EE263 homework 3 solutions

2.17 *Gradient of some common functions.* Recall that the gradient of a differentiable function $f : \mathbf{R}^n \to \mathbf{R}$, at a point $x \in \mathbf{R}^n$, is defined as the vector

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix},$$

where the partial derivatives are evaluated at the point x. The first order Taylor approximation of f, near x, is given by

$$\hat{f}_{\text{tay}}(z) = f(x) + \nabla f(x)^T(z - x).$$

This function is affine, *i.e.*, a linear function plus a constant. For z near x, the Taylor approximation $\hat{f}_{\text{tay}}$ is very near f. Find the gradient of the following functions. Express the gradients using matrix notation.

(a) $f(x) = a^T x + b$, where $a \in \mathbf{R}^n$, $b \in \mathbf{R}$.

(b) $f(x) = x^T A x$, for $A \in \mathbf{R}^{n \times n}$.

(c) $f(x) = x^T A x$, where $A = A^T \in \mathbf{R}^{n \times n}$. (Yes, this is a special case of the previous one.)

Solution:

(a)

$$\begin{aligned} \frac{g^T \Delta x}{\|\Delta x\|} &= \lim_{h \to 0} \frac{f(x + h\Delta x) - f(x)}{\|h\Delta x\|} \\ &= \lim_{h \to 0} \frac{a^T(x + h\Delta x) + b - a^T x - b}{\|h\Delta x\|} \\ &= \lim_{h \to 0} \frac{a^T x + h a^T \Delta x - a^T x}{\|h\Delta x\|} \\ &= \lim_{h \to 0} \frac{a^T \Delta x}{\|\Delta x\|} \\ &= \frac{a^T \Delta x}{\|\Delta x\|} \end{aligned}$$

so $g^T = a^T$, *i.e.*, $g = a$.

(b)

$$\begin{aligned}
\frac{g^T\Delta x}{\|\Delta x\|} &= \lim_{h\to 0}\frac{f(x+h\Delta x)-f(x)}{\|h\Delta x\|} \\
&= \lim_{h\to 0}\frac{(x+h\Delta x)^T A(x+h\Delta x)-x^TAx}{\|h\Delta x\|} \\
&= \lim_{h\to 0}\frac{x^TAx+x^TAh\Delta x+h\Delta x^TAx+h^2\Delta x^TA\Delta x)-x^TAx}{\|h\Delta x\|} \\
&= \lim_{h\to 0}\frac{h(x^TA\Delta x+\Delta x^TAx)+h^2\Delta x^TA\Delta x}{\|h\Delta x\|} \\
&= \lim_{h\to 0}\frac{x^TA\Delta x+\Delta x^TAx+h\Delta x^TA\Delta x}{\|\Delta x\|} \\
&= \frac{x^TA\Delta x+x^TA^T\Delta x}{\|\Delta x\|} \qquad (\Delta x^TAx \text{ is scalar})
\end{aligned}$$

so $g^T = x^TA + x^TA^T$, *i.e.*, $g = (A+A^T)x$.

(c) From the solution of 0b, clearly $g = 2Ax$.

Note: These derivations start off from the first version of the definition, the "exact" version (because they are formally sound that way). It is then easy to see how to derive the same things from the second version of the definition, the "approximation" version. It's very similar.

3.13 *Right inverses.* This problem concerns the specific matrix

$$A = \begin{bmatrix} -1 & 0 & 0 & -1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

This matrix is full rank (*i.e.*, its rank is 3), so there exists at least one right inverse. In fact, there are many right inverses of A, which opens the possibility that we can seek right inverses that in addition have other properties. For each of the cases below, either find a specific matrix $B \in \mathbf{R}^{5\times 3}$ that satisfies $AB = I$ and the given property, or explain why there is no such B. In cases where there is a right inverse B with the required property, you must briefly explain how you found your B. You must also attach a printout of some Matlab scripts that show the verification that $AB = I$. (We'll be very angry if we have to type in your 5×3 matrix into Matlab to check it.) When there is no right inverse with the given property, briefly explain why there is no such B.

(a) The second row of B is zero.

(b) The nullspace of B has dimension one.

(c) The third column of B is zero.

(d) The second and third rows of B are the same.

(e) B is upper triangular, *i.e.*, $B_{ij} = 0$ for $i > j$.

(f) B is lower triangular, *i.e.*, $B_{ij} = 0$ for $i < j$.

