Solution Manual for Scientific Computing with Case Studies

Dianne P. O'Leary ©2008

January 13, 2009

Unit I

SOLUTIONS: Preliminaries: Mathematical Modeling, Errors, Hardware and Software

Chapter 1

Solutions: Errors and Arithmetic

CHALLENGE 1.1.

(a) This is true. The integers are equally spaced, with distance equal to 1, and you can easily generate examples.

(b) This is only *approximately* true.

- For example, with a 53-bit mantissa, if $\texttt{r} = 1$, then $\texttt{f(r)} = 1 + 2^{-52}$, for a relative distance of 2^{-52}.
- Similarly, if $\texttt{r} = 8 = 1000_2$, then $\texttt{f(r)} = 1000_2 + 2^{-52+3}$, for a relative distance of $2^{-49}/2^3 = 2^{-52}$.
- But if $\texttt{r} = 1.25 = 1.01_2$, then $\texttt{f(r)} = 1.25 + 2^{-52}$, for a relative distance of $2^{-52}/1.25$.

In general, suppose we have a machine-representable number $\texttt{r}$ with positive mantissa z and exponent p. Then $\texttt{f(r)} = (z + 2^{-52}) \times 2^p$, so the relative distance is

$$\frac{(z + 2^{-52}) \times 2^p - (z) \times 2^p}{z \times 2^p} = \frac{2^{-52}}{z}.$$

Because $1 \le z < 2$, the relative distance is always between 2^{-52} and 2^{-53}, constant within a factor of 2. A similar argument holds for negative mantissas.

CHALLENGE 1.2.

(a) The machine number just larger than 1.0000 is 1.0001, so machine epsilon is 10^{-4}.

(b) The smallest positive normalized mantissa is 1.0000, and the smallest exponent is -9999, so the number is 1×10^{-9999}. (Note that this is much smaller than machine epsilon.)

CHALLENGE 1.3.

(a) The machine number just larger than 1.00000 is 1.00001_2. Therefore, for this machine, machine epsilon is $.00001_2 = 2^{-5}$.

(b) The smallest positive normalized mantissa is 1.00000_2, and the smallest exponent is $-1111_2 = -15$, so the smallest positive number is 2^{-15}.

(c) $1/10 = 1.1001100..._2 \times 2^{-4}$, so the mantissa is +1.10011 and the exponent is $-4 = -0100_2$.

CHALLENGE 1.4.

(a) The number `delta` is represented with a mantissa of 1 and an exponent of -53. When this is added to 1 the first time through the loop, the sum has a mantissa with 54 digits, but only 53 can be stored, so the low-order 1 is dropped and the answer is stored as 1. This is repeated 2^{20} times, and the final value of `x` is still 1.

(b) By mathematical reasoning, we have a loop that never terminates. In floating-point arithmetic, the loop is executed 1024 times, since eventually both `x` and `twox` are equal to the floating-point value `Inf`.

(c) There are very many possible answers. For example, for the associative case, we might choose `x` = 1 and choose `y` = `z` so that `x + y` = 1 but `x + 2 y` > 1. Then `(x + y) + z < x + (y + z)`.

(d) Again there are very many possible examples, including `0/0 = NaN and -1/0 = -Inf`.

(e) If `x` is positive, then the next floating-point number bigger than `x` is produced by adding `1` to the lowest-order bit in the mantissa of `x`. This is ϵ_m times 2 to the exponent of `x`, or approximately ϵ_m times `x`.

CHALLENGE 1.5. The problem is:

```
A = [2 1; 1.99 1];
b = [1;-1];
x = A \ b;
```

The problem with different units is:

```
C = [A(1,:)/100; A(2,:)]
d = [b(1)/100; b(2)]
z = C  \ d
```

The difference is `x-z = 1.0e-12 * [-0.4263; 0.8527]`

The two linear systems have exactly the same solution vector. The reason that the computed solution changed is that rescaling

- increased the rounding error in the first row of the data.
- changed the pivot order for Gauss elimination, so the computer performed a different sequence of arithmetic operations.

The quantity cond($\boldsymbol{C}$)ϵ_{mach} times the size of $\boldsymbol{b}$ is an estimate for the size of the change.

CHALLENGE 1.6.

(a)

$$\frac{|\tilde{x} - x|}{|x|} = \frac{|x(1-r) - x|}{|x|} = |r|.$$

The computation for y is similar.

(b)

$$\begin{aligned} \left|\frac{\tilde{x}\tilde{y} - xy}{xy}\right| &= \left|\frac{x(1-r)y(1-s) - xy}{xy}\right| \\ &= \left|\frac{xy(rs - r - s)}{xy}\right| \\ &\le |r| + |s| + |rs|\,. \end{aligned}$$

CHALLENGE 1.7. No.

- .1 is not represented exactly, so error occurs in each use of it.
- If we repeatedly add the machine value of .1, the exact machine value for the answer does not always fit in the same number of bits, so additional error is made in storing the answer. (Note that this error would occur even if .1 were represented exactly.)

(This was the issue in the Patriot Missile failure
`http://www.ima.umn.edu/~arnold/disasters/patriot.html`.)

CHALLENGE 1.8. No answer provided.

CHALLENGE 1.9.

(a) Ordinarily, relative error bounds add when we do multiplication, but the dominant error in this computation was the rounding of the answer from $3.2 \times 4.5 = 14.4$ to 14. (Perhaps we were told that we could store only 2 decimal digits.) Therefore, one way to express the forward error bound is that the true answer lies between 13.5 and 14.5.

(b) There are many correct answers. For example, we have exactly solved the problem $3.2 \times (14/3.2)$, or 3.2×4.37, so we have changed the second piece of data by 0.13.

CHALLENGE 1.10. Notice that $\boldsymbol{x}_c$ solves the linear system

$$\begin{bmatrix} 2 & 1 \\ 3 & 6 \end{bmatrix} \boldsymbol{x}_c = \begin{bmatrix} 5 \\ 21 \end{bmatrix},$$

so we have solved a linear system whose right-hand side is perturbed by

$$\boldsymbol{r} = \begin{bmatrix} 0.244 \\ 0.357 \end{bmatrix}.$$

The norm of $\boldsymbol{r}$ gives a bound on the change in the data, so it is a backward error bound.

(The true solution is $\boldsymbol{x}_{true} = [1.123, 2.998]^T$, and a forward error bound would be computed from $\|\boldsymbol{x}_{true} - \boldsymbol{x}_c\|$.)

CHALLENGE 1.11. The estimated volume is $3^3 = 27$ m^3.
The relative error in a side is bounded by $z = .005/2.995$.
Therefore, the relative error in the volume is bounded by $3z$ (if we ignore the high-order terms), so the absolute error is bounded by $27 * 3z \approx 27 * .005 = .135$ m^3.

CHALLENGE 1.12. Throwing out the imaginary part or taking $\pm$ the absolute value is dangerous, unless the imaginary part is within the desired error tolerance. You could check how well the real part of the computed solution satisfies the equation. It is probably best to try a different method; you have an answer that you believe is close to the true one, so you might, for example, use your favorite algorithm to solve $\min_x (f(x))^2$ using the real part of the previously computed solution as a starting guess.

Chapter 2

Solutions: Sensitivity Analysis: When a Little Means a Lot

CHALLENGE 2.1.

(a)

$$2x\,\mathrm{d}x + b\,\mathrm{d}x + x\,\mathrm{d}b = 0,$$

so

$$\frac{\mathrm{d}x}{\mathrm{d}b} = -\frac{x}{2x+b}.$$

(b) Differentiating we obtain

$$\frac{\mathrm{d}x}{\mathrm{d}b} = -\frac{1}{2} \pm \frac{1}{2}\frac{b}{\sqrt{b^2-4c}}.$$

Substituting the roots $x_{1,2}$ into the expression obtained in (a) gives the same result as this.

(c) The roots will be most sensitive when the derivative is large, which occurs when the discriminant $\sqrt{b^2-4c}$ is almost zero, and the two roots almost coincide. In contrast, a root will be insensitive when the derivative is close to zero. In this case, the root itself may be close to zero, so although the absolute change will be small, the relative change may be large.

CHALLENGE 2.2. The solution is given in `exlinsys.m`, found on the website, and the results are shown in Figure 2.1. From the top graphs, we see that if we "wiggle" the coefficients of the first linear system a little bit, then the intersection of the two lines does not change much; in contrast, since the two equations for the second linear system almost coincide, small changes in the coefficients can move the intersection point a great deal.

The middle graphs show that despite this sensitivity, the solutions to the perturbed systems satisfy the original systems quite well – to within residuals of 5×10^{-4}. This means that the backward error in the solution is small; we have solved a nearby problem $\boldsymbol{Ax} = \boldsymbol{b} + \boldsymbol{r}$ where the norm of $\boldsymbol{r}$ is small. This is characteristic of Gauss elimination, even on ill-conditioned problems.

The bottom graphs are quite different, though. The changes in the solutions $\boldsymbol{x}$ for the first system are all of the same order as the residuals, but for the second system they are nearly 500 times as big as the perturbation. Note that for the well-conditioned problem, the solutions give a rather circular cloud of points, whereas for the ill-conditioned problem, there is a direction, corresponding to the right singular vector for the small singular value, for which large perturbations occur.

The condition number of each matrix captures this behavior; it is about 2.65 for the first matrix and 500 for the second, so we expect that changes in the right-hand side for the second problem might produce a relative change 500 times as big in the solution $\boldsymbol{x}$.

CHALLENGE 2.3. The solution is given in `exlinpro.m`, and the results are shown in Figure 2.2. For the first example, the Lagrange multiplier predicts that the change in $\boldsymbol{c}^T\boldsymbol{x}$ should be about 3 times the size of the perturbation, and that is confirmed by the Monte Carlo experiments. The Lagrange multipliers for the other two examples (1.001 and 100 respectively) are also good predictors of the change in the function value. But note that something odd happens in the second example. Although the function value is not very sensitive to perturbations, the solution vector $\boldsymbol{x}$ is quite sensitive; it is sometimes close to $[0, 1]$ and sometimes close to $[1, 0]$! The solution to a (nondegenerate) linear programming problem must occur at a vertex of the feasible set. In our unperturbed problem there are three vertices: $[0, 1]$, $[1, 0]$, and $[0, 0]$. Since the gradient of $\boldsymbol{c}^T\boldsymbol{x}$ is almost parallel to the constraint $\boldsymbol{Ax} \leq \boldsymbol{b}$, we sometimes find the solution at the first vertex and sometimes at the second.

Therefore, in optimization problems, even if the function value is relatively stable, we may encounter situations in which the solution parameters have very large changes.

CHALLENGE 2.4. The results are computed by `exode.m` and shown in Figure 2.3. The Monte Carlo results predict that the growth is likely to be between 1.4 and 1.5. The two black curves, the solution to part (a), give very pessimistic upper and lower bounds on the growth: 1.35 and 1.57. This is typical of forward error bounds. Notice that the solution is the product of exponentials,

$$y(50) = \prod_{\tau=1}^{\tau=49} e^{a(\tau)},$$

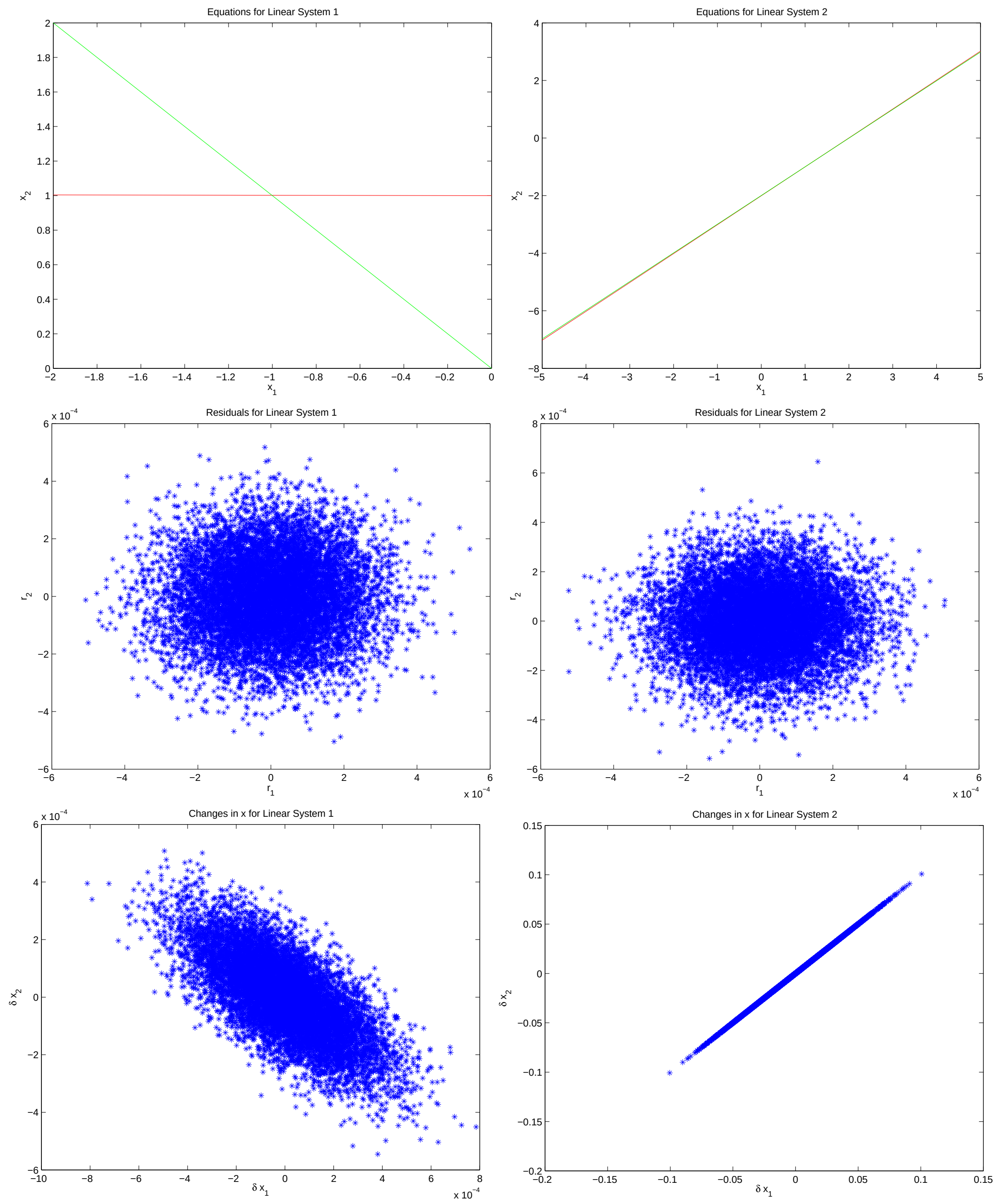


Figure 2.1. *Results of Challenge 2.2. The example on the left is typical of well-conditioned problems, while the example on the right is ill-conditioned, so the solution is quite sensitive to noise in the data. The graphs at top plot the linear equations, those in the middle plot residuals to perturbed systems, and those on the bottom plot the corresponding solution vectors.*

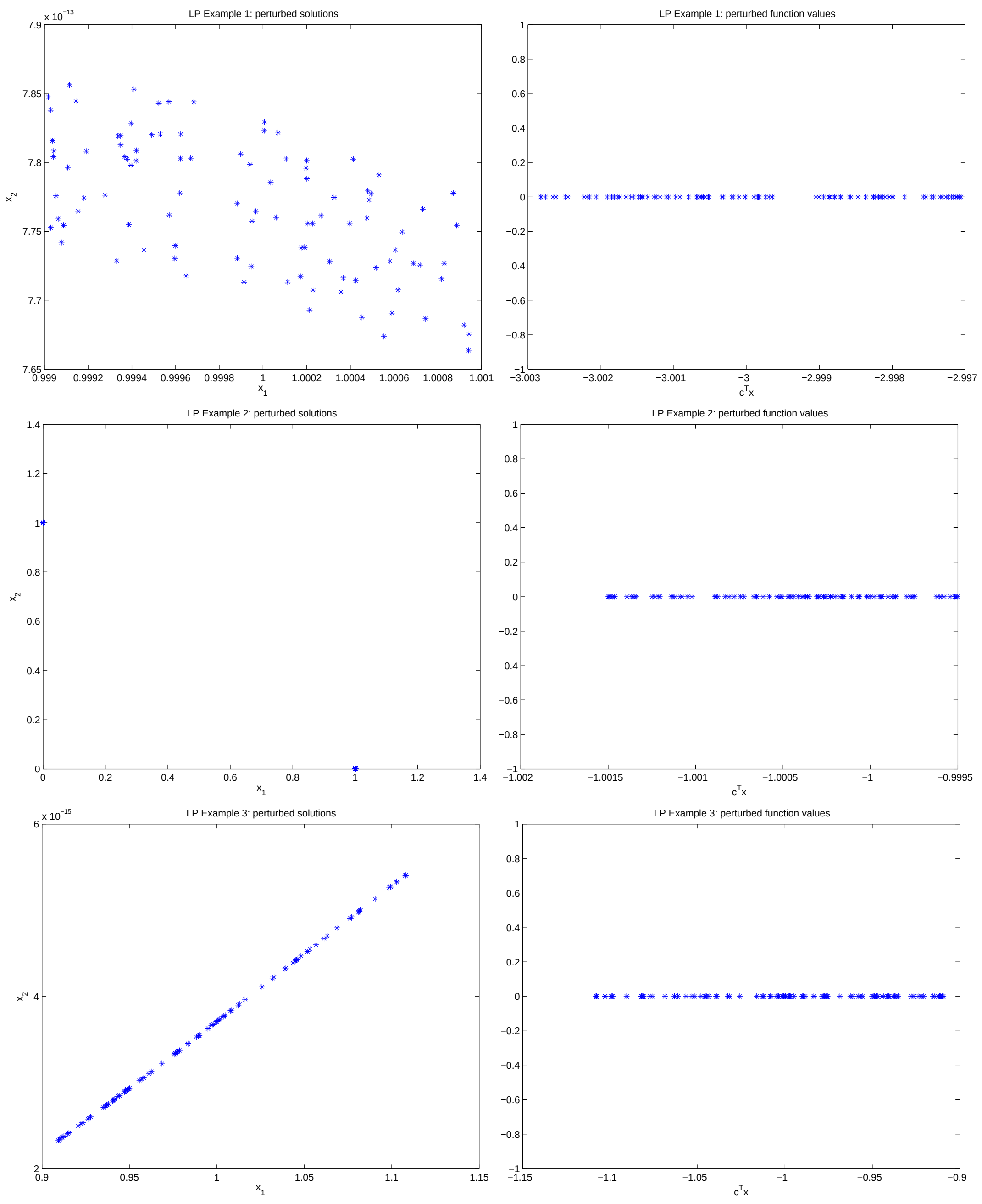


Figure 2.2. *Results of Challenge 2.3. The graphs on the left plot the perturbed solutions to the three problems, while those on the right plot the optimal function values. The optimal function values for the three problems are increasingly sensitive to small changes in the data. Note the vastly different vertical scales in the graphs on the left.*

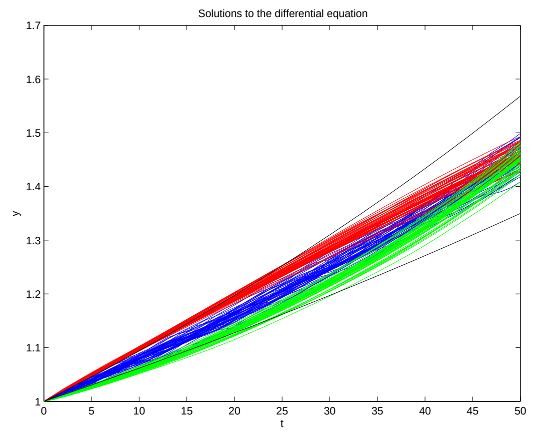


Figure 2.3. *Results of Challenge 2.4. The black curves result from setting* $a = 0.006$ *and* $a = 0.009$*. The blue curves have random rates chosen for each year. The red curves are the results of trials with the random rates ordered with largest to smallest. For the green curves, the rates were ordered smallest to largest.*

where $a(\tau)$ is the growth rate in year τ. Since exponentials commute, the final population is invariant with respect to the ordering of the rates, but the intermediate population (and thus the demand for social services and other resources) is quite different under the two assumptions.

CHALLENGE 2.5. The solution is given in `exlinsys.m`. The confidence intervals for the first example are

$$x_1 \in [-1.0228, -1.0017], \quad x_2 \in [1.0018, 1.0022]$$

and for the second example are

$$x_1 \in [0.965, 1.035], \quad x_2 \in [-1.035, -0.965].$$

Those for the second example are 20 times larger than for the first, since they are related to the size of $\boldsymbol{A}^{-1}$, but in both cases about 95% of the samples lie within the intervals, as expected.

Remember that these intervals should be calculated using a Cholesky decomposition or the backslash operator. Using `inv` or raising a matrix to the -1 power is slower when n is large, and generally is less accurate, as discussed in Chapter 5.

Chapter 3

Solutions: Computer Memory and Arithmetic: A Look Under the Hood

CHALLENGE 3.1. See `problem1.m` on the website.

CHALLENGE 3.2. The counts of the number of blocks moved are summarized in the following table:

		Dot-product	Total	Saxpy	Total
column	$\boldsymbol{A}$	32 x 128		128/8 x 32	
oriented	$\boldsymbol{x}$	32/8 x 128	4624	32/8 x 1	1028
storage	$\boldsymbol{y}$	128/8 x 1		128/8 x 32	
row	$\boldsymbol{A}$	32/8 x 128		128 x 32	
oriented	$\boldsymbol{x}$	32/8 x 128	1040	32/8 x 1	4612
storage	$\boldsymbol{y}$	128/8 x 1		128/8 x 32	

Therefore, for good performance, we should use the dot-product formulation if storage is row-oriented and the saxpy formulation if storage is column-oriented.

CHALLENGE 3.3. No answer provided.

CHALLENGE 3.4. Consider $m = 16$, $s = 1$. We access each of the 16 elements 16 times, and we have 2 cache misses, one for each block of 8 elements. So the total time is $256 + 2 * 16$ ns for the 256 accesses, for an average of 1.125 ns. When s is increased to 16, we access only $z(1)$, so the total time drops to $256 + 16$ ns.

For $m = 64$, the array no longer fits in cache and each block that we use must be reloaded for each cycle. For $s = 4$, we have a cache miss for every other access to the array, so the average access time is $(1 + 16/2) = 9$ ns.

The other entries are similar.

CHALLENGE 3.5. The data cannot be fully explained by our simple model, since, for example, this machine uses prefetching, two levels of cache, and a more complicated block replacement strategy than the least-recently-used one that we discussed. Some of the parameters can be extracted using our model, though.

The discontinuity in times as we pass from $m = 2^{14}$ to $m = 2^{15}$ indicates that the capacity of the cache is 2^{14} (single-precision) words (2^{16} bytes).

The discontinuity between stride $s = 2^3$ and $s = 2^4$ says that $\ell = 2^3$ words, so $b = 2^{11}$ words.

The elements toward the bottom left corner of the table indicate that $\alpha \approx 3$ ns.

The block of entries for $m \geq 2^{15}$ and $2^6 \leq s \leq 2^{10}$ indicates that perhaps $\mu \approx 18 - 3 = 15$ ns.

To further understand the results, consult a textbook on computer organization and the *UltraSPARC III Cu User's Manual* at
`http://www.sun.com/processors/manuals/USIIIv2.pdf`.

CHALLENGE 3.6. On my machine, the time for floating-point arithmetic is of the order of the time μ for a cache miss penalty. This is why misses noticeably slow down the execution time for matrix operations.

CHALLENGE 3.7. No answer provided.

CHALLENGE 3.8. No answer provided.

Chapter 4

Solutions: Design of Computer Programs: Writing Your Legacy

CHALLENGE 4.1. See `posteddoc.m` on the website.

CHALLENGE 4.2.

- Data that a function needs should be specified in variables, not constants. This is fine; `C` is a variable.
- Code should be modular, so that a user can pull out one piece and substitute another when necessary. The program `posted` factors a matrix into the product of two other matrices, and it would be easy to substitute a different factorization algorithm.
- On the other hand, there is considerable overhead involved in function calls, so each module should involve a substantial computation in order to mask this overhead. This is also satisfied; `posted` performs a significant computation ($O(mn^2)$ operations).
- Input parameters should be tested for validity, and clear error messages should be generated for invalid input. The factorization can be performed for any matrix or scalar, so input should be tested to be sure it is not a string, cell variable, etc.
- "Spaghetti code" should be avoided. In other words, the sequence of instructions should be top-to-bottom (including loops), without a lot of jumps in control. This is fine, although there is a lot of nesting of loops.
- The names of variables should be chosen to remind the reader of their purpose. The letter `q` is often used for an orthogonal matrix, and `r` is often used for an upper triangular one, but it would probably be better practice to use uppercase names for these matrices.

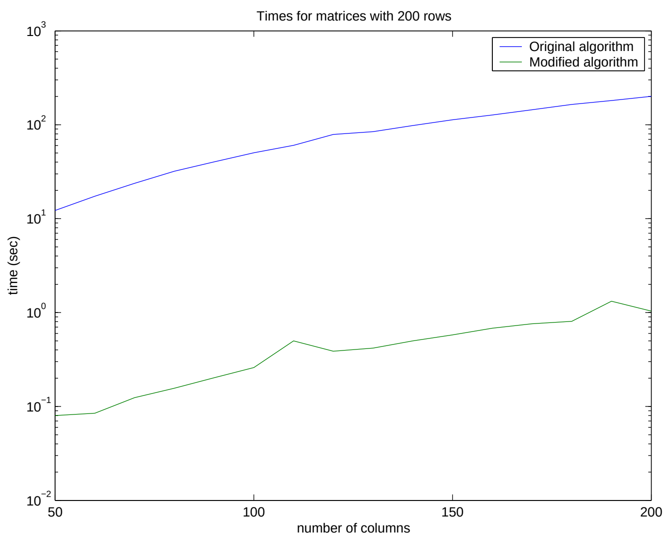


Figure 4.1. *Time taken by the two algorithms for matrices with 200 rows.*

CHALLENGE 4.3.

(a) This program computes a QR decomposition of the matrix `C` using the modified Gram-Schmidt algorithm.

(b) See website.

(c) This is corrected in `postedfact.m` on the website. The columns of `q` should be mutually orthogonal, but the number of columns in `q` should be the minimum of the row and column dimensions of `C`. Nonzero columns after that are just the result of rounding errors.

CHALLENGE 4.4. The resulting program is on the website, and the timing results are shown in Figure 4.1. The program `posted` has been modified in `postedfac` to use vector operations, use internal functions like `norm` when possible, and preallocate storage for `Q` and `R`. The `postedfact` function runs 150-200 times faster than `posted` on matrices with 200 rows, using a Sun UltraSPARC-III with clock speed 750 MHz running MATLAB 6. It is an interesting exercise to determine the relative importance of the three changes.

You also might think about how an efficient implementation in your favorite programming language might differ from this one.

Unit II

SOLUTIONS: Dense Matrix Computations

Chapter 5

Solutions: Matrix Factorizations

CHALLENGE 5.1.

```
s = zeros(m,1);
for j=1:n,
   s = s + abs(A(:,j));
end
```

Compare with:

```
for i=1:m,
    s(i) = norm(A(i,:),1);
end
```

CHALLENGE 5.2.

$$3x_2 = 6 \rightarrow x_2 = 2,$$
$$2x_1 + 5x_2 = 8 \rightarrow x_1 = \frac{1}{2}(8 - 10) = -1.$$

The determinant is $2 * 3 = 6$.

CHALLENGE 5.3.

```
x = b;
detA = 1;
for i=1:n,
   x(i) = x(i) / A(i,i);
   x(i+1:n) = x(i+1:n) - A(i+1:n,i)*x(i);
   detA = detA * A(i,i);
end
```

CHALLENGE 5.4.

(a)

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} \ell_{11} & \ell_{21} & \ell_{31} \\ 0 & \ell_{22} & \ell_{32} \\ 0 & 0 & \ell_{33} \end{bmatrix}$$
$$= \begin{bmatrix} \ell_{11}^2 & \ell_{11}\ell_{21} & \ell_{11}\ell_{31} \\ \ell_{21}\ell_{11} & \ell_{21}^2 + \ell_{22}^2 & \ell_{21}\ell_{31} + \ell_{22}\ell_{32} \\ \ell_{31}\ell_{11} & \ell_{31}\ell_{21} + \ell_{32}\ell_{22} & \ell_{31}^2 + \ell_{32}^2 + \ell_{33}^2 \end{bmatrix}$$

(b) The MATLAB function should use the following algorithm.

```
for $i = 1 : n$
  $\ell_{ii} = \sqrt{a_{ii} - \sum_{j=1}^{i-1} \ell_{ij}^2}$
  for $j = i+1 : n$
    $\ell_{ji} = (a_{ji} - \sum_{k=1}^{i-1} \ell_{jk}\ell_{ik})/\ell_{ii}$
  end
end
```

CHALLENGE 5.5. We compute a Givens matrix by setting

$$c = \frac{3}{\sqrt{9+16}}, \quad s = \frac{4}{\sqrt{25}}.$$

Then if

$$\boldsymbol{G} = \begin{bmatrix} 3/5 & 4/5 \\ -4/5 & 3/5 \end{bmatrix},$$

then

$$\boldsymbol{G} \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

so $z = 5$, the norm of the original vector.

