Introduction to MATLAB

Eric Kostelich
MSRI Climate Change Summer School

July 14, 2008

The goal of these exercises is to familiarize you with the basics of MATLAB. You do not need any previous experience with it, but some previous computer programming experience is helpful.

Some basics

To start MATLAB, double-click the corresponding icon on your desktop. This brings up, among other things, a "command window" and a "command history." You can type in any command at the >> prompt.

MATLAB is a powerful programming language with many built-in functions for numerical computation. Unlike languages like C, variables require no declaration; they spring into existence when you first assign them a value and disappear when you type `clear`.

Practice Problem 1. You can assign w the value 150 with either of the commands

```
w = 150
```

or

```
w = 150;
```
note the semicolon

followed by the return key. What is the difference between these two commands?

Practice Problem 2. MATLAB has the usual built-in arithmetic operations. Multiplication is denoted by $*$ (shift-8), division by /, and exponentiation by ^ (shift-6). These operations follow these precedence rules:

highest	( )	(parentheses)
	ˆ	(exponentiation)
	$*, /$	(multiplication and division)
lowest	$+, -$	(addition and subtraction)

Operations of equal precedence are performed from left to right, except for exponentiation, which is performed from right to left. (For example, the expression $2+3*4$ is 14. Since $*$ has higher precedence than $+$, the multiplication is performed first.) Find the values of each of the following in MATLAB:

- $2*3$ˆ2
- $(2+3)*4$
- 2ˆ3ˆ4

Practice Problem 3. *Body mass index* (BMI) is defined as the ratio w/h^2, where your weight (well, mass) w is expressed in kilograms and height in meters. Determine the body mass index of a person who is 5 feet 9 inches tall and weighs 150 pounds. (One inch is 0.0254 meters exactly, and assume that 1 kilogram is 2.2 pounds. There are 12 inches in a foot.)

Matrices and vectors

MATLAB's main strength is the ease with which one can do sophisticated linear algebra computations. Here's one way to define the matrix

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad \text{and vector} \quad \begin{pmatrix} 1 \\ 1 \end{pmatrix}:$$

```
A = [ 1 2 ; 3 4 ]    spaces between the numbers
b = [ 1 1 ]'   note the prime (single quote)
```

Here **A** is defined rowwise; semicolons delimit each row. The notation [1 1] by itself denotes a row vector (i.e., a 1×2 matrix). The prime denotes transposition, so b as defined above is an ordinary column vector (2×1 matrix). We could just as well have defined **b** as

```
b = [ 1 ; 1 ]
```

MATLAB defines many functions that operate on matrices and vectors. Here are a few that you can try:

- The product $\mathbf{Ab}$ is denoted `A * b`
- Multiplication and division of vectors by scalars are defined in the obvious way. For example, `2 * b` is the vector $2\mathbf{b}$ and `b / 2` is the vector $\frac{1}{2}\mathbf{b}$.
- The product $\mathbf{A}^T\mathbf{A}$ is `A' * A`
- Given $\mathbf{x} = (x_1, \ldots, x_n)^T$, the expression `x.^2` yields $(x_1^2, \ldots, x_n^2)^T$, i.e., the squaring operation is applied elementwise
- `norm(b)` returns the Euclidean norm (2-norm) of $\mathbf{b}$
- `sum(b)` returns the sum of the elements of $\mathbf{b}$
- `det(A)` returns the determinant of $\mathbf{A}$
- `eig(A)` returns the eigenvalues of $\mathbf{A}$
- To solve the linear system $\mathbf{Ax} = \mathbf{b}$, simply say `x = A \ b`

Some syntax notes:

- To see the current value of the variable `x`, simply type `x`.
- Names in MATLAB are case sensitive: `AB` and `ab` are distinct identifiers.
- Vectors are indexed from 1: `x(1)` is the first element of the vector $\mathbf{x}$; `x(1:3)` is the 3-vector $(x_1, x_2, x_3)^T$ (assuming of course that $\mathbf{x}$ contains at least 3 elements). Similarly, `A(i,j)` is $\mathbf{A}_{ij}$, that is, the element in the ith row and jth column of $\mathbf{A}$.
- A *slice* is a submatrix of a given matrix. Examples:
  - `A(:,1)` is the first column of $\mathbf{A}$
  - `A(1,:)` is the first row of $\mathbf{A}$
  - `A(1:k,1:k)` is the upper left $k \times k$ submatrix of $\mathbf{A}$

- Use the up- and down-arrow keys to recall previous commands and the left- and right-arrow keys to edit a given command. When you are done editing, simply hit the return key; it is not necessary to move the cursor all the way to the right of the command.