Solution:

(a) The second row of B is zero. This means that the second column of A isn't used in forming AB. Let $\tilde{A}$ be the matrix A with its second column removed, and let $\tilde{B}$ denote the matrix B with its second row (which is supposed to be zero) removed. We have $\tilde{A}\tilde{B} = AB = I$, so $\tilde{B}$ is a right inverse of $\tilde{A}$. There is such a matrix if and only if $\tilde{A}$ is full rank, which it is. We can take $\tilde{B} = \tilde{A}^T(\tilde{A}\tilde{A}^T)^{-1}$. Finally to construct B we simply insert a zero second row, moving rows 2, 3, 4 down by one. This gives the matrix

$$B = \begin{bmatrix} 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} \\ 1 & 0 & 1 \end{bmatrix}.$$

There are other possible choices as well.

(b) The nullspace of B has dimension one. This means that B has rank 2, so the rank of AB is at most 2, which rules out the possibility that $AB = I$. So this is impossible.

(c) The third column of B is zero. This implies B has a nullspace with dimension at least one, so by part (b) above, this is impossible too.

(d) The second and third rows of B are the same. Let $\tilde{B}$ denote B with one of the (identical) rows 2 and 3 deleted. Then we have $AB = \tilde{A}\tilde{B}$, where $\tilde{A}$ is obtained from the matrix A by replacing its second column with the sum of its second and third columns, and deleting its third column. Thus, we need to find a right inverse for $\tilde{A}$, provided it is full rank. It is, so we can take $\tilde{B} = \tilde{A}(\tilde{A}\tilde{A}^T)^{-1}$. Finally to construct B we simply insert a second copy of the second row of $\tilde{B}$ as a new third row. This gives

$$B = \begin{bmatrix} 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \\ 1 & 0 & 1 \end{bmatrix}.$$

This matrix also happens to be the pseudo-inverse of A, $B = A^T(AA^T)^{-1}$, and some of you noticed this immediately and used the pseudo-inverse to answer this question. That's a fine answer; it was our mistake to choose A so that the pseudo-inverse satisfied this condition. In general, of course, it would not.

(e) B is upper triangular, *i.e.*, $B_{ij} = 0$ for $i > j$. If B is upper triangular, then it has the form

$$\begin{bmatrix} \tilde{B} \\ 0 \end{bmatrix},$$

where $\tilde{B}$ is square and upper triangular. If $AB = I$, then $\tilde{A}\tilde{B} = I$, where $\tilde{A}$ is the matrix formed from the first 3 columns of A. Thus we have $\tilde{A} = \tilde{B}^{-1}$. But the inverse of an upper triangular matrix is also upper triangular, so unless $\tilde{A}$ is upper triangular (and it isn't, in this case), we can't possibly have $\tilde{A}\tilde{B} = I$. So there is no such B in this case.

(f) B is lower triangular, *i.e.*, $B_{ij} = 0$ for $i < j$. Let's label the columns of B as

$$b_1, \qquad b_2 = \begin{bmatrix} 0 \\ \tilde{b}_2 \end{bmatrix}, \qquad b_3 = \begin{bmatrix} 0 \\ 0 \\ \tilde{b}_3 \end{bmatrix},$$

where $\tilde{b}_2 \in \mathbf{R}^4$ and $\tilde{b}_3 \in \mathbf{R}^3$. To say that $AB = I$ is the same as saying that $Ab_1 = e_1$, $Ab_2 = e_2$, and $Ab_3 = e_3$, where e_1, e_2, e_3 are the unit vectors. We can solve these equations separately. The first equation is easy; the second we reduce to $\tilde{A}\tilde{b}_2 = e_2$, where here $\tilde{A}$ is A with its first column removed. The third is handled similarly. These equations do have a solution; we get

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}.$$

Another way: we set it up as a set of 9 linear equations (one for each entry of $AB = I$) in $5+4+3 = 12$ variables. The variables are the first column of B (with 5 entries), the nonzero part of the second column of B (with 4 entries), and the nonzero part of the third second column of B (with 3 entries). We then attempt to solve these 9 equations in 12 variables. Some equations immediately give us the B matrix coefficients, while the others can be solved by inspection to obtain a rather simple matrix

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}.$$

4.1 *Bessel's inequality.* Suppose the columns of $U \in \mathbf{R}^{n\times k}$ are orthonormal. Show that $\|U^T x\| \leq \|x\|$. When do we have $\|U^T x\| = \|x\|$? *Solution:*

We know that if U is an orthogonal matrix (has orthonormal columns and is square) then $\|U^T x\| = \|x\|$ because

$$\begin{aligned} \|U^T x\|^2 &= x^T U U^T x \\ &= x^T x \qquad (\text{since } UU^T = I). \\ &= \|x\|^2. \end{aligned}$$