CHALLENGE 5.6.

```
for i=1:3,

    W = planerot(A(i:i+1,i));

% Note that the next instruction just operates on the part
% of A that changes.  It is wasteful to do multiplications
% on the rest.

    A(i:i+1,i:n) = W * A(i:i+1,i:n);

end
```

CHALLENGE 5.7. For $\boldsymbol{G}$ to be unitary, we need

$$\begin{aligned} \boldsymbol{I} &= \boldsymbol{G}^* \boldsymbol{G} \\ &= \begin{bmatrix} c & -s \\ \bar{s} & c \end{bmatrix} \begin{bmatrix} c & s \\ -\bar{s} & c \end{bmatrix} \\ &= \begin{bmatrix} |c|^2 + |s|^2 & cs - sc \\ \bar{s}c - c\bar{s} & |c|^2 + |s|^2 \end{bmatrix}. \end{aligned}$$

Therefore, it is sufficient that $|c|^2 + |s|^2 = 1$.

Now

$$\boldsymbol{G}\boldsymbol{z} = \begin{bmatrix} cz_1 + sz_2 \\ -\bar{s}z_1 + cz_2 \end{bmatrix} = \begin{bmatrix} \frac{|z_1|}{\|\boldsymbol{z}\|} z_1 + sz_2 \\ -\bar{s}z_1 + \frac{|z_1|}{\|\boldsymbol{z}\|} z_2 \end{bmatrix}.$$

Therefore, if $z_1 \neq 0$, we make the second entry zero by setting

$$\bar{s} = \frac{|z_1|}{\|\boldsymbol{z}\|} \frac{z_2}{z_1}.$$

If $z_1 = 0$, we can take $s = \bar{z}_2/|\bar{z}_2|$. (The only restriction on s in this case is that its absolute value equals 1.)

CHALLENGE 5.8. Using Givens, $c = s = 3/\sqrt{18} = 1/\sqrt{2}$ so

$$\boldsymbol{G} = \boldsymbol{Q}^T = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix},$$

$$\boldsymbol{R} = \boldsymbol{Q}^T \boldsymbol{A} = \frac{1}{\sqrt{2}} \begin{bmatrix} 6 & 4 \\ 0 & -2 \end{bmatrix}.$$

Alternatively, using Gram-Schmidt orthogonalization,

$$r_{11} = \sqrt{3^2+3^2} = 3\sqrt{2},$$

$$\boldsymbol{q}_1 = \frac{1}{3\sqrt{2}}\begin{bmatrix} 3 \\ 3 \end{bmatrix}.$$

Then

$$r_{12} = \boldsymbol{q}_1^T \begin{bmatrix} 3 \\ 1 \end{bmatrix} = 4/\sqrt{2},$$

$$\widehat{\boldsymbol{q}}_2 = \begin{bmatrix} 3 \\ 1 \end{bmatrix} - 4/\sqrt{2}\boldsymbol{q}_1,$$

and r_{22} = the norm of this vector = $\sqrt{2}$, so $\boldsymbol{q}_2 = \widehat{\boldsymbol{q}}_2/\sqrt{2}$. If we complete the arithmetic, we get the same $\boldsymbol{QR}$ as above, up to choice of signs for columns of $\boldsymbol{Q}$ and rows of $\boldsymbol{R}$.

CHALLENGE 5.9. Note that after we finish the iteration $i = 1$, we have $\boldsymbol{q}_{k+1}^{new} = \boldsymbol{q}_{k+1}^{old} - r_{1,k+1}\boldsymbol{q}_1$, so

$$\boldsymbol{q}_1^* \boldsymbol{q}_{k+1}^{new} = \boldsymbol{q}_1^* \boldsymbol{q}_{k+1}^{old} - r_{1,k+1}\boldsymbol{q}_1^*\boldsymbol{q}_1 = 0$$

by the definition of $r_{1,k+1}$ and the fact that $\boldsymbol{q}_1^*\boldsymbol{q}_1 = 1$.

Assume that after we finish iteration $i = j-1$, for a given value of k, we have $\boldsymbol{q}_\ell^*\boldsymbol{q}_{k+1} = 0$ for $\ell \le j-1$ and $\boldsymbol{q}_j^*\boldsymbol{q}_\ell = 0$ for $j < \ell \le k$. After we finish iteration $i = j$ for that value of k, we have $\boldsymbol{q}_j^*\boldsymbol{q}_{k+1}^{new} = 0$ by the same argument we used above, and we also have that $\boldsymbol{q}_\ell^*\boldsymbol{q}_{k+1}^{new} = 0$, for $\ell \le j-1$, since all we have done to $\boldsymbol{q}_{k+1}$ is to add a multiple of $\boldsymbol{q}_j$ to it, and $\boldsymbol{q}_j$ is orthogonal to $\boldsymbol{q}_\ell$. Thus, after iteration j, $\boldsymbol{q}_\ell^*\boldsymbol{q}_{k+1}^{new} = 0$ for $\ell \le j$, and the induction is complete when $j = k$ and $k = n-1$.

CHALLENGE 5.10.

(a) We verify that $\boldsymbol{Q}$ is unitary by showing that its conjugate transpose is its inverse:

$$\begin{aligned}\boldsymbol{Q}^*\boldsymbol{Q} &= (\boldsymbol{I} - 2\boldsymbol{u}\boldsymbol{u}^*)(\boldsymbol{I} - 2\boldsymbol{u}\boldsymbol{u}^*) \\ &= \boldsymbol{I} - 4\boldsymbol{u}\boldsymbol{u}^* + 4\boldsymbol{u}\boldsymbol{u}^*\boldsymbol{u}\boldsymbol{u}^* \\ &= \boldsymbol{I},\end{aligned}$$

since $\boldsymbol{u}^*\boldsymbol{u} = 1$. For the second part, we compute

$$\begin{aligned}\boldsymbol{v}^*\boldsymbol{z} &= (\boldsymbol{z}^* - \alpha^*\boldsymbol{e}_1^T)\boldsymbol{z} \\ &= \boldsymbol{z}^*\boldsymbol{z} - \alpha^* z_1 \\ &= \boldsymbol{z}^*\boldsymbol{z} - e^{-i\theta}\|\boldsymbol{z}\|e^{i\theta}\zeta \\ &= \boldsymbol{z}^*\boldsymbol{z} - \|\boldsymbol{z}\|\zeta,\end{aligned}$$

and

$$\begin{aligned}\|\boldsymbol{v}\|^2 &= (\boldsymbol{z}^* - \alpha^* \boldsymbol{e}_1^T)(\boldsymbol{z} - \alpha \boldsymbol{e}_1)\\ &= \boldsymbol{z}^*\boldsymbol{z} - \alpha^* z_1 - \alpha z_1^* + \alpha^*\alpha\\ &= \boldsymbol{z}^*\boldsymbol{z} - e^{-i\theta}\|\boldsymbol{z}\|e^{i\theta}\zeta - e^{i\theta}\|\boldsymbol{z}\|e^{-i\theta}\zeta + \|\boldsymbol{z}\|^2\\ &= 2\boldsymbol{z}^*\boldsymbol{z} - 2\|\boldsymbol{z}\|\zeta.\end{aligned}$$

Then

$$\begin{aligned}\boldsymbol{Qz} &= (\boldsymbol{I} - 2\boldsymbol{uu}^*)\boldsymbol{z}\\ &= (\boldsymbol{I} - \frac{2}{\|\boldsymbol{v}\|^2}\boldsymbol{vv}^*)\boldsymbol{z}\\ &= \boldsymbol{z} - \frac{2\boldsymbol{v}^*\boldsymbol{z}}{\|\boldsymbol{v}\|^2}\boldsymbol{v}\\ &= \boldsymbol{z} - \boldsymbol{v}\\ &= \alpha\boldsymbol{e}_1.\end{aligned}$$

(b) Let the second column of $\boldsymbol{A}_1$ be $[a, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{n-1}]^T$. Use the vector $\boldsymbol{v}$ to form the Householder transformation. Then the product $\boldsymbol{QA}_1$ leaves the first column of $\boldsymbol{A}_1$ unchanged and puts zeros below the main diagonal in the second column.

(c) Assume $m > n$. (The other case is similar and left to the reader.)

```
Initially, let $\boldsymbol{R} = \boldsymbol{A}$ and $\boldsymbol{Q} = \boldsymbol{I}$ (dimension $m$).
**for** $j = 1 : n$,
  (1) Let $\boldsymbol{z} = [r_{jj}, \ldots r_{mj}]^T$.
  (2) Let the polar coordinate representation of $z_1$ be $e^{i\theta}\zeta$.
  (3) Define $\boldsymbol{v} = \boldsymbol{z} - \alpha\boldsymbol{e}_1$ where $\alpha = -e^{i\theta}\|\boldsymbol{z}\|$.
  (4) Let $\boldsymbol{u} = \boldsymbol{v}/\|\boldsymbol{v}\|$, and define the Householder transformation by $\widehat{\boldsymbol{Q}} = \boldsymbol{I} - 2\boldsymbol{uu}^*$.
  (5) Apply the transformation to $\boldsymbol{R}$ by setting $\boldsymbol{R}(j : m, j : n) = \boldsymbol{R}(j : m, j : n) - 2\boldsymbol{u}(\boldsymbol{u}^*\boldsymbol{R}(j : m, j : n))$.
  (6) Update the matrix $\boldsymbol{Q}$ by $\boldsymbol{Q}(j : m, j : m) = \boldsymbol{Q}(j : m, j : m) - 2(\boldsymbol{Q}(j : m, j : m)\boldsymbol{u})\boldsymbol{u}^*$.
**end**
```

Note that we used the associative law in the updates to $\boldsymbol{Q}$ and $\boldsymbol{R}$ to avoid ever forming $\widehat{\boldsymbol{Q}}$ and to reduce the arithmetic cost.

(d) Let $k = m - j + 1$. Then the cost per step is:

(1) No multiplications.
(2) $O(1)$ multiplications.
(3) k multiplications.
(4) k divisions and k multiplications to form $\boldsymbol{u}$.
(5) $2k(n - j + 1)$ multiplications.
(6) $3k^2$ multiplications.

We need to sum this from $j = 1$ to n, but we can neglect all but the highest order terms (mn^3 and n^3), so only the cost of steps (5) and (6) are significant. For (5) we get

$$\sum_{j=1}^{n} 2(m-j+1)(n-j+1) \approx mn^2 - \frac{1}{3}n^3,$$

since $\sum_{j=1}^{n} j \approx n^2/2$ and $\sum_{j=1}^{n} j^2 \approx n^3/3$. When $m = n$, this reduces to $2n^3/3 + O(n^2)$ multiplications. Determining the cost of (6) is left to the reader.

For completeness, we include the operations counts for the other algorithms:

Householder (***R*** only): for columns $i = 1 : n$, each entry of the submatrix of $\boldsymbol{A}$ is used once, and then we compute an outer product of the same size.

$$\sum_{i=1}^{n} 2(m-i)(n-i) \approx mn^2 - n^3/3.$$

Givens (***R*** only): for columns $j = 1 : n$, for rows $i = j+1 : m$, we count the cost of multiplying a matrix with 2 rows and $(n-j)$ columns by a Givens matrix.

$$\sum_{j=1}^{n} \sum_{i=j+1}^{m} 4(n-j) \approx 2mn^2 - 2n^2/3$$

Gram-Schmidt: At each step $k = 1 : n-1$ there is one inner product and one axpy of vectors of length m.

$$\sum_{k=1}^{n-1} \sum_{i=1}^{k} 2m \approx mn^2$$

CHALLENGE 5.11.

- Suppose $z = \|\boldsymbol{b} - \boldsymbol{A}\tilde{\boldsymbol{x}}\| \le \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|$ for all values of $\boldsymbol{x}$. Then by multiplying this inequality by itself we see that $z^2 = \|\boldsymbol{b} - \boldsymbol{A}\tilde{\boldsymbol{x}}\|^2 \le \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|^2$, so $\tilde{\boldsymbol{x}}$ is also a minimizer of the square of the norm.
- Since $\boldsymbol{Q}\boldsymbol{Q}^* = \boldsymbol{I}$, we see that $||\boldsymbol{Q}^*\boldsymbol{y}||_2^2 = (\boldsymbol{Q}^*\boldsymbol{y})^*(\boldsymbol{Q}^*\boldsymbol{y}) = \boldsymbol{y}^*\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{y} = \boldsymbol{y}^*\boldsymbol{y} = ||\boldsymbol{y}||_2^2$, Since norms are nonnegative quantities, take the square root and conclude that $||\boldsymbol{Q}^*\boldsymbol{y}||_2 = ||\boldsymbol{y}||_2$.
- Suppose $\boldsymbol{y}_1$ contains the first p components of the m-vector $\boldsymbol{y}$. Then

$$\begin{aligned} \|\boldsymbol{y}\|_2^2 &= \sum_{j=1}^{m} |y_j|^2 \\ &= \sum_{j=1}^{p} |y_j|^2 + \sum_{j=p+1}^{m} |y_j|^2 \\ &= \|\boldsymbol{y}_1\|_2^2 + \|\boldsymbol{y}_2\|_2^2. \end{aligned}$$

CHALLENGE 5.12. Define

$$\boldsymbol{c} = \boldsymbol{Q}^* \boldsymbol{b} = \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{bmatrix}, \boldsymbol{R} = \begin{bmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0} \end{bmatrix},$$

where $\boldsymbol{c}_1$ is $n \times 1$, $\boldsymbol{c}_2$ is $(m-n) \times 1$, $\boldsymbol{R}_1$ is $n \times n$, and $\boldsymbol{0}$ is $(m-n) \times n$. Then

$$\begin{aligned} \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|^2 &= \|\boldsymbol{Q}^*(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x})\|^2 \\ &= \|\boldsymbol{c} - \boldsymbol{R}\boldsymbol{x}\|^2 \\ &= \|\boldsymbol{c}_1 - \boldsymbol{R}_1\boldsymbol{x}\|^2 + \|\boldsymbol{c}_2 - \boldsymbol{0}\boldsymbol{x}\|^2 \\ &= \|\boldsymbol{c}_1 - \boldsymbol{R}_1\boldsymbol{x}\|^2 + \|\boldsymbol{c}_2\|^2. \end{aligned}$$

To minimize this quantity, we make the first term zero by taking $\boldsymbol{x}$ to be the solution to the $n \times n$ linear system $\boldsymbol{R}_1\boldsymbol{x} = \boldsymbol{c}_1$, so we see that the minimum value of $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|$ is $\|\boldsymbol{c}_2\|$. Note that this derivation is based on the three fundamental facts proved in the previous challenge.

CHALLENGE 5.13. The two zeros of the function `y = norm([.5 .4 a; a .3 .4; .3 .3 .3]) - 1` define the endpoints of the interval. Plotting tells us that one is between 0 and 1 and the other is between 0 and -1. MATLAB's `fzero` can be used to find both roots. The roots are -0.5398 and 0.2389.

CHALLENGE 5.14. Suppose that $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\}$ form a basis for $\mathcal{S}$. Then any vector in $\mathcal{S}$ can be expressed as $\alpha_1\boldsymbol{u}_1 + \ldots + \alpha_k\boldsymbol{u}_k$. Since $\boldsymbol{A}\boldsymbol{u}_i = \lambda\boldsymbol{u}_i$, we see that $\boldsymbol{A}(\alpha_1\boldsymbol{u}_1 + \ldots + \alpha_k\boldsymbol{u}_k) = \lambda_1\alpha_1\boldsymbol{u}_1 + \ldots + \lambda_k\alpha_k\boldsymbol{u}_k$ is also in $\mathcal{S}$, since it is a linear combination of the basis vectors. Therefore, if a subset of the eigenvectors of $\boldsymbol{A}$ form a basis for $\mathcal{S}$, then $\mathcal{S}$ is an invariant subspace.

Now suppose that $\mathcal{S}$ is an invariant subspace for $\boldsymbol{A}$, so for any $\boldsymbol{x} \in \mathcal{S}$, the vector $\boldsymbol{A}\boldsymbol{x}$ is also in $\mathcal{S}$. Suppose the dimension of $\mathcal{S}$ is k, and that some vector $\boldsymbol{x} \in \mathcal{S}$ has components of eigenvectors corresponding to more than k eigenvalues:

$$\boldsymbol{x} = \sum_{j=1}^{r} \alpha_j \boldsymbol{u}_j,$$

where $r > k$ and $\alpha_j \neq 0$ for $j = 1, \ldots, r$. Consider the vectors $\boldsymbol{x}, \boldsymbol{A}\boldsymbol{x}, \ldots, \boldsymbol{A}^{r-1}\boldsymbol{x}$, all of which are in $\mathcal{S}$, since each is formed from taking $\boldsymbol{A}$ times the previous one. Then

$$\begin{bmatrix} \boldsymbol{x} & \boldsymbol{A}\boldsymbol{x} & \boldsymbol{A}^2\boldsymbol{x} & \ldots & \boldsymbol{A}^{r-1}\boldsymbol{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_1 & \boldsymbol{u}_2 & \boldsymbol{u}_3 & \ldots & \boldsymbol{u}_r \end{bmatrix} \boldsymbol{D}\boldsymbol{W},$$

where $\boldsymbol{D}$ is a diagonal matrix containing the values $\alpha_1, \ldots, \alpha_r$, and

$$\boldsymbol{W} = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \ldots & \lambda_1^{r-1} \\ 1 & \lambda_2 & \lambda_2^2 & \ldots & \lambda_2^{r-1} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & \lambda_r & \lambda_r^2 & \ldots & \lambda_r^{r-1} \end{bmatrix}.$$

Now $\boldsymbol{W}$ is a Vandermonde matrix and has full rank r, and so does the matrix formed by the $\boldsymbol{u}$ vectors. Therefore the vectors $\boldsymbol{x}, \boldsymbol{A}\boldsymbol{x}, \ldots, \boldsymbol{A}^{r-1}\boldsymbol{x}$ must form a matrix of rank r and therefore are linearly independent, which contradicts the statement that $\mathcal{S}$ has dimension $k < r$. Therefore, every vector in $\mathcal{S}$ must have components of at most k different eigenvectors, and we can take them as a basis.

CHALLENGE 5.15.

(a) Subtracting the relations

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{A}\boldsymbol{x}^{(k)} + \boldsymbol{b}$$

and

$$\boldsymbol{x}_{true} = \boldsymbol{A}\boldsymbol{x}_{true} + \boldsymbol{b},$$

we obtain

$$\boldsymbol{e}^{(k+1)} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}_{true} = \boldsymbol{A}(\boldsymbol{x}^{(k)} - \boldsymbol{x}_{true}) = \boldsymbol{A}\boldsymbol{e}^{(k)}.$$

(b) If $k = 1$, then the result holds by part (a). As an induction hypothesis, suppose

$$\boldsymbol{e}^{(k)} = \boldsymbol{A}^k \boldsymbol{e}^{(0)}.$$

Then $\boldsymbol{e}^{(k+1)} = \boldsymbol{A}\boldsymbol{e}^{(k)}$ by part (a), and substituting the induction hypothesis yields $\boldsymbol{e}^{(k+1)} = \boldsymbol{A}\boldsymbol{A}^k \boldsymbol{e}^{(0)} = \boldsymbol{A}^{k+1}\boldsymbol{e}^{(0)}$. Therefore, the result holds for all $k = 1, 2, \ldots$, by mathematical induction.

(c) Following the hint, we express

$$\boldsymbol{e}^{(0)} = \sum_{j=1}^{n} \alpha_j \boldsymbol{u}_j,$$

so, by part (b),

$$\boldsymbol{e}^{(k)} = \sum_{j=1}^{n} \alpha_j \lambda_j^k \boldsymbol{u}_j.$$

Now, if all eigenvalues λ_j lie within the unit circle, then $\lambda_j^k \to 0$ as $k \to \infty$, so $\boldsymbol{e}^{(k)} \to 0$. On the other hand, if some eigenvalue λ_ℓ is outside the unit circle, then by choosing $\boldsymbol{x}^{(0)}$ so that $\boldsymbol{e}^{(0)} = \boldsymbol{u}_\ell$, we see that $\boldsymbol{e}^{(k)} = \lambda_\ell^k \boldsymbol{u}_\ell$ does not converge to zero, since its norm $|\lambda_\ell^k| \|\boldsymbol{u}_\ell\| \to \infty$.

CHALLENGE 5.16.

Form $\boldsymbol{y} = \boldsymbol{U}^* \boldsymbol{b}$	n^2 multiplications
Form $\boldsymbol{z} = \boldsymbol{\Sigma}^{-1} \boldsymbol{y}$	n multiplications $(z_i = y_i/\sigma_i)$
Form $\boldsymbol{x} = \boldsymbol{V}\boldsymbol{z}$	n^2 multiplications
Total:	$2n^2 + n$ multiplications

CHALLENGE 5.17.

(a) The columns of $\boldsymbol{U}$ corresponding to nonzero singular values form such a basis, since for any vector $\boldsymbol{y}$,

$$\begin{aligned} \boldsymbol{A}\boldsymbol{y} &= \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*\boldsymbol{y} \\ &= \sum_{j=1}^{n} \boldsymbol{u}_j (\boldsymbol{\Sigma}\boldsymbol{V}^*\boldsymbol{y})_j \\ &= \sum_{\sigma_j > 0} \boldsymbol{u}_j (\boldsymbol{\Sigma}\boldsymbol{V}^*\boldsymbol{y})_j, \end{aligned}$$

so any vector $\boldsymbol{A}\boldsymbol{y}$ can be expressed as a linear combination of these columns of $\boldsymbol{U}$. Conversely, any linear combination of these columns of $\boldsymbol{U}$ is in the range of $\boldsymbol{A}$, so they form a basis for exactly the range.

(b) Similar reasoning shows that the remaining columns of $\boldsymbol{U}$ form this basis.

CHALLENGE 5.18. We'll work through (a) using the sum-of-rank-one-matrices formulation, and (b) using the matrix formulation.

(a) The equation has a solution only if $\boldsymbol{b}$ is in the range of $\boldsymbol{A}$, so $\boldsymbol{b}$ must be a linear combination of the vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_p$. For definiteness, let $\beta_j = \boldsymbol{u}_j^* \boldsymbol{b}$, so that

$$\boldsymbol{b} = \sum_{j=1}^{p} \beta_j \boldsymbol{u}_j.$$

Let β_j, $j = p+1, \ldots, n$ be arbitrary. Then if we let

$$\boldsymbol{x} = \sum_{j=1}^{p} \frac{\beta_j}{\sigma_j} \boldsymbol{v}_j + \sum_{j=p+1}^{n} \beta_j \boldsymbol{v}_j,$$

we can verify that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

(b) Substituting the SVD our equation becomes

$$\boldsymbol{A}^*\boldsymbol{x} = \boldsymbol{V}\left[\begin{array}{cc} \boldsymbol{\Sigma}_1 & \boldsymbol{0} \end{array}\right]\boldsymbol{U}^*\boldsymbol{x} = \boldsymbol{b},$$

where $\boldsymbol{\Sigma}_1$ is $n \times n$ with the singular values on the main diagonal. Letting $\boldsymbol{y} = \boldsymbol{U}^*\boldsymbol{x}$, we see that a particular solution is

$$\boldsymbol{y}_{good} = \left[\begin{array}{c} \boldsymbol{\Sigma}_1{}^{-1}\boldsymbol{V}^*\boldsymbol{b} \\ \boldsymbol{0} \end{array}\right],$$

so

$$\boldsymbol{x}_{good} = \boldsymbol{U}\left[\begin{array}{c} \boldsymbol{\Sigma}_1{}^{-1}\boldsymbol{V}^*\boldsymbol{b} \\ \boldsymbol{0} \end{array}\right] = \sum_{j=1}^{n} \frac{\boldsymbol{v}_j^*\boldsymbol{b}}{\sigma_j}\boldsymbol{u}_j.$$

Every solution can be expressed as $\boldsymbol{x}_{good} + \boldsymbol{U}_2^*\boldsymbol{v}$ for some vector $\boldsymbol{v}$ since $\boldsymbol{A}^*\boldsymbol{U}_2^* = \boldsymbol{0}$.

CHALLENGE 5.19.

(a) Since we want to minimize

$$(c_1 - \sigma_1 w_1)^2 + \ldots + (c_n - \sigma_n w_n)^2 + c_{n+1}^2 + \ldots + c_m^2\,,$$

we set $w_i = c_i/\sigma_i = \boldsymbol{u}_i^*\boldsymbol{b}_{true}/\sigma_i$.

(b) If we express $\boldsymbol{x} - \boldsymbol{x}_{true}$ as a linear combination of the vectors $\boldsymbol{v}_1, \ldots \boldsymbol{v}_n$, then multiplying by the matrix $\boldsymbol{A}$ stretches each component by the corresponding singular value. Since σ_n is the smallest singular value, $\|\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}_{true})\|$ is bounded below by $\sigma_n\|\boldsymbol{x} - \boldsymbol{x}_{true}\|$. Therefore

$$\sigma_n\|\boldsymbol{x} - \boldsymbol{x}_{true}\| \le (\|\boldsymbol{b} - \boldsymbol{b}_{true} + \boldsymbol{r}\|),$$

and the statement follows.

For any matrix $\boldsymbol{C}$ and any vector $\boldsymbol{z}$, $\|\boldsymbol{C}\boldsymbol{z}\| \le \|\boldsymbol{C}\|\|\boldsymbol{z}\|$. Therefore, $\|\boldsymbol{b}_{true}\| = \|\boldsymbol{A}\boldsymbol{x}_{true}\| \le \|\boldsymbol{A}\|\|\boldsymbol{x}_{true}\|$.

(c) Using the given fact and the second statement, we see that

$$\|\boldsymbol{x}_{true}\| \ge \frac{1}{\sigma_1}\|\boldsymbol{b}_{true}\|.$$

Dividing the first statement by this one gives

$$\frac{\|\boldsymbol{x} - \boldsymbol{x}_{true}\|}{\|\boldsymbol{x}_{true}\|} \le \frac{\sigma_1}{\sigma_n}\frac{\|\boldsymbol{b} - \boldsymbol{b}_{true} + \boldsymbol{r}\|}{\|\boldsymbol{b}_{true}\|} = \kappa(\boldsymbol{A})\frac{\|\boldsymbol{b} - \boldsymbol{b}_{true} + \boldsymbol{r}\|}{\|\boldsymbol{b}_{true}\|}.$$

CHALLENGE 5.20.

(a) Since $\boldsymbol{U\Sigma V^*x}=\boldsymbol{b}$, we have

$$\boldsymbol{x}=\boldsymbol{V\Sigma}^{-1}\boldsymbol{U}^*\boldsymbol{b}.$$

If we let $\boldsymbol{c}=\boldsymbol{U}^*\boldsymbol{b}$, then $\alpha_1=c_1/\sigma_1$, and $\alpha_2=c_2/\sigma_2$.

(b) Here is one way to look at it. This system is very ill-conditioned. The condition number is the ratio of the largest singular value to the smallest, so this must be large. In other words, σ_2 is quite small compared to σ_1.

For the perturbed problems,

$$\boldsymbol{Ax}^{(i)}=\boldsymbol{b}-\boldsymbol{E}^{(i)}\boldsymbol{x}^{(i)},$$

so it is as if we solve the linear system with a slightly perturbed right-hand side. So, letting $\boldsymbol{f}^{(i)}=\boldsymbol{U}^*\boldsymbol{E}^{(i)}\boldsymbol{x}^{(i)}$, the computed solution is

$$\boldsymbol{x}^{(i)}=\alpha_1^{(i)}\boldsymbol{v}_1+\alpha_2^{(i)}\boldsymbol{v}_2,$$

with

$$\alpha_1^{(i)}=(c_1+f_1^{(i)})/\sigma_1,\quad \alpha_2^{(i)}=(c_2+f_2^{(i)})/\sigma_2.$$

From the figure, we know that $\boldsymbol{f}^{(i)}$ must be small, so $\alpha_1^{(i)}\approx\alpha_1$. But because σ_2 is close to zero, $\alpha_2^{(i)}$ can be quite different from α_2, so the solutions lie almost on a straight line in the direction $\boldsymbol{v}_2$.

