

Chapter 10

Eigenvalues and Singular Values

This chapter is about eigenvalues and singular values of matrices. Computational algorithms and sensitivity to perturbations are both discussed.

10.1 Eigenvalue and Singular Value Decompositions

An *eigenvalue* and *eigenvector* of a square matrix A are a scalar λ and a nonzero vector x so that

$$Ax = \lambda x.$$

A *singular value* and pair of *singular vectors* of a square or rectangular matrix A are a nonnegative scalar σ and two nonzero vectors u and v so that

$$\begin{aligned} Av &= \sigma u, \\ A^H u &= \sigma v. \end{aligned}$$

The superscript on A^H stands for *Hermitian transpose* and denotes the complex conjugate transpose of a complex matrix. If the matrix is real, then A^T denotes the same matrix. In MATLAB, these transposed matrices are denoted by \mathbf{A}' .

The term “eigenvalue” is a partial translation of the German “eigenwert.” A complete translation would be something like “own value” or “characteristic value,” but these are rarely used. The term “singular value” relates to the distance between a matrix and the set of singular matrices.

Eigenvalues play an important role in situations where the matrix is a transformation from one vector space onto itself. Systems of linear ordinary differential equations are the primary examples. The values of λ can correspond to frequencies of vibration, or critical values of stability parameters, or energy levels of atoms. Singular values play an important role where the matrix is a transformation from one vector space to a different vector space, possibly with a different dimension. Systems of over- or underdetermined algebraic equations are the primary examples.

The definitions of eigenvectors and singular vectors do not specify their normalization. An eigenvector x , or a pair of singular vectors u and v , can be scaled by any nonzero factor without changing any other important properties. Eigenvectors of symmetric matrices are usually normalized to have Euclidean length equal to one, $\|x\|_2 = 1$. On the other hand, the eigenvectors of nonsymmetric matrices often have different normalizations in different contexts. Singular vectors are almost always normalized to have Euclidean length equal to one, $\|u\|_2 = \|v\|_2 = 1$. You can still multiply eigenvectors, or pairs of singular vectors, by -1 without changing their lengths.

The eigenvalue-eigenvector equation for a square matrix can be written

$$(A - \lambda I)x = 0, \quad x \neq 0.$$

This implies that $A - \lambda I$ is singular and hence that

$$\det(A - \lambda I) = 0.$$

This definition of an eigenvalue, which does not directly involve the corresponding eigenvector, is the *characteristic equation* or *characteristic polynomial* of A . The degree of the polynomial is the order of the matrix. This implies that an n -by- n matrix has n eigenvalues, counting multiplicities. Like the determinant itself, the characteristic polynomial is useful in theoretical considerations and hand calculations, but does not provide a sound basis for robust numerical software.

Let $\lambda_1, \lambda_2, \dots, \lambda_n$ be the eigenvalues of a matrix A , let x_1, x_2, \dots, x_n be a set of corresponding eigenvectors, let Λ denote the n -by- n diagonal matrix with the λ_j on the diagonal, and let X denote the n -by- n matrix whose j th column is x_j . Then

$$AX = X\Lambda.$$

It is necessary to put Λ on the right in the second expression so that each column of X is multiplied by its corresponding eigenvalue. Now make a key assumption that is not true for all matrices—assume that the eigenvectors are linearly independent. Then X^{-1} exists and

$$A = X\Lambda X^{-1},$$

with nonsingular X . This is known as the *eigenvalue decomposition* of the matrix A . If it exists, it allows us to investigate the properties of A by analyzing the diagonal matrix Λ . For example, repeated matrix powers can be expressed in terms of powers of scalars:

$$A^p = X\Lambda^p X^{-1}.$$

If the eigenvectors of A are not linearly independent, then such a diagonal decomposition does not exist and the powers of A exhibit a more complicated behavior.

If T is any nonsingular matrix, then

$$B = T^{-1}AT$$

is known as a *similarity transformation* and A and B are said to be *similar*. If $Ax = \lambda x$ and $y = Tx$, then $By = \lambda y$. In other words, a similarity transformation preserves eigenvalues. The eigenvalue decomposition is an attempt to find a similarity transformation to diagonal form.

Written in matrix form, the defining equations for singular values and vectors are

$$\begin{aligned} AV &= U\Sigma, \\ A^H U &= V\Sigma^H. \end{aligned}$$

Here Σ is a matrix the same size as A that is zero except possibly on its main diagonal. It turns out that singular vectors can always be chosen to be perpendicular to each other, so the matrices U and V , whose columns are the normalized singular vectors, satisfy $U^H U = I$ and $V^H V = I$. In other words, U and V are *orthogonal* if they are real, or *unitary* if they are complex. Consequently,

$$A = U\Sigma V^H,$$

with diagonal Σ and orthogonal or unitary U and V . This is known as the *singular value decomposition*, or *SVD*, of the matrix A .

In abstract linear algebra terms, eigenvalues are relevant if a square, n -by- n matrix A is thought of as mapping n -dimensional space onto itself. We try to find a basis for the space so that the matrix becomes diagonal. This basis might be complex even if A is real. In fact, if the eigenvectors are not linearly independent, such a basis does not even exist. The SVD is relevant if a possibly rectangular, m -by- n matrix A is thought of as mapping n -space onto m -space. We try to find one change of basis in the domain and a usually different change of basis in the range so that the matrix becomes diagonal. Such bases always exist and are always real if A is real. In fact, the transforming matrices are orthogonal or unitary, so they preserve lengths and angles and do not magnify errors.

If A is m by n with m larger than n , then in the full SVD, U is a large, square m -by- m matrix. The last $m - n$ columns of U are “extra”; they are not needed

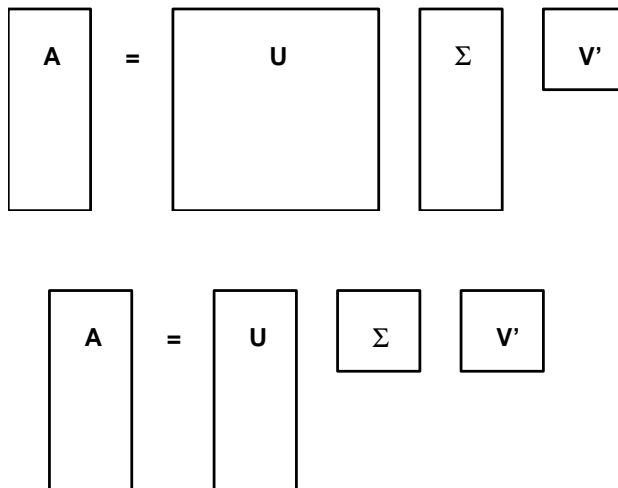


Figure 10.1. Full and economy SVDs.

to reconstruct A . A second version of the SVD that saves computer memory if A is rectangular is known as the *economy-sized* SVD. In the economy version, only the first n columns of U and first n rows of Σ are computed. The matrix V is the same n -by- n matrix in both decompositions. Figure 10.1 shows the shapes of the various matrices in the two versions of the SVD. Both decompositions can be written $A = U\Sigma V^H$, even though the U and Σ in the economy decomposition are submatrices of the ones in the full decomposition.

10.2 A Small Example

An example of the eigenvalue and singular value decompositions of a small, square matrix is provided by one of the test matrices from the MATLAB gallery.

```
A = gallery(3)
```

The matrix is

$$A = \begin{pmatrix} -149 & -50 & -154 \\ 537 & 180 & 546 \\ -27 & -9 & -25 \end{pmatrix}.$$

This matrix was constructed in such a way that the characteristic polynomial factors nicely:

$$\begin{aligned} \det(A - \lambda I) &= \lambda^3 - 6\lambda^2 + 11\lambda - 6 \\ &= (\lambda - 1)(\lambda - 2)(\lambda - 3). \end{aligned}$$

Consequently, the three eigenvalues are $\lambda_1 = 1$, $\lambda_2 = 2$, and $\lambda_3 = 3$, and

$$\Lambda = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}.$$

The matrix of eigenvectors can be normalized so that its elements are all integers:

$$X = \begin{pmatrix} 1 & -4 & 7 \\ -3 & 9 & -49 \\ 0 & 1 & 9 \end{pmatrix}.$$

It turns out that the inverse of X also has integer entries:

$$X^{-1} = \begin{pmatrix} 130 & 43 & 133 \\ 27 & 9 & 28 \\ -3 & -1 & -3 \end{pmatrix}.$$

These matrices provide the eigenvalue decomposition of our example:

$$A = X\Lambda X^{-1}.$$

The SVD of this matrix cannot be expressed so neatly with small integers. The singular values are the positive roots of the equation

$$\sigma^6 - 668737\sigma^4 + 4096316\sigma^2 - 36 = 0,$$

but this equation does not factor nicely. The Symbolic Toolbox statement

```
svd(sym(A))
```

returns exact formulas for the singular values, but the overall length of the result is 822 characters. So we compute the SVD numerically.

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

produces

```
U =
   -0.2691   -0.6798    0.6822
    0.9620   -0.1557    0.2243
   -0.0463    0.7167    0.6959

S =
  817.7597         0         0
         0    2.4750         0
         0         0    0.0030

V =
    0.6823   -0.6671    0.2990
    0.2287   -0.1937   -0.9540
    0.6944    0.7193    0.0204
```

The expression $U*S*V'$ generates the original matrix to within roundoff error.

For `gallery(3)`, notice the big difference between the eigenvalues, 1, 2, and 3, and the singular values, 817, 2.47, and 0.003. This is related, in a way that we will make more precise later, to the fact that this example is very far from being a symmetric matrix.

