathbf{c} = \mathbf{c}_+ + \mathbf{z}$ for any $\mathbf{z} \in \mathcal{N}(\mathbf{A})$.

   Keep these facts in mind as you answer the following questions about *polynomial interpolation*.

   Suppose we want to find a quadratic polynomial $q(x) = c_0 + c_1x + c_2x^2$ that matches the value of some function f at the points $x = -1$, $x = 0$, and $x = 1$. That is, we want to find q such that

   $$q(-1) = f(-1), \qquad q(0) = f(0), \qquad q(1) = f(1).$$

   To find the quadratic q, we need to find the three unknown coefficients c_0, c_1, and c_2. We require:

   $$\begin{array}{rcrcrcl} q(-1) = f(-1) & \Longrightarrow & c_0 & +c_1(-1) & +c_2(-1)^2 & = f(-1) \\ q(0) = f(0) & \Longrightarrow & c_0 & +c_1(0) & +c_2(0)^2 & = f(0) \\ q(1) = f(1) & \Longrightarrow & c_0 & +c_1(1) & +c_2(1)^2 & = f(1) \end{array}$$

   These three equations in the three unknowns can be arranged as the linear system $\mathbf{Ac} = \mathbf{f}$:

   $$\begin{bmatrix} 1 & -1 & (-1)^2 \\ 1 & 0 & 0^2 \\ 1 & 1 & 1^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} f(-1) \\ f(0) \\ f(1) \end{bmatrix}.$$

   (a) Suppose we have $f(-1) = -2$, $f(0) = -1$, and $f(1) = 2$. Given these values, solve the above linear system for $\mathbf{c}$, and write down the quadratic $q(x) = c_0 + c_1x + c_2x^2$.

   (b) Suppose we want to replace the condition $q(1) = f(1)$ with the new condition $q'(1) = f'(1)$. Since $q'(x) = c_1 + c_2 \cdot 2x$, we would replace the equation $q(1) = f(1)$ above with

   $$q'(1) = f'(1) \quad \Longrightarrow \quad 0 \ +c_1 \ +c_2 \cdot 2(1) \ = f'(1)$$

   and replace the last row of $\mathbf{A}$ and $\mathbf{f}$ to get the new linear system $\mathbf{Ac} = \mathbf{f}$:

   $$\begin{bmatrix} 1 & -1 & (-1)^2 \\ 1 & 0 & 0^2 \\ 0 & 1 & 2 \cdot 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} f(-1) \\ f(0) \\ f'(1) \end{bmatrix}.$$

Suppose we have $f(-1) = 2$, $f(0) = 1$, and $f'(1) = 2$. Solve this new linear system for $\mathbf{c}$, and write down the quadratic $q(x) = c_0 + c_1 x + c_2 x^2$.

(c) Suppose we wish to construct a polynomial

$$p(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4 + c_5 x^5$$

that matches a function f in the following (rather unusual) manner: $p(-1) = f(-1)$; $p'(-1) = f'(-1)$; $p(0) = f(0)$; $p''(0) = f''(0)$; $p(1) = f(1)$; $p'(1) = f'(1)$.
Explain how the above equations lead to the linear system $\mathbf{Ac} = \mathbf{f}$ of the form:

$$\begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ 0 & 1 & -2 & 3 & -4 & 5 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{bmatrix} f(-1) \\ f'(-1) \\ f(0) \\ f''(0) \\ f(1) \\ f'(1) \end{bmatrix}.$$

(d) What is the null space $\mathcal{N}(\mathbf{A})$ of this 6×6 matrix $\mathbf{A}$?
(You may use Python commands, such as `np.linalg.svd` or `scipy.linalg.null_space`.)
Given your answer, under what conditions will a polynomial p with the desired properties exist? If it exists, will it be unique?

(e) Consider the data: $f(-1) = -1$, $f'(-1) = 0$, $f(0) = 1$, $f''(0) = -2$, $f(1) = 3$, $f'(1) = 4$.

In Python, determine *five* different solutions to $\mathbf{Ac} = \mathbf{f}$ of the form $\mathbf{c} = \mathbf{c}_+ + \mathbf{z}$, where $\mathbf{z} \in \mathcal{N}(\mathbf{A})$. Plot all five solutions on the same plot. The following commands show one way to plot a solution:

```
x = np.linspace(-1.25,1.25)
plt.plot(x, c[0] + c[1]*x + c[2]*(x**2) + c[3]*(x**3) + c[4]*(x**4) + c[5]*(x**5))
```

Solution.