- To learn more about the eigenvalue command, for example, simply type `help eig` (or type `eig` in the search box in the "Help Navigator" window). The Help window also contains links to a "Getting Started" guide (and many others).

Some matrix factorizations

In applications, it is often convenient to factor a given matrix $\mathbf{A}$ into a product of two or more matrices, each of which has a particular property. Here's a simple one: Suppose that the $n \times n$ matrix $\mathbf{A}$ has n distinct eigenvalues, with corresponding eigenvectors, that is, $\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i$, $i = 1, \ldots, n$. If $\mathbf{V}$ is the matrix whose ith column is $\mathbf{v}_i$, then

$$\mathbf{AV} = \mathbf{VD},$$

where $\mathbf{D}$ is the diagonal matrix such that $D_{ii} = \lambda_i$. Insofar as the eigenvectors corresponding to distinct eigenvalues are linearly independent, $\mathbf{V}$ is nonsingular, and so

$$\mathbf{A} = \mathbf{VDV}^{-1} \quad \text{and} \quad \mathbf{D} = \mathbf{V}^{-1}\mathbf{AV}.$$

We say that $\mathbf{V}$ *diagonalizes* $\mathbf{A}$.

Practice Problem 4. Let

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

as before.

- Run the command

  ```
  [ V, D ] = eig(A)
  ```
  comma after `V`

  to obtain the eigenvector decomposition of $\mathbf{A}$ as described above. (Notice that many built-in MATLAB functions return different results depending on the left-hand side of the assignment.)

- Verify that $\mathbf{A}\mathbf{v}_1 = \lambda_1 \mathbf{v}_1$ (up to roundoff) with the command

```
A*V(:,1) - D(1,1)*V(:,1)
```

and similarly for $\mathbf{v}_2$. Note: MATLAB writes a small vector like $(10^{-16}, 2 \times 10^{-16})^{\mathrm{T}}$ as

```
1.0e–16 *
1.0000
2.0000
```

- Verify that MATLAB returns eigenvectors as unit vectors by calling the norm function on the appropriate slices of V.

Gram-Schmidt orthogonalization

Given a linearly independent set of vectors $V = \{\mathbf{v}_1, \dots, \mathbf{v}_n\}$, the Gram-Schmidt algorithm provides a method to convert V to an orthonormal basis. Start by making $\mathbf{v}_1$ into a unit vector:

$$\mathbf{u}_1 = \frac{\mathbf{v}_1}{||\mathbf{v}_1||},$$

where $||\cdot||$ denotes the usual Euclidean norm (2-norm). We find $\mathbf{u}_2$ by subtracting off the component of $\mathbf{v}_2$ along $\mathbf{u}_1$ and normalizing the result:

$$\mathbf{u}_2 = \frac{\mathbf{v}_2 - \langle \mathbf{v}_2, \mathbf{u}_1 \rangle \mathbf{u}_1}{||\mathbf{v}_2 - \langle \mathbf{v}_2, \mathbf{u}_1 \rangle \mathbf{u}_1||}.$$

This process can be continued inductively. That is, $\mathbf{u}_3$ is obtained by subtracting off the component of $\mathbf{v}_3$ along the span of $\mathbf{u}_1$ and $\mathbf{u}_2$ and normalizing the result; and so on.

Practice Problem 5. Use appropriate MATLAB commands to apply the Gram-Schmidt algorithm to the columns of the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}.$$

Since we're working in $\mathbf{R}^2$, the inner product $\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^{\mathrm{T}}\mathbf{b} =$ `a' * b` in MATLAB.