10.3 eigshow

The function `eigshow` is available in the MATLAB `demons` directory. The input to `eigshow` is a real, 2-by-2 matrix A , or you can choose an A from a pull-down list in the title. The default A is

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

Initially, `eigshow` plots the unit vector $x = [1, 0]'$, as well as the vector Ax , which starts out as the first column of A . You can then use your mouse to move x , shown in green, around the unit circle. As you move x , the resulting Ax , shown in blue, also moves. The first four subplots in Figure 10.2 show intermediate steps as x traces out a green unit circle. What is the shape of the resulting orbit of Ax ? An important, and nontrivial, theorem from linear algebra tells us that the blue curve is an ellipse. `eigshow` provides a “proof by GUI” of this theorem.

The caption for `eigshow` says “Make Ax parallel to x .” For such a direction x , the operator A is simply a stretching or magnification by a factor λ . In other words, x is an eigenvector and the length of Ax is the corresponding eigenvalue.

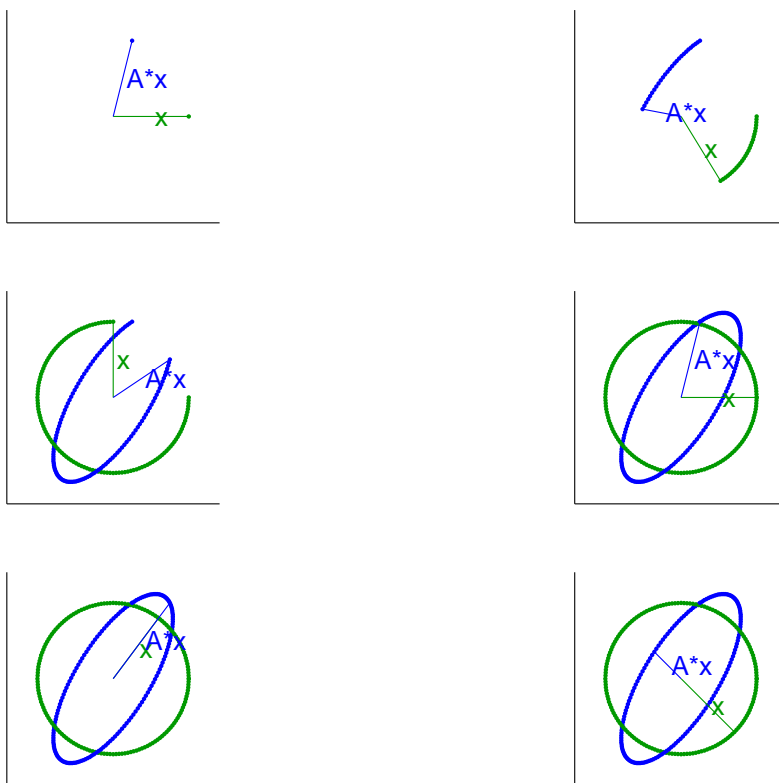


Figure 10.2. eigshow.

The last two subplots in Figure 10.2 show the eigenvalues and eigenvectors of our 2-by-2 example. The first eigenvalue is positive, so Ax lies on top of the eigenvector x . The length of Ax is the corresponding eigenvalue; it happens to be $5/4$ in this example. The second eigenvalue is negative, so Ax is parallel to x , but points in the opposite direction. The length of Ax is $1/2$, and the corresponding eigenvalue is actually $-1/2$.

You might have noticed that the two eigenvectors are not the major and minor axes of the ellipse. They would be if the matrix were symmetric. The default `eigshow` matrix is close to, but not exactly equal to, a symmetric matrix. For other matrices, it may not be possible to find a real x so that Ax is parallel to x . These examples, which we pursue in the exercises, demonstrate that 2-by-2 matrices can have fewer than two real eigenvectors.

The axes of the ellipse do play a key role in the SVD. The results produced by the `svd` mode of `eigshow` are shown in Figure 10.3. Again, the mouse moves x around the unit circle, but now a second unit vector, y , follows x , staying perpendicular to it. The resulting Ax and Ay traverse the ellipse, but are not usually perpendicular to each other. The goal is to make them perpendicular. If they are,

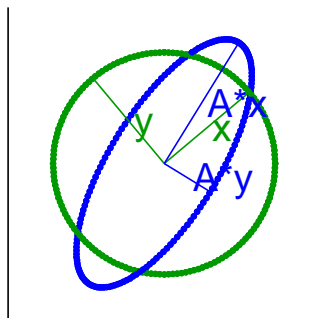


Figure 10.3. eigshow(svd).

they form the axes of the ellipse. The vectors x and y are the columns of U in the SVD, the vectors Ax and Ay are multiples of the columns of V , and the lengths of the axes are the singular values.

10.4 Characteristic Polynomial

Let A be the 20-by-20 diagonal matrix with $1, 2, \dots, 20$ on the diagonal. Clearly, the eigenvalues of A are its diagonal elements. However, the characteristic polynomial $\det(A - \lambda I)$ turns out to be

$$\begin{aligned} & \lambda^{20} - 210\lambda^{19} + 20615\lambda^{18} - 1256850\lambda^{17} + 53327946\lambda^{16} \\ & -1672280820\lambda^{15} + 40171771630\lambda^{14} - 756111184500\lambda^{13} \\ & +11310276995381\lambda^{12} - 135585182899530\lambda^{11} \\ & +1307535010540395\lambda^{10} - 10142299865511450\lambda^9 \\ & +63030812099294896\lambda^8 - 311333643161390640\lambda^7 \\ & +1206647803780373360\lambda^6 - 3599979517947607200\lambda^5 \\ & +8037811822645051776\lambda^4 - 12870931245150988800\lambda^3 \\ & +13803759753640704000\lambda^2 - 8752948036761600000\lambda \\ & +2432902008176640000. \end{aligned}$$

The coefficient of $-\lambda^{19}$ is 210, which is the sum of the eigenvalues. The coefficient of λ^0 , the constant term, is $20!$, which is the product of the eigenvalues. The other coefficients are various sums of products of the eigenvalues.

We have displayed all the coefficients to emphasize that doing any floating-point computation with them is likely to introduce large roundoff errors. Merely representing the coefficients as IEEE floating-point numbers changes five of them. For example, the last 3 digits of the coefficient of λ^4 change from 776 to 392. To 16 significant digits, the exact roots of the polynomial obtained by representing the coefficients in floating point are as follows.

```

1.000000000000000
2.000000000000096
2.99999999986640
4.00000000495944
4.99999991473414
6.00000084571661
6.99999455544845
8.00002443256894
8.99992001186835
10.00019696490537
10.99962843024064
12.00054374363591
12.99938073455790
14.00054798867380
14.99962658217055
16.00019208303847
16.99992773461773
18.00001875170604
18.99999699774389
20.00000022354640

```

We see that just storing the coefficients in the characteristic polynomial as double-precision floating-point numbers changes the computed values of some of the eigenvalues in the fifth significant digit.

This particular polynomial was introduced by J. H. Wilkinson around 1960. His perturbation of the polynomial was different than ours, but his point was the same, namely that representing a polynomial in its power form is an unsatisfactory way to characterize either the roots of the polynomial or the eigenvalues of the corresponding matrix.

10.5 Symmetric and Hermitian Matrices

A real matrix is symmetric if it is equal to its transpose, $A = A^T$. A complex matrix is Hermitian if it is equal to its complex conjugate transpose, $A = A^H$. The eigenvalues and eigenvectors of a real symmetric matrix are real. Moreover, the matrix of eigenvectors can be chosen to be orthogonal. Consequently, if A is real and $A = A^T$, then its eigenvalue decomposition is

$$A = X\Lambda X^T,$$

with $X^T X = I = X X^T$. The eigenvalues of a complex Hermitian matrix turn out to be real, although the eigenvectors must be complex. Moreover, the matrix of eigenvectors can be chosen to be unitary. Consequently, if A is complex and $A = A^H$, then its eigenvalue decomposition is

$$A = X\Lambda X^H,$$

with Λ real and $X^H X = I = X X^H$.

For symmetric and Hermitian matrices, the eigenvalues and singular values are obviously closely related. A nonnegative eigenvalue, $\lambda \geq 0$, is also a singular value, $\sigma = \lambda$. The corresponding vectors are equal to each other, $u = v = x$. A negative eigenvalue, $\lambda < 0$, must reverse its sign to become a singular value, $\sigma = |\lambda|$. One of the corresponding singular vectors is the negative of the other, $u = -v = x$.

10.6 Eigenvalue Sensitivity and Accuracy

The eigenvalues of some matrices are sensitive to perturbations. Small changes in the matrix elements can lead to large changes in the eigenvalues. Roundoff errors introduced during the computation of eigenvalues with floating-point arithmetic have the same effect as perturbations in the original matrix. Consequently, these roundoff errors are magnified in the computed values of sensitive eigenvalues.

To get a rough idea of this sensitivity, assume that A has a full set of linearly independent eigenvectors and use the eigenvalue decomposition

$$A = X\Lambda X^{-1}.$$

Rewrite this as

$$\Lambda = X^{-1}AX.$$

Now let δA denote some change in A , caused by roundoff error or any other kind of perturbation. Then

$$\Lambda + \delta\Lambda = X^{-1}(A + \delta A)X.$$

Hence

$$\delta\Lambda = X^{-1}\delta AX.$$

Taking matrix norms,

$$\|\delta\Lambda\| \leq \|X^{-1}\| \|X\| \|\delta A\| = \kappa(X) \|\delta A\|,$$

where $\kappa(X)$ is the matrix condition number introduced in Chapter 2, Linear Equations. Note that the key factor is the condition of X , the matrix of eigenvectors, not the condition of A itself.