CHALLENGE 5.21. Some possibilities:

- Any right eigenvector $\boldsymbol{u}$ of $\boldsymbol{A}$ corresponding to a zero eigenvalue satisfies $\boldsymbol{Au}=0\boldsymbol{u}=\boldsymbol{0}$. With rounding, the computed eigenvalue is not exactly zero, so we can choose the eigenvector of $\boldsymbol{A}$ corresponding to the smallest magnitude eigenvalue.
- Similarly, if $\boldsymbol{v}$ is a right singular vector of $\boldsymbol{A}$ corresponding to a zero singular value, then $\boldsymbol{Av}=\boldsymbol{0}$, so choose a singular vector corresponding to the smallest singular value.
- Let $\boldsymbol{e}_n$ be the nth column of the identity matrix. If we perform a rank-revealing QR decomposition of $\boldsymbol{A}^*$, so that $\boldsymbol{A}^*\boldsymbol{P}=\boldsymbol{QR}$, and let $\boldsymbol{q}_n$ be the last column of $\boldsymbol{Q}$, then $\boldsymbol{q}_n^*\boldsymbol{A}^*\boldsymbol{P}=\boldsymbol{q}_n^*\boldsymbol{QR}=\boldsymbol{e}_n^T\boldsymbol{R}=r_{nn}\boldsymbol{e}_n^T=\boldsymbol{0}$. Multiplying through by $\boldsymbol{P}^{-1}$ we see that $\boldsymbol{Aq}_n=\boldsymbol{0}$, so choose $\boldsymbol{z}=\boldsymbol{q}_n$.

CHALLENGE 5.22. (Partial Solution)

(a) Find the null space of a matrix: QR (fast; relatively stable) or SVD (slower but more reliable)

(b) Solve a least squares problem: QR when the matrix is well conditioned. Don't try QR if the matrix is *not* well-conditioned; use the SVD method.

(c) Determine the rank of a matrix: RR-QR (fast, relatively stable); SVD (slower but more reliable).

(d) Find the determinant of a matrix: LU with pivoting.

(e) Determine whether a symmetric matrix is positive definite: Cholesky or eigendecomposition (slower but more reliable) The $\boldsymbol{LL}^T$ version of Cholesky breaks down if the matrix has a negative eigenvalue by taking the square root of a negative number, so it is a good diagnostic. If the matrix is singular, (positive semi-definite), then we get a 0 on the main diagonal, but with rounding error, this is impossible to detect.

Chapter 6

Solutions: Case Study: Image Deblurring: I Can See Clearly Now

(coauthored by James G. Nagy)

CHALLENGE 6.1. Observe that if $\boldsymbol{y}$ is a $p \times 1$ vector and $\boldsymbol{z}$ is a $q \times 1$ vector then

$$\left\| \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{z} \end{bmatrix} \right\|_2^2 = \sum_{i=1}^{p} y_i^2 + \sum_{i=1}^{q} z_i^2 = \|\boldsymbol{y}\|_2^2 + \|\boldsymbol{z}\|_2^2.$$

Therefore,

$$\left\| \begin{bmatrix} \boldsymbol{g} \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{K} \\ \alpha \boldsymbol{I} \end{bmatrix} \boldsymbol{f} \right\|_2^2 = \left\| \begin{bmatrix} \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f} \\ -\alpha \boldsymbol{f} \end{bmatrix} \right\|_2^2 = \|\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}\|_2^2 + \|\alpha \boldsymbol{f}\|_2^2 = \|\boldsymbol{g} - \boldsymbol{K}\boldsymbol{f}\|_2^2 + \alpha^2 \|\boldsymbol{f}\|_2^2 .$$

CHALLENGE 6.2. First note that

$$\boldsymbol{Q} \equiv \begin{bmatrix} \boldsymbol{U}^T & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{V}^T \end{bmatrix}$$

is an orthogonal matrix since $\boldsymbol{Q}^T \boldsymbol{Q} = \boldsymbol{I}$, and recall that the 2-norm of a vector is invariant under multiplication by an orthogonal matrix: $\|\boldsymbol{Q}\boldsymbol{z}\|_2 = \|\boldsymbol{z}\|_2$. Therefore,

$$\begin{aligned} \left\| \begin{bmatrix} \boldsymbol{g} \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{K} \\ \alpha \boldsymbol{I} \end{bmatrix} \boldsymbol{f} \right\|_2^2 &= \left\| \begin{bmatrix} \boldsymbol{g} - \boldsymbol{K}\boldsymbol{f} \\ -\alpha \boldsymbol{f} \end{bmatrix} \right\|_2^2 \\ &= \left\| \boldsymbol{Q} \begin{bmatrix} \boldsymbol{g} - \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \boldsymbol{f} \\ -\alpha \boldsymbol{f} \end{bmatrix} \right\|_2^2 \end{aligned}$$

$$= \left\| \begin{bmatrix} \boldsymbol{U}^T \boldsymbol{g} - \boldsymbol{\Sigma} \boldsymbol{V}^T \boldsymbol{f} \\ -\alpha \boldsymbol{V}^T \boldsymbol{f} \end{bmatrix} \right\|_2^2$$

$$= \left\| \begin{bmatrix} \widehat{\boldsymbol{g}} - \boldsymbol{\Sigma} \widehat{\boldsymbol{f}} \\ -\alpha \widehat{\boldsymbol{f}} \end{bmatrix} \right\|_2^2$$

$$= \left\| \begin{bmatrix} \widehat{\boldsymbol{g}} \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{\Sigma} \\ \alpha \boldsymbol{I} \end{bmatrix} \widehat{\boldsymbol{f}} \right\|_2^2 .$$

CHALLENGE 6.3. Let's write the answer for a slightly more general case: $\boldsymbol{K}$ of dimension $m \times n$ with $m \geq n$.

$$\left\| \begin{bmatrix} \widehat{\boldsymbol{g}} \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{\Sigma} \\ \alpha \boldsymbol{I} \end{bmatrix} \widehat{\boldsymbol{f}} \right\|_2^2$$
$$= \|\widehat{\boldsymbol{g}} - \boldsymbol{\Sigma}\widehat{\boldsymbol{f}}\|_2^2 + \alpha^2 \|\widehat{\boldsymbol{f}}\|_2^2$$
$$= \sum_{i=1}^{n} (\hat{g}_i - \sigma_i \hat{f}_i)^2 + \sum_{i=n+1}^{m} \hat{g}_i^2 + \alpha^2 \sum_{i=1}^{n} \hat{f}_i^2 .$$

Setting the derivative with respect to f_i to zero we obtain

$$-2\sigma_i(\hat{g}_i - \sigma_i \hat{f}_i) + 2\alpha^2 \hat{f}_i = 0,$$

so

$$\hat{f}_i = \frac{\sigma_i \hat{g}_i}{\sigma_i^2 + \alpha^2} .$$

CHALLENGE 6.4. From Challenges 2 and 3 above, with $\alpha = 0$, the solution is

$$\hat{f}_i = \frac{\sigma_i \hat{g}_i}{\sigma_i^2} = \frac{\hat{g}_i}{\sigma_i} .$$

Note that $\hat{g}_i = \boldsymbol{u}_i^T \boldsymbol{g}$. Now, since $\boldsymbol{f} = \boldsymbol{V}\hat{f}$, we have

$$\boldsymbol{f} = \boldsymbol{v}_1 \hat{f}_1 + \ldots + \boldsymbol{v}_n \hat{\boldsymbol{f}}_n$$

and the formula follows.

CHALLENGE 6.5. See the posted program. Some comments:

- One common bug in the TSVD section: zeroing out pieces of $\boldsymbol{S}_A$ and $\boldsymbol{S}_B$. This does not zero the *smallest* singular values, and although it is very efficient in time, it gives worse results than doing it correctly.
- The data was generated by taking an original image $\boldsymbol{F}$ (posted on the website), multiplying by $\boldsymbol{K}$, and adding random noise using the MATLAB statement `G = B * F * A' + .001 * rand(256,256)`. Note that the noise prevents us from completely recovering the initial image.
- In this case, the best way to choose the regularization parameter is by eye: choose a detailed part of the image and watch its resolution for various choices of the regularization parameter, probably using bisection to find the best parameter. Figure 6.1 shows data and two reconstructions. Although both algorithms yield images in which the text can be read, noise in the data appears in the background of the reconstructions. Some nonlinear reconstruction algorithms reduce this noise.
- In many applications, we need to choose the regularization parameter by automatic methods rather than by eye. If the noise-level is known, then the *discrepancy principle* is the best: choose the parameter to make the residual $\boldsymbol{Kf} - \boldsymbol{g}$ close in norm to the expected norm of the noise. If the noise-level is not known, then *generalized cross validation* and *the L-curve* are popular methods. See [1,2] for discussion of such methods.

[1] Per Christian Hansen, James M. Nagy, and Dianne P. O'Leary. *Deblurring Images: Matrices, Spectra, and Filtering.* SIAM Press, Philadelphia, 2006.

[2] Bert W. Rust and Dianne P. O'Leary, "Residual Periodograms for Choosing Regularization Parameters for Ill-Posed Problems", *Inverse Problems,* 24 (2008) 034005.

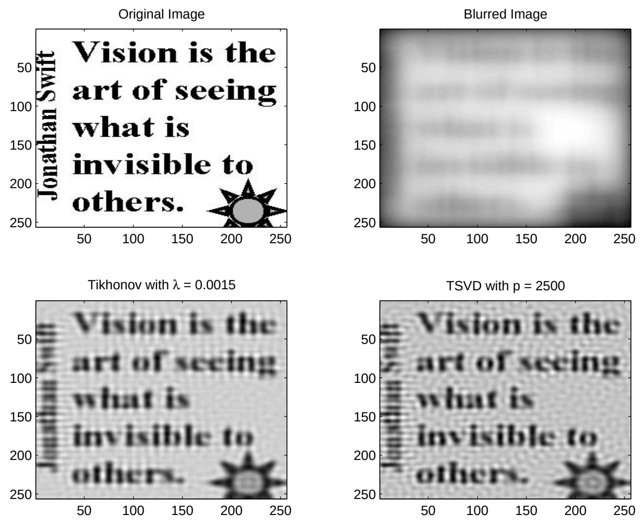


Figure 6.1. *The original image, blurred image, and results of the two algorithms*

Chapter 7

Solutions: Case Study: Updating and Downdating Matrix Factorizations: A Change in Plans

CHALLENGE 7.1.

(a) Set the columns of $\boldsymbol{Z}$ to be the differences between the old columns and the new columns, and set the columns of $\boldsymbol{V}$ to be the 6th and 7th columns of the identity matrix.

(b) The first column of $\boldsymbol{Z}$ can be the difference between the old column 6 and the new one; the second can be the 4th column of the identity matrix. The first column of $\boldsymbol{V}$ is then the 6th column of the identity matrix and the second is the difference between the old row 4 and the new one but set the 6th element to zero.

CHALLENGE 7.2.

(a) This is verified by direct computation.

(b) We use several facts to get an algorithm that is $O(kn^2)$ instead of $O(n^3)$ for dense matrices:

- $\boldsymbol{x} = (\boldsymbol{A} - \boldsymbol{Z}\boldsymbol{V}^T)^{-1}\boldsymbol{b} = (\boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{Z}(\boldsymbol{I} - \boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{Z})^{-1}\boldsymbol{V}^T\boldsymbol{A}^{-1})\boldsymbol{b}$.
- Forming $\boldsymbol{A}^{-1}$ from the LU decomposition takes $O(n^3)$ operations, but forming $\boldsymbol{A}^{-1}\boldsymbol{b}$ as $\boldsymbol{U}\backslash(\boldsymbol{L}\backslash\boldsymbol{b})$ uses forward- and back-substitution and just takes $O(n^2)$.
- $(\boldsymbol{I} - \boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{Z})$ is only $k \times k$, so factoring it is cheap: $O(k^3)$. (Forming it is more expensive, with a cost that is $O(kn^2)$.)
- Matrix multiplication is associative.

Using MATLAB notation, once we have formed `[L,U]=lu(A)`, the resulting algorithm is

```
y = U \ (L \ b);
Zh = U \ (L \ Z);
```

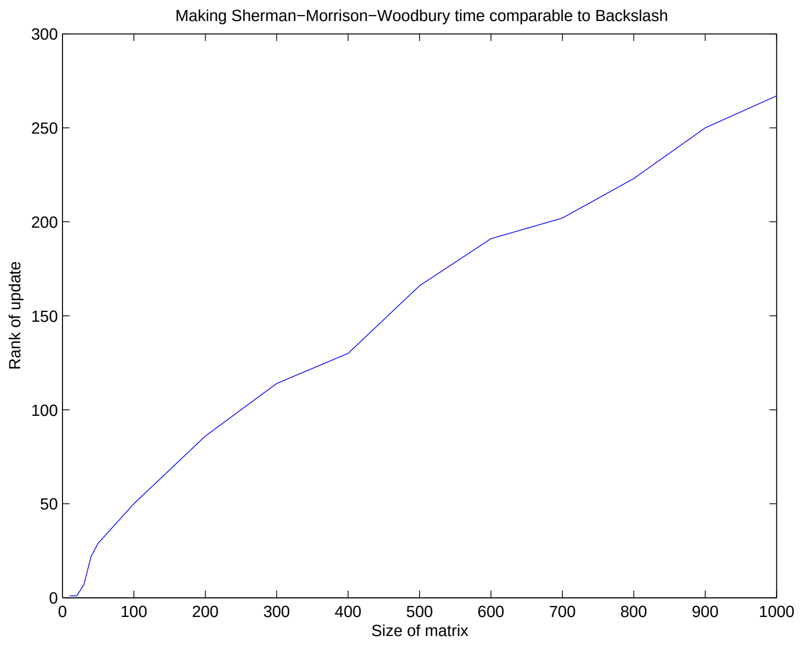


Figure 7.1. *Results of Challenge 4. The plot shows the rank k_0 of the update for which the time for using the Sherman–Morrison–Woodbury formula was approximately the same as the time for solving using factorization. Sherman–Morrison–Woodbury was faster for $n \geq 40$ when the rank of the update was less than $0.25n$.*

```
t = (eye(k) - V'*Zh) \ (V'*y);
x = y + Zh*t;
```

CHALLENGE 7.3. See `sherman_mw.m` on the website.

CHALLENGE 7.4. The solution is given in `problem4.m` on the website, and the results are plotted in the textbook. In this experiment (Sun UltraSPARC-III with clock speed 750 MHz running MATLAB 6) Sherman-Morrison-Woodbury was faster for $n \geq 40$ when the rank of the update was less than $0.25n$.

CHALLENGE 7.5.

(a) (Review)

$$\boldsymbol{Gz} = \begin{bmatrix} cz_1 + sz_2 \\ sz_1 - cz_2 \end{bmatrix} = x\boldsymbol{e}_1$$

Multiplying the first equation by c, the second by s, and adding yields

$$(c^2 + s^2)z_1 = cx\,,$$

so

$$c = z_1/x\,.$$

Similarly, we can determine that

$$s = z_2/x\,.$$

Since $c^2 + s^2 = 1$, we conclude that

$$z_1^2 + z_2^2 = x^2\,,$$

so we can take

$$c = \frac{z_1}{\sqrt{z_1^2 + z_2^2}},$$
$$s = \frac{z_2}{\sqrt{z_1^2 + z_2^2}}.$$

(b) The first rotation matrix is chosen to zero a_{61}. The second zeros the resulting entry in row 6, column 2, and the final one zeros row 6, column 3.

CHALLENGE 7.6. See `problem6.m` and `qrcolchange.m` on the website.

CHALLENGE 7.7.

$$\boldsymbol{A}_{new} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} a_{n+1,1} & \dots & a_{n+1,n} & (a_{n+1,n+1} - 1) \end{bmatrix}$$

$$+ \begin{bmatrix} a_{1,n+1} \\ a_{2,n+1} \\ \vdots \\ a_{n,n+1} \\ 0 \end{bmatrix} \begin{bmatrix} 0 & \dots & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{A} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} + \begin{bmatrix} 0 & a_{1,n+1} \\ 0 & a_{2,n+1} \\ \vdots & \vdots \\ 0 & a_{n,n+1} \\ 1 & (a_{n+1,n+1} - 1) \end{bmatrix} \begin{bmatrix} a_{n+1,1} & \dots & a_{n+1,n} & 0 \\ 0 & \dots & 0 & 1 \end{bmatrix}.$$

So we can take

$$\boldsymbol{Z} = - \begin{bmatrix} 0 & a_{1,n+1} \\ 0 & a_{2,n+1} \\ \vdots & \vdots \\ 0 & a_{n,n+1} \\ 1 & (a_{n+1,n+1} - 1) \end{bmatrix}; \boldsymbol{V}^T = \begin{bmatrix} a_{n+1,1} & \dots & a_{n+1,n} & 0 \\ 0 & \dots & 0 & 1 \end{bmatrix}.$$

Chapter 8

Solutions: Case Study: The Direction-of-Arrival Problem: Coming at You

CHALLENGE 8.1. Let $\boldsymbol{w}_k = \boldsymbol{SCz}_k$, and multiply the equation $\boldsymbol{BA\Phi SCz}_k = \lambda_k \boldsymbol{BASCz}_k$ by $(\boldsymbol{BA})^{-1}$ to obtain

$$\boldsymbol{\Phi w}_k = \lambda_k \boldsymbol{w}_k\,, \quad k = 1, \ldots, d.$$

Then, by the definition of eigenvalue, we see that λ_k is an eigenvalue of $\boldsymbol{\Phi}$ corresponding to the eigenvector $\boldsymbol{w}_k$. Since $\boldsymbol{\Phi}$ is a diagonal matrix, its eigenvalues are its diagonal entries, so the result follows.

CHALLENGE 8.2. Using the SVD of $\widehat{\boldsymbol{U}}$, we see that

$$T^*[\boldsymbol{U}_1\,\boldsymbol{U}_2] = \boldsymbol{\Delta}\,\boldsymbol{V}^* = \left[\ \boldsymbol{\Delta}\begin{bmatrix} \boldsymbol{V}_1^* \\ \boldsymbol{V}_3^* \end{bmatrix} \quad \boldsymbol{\Delta}\begin{bmatrix} \boldsymbol{V}_2^* \\ \boldsymbol{V}_4^* \end{bmatrix}\ \right].$$

Now we compute the matrices from Problem 1:

$$\begin{aligned}\boldsymbol{BASC} &= [{\boldsymbol{\Delta}_1}^{-1}, \mathbf{0}_{d\times(m-d)}]\,\boldsymbol{T}^*\,\boldsymbol{U}_1[\boldsymbol{\Sigma}_1, \mathbf{0}_{d\times(n-d)}]\,\boldsymbol{W}^*\,\boldsymbol{W}\begin{bmatrix} {\boldsymbol{\Sigma}_1}^{-1} \\ \mathbf{0}_{(n-d)\times d}\end{bmatrix}\\ &= [{\boldsymbol{\Delta}_1}^{-1}, \mathbf{0}_{d\times(m-d)}]\boldsymbol{\Delta}\begin{bmatrix} \boldsymbol{V}_1^* \\ \boldsymbol{V}_3^* \end{bmatrix}\\ &= \boldsymbol{V}_1^*\,.\end{aligned}$$

$$\begin{aligned}\boldsymbol{BA\Phi SC} &= [{\boldsymbol{\Delta}_1}^{-1}, \mathbf{0}_{d\times(m-d)}]\,\boldsymbol{T}^*\,\boldsymbol{U}_2[\boldsymbol{\Sigma}_1, \mathbf{0}_{d\times(n-d)}]\,\boldsymbol{W}^*\,\boldsymbol{W}\begin{bmatrix} {\boldsymbol{\Sigma}_1}^{-1} \\ \mathbf{0}_{(n-d)\times d}\end{bmatrix}\\ &= [{\boldsymbol{\Delta}_1}^{-1}, \mathbf{0}_{d\times(m-d)}]\boldsymbol{\Delta}\begin{bmatrix} \boldsymbol{V}_2^* \\ \boldsymbol{V}_4^* \end{bmatrix}\\ &= \boldsymbol{V}_2^*\,.\end{aligned}$$

Thus, with this choice of $\boldsymbol{B}$ and $\boldsymbol{C}$, the eigenvalue problem of Challenge 1 reduces to $\boldsymbol{V}_2^*\boldsymbol{z}_k = \lambda_k\,\boldsymbol{V}_1^*\boldsymbol{z}_k$.

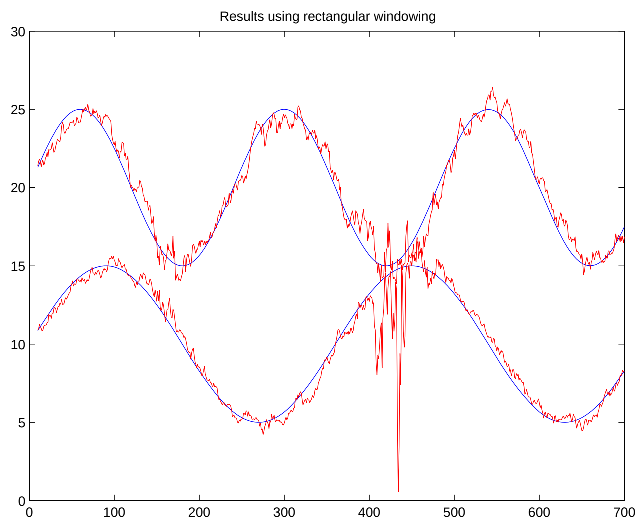


Figure 8.1. *Results of Challenge 3: the true DOA (blue) and the DOA estimated by rectangular windowing (red) as a function of time.*

CHALLENGE 8.3. The results are shown in Figure 8.1. The average error in the angle estimate is 0.62 degrees, and the average relative error is 0.046. The estimated DOAs are quite reliable except when the signals get very close.

CHALLENGE 8.4.

$$\boldsymbol{X}_{new}\boldsymbol{X}_{new}^* = \begin{bmatrix} f\boldsymbol{X} & x \end{bmatrix} \begin{bmatrix} f\boldsymbol{X}^* \\ x^* \end{bmatrix} = f^2\boldsymbol{X}\boldsymbol{X}^* + xx^* \,.$$

The matrix $\boldsymbol{X}\boldsymbol{X}^*$ has $4m^2$ entries, so multiplying it by f^2 requires $O(m^2)$ multiplications. The number of multiplications needed to form xx^* is also $O(m^2)$ multiplications.

CHALLENGE 8.5. The results are shown in Figure 8.2. The average error in the angle estimate is 0.62 degrees, and the average relative error is 0.046. The results are quite similar to those for rectangular windowing.

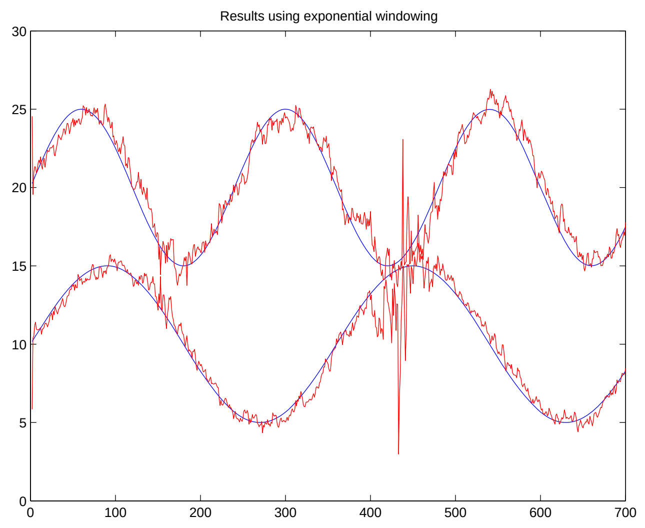


Figure 8.2. *Results of Problem 5: the true DOA (blue) and the DOA estimated by exponential windowing (red) as a function of time.*

CHALLENGE 8.6.

(a) The sum of the squares of the entries of $\boldsymbol{X}$ is the square of the *Frobenius norm* of $\boldsymbol{X}$, and this norm is invariant under multiplication by an orthogonal matrix. Therefore,

$$\|\boldsymbol{X}\|_F^2 = \|\boldsymbol{\Sigma}\|_F^2 = \sigma_1^2 + \ldots + \sigma_m^2 \, .$$

(b) The expected value of the square of each entry of $\boldsymbol{X}$ is ψ^2, so the sum of these mn values has expected value $\psi^2 mn$.

(c) The expected value is now

$$\sum_{k=1}^{m} \sum_{j=1}^{n} f^{2j} E(x_{kj}^2) = m \sum_{j=1}^{n} f^{2j} \psi^2 \to \frac{m f^2 \psi^2}{1 - f^2}$$

for large n, where E denotes expected value.

CHALLENGE 8.7. The software on the website varies κ between 2 and 6. For rectangular windowing, a window size of 4 produced fewer d-failures than window sizes of 6 or 8 at a price of increasing the average error to 0.75 degrees. As κ increased, the number of d-failures also increased, but the average error when d was correct decreased.

For exponential windowing, the fewest d-failures (8) occurred for $f = 0.7$ and $\kappa = 2$, but the average error in this case was 1.02. As κ increased, the number of d-failures increased but again the average error when d was correct decreased.

We have seen that matrix-based algorithms are powerful tools for signal processing, but they must be used in the light of statistical theory and the problem's geometry.

CHALLENGE 8.8. No answer provided.

Unit III

SOLUTIONS: Optimization and Data Fitting

Chapter 9

Solutions: Numerical Methods for Unconstrained Optimization

CHALLENGE 9.1.

$$f(\boldsymbol{x}) = x_1^4 + x_2(x_2 - 1),$$

$$\boldsymbol{g}(\boldsymbol{x}) = \begin{bmatrix} 4x_1^3 \\ 2x_2 - 1 \end{bmatrix}, \quad \boldsymbol{H}(\boldsymbol{x}) = \begin{bmatrix} 12x_1^2 & 0 \\ 0 & 2 \end{bmatrix}.$$

Step 1:

$$\boldsymbol{p} = -\begin{bmatrix} 12x_1^2 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 4x_1^3 \\ 2x_2 - 1 \end{bmatrix} = -\begin{bmatrix} 48 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 32 \\ -3 \end{bmatrix} = \begin{bmatrix} -32/48 \\ +3/2 \end{bmatrix},$$

so

$$\boldsymbol{x} \leftarrow \begin{bmatrix} 2 \\ -1 \end{bmatrix} + \begin{bmatrix} -2/3 \\ +3/2 \end{bmatrix} = \begin{bmatrix} 4/3 \\ 1/2 \end{bmatrix}.$$

CHALLENGE 9.2.

$$\boldsymbol{g}(\boldsymbol{x}) = \begin{bmatrix} e^{x_1+x_2}(1 + x_1) \\ e^{x_1+x_2}x_1 + 2x_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 4.7726 \end{bmatrix};$$

$$\boldsymbol{H}(\boldsymbol{x}) = \begin{bmatrix} e^{x_1+x_2}(2 + x_1) & e^{x_1+x_2}(1 + x_1) \\ e^{x_1+x_2}(1 + x_1) & e^{x_1+x_2}x_1 + 2 \end{bmatrix} = \begin{bmatrix} 12 & 8 \\ 8 & 6 \end{bmatrix}.$$

Now $\det(\boldsymbol{H}) = 72 - 64 = 8$, so

$$\boldsymbol{p} = -\boldsymbol{H}^{-1}\boldsymbol{g} = \frac{1}{8}\begin{bmatrix} 6 & -8 \\ -8 & 12 \end{bmatrix} \begin{bmatrix} -8 \\ -4.7726 \end{bmatrix} = \begin{bmatrix} -1.2274 \\ 0.8411 \end{bmatrix}.$$

We'd never use this inverse formula on a computer, except possibly for 2x2 matrices. Gauss elimination is generally better:

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 \\ 2/3 & 1 \end{bmatrix}, \boldsymbol{U} = \begin{bmatrix} 12 & 8 \\ 0 & 2/3 \end{bmatrix}.$$

Note that $\boldsymbol{p}^T\boldsymbol{g} = -5.8050 < 0$, so the direction is downhill.

CHALLENGE 9.3.

```
x = [1;2];
for i=1:5,
   g = [4*(x(1) - 5)^3 - x(2);
        4*(x(2) + 1)^3 - x(1)];
   H = [12*(x(1) - 5)^2, -1;
        -1,  12*(x(2) + 1)^2];
   p = -H \ g;
   x = x + p;
end
```

CHALLENGE 9.4. The Lagrangian function is

$$L(\boldsymbol{p},\lambda) = f(\boldsymbol{x}) + \boldsymbol{p}^T\boldsymbol{g} + \frac{1}{2}\boldsymbol{p}^T\boldsymbol{H}\boldsymbol{p} + \frac{\lambda}{2}(\boldsymbol{p}^T\boldsymbol{p} - h^2).$$

Setting the partial derivatives to zero yields

$$\begin{aligned} &\boldsymbol{g} + \boldsymbol{H}\boldsymbol{p} + \lambda\boldsymbol{p} = \boldsymbol{0}, \\ &\frac{1}{2}(\boldsymbol{p}^T\boldsymbol{p} - h^2) = 0. \end{aligned}$$

Thus the first equation is equivalent to $(\boldsymbol{H} + \lambda\boldsymbol{I})\boldsymbol{p} = -\boldsymbol{g}$, or $\widehat{\boldsymbol{H}}\boldsymbol{p} = -\boldsymbol{g}$, as in the Levenberg-Marquardt algorithm. The parameter λ is chosen so that $\boldsymbol{p}^T\boldsymbol{p} = h^2$.

CHALLENGE 9.5. If f is quadratic, then

$$\boldsymbol{H}^{(k)}\boldsymbol{s}^{(k)} = \boldsymbol{g}(\boldsymbol{x}^{(k+1)}) - \boldsymbol{g}(\boldsymbol{x}^{(k)}),$$

where $\boldsymbol{H}$ is the Hessian matrix of f. Close to $\boldsymbol{x}^{(k+1)}$, a quadratic model is a close fit to any function, so we demand this property to hold for our approximation to $\boldsymbol{H}$.