(a) The system can be readily solved (by hand, or with Python) to get

$$\mathbf{c} = \begin{bmatrix} -1 \\ 2 \\ 1 \end{bmatrix}, \qquad \Longrightarrow \qquad c_0 = -1, \quad , \; c_1 = 2, \quad c_2 = 1.$$

These coefficients give the polynomial

$$q(x) = -1 + 2x + x^2.$$

(b) This system can be similarly solved, giving

$$\mathbf{c} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \qquad \Longrightarrow \qquad c_0 = 1, \quad , \; c_1 = 0, \quad c_2 = 1.$$

These coefficients give the polynomial

$$q(x) = 1 + x^2.$$

(c) We seek the coefficients $c_0, \ldots, c_5$ to the polynomial

$$p(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4 + c_5 x^5.$$

Compute derivatives:

$$p'(x) = 0 + c_1 + c_2 \cdot 2x + c_3 \cdot 3x^2 + c_4 \cdot 4x^3 + c_5 \cdot 5x^4$$
$$p'(x) = 0 + 0 + c_2 \cdot 2 + c_3 \cdot 6x + c_4 \cdot 12x^2 + c_5 \cdot 20x^3.$$

Evaluate these polynomials at the six constraints to get six equations in six unknowns:

$$\begin{aligned}
p(-1) &= c_0 + c_1 \cdot (-1) + c_2 \cdot (-1)^2 + c_3 \cdot (-1)^3 + c_4 \cdot (-1)^4 + c_5 \cdot (-1)^5 = f(-1) \\
p'(-1) &= 0 + c_1 + c_2 \cdot 2(-1) + c_3 \cdot 3(-1)^2 + c_4 \cdot 4(-1)^3 + c_5 \cdot 5(-1)^4 = f'(-1) \\
p(0) &= c_0 + c_1 \cdot (0) + c_2 \cdot (0)^2 + c_3 \cdot (0)^3 + c_4 \cdot (0)^4 + c_5 \cdot (0)^5 = f(0) \\
p''(0) &= 0 + 0 + c_2 \cdot 2 + c_3 \cdot 6(0) + c_4 \cdot 12(0) + c_5 \cdot 20(0) = f''(0) \\
p(1) &= c_0 + c_1 \cdot (1) + c_2 \cdot (1)^2 + c_3 \cdot (1)^3 + c_4 \cdot (1)^4 + c_5 \cdot (1)^5 = f(1) \\
p'(1) &= 0 + c_1 + c_2 \cdot 2(1) + c_3 \cdot 3(1)^2 + c_4 \cdot 4(1)^3 + c_5 \cdot 5(1)^4 = f'(1).
\end{aligned}$$

These equations can be written in the matrix form

$$\begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ 0 & 1 & -2 & 3 & -4 & 5 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f'(x_0) \\ f(x_1) \\ f''(x_1) \\ f(x_2) \\ f'(x_2) \end{bmatrix}.$$

(d) Using Python, one can find that $\mathbf{A}$ has a one-dimensional null space spanned by the vector

$$\begin{bmatrix} 0.00000000e+00 \\ 4.08248290e-01 \\ 5.55111512e-17 \\ -8.16496581e-01 \\ 5.55111512e-17 \\ 4.08248290e-01 \end{bmatrix}$$

which is ugly, but can easily be cleaned up to:

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ -2 \\ 0 \\ 1 \end{bmatrix}.$$

Since $\mathcal{N}(\mathbf{A})$ is nontrivial, there are two possible situations: there will either be *no polynomial*, or there will be *infinitely many polynomials* that satisfy the six interpolation conditions. Which of the two depends on the actual interpolation conditions.

(e) The vector

$$\mathbf{f} = \begin{bmatrix} f(x_0) \\ f'(x_0) \\ f(x_1) \\ f''(x_1) \\ f(x_2) \\ f'(x_2) \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \\ -2 \\ 3 \\ 4 \end{bmatrix}$$

is in the column space of $\mathbf{A}$, since we can write

$$\begin{bmatrix} -1 \\ 0 \\ 1 \\ -2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ 0 & 1 & -2 & 3 & -4 & 5 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 5/3 \\ -1 \\ 2/3 \\ 1 \\ -1/3 \end{bmatrix}.$$