This simple analysis tells us that, in terms of matrix norms, a perturbation $\|\delta A\|$ can be magnified by a factor as large as $\kappa(X)$ in $\|\delta\Lambda\|$. However, since $\delta\Lambda$ is usually not a diagonal matrix, this analysis does not immediately say how much the eigenvalues themselves may be affected. Nevertheless, it leads to the correct overall conclusion:

The sensitivity of the eigenvalues is estimated by the condition number of the matrix of eigenvectors.

You can use the function `condest` to estimate the condition number of the eigenvector matrix. For example,

```
A = gallery(3)
[X,lambda] = eig(A);
condest(X)
```

yields

1.2002e+003

A perturbation in `gallery(3)` could result in perturbations in its eigenvalues that are $1.2 \cdot 10^3$ times as large. This says that the eigenvalues of `gallery(3)` are slightly badly conditioned.

A more detailed analysis involves the *left eigenvectors*, which are *row* vectors y^H that satisfy

$$y^H A = \lambda y^H.$$

In order to investigate the sensitivity of an individual eigenvalue, assume that A varies with a perturbation parameter and let \dot{A} denote the derivative with respect to that parameter. Differentiate both sides of the equation

$$Ax = \lambda x$$

to get

$$\dot{A}x + A\dot{x} = \dot{\lambda}x + \lambda\dot{x}.$$

Multiply through by the left eigenvector:

$$y^H \dot{A}x + y^H A\dot{x} = y^H \dot{\lambda}x + y^H \lambda\dot{x}.$$

The second terms on each side of this equation are equal, so

$$\dot{\lambda} = \frac{y^H \dot{A}x}{y^H x}.$$

Taking norms,

$$|\dot{\lambda}| \leq \frac{\|y\| \|x\|}{y^H x} \|\dot{A}\|.$$

Define the *eigenvalue condition number* to be

$$\kappa(\lambda, A) = \frac{\|y\| \|x\|}{y^H x}.$$

Then

$$|\dot{\lambda}| \leq \kappa(\lambda, A) \|\dot{A}\|.$$

In other words, $\kappa(\lambda, A)$ is the magnification factor relating a perturbation in the matrix A to the resulting perturbation in an eigenvalue λ . Notice that $\kappa(\lambda, A)$ is independent of the normalization of the left and right eigenvectors, y and x , and that

$$\kappa(\lambda, A) \geq 1.$$

If you have already computed the matrix X whose columns are the right eigenvectors, one way to compute the left eigenvectors is to let

$$Y^H = X^{-1}.$$

Then, since

$$Y^H A = \Lambda Y^H,$$

the rows of Y^H are the left eigenvectors. In this case, the left eigenvectors are normalized so that

$$Y^H X = I,$$

so the denominator in $\kappa(\lambda, A)$ is $y^H x = 1$ and

$$\kappa(\lambda, A) = \|y\| \|x\|.$$

Since $\|x\| \leq \|X\|$ and $\|y\| \leq \|X^{-1}\|$, we have

$$\kappa(\lambda, A) \leq \kappa(X).$$

The condition number of the eigenvector matrix is an upper bound for the individual eigenvalue condition numbers.

The MATLAB function `condeig` computes eigenvalue condition numbers. Continuing with the `gallery(3)` example,

```
A = gallery(3)
lambda = eig(A)
kappa = condeig(A)
```

yields

```
lambda =
    1.0000
    2.0000
    3.0000
```

```
kappa =
   603.6390
   395.2366
   219.2920
```

This indicates that $\lambda_1 = 1$ is slightly more sensitive than $\lambda_2 = 2$ or $\lambda_3 = 3$. A perturbation in `gallery(3)` may result in perturbations in its eigenvalues that are 200 to 600 times as large. This is consistent with the cruder estimate of $1.2 \cdot 10^3$ obtained from `condest(X)`.

To test this analysis, let's make a small random perturbation in `A = gallery(3)` and see what happens to its eigenvalues.

```
format long
delta = 1.e-6;
lambda = eig(A + delta*randn(3,3))
```

```
lambda =

    1.00011344999452
    1.99992040276116
    2.99996856435075
```

The perturbation in the eigenvalues is

```
lambda - (1:3)'  
  
ans =  
1.0e-003 *  
0.11344999451923  
-0.07959723883699  
-0.03143564924635
```

This is smaller than, but roughly the same size as, the estimates provided by `condeig` and the perturbation analysis.

```
delta*condeig(A)  
  
ans =  
1.0e-003 *  
0.60363896495665  
0.39523663799014  
0.21929204271846
```

If A is real and symmetric, or complex and Hermitian, then its right and left eigenvectors are the same. In this case,

$$y^H x = \|y\| \|x\|,$$

so, for symmetric and Hermitian matrices,

$$\kappa(\lambda, A) = 1.$$

The eigenvalues of symmetric and Hermitian matrices are perfectly well conditioned. Perturbations in the matrix lead to perturbations in the eigenvalues that are roughly the same size. This is true even for multiple eigenvalues.

At the other extreme, if λ_k is a multiple eigenvalue that does not have a corresponding full set of linearly independent eigenvectors, then the previous analysis does not apply. In this case, the characteristic polynomial for an n -by- n matrix can be written

$$p(\lambda) = \det(A - \lambda I) = (\lambda - \lambda_k)^m q(\lambda),$$

where m is the multiplicity of λ_k and $q(\lambda)$ is a polynomial of degree $n - m$ that does not vanish at λ_k . A perturbation in the matrix of size δ results in a change in the characteristic polynomial from $p(\lambda) = 0$ to something like

$$p(\lambda) = O(\delta).$$

In other words,

$$(\lambda - \lambda_k)^m = O(\delta)/q(\lambda).$$

The roots of this equation are

$$\lambda = \lambda_k + O(\delta^{1/m}).$$

This m th root behavior says that multiple eigenvalues without a full set of eigenvectors are extremely sensitive to perturbation.

As an artificial, but illustrative, example, consider the 16-by-16 matrix with 2's on the main diagonal, 1's on the superdiagonal, δ in the lower left-hand corner, and 0's elsewhere:

$$A = \begin{pmatrix} 2 & 1 & & & \\ & 2 & 1 & & \\ & & \ddots & \ddots & \\ & & & 2 & 1 \\ \delta & & & & 2 \end{pmatrix}.$$

The characteristic equation is

$$(\lambda - 2)^{16} = \delta.$$

If $\delta = 0$, this matrix has an eigenvalue of multiplicity 16 at $\lambda = 2$, but there is only 1 eigenvector to go along with this multiple eigenvalue. If δ is on the order of floating-point roundoff error, that is, $\delta \approx 10^{-16}$, then the eigenvalues are on a circle in the complex plane with center at 2 and radius

$$(10^{-16})^{1/16} = 0.1.$$

A perturbation on the size of roundoff error changes the eigenvalue from 2.0000 to 16 different values, including 1.9000, 2.1000, and $2.0924 + 0.0383i$. A tiny change in the matrix elements causes a much larger change in the eigenvalues.

Essentially the same phenomenon, but in a less obvious form, explains the behavior of another MATLAB gallery example,

```
A = gallery(5)
```

The matrix is

```
A =
    -9      11     -21      63     -252
     70     -69      141    -421     1684
   -575     575   -1149     3451   -13801
   3891   -3891     7782   -23345    93365
   1024   -1024     2048    -6144   24572
```

The computed eigenvalues, obtained from `lambda = eig(A)`, are

```
lambda =
   -0.0408
  -0.0119 + 0.0386i
  -0.0119 - 0.0386i
   0.0323 + 0.0230i
   0.0323 - 0.0230i
```

How accurate are these computed eigenvalues?

The `gallery(5)` matrix was constructed in such a way that its characteristic equation is

$$\lambda^5 = 0.$$

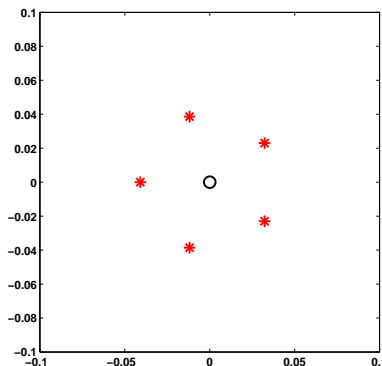


Figure 10.4. `plot(eig(gallery(5)))`.

You can confirm this by noting that A^5 , which is computed without any roundoff error, is the zero matrix. The characteristic equation can easily be solved by hand. All five eigenvalues are actually equal to zero. The computed eigenvalues give little indication that the “correct” eigenvalues are all zero. We certainly have to admit that the computed eigenvalues are not very accurate.

The MATLAB `eig` function is doing as well as can be expected on this problem. The inaccuracy of the computed eigenvalues is caused by their sensitivity, not by anything wrong with `eig`. The following experiment demonstrates this fact. Start with

```
A = gallery(5)
e = eig(A)
plot(real(e), imag(e), 'r*', 0, 0, 'ko')
axis(.1*[-1 1 -1 1])
axis square
```

Figure 10.4 shows that the computed eigenvalues are the vertices of a regular pentagon in the complex plane, centered at the origin. The radius is about 0.04.

Now repeat the experiment with a matrix where each element is perturbed by a single roundoff error. The elements of `gallery(5)` vary over four orders of magnitude, so the correct scaling of the perturbation is obtained with

```
e = eig(A + eps*randn(5,5).*A)
```

Put this statement, along with the `plot` and `axis` commands, on a single line and use the up arrow to repeat the computation several times. You will see that the pentagon flips orientation and that its radius varies between 0.03 and 0.07, but that the computed eigenvalues of the perturbed problems behave pretty much like the computed eigenvalues of the original matrix.