CHALLENGE 9.6. The formula for Broyden's good method is

$$\boldsymbol{B}^{(k+1)} = \boldsymbol{B}^{(k)} - (\boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)} - \boldsymbol{y}^{(k)})\frac{\boldsymbol{s}^{(k)T}}{\boldsymbol{s}^{(k)T}\boldsymbol{s}^{(k)}}.$$

To verify the secant condition, compute

$$\begin{aligned}\boldsymbol{B}^{(k+1)}\boldsymbol{s}^{(k)} &= \boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)} - (\boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)} - \boldsymbol{y}^{(k)})\frac{\boldsymbol{s}^{(k)T}\boldsymbol{s}^{(k)}}{\boldsymbol{s}^{(k)T}\boldsymbol{s}^{(k)}} \\ &= \boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)} - (\boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)} - \boldsymbol{y}^{(k)}) \\ &= \boldsymbol{y}^{(k)},\end{aligned}$$

as desired. If $\boldsymbol{v}^T\boldsymbol{s}^{(k)} = 0$, then

$$\begin{aligned}\boldsymbol{B}^{(k+1)}\boldsymbol{v} &= \boldsymbol{B}^{(k)}\boldsymbol{v} - (\boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)} - \boldsymbol{y}^{(k)})\frac{\boldsymbol{s}^{(k)T}\boldsymbol{v}}{\boldsymbol{s}^{(k)T}\boldsymbol{s}^{(k)}} \\ &= \boldsymbol{B}^{(k)}\boldsymbol{v},\end{aligned}$$

as desired.

CHALLENGE 9.7.

$$\begin{aligned}\boldsymbol{B}^{(k+1)}\boldsymbol{s}^{(k)} &= \boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)} - \frac{\boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)}\boldsymbol{s}^{(k)T}\boldsymbol{B}^{(k)}}{\boldsymbol{s}^{(k)T}\boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)}}\boldsymbol{s}^{(k)} + \frac{\boldsymbol{y}^{(k)}\boldsymbol{y}^{(k)T}}{\boldsymbol{y}^{(k)T}\boldsymbol{s}^{(k)}}\boldsymbol{s}^{(k)} \\ &= \boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)} - \frac{\boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)}\boldsymbol{s}^{(k)T}\boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)}}{\boldsymbol{s}^{(k)T}\boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)}} + \frac{\boldsymbol{y}^{(k)}\boldsymbol{y}^{(k)T}\boldsymbol{s}^{(k)}}{\boldsymbol{y}^{(k)T}\boldsymbol{s}^{(k)}} \\ &= \boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)} - \boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)}\frac{\boldsymbol{s}^{(k)T}\boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)}}{\boldsymbol{s}^{(k)T}\boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)}} + \boldsymbol{y}^{(k)}\frac{\boldsymbol{y}^{(k)T}\boldsymbol{s}^{(k)}}{\boldsymbol{y}^{(k)T}\boldsymbol{s}^{(k)}} \\ &= \boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)} - \boldsymbol{B}^{(k)}\boldsymbol{s}^{(k)} + \boldsymbol{y}^{(k)} \\ &= \boldsymbol{y}^{(k)}.\end{aligned}$$

CHALLENGE 9.8. Dropping superscripts for brevity, and taking advantage of symmetry of $\boldsymbol{H}$, we obtain

$$\begin{aligned}f(\boldsymbol{x}^{(0)} + \alpha\boldsymbol{p}^{(0)}) &= \frac{1}{2}(\boldsymbol{x}+\alpha\boldsymbol{p})^T\boldsymbol{H}(\boldsymbol{x}+\alpha\boldsymbol{p}) - (\boldsymbol{x}+\alpha\boldsymbol{p})^T\boldsymbol{b} \\ &= \frac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{b} + \alpha\boldsymbol{p}^T\boldsymbol{H}\boldsymbol{x} + \frac{1}{2}\alpha^2\boldsymbol{p}^T\boldsymbol{H}\boldsymbol{p} - \alpha\boldsymbol{p}^T\boldsymbol{b}.\end{aligned}$$

Differentiating with respect to α we obtain

$$\boldsymbol{p}^T\boldsymbol{H}\boldsymbol{x} + \alpha\boldsymbol{p}^T\boldsymbol{H}\boldsymbol{p} - \boldsymbol{p}^T\boldsymbol{b} = 0,$$

so

$$\alpha = \frac{\boldsymbol{p}^T\boldsymbol{b} - \boldsymbol{p}^T\boldsymbol{H}\boldsymbol{x}}{\boldsymbol{p}^T\boldsymbol{H}\boldsymbol{p}} = \frac{\boldsymbol{p}^T\boldsymbol{r}}{\boldsymbol{p}^T\boldsymbol{H}\boldsymbol{p}},$$

where $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}$.

If we differentiate a second time, we find that the second derivative of f with respect to α is $\boldsymbol{p}^T\boldsymbol{H}\boldsymbol{p} > 0$ (when $\boldsymbol{p} \neq \boldsymbol{0}$), so we have found a minimizer.

CHALLENGE 9.9.

```
function Hv = Htimes(x,v,h)

% Input:
%     x  is the current evaluation point.
%     v  is the direction of change in x.
%     h  is the stepsize for the change,
%        a small positive parameter (e.g., h = 0.001).
% We use a function [f,g] = myfnct(x),
% which returns the function value f(x) and the gradient g(x).
%
% Output:
%     Hv is a finite difference approximation to H*v,
%         where H is the Hessian matrix of myfnct.
%
% DPO

[f, g ] = myfnct(x);
[fp,gp] = myfnct(x + h * v);
Hv = (gp - g)/h;
```

CHALLENGE 9.10. Here is one way to make the decision:

- If the function is not differentiable, use Nelder-Meade.
- If 2nd derivatives (Hessians) are cheaply available and there is enough storage for them, use Newton.
- Otherwise, use quasi-Newton (with a finite-difference 1st derivative if necessary).

CHALLENGE 9.11.

- Newton: often converges with a quadratic rate when started close enough to a solution, but requires both first and second derivatives (or good approximations of them) as well as storage and solution of a linear system with a matrix of size 2000×2000.

- Quasi-Newton: often converges superlinearly when started close enough to a solution, but requires first derivatives (or good approximations of them) and storage of a matrix of size 2000×2000, unless the matrix is accumulated implicitly by saving the vectors $\boldsymbol{s}$ and $\boldsymbol{y}$.
- Pattern search: converges only linearly, but has good global behavior and requires only function values, no derivatives.

If first derivatives (or approximations) were available, I would use quasi-Newton, with updating of the matrix decomposition (or a limited memory version). Otherwise, I would use pattern search.

CHALLENGE 9.12. I would use pattern search to minimize $F(\boldsymbol{x}) = -y(1)$ as a function of $\boldsymbol{x}$. When a function value $F(\boldsymbol{x})$ is needed, I would call one of MATLAB's stiff ODE solvers, since I don't know whether the problem is stiff or not, and return the value computed as $y(1)$. The value of $\boldsymbol{x}$ would need to be passed to the function that evaluates f for the ode solver.

I chose pattern search because it has proven convergence and does not require derivatives of F with respect to $\boldsymbol{x}$. Note that these derivatives are not available for this problem: we can compute derivatives of y with respect to t but not with respect to $\boldsymbol{x}$. And since our value of $y(1)$ is only an approximation, the use of finite differences to estimate derivatives with respect to $\boldsymbol{x}$ would yield values too noisy to be useful.

CHALLENGE 9.13.

Method	conv. rate	Storage	f evals/itn	$\boldsymbol{g}$ evals/itn	$\boldsymbol{H}$ evals/itn
Truncated Newton	> 1	$O(n)$	0	$\leq n+1$	0
Newton	2	$O(n^2)$	0^1	1	1
Quasi-Newton	$> 1^2$	$O(n^2)$	0^1	1	0
steepest descent	1	$O(n)$	0^1	1	0
Conjugate gradients	1	$O(n)$	0^1	1	0

Notes on the table:

1. Once the counts for the linesearch are omitted, no function evaluations are needed.

2. For a single step, Quasi-Newton is superlinear; it is n-step quadratic.

Chapter 10

Solutions: Numerical Methods for Constrained Optimization

CHALLENGE 10.1. The graphical solutions are left to the reader.

(a) The optimality condition is that the gradient should be zero. We calculate

$$\boldsymbol{g}(\boldsymbol{x}) = \begin{bmatrix} 2x_1 - x_2 + 5 \\ 8x_2 - x_1 + 3 \end{bmatrix},$$

$$\boldsymbol{H}(\boldsymbol{x}) = \begin{bmatrix} 2 & -1 \\ -1 & 8 \end{bmatrix}.$$

Since $\boldsymbol{H}$ is positive definite (Gerschgorin theorem), $f(\boldsymbol{x})$ has a unique minimizer satisfying $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$, so

$$\boldsymbol{H}(\boldsymbol{x})\,\boldsymbol{x} = \begin{bmatrix} -5 \\ -3 \end{bmatrix},$$

and therefore $\boldsymbol{x} = [-2.8667, -0.7333]^T$.

(b) Using a Lagrange multiplier we obtain

$$L(\boldsymbol{x}, \lambda) = x_1^2 + 4x_2^2 - x_1x_2 + 5x_1 + 3x_2 + 6 - \lambda(x_1 + x_2 - 2).$$

Differentiating gives the optimality conditions

$$\begin{aligned} 2x_1 - x_2 + 5 - \lambda &= 0, \\ 8x_2 - x_1 + 3 - \lambda &= 0, \\ x_1 + x_2 - 2 &= 0. \end{aligned}$$

Solving this linear system of equations yields $\boldsymbol{x} = [1.3333, 0.6667]^T$, $\lambda = 7$.

(c) In this case, $\boldsymbol{A}^T = \boldsymbol{I}$, so the conditions are

$$\begin{aligned} \boldsymbol{\lambda} &= \begin{bmatrix} 2x_1 - x_2 + 5 \\ 8x_2 - x_1 + 3 \end{bmatrix}, \\ \boldsymbol{\lambda} &\geq \boldsymbol{0}, \\ \boldsymbol{x} &\geq \boldsymbol{0}, \\ \lambda_1 x_1 + \lambda_2 x_2 &= 0. \end{aligned}$$

The solution is $\boldsymbol{x} = \mathbf{0}$, $\boldsymbol{\lambda} = [5,3]^T$.

(d) Remembering to convert the first constraint to $\geq$ form, we get

$$\begin{aligned} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -2x_1 & -2x_2 \end{bmatrix}^T \boldsymbol{\lambda} &= \begin{bmatrix} 2x_1 - x_2 + 5 \\ 8x_2 - x_1 + 3 \end{bmatrix}, \\ \boldsymbol{\lambda} &\geq \mathbf{0}, \\ \boldsymbol{x} &\geq \mathbf{0}, \\ 1 - x_1^2 - x_2^2 &\geq 0, \\ \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3(1 - x_1^2 - x_2^2) &= 0. \end{aligned}$$

The solution is $\boldsymbol{x} = \mathbf{0}$, $\boldsymbol{\lambda} = [5,3,0]^T$.

CHALLENGE 10.2. (Partial Solution) We need to verify that $\boldsymbol{Z}^T \nabla_{xx} L(x,\lambda)\boldsymbol{Z}$ is positive semidefinite.

(a)

$$\boldsymbol{Z}^T \nabla_{xx} L(x,\lambda)\boldsymbol{Z} = \boldsymbol{H}(\boldsymbol{x}) = \begin{bmatrix} 2 & -1 \\ -1 & 8 \end{bmatrix}.$$

(b)

$$\nabla_{xx} L(x,\lambda) = \begin{bmatrix} 2 & -1 \\ -1 & 8 \end{bmatrix},$$

and we can take $\boldsymbol{Z}^T = [1, -1]$.

(c)

$$\nabla_{xx} L(x,\lambda) = \begin{bmatrix} 2 & -1 \\ -1 & 8 \end{bmatrix}.$$

Both constraints are active at the optimal solution, so $\boldsymbol{Z}$ is the empty matrix and the optimality condition is satisfied trivially.

(d) The $\boldsymbol{x} \geq \mathbf{0}$ constraints are active, so the solution is as in part (c).

CHALLENGE 10.3. The vector $[6,1]^T$ is a particular solution to $x_1 - 2x_2 = 4$, and the vector $[2,1]^T$ is a basis for the nullspace of the matrix $\boldsymbol{A} = [1, -2]$. (These choices are not unique, so there are many correct answers.) Using our choices, any solution to the equality constraint can be expressed as

$$\boldsymbol{x} = \begin{bmatrix} 6 \\ 1 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix} v = \begin{bmatrix} 6 + 2v \\ 1 + v \end{bmatrix}.$$

Therefore, our problem is equivalent to

$$\min_v 5(6+2v)^4 + (6+2v)(1+v) + 6(1+v)^2$$

subject to

$$\begin{aligned} 6+2v &\geq 0, \\ 1+v &\geq 0. \end{aligned}$$

Using a log barrier function for these constraints, we obtain the unconstrained problem

$$\min_v B_\mu(v)$$

where

$$B_\mu(v) = 5(6+2v)^4 + (6+2v)(1+v) + 6(1+v)^2 - \mu\log(6+2v) - \mu\log(1+v).$$

Notice that if $1+v \geq 0$, then $6+2v \geq 0$. Therefore, the first log term can be dropped from $B_\mu(v)$.

CHALLENGE 10.4.

(a) The central path is defined by

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x} &= \boldsymbol{b}, \\ \boldsymbol{A}^*\boldsymbol{w} + \boldsymbol{s} &= \boldsymbol{c}, \\ \mu F'(\boldsymbol{x}) + \boldsymbol{s} &= \boldsymbol{0}, \end{aligned}$$

where $F'(\boldsymbol{x}) = \boldsymbol{X}^{-1}\boldsymbol{e}$, $\boldsymbol{w}$ is an $m \times 1$ vector, $\boldsymbol{e}$ is the column vector of ones, and $\boldsymbol{X}$ is diag($\boldsymbol{x}$).

(b)The central path is defined by

$$\begin{aligned} \text{trace}(\boldsymbol{a}_i\boldsymbol{x}) &= b_i. \quad i = 1, \ldots, m, \\ \sum_{i=1}^m w_i\boldsymbol{a}_i + \boldsymbol{s} &= \boldsymbol{c}, \\ \mu F'(\boldsymbol{x}) + \boldsymbol{s} &= \boldsymbol{0}, \end{aligned}$$

where $\boldsymbol{w}$ is an $m \times 1$ vector and $\boldsymbol{x}$ and $\boldsymbol{s}$ are symmetric $n \times n$ matrices. Since $F(\boldsymbol{x}) = \log(\det(\boldsymbol{x}))$, we can compute $F'(\boldsymbol{x}) = -\boldsymbol{x}^{-1}$.

CHALLENGE 10.5.

(a) K^* is the set of vectors that form nonnegative inner products with every vector that has nonnegative entries. Therefore, $K = K^* = \{\boldsymbol{x} : \boldsymbol{x} \geq \boldsymbol{0}\}$, the positive orthant in $\mathcal{R}^n$.

(b) The dual problem is

$$\max_{\boldsymbol{w}} \boldsymbol{w}^T \boldsymbol{b}$$

subject to $\boldsymbol{A}^T\boldsymbol{w} + \boldsymbol{s} = \boldsymbol{c}$ and $\boldsymbol{s} \geq \boldsymbol{0}$.

(c) Since $\boldsymbol{s} \geq \boldsymbol{0}$, the constraint $\boldsymbol{A}^T\boldsymbol{w} + \boldsymbol{s} = \boldsymbol{c}$ means that each component of $\boldsymbol{A}^T\boldsymbol{w}$ is less than or equal to each component of $\boldsymbol{c}$, since we need to add $\boldsymbol{s}$ on in order to get equality. Therefore, $\boldsymbol{A}^T\boldsymbol{w} \leq \boldsymbol{c}$.

(d)

First order optimality conditions: The constraints are

$$\begin{aligned} \boldsymbol{x} &\geq \boldsymbol{0}, \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} &= \boldsymbol{0}. \end{aligned}$$

The derivative matrix for the constraints becomes

$$\begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{A} \end{bmatrix}.$$

Using a bit of foresight, let's call the Lagrange multipliers for the inequality constraints $\boldsymbol{s}$ and those for the equality constraints $\boldsymbol{\lambda}$. Then the optimality conditions are

$$\begin{aligned} \boldsymbol{I}\boldsymbol{s} + \boldsymbol{A}^T\boldsymbol{\lambda} &= \boldsymbol{c}, \\ \boldsymbol{s} &\geq \boldsymbol{0}, \\ \boldsymbol{x} &\geq \boldsymbol{0}, \\ \boldsymbol{A}\boldsymbol{x} &= \boldsymbol{b}, \\ \boldsymbol{s}^T\boldsymbol{x} &= 0. \end{aligned}$$

The central path:

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x} &= \boldsymbol{b}, \\ \boldsymbol{A}^*\boldsymbol{w} + \boldsymbol{s} &= \boldsymbol{c}, \\ \mu\boldsymbol{X}^{-1}\boldsymbol{e} + \boldsymbol{s} &= \boldsymbol{0}, \end{aligned}$$

where $\boldsymbol{s}, \boldsymbol{x} \geq 0$ (since the feasible cone is the positive orthant).

Both sets of conditions have $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. Setting $\boldsymbol{\lambda} = \boldsymbol{w}$ shows the equivalence of $\boldsymbol{s} + \boldsymbol{A}^T\boldsymbol{\lambda} = \boldsymbol{c}$ and $\boldsymbol{A}^*\boldsymbol{w} + \boldsymbol{s} = \boldsymbol{c}$, since $\boldsymbol{A}^* = \boldsymbol{A}^T$. Multiplying $\mu\boldsymbol{X}^{-1}\boldsymbol{e} + \boldsymbol{s} = \boldsymbol{0}$ by $\boldsymbol{X}$, we obtain $\mu\boldsymbol{e} + \boldsymbol{X}\boldsymbol{s} = \boldsymbol{0}$, which is equivalent to $\boldsymbol{s}^T\boldsymbol{x} = 0$ when $\mu = 0$, since $\boldsymbol{x}$ and $\boldsymbol{s}$ are nonnegative. Therefore the conditions are equivalent.

CHALLENGE 10.6. (Partial Solution) We minimize by making both factors large in absolute value and with the same sign. The largest absolute values occur at the corners, and the minimizers occur when $x_1 = x_2 = 1$ and when $x_1 = x_2 = 0$. In both cases, $f(\boldsymbol{x}) = -1/4$. By changing one of the 1/2 values to 1/3 (for example), one of these becomes a local minimizer.

Chapter 11

Solutions: Case Study: Classified Information: The Data Clustering Problem

(coauthored by Nargess Memarsadeghi)

CHALLENGE 11.1. The original image takes mpb bits, while the clustered image takes kb bits to store the cluster centers and $mp\lceil\log_2 k\rceil$ bits to store the cluster indices for all mp pixels. For jpeg images with RGB (red, green, blue) values ranging between 0 and 255, we need 8 bits for each of the $q = 3$ values (red, green, and blue). Therefore, an RGB image with 250,000 pixels takes $24 * 250,000 = 6,000,000$ bits, while the clustered image takes about $250,000 \log_2 4 = 500,000$ bits if we have 3 or 4 clusters and 250,000 bits if we have 2 clusters. These numbers can be further reduced by compression techniques such as run-length encoding.

CHALLENGE 11.2.

(a) Neither D nor R is convex everywhere. Figure 11.2 plots these functions for a particular choice of points as one of the cluster centers is moved. We fix the data points at 0 and 1 and one of the centers at 1.2, and plot D and R as a function of the second center c. For $c < -1.2$ and $c > 1.2$, the function D is constant, since the second cluster is empty, while for $-1.2 < c < 1.2$, the function is quadratic. Since each function is above some of its secants (the line connecting two points on the graph), each function fails to be convex.

(b) Neither D nor R is differentiable everywhere. Again see Figure 11.1. The function D fails to be differentiable at $c = -1.2$ and $c = 1.2$. Trouble occurs at the points where a data value moves from one cluster to another.

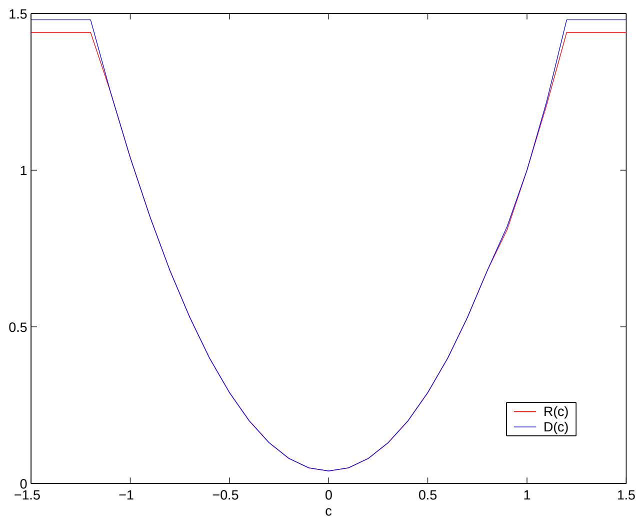


Figure 11.1. *The functions R and D for a particular dataset.*

(c) We want to minimize the function

$$D(\boldsymbol{c}) = \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{c}\|^2$$

over all choices of $\boldsymbol{c}$. Since there is only one center $\boldsymbol{c}$, this function is convex and differentiable everywhere, and the solution must be a zero of the gradient. Differentiating with respect to $\boldsymbol{c}$ we obtain

$$\sum_{i=1}^{n} 2(\boldsymbol{x}_i - \boldsymbol{c}) = \mathbf{0},$$

so

$$\boldsymbol{c} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i.$$

It is easy to verify that this is a minimizer, not a maximizer or a stationary point, so the solution is to choose $\boldsymbol{c}$ to be the centroid (mean value) of the data points.

(d) As one data point moves from the others, eventually a center will "follow" it, giving it its own cluster. So the clustering algorithm will fit $k - 1$ clusters to the remaining $n - 1$ data points.

CHALLENGE 11.3. A sample program is given on the website. Note that when using a general purpose optimization function, it is important to match the

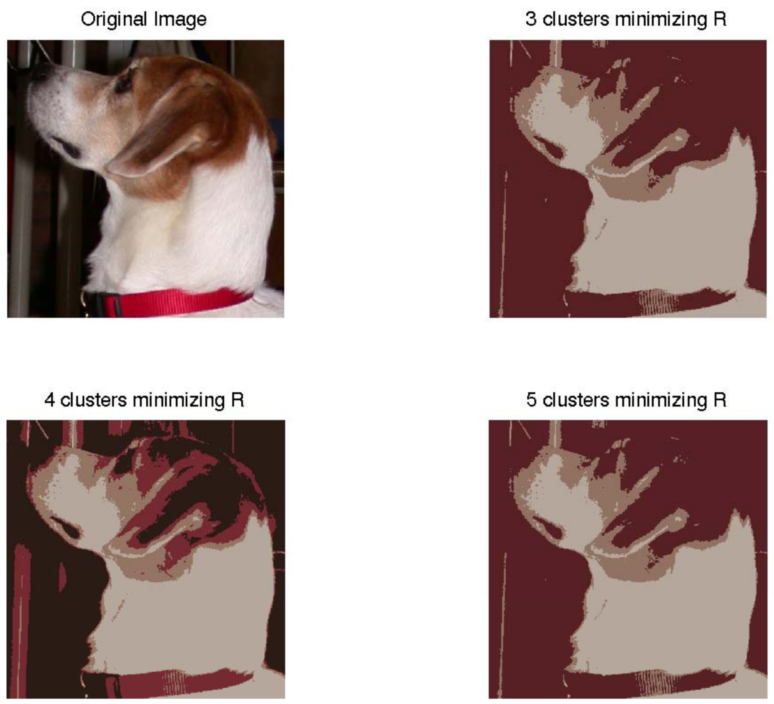


Figure 11.2. *The images resulting from minimizing R.*

termination criteria to the scaling of the variables and the function to be minimized; otherwise the function might terminate prematurely (even with negative entries in the cluster centers if, for example, the objective function values are all quite close together) or never terminate (if small changes in the variables lead to large changes in the function). We chose to scale R and D by dividing by 256 and by the number of points in the summation.

The solution is *very* sensitive to the initial guess, since there are many local minimizers.

(a) The number of variables is kq.

(b) Although the number of variables is quite small (9 for $k = 3$ and 15 for $k = 5$), evaluating the function R or D is quite expensive, since it involves a mapping each of the 1,000 pixels to a cluster. Therefore, the overhead of the algorithm is insignificant and the time is proportional to the number of function evaluations. The functions are not differentiable, so modeling as a quadratic function is not so effective. This slows the convergence rate, although only 15-25 iterations are used. This was enough to converge when minimizing D, but not enough for R to converge.

Actually, a major part of the time in the sample implementation is postprocessing: the construction of the resulting image!

(c) Figures 11.2 and 11.3 show the results with $k = 3, 4, 5$ clusters. The solution

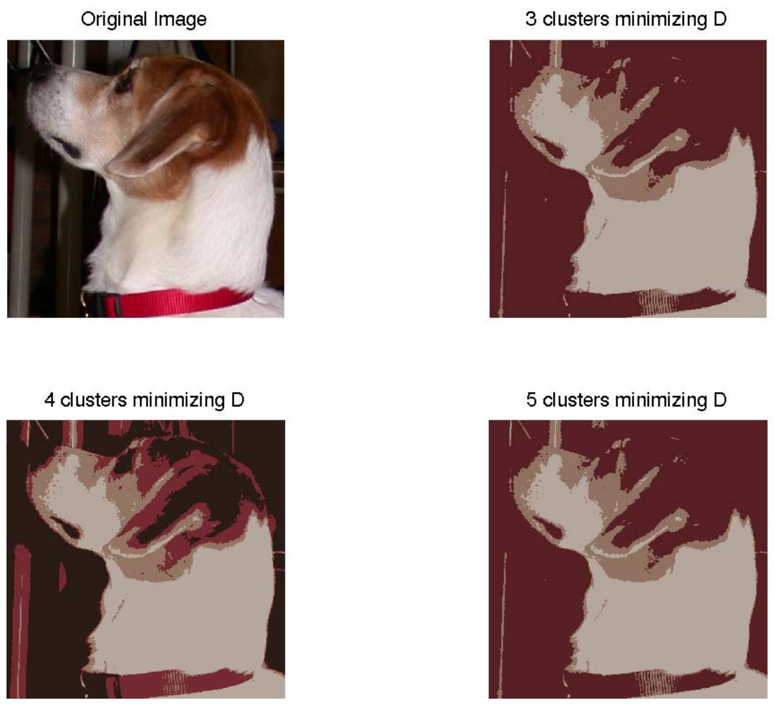


Figure 11.3. *The images resulting from minimizing D.*

is very dependent on the initial guess, but the rather unlikely choice that we made (all zeros in the green coordinate) gave some of the best results.

Our first evaluation criterion should be how the image looks, sometimes called the "eyeball norm". In the results for minimizing D, it is harder to differentiate the dog from the background. For minimizing R with $k = 3$, his white fur is rendered as green and the image quality is much worse than for minimizing D or using k-means. For $k = 4$ or $k = 5$, though, minimizing R yields a good reconstruction, with good shading in the fur and good rendering of the table legs in the background, and the results look better than those for minimizing D. (Note that the table legs were not part of the sample that determined the cluster centers.)

We can measure the quantitative change in the images, too. Each pixel value x_i in the original or the clustered image is a vector with q dimensions, and we can measure the relative change in the image as

$$\left(\frac{\sum_{i=1}^{n} ||x_i^{original} - x_i^{clustered}||^2}{\sum_{i=1}^{n} ||x_i^{original}||^2} \right)^{1/2} .$$

This measure is usually smaller when minimizing R rather than D: .363 vs .271 for $k = 3$, .190 vs .212 for $k = 4$, and .161 vs .271 for $k = 5$. The optimization program sometimes stops with negative coordinates for a cluster center or no points in the

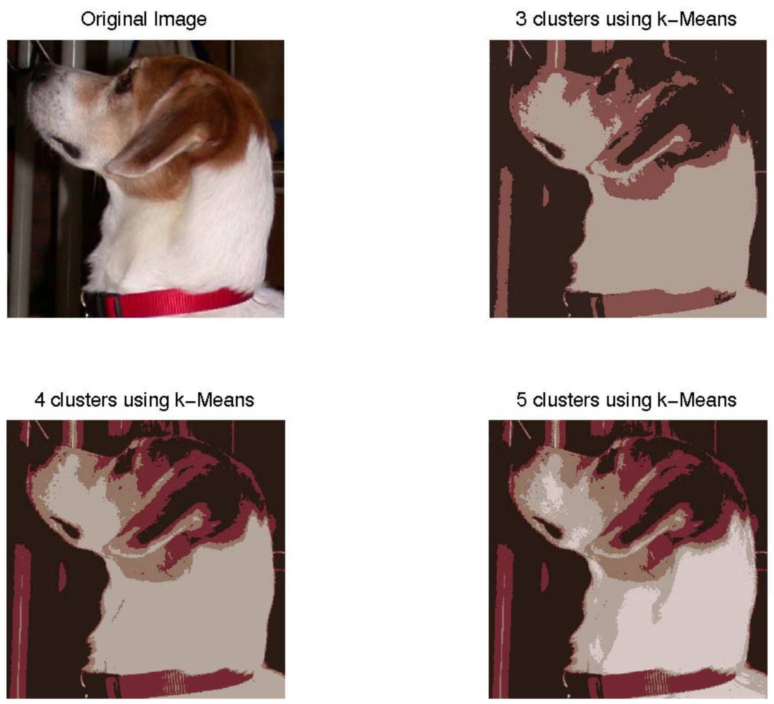


Figure 11.4. *The images resulting from k-means.*

cluster; for example, for $k = 5$, minimizing D produced only 3 nonempty clusters, and for $k = 2$, minimizing R produced only 2 nonempty clusters.