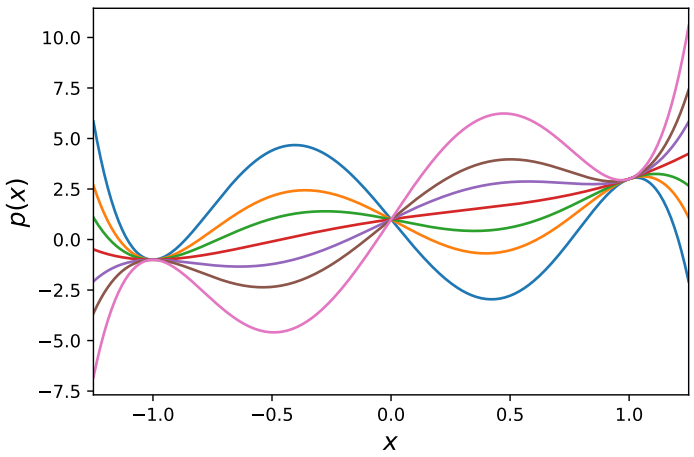
The vector on the right is $\mathbf{c}_+ = \mathbf{A}^+\mathbf{f}$, which can be obtained from the `scipy.linalg.pinv` command.

Since $\mathbf{f}$ is in $\mathcal{R}(\mathbf{A})$, there are infinitely many choices for the coefficients $c_0, \ldots, c_5$ that will satisfy the six constraints. All solutions have the form

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 5/3 \\ -1 \\ 2/3 \\ 1 \\ -1/3 \end{bmatrix} + \gamma \begin{bmatrix} 0 \\ 1 \\ 0 \\ -2 \\ 0 \\ 1 \end{bmatrix}$$

for arbitrary γ.

The plot below shows polynomials for the above coefficients chosen with $\gamma = -16, -8, -4, 0, 4, 8, 16$. Students will have different choices of vectors from the null space, but all polynomials should match at $x = -1$, $x = 0$, and $x = 1$, and the required derivative conditions should also match at the three points..



---

2. [27 points: 5 points each for (a), (b); 10 points for (c); 7 points for (d)]

   The linear system $\mathbf{Ax} = \mathbf{b}$ given by

   $$\begin{bmatrix} 1 & 50 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

   has infinitely many solutions; the one with smallest norm is

   $$\mathbf{x}_+ = \mathbf{A}^+\mathbf{b} = \begin{bmatrix} 2/2501 \\ 100/2501 \end{bmatrix} \approx \begin{bmatrix} 0.0008 \\ 0.0400 \end{bmatrix}.$$

For all $\varepsilon \neq 0$, the perturbed equation $\mathbf{A}_\varepsilon \mathbf{x}_\varepsilon = \mathbf{b}_\varepsilon$,

$$\begin{bmatrix} 1 & 50 \\ 0 & \varepsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ \varepsilon \end{bmatrix},$$

has a unique solution. Since we are only making small changes (adding 'noise') to the problem, we might expect to get a solution $\mathbf{x}_\varepsilon$ that is close to $\mathbf{x}_+$. This problem explores whether that is the case (conceptually following on from Problem 1 on Problem Set 6).

(a) For any $\varepsilon \neq 0$, solve $\mathbf{A}_\varepsilon \mathbf{x}_\varepsilon = \mathbf{b}_\varepsilon$ for the unique solution $\mathbf{x}_\varepsilon$.
How does it compare to the best solution to the unperturbed problem, $\mathbf{x}_+$?
Is $\mathbf{x}_\varepsilon$ close to *a* solution of the unperturbed problem $\mathbf{A}\mathbf{x} = \mathbf{b}$?

(b) Let $\varepsilon = 0.01$. In Python, compute the SVD

$$\mathbf{A}_\varepsilon = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T,$$

and construct the *truncated SVD* solution for the perturbed problem:

$$\mathbf{x}_{\text{SVD}} = \frac{1}{\sigma_1} \mathbf{v}_1 \mathbf{u}_1^T \mathbf{b}_\varepsilon.$$

How does this 'solution' $\mathbf{x}_{\text{SVD}}$ to the perturbed problem compare to $\mathbf{x}_\varepsilon$ and $\mathbf{x}_+$?