The experiment provides evidence for the fact that the computed eigenvalues are the exact eigenvalues of a matrix $A + E$, where the elements of E are on the order of roundoff error compared to the elements of A . This is the best we can expect to achieve with floating-point computation.

10.7 Singular Value Sensitivity and Accuracy

The sensitivity of singular values is much easier to characterize than the sensitivity of eigenvalues. The singular value problem is always perfectly well conditioned. A perturbation analysis would involve an equation like

$$\Sigma + \delta\Sigma = U^H(A + \delta A)V.$$

But, since U and V are orthogonal or unitary, they preserve norms. Consequently, $\|\delta\Sigma\| = \|\delta A\|$. Perturbations of any size in any matrix cause perturbations of roughly the same size in its singular values. There is no need to define condition numbers for singular values because they would always be equal to one. The MATLAB function `svd` always computes singular values to full floating-point accuracy.

We have to be careful about what we mean by “same size” and “full accuracy.” Perturbations and accuracy are measured relative the norm of the matrix or, equivalently, the largest singular value:

$$\|A\|_2 = \sigma_1.$$

The accuracy of the smaller singular values is measured relative to the largest one. If, as is often the case, the singular values vary over several orders of magnitude, the smaller ones might not have full accuracy relative to themselves. In particular, if the matrix is singular, then some of the σ_i must be zero. The computed values of these σ_i will usually be on the order of $\epsilon\|A\|$, where ϵ is `eps`, the floating-point accuracy parameter.

This can be illustrated with the singular values of `gallery(5)`. The statements

```
A = gallery(5)
format long e
svd(A)
```

produce

```
1.010353607103610e+005
1.679457384066496e+000
1.462838728086172e+000
1.080169069985612e+000
4.988578262459575e-014
```

The largest element of A is 93365, and we see that the largest singular value is a little larger, about 10^5 . There are three singular values near 10^0 . Recall that all the eigenvalues of this matrix are zero, so the matrix is singular and the smallest singular value should theoretically be zero. The computed value is somewhere between ϵ and $\epsilon\|A\|$.

Now let's perturb the matrix. Let this infinite loop run for a while.

```
while 1
    clc
    svd(A+eps*randn(5,5).*A)
    pause(.25)
end
```

This produces varying output like this.

```
1.010353607103610e+005
1.67945738406****e+000
1.46283872808****e+000
1.08016906998****e+000
*,*****-0**
```

The asterisks show the digits that change as we make the random perturbations. The 15-digit format does not show any changes in σ_1 . The changes in σ_2 , σ_3 , and σ_4 are smaller than $\epsilon\|A\|$, which is roughly 10^{-11} . The computed value of σ_5 is all roundoff error, less than 10^{-11} .

The `gallery(5)` matrix was constructed to have very special properties for the eigenvalue problem. For the singular value problem, its behavior is typical of any singular matrix.

10.8 Jordan and Schur Forms

The eigenvalue decomposition attempts to find a diagonal matrix Λ and a nonsingular matrix X so that

$$A = X\Lambda X^{-1}.$$

There are two difficulties with the eigenvalue decomposition. A theoretical difficulty is that the decomposition does not always exist. A numerical difficulty is that, even if the decomposition exists, it might not provide a basis for robust computation.

The solution to the nonexistence difficulty is to get as close to diagonal as possible. This leads to the Jordan canonical form (JCF). The solution to the robustness difficulty is to replace “diagonal” with “triangular” and to use orthogonal and unitary transformations. This leads to the Schur form.

A *defective* matrix is a matrix with at least one multiple eigenvalue that does not have a full set of linearly independent eigenvectors. For example, `gallery(5)` is defective; zero is an eigenvalue of multiplicity five that has only one eigenvector.

The JCF is the decomposition

$$A = XJX^{-1}.$$

If A is not defective, then the JCF is the same as the eigenvalue decomposition. The columns of X are the eigenvectors and $J = \Lambda$ is diagonal. But if A is defective, then X consists of eigenvectors and *generalized* eigenvectors. The matrix J has the

eigenvalues on the diagonal and ones on the superdiagonal in positions corresponding to the columns of X that are not ordinary eigenvectors. The rest of the elements of J are zero.

The function `jordan` in the MATLAB Symbolic Toolbox uses Maple and unlimited-precision rational arithmetic to try to compute the JCF of small matrices whose entries are small integers or ratios of small integers. If the characteristic polynomial does not have rational roots, Maple regards all the eigenvalues as distinct and produces a diagonal JCF.

The JCF is a discontinuous function of the matrix. Almost any perturbation of a defective matrix can cause a multiple eigenvalue to separate into distinct values and eliminate the ones on the superdiagonal of the JCF. Matrices that are nearly defective have badly conditioned sets of eigenvectors, and the resulting similarity transformations cannot be used for reliable numerical computation.

A numerically satisfactory alternative to the JCF is provided by the Schur form. Any matrix can be transformed to upper triangular form by a unitary similarity transformation:

$$B = T^H A T.$$

The eigenvalues of A are on the diagonal of its Schur form B . Since unitary transformations are perfectly well conditioned, they do not magnify any errors.

For example,

```
A = gallery(3)
[T,B] = schur(A)
```

produces

```
A =
  -149   -50  -154
   537   180   546
  -27    -9   -25

T =
   0.3162   -0.6529    0.6882
  -0.9487   -0.2176    0.2294
   0.0000    0.7255    0.6882

B =
   1.0000   -7.1119  -815.8706
           0    2.0000  -55.0236
           0         0    3.0000
```

The diagonal elements of B are the eigenvalues of A . If A were symmetric, B would be diagonal. In this case, the large off-diagonal elements of B measure the lack of symmetry in A .

10.9 The QR Algorithm

The QR algorithm is one of the most important, widely used, and successful tools we have in technical computation. Several variants of it are in the mathematical

core of MATLAB. They compute the eigenvalues of real symmetric matrices, real nonsymmetric matrices, and pairs of complex matrices, and the singular values of general matrices. These functions are used, in turn, to find zeros of polynomials, to solve special linear systems, and to assess stability, and for many other tasks in various toolboxes.

Dozens of people have contributed to the development of the various QR algorithms. The first complete implementation and an important convergence analysis are due to J. H. Wilkinson. Wilkinson's book, *The Algebraic Eigenvalue Problem* [56], as well as two fundamental papers, was published in 1965.

The QR *algorithm* is based on repeated use of the QR *factorization* that we described in Chapter 5, Least Squares. The letter “Q” denotes orthogonal and unitary matrices and the letter “R” denotes right, or upper, triangular matrices. The `qr` function in MATLAB factors any matrix, real or complex, square or rectangular, into the product of a matrix Q with orthonormal columns and matrix R that is nonzero only its upper, or right, triangle.

Using the `qr` function, a simple variant of the QR algorithm, known as the single-shift algorithm, can be expressed as a MATLAB one-liner. Let A be any square matrix. Start with

```
n = size(A,1)
I = eye(n,n)
```

Then one step of the single-shift QR iteration is given by

```
s = A(n,n); [Q,R] = qr(A - s*I); A = R*Q + s*I
```

If you enter this on one line, you can use the up arrow key to iterate. The quantity s is the shift; it accelerates convergence. The QR factorization makes the matrix triangular:

$$A - sI = QR.$$

Then the reverse-order multiplication RQ restores the eigenvalues because

$$RQ + sI = Q^T(A - sI)Q + sI = Q^T A Q,$$

so the new A is orthogonally similar to the original A . Each iteration effectively transfers some “mass” from the lower to the upper triangle while preserving the eigenvalues. As the iterations are repeated, the matrix often approaches an upper triangular matrix with the eigenvalues conveniently displayed on the diagonal.

For example, start with $A = \text{gallery}(3)$.

```
-149    -50   -154
 537    180    546
 -27     -9    -25
```

The first iterate,

```
28.8263 -259.8671  773.9292
  1.0353  -8.6686  33.1759
-0.5973   5.5786 -14.1578
```

already has its largest elements in the upper triangle. After five more iterations, we have

```

2.7137  -10.5427  -814.0932
-0.0767   1.4719  -76.5847
0.0006   -0.0039   1.8144

```

As we know, this matrix was contrived to have its eigenvalues equal to 1, 2, and 3. We can begin to see these three values on the diagonal. Five more iterations gives

```

3.0716  -7.6952  802.1201
0.0193   0.9284  158.9556
-0.0000   0.0000   2.0000

```

One of the eigenvalues has been computed to full accuracy and the below-diagonal element adjacent to it has become zero. It is time to deflate the problem and continue the iteration on the 2-by-2 upper left submatrix.

The QR algorithm is never practiced in this simple form. It is always preceded by a reduction to Hessenberg form, in which all the elements below the subdiagonal are zero. This reduced form is preserved by the iteration, and the factorizations can be done much more quickly. Furthermore, the shift strategy is more sophisticated and is different for various forms of the algorithm.

The simplest variant involves real, symmetric matrices. The reduced form in this case is tridiagonal. Wilkinson provided a shift strategy that allowed him to prove a global convergence theorem. Even in the presence of roundoff error, we do not know of any examples that cause the implementation in MATLAB to fail.

The SVD variant of the QR algorithm is preceded by a reduction to a bidiagonal form that preserves the singular values. It has the same guaranteed convergence properties as the symmetric eigenvalue iteration.