In this problem, U is not necessarily square and in general is skinny ($k \leq n$). If $k = n$ we know then $\|U^T x\| \leq \|x\|$ holds with the equality. For $k < n$, it is always possible to find $n - k$ vectors $\tilde{u}_1, \ldots, \tilde{u}_{n-k}$ such that

$$Q = \begin{bmatrix} U & \tilde{U} \end{bmatrix} \in \mathbf{R}^{n \times n}$$

becomes an orthogonal matrix with $\tilde{U} = [\tilde{u}_1 \cdots \tilde{u}_{n-k}]$. For example, this can be done by finding a QR decomposition of $\tilde{A} = [U\ I]$. Now since Q is orthogonal $\|Q^T x\|^2 = \|x\|^2$ and therefore

$$\|[U\ \tilde{U}]^T x\|^2 = \|x\|^2$$

or

$$\left\| \begin{bmatrix} U^T \\ \tilde{U}^T \end{bmatrix} x \right\|^2 = \|x\|^2.$$

But $\begin{bmatrix} U^T \\ \tilde{U}^T \end{bmatrix} x = \begin{bmatrix} U^T x \\ \tilde{U}^T x \end{bmatrix}$ and we get

$$\left\| \begin{bmatrix} U^T x \\ \tilde{U}^T x \end{bmatrix} \right\|^2 = \|x\|^2$$

so

$$\|U^T x\|^2 + \|\tilde{U}^T x\|^2 = \|x\|^2 \tag{1}$$

where we have used the simple fact that

$$\left\| \begin{bmatrix} a \\ b \end{bmatrix} \right\|^2 = \|a\|^2 + \|b\|^2$$

for any column vectors a and b (here $a = U^T x$ and $b = \tilde{U}^T x$). Now it is easy to see from (1) that $\|U^T x\| \leq \|x\|$. Since $\|\tilde{U}^T x\|^2 \geq 0$ we have $\|U^T x\|^2 \leq \|x\|^2$ and we are done.

4.2 *Orthogonal matrices.*

(a) Show that if U and V are orthogonal, then so is UV.

(b) Show that if U is orthogonal, then so is U^{-1}.

(c) Suppose that $U \in \mathbf{R}^{2\times 2}$ is orthogonal. Show that U is either a rotation or a reflection. Make clear how you decide whether a given orthogonal U is a rotation or reflection.

Solution:

(a) To prove that UV is orthogonal we have to show that $(UV)^T(UV) = I$ given $U^TU = I$ and $V^TV = I$. We have

$$\begin{aligned} (UV)^T(UV) &= V^TU^TUV \\ &= V^TV \qquad (\text{since } U^TU = I) \\ &= I \qquad (\text{since } V^TV = I) \end{aligned}$$

and we are done.

(b) Since U is square and orthogonal we have $U^{-1} = U^T$ and therefore by taking inverses of both sides $U = (U^T)^{-1}$ or equivalently $U = (U^{-1})^T$ (the inverse and transpose operations commute.) But $U^TU = I$ and by substitution $U^{-1}(U^{-1})^T = I$. Since U^{-1} is square this also implies that $(U^{-1})^TU^{-1} = I$ so U^{-1} is orthogonal.

(c) Suppose that $U = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathbf{R}^{2\times 2}$ is orthogonal. This is true if and only if

- columns of U are of unit length, *i.e.*, $a^2 + c^2 = 1$ and $b^2 + d^2 = 1$,
- columns of U are orthogonal, *i.e.*, $ab + cd = 0$.

Since $a^2 + c^2 = 1$ we can take a and c as the cosine and sine of an angle α respectively, *i.e.*, $a = \cos\alpha$ and $c = \sin\alpha$. For a similar reason, we can take $b = \sin\beta$ and $d = \cos\beta$. Now $ab + cd = 0$ becomes

$$\cos\alpha\sin\beta + \sin\alpha\cos\beta = 0$$

or

$$\sin(\alpha + \beta) = 0.$$

The sine of an angle is zero if and only if the angle is an integer multiple of π. So $\alpha + \beta = k\pi$ or $\beta = k\pi - \alpha$ with $k \in \mathbf{Z}$. Therefore

$$U = \begin{bmatrix} \cos\alpha & \sin(k\pi - \alpha) \\ \sin\alpha & \cos(k\pi - \alpha) \end{bmatrix}.$$

Now two things can happen:

- k is even so $\sin(k\pi - \alpha) = -\sin\alpha$ and $\cos(k\pi - \alpha) = \cos\alpha$, and therefore

$$U = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}.$$

Clearly, from the lecture notes, this represents a rotation. Note that in this case $\det U = \cos^2\alpha + \sin^2\alpha = 1$.