(d) If $q < 4$, then k might be chosen by plotting the data. For larger values of q, we might try increasing values of k, stopping when the cluster radii fall below the noise level in the data or when the cluster radii stay relatively constant.

Only one choice of data values appears in the sample program, but we can easily modify the program to see how sensitive the solution is to the choice of data.

CHALLENGE 11.4. A sample program is given on the website and results are shown in Figure 11.4. This k-Means function is much faster than the algorithm for Challenge 3. The best results for $k = 3$ and $k = 5$ are those from k-means, but the $k = 4$ result from minimizing R seems somewhat better than the other $k = 4$ results. The quantitative measures are mixed: the 2-norm of the relative change is .247, .212, and .153 for $k = 3, 4, 5$ respectively, although the algorithm was not run to convergence.

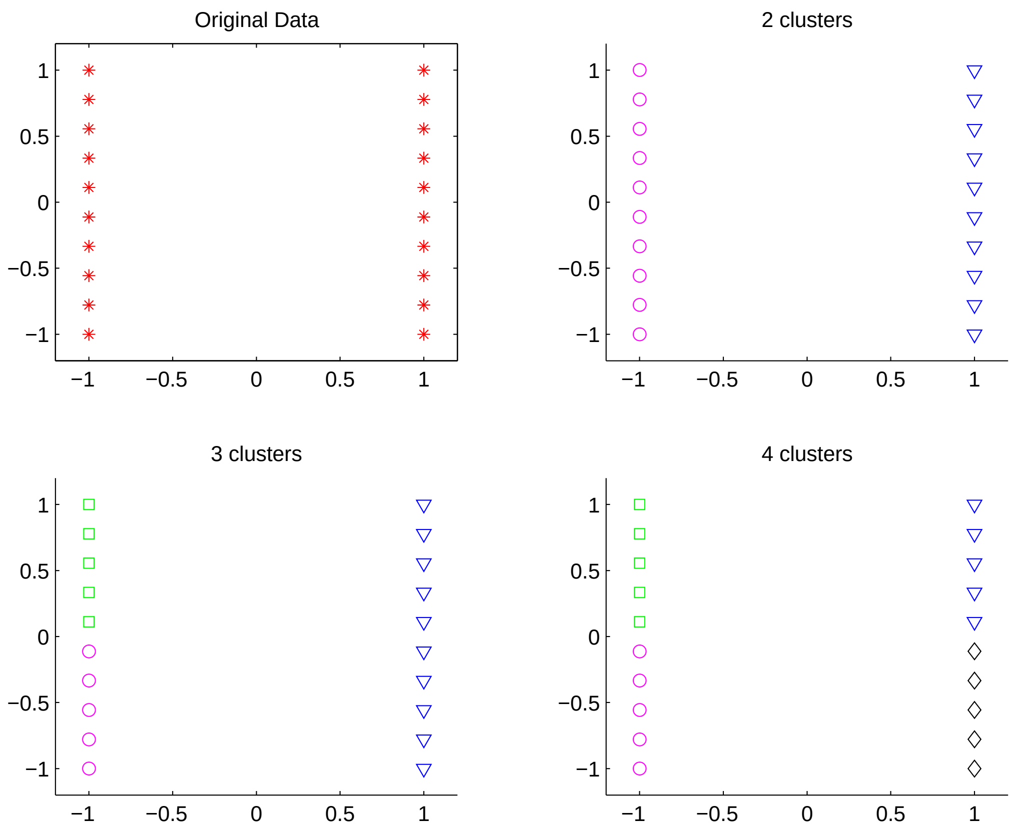


Figure 11.5. *Clustering the data for Challenge 5 using the first initialization of centers.*

CHALLENGE 11.5. The website contains a sample program, and Figures 11.5 and 11.6 display the results. Each datapoint is displayed with a color and symbol that represent its cluster. An intuitive clustering of this data is to make two vertical clusters, as determined by the algorithm with the first initialization and $k = 2$. Note, however, that the distance between the top and bottom data points in each cluster is the same as the distance between the clusters (measured by the minimum distance between points in different clusters)! The two clusters determined by the second initialization have somewhat greater radii, but are not much worse. What is worse about them, though, is that there is less distance between clusters.

If we choose to make too many clusters ($k > 2$), we add artificial distinctions between data points.

CHALLENGE 11.6. The website contains a sample program, and Figures 11.7

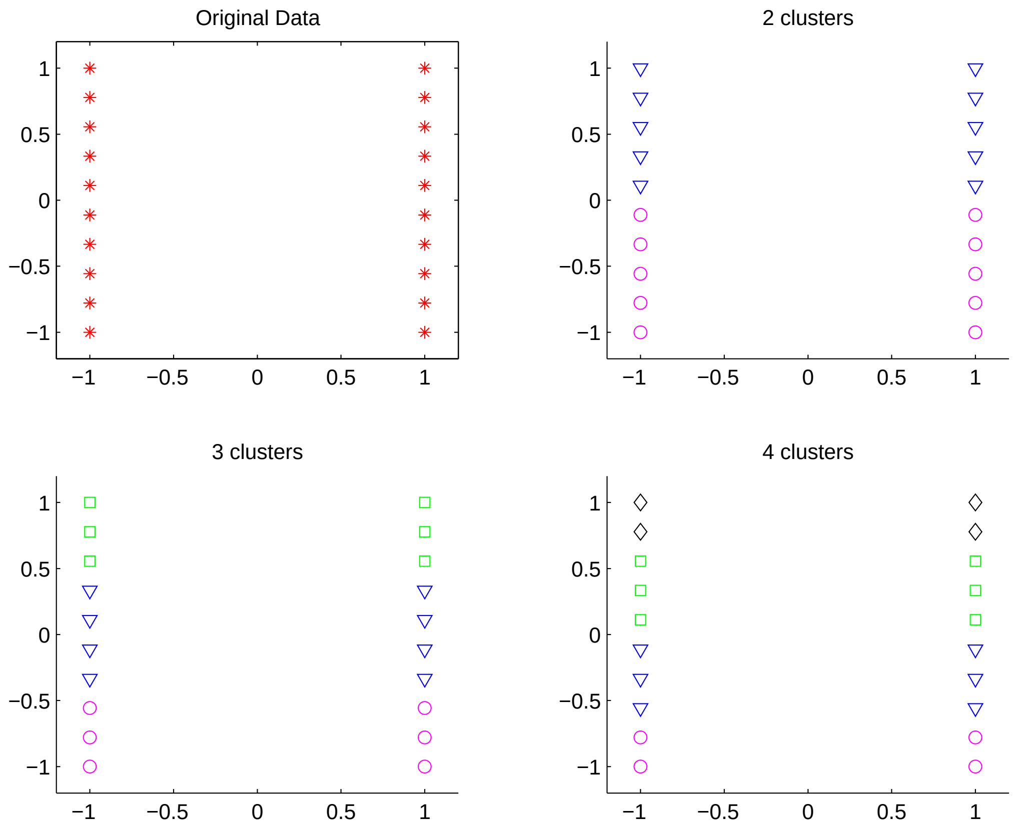


Figure 11.6. *Clustering the data for Challenge 5 using the second initialization of centers.*

and 11.8 display the results.

Coordinate scaling definitely changes the merits of the resulting clusters. The clusters produced by the second initialization have much smaller radii. Nonlinear scalings of the data also affect clustering; for example, the results of clustering the pixels in the Charlie image could be very different if we represented the image in coordinates other than RGB.

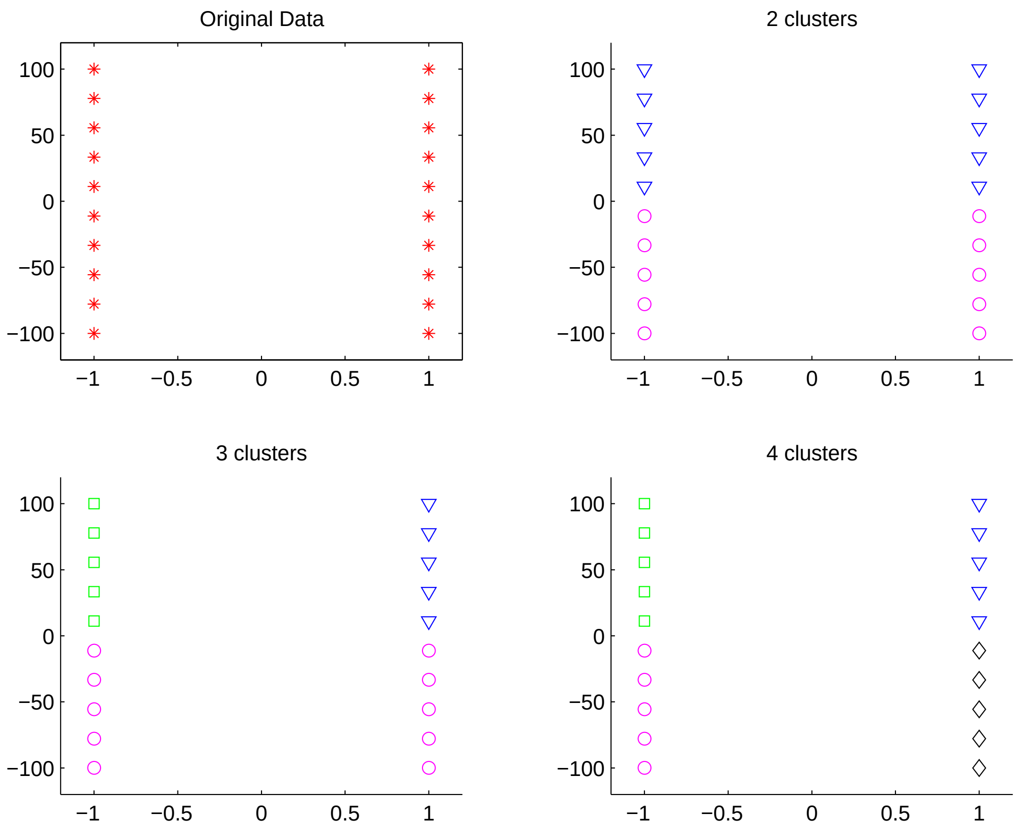


Figure 11.7. *Clustering the data for Challenge 6 using the first initialization of centers. Note that the vertical scale is far different from the horizontal.*

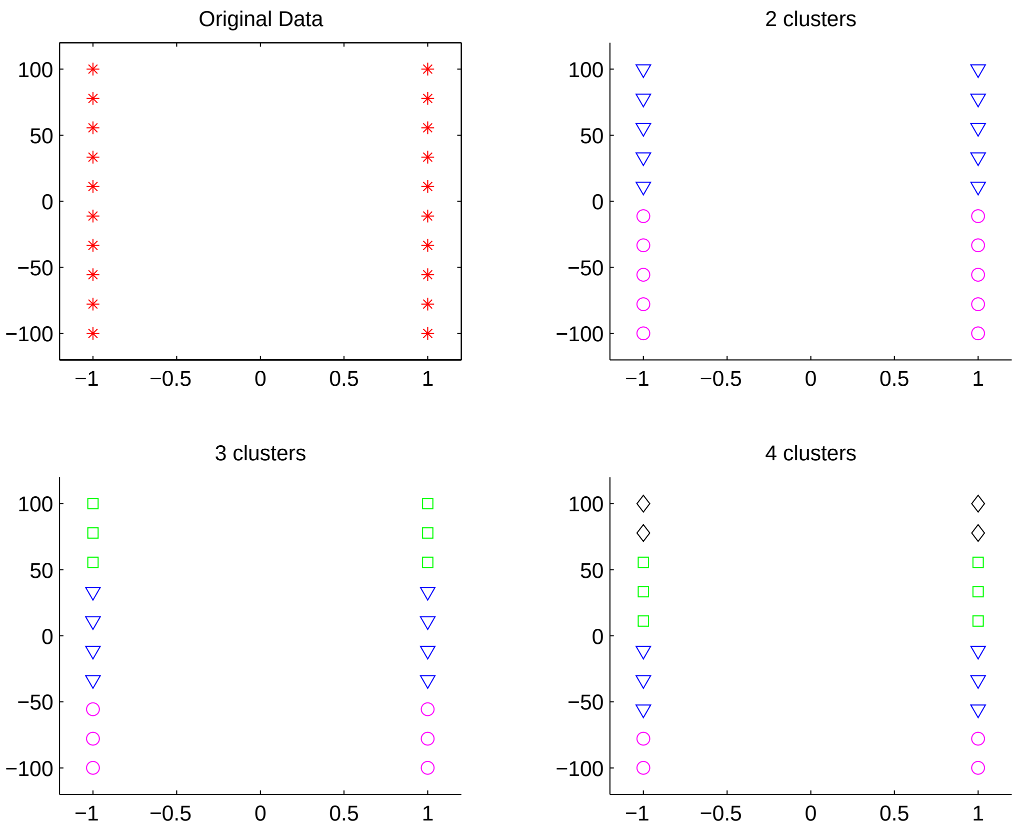


Figure 11.8. *Clustering the data for Challenge 6 using the second initialization of centers. Note that the vertical scale is far different from the horizontal.*

Chapter 12

Solutions: Case Study: Achieving a Common Viewpoint: Yaw, Pitch, and Roll

(coauthored by David A. Schug)

CHALLENGE 12.1.

(a) First we rotate the object by an angle ϕ in the xy-plane. Then we rotate by an angle $-\theta$ in the new xz-plane, and finish with a rotation of ψ in the resulting yz-plane.

(b) We will use the QR decomposition of a matrix; any nonsingular matrix can be expressed as the product of an orthogonal matrix times an upper triangular one. One way to compute this is to use plane rotations to reduce elements below the diagonal of our matrix to zero. Let's apply this to the matrix $\boldsymbol{Q}^T$. Then by choosing ϕ appropriately, we can make $\boldsymbol{Q}_{yaw}\boldsymbol{Q}^T$ have a zero in row 2, column 1. Similarly, by choosing θ, we can force a zero in row 3, column 1 of $\boldsymbol{Q}_{pitch}\boldsymbol{Q}_{yaw}\boldsymbol{Q}^T$ (without ruining our zero in row 2, column 1). (Note that since we require $-\pi/2 < \theta < \pi/2$, if $\cos\theta$ turns out to be negative, we need to change the signs on $\cos\theta$ and $\sin\theta$ to compensate.) Finally, we can choose ψ to force $\boldsymbol{Q}_{roll}\boldsymbol{Q}_{pitch}\boldsymbol{Q}_{yaw}\boldsymbol{Q}^T$ to be upper triangular. Since the product of orthogonal matrices is orthogonal, and the only upper triangular orthogonal matrices are diagonal, we conclude that $\boldsymbol{Q}_{roll}\boldsymbol{Q}_{pitch}\boldsymbol{Q}_{yaw}$ is a diagonal matrix (with entries ± 1) times $(\boldsymbol{Q}^T)^{-1}$. Now convince yourself that the angles can be chosen so that the diagonal matrix is the identity. This method for proving this property is particularly nice because it leads to a fast algorithm that we can use in Challenge 4 to recover the Euler angles given an orthogonal matrix $\boldsymbol{Q}$.

CHALLENGE 12.2. A sample MATLAB program to solve this problem is available on the website. The results are shown in Figure 12.1. In most cases, 2-4

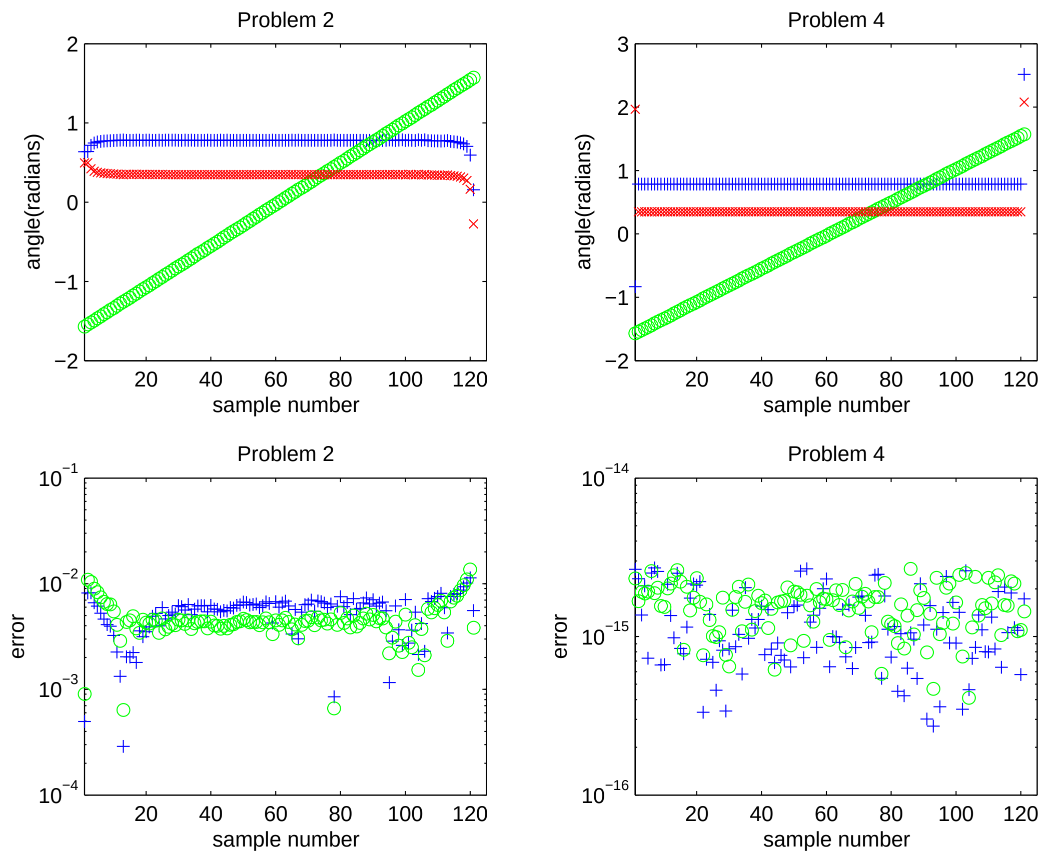


Figure 12.1. *Results of Challenge 2 (left) and Challenge 4 (right). The top graphs show the computed yaw (blue plusses), pitch (green circles), and roll (red x's), and the bottom graphs show the error in* **Q** *(blue plusses) and the error in the rotated positions (green circles).*

digit accuracy is achieved for the angles and positions, but trouble is observed when the pitch is close to vertical ($\pm\pi/2$).

CHALLENGE 12.3.

(a) Suppose that $\boldsymbol{C}$ is $m \times n$. The first fact follows from

$$\text{trace}(\boldsymbol{C}^T\boldsymbol{C}) = \sum_{k=1}^{n}(\sum_{i=1}^{m} c_{ik}^2) = \sum_{k=1}^{n}\sum_{i=1}^{m} c_{ik}^2 = \|\boldsymbol{C}\|_F^2.$$

To prove the second fact, note that

$$\text{trace}(\boldsymbol{CD}) = \sum_{k=1}^{m}(\sum_{i=1}^{n}(c_{ki}d_{ik})),$$

while

$$\text{trace}(\boldsymbol{DC}) = \sum_{i=1}^{n}(\sum_{k=1}^{m}(d_{ik}c_{ki})),$$

which is the same.

(b) Note that

$$\|\boldsymbol{B}-\boldsymbol{QA}\|_F^2 = \text{trace}((\boldsymbol{B}-\boldsymbol{QA})^T(\boldsymbol{B}-\boldsymbol{QA})) = \text{trace}(\boldsymbol{B}^T\boldsymbol{B}+\boldsymbol{A}^T\boldsymbol{A})-2\,\text{trace}(\boldsymbol{A}^T\boldsymbol{Q}^T\boldsymbol{B}),$$

so we can minimize the left-hand side by maximizing trace($\boldsymbol{A}^T\boldsymbol{Q}^T\boldsymbol{B}$).

(c) We compute

$$\begin{aligned}\text{trace}(\boldsymbol{Q}^T\boldsymbol{BA}^T) = \text{trace}(\boldsymbol{Q}^T\boldsymbol{U}\Sigma\boldsymbol{V}^T) = \text{trace}(\boldsymbol{V}^T\boldsymbol{Q}^T\boldsymbol{U}\Sigma) = \text{trace}(\boldsymbol{Z}\Sigma)\\ = \sum_{i=1}^{m}\sigma_i z_{ii} \le \sum_{i=1}^{m}\sigma_i, \qquad (12.1)\end{aligned}$$

where the inequality follows from the fact that elements of an orthogonal matrix lie between −1 and 1.

(d) Since $\boldsymbol{Z} = \boldsymbol{V}^T\boldsymbol{Q}^T\boldsymbol{U}$, we have $\boldsymbol{Z} = \boldsymbol{I}$ if $\boldsymbol{Q} = \boldsymbol{UV}^T$.

CHALLENGE 12.4. The results are shown in Figure 12.1. The computed results are *much* better than those of Challenge 2, with errors at most 10^{-14} and no trouble when the pitch is close to vertical.

CHALLENGE 12.5. We compute

$$\|\boldsymbol{B}-\boldsymbol{QA}-\boldsymbol{te}^T\|_F^2 = \sum_{i=1}^{m}\sum_{j=1}^{n}(\boldsymbol{B}-\boldsymbol{QA})_{ij}^2 - 2t_i(\boldsymbol{B}-\boldsymbol{QA})_{ij} + nt_i^2,$$

and setting the partial derivative with respect to t_i to zero yields

$$t_i = \frac{1}{n}\sum_{j=1}^{n}(\boldsymbol{B}-\boldsymbol{QA})_{ij}.$$

Therefore,

$$\begin{aligned} \boldsymbol{t} &= \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{b}_j - \frac{1}{n}\boldsymbol{Q}\boldsymbol{a}_j \\ &= \boldsymbol{c}_B - \boldsymbol{Q}\boldsymbol{c}_A. \end{aligned}$$

This very nice observation was made by Hanson and Norris [1].

CHALLENGE 12.6. The results are shown in Figure 12.2. With no perturbation, the errors in the angles, the error in the matrix $\boldsymbol{Q}$, and the RMSD are all less than 10^{-15}. With perturbation in each element uniformly distributed between -10^{-3} and 10^{-3}, the errors rise to about 10^{-4}.

Comparison of the SVD method with other methods can be found in [2] and [3], although none of these authors knew that the method was due to Hanson and Norris.

CHALLENGE 12.7.

(a) Yes. Since in this case the rank of matrix $\boldsymbol{A}$ is 1, we have two singular values $\sigma_2 = \sigma_3 = 0$. Therefore we only need $z_{11} = 1$ in equation (12.1) and we don't care about the values of z_{22} or z_{33}.

(b) Degenerate cases result from unfortunate choices of the points in $\boldsymbol{A}$ and $\boldsymbol{B}$. If all of the points in $\boldsymbol{A}$ lie on a line or a plane, then there exist multiple solution matrices $\boldsymbol{Q}$. Additionally, if two singular values of the matrix $\boldsymbol{B}^T\boldsymbol{A}$ are nonzero but equal, then small perturbations in the data can create large changes in the matrix $\boldsymbol{Q}$. See [1].

(c) A degenerate case and a case of gymbal lock are illustrated on the website.

[1] Richard J. Hanson and Michael J. Norris, "Analysis of measurements based on the singular value decomposition," *SIAM J. Scientific and Statistical Computing*, 2(3):363-373, 1981.

[2] Kenichi Kanatani, "Analysis of 3-d rotation fitting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):543-549, May 1994.

[3] D.W. Eggert and A. Lorusso and R.B. Fisher, "Estimating 3-d rigid body transformations: a comparison of four major algorithms," *Machine Learning and Applications*, 9:272-290, 1997.

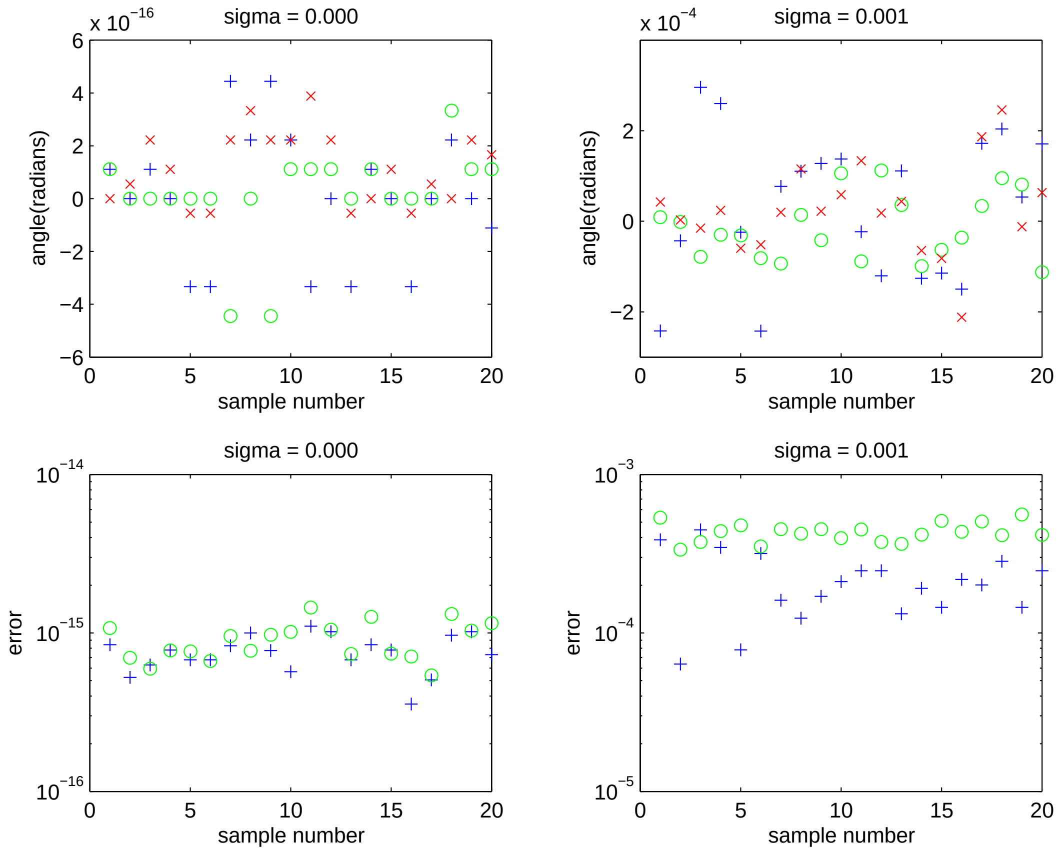


Figure 12.2. *Results of Challenge 6. The left column shows the result with no perturbation and the right column with perturbation of order* 10^{-3}*. The top graphs show the computed yaw (blue plusses), pitch (green circles), and roll (red x's), and the bottom graphs show the error in* $\mathbf{Q}$ *(blue plusses) and the error in the rotated positions (green circles).*

Chapter 13

Solutions: Case Study: Fitting Exponentials: An Interest in Rates

CHALLENGE 13.1. Sample MATLAB programs to solve this problem (and the others in this chapter) are available on the website. The results are shown in Figures 13.1 and 13.2. Note that the shape of the $\boldsymbol{w}$ clusters are rather circular; the sensitivity in the two components is approximately equal. This is not true of the $\boldsymbol{x}$ clusters; they are elongated in the direction corresponding to the eigenvector of the smallest singular value, since small changes in the data in this direction cause large changes in the solution. The length of the $\boldsymbol{x}$ cluster (and thus the sensitivity of the solution) is greater in Figure 13.2 because the condition number is larger.

CHALLENGE 13.2. The results are shown in Figure 13.3. One thing to note is that the sensitivity is not caused by the conditioning of the *linear* parameters; as t_{final} is varied, the condition number $\kappa(\boldsymbol{A})$ varies from 62 to 146, which is quite small. But the plots dramatically illustrate the fact that a wide range of $\boldsymbol{\alpha}$ values produce small residuals for this problem. This is an inherent limitation in the problem and we cannot change it. It means, though, that we need to be very careful in computing and reporting results of exponential fitting.