The situation for real, nonsymmetric matrices is much more complicated. In this case, the given matrix has real elements, but its eigenvalues may well be complex. Real matrices are used throughout, with a double-shift strategy that can handle two real eigenvalues, or a complex conjugate pair. Even thirty years ago, counterexamples to the basic iteration were known and Wilkinson introduced an ‘ad hoc’ shift to handle them. But no one has been able to prove a complete convergence theorem. In principle, it is possible for the `eig` function in MATLAB to fail with an error message about lack of convergence.

10.10 eigsvdgui

Figures 10.5 and 10.6 are snapshots of the output produced by `eigsvdgui` showing steps in the computation of the eigenvalues of a nonsymmetric matrix and of a symmetric matrix. Figure 10.7 is a snapshot of the output produced by `eigsvdgui` showing steps in the computation of the singular values of a nonsymmetric matrix.

The first phase in the computation shown in Figure 10.5 of the eigenvalues of a real, nonsymmetric, n -by- n matrix is a sequence of $n - 2$ orthogonal similarity transformations. The k th transformation uses Householder reflections to introduce

zeros below the subdiagonal in the k th column. The result of this first phase is known as a Hessenberg matrix; all the elements below the first subdiagonal are zero.

```

for k = 1:n-2
    u = A(:,k);
    u(1:k) = 0;
    sigma = norm(u);
    if sigma ~= 0
        if u(k+1) < 0, sigma = -sigma; end
        u(k+1) = u(k+1) + sigma;
        rho = 1/(sigma*u(k+1));
        v = rho*A*u;
        w = rho*(u'*A)';
        gamma = rho/2*u'*v;
        v = v - gamma*u;
        w = w - gamma*u;
        A = A - v*u' - u*w';
        A(k+2:n,k) = 0;
    end
end
end

```

The second phase uses the QR algorithm to introduce zeros in the first subdiagonal. A real, nonsymmetric matrix will usually have some complex eigenvalues, so it is not possible to completely transform it to the upper triangular Schur form. Instead, a real Schur form with 1-by-1 and 2-by-2 submatrices on the diagonal is produced. Each 1-by-1 matrix is a real eigenvalue of the original matrix. The eigenvalues of each 2-by-2 block are a pair of complex conjugate eigenvalues of the original matrix.

The computation of the eigenvalues of a symmetric matrix shown in Figure 10.6 also has two phases. The result of the first phase is a matrix that is both symmetric and Hessenberg, so it is tridiagonal. Then, since all the eigenvalues of a real, symmetric matrix are real, the QR iterations in the second phase can completely zero the subdiagonal and produce a real, diagonal matrix containing the eigenvalues.

Figure 10.7 shows the output produced by `eigsvdgui` as it computes the singular values of a nonsymmetric matrix. Multiplication by any orthogonal matrix preserves singular values, so it is not necessary to use similarity transformations. The first phase uses a Householder reflection to introduce zeros below the diagonal in each column, then a different Householder reflection to introduce zeros to the right of the first superdiagonal in the corresponding row. This produces an upper bidiagonal matrix with the same singular values as the original matrix. The QR iterations then zero the superdiagonal to produce a diagonal matrix containing the singular values.

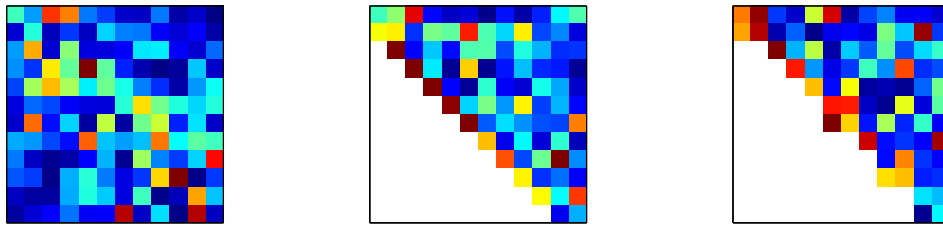


Figure 10.5. eigsvdgui, nonsymmetric matrix.

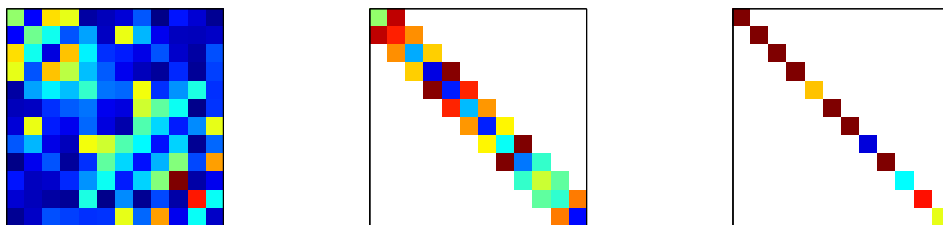


Figure 10.6. eigsvdgui, symmetric matrix.

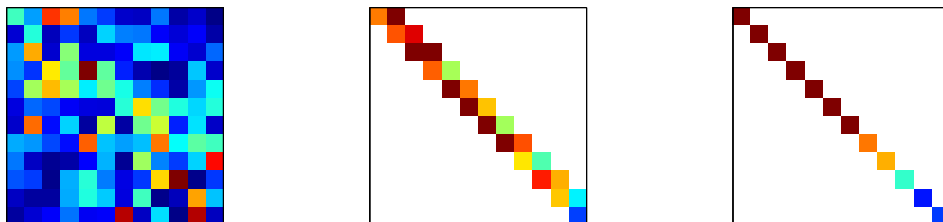


Figure 10.7. eigsvdgui, SVD.

10.11 Principal Components

Principal component analysis, or PCA, approximates a general matrix by a sum of a few “simple” matrices. By “simple” we mean rank one; all of the rows are multiples of each other, and so are all of the columns. Let A be any real m -by- n matrix. The economy-sized SVD

$$A = U\Sigma V^T$$

can be rewritten

$$A = E_1 + E_2 + \cdots + E_p,$$

where $p = \min(m, n)$. The component matrices E_k are rank one outer products:

$$E_k = \sigma_k u_k v_k^T.$$

Each column of E_k is a multiple of u_k , the k th column of U , and each row is a multiple of v_k^T , the transpose of the k th column of V . The component matrices are orthogonal to each other in the sense that

$$E_j E_k^T = 0, \quad j \neq k.$$

The norm of each component matrix is the corresponding singular value

$$\|E_k\| = \sigma_k.$$

Consequently, the contribution each E_k makes to reproducing A is determined by the size of the singular value σ_k .

If the sum is truncated after $r < p$ terms,

$$A_r = E_1 + E_2 + \cdots + E_r,$$

the result is a rank r approximation to the original matrix A . In fact, A_r is the closest rank r approximation to A . It turns out that the error in this approximation is

$$\|A - A_r\| = \sigma_{r+1}.$$

Since the singular values are in decreasing order, the accuracy of the approximation increases as the rank increases.

PCA is used in a wide range of fields, including statistics, earth sciences, and archaeology. The description and notation also vary widely. Perhaps the most common description is in terms of eigenvalues and eigenvectors of the cross-product matrix $A^T A$. Since

$$A^T A V = V \Sigma^2,$$

the columns of V are the eigenvectors $A^T A$. The columns of U , scaled by the singular values, can then be obtained from

$$U \Sigma = A V.$$

The data matrix A is frequently *standardized* by subtracting the means of the columns and dividing by their standard deviations. If this is done, the cross-product matrix becomes the correlation matrix.

Factor analysis is a closely related technique that makes additional statistical assumptions about the elements of A and modifies the diagonal elements of $A^T A$ before computing the eigenvalues and eigenvectors.

For a simple example of PCA on the unmodified matrix A , suppose we measure the height and weight of six subjects and obtain the following data.

A =		
	47	15
	93	35
	53	15
	45	10
	67	27
	42	10

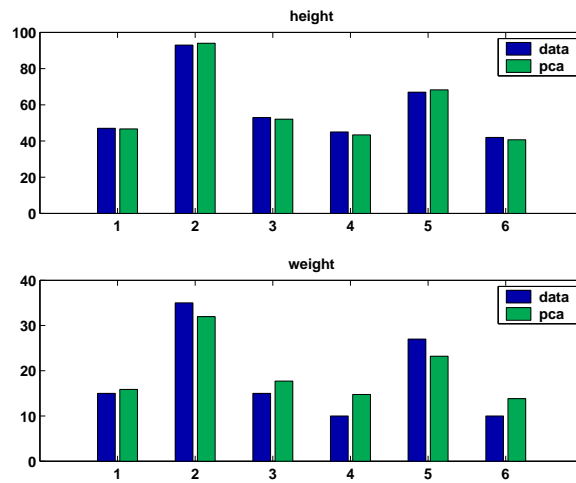


Figure 10.8. *PCA of data.*

The dark bars in Figure 10.8 plot this data.

We expect height and weight to be strongly correlated. We believe there is one underlying component—let’s call it “size”—that predicts both height and weight. The statement

```
[U,S,V] = svd(A,0)
sigma = diag(S)
```

produces

```
U =
  0.3153    0.1056
  0.6349   -0.3656
  0.3516    0.3259
  0.2929    0.5722
  0.4611   -0.4562
  0.2748    0.4620
```

```
V =
  0.9468    0.3219
  0.3219   -0.9468
```

```
sigma =
  156.4358
    8.7658
```

Notice that σ_1 is much larger than σ_2 .

The rank one approximation to A is

```
E1 = sigma(1)*U(:,1)*V(:,1)'
```

```
E1 =
 46.7021  15.8762
 94.0315  31.9657
 52.0806  17.7046
 43.3857  14.7488
 68.2871  23.2139
 40.6964  13.8346
```

In other words, the single underlying principal component is

```
size = sigma(1)*U(:,1)
```

```
size =
 49.3269
 99.3163
 55.0076
 45.8240
 72.1250
 42.9837
```

The two measured quantities are then well approximated by

```
height ≈ size*V(1,1)
weight ≈ size*V(2,1)
```

The light bars in Figure 10.8 plot these approximations.