(c) The truncated SVD approach is expensive for large problems, since it requires computation of the singular value decomposition. It can be cheaper to solve the *regularized* least-squares problem

$$\min_{\mathbf{x} \in \mathbb{R}^2} \|\mathbf{A}_\varepsilon \mathbf{x} - \mathbf{b}_\varepsilon\|^2 + \lambda^2 \|\mathbf{x}\|^2$$

for various values of λ. As we shall see in Lecture 16, the optimal $\mathbf{x}_\lambda$ satisfies the least squares problem

$$\min_{\mathbf{x} \in \mathbb{R}^2} \left\| \begin{bmatrix} \mathbf{A}_\varepsilon \\ \lambda \mathbf{I} \end{bmatrix} \mathbf{x} - \begin{bmatrix} \mathbf{b}_\varepsilon \\ \mathbf{0} \end{bmatrix} \right\|,$$

where

$$\mathbf{A}_\lambda = \begin{bmatrix} \mathbf{A}_\varepsilon \\ \lambda \mathbf{I} \end{bmatrix} \in \mathbb{R}^{4 \times 2}, \qquad \widehat{\mathbf{b}} = \begin{bmatrix} \mathbf{b}_\varepsilon \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^4.$$

In Python, you can solve this least squares problem (`from scipy import linalg as la`) via the command
`xlambda = la.lstsq(Alambda, bhat)[0]`. (Note that `la.lstsq` returns diagnostic arguments, in addition to the solution; the `[0]` at the end of the command returns only the first argument, the solution we seek.)

Solve the regularized problem for 100 values of λ logarithmically spaced between $\lambda = 10^{-5}$ and $\lambda = 10^2$ (try `lam = np.logspace(-5,2,100)`). Produce one `plt.loglog` plot that shows, for each value of λ, a dot corresponding to $\|\mathbf{A}_\varepsilon \mathbf{x}_\lambda - \mathbf{b}_\varepsilon\|$ on the horizontal axis, and $\|\mathbf{x}_\lambda\|$ on the vertical axis. Your plot should include a prominent 'L' shape, as we shall discuss in Lecture 16.

(d) Show the value of the solution $\mathbf{x}_\lambda$ for three values of λ corresponding to (i) a point well above the right-angle in the L; (ii) a point at the angle of the L; (iii) a point after the angle of the L. Which of these most closely agrees with the original unperturbed solution $\mathbf{x}_+$?

Solution.

(a) For all $\varepsilon > 0$, we find

$$\mathbf{x}_\varepsilon = \begin{bmatrix} -48 \\ 1 \end{bmatrix},$$

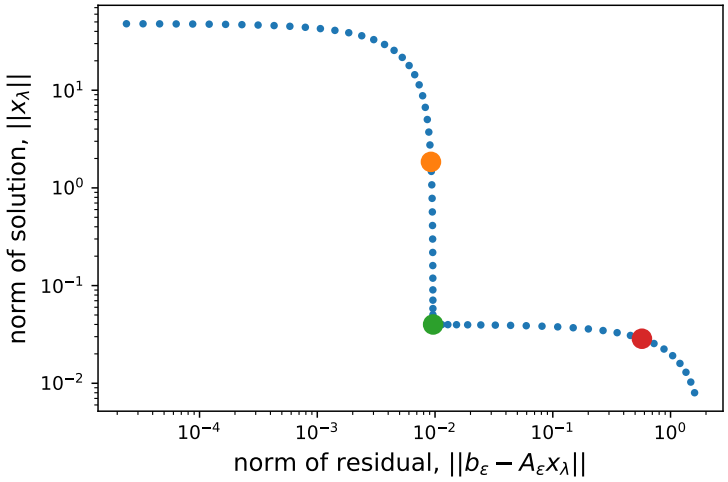
which is quite far from the solution $\mathbf{x}_+$ to the unperturbed problem.
However, $\mathbf{x}_\varepsilon$ is exactly a solution of the unperturbed problem, just not the minimum norm solution.

(b) For $\varepsilon = 0.01$, the truncated SVD gives

$$\mathbf{x}_{\text{SVD}} = \frac{1}{\sigma_1}\mathbf{v}_1\mathbf{u}_1^T\mathbf{b}_\varepsilon = \begin{bmatrix} 0.00079968\ldots \\ 0.03998404\ldots \end{bmatrix},$$

which is quite far from $\mathbf{x}_\varepsilon$, but is actually quite close to the solution $\mathbf{x}_+$ to the unperturbed problem. The truncated SVD has essentially helped us ignore the affect of the perturbation.

(c) The L-curve is shown in the plot below.