- k is odd so $\sin(k\pi - \alpha) = \sin\alpha$ and $\cos(k\pi - \alpha) = -\cos\alpha$, and therefore

$$U = \begin{bmatrix} \cos\alpha & \sin\alpha \\ \sin\alpha & -\cos\alpha \end{bmatrix}.$$

  From the lecture notes, this represents a reflection. The determinant in this case is $\det U = -\cos^2\alpha - \sin^2\alpha = -1$.

Therefore we have shown that any orthogonal matrix in $\mathbf{R}^{2\times 2}$ is either a rotation or reflection whether its determinant is $+1$ or -1 respectively.

4.3 *Projection matrices.* A matrix $P \in \mathbf{R}^{n\times n}$ is called a *projection matrix* if $P = P^T$ and $P^2 = P$.

(a) Show that if P is a projection matrix then so is $I - P$.

(b) Suppose that the columns of $U \in \mathbf{R}^{n\times k}$ are orthonormal. Show that UU^T is a projection matrix. (Later we will show that the converse is true: every projection matrix can be expressed as UU^T for some U with orthonormal columns.)

(c) Suppose $A \in \mathbf{R}^{n\times k}$ is full rank, with $k \leq n$. Show that $A(A^TA)^{-1}A^T$ is a projection matrix.

(d) If $S \subseteq \mathbf{R}^n$ and $x \in \mathbf{R}^n$, the point y in S closest to x is called the *projection of x on S*. Show that if P is a projection matrix, then $y = Px$ is the projection of x on $\mathcal{R}(P)$. (Which is why such matrices are called projection matrices ...)

Solution:

(a) To show that $I - P$ is a projection matrix we need to check two properties:

i. $I - P = (I - P)^T$

ii. $(I - P)^2 = I - P$.

The first one is easy: $(I - P)^T = I - P^T = I - P$ because $P = P^T$ (P is a projection matrix.) The show the second property we have

$$\begin{aligned} (I-P)^2 &= I - 2P + P^2 \\ &= I - 2P + P \qquad (\text{since } P = P^2) \\ &= I - P \end{aligned}$$

and we are done.

(b) Since the columns of U are orthonormal we have $U^TU = I$. Using this fact it is easy to prove that UU^T is a projection matrix, *i.e.*, $(UU^T)^T = UU^T$ and $(UU^T)^2 = UU^T$. Clearly, $(UU^T)^T = (U^T)^TU^T = UU^T$ and

$$\begin{aligned} (UU^T)^2 &= (UU^T)(UU^T) \\ &= U(U^TU)U^T \\ &= UU^T \qquad (\text{since } U^TU = I). \end{aligned}$$

(c) First note that $\left(A(A^TA)^{-1}A^T\right)^T = A(A^TA)^{-1}A^T$ because

$$\begin{aligned}\left(A(A^TA)^{-1}A^T\right)^T &= (A^T)^T\left((A^TA)^{-1}\right)^T A^T \\ &= A\left((A^TA)^T\right)^{-1}A^T \\ &= A(A^TA)^{-1}A^T.\end{aligned}$$

Also $\left(A(A^TA)^{-1}A^T\right)^2 = A(A^TA)^{-1}A^T$ because

$$\begin{aligned}\left(A(A^TA)^{-1}A^T\right)^2 &= \left(A(A^TA)^{-1}A^T\right)\left(A(A^TA)^{-1}A^T\right) \\ &= A\left((A^TA)^{-1}A^TA\right)(A^TA)^{-1}A^T \\ &= A(A^TA)^{-1}A^T \qquad \text{(since } (A^TA)^{-1}A^TA = I\text{)}.\end{aligned}$$

(d) To show that Px is the projection of x on $\mathcal{R}(P)$ we verify that the "error" $x - Px$ is orthogonal to *any* vector in $\mathcal{R}(P)$. Since $\mathcal{R}(P)$ is nothing but the span of the columns of P we only need to show that $x - Px$ is orthogonal to the columns of P, or in other words, $P^T(x - Px) = 0$. But

$$\begin{aligned}P^T(x - Px) &= P(x - Px) \qquad \text{(since } P = P^T\text{)} \\ &= Px - P^2x \\ &= 0 \qquad \text{(since } P^2 = P\text{)}\end{aligned}$$

and we are done.