A larger example involves digital image processing. The statements

```
load detail
subplot(2,2,1)
image(X)
colormap(gray(64))
axis image, axis off
r = rank(X)
title(['rank = ' int2str(r)])
```

produce the first subplot in Figure 10.9. The matrix X obtained with the `load` statement is 359 by 371 and is numerically of full rank. Its elements lie between 1 and 64 and serve as indices into a gray-scale color map. The resulting picture is a detail from Albrecht Dürer's etching "Melancholia II," showing a 4-by-4 magic square. The statements

```
[U,S,V] = svd(X,0);
sigma = diag(S);
semilogy(sigma, '.')
```

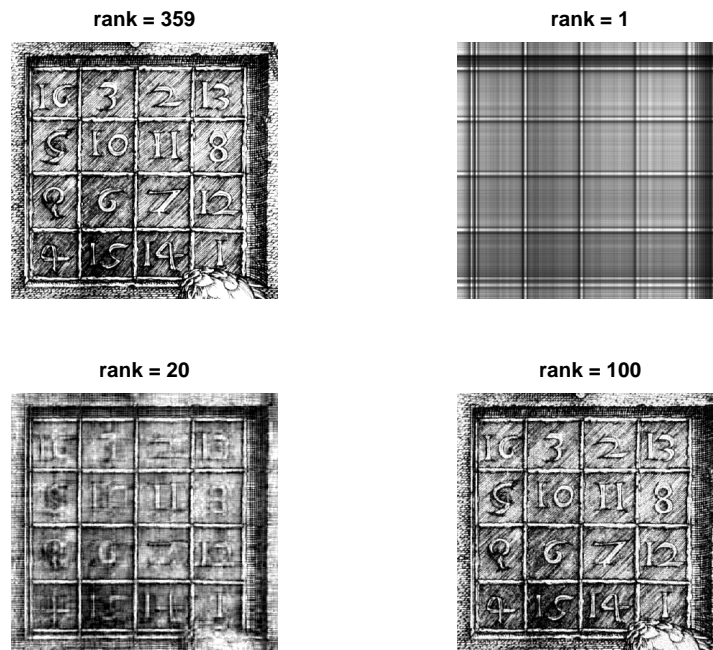



Figure 10.9. *Principal components of Dürer's magic square.*

produce the logarithmic plot of the singular values of X shown in Figure 10.10. We see that the singular values decrease rapidly. There are one greater than 10^4 and only six greater than 10^3 .

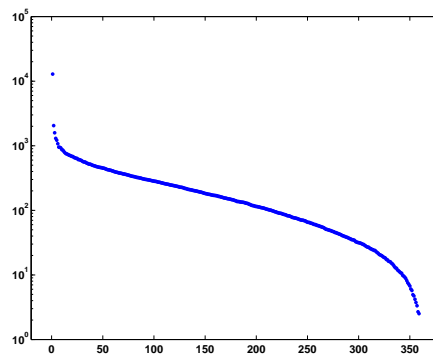


Figure 10.10. *Singular values (log scale).*

The other three subplots in Figure 10.9 show the images obtained from principal component approximations to X with $r = 1$, $r = 20$, and $r = 100$. The rank one approximation shows the horizontal and vertical lines that result from a single outer product, $E_1 = \sigma_1 u_1 v_1^T$. This checkerboard-like structure is typical of low-

rank principal component approximations to images. The individual numerals are recognizable in the $r = 20$ approximation. There is hardly any visible difference between the $r = 100$ approximation and the full-rank image.

Although low-rank matrix approximations to images do require less computer storage and transmission time than the full-rank image, there are more effective data compression techniques. The primary uses of PCA in image processing involve feature recognition.

10.12 Circle Generator

The following algorithm was used to plot circles on some of the first computers with graphical displays. At the time, there was no MATLAB and no floating-point arithmetic. Programs were written in machine language and arithmetic was done on scaled integers. The circle-generating program looked something like this.

```

x = 32768
y = 0
L: load y
  shift right 5 bits
  add x
  store in x
  change sign
  shift right 5 bits
  add y
  store in y
  plot x y
  go to L

```

Why does this generate a circle? In fact, does it actually generate a circle? There are no trig functions, no square roots, no multiplications or divisions. It's all done with shifts and additions.

The key to this algorithm is the fact that the new x is used in the computation of the new y . This was convenient on computers at the time because it meant you needed only two storage locations, one for x and one for y . But, as we shall see, it is also why the algorithm comes close to working at all.

Here is a MATLAB version of the same algorithm.

```

h = 1/32;
x = 1;
y = 0;
while 1
  x = x + h*y;
  y = y - h*x;
  plot(x,y,'.')
  drawnow
end

```

The M-file `circlegen` lets you experiment with various values of the step size h . It provides an actual circle in the background. Figure 10.11 shows the output for the carefully chosen default value, $h = 0.20906$. It's not quite a circle. However, `circlegen(h)` generates better circles with smaller values of h . Try `circlegen(h)` for various h yourself.

If we let (x_n, y_n) denote the n th point generated, then the iteration is

$$\begin{aligned}x_{n+1} &= x_n + hy_n, \\y_{n+1} &= y_n - hx_{n+1}.\end{aligned}$$

The key is the fact that x_{n+1} appears on the right in the second equation. Substi-

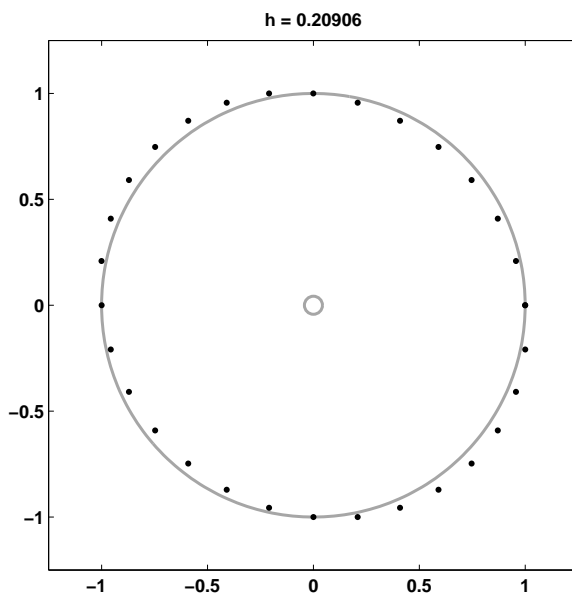


Figure 10.11. `circlegen`.

tuting the first equation into the second gives

$$\begin{aligned}x_{n+1} &= x_n + hy_n, \\y_{n+1} &= -hx_n + (1 - h^2)y_n.\end{aligned}$$

Let's switch to matrix-vector notation. Let x_n now denote the 2-vector specifying the n th point and let A be the *circle generator* matrix

$$A = \begin{pmatrix} 1 & h \\ -h & 1 - h^2 \end{pmatrix}.$$

With this notation, the iteration is simply

$$x_{n+1} = Ax_n.$$

This immediately leads to

$$x_n = A^n x_0.$$

So, the question is, for various values of h , how do powers of the circle generator matrix behave?

For most matrices A , the behavior of A^n is determined by its *eigenvalues*. The MATLAB statement

```
[X,Lambda] = eig(A)
```

produces a *diagonal* eigenvalue matrix Λ and a corresponding eigenvector matrix X so that

$$AX = X\Lambda.$$

If X^{-1} exists, then

$$A = X\Lambda X^{-1}$$

and

$$A^n = X\Lambda^n X^{-1}.$$

Consequently, the powers A^n remain bounded if the eigenvector matrix is nonsingular and the eigenvalues λ_k , which are the diagonal elements of Λ , satisfy

$$|\lambda_k| \leq 1.$$

Here is an easy experiment. Enter the line

```
h = 2*rand, A = [1 h; -h 1-h^2], lambda = eig(A), abs(lambda)
```

Repeatedly press the up arrow key, then the Enter key. You should eventually become convinced, at least experimentally, of the following:

For any h in the interval $0 < h < 2$, the eigenvalues of the circle generator matrix A are complex numbers with absolute value 1.

The Symbolic Toolbox provides some assistance in actually proving this fact.

```
syms h
A = [1 h; -h 1-h^2]
lambda = eig(A)
```

creates a symbolic version of the iteration matrix and finds its eigenvalues.

```
A =
[      1,      h]
[     -h, 1-h^2]

lambda =
[ 1-1/2*h^2+1/2*(-4*h^2+h^4)^(1/2)]
[ 1-1/2*h^2-1/2*(-4*h^2+h^4)^(1/2)]
```

The statement

```
abs(lambda)
```

does not do anything useful, in part because we have not yet made any assumptions about the symbolic variable `h`.

We note that the eigenvalues will be complex if the quantity involved in the square root is negative, that is, if $|h| < 2$. The determinant of a matrix should be the product of its eigenvalues. This is confirmed with

```
d = det(A)
```

or

```
d = simple(prod(lambda))
```

Both produce

```
d =
1
```

Consequently, if $|h| < 2$, the eigenvalues, λ , are complex and their product is 1, so they must satisfy $|\lambda| = 1$.