(d) We choose three values of λ, one before the critical bend in the L ($\lambda = 10^{-3}$), one at the bend ($\lambda = 10^0$), and one beyond it ($\lambda = 10^1$) (shown as dots on the plot above – that is not required, but is helpful to see). The critical λ gives the closest approximation to the solution $\mathbf{x}_+$ to the unperturbed problem:

$$\lambda = 10^{-3}, \qquad \mathbf{x}_\lambda = \begin{bmatrix} -1.844675053320961 \\ 0.076893537959920 \end{bmatrix}$$

$$\lambda = 10^{0}, \qquad \mathbf{x}_\lambda = \begin{bmatrix} 0.000797441982619 \\ 0.039968102320695 \end{bmatrix}$$

$$\lambda = 10^{1.5}, \qquad \mathbf{x}_\lambda = \begin{bmatrix} 0.000571263965386 \\ 0.028563295412972 \end{bmatrix}$$

Students do not need to have these three particular values of λ, but they should be before, near, and after the bend in the L curve.

---

3. [48 points: 8 points per part]

In class we shall encounter the *integral equation*

$$\int_0^1 h(s,t)f(t)\,\mathrm{d}t = b(s),$$

where the *kernel* $h(s,t)$ acts to *smooth* the function f, so we may regard b as a *blurred* version of f. (In typical applications we can *observe* the blurred function b – for example, through a camera or telescope – and we seek to find the original image f that got blurred.)

As described in Section 7.3 of the course notes, we can turn this calculus problem into a linear algebra problem. Break the interval $[0,1]$ into discrete points:

$$s_j = \frac{j-1/2}{n}, \qquad t_k = \frac{k-1/2}{n}, \qquad j,k = 1,\ldots,n,$$

for some $n \geq 1$. Then approximate the integral by the simple *midpoint quadrature rule*:

$$\frac{1}{n}\sum_{k=1}^{n} h(s_j, t_k) f_k = b_j, \qquad j = 1, \ldots, n, \tag{$*$}$$

where $b_j = b(s_j)$ and (hopefully) $f_k \approx f(t_k)$.

Arranging all n of the equations $(*)$ for $j = 1, \ldots, n$ into matrix form gives $\mathbf{A}\mathbf{f} = \mathbf{b}$:

$$\frac{1}{n}\begin{bmatrix} h(s_1,t_1) & h(s_1,t_2) & \cdots & h(s_1,t_n) \\ h(s_2,t_1) & h(s_2,t_2) & \cdots & h(s_2,t_n) \\ \vdots & \vdots & \ddots & \vdots \\ h(s_n,t_1) & h(s_n,t_2) & \cdots & h(s_n,t_n) \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Be sure to include the $1/n$ term in the matrix $\mathbf{A}$; missing the $1/n$ factor is a common mistake.

We will study how the blurring operation smooths out f, and how the singular values of the matrix $\mathbf{A}$ behave.

We shall consider Gaussian kernels: for some constant $z > 0$, let

$$h(s,t) = \frac{1}{\sqrt{\pi} z} \mathrm{e}^{-(s-t)^2/z^2}.$$

(a) To get to know the Gaussian kernel, $h(s,t)$, produce a plot showing $h(0.5,t)$ for $t \in [0,1]$ for the three values $z = 0.2, 0.1, 0.05$.

- Make one plot showing all three z values together, with $s = 0.5$ and t varying from 0 to 1.
- You should sample $h(0.5,t)$ at many values of t in the interval $[0,1]$ and plot $h(0.5,t)$ as a connected line.
- The command `t = np.linspace(0,1,500)` creates an array of 500 t values between 0 and 1.
- *Refer to Figure 7.4 in the course notes, which shows a similar plot (but for only one value of z and a different kernel function h).*

(b) Write Python code to construct $\mathbf{A}$ for this specific Gaussian kernel $h(s,t)$, given values of z and n. *Refer to the code in Problem Set 3 (Problem 1) for constructing matrices of this general type.*

(c) Set $n = 500$ and consider the function

$$f(t) = \begin{cases} 1, & t \in [1/4, 3/4]; \\ 0, & t \notin [1/4, 3/4]. \end{cases}$$

- Create the vector $\mathbf{f} \in \mathbb{R}^n$, having kth entry $f_k = f(t_k)$, where $t_k = (k - 1/2)/n$. *The values of t_k should be between 0 and 1.*
- Create one plot comparing $\mathbf{f}$ to $\mathbf{b} = \mathbf{A}\mathbf{f}$ for the three values $z = 0.2$, $z = 0.1$, and $z = 0.05$. *This plot should have the index j on the horizontal axis and the jth entries f_j and b_j of $\mathbf{f}$ and $\mathbf{b}$ on the vertical axis.*

How does the value of z affect the amount of blurring? Is the blurring more severe when z is large or small? *Justify your answer.*

(d) For each of $z = 0.2$, $z = 0.1$, and $z = 0.05$, create a `plt.loglog` plot showing the singular values of $\mathbf{A}$ for the three values $n = 250$, $n = 500$, and $n = 1000$. (Produce one plot for each z, giving three total plots.)
Describe how the singular values behave as a function of z and n.