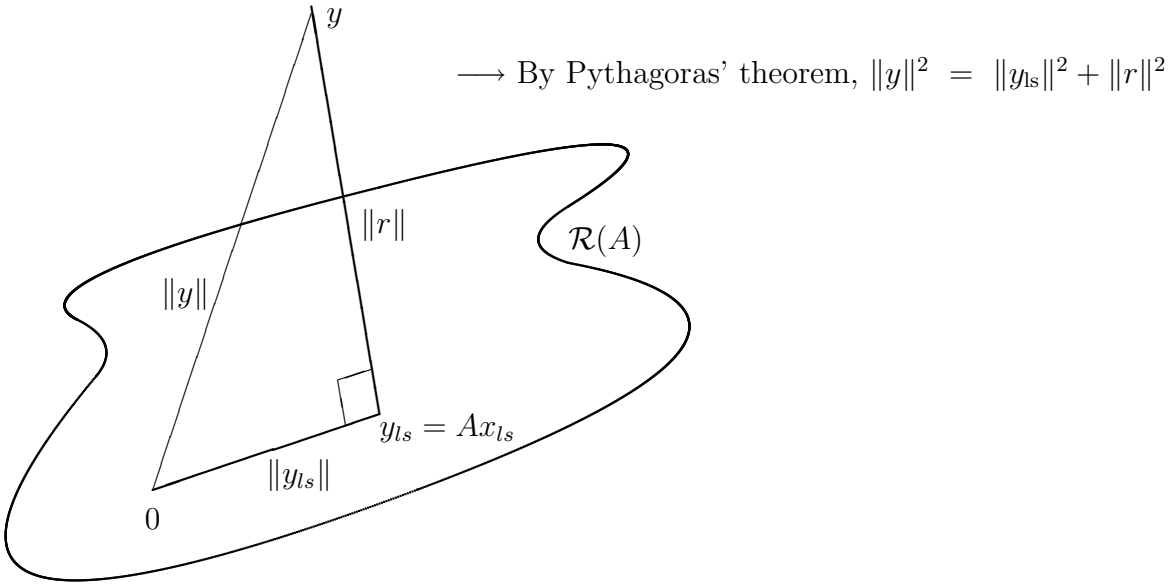
5.1 *Least-squares residuals.* Let the matrix A be skinny and full-rank. Let x_{ls} be the least-squares solution of $Ax = y$, and let $y_{\text{ls}} = Ax_{\text{ls}}$. Show that the residual vector $r = y - y_{\text{ls}}$ satisfies

$$\|r\|^2 = \|y\|^2 - \|y_{\text{ls}}\|^2.$$

Also, give a brief geometric interpretation of this equality (just a couple of sentences, and maybe a conceptual drawing). *Solution:*
Let us first show that $r \perp y_{\text{ls}}$. Since $y_{\text{ls}} = Ax_{\text{ls}} = AA^\dagger y = A(A^TA)^{-1}A^Ty$

$$\begin{aligned}{y_{\text{ls}}}^T r &= {y_{\text{ls}}}^T(y - y_{\text{ls}}) = {y_{\text{ls}}}^T y \;-\; {y_{\text{ls}}}^T y_{\text{ls}} \\ &= y^TA(A^TA)^{-1}A^Ty \;-\; y^TA(A^TA)^{-1}A^TA(A^TA)^{-1}A^Ty \\ &= y^TA(A^TA)^{-1}A^Ty \;-\; y^TA(A^TA)^{-1}(A^TA)(A^TA)^{-1}A^Ty \\ &= y^TA(A^TA)^{-1}A^Ty \;-\; y^TA(A^TA)^{-1}A^Ty \\ &= 0.\end{aligned}$$

Thus, $\|y\|^2 = \|y_{\text{ls}} + r\|^2 = (y_{\text{ls}} + r)^T(y_{\text{ls}} + r) = \|y_{\text{ls}}\|^2 + 2{y_{\text{ls}}}^T r + \|r\|^2 = \|y_{\text{ls}}\|^2 + \|r\|^2$. Therefore $\|r\|^2 = \|y\|^2 - \|y_{\text{ls}}\|^2$.



6.9 *Image reconstruction from line integrals.* In this problem we explore a simple version of a tomography problem. We consider a square region, which we divide into an $n \times n$ array of square pixels, as shown in Figure 1. The pixels are indexed column first, by a single

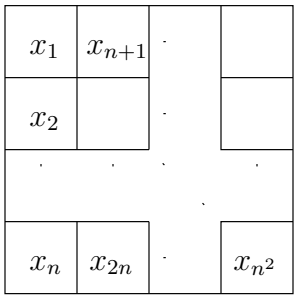


Figure 1: The square region and its division into pixels

index i ranging from 1 to n^2, as shown in Figure 1. We are interested in some physical property such as density (say) which varies over the region. To simplify things, we'll assume that the density is constant inside each pixel, and we denote by x_i the density

in pixel i, $i = 1, \ldots, n^2$. Thus, $x \in \mathbf{R}^{n^2}$ is a vector that describes the density across the rectangular array of pixels. The problem is to estimate the vector of densities x, from a set of sensor measurements that we now describe. Each sensor measurement is a *line integral* of the density over a line L. In addition, each measurement is corrupted by a (small) noise term. In other words, the sensor measurement for line L is given by

$$\sum_{i=1}^{n^2} l_i x_i + v,$$

where l_i is the length of the intersection of line L with pixel i (or zero if they don't intersect), and v is a (small) measurement noise. Figure 2 gives an example, with a graphical explanation of l_i. Now suppose we have N line integral measurements,