Because

$$\lambda = 1 - h^2/2 \pm h\sqrt{-1 + h^2/4},$$

it is plausible that, if we define θ by

$$\cos \theta = 1 - h^2/2$$

or

$$\sin \theta = h\sqrt{1 - h^2/4},$$

then

$$\lambda = \cos \theta \pm i \sin \theta.$$

The Symbolic Toolbox confirms this with

```
theta = acos(1-h^2/2);
Lambda = [cos(theta)-i*sin(theta); cos(theta)+i*sin(theta)]
diff = simple(lambda-Lambda)
```

which produces

```
Lambda =
[ 1-1/2*h^2-1/2*i*(4*h^2-h^4)^(1/2)]
[ 1-1/2*h^2+1/2*i*(4*h^2-h^4)^(1/2)]

diff =
[ 0]
[ 0]
```

In summary, this proves that, if $|h| < 2$, the eigenvalues of the circle generator matrix are

$$\lambda = e^{\pm i\theta}.$$

The eigenvalues are distinct, and hence X must be nonsingular and

$$A^n = X \begin{pmatrix} e^{in\theta} & 0 \\ 0 & e^{-in\theta} \end{pmatrix} X^{-1}.$$

If the step size h happens to correspond to a value of θ that is $2\pi/p$, where p is an integer, then the algorithm generates only p discrete points before it repeats itself.

How close does our circle generator come to actually generating circles? In fact, it generates ellipses. As the step size h gets smaller, the ellipses get closer to circles. The *aspect ratio* of an ellipse is the ratio of its major axis to its minor axis. It turns out that the aspect ratio of the ellipse produced by the generator is equal to the *condition number* of the matrix of eigenvectors, X . The condition number of a matrix is computed by the MATLAB function `cond(X)` and is discussed in more detail in Chapter 2, Linear Equations.

The solution to the 2-by-2 system of ordinary differential equations

$$\dot{x} = Qx,$$

where

$$Q = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix},$$

is a circle

$$x(t) = \begin{pmatrix} \cos t & \sin t \\ -\sin t & \cos t \end{pmatrix} x(0).$$

So the iteration matrix

$$\begin{pmatrix} \cos h & \sin h \\ -\sin h & \cos h \end{pmatrix}$$

generates perfect circles. The Taylor series for $\cos h$ and $\sin h$ show that the iteration matrix for our circle generator,

$$A = \begin{pmatrix} 1 & h \\ -h & 1 - h^2 \end{pmatrix},$$

approaches the perfect iterator as h gets small.

10.13 Further Reading

The reference books on matrix computation [6, 7, 8, 9, 10, 11] discuss eigenvalues. In addition, the classic by Wilkinson [1] is still readable and relevant. ARPACK, which underlies the sparse `eigs` function, is described in [2].

Exercises

- 10.1. Match the following matrices to the following properties. For each matrix, choose the most descriptive property. Each property can be matched to one or more of the matrices.

<code>magic(4)</code>	Symmetric
<code>hess(magic(4))</code>	Defective
<code>schur(magic(5))</code>	Orthogonal
<code>pascal(6)</code>	Singular
<code>hess(pascal(6))</code>	Tridiagonal
<code>schur(pascal(6))</code>	Diagonal
<code>orth(gallery(3))</code>	Hessenberg form
<code>gallery(5)</code>	Schur form
<code>gallery('frank',12)</code>	Jordan form
<code>[1 1 0; 0 2 1; 0 0 3]</code>	
<code>[2 1 0; 0 2 1; 0 0 2]</code>	

- 10.2. (a) What is the largest eigenvalue of `magic(n)`? Why?
 (b) What is the largest singular value of `magic(n)`? Why?
- 10.3. As a function of n , what are the eigenvalues of the n -by- n Fourier matrix, `fft(eye(n))`?
- 10.4. Try this:

```
n = 101;
d = ones(n-1,1);
A = diag(d,1) + diag(d,-1);
e = eig(A)
plot(-(n-1)/2:(n-1)/2,e, 'o')
```

Do you recognize the resulting curve? Can you guess a formula for the eigenvalues of this matrix?

- 10.5. Plot the trajectories in the complex plane of the eigenvalues of the matrix A with elements

$$a_{i,j} = \frac{1}{i-j+t}$$

as t varies over the interval $0 < t < 1$. Your plot should look something like Figure 10.12.

- 10.6. (a) In theory, the elements of the vector obtained from

```
condeig(gallery(5))
```

should be infinite. Why?

- (b) In practice, the computed values are only about 10^{10} . Why?

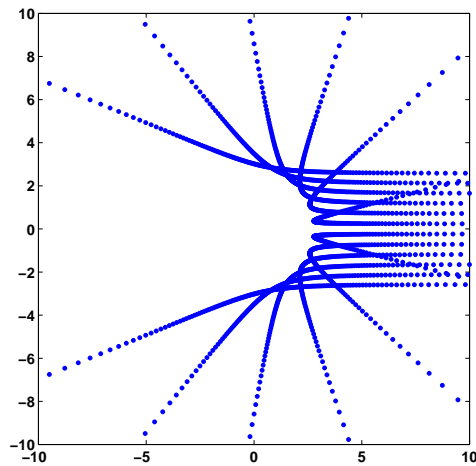


Figure 10.12. *Eigenvalue trajectories.*

10.7. This exercise uses the Symbolic Toolbox to study a classic eigenvalue test matrix, the Rosser matrix.

(a) You can compute the eigenvalues of the Rosser matrix exactly and order them in increasing order with

```
R = sym(rosser)
e = eig(R)
[ignore,k] = sort(double(e))
e = e(k)
```

Why can't you just use `e = sort(eig(R))`?

(b) You can compute and display the characteristic polynomial of `R` with

```
p = poly(R)
f = factor(p)
pretty(f)
```

Which terms in `f` correspond to which eigenvalues in `e`?

(c) What does each of these statements do?

```
e = eig(sym(rosser))
r = eig(rosser)
double(e) - r
double(e - r)
```

(d) Why are the results in (c) on the order of 10^{-12} instead of `eps`?

(e) Change `R(1,1)` from 611 to 612 and compute the eigenvalues of the modified matrix. Why do the results appear in a different form?

10.8. Both of the matrices

```
P = gallery('pascal',12)
F = gallery('frank',12)
```

have the property that, if λ is an eigenvalue, so is $1/\lambda$. How well do the computed eigenvalues preserve this property? Use `condeig` to explain the different behavior for the two matrices.

10.9. Compare these three ways to compute the singular values of a matrix.

```
svd(A)
sqrt(eig(A'*A))
Z = zeros(size(A)); s = eig([Z A; A' Z]); s = s(s>0)
```

10.10. Experiment with `eigsvdgui` on random symmetric and nonsymmetric matrices, `randn(n)`. Choose values of `n` appropriate for the speed of your computer and investigate the three variants `eig`, `symm`, and `svd`. The title in the `eigsvdgui` shows the number of iterations required. Roughly, how does the number of iterations for the three different variants depend upon the order of the matrix?

10.11. Pick a value of `n` and generate a matrix with

```
A = diag(ones(n-1,1),-1) + diag(1,n-1);
```

Explain any atypical behavior you observe with each of the following.

```
eigsvdgui(A,'eig')
eigsvdgui(A,'symm')
eigsvdgui(A,'svd')
```

10.12. The NCM file `imagesvd.m` helps you investigate the use of PCA in digital image processing. If you have them available, use your own photographs. If you have access to the MATLAB Image Processing Toolbox, you may want to use its advanced features. However, it is possible to do basic image processing without the toolbox.

For an m -by- n color image in JPEG format, the statement

```
X = imread('myphoto.jpg');
```

produces a three-dimensional m -by- n -by-3 array `X` with m -by- n integer subarrays for the red, green, and blue intensities. It would be possible to compute three separate m -by- n SVDs of the three colors. An alternative that requires less work involves altering the dimensions of `X` with

```
X = reshape(X,m,3*n)
```

and then computing one m -by- $3n$ SVD.

(a) The primary computation in `imagesvd` is done by

```
[V,S,U] = svd(X',0)
```

How does this compare with

$$[U, S, V] = \text{svd}(X, 0)$$

(b) How does the choice of approximating rank affect the visual qualities of the images? There are no precise answers here. Your results will depend upon the images you choose and the judgments you make.

- 10.13. This exercise investigates a model of the human gait developed by Nikolaus Troje at the Bio Motion Lab of Ruhr University in Bochum, Germany. Their Web page provides an interactive demo [3]. Two papers describing the work are also available on the Web [4, 5]. Troje's data result from motion capture experiments involving subjects wearing reflective markers walking on a treadmill. His model is a five-term Fourier series with vector-valued coefficients obtained by principal component analysis of the experimental data. The components, which are also known as *postures* or *eigenpostures*, correspond to static position, forward motion, sideways sway, and two hopping/bouncing movements that differ in the phase relationship between the upper and lower portions of the body. The model is purely descriptive; it does not make any direct use of physical laws of motion.