(e) Create the blurring matrix $\mathbf{A}$ for $n = 500$ and $z = 0.05$.
For $k = 1, 2, 3, 498, 499, 500$ (six values of k), produce a plot of the right singular $\mathbf{v}_j$ of $\mathbf{A}$.

- You should produce *six* plots; each one should range from $j = 1, \ldots, n$ on the horizontal axis and show $(\mathbf{v}_k)_j$, the jth entry of $\mathbf{v}_k$, on the vertical axis.

- Remember that if you compute the SVD via `U, S, Vt = la.svd(A)`, then $\mathbf{v}_k$ will be stored in row $k-1$ of `Vt`, i.e., `Vt[k-1,:]`.
- *See Figure 7.13 in the notes, which shows such singular vectors for a different kernel function h.*

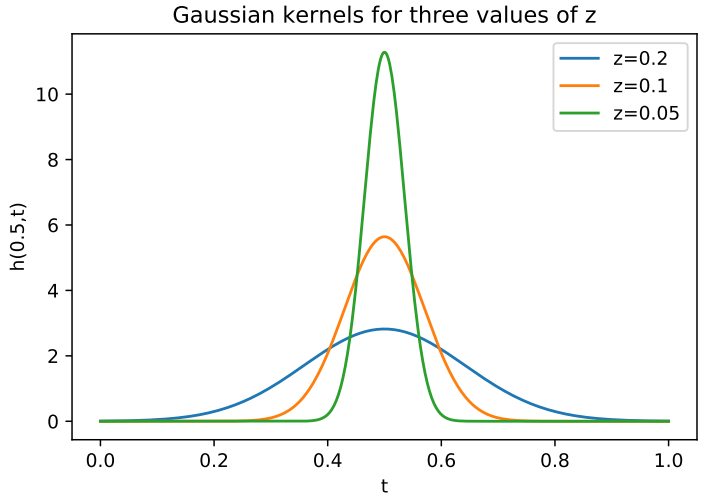
(f) For $n = 500$ and $z = 0.05$, create $\mathbf{f}$ and form $\mathbf{b} = \mathbf{A}\mathbf{f}$, just as in part (c), and then use the command `frec = la.solve(A,b)` to compute $\mathbf{f}_{\text{rec}}$, the *recovered* vector $\mathbf{f}_{\text{rec}} = \mathbf{A}^{-1}\mathbf{b}$.
Produce a plot showing both $\mathbf{f}$ and the vector $\mathbf{f}_{\text{rec}}$.

- Since we formed $\mathbf{b} = \mathbf{A}\mathbf{f}$, we *expect* $\mathbf{f}_{\text{rec}} = \mathbf{A}^{-1}\mathbf{b} = \mathbf{A}^{-1}\mathbf{A}\mathbf{f} = \mathbf{f}$.
- If Python throws a warning during this computation, please note it in your solution.
- Do the vectors $\mathbf{f}$ and $\mathbf{f}_{\text{rec}}$ agree in your computation?
  If not, use your answer from questions (d) and (e) to speculate about why they are different.

Solution.

(a) The following code will produce the desired plot, shown below.

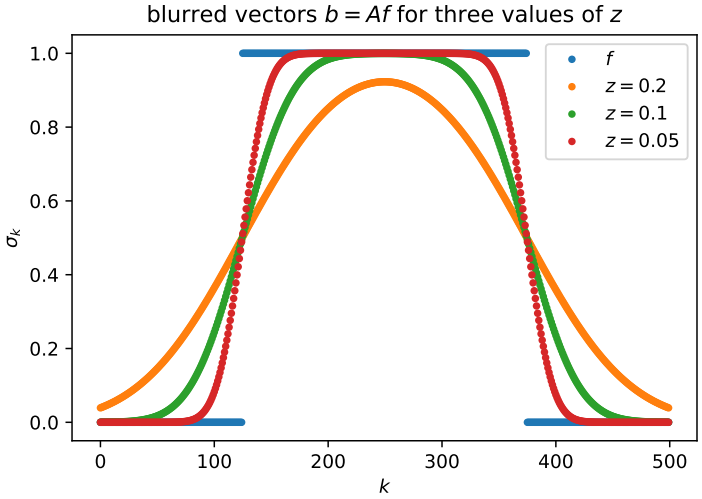
```
def h(s,t,z=.1):
    return (1/(np.sqrt(np.pi)*z))*np.exp(-np.square(s-t)/(z**2))

tt = np.linspace(0,1,500)
plt.figure()
for z in (0.2, 0.1, 0.05):
    plt.plot(tt,h(0.5,tt,z),'-')
plt.xlabel('$t$',fontsize=13)
plt.ylabel('$h(0.5,t)$',fontsize=13)
plt.title('Gaussian kernels for three values of $z$');
plt.legend(('$z=0.2$','$z=0.1$','$z=0.05$'))
```