One important requirement on the data is that there be a sufficiently large number of points in the range where each of the exponential terms is large.

CHALLENGE 13.3. When the true $\boldsymbol{\alpha} = [-0.3, -0.4]$, the computations with 4 parameters produced unreliable results: $[-0.343125, -2.527345]$ for the first guess and $[-0.335057, -0.661983]$ for the second. The results for 2 parameters were somewhat better but still unreliable: $[-0.328577, -0.503422]$ for the first guess and $[-0.327283, -0.488988]$ for the second. Note that all of the runs produced one significant figure for the larger of the rate constants but had more trouble with the smaller.

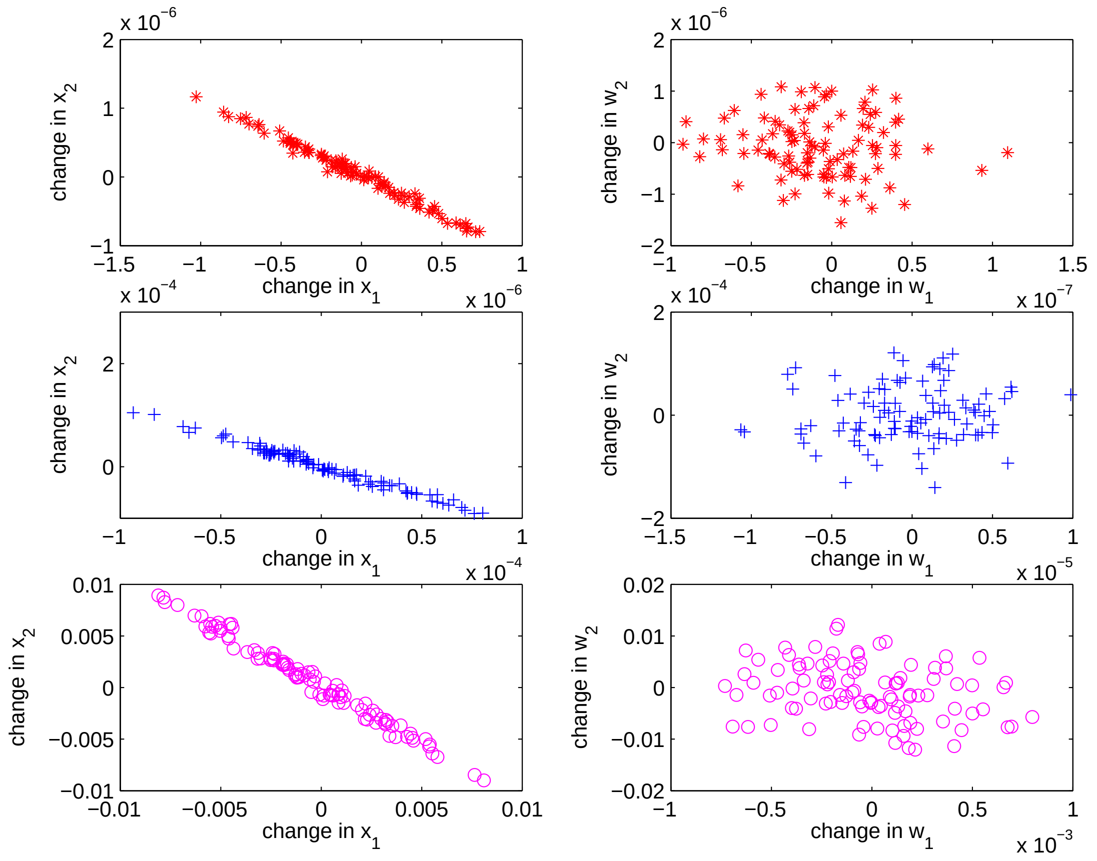


Figure 13.1. *Challenge 1, with* $\boldsymbol{\alpha} = -[0.3, 0.4]$ *and* $\eta = 10^{-6}$ *(top row),* $\eta = 10^{-4}$ *(middle row), and* $\eta = 10^{-2}$ *(bottom row). On the left, we plot the two components of* $\mathbf{x} - \mathbf{x}_{true}$ *and on the right* $\mathbf{w} - \mathbf{w}_{true}$.

For the harder problem, when the true $\boldsymbol{\alpha} = [-0.30, -0.31]$, the computations with 4 parameters produced $[-0.304889, -2.601087]$ for the first guess and $[-0.304889, -2.601087]$ for the second. The results for 2 parameters were again better but unreliable for the smaller rate constant: $[-0.304889, -0.866521]$ for the first guess and $[-0.304889, -0.866521]$ for the second.

The residuals for each of these fits are plotted in Figures 13.4 and 13.5. From the fact that none of the residuals from our computed solutions for the first problem resemble white noise, we can note that the solutions are not good approximations to the data. Troubles in the second problem are more difficult to diagnose, since the residual looks rather white. A single exponential function gives a good approximation to this data and the second term has very little effect on the residual. This is true to a lesser extent for the first dataset.

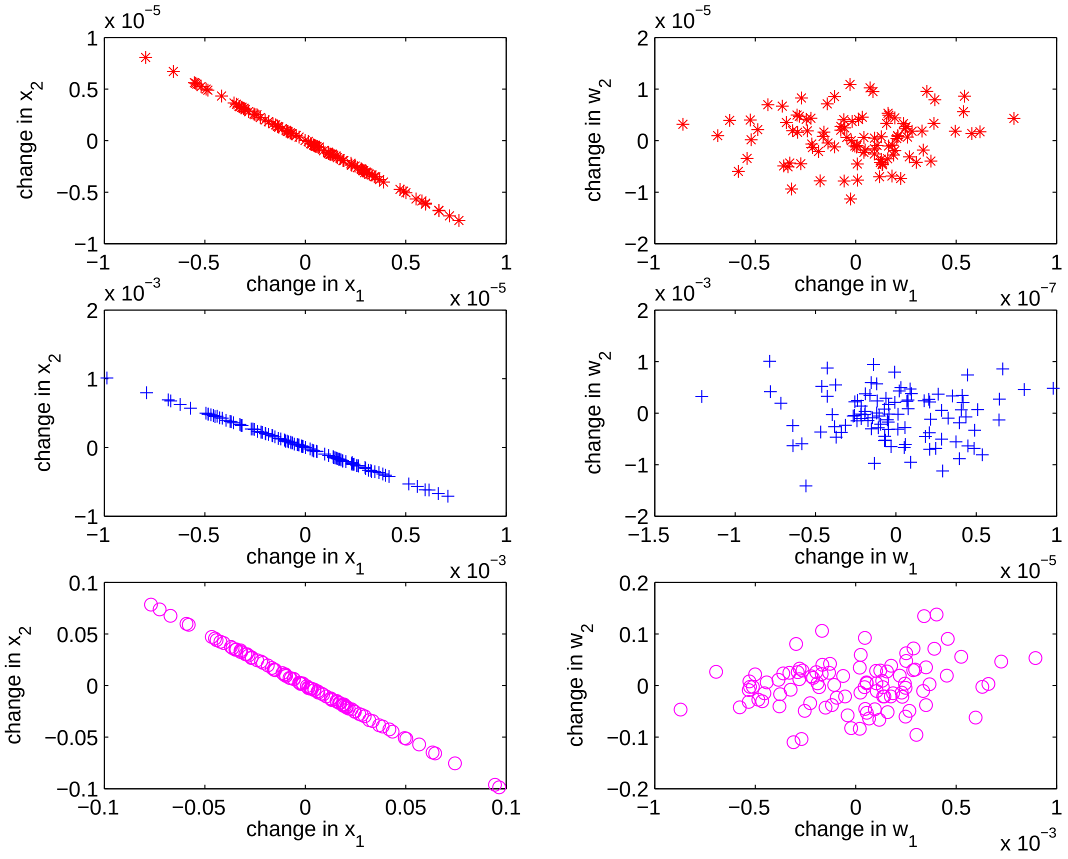


Figure 13.2. *Challenge 1, with* $\boldsymbol{\alpha} = -[0.30, 0.31]$ *and* $\eta = 10^{-6}$ *(top row),* $\eta = 10^{-4}$ *(middle row), and* $\eta = 10^{-2}$ *(bottom row). On the left, we plot the two components of* $\mathbf{x} - \mathbf{x}_{true}$ *and on the right* $\mathbf{w} - \mathbf{w}_{true}$*.*

Note that results will vary with the particular sequence of random errors generated.

CHALLENGE 13.4. We solved this problem using MATLAB's `lsqnonlin` and the two parameters $\boldsymbol{\alpha}$ using several initial guesses: $[-1, -2]$, $[-5, -6]$, $[-2, -6]$, $[0, -6]$, and $[-1, -3]$. All runs except the fourth produced values $\boldsymbol{\alpha} = [-1.6016, -2.6963]$ and a residual of 0.0024011. The fourth run produced a residual of .49631. The residuals for the five runs are shown in Figure 13.6. The four "good" residuals look like white noise of size about 10^{-4}, giving some confidence in the fit.

We tested the sensitivity of the residual norm to changes in the parameters by creating a contour plot in the neighborhood of the optimal values computed above,

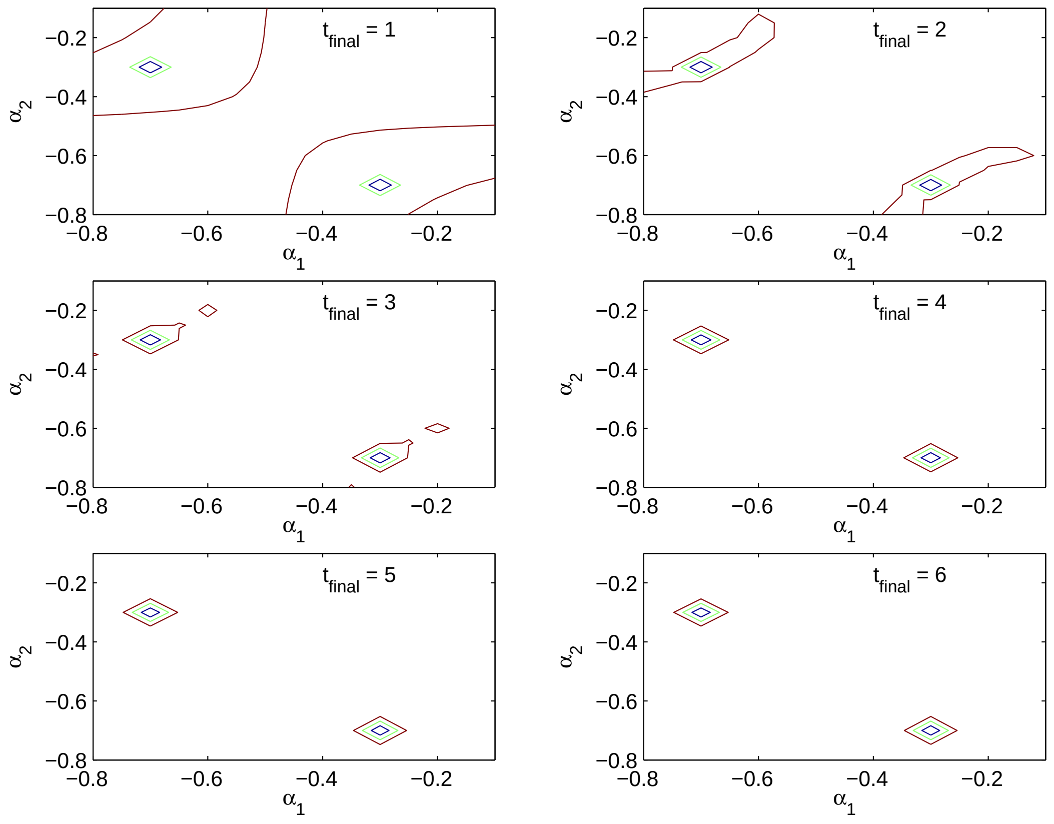


Figure 13.3. *Challenge 2. Contour plots of the residual norm as a function of the estimates of $\boldsymbol{\alpha}$ for various values of t_{final}. The contours marked are 10^{-2}, 10^{-6}, and 10^{-10}.*

shown in Figure 13.7. If the contours were square, then reporting the uncertainty in $\boldsymbol{\alpha}$ as $\pm$ some value would be appropriate, but as we can see, this is far from the case. The log=-2.6 contour outlines a set of $\boldsymbol{\alpha}$ values that changes the residual norm by less than 1%, the log = -2.5 contour denotes a change of less than 5%, and the log = -2.36 contour corresponds to a 10% change. The best value found was $\boldsymbol{\alpha} = [-1.601660, -2.696310]$, with residual norm $0.002401137 = 10^{-2.6196}$. Our uncertainty in the rate constants is rather large.

The "true solution," the value used to generate the data, was $\boldsymbol{\alpha} = [-1.6, -2.7]$ with $x_1 = -x_2 = 0.75$, and the standard deviation of the white noise was 10^{-4}.

Variants of Prony's method [1] provide alternate approaches to exponential fitting.

Exponential fitting is a very difficult problem, even when the number of terms n is known. It becomes even easier to be fooled when determining n is part of the problem!

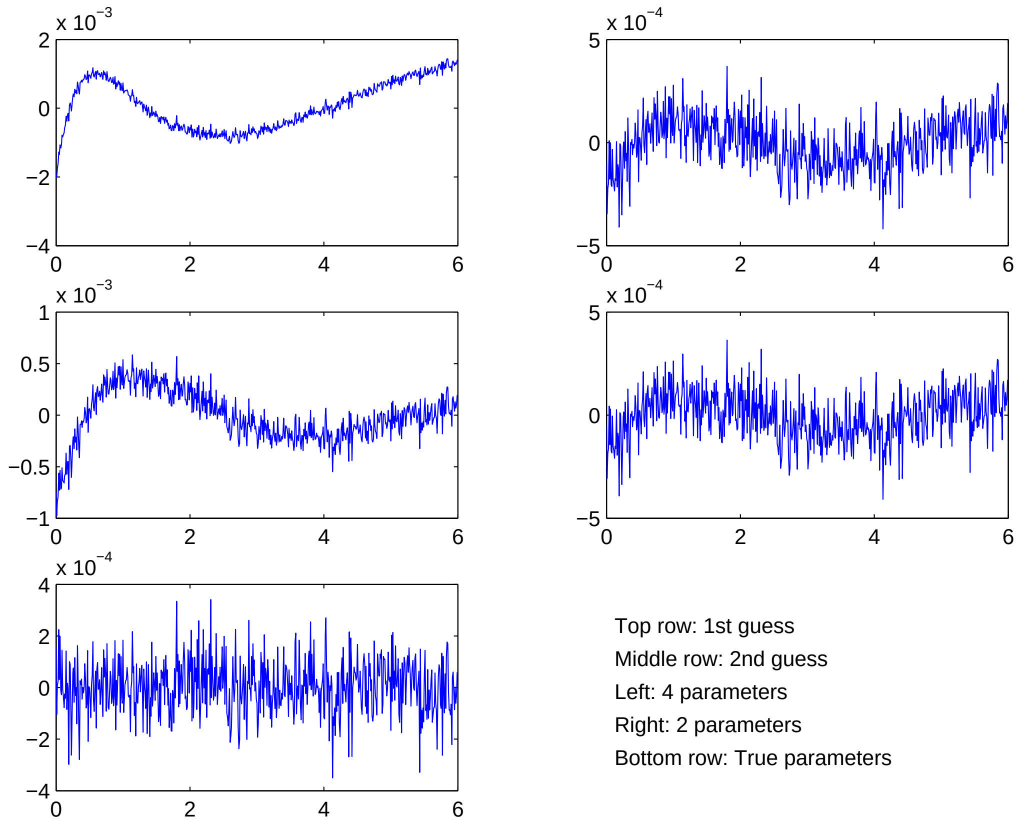


Figure 13.4. *Challenge 3. Residuals produced for the data with true* $\boldsymbol{\alpha} = [-0.3, -0.4]$ *by minimizing with 2 or 4 parameters and two initial guesses, and the residual provided by the true parameter values.*

[1] M. R. Osborne and G. K. Smyth, "A modified Prony algorithm for exponential function fitting," *SIAM J. Scientific Computing*, 16(1):119-138, 1995.

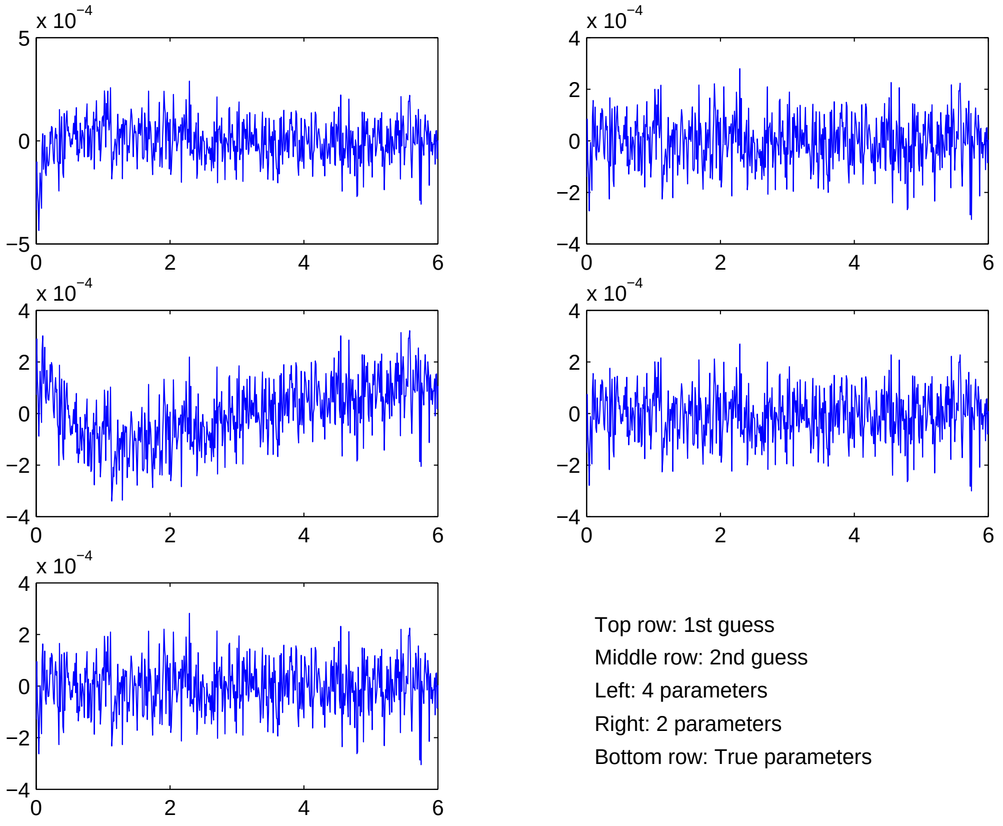


Figure 13.5. *Challenge 3. Residuals produced for the data with true* $\boldsymbol{\alpha} = [-0.30, -0.31]$ *by minimizing with 2 or 4 parameters and two initial guesses, and the residual provided by the true parameter values.*

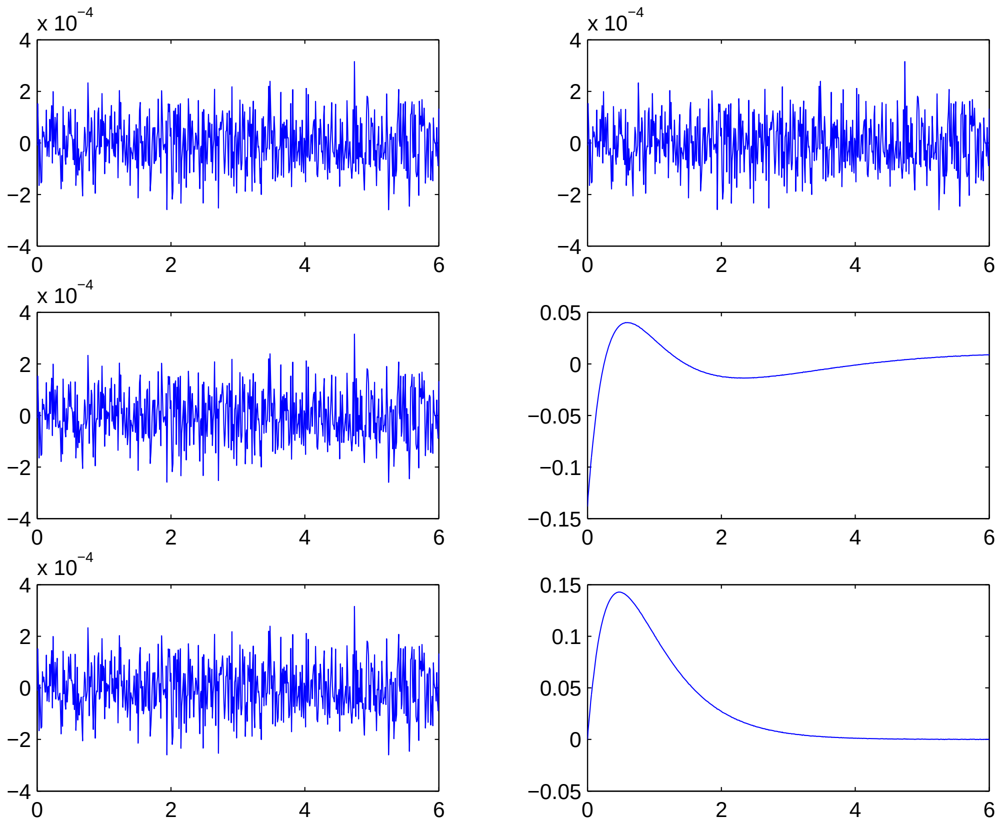


Figure 13.6. *Challenge 4. Residuals for the five computed solutions (residual component vs t), and, in the lower right, the data.*

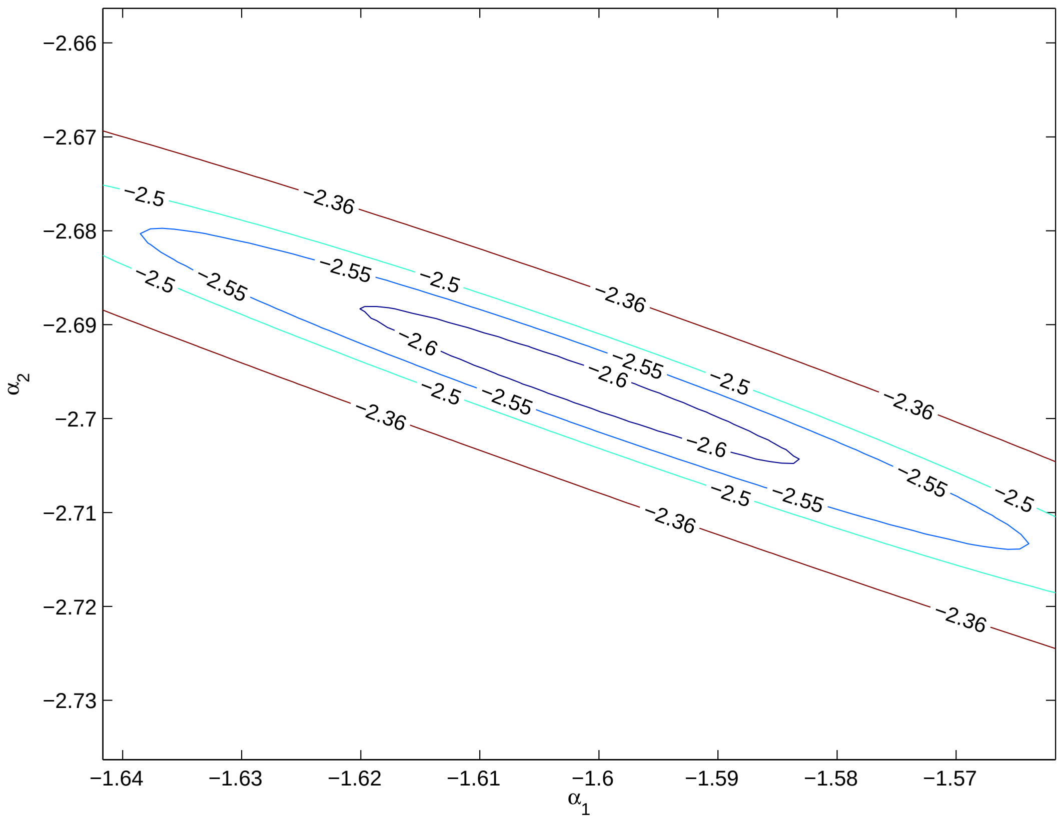


Figure 13.7. *Challenge 4. Contour plot of* $\log_{10}$ *of the norm of the residual for various values of the* $\boldsymbol{\alpha}$ *parameters.*

Chapter 14

Solutions: Case Study: Blind Deconvolution: Errors, Errors Everywhere

CHALLENGE 14.1. See the posted program `problem1_and_3.m`. The program is not difficult, but it is important to make sure that you do the SVD only once (at a cost of $O(mn^3)$) and then form each of the trial solutions at a cost of $O(n^2)$. This requires using the associative law of multiplication.

In fact, it is possible to form each solution by updating a previous one (by adding the newly added term) at a cost of $O(n)$, and this would be an even better algorithm, left as an exercise.

CHALLENGE 14.2.

(a) We know that

$$\begin{bmatrix} \boldsymbol{K} & \boldsymbol{g} \end{bmatrix} = \sum_{i=1}^{n+1} \tilde{\sigma}_i \tilde{\boldsymbol{u}}_i \tilde{\boldsymbol{v}}_i^T ,$$

so using the formula for $\begin{bmatrix} \boldsymbol{E} & \boldsymbol{r} \end{bmatrix}$ we see that

$$\begin{bmatrix} \boldsymbol{K} & \boldsymbol{g} \end{bmatrix} + \begin{bmatrix} \boldsymbol{E} & \boldsymbol{r} \end{bmatrix} = \sum_{i=1}^{n} \tilde{\sigma}_i \tilde{\boldsymbol{u}}_i \tilde{\boldsymbol{v}}_i^T .$$

Now, since $\tilde{\boldsymbol{v}}_{n+1}$ is orthogonal to $\tilde{\boldsymbol{v}}_i$ for $k = 1, \ldots, n$, it follows that

$$\left(\begin{bmatrix} \boldsymbol{K} & \boldsymbol{g} \end{bmatrix} + \begin{bmatrix} \boldsymbol{E} & \boldsymbol{r} \end{bmatrix} \right) \begin{bmatrix} \boldsymbol{f} \\ -1 \end{bmatrix} = - \sum_{i=1}^{n} \tilde{\sigma}_i \tilde{\boldsymbol{u}}_i \tilde{\boldsymbol{v}}_i^T \frac{1}{\tilde{v}_{n+1,n+1}} \tilde{\boldsymbol{v}}_{n+1} = \boldsymbol{0}.$$

Note that $\|[\boldsymbol{E}, \boldsymbol{r}]\|_F = \tilde{\sigma}_{n+1}$.

(b) This can be proven using the fact that $\|\boldsymbol{A}\|_F^2 = \text{trace}(\boldsymbol{A}^T \boldsymbol{A})$ where $\text{trace}(\boldsymbol{B})$ is the trace of the matrix $\boldsymbol{B}$, equal to the sum of its diagonal elements (or the sum of its eigenvalues). We can use the fact that trace($\boldsymbol{AB}$) = trace($\boldsymbol{BA}$). (See Challenge 12.3.)

It can also be proven just from the definition of the Frobenius norm and the fact that $\|\boldsymbol{Ux}\|_2 = \|\boldsymbol{x}\|_2$ for all vectors $\boldsymbol{x}$ and orthogonal matrices $\boldsymbol{U}$. Using this fact, and letting $\boldsymbol{a}_i$ be the ith column of $\boldsymbol{A}$, we see that

$$\|\boldsymbol{UA}\|_F^2 = \sum_{i=1}^{n} \|\boldsymbol{Ua}_i\|_2^2 = \sum_{i=1}^{n} \|\boldsymbol{a}_i\|_2^2 = \|\boldsymbol{A}\|_F^2 .$$

Similarly, letting $\widehat{\boldsymbol{a}}_i^T$ be the ith row of $\boldsymbol{A}$,

$$\|\boldsymbol{AV}\|_F^2 = \sum_{i=1}^{m} \|\widehat{\boldsymbol{a}}_i^T \boldsymbol{V}\|_2^2 = \sum_{i=1}^{n} \|\widehat{\boldsymbol{a}}_i\|_2^2 = \|\boldsymbol{A}\|_F^2 ,$$

and the conclusion follows.