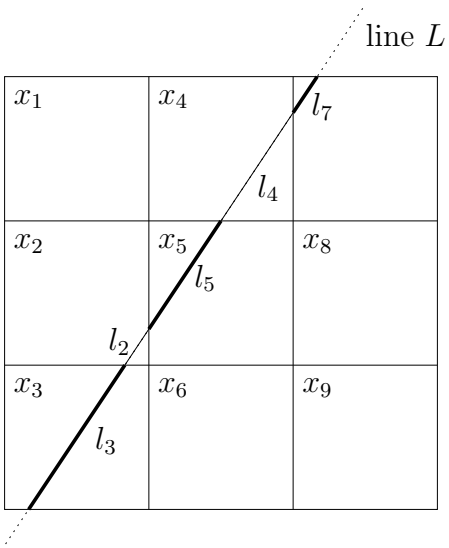


Figure 2: An example of a 3-by-3 pixel patch, with a line L and its intersections l_i with the pixels. Clearly, $l_1 = l_6 = l_8 = l_9 = 0$.

associated with lines $L_1, \ldots, L_N$. From these measurements, we want to estimate the vector of densities x. The lines are characterized by the intersection lengths

$$l_{ij}, \quad i = 1, \ldots, n^2, \quad j = 1, \ldots, N,$$

where l_{ij} gives the length of the intersection of line L_j with pixel i. Then, the whole set of measurements forms a vector $y \in \mathbf{R}^N$ whose elements are given by

$$y_j = \sum_{i=1}^{n^2} l_{ij} x_i + v_j, \quad j = 1, \ldots, N.$$

And now the problem: you will reconstruct the pixel densities x from the line integral measurements y. The class webpage contains the M-file `tomodata.m`, which you should download and run in Matlab. It creates the following variables:

- `N`, the number of measurements (N),
- `n_pixels`, the side length in pixels of the square region (n),
- `y`, a vector with the line integrals $y_j,\ j = 1, \ldots, N$,
- `lines_d`, a vector containing the displacement $d_j,\ j = 1, \ldots, N$, (distance from the center of the region in pixels lengths) of each line, and
- `lines_theta`, a vector containing the angles $\theta_j,\ j = 1, \ldots, N$, of each line.

The file `tmeasure.m`, on the same webpage, shows how the measurements were computed, in case you're curious. You should take a look, but you don't need to understand it to solve the problem. We also provide the function `line_pixel_length.m` on the webpage, which you do need to use in order to solve the problem. This function computes the pixel intersection lengths for a given line. That is, given d_j and θ_j (and the side length n), `line_pixel_length.m` returns a $n \times n$ matrix, whose i, jth element corresponds to the intersection length for pixel i, j on the image. Use this information to find x, and display it as an image (of n by n pixels). You'll know you have it right when the image of x forms a familiar acronym... *Matlab hints:* Here are a few functions that you'll find useful to display an image:

- `A=reshape(v,n,m)`, converts the vector `v` (which must have `n*m` elements) into an $n \times m$ matrix (the first column of `A` is the first n elements of `v`, *etc.*),
- `imagesc(A)`, displays the matrix `A` as an image, scaled so that its lowest value is black and its highest value is white,
- `colormap gray`, changes Matlab's image display mode to grayscaled (you'll want to do this to view the pixel patch),
- `axis image`, redefines the axes of a plot to make the pixels square.

Note: While irrelevant to your solution, this is actually a simple version of *tomography*, best known for its application in medical imaging as the CAT scan. If an *x-ray* gets attenuated at rate x_i in pixel i (a little piece of a cross-section of your body), the j-th measurement is

$$z_j = \prod_{i=1}^{n^2} e^{-x_i l_{ij}},$$

with the l_{ij} as before. Now define $y_j = -\log z_j$, and we get

$$y_j = \sum_{i=1}^{n^2} x_i l_{ij}.$$

Solution:
The first thing to do is to restate the problem in the familiar form $y = Ax + v$. Here, $y \in \mathbf{R}^N$ is the measurement (given), $x \in \mathbf{R}^{n^2}$ is the physical quantity we are interested in, $A \in \mathbf{R}^{N \times n^2}$ is the relation between them, and $v \in \mathbf{R}^N$ is the noise, or measurement

error (unknown, and we'll not worry about it). So we need to find the elements of A ... how do we do that?