(b) The following Python functions will construct $\mathbf{A}$ for a given n and z.

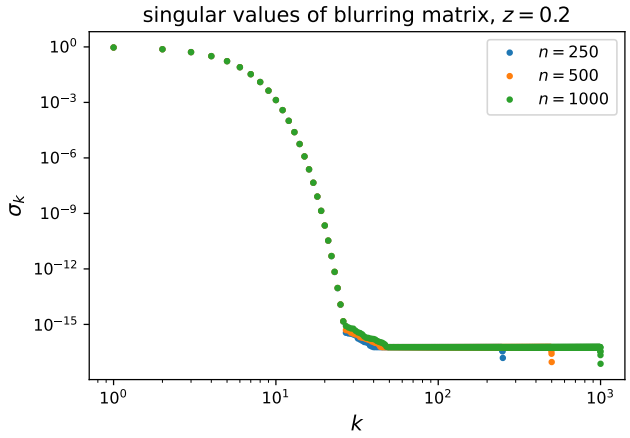
```
def build_blur_A(n=32,z=.25):
    A = np.zeros((n,n));
    s = np.array([(j-.5)/n for j in range(1,n+1)])
    t = np.array([(k-.5)/n for k in range(1,n+1)])
    for j in range(0,n):
        A[j,:] = h(s[j],t,z)/n
    return A
```

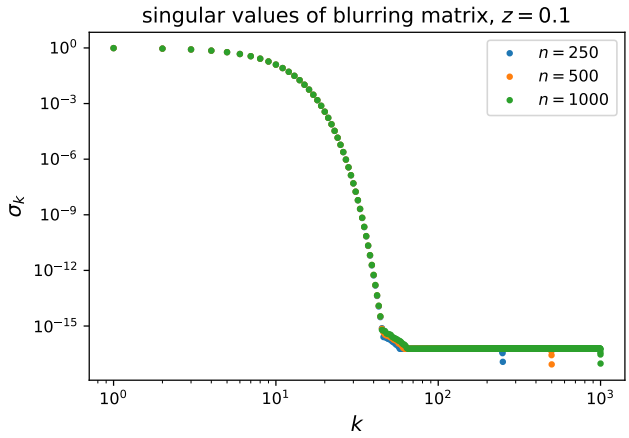
(c) As z *decreases*, the Gaussian kernel is more focused, so the blurring effect is *less severe* when z *is small*.

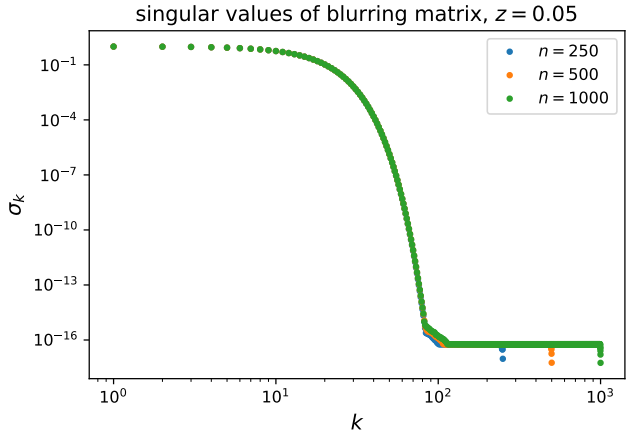


(d) The singular values of $\mathbf{A}$ decay rapidly for all the examples illustrated below. The effect of the decay appears to be largely independent of the parameter n; the decay is faster when z is larger.