(c) From the constraint

$$\begin{bmatrix} \boldsymbol{K}+\boldsymbol{E} & \boldsymbol{g}+\boldsymbol{r} \end{bmatrix} \begin{bmatrix} \boldsymbol{f} \\ -1 \end{bmatrix} = \boldsymbol{0},$$

we see that

$$\tilde{\boldsymbol{U}}^T \begin{bmatrix} \boldsymbol{K}+\boldsymbol{E} & \boldsymbol{g}+\boldsymbol{r} \end{bmatrix} \tilde{\boldsymbol{V}}\tilde{\boldsymbol{V}}^T \begin{bmatrix} \boldsymbol{f} \\ -1 \end{bmatrix} = \boldsymbol{0},$$

so

$$(\tilde{\boldsymbol{\Sigma}} + \tilde{\boldsymbol{E}})\tilde{\boldsymbol{f}} = \boldsymbol{0},$$

where $\tilde{\boldsymbol{E}} = \tilde{\boldsymbol{U}}^T [\boldsymbol{E}, \boldsymbol{r}]\, \tilde{\boldsymbol{V}}$ and $\tilde{\boldsymbol{f}} = \tilde{\boldsymbol{V}}^T \begin{bmatrix} \boldsymbol{f} \\ -1 \end{bmatrix}$. From part (b) we know that minimizing $\|[\boldsymbol{E}, \boldsymbol{r}]\|_F$ is the same as minimizing $\|\tilde{\boldsymbol{E}}\|_F$.

Therefore, to solve our problem, we want to make the smallest change to $\tilde{\boldsymbol{\Sigma}}$ that makes the matrix $\tilde{\boldsymbol{\Sigma}} + \tilde{\boldsymbol{E}}$ rank deficient, so that the constraint can be satisfied by a nonzero $\tilde{\boldsymbol{f}}$. Changing the $(n+1, n+1)$ element of $\tilde{\boldsymbol{\Sigma}}$ from $\tilde{\sigma}_{n+1}$ to 0 certainly makes the constraint feasible (by setting the last component of $\tilde{\boldsymbol{f}}$ nonzero and the other components zero). Any other change gives a bigger $\|\tilde{\boldsymbol{E}}\|_F$. Thus the smallest value of the minimization function is $\tilde{\sigma}_{n+1}$, and since we verified in part (a) that our solution has this value, we are finished.

If you don't find that argument convincing, we can be more precise. We use a fact found in the first pointer in Chapter 2: for any matrix $\boldsymbol{B}$ and vector $\boldsymbol{z}$ for which $\boldsymbol{Bz}$ is defined: $\|\boldsymbol{Bz}\|_2 \le \|\boldsymbol{B}\|_2 \|\boldsymbol{z}\|_2$, where $\|\boldsymbol{B}\|_2$ is defined to be the largest singular value of $\boldsymbol{B}$. Therefore,

- $\|\boldsymbol{B}\|_2 \le \|\boldsymbol{B}\|_F$, since we can see from part (b) and the singular value decomposition of $\boldsymbol{B}$ that the Frobenius norm of $\boldsymbol{B}$ is just the square root of the sum of the squares of its singular values.
- If $(\tilde{\boldsymbol{\Sigma}} + \tilde{\boldsymbol{E}})\tilde{\boldsymbol{f}} = \boldsymbol{0}$, then $\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{f}} = -\tilde{\boldsymbol{E}}\tilde{\boldsymbol{f}}$.

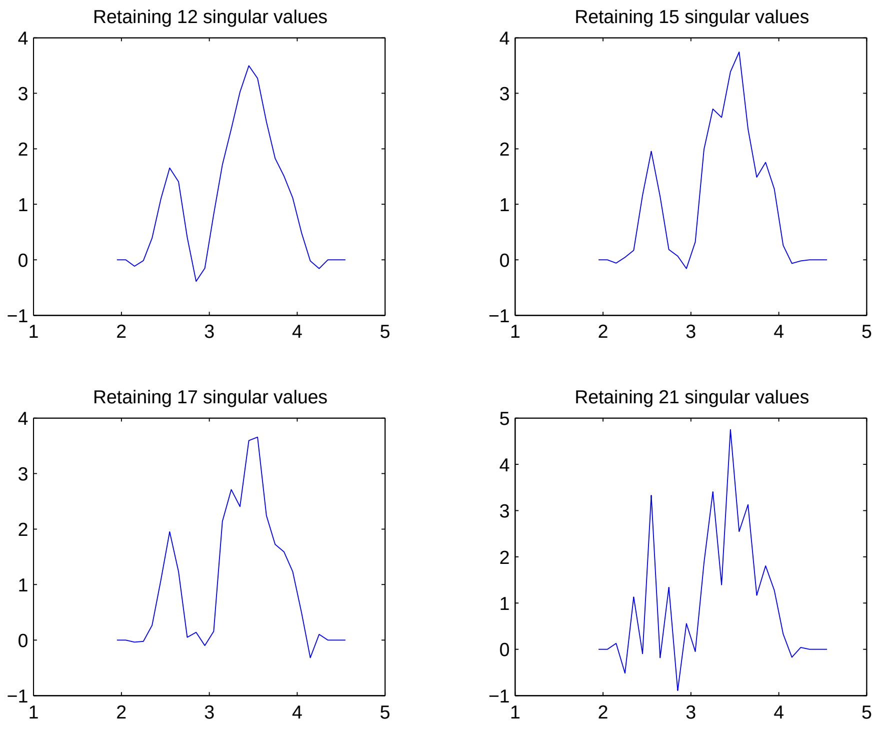


Figure 14.1. *Computed least squares solutions (counts vs. energies) for various values of the cutoff parameter $\tilde{n}$.*

- $\tilde{\sigma}_{n+1}^2 \|\tilde{\boldsymbol{f}}\|_2^2 = \sum_{i=1}^{n+1} \tilde{\sigma}_{n+1}^2 \tilde{\boldsymbol{f}}_i^2 \le \sum_{i=1}^{n+1} \tilde{\sigma}_i^2 \tilde{\boldsymbol{f}}_i^2 = \|\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{f}}\|_2^2$
- Therefore, $\tilde{\sigma}_{n+1}\|\tilde{\boldsymbol{f}}\|_2 \le \|\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{f}}\|_2 = \|\tilde{\boldsymbol{E}}\tilde{\boldsymbol{f}}\|_2 \le \|\tilde{\boldsymbol{E}}\|_2\|\tilde{\boldsymbol{f}}\|_2 \le \|\tilde{\boldsymbol{E}}\|_F\|\tilde{\boldsymbol{f}}\|_2$, so we conclude that $\|\tilde{\boldsymbol{E}}\|_F \ge \tilde{\sigma}_{n+1}$, and we have found a minimizing solution.

CHALLENGE 14.3. See the program `problem1_and_3.m` on the website.

CHALLENGE 14.4.

Model 1: Least squares.

To estimate the variance of the error, note that in the least squares model, the last 5 components of the right-hand side $\boldsymbol{U}^T\boldsymbol{g}$ cannot be zeroed by any choice of $\boldsymbol{f}$, so if we believe the model, we believe thast these are entirely due to error. All

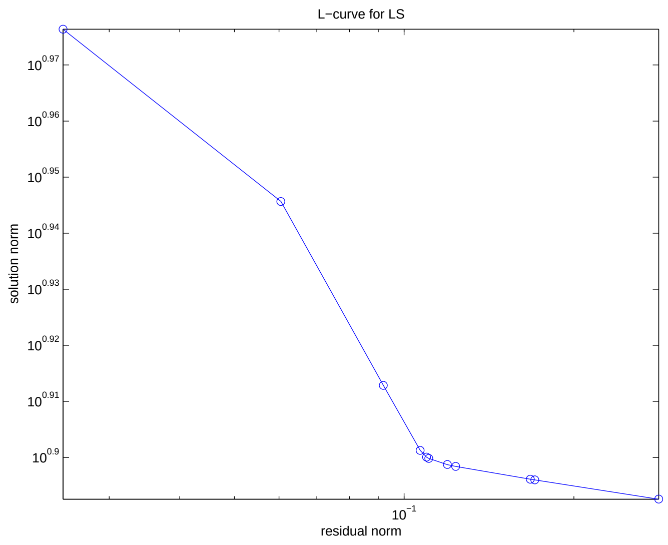


Figure 14.2. *The L-curve for least squares solutions.*

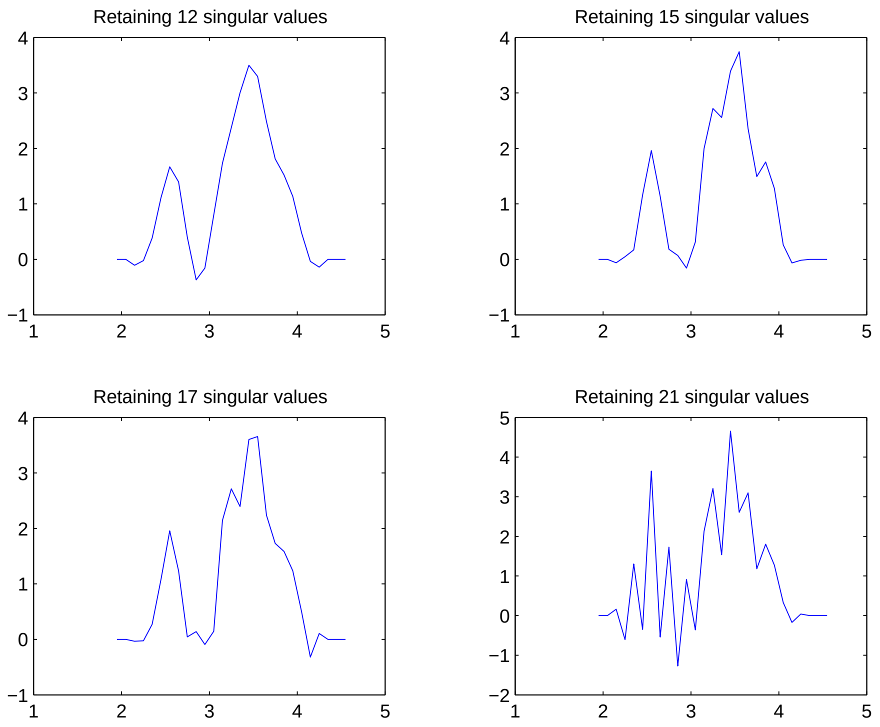


Figure 14.3. *Computed total least squares solutions (counts vs. energies) for various values of the cutoff parameter $\tilde{n}$.*

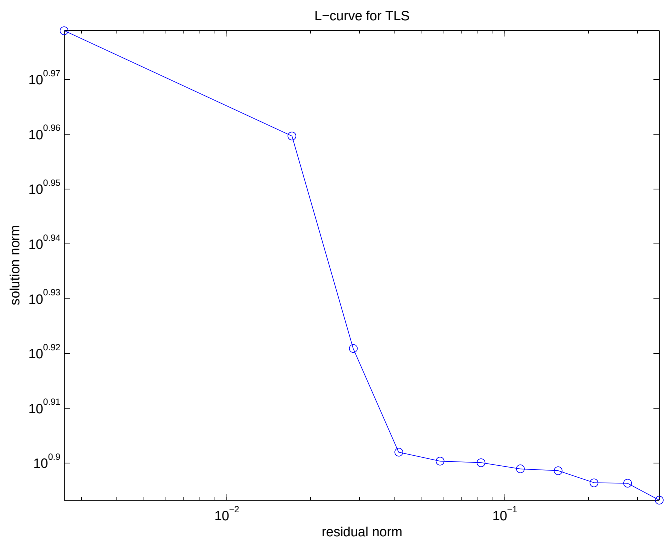


Figure 14.4. *The L-curve for total least squares solutions.*

other components should have at least some data in addition to noise. Therefore, estimate the variance using the last 5 to get $\delta^2 = 1.2349 \times 10^{-4}$.

The condition number of the matrix, the ratio of largest to smallest singular value, is 61.8455. This is a well-conditioned matrix! Most spectroscopy problems have a very ill-conditioned matrix. (An ill-conditioned one would have a condition number of 10^3 or more.) This is a clue that there is probably error in the matrix, moving the small singular values away from zero.

We try various choices of $\tilde{n}$, the number of singular values retained, and show the results in Figure 14.1 (blue solid curves). The discrepancy principle predicts the residual norm to be $\delta\sqrt{m} = 0.0577$. This is most closely matched by retaining 21 singular values, which gives 7 peaks, contradicting the given information that there are at most 5 peaks. It also produces some rather large magnitude negative peaks, and we know that counts need to be nonnegative. So the least squares model does not seem to fit the data well.

An alternate way to pick $\tilde{n}$ is to use the L-curve. This is a plot of the log of the solution norm vs the log of the residual norm. It is called an L-curve because its shape often resembles that of the letter L. What we really want is a small residual and a solution norm that is not unreasonably big. So it has been proposed that we take the value of $\tilde{n}$ at the corner of the L-curve, since if we take a smaller $\tilde{n}$, the residual norm increases fast, and if we take a larger one, the solution norm increases fast. This plot, shown in Figure 14.2, advises that we should retain 15-17 singular values, and referring to Figure 14.1, this yields 4 peaks, consistent with our given information.

(The theoretical properties of the L-curve, as well as any other method that does not require the variance to be given in advance, are not good, but often this method is more robust to errors in assumptions about the model, such as underestimating the variance or not accounting for errors in the matrix.)

An alternate heuristic is to look for a value of $\tilde{n}$ that makes the residual look most like white noise, but since our error is not normally distributed, this heuristic doesn't have much meaning for our problem.

An excellent way to approach this problem is to generate your own test data, for which you know the true solution, and use it to gain insight into the choice of $\tilde{n}$.

Model 2: TLS.

Sample solutions are shown in Figure 14.3. The discrepancy principle doesn't give much insight for TLS, so we use more heuristic methods, giving us even less confidence in our choice.

For example, from Figure 14.4 we see that the L-curve corner occurs when 15 singular values are retained, giving a solution that looks very much like the L-curve least squares solution. Because the number of peaks is reasonable, and because there are only a small number of negative values in the solution, and these have small magnitude, we might accept this solution.

Now we need to extract the energies and estimated counts for the 4 types of particles. I have normalized so that the count for the lowest energy peak is 1.

The Computed Estimate to Energy Levels and Counts

bin centers	2.55	3.25	3.55	3.85
relative counts	1.00	1.39	1.91	0.90

A spectroscopist would actually estimate the counts by taking the integral under each of the 4 peaks, and estimate the energy by the centroid of the peak, but this is difficult since three of the peaks are not well separated.

The Truth.

The program used to generate the data is posted. The variance of the error is 10^{-4}.

The True Energy Levels and Counts

energy	2.54	3.25	3.53	3.85
relative counts	1	1.5	2	1

So, despite all of the errors, our computed solution estimates the energy levels to 2 digits and the relative counts to within 10%.

Chapter 15

Solutions: Case Study: Blind Deconvolution: A Matter of Norm

CHALLENGE 15.1. Writing out the expressions $\boldsymbol{Ef}$ and $\boldsymbol{F}\widehat{\boldsymbol{e}}$ component by component, we find that $\boldsymbol{F}$ is a Toeplitz matrix of size $m \times (m+n-1)$ with first row equal to $[f_n, f_{n-1}, \ldots, f_1, 0, \ldots, 0]$ and first column equal to $[f_n, 0, \ldots, 0]$.

CHALLENGE 15.2. Let $\boldsymbol{d} = [1, \sqrt{2}, \ldots, \sqrt{n}, \ldots, \sqrt{n}, \sqrt{n-1}, \ldots, 1]$ be a vector of length $m+n-1$ and let $\boldsymbol{D}$ be the diagonal matrix with entries $\boldsymbol{d}$. Then

$$\begin{aligned} F(\widehat{\boldsymbol{e}}, \boldsymbol{f}) &\equiv \frac{1}{2}\|\boldsymbol{E}\|_F^2 + \frac{1}{2}\|\boldsymbol{r}\|_2^2 \\ &= \frac{1}{2}\sum_{i=1}^{m+n-1} d_i^2 \widehat{e}_i^2 + \frac{1}{2}\sum_{i=1}^{m} r_i^2 \\ &= \frac{1}{2}\sum_{i=1}^{m+n-1} d_i^2 \widehat{e}_i^2 + \frac{1}{2}\sum_{i=1}^{m}(g_i - \sum_{j=1}^{n}(k_{ij}+e_{ij})f_j)^2. \end{aligned}$$

We need the gradient and Hessian matrix of this function. Noting that $\boldsymbol{E}_{ij} = \widehat{e}_{n+i-j}$, and letting $\delta_{ij} = 0$ if $i \neq j$ and 1 if $i = j$, we compute

$$\begin{aligned} \frac{\partial F(\widehat{\boldsymbol{e}}, \boldsymbol{f})}{\partial \widehat{e}_\ell} &= d_\ell^2 \widehat{e}_\ell - \sum_{i=1}^{m} r_i f_{n+i-\ell} = (\boldsymbol{D}^2 \widehat{\boldsymbol{e}} - \boldsymbol{F}^T \boldsymbol{r})_\ell, \\ \frac{\partial F(\widehat{\boldsymbol{e}}, \boldsymbol{f})}{\partial f_\ell} &= -\sum_{i=1}^{m} r_i (k_{i\ell} + \widehat{e}_{n+i-\ell}) = -((\boldsymbol{K}+\boldsymbol{E})^T \boldsymbol{r})_\ell, \\ \frac{\partial^2 F(\widehat{\boldsymbol{e}}, \boldsymbol{f})}{\partial \widehat{e}_\ell \partial \widehat{e}_q} &= \delta_{\ell,q} d_\ell^2 + \sum_{i=1}^{m} f_{n+i-\ell} f_{n+i-q} = (\boldsymbol{D}^2 + \boldsymbol{F}^T \boldsymbol{F})_{\ell q}, \\ \frac{\partial^2 F(\widehat{\boldsymbol{e}}, \boldsymbol{f})}{\partial \widehat{e}_\ell \partial f_q} &= r_{\ell+q} + \sum_{i=1}^{m} (k_{iq} + e_{iq}) f_{n+i-\ell} = (\boldsymbol{R} + (\boldsymbol{K}+\boldsymbol{E})^T \boldsymbol{F})_{\ell q}, \end{aligned}$$

$$\frac{\partial F(\widehat{\boldsymbol{e}}, \boldsymbol{f})}{\partial f_\ell \partial f_q} = \sum_{i=1}^{m} (k_{i\ell} + e_{i\ell})(k_{iq} + e_{iq}) = ((\boldsymbol{K} + \boldsymbol{E})^T (\boldsymbol{K} + \boldsymbol{E}))_{\ell q},$$

where out-of-range entries in summations are assumed to be zero and $\boldsymbol{R}$ is a matrix whose nonzero entries are components of $\boldsymbol{r}$. So

$$\boldsymbol{g} = \nabla F(\widehat{\boldsymbol{e}}, \boldsymbol{f}) = \left[\begin{array}{c} \boldsymbol{D}^2 \widehat{\boldsymbol{e}} - \boldsymbol{F}^T \boldsymbol{r} \\ (\boldsymbol{K} + \boldsymbol{E})^T \boldsymbol{r} \end{array} \right],$$

$$\boldsymbol{H}(\widehat{\boldsymbol{e}}, \boldsymbol{f}) = \left[\begin{array}{cc} \boldsymbol{D}^2 + \boldsymbol{F}^T \boldsymbol{F} & \boldsymbol{R}^T + \boldsymbol{F}^T (\boldsymbol{K} + \boldsymbol{E}) \\ \boldsymbol{R} + (\boldsymbol{K} + \boldsymbol{E})^T \boldsymbol{F} & (\boldsymbol{K} + \boldsymbol{E})^T (\boldsymbol{K} + \boldsymbol{E}) \end{array} \right].$$

The Newton direction is the solution to $\boldsymbol{H}(\widehat{\boldsymbol{e}}, \boldsymbol{f})\boldsymbol{p} = -\boldsymbol{g}$.

CHALLENGE 15.3. The least squares problem is of the form

$$\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2,$$

where

$$\boldsymbol{x} = \left[\begin{array}{c} \Delta \widehat{\boldsymbol{e}} \\ \Delta \boldsymbol{f} \end{array} \right]$$

and $\boldsymbol{A}$ and $\boldsymbol{b}$ are the given matrix and vector. So to minimize $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2 = (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$, we set the derivative equal to zero, obtaining

$$\boldsymbol{A}^T \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^T \boldsymbol{b} = \boldsymbol{0}.$$

The solution to this equation is a minimizer if the second derivative $\boldsymbol{A}^T\boldsymbol{A}$ is positive definite (which requires that $\boldsymbol{A}$ have full column rank). Returning to our original notation, we obtain

$$\left[\begin{array}{cc} \boldsymbol{F} & \boldsymbol{K} + \boldsymbol{E} \\ \boldsymbol{D} & \boldsymbol{0} \end{array} \right]^T \left[\begin{array}{cc} \boldsymbol{F} & \boldsymbol{K} + \boldsymbol{E} \\ \boldsymbol{D} & \boldsymbol{0} \end{array} \right] \left[\begin{array}{c} \Delta \widehat{\boldsymbol{e}} \\ \Delta \boldsymbol{f} \end{array} \right] = - \left[\begin{array}{cc} \boldsymbol{F} & \boldsymbol{K} + \boldsymbol{E} \\ \boldsymbol{D} & \boldsymbol{0} \end{array} \right]^T \left[\begin{array}{c} -\boldsymbol{r} \\ \boldsymbol{D}\widehat{\boldsymbol{e}} \end{array} \right],$$

and this matches the expression $\boldsymbol{H}\boldsymbol{p} = -\boldsymbol{g}$ from Challenge 2 except that the matrix $\boldsymbol{R}$ (which should be small if the model is good) is omitted.

CHALLENGE 15.4. See the MATLAB program posted on the website.

CHALLENGE 15.5.

(a) Given any $\Delta\widehat{\boldsymbol{e}}$, $\Delta\boldsymbol{f}$, let

$$\begin{bmatrix} \bar{\sigma}_{\mathbf{1}} \\ \bar{\sigma}_{\mathbf{2}} \\ \bar{\sigma}_{\mathbf{3}} \end{bmatrix} = \left| \begin{bmatrix} \boldsymbol{F} & \boldsymbol{K}+\boldsymbol{E} \\ \boldsymbol{D} & \mathbf{0} \\ \mathbf{0} & \lambda\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \Delta\widehat{\boldsymbol{e}} \\ \Delta\boldsymbol{f} \end{bmatrix} + \begin{bmatrix} -\boldsymbol{r} \\ \boldsymbol{D}\widehat{\boldsymbol{e}} \\ \lambda\boldsymbol{f} \end{bmatrix} \right|.$$

Then $\Delta\widehat{\boldsymbol{e}}$, $\Delta\boldsymbol{f}$, $\bar{\sigma}_{\mathbf{1}}$, $\bar{\sigma}_{\mathbf{2}}$, and $\bar{\sigma}_{\mathbf{3}}$ form a feasible solution to the linear programming problem, and

$$\bar{\sigma} = \sum_{i=1}^{m} \bar{\sigma}_{1_i} + \sum_{i=1}^{q} \bar{\sigma}_{2_i} + \sum_{i=1}^{n} \bar{\sigma}_{3_i} = \left\| \begin{bmatrix} \boldsymbol{F} & \boldsymbol{K}+\boldsymbol{E} \\ \boldsymbol{D} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta\widehat{\boldsymbol{e}} \\ \Delta\boldsymbol{f} \end{bmatrix} + \begin{bmatrix} -\boldsymbol{r} \\ \boldsymbol{D}\widehat{\boldsymbol{e}} \end{bmatrix} \right\|_p.$$

Therefore, a solution to the linear programming problem minimizes the norm, and a minimizer of the norm is a solution to the linear programming problem, so the two are equivalent.

(b) By similar reasoning, we obtain

$$\begin{array}{lrcccl} \min_{\Delta\widehat{\boldsymbol{e}},\Delta\boldsymbol{f},\bar{\sigma}} & \bar{\sigma} & & & & \\ & & & & & \\ \text{subject to} & -\bar{\sigma}\mathbf{1} & \leq & \boldsymbol{F}\Delta\widehat{\boldsymbol{e}} + (\boldsymbol{K}+\boldsymbol{E})\Delta\boldsymbol{f} - \boldsymbol{r} & \leq & \bar{\sigma}\mathbf{1} \\ & -\bar{\sigma}\mathbf{1} & \leq & \boldsymbol{D}\Delta\widehat{\boldsymbol{e}} + \boldsymbol{D}\widehat{\boldsymbol{e}} & \leq & \bar{\sigma}\mathbf{1} \\ & -\bar{\sigma}_3\mathbf{1} & \leq & \lambda\Delta\boldsymbol{f} + \lambda\boldsymbol{f} & \leq & \bar{\sigma}_3\mathbf{1} \end{array}$$

where $\mathbf{1}$ is a column vector with each entry equal to 1, and of dimension m in the first two inequalities, q in the second two, and n in the last two.

CHALLENGE 15.6. See the MATLAB program posted on the website.

CHALLENGE 15.7. Results for various values of λ are shown in Figures 15.1 and 15.2. The estimated counts are summarized in the following table:

The Computed Estimate to Energy Levels and Counts				
bin centers	2.55	3.25	3.55	3.85
True counts	1.00	1.50	2.00	1.00
Least Squares	1.00	1.39	1.91	0.90
STLS	1.00	1.20	1.59	0.64
STLN, 1-Norm	1.00	0.96	1.36	0.60

STLN using the ∞-norm produced counts that were sometimes quite negative; nonnegativity constraints could be added to improve the results. All of the structured algorithms had a linear convergence rate, rather than the quadratic rate

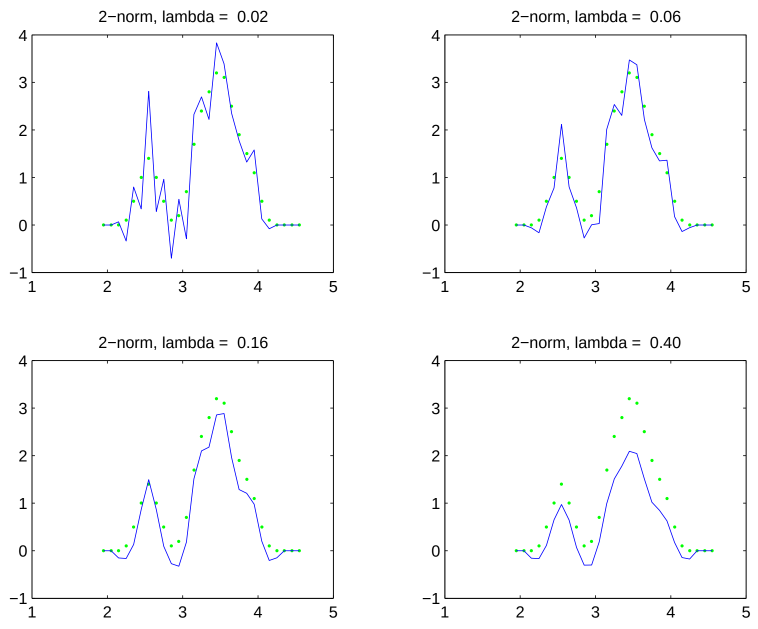


Figure 15.1. *Results from the Structured Total Least Squares algorithm for various values of λ.*

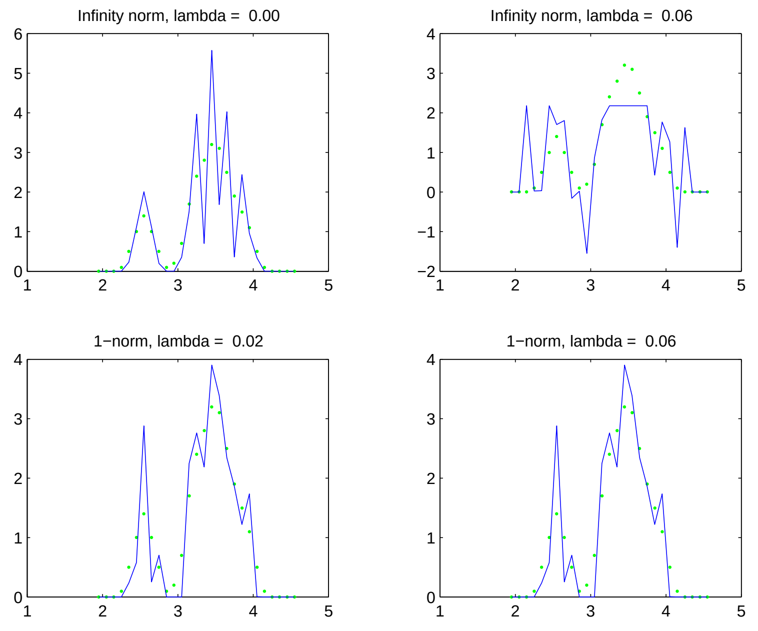


Figure 15.2. *Results from the Structured Total Least Norm algorithm, using the 1-norm and the ∞-norm, for various values of λ.*

expected from Newton's method, because the residual $\boldsymbol{r}$ in this problem is large, so the approximate Newton direction is not very accurate.

Least squares works best on this dataset, because the Toeplitz assumption used by the structured algorithms STLS and STLN is violated by the way the data was generated. It is worthwhile to generate a new data set, satisfying this assumption, and experiment further.