QR decomposition

The Gram-Schmidt algorithm leads to another useful matrix factorization, called the *QR decomposition*. Unlike diagonalization by eigenvectors, the initial matrix $\mathbf{A}$ need not be square. Usually we are interested in the case where $\mathbf{A}$ has at least as many rows as columns, i.e., $\mathbf{A}$ is $m \times n$ where $m \geq n$. The *QR* factorization of $\mathbf{A}$ is

$$\mathbf{A} = \mathbf{QR},$$

where $\mathbf{Q}$ is an $m \times n$ matrix whose columns are the Gram-Schmidt orthogonalization of the columns of $\mathbf{A}$ and $\mathbf{R}$ is an $n \times n$ upper-triangular matrix. Because the columns of $\mathbf{Q}$ are orthonormal, $\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{I}_{n\times n}$. Furthermore,

$$\mathbf{A}^{\mathrm{T}}\mathbf{A} = (\mathbf{QR})^{\mathrm{T}}(\mathbf{QR}) = \mathbf{R}^{\mathrm{T}}(\mathbf{Q}^{\mathrm{T}}\mathbf{Q})\mathbf{R} = \mathbf{R}^{\mathrm{T}}\mathbf{R}.$$

The rightmost expression is the *Cholesky decomposition* of the $n \times n$ symmetric matrix $\mathbf{A}^{\mathrm{T}}\mathbf{A}$.

The normal equations for linear least squares

Suppose we are given a collection of data points of the form $\{(x_i, y_i)\}_{i=1}^{n}$. The simplest potential statistical relationship between x and y is that of a straight line; more precisely,

$$y_i = b_1 + b_2 x_i + \varepsilon_i, \tag{1}$$

where b_1 and b_2 give the y-intercept and slope, respectively, and ε_i is the “noise.” In the usual statistical theory, ε is assumed to be drawn from a normal distribution of mean 0 and variance σ^2. The mathematical problem is to find the values of b_1 and b_2 that “best fit” the observations.

Statisticians write the estimation problem in the following form:

$$\mathbf{y} = \mathbf{X}\beta + \varepsilon, \tag{2}$$

or, in components,

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}. \tag{3}$$

Here, "best fit" means to find β that minimizes the quantity $\|\mathbf{y}-\mathbf{X}\beta\|$, i.e., the square root of $\sum[y_i-(b_1+b_2x_i)]^2$ (hence the term *least-squares fit*).

It can be shown that the least-squares solution β of Eq. (2) satisfies the *normal equation*

$$\mathbf{X}^{\mathrm{T}}\mathbf{X}\beta = \mathbf{X}^{\mathrm{T}}\mathbf{y} \tag{4}$$

and that, in fact, this minimizer is unique.

Equation (1) can be extended to any polynomial model. For example, if we suppose that

$$y_i = b_1 + b_2x_i + b_3x_i^2 + \cdots + b_px_i^{p-1} + \varepsilon_i, \tag{5}$$

then we have to estimate the p unknowns $b_1,\ldots,b_p$. The normal equation is set up in exactly the same way, where $\mathbf{y}=(y_1,y_2,\ldots,y_n)^{\mathrm{T}}$ is an n-vector of observations $\beta=(b_1,b_2,\ldots,b_p)^{\mathrm{T}}$ is the p-vector of unknown parameters, ε is the n-vector of noises, and $\mathbf{X}$ is the $n\times p$ matrix

$$\mathbf{X} = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{p-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{p-1} \\ & & \vdots & & \\ 1 & x_n & x_n^2 & \cdots & x_n^{p-1} \end{pmatrix}.$$

It is straightforward to verify that the ith row of the matrix-vector product, plus the ith component of ε, is simply the right-hand side of (5).

Note. Other functional relationships can be modeled in a similar way. For example, the model $y_i = b_1e^{b_2x}$ can be fitted by taking the logarithm of each side, which turns the problem into the same form as (1).