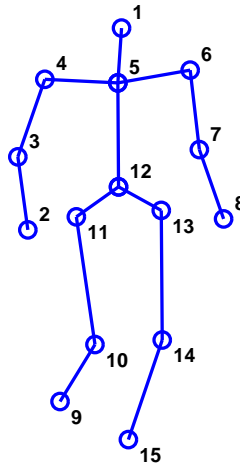


Figure 10.13. *Walker at rest.*

The moving position $v(t)$ of the human body is described by 45 functions of time, which correspond to the location of 15 points in three-dimensional space. Figure 10.13 is a static snapshot. The model is

$$v(t) = v_1 + v_2 \sin \omega t + v_3 \cos \omega t + v_4 \sin 2\omega t + v_5 \cos 2\omega t.$$

If the postures v_1, \dots, v_5 are regarded as the columns of a single 45-by-5 matrix V , the calculation of $v(t)$ for any t involves a matrix-vector multiplication. The resulting vector can then be reshaped into a 15-by-3 array that exposes

the spatial coordinates. For example, at $t = 0$, the time-varying coefficients form the vector $\mathbf{w} = [1 \ 0 \ 1 \ 0 \ 1]'$. Consequently, `reshape(V*w,15,3)` produces the coordinates of the initial position. The five postures for an individual subject are obtained by a combination of principal component and Fourier analysis. The individual characteristic frequency ω is an independent speed parameter. If the postures are averaged over the subjects with a particular characteristic, the result is a model for the “typical” walker with that characteristic. The characteristics available in the demo on the Web page include male/female, heavy/light, nervous/relaxed, and happy/sad. Our M-file `walker.m` is based on the postures for a typical female walker, f_1, \dots, f_5 , and a typical male walker, m_1, \dots, m_5 . Slider s_1 varies the time increment and hence the apparent walking speed. Sliders s_2, \dots, s_5 vary the amount that each component contributes to the overall motion. Slider s_6 varies a linear combination of the female and male walkers. A slider setting greater than 1.0 overemphasizes the characteristic. Here is the complete model, including the sliders:

$$\begin{aligned} f(t) &= f_1 + s_2 f_2 \sin \omega t + s_3 f_3 \cos \omega t + s_4 f_4 \sin 2\omega t + s_5 f_5 \cos 2\omega t, \\ m(t) &= m_1 + s_2 m_2 \sin \omega t + s_3 m_3 \cos \omega t + s_4 m_4 \sin 2\omega t + s_5 m_5 \cos 2\omega t, \\ v(t) &= (f(t) + m(t))/2 + s_6 (f(t) - m(t))/2. \end{aligned}$$

- Describe the visual differences between the gaits of the typical female and male walkers.
- File `walkers.mat` contains four data sets. **F** and **M** are the postures of the typical female and typical male obtained by analyzing all the subjects. **A** and **B** are the postures of two individual subjects. Are **A** and **B** male or female?
- Modify `walker.m` to add a waving hand as an additional, artificial, posture.
- What does this program do?

```
load walkers
F = reshape(F,15,3,5);
M = reshape(M,15,3,5);
for k = 1:5
    for j = 1:3
        subplot(5,3,j+3*(k-1))
        plot([F(:,j,k) M(:,j,k)])
        ax = axis;
        axis([1 15 ax(3:4)])
    end
end
```

- Change `walker.m` to use a Fourier model parametrized by amplitude and phase. The female walker is

$$f(t) = f_1 + s_2 a_1 \sin(\omega t + s_3 \phi_1) + s_4 a_2 \sin(2\omega t + s_5 \phi_2).$$

A similar formulation is used for the male walker. The linear combination of the two walkers using s_6 is unchanged. The amplitude and phase are

$$\begin{aligned} a_1 &= \sqrt{f_2^2 + f_3^2}, \\ a_2 &= \sqrt{f_4^2 + f_5^2}, \\ \phi_1 &= \tan^{-1}(f_3/f_2), \\ \phi_2 &= \tan^{-1}(f_5/f_4). \end{aligned}$$

- 10.14. In English, and in many other languages, vowels are usually followed by consonants and consonants are usually followed by vowels. This fact is revealed by a principal component analysis of the digraph frequency matrix for a sample of text. English text uses 26 letters, so the digraph frequency matrix is a 26-by-26 matrix, A , with counts of pairs of letters. Blanks and all other punctuation are removed from the text and the entire sample is thought of as circular or periodic, so the first letter follows the last letter. The matrix entry $a_{i,j}$ is the number of times the i th letter is followed by the j th letter in the text. The row and column sums of A are the same; they count the number of times individual letters occur in the sample. So the fifth row and fifth column usually have the largest sums because the fifth letter, which is “E,” is usually the most frequent.

A principal component analysis of A produces a first component,

$$A \approx \sigma_1 u_1 v_1^T,$$

that reflects the individual letter frequencies. The first right- and left-singular vectors, u_1 and v_1 , have elements that are all of the same sign and that are roughly proportional to the corresponding frequencies. We are primarily interested in the second principal component,

$$A \approx \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T.$$

The second term has positive entries in vowel-consonant and consonant-vowel positions and negative entries in vowel-vowel and consonant-consonant positions. The NCM collection contains a function `digraph.m` that carries out this analysis. Figure 10.14 shows the output produced by analyzing Lincoln’s Gettysburg Address with

```
digraph('gettysburg.txt')
```

The i th letter of the alphabet is plotted at coordinates $(u_{i,2}, v_{i,2})$. The distance of each letter from the origin is roughly proportional to its frequency, and the sign patterns cause the vowels to be plotted in one quadrant and the consonants to be plotted in the opposite quadrant. There is more detail. The letter “N” is usually preceded by a vowel and often followed by another consonant, like “D” or “G,” and so it shows up in a quadrant pretty much by itself. On the other hand, “H” is often preceded by another consonant, namely “T,” and followed by a vowel, “E,” so it also gets its own quadrant.

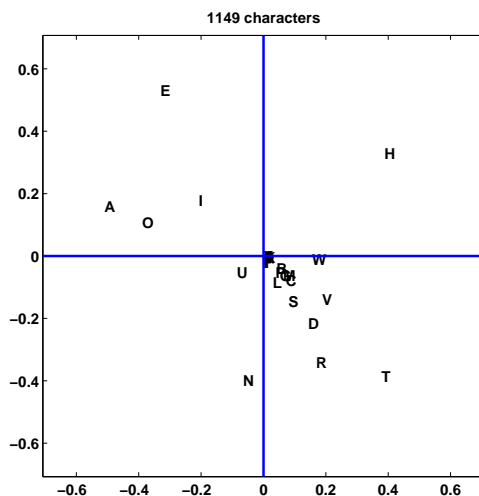


Figure 10.14. *The second principal component of a digraph matrix.*

- (a) Explain how `digraph` uses `sparse` to count letter pairs and create the matrix. `help sparse` should be useful.
- (b) Try `digraph` on other text samples. Roughly how many characters are needed to see the vowel-consonant frequency behavior?
- (c) Can you find any text with at least several hundred characters that does not show the typical behavior?
- (d) Try `digraph` on M-files or other source code. Do computer programs typically have the same vowel-consonant behavior as prose?
- (e) Try `digraph` on samples from other languages. Hawaiian and Finnish are particularly interesting. You may need to modify `digraph` to accommodate more or fewer than 26 letters. Do other languages show the same vowel-consonant behavior as English?
- 10.15. Explain the behavior of `circlegen` for each of the following values of the step size h . What, if anything, is special about these particular values? Is the orbit a discrete set of points? Does the orbit stay bounded, grow linearly, or grow exponentially? If necessary, increase the axis limits in `circlegen` so that it shows the entire orbit. Recall that $\phi = (1 + \sqrt{5})/2$ is the golden ratio:

$$h = \sqrt{2 - 2 \cos(2\pi/30)} \text{ (the default),}$$

$$h = 1/\phi,$$

$$h = \phi,$$

$$h = 1.4140,$$

$$h = \sqrt{2},$$

$$h = 1.4144,$$

$$h < 2,$$

$$h = 2,$$

$$h > 2.$$

- 10.16. (a) Modify `circlegen` so that both components of the new point are determined from the old point, that is,

$$\begin{aligned}x_{n+1} &= x_n + hy_n, \\y_{n+1} &= y_n - hx_n.\end{aligned}$$

(This is the *explicit Euler's method* for solving the circle ordinary differential equation.) What happens to the “circles”? What is the iteration matrix? What are its eigenvalues?

- (b) Modify `circlegen` so that the new point is determined by solving a 2-by-2 system of simultaneous equations:

$$\begin{aligned}x_{n+1} - hy_{n+1} &= x_n, \\y_{n+1} + hx_{n+1} &= y_n.\end{aligned}$$

(This is the *implicit Euler's method* for solving the circle ordinary differential equation.) What happens to the “circles”? What is the iteration matrix? What are its eigenvalues?

- 10.17. Modify `circlegen` so that it keeps track of the maximum and minimum radius during the iteration and returns the ratio of these two radii as the value of the function. Compare this computed aspect ratio with the eigenvector condition number, `cond(X)`, for various values of `h`.

Bibliography

- [1] J. WILKINSON, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.
- [2] R. B. LEHOUCQ, D. C. SORENSEN, AND C. YANG, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, SIAM, Philadelphia, 1998.
<http://www.caam.rice.edu/software/ARPACK>
- [3] BIO MOTION LAB, RUHR UNIVERSITY.
<http://www.bml.psy.ruhr-uni-bochum.de/Demos/BMLwalker.html>
- [4] N. TROJE.
<http://journalofvision.org/2/5/2>
- [5] N. TROJE.
http://www.biomotionlab.de/Text/WDP2002_Troje.pdf
- [6] E. ANDERSON, Z. BAI, C. BISCHOF, S. BLACKFORD, J. DEMMEL, J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, AND D. SORENSEN, *LAPACK Users' Guide*, Third Edition, SIAM, Philadelphia, 1999.
<http://www.netlib.org/lapack>
- [7] J. W. DEMMEL, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.
- [8] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Third Edition, The Johns Hopkins University Press, Baltimore, 1996.
- [9] G. W. STEWART, *Introduction to Matrix Computations*, Academic Press, New York, 1973.
- [10] G. W. STEWART, *Matrix Algorithms: Basic Decompositions*, SIAM, Philadelphia, 1998.
- [11] L. N. TREFETHEN AND D. BAU, III, *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.