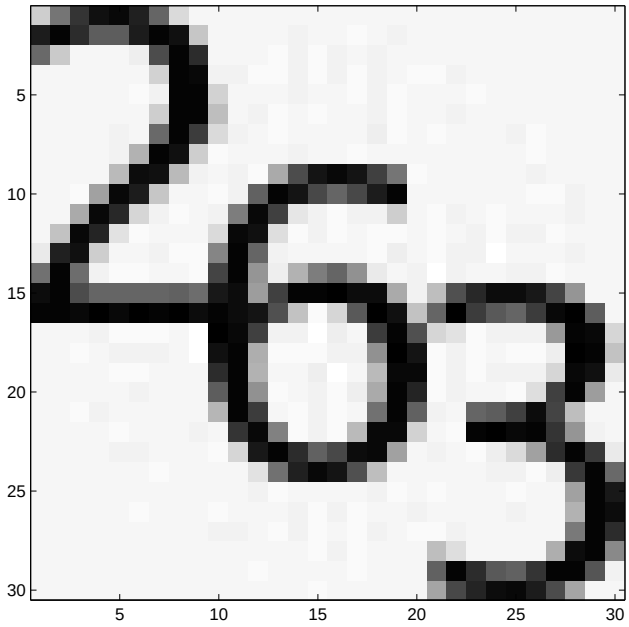
$$\text{Comparing} \quad y = Ax, \quad \textit{i.e.}, \quad y_j = \sum_{i=1}^{n^2} A_{ji} x_i \quad \text{with our model} \quad y_j = \sum_{i=1}^{n^2} l_{ij} x_i + v_j$$

it is clear that $A_{ji} = l_{ij}, \quad j = 1 \ldots N, \quad i = 1 \ldots n^2$. We have thus determined a standard linear model that we want to "invert" to find x. On running `tomodata.m`, we find that $n = 8$ and $N = 121$, so $N > n^2$, *i.e.*, we have more rows than columns – a skinny matrix. If A is full-rank the problem is overdetermined. We can find a unique (but not exact) solution x_{ls} – the least-squares solution that minimizes $\|Ax - y\|$ – which, due to its noise-reducing properties, provides a good estimate of x (it is the *best linear unbiased estimate*). So here's what we do: we construct A element for element by finding the length of the intersection of each line with each pixel. That's done using the function `line_pixel_length.m` provided. A turns out to be full-rank (`rank(A)` returns 64), so we can compute a unique x_{ls}. We go ahead and solve the least-squares problem, and then display the result.

Here comes a translation of the above paragraph into Matlab code:

```
% Load the problem data.
tomodata

%%%% Create the matrix A, then compute its elements.
A=zeros(N,n_pixels^2); for i=1:N
  % Compute the intersection lengths of line i with all pixels
  l=line_pixel_length(lines_d(i),lines_theta(i),n_pixels);
  A(i,:)=l'; % Those lengths become the i'th row of A
end

%%%% Check the rank of A
r=rank(A)    % Returns 900, i.e. n_pixels^2, i.e. number of
columns

%%%% Solve the least-squares problem: choose x to minimize
||y-Ax||
x_ls=A\y;

%%%% Reorder elements of vector x into a matrix X, for display.
X_ls=reshape(x_ls,n_pixels,n_pixels);

%%%% Display the reconstructed grayscale image.
figure(1) colormap gray imagesc(X_ls) axis image
```

And here's the end result, the reconstructed image



Solutions to additional exercises

1. *Simple fitting.* You are given some data $x_1, \ldots, x_N \in \mathbf{R}$ and $y_1, \ldots, y_N \in \mathbf{R}$. These data are available in `simplefitdata.m` on the course web site.

   (a) Find the best affine fit, *i.e.*, $y_i \approx ax_i + b$, where 'best' means minimizing $\sum_{i=1}^{N}(y_i - (ax_i + b))^2$. (This is often called the 'best linear fit'.) Set this up and solve it as a least-squares problem. Plot the data and the fit in the same figure. Give us a and b, and submit the code you used to find a and b.

   (b) Repeat for the best least-squares cubic fit, *i.e.*, $y_i \approx ax_i^3 + bx_i^2 + cx_i + d$.

   Solution. The following script builds the vectors x and y and caries out both parts of the exercise.