The *QR* decomposition and least squares

The normal equation (4) is solved numerically by finding the *QR* decomposition of $\mathbf{X}$. If $\mathbf{X}$ has linearly independent columns, which is typically true in practice, then $\mathbf{R}$ is nonsingular, and we have

$$\begin{aligned} \mathbf{X}^{\mathrm{T}}\mathbf{X}\beta &= \mathbf{X}^{\mathrm{T}}\mathbf{y} \\ (\mathbf{QR})^{\mathrm{T}}(\mathbf{QR})\beta &= (\mathbf{QR})^{\mathrm{T}}\mathbf{y} \\ \mathbf{R}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{QR}\beta &= \mathbf{R}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{y} \\ \mathbf{R}^{\mathrm{T}}\mathbf{R}\beta &= \mathbf{R}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{y} \quad \text{since } \mathbf{Q}^{\mathrm{T}}\mathbf{Q}=\mathbf{I} \\ \mathbf{R}\beta &= \mathbf{Q}^{\mathrm{T}}\mathbf{y} \quad \text{since } \mathbf{R}^{\mathrm{T}} \text{ is nonsingular.} \end{aligned}$$

MATLAB can be used to solve Eq. (4) according to the following recipe:

1. Read in the n-vector of independent variables $\mathbf{x}$ and the n-vector of corresponding dependent variables $\mathbf{y}$.

2. Form the matrix $\mathbf{X}$. Given the n-vector $\mathbf{x}$, you can form the $n \times 2$ matrix $\mathbf{X}$ whose first column is all 1's and second column is $\mathbf{x}$ with `X = [ ones(n,1) x ]`. (Note: *no comma* before the x; just a space.)

3. Find the QR decomposition of $\mathbf{X}$:

   ```
   [ Q, R ] = qr(X, 0);
   ```

   Usually you'll want to include the semicolon to keep MATLAB from printing out all the results. (Since $\mathbf{X}$, and hence $\mathbf{Q}$, are $n \times p$, the Gram-Schmidt calculation can be extended to find an orthonormal basis of $\mathbf{R}^n$. However, we are only interested in the first p basis vectors. The 0 causes MATLAB to truncate the Gram-Schmidt algorithm after p steps.)

4. Define $\mathbf{z} = \mathbf{Q}^{\mathrm{T}}\mathbf{y}$.

5. Solve the linear system $\mathbf{R}\beta = \mathbf{z}$.

The components of β are the coefficients of the least-squares line (or polynomial).

Practice Problem 6. Fit the straight-line model, Eq. (1), to the three data points $(0,1)$, $(3,4)$, and $(6,5)$. Here

$$\mathbf{X} = \begin{pmatrix} 1 & 0 \\ 1 & 3 \\ 1 & 6 \end{pmatrix} \quad \text{and} \quad \mathbf{y} = \begin{pmatrix} 1 \\ 4 \\ 5 \end{pmatrix}.$$

(a) What are $\mathbf{Q}$ and $\mathbf{R}$?

(b) Verify that $\mathbf{Q}^{\mathrm{T}}\mathbf{Q}$ is the 2×2 identity matrix (up to roundoff).

(c) Use the recipe above to solve for β. (You should find $\beta = (\frac{4}{3}, \frac{2}{3})^{\mathrm{T}}$.)

Once you have found the coefficients b_1 and b_2, you can plot the least-squares line with the original data as follows:

```
yhat = b(1) + b(2) * X(:,2)
hold on
plot(y, X(:,2), '-o')
plot(yhat, X(:,2), '-r')
hold off
```

The `hold on` command concatenates all subsequent plot commands onto a single graph, and `hold off` turns off this feature. The `-o` option displays the original data as circles connected by a blue line. The `-r` option plots the least-squares line in solid red. (Type `help plot` or look up the `plot` command in the help window for a complete description of the available options.)

Practice Problem 7. Fit the quadratic model $y = b_1 + b_2 x + b_3 x^2$ to the data points $(0,3)$, $(1,2)$, $(2,4)$, and $(3,4)$. You should get $y = 2.75 - 0.25x + 0.25x^2$. (Simply set up $\mathbf{X}$ in the obvious way. All the remaining steps of the recipe are unchanged.) Plot the least-squares curve and the original data.