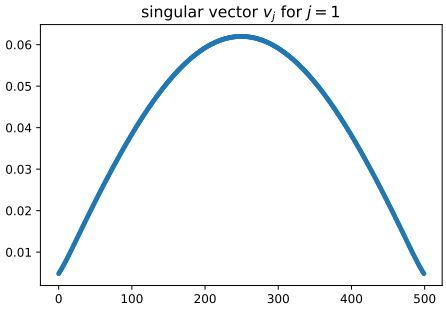
- These values of n are large enough that the dominant singular values of $\mathbf{A}$ have "converged" in some sense (which we could likely make precise using language from functional analysis). Increasing n further mostly just adds more small singular values. (This behavior is typical for integral operators.)
- In part (b) we saw that larger values of z correspond to more severe blurring. The more severe the blurring operation, the more effective $\mathbf{A}$ is at erasing fine detail from $\mathbf{f}$. The greater the blur, the more $\mathbf{A}$ is dominated by its leading singular values, and the more rapidly the singular values will decay. (You might think of the most severe blurring of all: the $\mathbf{A}$ that sends all $\mathbf{f}$ to a constant function. Such an $\mathbf{A}$ would have rank-1, with $n - 1$ zero singular values!)

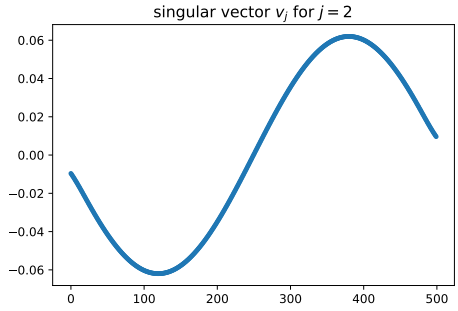


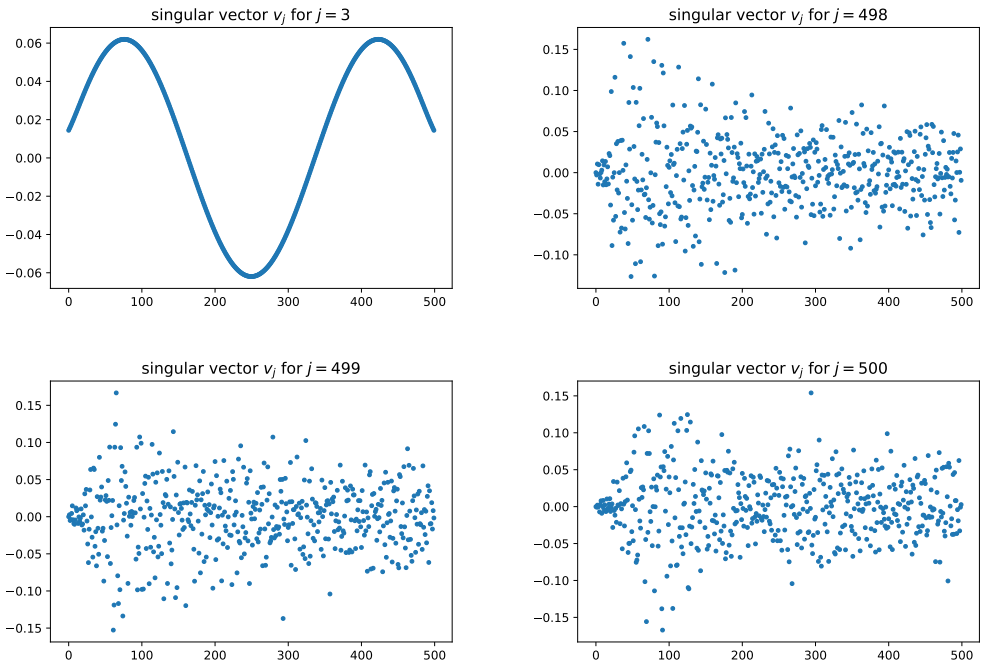




(e) The requested right singular vectors are shown below. Just like the example in the class notes (with a different h), the leading singular vectors are low in frequency, while the latter ones are very noisy.







(f) The `la.solve` command throws the flag

```
LinAlgWarning: Ill-conditioned matrix (rcond=1.79357e-21): result may not be accurate.
```

The solution is inaccurate. The fact that $\mathbf{A}$ has so many very small singular values suggests that the inverse

$$\mathbf{f}_{\text{rec}} = \mathbf{A}^{-1}\mathbf{f} = \sum_{j=1}^{n} \frac{\mathbf{u}_j^T \mathbf{b}}{\sigma_j} \mathbf{v}_j$$

will, when polluted by slight floating point rounding error, be dominated by the $\mathbf{v}_j$ vectors corresponding to small singular vectors. In the last part of the problem we saw that these singular vectors were highly oscillatory, which reflects the high-frequency garbage see in the plot of $\mathbf{f}_{\text{rec}}$.