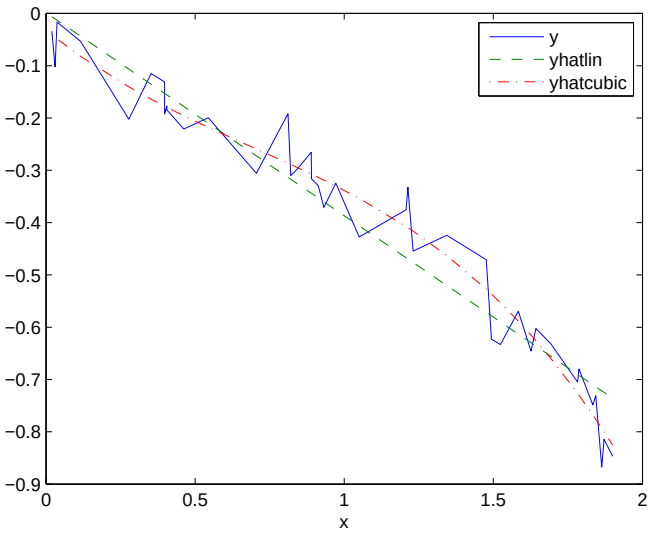
```
randn('state',0); % make repeatable
rand('state',0); % make repeatable
N=40; % number of points
x= 2*rand(N,1);  % uniform on [0,2]
x=sort(x);
y = 0.20-0.63*x; % linear trend
y = y -0.2*cos(2*x);
y = y +0.05*randn(N,1);
% first the linear
ab=[x ones(N,1)]\y;
a=ab(1), b=ab(2),
```

```
yhatlin = a*x+b;
% now the cubic
abcd=[x.^3 x.^2 x ones(N,1)]\y;
yhatcubic = [x.^3 x.^2 x ones(N,1)]*abcd;
plot(x,y,x,yhatlin,'r',x,yhatcubic,'g')
```

Running this script yields

```
a =
   -0.3881
b =
    0.0014
abcd =
   -0.1476
    0.3045
   -0.4633
   -0.0320
```

The figure below shows the data and the fit that we get in both parts a and b of the exercise.



2. *Estimating parameters from noisy measurements.* In this problem you will compare a least-squares estimate of a parameter vector (which uses all available measurements) with a method that uses just enough measurements. Carry out the following steps.

   (a) First we generate some problem data in Matlab. (You're free to use any other software system instead.) Generate a 50×20 matrix A using `A=randn(50,20)`.

(This chooses the entries from a normal distribution, but this doesn't really matter for us.) Generate a noise vector v of length 50 using `v=0.1*randn(50,1)`. Generate a measurement vector x of length 20 using `x=randn(20,1)`. Finally, generate a measurement vector $y = Ax + v$.

(b) Find the least-squares approximate solution of $y = Ax$, and call it x^{ls}. Find the relative error $\|x^{\text{ls}} - x\|/\|x\|$.

(c) Now form a 20-long truncated measurement vector y^{trunc} which consists of the first 20 entries of y. Form an estimate of x from y^{trunc}. Call this estimate x^{jem} ('Just Enough Measurements'). Find the relative error of x^{jem}.

(d) Run your script (*i.e.*, (a)–(c)) several times. You'll generate different different data each time, and you'll get different numerical results in parts (b) and (c). Give a one sentence comment about what you observe.

Note. Since you are generating the data randomly, it is remotely possible that the second method will work better than the first, at least for one run. If this happens to you, quickly run your script again. Do not mention the incident to anyone.

Solution. The following script carries out all parts of the problem.

```
randn('state',0);  % this makes the scirpt repeatable
K = 10;
% running the script K times
for i=1:K
    % Generating matrix A(50x20)
    A = randn(50,20);
    % Generating vector x(20x1)
    x = randn(20,1);
    % Generating noise vector & measurement
    v = 0.1*randn(50,1);
    y = A*x+v;
    % finding the least-squares approximate solution of y=Ax
    x_ls = A\y;
    rel_error_ls(i) = norm(x_ls-x)/norm(x);
    % the truncated measurement vector
    y_trunc=y(1:20,1);
    % truncated matrix A (the 20 first rows)
    A_trunc = A(1:20,:);
    % finding an estimate of x from y_trunc
    x_jem=A_trunc\y_trunc;  % this just inverts A_trunc!
    rel_error_jem(i) = norm(x_jem-x)/norm(x);
end
rel_error_ls
```

```
rel_error_jem
rel_error_ls_over_jem= rel_error_ls./rel_error_jem
```

Running this script yields

```
rel_error_ls =
  0.0196  0.0143  0.0212  0.0203  0.0168  0.0224  0.0223  0.0142    0.0202    0.013
rel_error_jem =
  41.7832  0.1557  0.1926  0.1599  0.2406  0.0693  0.2144  0.1117    0.0589    0.05
rel_error_ls_over_jem =
  0.0005  0.0918  0.1101  0.1270  0.0699  0.3225  0.1038  0.1273    0.3419    0.246
```

We see that in all cases the least-squares method does better, sometimes by a large factor.