Notation

Suppose $\mathbf{y}$ is a vector and s is a scalar. I will write $\mathbf{y} - s$ to denote the vector that arises when s is subtracted from each component of $\mathbf{y}$. That is,

$$\text{if} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad \text{then} \quad \mathbf{y} - s = \begin{pmatrix} y_1 - s \\ y_2 - s \\ \vdots \\ y_n - s \end{pmatrix}.$$

We are particularly interested in the case where $s = \bar{y}$, the mean of the y_i's:

$$s = \bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i.$$

Analysis of variance

There are many statistical techniques to test whether, under the hypothesis that the noise is Gaussian, it is likely that there is a functional relationship between x and y. Here we will define some of the fundamental quantities of interest.

SSTO is the *total sum of squares*, defined as

$$\|\mathbf{y} - \bar{y}\|^2 = \sum_{i=1}^{n} (y_i - \bar{y})^2.$$

$\hat{\mathbf{y}}$ is the vector of y values that are *predicted* by the least-squares model, Eq. (1). That is, if $\hat{\beta} = (\hat{b}_1, \hat{b}_2)^{\mathrm{T}}$ solves Eq. (4), then

$$\hat{y}_i = \hat{b}_1 + \hat{b}_2 x_i,$$

which is expressed in matrix notation as $\hat{\mathbf{y}} = \mathbf{X}\hat{\beta}$.

residual vector is $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{X}\hat{\beta}$, i.e., the difference that is *unexplained* by the fitted model.

SSE is the *sum of squared errors*, defined as

$$\text{SSE} = \|\mathbf{r}\|^2 = \mathbf{r}^{\mathrm{T}}\mathbf{r}.$$

SSR is the *sum of squares of regression*, and gives the amount of the total deviation that is *explained* by the fitted model. It is defined as

$$\text{SSR} = \|\hat{\mathbf{y}} - \bar{y}\|^2.$$

In other words, the total variance is partitioned as

$$\text{SSTO} = \text{SSR} + \text{SSE}.$$

This partitioning is called the *analysis of variance* approach to regression, abbreviated ANOVA. The statistical analysis involves asking questions as to how likely it is that the partitioning obtained from a given least-squares fit is due to chance.

Here we'll discuss only the simplest idea, sometimes called the *sample coefficient of multiple determination*, denoted R^2. It is defined as

$$R^2 = \frac{\text{SSR}}{\text{SSTO}}.$$

Notice that $0 \leq R^2 \leq 1$. A value of 0 means that there is no correlation between x and y, and value of 1 means a perfect correlation, i.e., the fitted model exactly describes all the data.

All statistical analyses are based on the idea that the closer R^2 is to 1 with fewer fitted parameters, then the more likely it is that the functional relationship is *not* due to chance. It is always possible to make $R^2 = 1$ by fitting n data points to an n-parameter model. But the statistical tests impose a penalty for more parameters.

Practice Problem 8. Compute SSTO, SSR, and SSE for the data in Practice Problem 6.

Practice Problem 9. Compute SSTO, SSR, and SSE for the data in Practice Problem 7.

Practice Problem 10. Compute SSTO, SSR, SSE, and R^2 for a straight-line fit to the data $(1,y_1)$, $(2,y_2)$, …, $(100,y_{100})$ where the y's are random numbers with mean 0 and variance 1. To answer this question, generate the data as follows:

```
x  =  [1 : 1 : 100]'   note the prime
y  =  randn(100,1)     a column vector of 100 standard normal values
```

Then proceed as usual. What are the "expected" components of β if there is no functional relationship between x and y?

Practice Problem 11. Download the file co2.dat from the web page. This file contains two columns of numbers: the year from 1959 to 2007 and the average annual carbon dioxide concentration in the atmosphere (in parts per million), as measured at the Mauna Loa observatory in Hawaii. The data come from the Earth Science Research Laboratory of the National Oceanographic and Atmospheric Administration (which is the government agency charged with developing weather and climate forecast models. See www.esrl.noaa.gov/gmd/ccgg/trends for more complete datasets.) The estimated standard error in the observations is 0.12 ppm.

Save the file to the tmp directory on the C drive. Then read it into MATLAB with

```
dat = load('c:/tmp/co2.dat')
```

This creates the 49×2 array dat whose first column is the year and whose second column is the CO_2 concentration. Define t as time in years since 1959 and let $\mathbf{y}$ be the observed CO_2 concentrations. Fit the linear model $y = b_1 + b_2 t$ using the QR decomposition and compute SSTO, SSR, SSE, and R^2. Plot the least-squares line and the original data.

MATLAB hints. You can define a new vector t of times with

```
t = dat(:,1) − 1959;
```

and the vector y with

```
y = dat(:,2);
```

Be careful, however, to use a consistent time axis when you make your plots.

Practice Problem 12. Fit the exponential model $y = b_1 e^{b_2 t}$ to the CO_2 data and compute SSTO, SSR, SSE, and R^2. (Take the logarithm of each side, which yields the linear equation $\log y = (\log b_1) + b_2 t$. So you'll obtain $\log b_1$ and b_2 from a linear fit through the points $(\log y_i, t_i)$. Compute SSTO, SSR, SSE, and R^2.

Practice Problem 13. Fit the quadratic model $y = b_1 + b_2 t + b_3 t^2$ to the annual CO_2 data. Use the QR decomposition and compute SSTO, SSR, SSE, and R^2.

Practice Problem 14. Assuming (probably by a gross oversimplification!) that the models you have fit can be extrapolated to the future, when do your various fitted models predict that the annual average atmospheric CO_2 concentrations will reach 450 ppm?

Introduction to SVD and Applications

Eric Kostelich and Dave Kuhl
MSRI Climate Change Summer School

July 18, 2008

Introduction

The goal of this exercise is to familiarize you with the basics of the singular value decomposition (SVD). The SVD is known by many names, such as *principal component analysis*. It has many uses. In numerical analysis, the SVD provides a measure of the effective rank of a given matrix. In statistics and time series analysis, the SVD a particularly useful tool for finding least-squares solutions and approximations.

Don't worry if you don't finish all the exercises today. However, I strongly encourage you to complete them over the weekend. Monday's lectures will be easier to understand once you have some geometric intuition about the SVD.

Let $\mathbf{A}$ be an $m \times n$ real matrix; m and n may be any positive integers. The SVD of $\mathbf{A}$ is the factorization

$$\mathbf{A}_{m\times n} = \mathbf{U}_{m\times n}\mathbf{S}_{n\times n}\mathbf{V}^{\mathrm{T}}_{n\times n}, \tag{1}$$

where $\mathbf{S} = \operatorname{diag}(s_1, \ldots, s_n)$. The s_i's are the *singular values* of $\mathbf{A}$. By convention, they are ordered so that $s_1 \geq s_2 \geq \cdots \geq s_n \geq 0$. The columns of $\mathbf{U}$ and $\mathbf{V}$ may be chosen so that they form an orthonormal basis of the column space and row space, respectively, of $\mathbf{A}$. If $\mathbf{A}$ has full rank, then its singular values are all positive, and when they are ordered as indicated, then the SVD is unique up to the signs of the columns of $\mathbf{U}$ and $\mathbf{V}$.

Note. There are other essentially equivalent formulations of Eq. (1). For example, when $m \geq n$ and $\operatorname{rank}\mathbf{A} = n$, it is possible to extend $\mathbf{U}$ to be an $m \times m$ matrix whose columns form an orthonormal basis for $\mathbf{R}^m$ (and whose first n columns span the column space of $\mathbf{A}$). In this case, $\mathbf{S}$ may be taken to be an $m \times n$ matrix such that $S_{ii} = s_i$, $i = 1, \ldots, n$, and whose other entries are zero.

All of this can be extended to a general $m \times n$ complex matrix $\mathbf{A}$. You can find more details in most any textbook on advanced linear algebra and on a variety of Web sites. We'll stick to real matrices for today.

The decomposition in Eq. (1) implies that

$$\begin{aligned} \mathbf{A}^{\mathrm{T}}\mathbf{A} &= (\mathbf{USV}^{\mathrm{T}})^{\mathrm{T}}(\mathbf{USV}^{\mathrm{T}}) \\ &= \mathbf{VSU}^{\mathrm{T}}\mathbf{USV}^{\mathrm{T}} \\ &= \mathbf{VS}^2\mathbf{V}^{\mathrm{T}} \quad \text{since } \mathbf{U} \text{ is orthonormal and } \mathbf{S} \text{ is diagonal.} \end{aligned}$$

Also, because $\mathbf{V}^{\mathrm{T}} = \mathbf{V}^{-1}$, we have $(\mathbf{A}^{\mathrm{T}}\mathbf{A})\mathbf{V} = \mathbf{VS}^2$, which shows that the columns of $\mathbf{V}$ are eigenvectors of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$. The singular values of $\mathbf{A}$ are the square roots of the corresponding eigenvalues.

Practice Problem 1. Show that the columns of $\mathbf{U}$ are eigenvectors of $\mathbf{AA}^{\mathrm{T}}$.

Practice Problem 2. Consider the linear least-squares problem, where we fit a p-parameter model of the form

$$y_i = b_1 + b_2 x + \cdots + b_p x^{p-1} \tag{2}$$

to a collection of data points $\{(x_i, y_i)\}_{i=1}^{n}$. Equation (2) can be written in matrix-vector form as

$$\mathbf{y} = \mathbf{X}\beta = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 & \cdots & x_1^{p-1} \\ 1 & x_2 & \cdots & x_2^{p-1} \\ & & \vdots & \\ 1 & x_n & \cdots & x_n^{p-1} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{pmatrix}.$$

The least-squares solution is given by the solution to the *normal equation*

$$(\mathbf{X}^{\mathrm{T}}\mathbf{X})\beta = \mathbf{X}^{\mathrm{T}}\mathbf{y}. \tag{3}$$

Assuming that $n \geq p$ and $\operatorname{rank}\mathbf{X} = p$, show that

$$\beta = \mathbf{VS}^{-1}\mathbf{U}^{\mathrm{T}}\mathbf{y}$$

where $\mathbf{X} = \mathbf{USV}^{\mathrm{T}}$ is the SVD of $\mathbf{X}$. Hence the SVD is an alternative to the QR decomposition for solving linear least-squares problems.

Matrix times circle equals ellipse

Practice Problem 3. Generate a circle of points with 1° separation as follows (mind the semicolons!):

```
circ = zeros(2, 360);   a 2 × 360 zero array
deg = [ 1 : 1 : 360 ];
circ(1, :) = cos(deg * pi / 180);   pi is a built-in constant
circ(2, :) = sin(deg * pi / 180);
```

Now define **A** to be your favorite nonsingular 2×2 matrix (choose $\mathbf{A} \neq \mathbf{I}$ to keep things interesting). The syntax

```
A = [ a b ; c d ]
```

defines $\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. Compute the SVD of **A**:

```
[ U S V ] = svd(A)
```

Now plot $\mathbf{A} \times$ circle as follows:

```
hold on   concatenate all subsequent plots on the same graph
axis equal
plot(circ(1, :), circ(2, :), 'o')   no semicolon
Acirc = A * circ;   semicolon!
plot(Acirc(1, :), Acirc(2, :), 'o')   no semicolon
```

The major and minor axes of the ellipse are given by the eigenvectors of $\mathbf{AA}^{\mathrm{T}}$, that is, the columns of **U**. Add them to your plot as follows:

```
Uline = [ U(:,1)*S(1,1)  [0 0]'  U(:,2)*S(2,2) ];   note the prime
plot(Uline(1, :), Uline(2, :), '–r')
hold off
```

Rank-*r* approximations

Often it is convenient to approximate a given full-rank matrix **A** by a matrix $\hat{\mathbf{A}}$ of less than full rank. The approximation $\hat{\mathbf{A}}$ is obtained by Eq. (1) where the smallest $n - r$ singular values in **S** are replaced with 0. Equivalently, we may write

$$\hat{\mathbf{A}} = \hat{\mathbf{U}}_{m\times r}\hat{\mathbf{S}}_{r\times r}\hat{\mathbf{V}}^{\mathrm{T}}_{r\times n} \tag{4}$$

that is, the product of the first r columns of $\mathbf{U}$, the upper $r \times r$ block of $\mathbf{S}$, and the first r columns of $\mathbf{V}$. (In other words, for every $\mathbf{x}$, we project $\mathbf{Ax}$ onto the subspace spanned by the first r columns of $\mathbf{U}$.)

The notion of a rank-r approximation is a key idea in the data assimilation algorithm that we'll discuss next week. One goal today is to illustrate that r need not be very large to get a good approximation. The example that we'll consider gives an image compression algorithm.

Step 1. Download the file dog.jpg from the Web page to the your working MATLAB directory. Then load it into Matlab with the command

```
A = imread('./dog.jpg');    semicolon!
```

The semincolon is necessary so that MATLAB does not print out many screenfuls of data. The result is a 310×338 matrix of grayscale values corresponding to a black and white picture of a dog. (The matrix has 104,780 entries.)

Step 2. We need to do some initial processing. Type

```
B = double(A(:, :, 1)) + 1;    semicolon!
```

which converts A into the double-precision format that is needed for the singular value decomposition. Now say

```
B = B / 256;    semicolon!
[ U S V ] = svd(B);    semicolon!
```

The gray scale goes from 0 to 256 in a black-and-white JPEG image. We divide $\mathbf{B}$ by 256 to obtain values between 0 and 1, which is required for the MATLAB imaging routines that we'll use later.

Practice Problem 4. What are the dimensions of $\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}$?

Here $\mathbf{S}$ has more columns than rows; in fact, columns 311 to 338 are all zeros. (When $m < n$, we pad $\mathbf{S}$ on the right with zero columns to turn $\mathbf{S}$ into an $m \times n$ matrix rather than an $n \times n$ matrix.)

Practice Problem 5. Compute the best rank-1 approximation to $\mathbf{B}$ and store it in the matrix rank1. (See Eq. (4). The matrix slice consisting of the first r columns of $\mathbf{U}$ is denoted U(:, 1:r) in MATLAB. The upper $r \times r$ portion of $\mathbf{S}$ is S(1:r, 1:r), and so on.)

Step 3. Let's visualize rank1. To do that, first create

```
C = zeros(size(A));    semicolon!
```

This creates an array of zeroes, **C**, of the same dimensions as the original image matrix **A**. Now **A** is actually $310 \times 338 \times 3$. To create an image from the matrix **C**, MATLAB uses $\mathbf{C}(i, j, 1:3)$ as the RGB values that color the ijth pixel. We have a black-and-white photo, so we'll set all the RGB values to be the same, namely $1/2$:

```
C(:,:,:) = 0.5;    semicolon!
```

Step 4. Copy the rank-1 image into **C** as follows:

```
C(:,:,1) = rank1;    semicolons on all of these
C(:,:,2) = rank1;
C(:,:,3) = rank1;
```

Step 5. We're almost done, except for one hitch. MATLAB does all its scaling using values from 0 to 1 (and maps them to values between 0 and 256 for the graphics hardware). Lower-rank approximations to the actual image can have values that are slightly less than 0 and greater than 1. So we'll truncate them to fit, as follows:

```
C(:,:,:) = min(1, C(:,:,:));    semicolon!
C(:,:,:) = max(0, C(:,:,:));    semicolon!
```

Step 6. View the resulting image:

```
image(C)    no semicolon
```

Practice Problem 6. Create and view a rank-10 approximation to the original picture. (Use Steps 4–6 but with `rank10` instead of `rank1`. If you mess up—for example, you get an all-black picture—then start over from Step 3.)

Practice Problem 7. Repeat with approximations of rank 20, 30, and 40 (and any others that you'd like to experiment with). What is the smallest rank that, in your opinion, gives an acceptable approximation to the original picture?

Practice Problem 8. What rank-r approximation exactly reproduces the original picture?