Unit IV

SOLUTIONS: Monte Carlo Computations

Chapter 16

Solutions: Monte Carlo Principles

CHALLENGE 16.1. The mean is the sum of the samples divided by the number of samples: $\mu_6 = 24/6 = 4$. The variance is

$$\sigma_6^2 = \frac{1}{6}\left[(1-4)^2 + (2-4)^2 + (5-4)^2 + (8-4)^2 + (6-4)^2 + (2-4)^2\right] = \frac{19}{3}.$$

CHALLENGE 16.2. The probability of drawing each card is 1/10, so the mean of the distribution is

$$\mu = \frac{1}{10}(1+2+3+4+5+6+7+8+10+10) = 5.6.$$

The variance is

$$\sigma^2 = \frac{1}{10}\left[(1-5.6)^2 + (2-5.6)^2 + (3-5.6)^2 + (4-5.6)^2 + (5-5.6)^2 + (6-5.6)^2 \right.$$
$$\left. +(7-5.6)^2 + (8-5.6)^2 + (10-5.6)^2 + (10-5.6)^2\right] = 9.04\,.$$

CHALLENGE 16.3. Clearly $f(x) \geq 0$, and

$$\int_0^1 3x^2 \mathrm{d}x = x^3\Big|_0^1 = 1.$$

We calculate

$$\mu = \int_0^1 x\,(3x^2)\mathrm{d}x = \left.\frac{3x^4}{4}\right|_0^1 = \frac{3}{4}$$

and

$$\sigma^2 = \int_0^1 (x-3/4)^2(3x^2)\mathrm{d}x = 0.0375\,.$$

CHALLENGE 16.4.

```
function y = strange_random()

% We subtract 2 from the average sample value for randmy, to make the mean 0.
% Then we divide by the standard deviation, to make the resulting variance 1.

y = sqrt(1000)*(sum(randmy(1000))/1000 - 2)/sqrt(5);
```

CHALLENGE 16.5. In this program, z is a sample from a uniform distribution on [0,1] and y is a sample from the desired distribution.

```
z = rand(1);
if (z < .6) then
  y = 0;
else
  y = 1;
end
```

Chapter 17

Case Study: Monte-Carlo Minimization and Counting: One, Two, . . . , Too Many

(coauthored by Isabel Beichl and Francis Sullivan)

CHALLENGE 17.1. The programs `myfmin.m` and `myfminL.m` on the website solve this problem but do not make the graph.

CHALLENGE 17.2. **(Partial Solution)** The program `sim_anneal.m` on the website is one implementation of simulated annealing, and it can be run using `problem1_and_2.m`. To finish the problem, experiment with the program. Be sure to measure reliability as well as cost, and run multiple experiments to account for the fact that the method is randomized. Also comment on the number of runs that converge to $x = 1.7922$, which is a local minimizer with a function value not much worse than the global minimizer.

CHALLENGE 17.3.

(a) Experiments with MATLAB's `travel` program show that it works well for up to 50 cities but, as is to be expected, slows down for larger sets. It's interesting and important to note that the solution is always a tour that does not cross itself. We'll return to this point shortly.

(b) Figures 17.1–17.3 show the results of simulated annealing for 100 random locations with temperature $T = 1, 0.1, 0.01$, where "score" is the length of the tour. The actual tours for $T = 0.1$ and $T = 0.01$ are shown in Figures 17.4 and 17.5. Note that the result for 0.01 looks pretty good but not that much better than the output

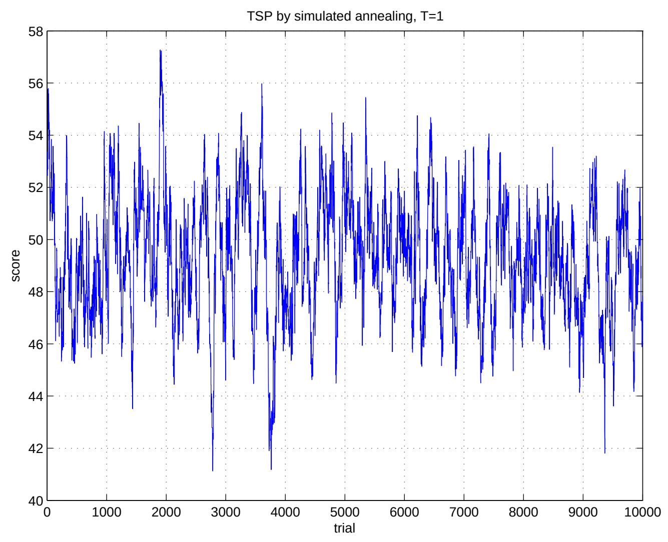


Figure 17.1. *TSP by simulated annealing, $T = 1$.*

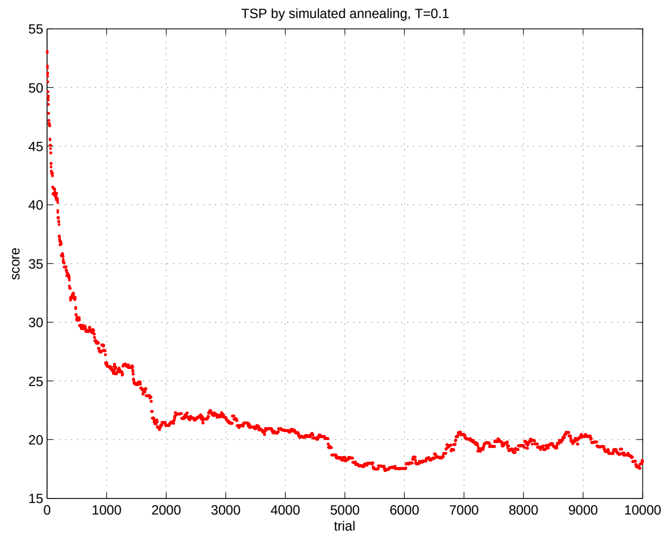


Figure 17.2. *TSP by simulated annealing, $T = 0.1$.*

for $T = 0.1$. However, the $T = 1$ result looks completely random and gets nowhere near a low cost tour. This demonstrates that lowering the temperature really does give a better approximation. However, because the $T = 0.01$ tour crosses itself, we know that it's still not the true solution. And we don't know the true minimum score (distance) or an algorithm for setting and changing T. Figuring out how to

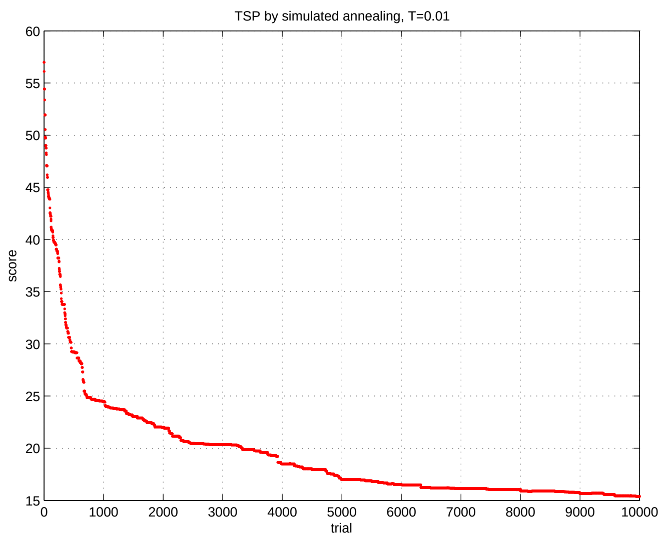


Figure 17.3. *TSP by simulated annealing, $T = 0.01$.*

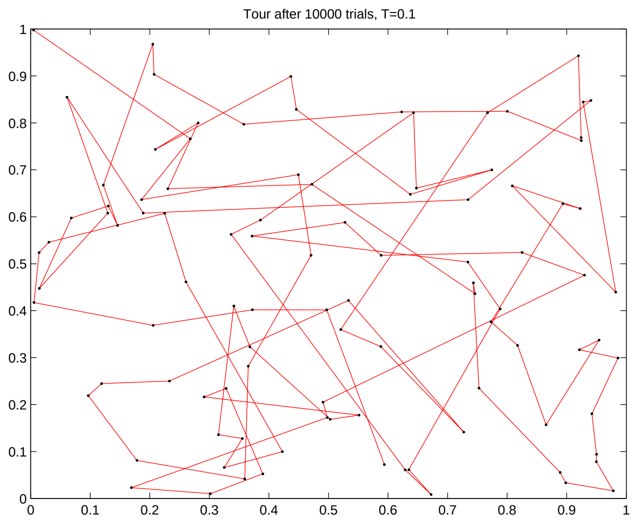


Figure 17.4. *Tour produced for TSP by simulated annealing, $T = 0.1$.*

vary T is called determining the cooling schedule. One generally wants to use a lower value of T as the solution is approached. The idea is to avoid a move that would bounce away from the solution when we're almost there.

How one designs a cooling schedule depends on analysis of the problem at hand. Some general techniques have been proposed but cooling schedules are still an active research topic.

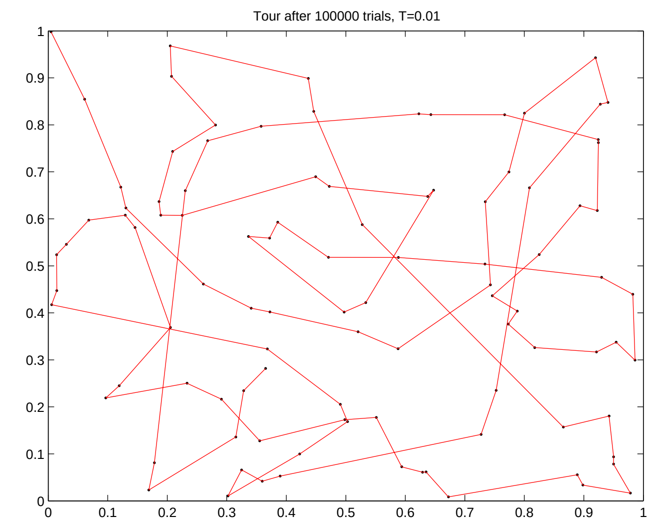


Figure 17.5. *Tour produced for TSP by simulated annealing, $T = 0.01$*

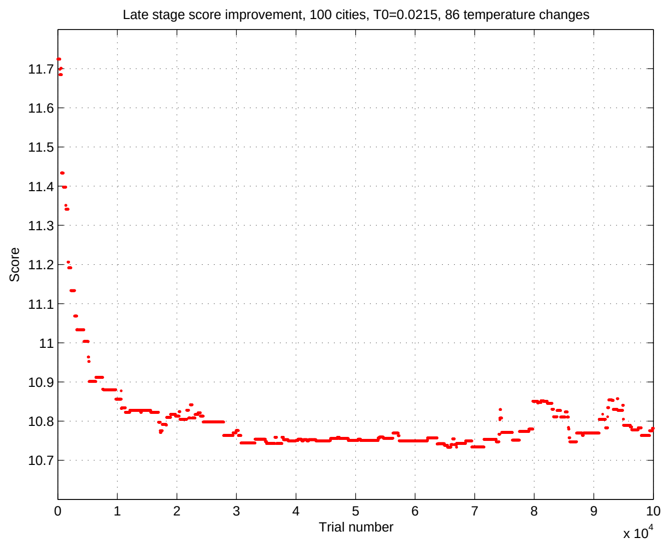


Figure 17.6. *TSP scores by simulated annealing, $T = 0.0215$, logarithmic cooling schedule.*

The most popular general approach setting a cooling schedule is to change T whenever a proposed move is accepted. Suppose that the initial temperature is T_0 and a proposed move is finally accepted after k trials. Then the temperature is reset to $T_0/\log(k)$. The idea behind the use of $\log(k)$ in the denominator is that the number of trials k required before generating a random number less than

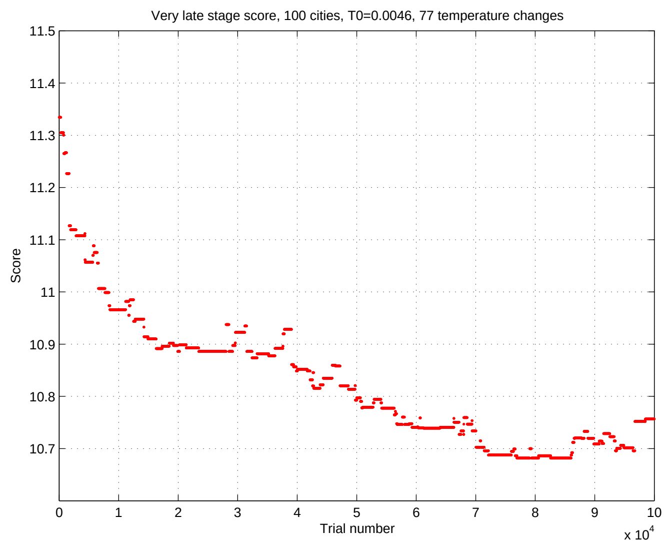


Figure 17.7. *TSP scores by simulated annealing,* $T = 0.0046$*, logarithmic cooling schedule.*

$\exp(-1/T)$ is $\exp(1/T)$ on average, and so $1/T$ should look something like $\log(k)$. This is the famous logarithmic cooling schedule [1].

Figures 17.6, 17.7 and 17.8 illustrate application of simulated annealing with a logarithmic cooling schedule to a TSP with 100 random locations. The first two graphs show how the score evolves over 100,000 trials at a low temperature. Note that not many proposed moves that increase the score are accepted and that the score does not improve very much. The last figure is a picture of the best tour obtained. Because it crosses itself, it's not the optimal tour. Getting that requires more computation and/or more sophisticated cooling schedules. Solving TSP for 100 random locations is really quite difficult!

If you think this use of simulated annealing to attack the TSP seems quite informal and heuristic rather than analytic, you're right. In fact, some have argued that simulated annealing is not really an optimization *method* but rather a collection of heuristic techniques that help in some cases. However, there is an important, recently discovered connection between the central idea of simulated annealing and use of Monte Carlo to approximate solutions to NP-hard problems, including determining the volume of a bounded convex region K in $\mathcal{R}^n$. If n is large, finding $Vol(K)$ can be a very hard problem. The most well-developed approach is to define a sequence of convex sets:

$$K_0 \subset K_1 \subset K_2 \subset \ldots \subset K_m = K$$

where $Vol(K_0)$ is easy to evaluate. For each i, perform a random walk in K_i and count how many walkers happen to be in K_{i-1}. This gives an estimate of $Vol(K_{i-1})/Vol(K_i)$ and the product of these estimates for all i is an estimate for

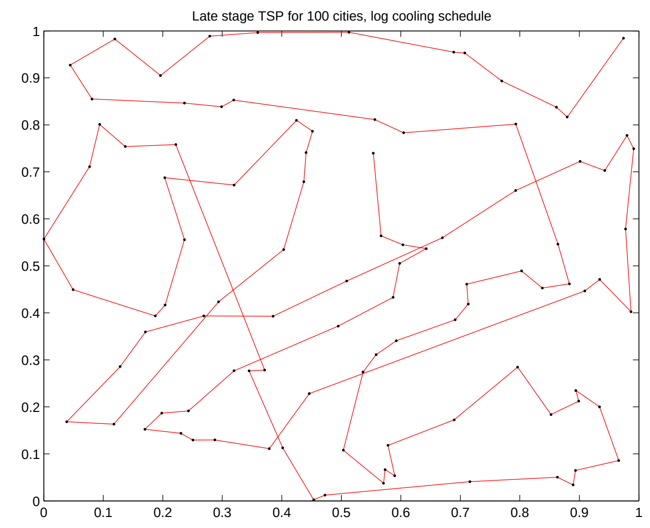


Figure 17.8. *Tour produced for TSP by simulated annealing, logarithmic cooling schedule.*

$Vol(K)$.

The connection to simulated annealing comes in a couple of ways. For one thing, the random walk can done using a Metropolis algorithm with a different rejection rate (i.e. a different temperature) for each i. A more recent idea is to recognize that the volume is the integral of the characteristic function of the set K so we can try to approach this integral by integrating a sequence of other, easier functions instead. In particular we can embed the problem in $\mathcal{R}^{n+1}$ by adding an extra coefficient x_0 to the points in $\mathcal{R}^n$ and then choose functions $f_0 < f_1 < f_2 < ...f_m$ where f_m is the characteristic function of K but the others look like $\exp(-x_0/T)$ in the extra coefficient, x_0.

Another example of use of the idea of simulated annealing is the KRS algorithm used in the next challenge. Those who have become fascinated by this subject might want to try to identify the "temperature" in this case in order to understand why KRS is a form of simulated annealing.

CHALLENGE 17.4.

(a) Here are some explicit counts, some done by hand and some by `latticecount.m` by Thomas DuBois.

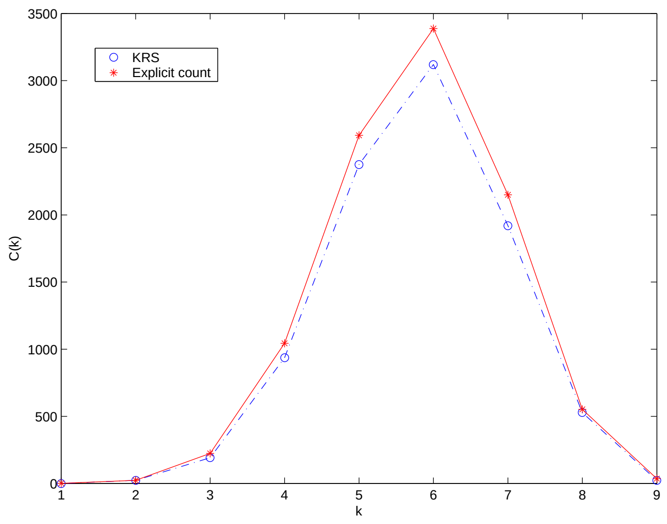


Figure 17.9. *Counts obtained by the KRS algorithm and by explicit counting for a 4×4 lattice. For KRS we set the probabilities to 0.5, the number of steps between records to $\ell = 4$, and the total number of steps to 10^5. Because ℓ was so small, the samples were highly correlated, but the estimates are still quite good.*

	$C(0)$	$C(1)$	$C(2)$	$C(3)$	$C(4)$	$C(5)$	$C(6)$	$C(7)$	$C(8)$
2×2	1	4	2						
2×3	1	7	11	3					
3×3	1	12	44	56	18				
4×4	1	24	224	1044	2593	3388	2150	552	36
6×6	1	60	1622	26172	281514	2135356	11785382	48145820	146702793

(b) One of the more interesting programming issues in this problem is the data structure.

- If we keep track of each edge of the lattice, then we need to enumerate rules for deciding whether two edges can be covered at the same time. For example, in our 2×2 lattice, we cannot simultaneously have a dimer on a vertical edge and one on a horizontal edge.

- If we keep track of each node of the lattice, then we need to know whether it is occupied by a dimer, so our first idea might be to represent a monomer by a zero and a dimer by a 1. But we need more information – whether its dimer partner is above, below, left, or right. Without this additional information, the array

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

tells us that the 2×2 lattice has two dimers on it, but we can't tell whether they are horizontal or vertical.

- A third alternative is to keep track of both edges and nodes. Think of it as a matching problem: each node can be matched with any of its four neighbors in a dimer, or it can be a monomer. We maintain an array of nodes, where the jth value is 0 if the node is a monomer, and equal to k, if (k, j) is a dimer. We store the edges in an $n^2 \times 4$ array, where the row index indicates the node at the beginning of the edge, and the nonzero entries in the row record the indices of the neighboring nodes. Thus, each physical edge has two entries in the array (in rows corresponding to its two nodes), and a few of the entries at the edges are 0, since some nodes have fewer than 4 edges. We can generate a KRS change by picking an edge from this array, and we update the node array after we decide whether an addition, deletion, or swap should be considered.

The program `KRS.m`, by Sungwoo Park, on the website, is an efficient implementation of the second alternative. Sample results are shown in Figure 17.9.

Please refer to the original paper [2] for information on how to set the parameters to KRS. Kenyon, Randall, and Sinclair showed that the algorithm samples well if both the number of steps and the interval between records are very large, but in practice the algorithm is considerably less sensitive than the analysis predicts.

[1] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statistical Science* 8(1):10-15, 1993.

[2] C. Kenyon, D. Randall, and A. Sinclair, "Approximating the number of monomer-dimer coverings of a lattice," *J. Stat. Phys.* 83(3-4):637-659, 1996.

Chapter 18

Solutions: Case Study: Multidimensional Integration: Partition and Conquer

CHALLENGE 18.1. A sample program is given on the website. Method 2 gives somewhat better results, since it averages the function values themselves rather than just using them to decide whether a point is inside or outside the region. Three digit accuracy is achieved for 100000 points in Method 1 and for 1000 and 100000 points for Method 2. The convergence rate for Method 1 is consistent with $1/\sqrt{n}$, since the product of the error with $\sqrt{n}$ is approximately constant for large n, but for Method 2, the results are somewhat more variable. MATLAB's function `quad` uses 13 function evaluations to get three digit accuracy.

Clearly, for low dimensional integration of smooth functions, Monte Carlo methods are not the methods of choice! Their value becomes apparent only when the dimension d is large so that methods like `quad` would be forced to use a lot of function evaluations.

CHALLENGE 18.2. See `challenge2.m` on the website.

CHALLENGE 18.3. A sample program is available on the website. Importance sampling produces better estimates at lower cost: see the answer to Challenge 4 for detailed results.

CHALLENGE 18.4. The results are shown in Figure 18.1. The pseudo-random points from MATLAB's `rand` are designed to have good statistical properties, but they leave large gaps in space. The quasi-random points are both more predictable and more evenly distributed. They tend to lie on diagonal lines, with longer strings as the coordinate number increases. Other algorithms for generating quasi-random points avoid this defect.

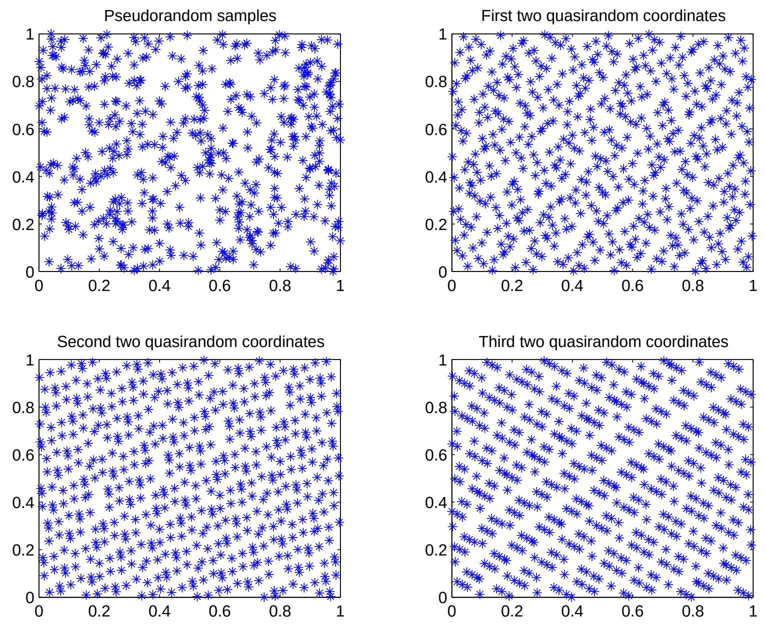


Figure 18.1. *500 pseudo-random points (upper left). 500 quasi-random points: coordinates 1-2 (upper right), coordinates 3-4 (lower left), coordinates 5-6 (lower right).*

CHALLENGE 18.5. The following table gives the absolute value of the errors in the estimates from each of the four methods.

n	Method 1	Method 2	Method 3	Method 4
10	3.65e-03	1.11e-02	4.67e-03	1.50e-02
100	1.35e-03	3.38e-03	1.02e-03	2.49e-03
1000	2.85e-03	2.38e-04	1.22e-05	3.00e-04
10000	1.57e-03	1.14e-03	1.75e-04	4.10e-05
100000	4.97e-04	1.72e-04	1.44e-05	5.14e-06

Convergence rates can be determined from the slope of a straight line fit to the logs of each set of errors.

The best results were obtained by Method 4, using quasi-random numbers in Method 2. Method 3, importance sampling, was also quite good.

CHALLENGE 18.6. No answer provided.

Chapter 19

Solutions: Case Study: Models of Infection: Person to Person

CHALLENGE 19.1. See the solution to Challenge 3.

CHALLENGE 19.2. See the solution to Challenge 3.

CHALLENGE 19.3. The results of a simulation of each of these three models are given in Figures 19.1-19.3. The MATLAB program that generated these results can be found on the website. In general, mobility increases the infection rate and vaccination decreases it dramatically. In our sample runs, the infection peaks around day 18 with no mobility, and around day 15 when patients are moved. Individual runs may vary.

CHALLENGE 19.4. The histograms for $\nu = 0$, 0.1, 0.2, and 0.3 are shown in Figure 19.4. The mean percent of the population infected drops from 73.6% for $\nu = 0$ (with a variance of 4.5%), to 4.1% for $\nu = 0.3$ (with a variance of only 0.06%).

CHALLENGE 19.5. From Challenge 4, we know that a very low vaccination rate is sufficient to dramatically reduce the infection rate: somewhat less than $\nu = 0.1$. But using a nonlinear equation solver on a noisy function is quite dangerous; it is easily fooled by outliers, and by changing the starting guess, you can make it produce almost any value.

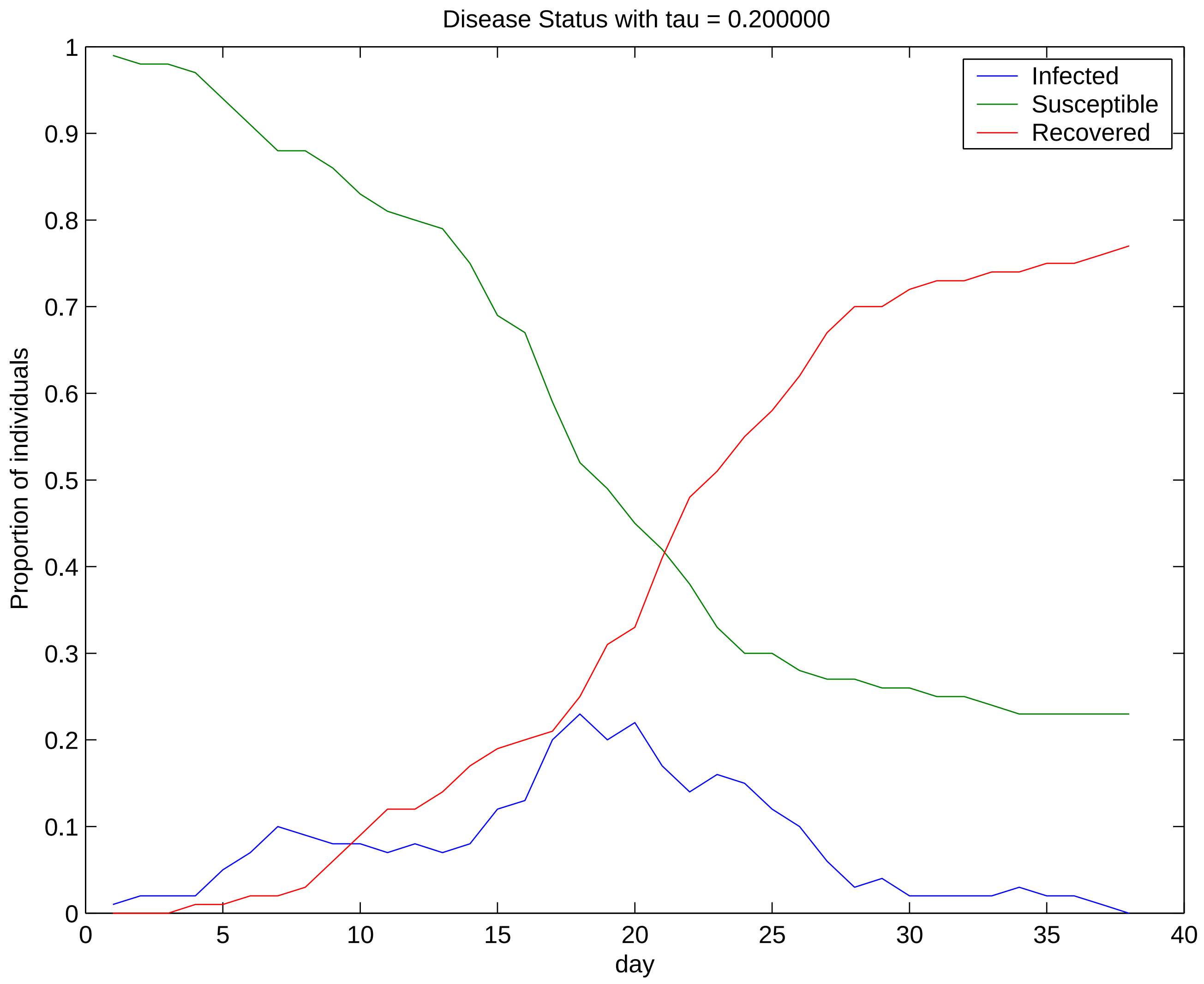


Figure 19.1. *Proportion of individuals infected by day in a* 10×10 *grid of hospital beds, with infection rate* $\tau = 0.2$.

CHALLENGE 19.6.

(a) The transition probabilities are given in Figure 19.5, and the matrix is given in the MATLAB program found on the website.

(b) $\boldsymbol{A}\boldsymbol{e}_1$ is equal to column 1 of $\boldsymbol{A}$, which contains the probabilities of transitioning from state 1 to any of the other states. More generally, if $\boldsymbol{p}$ is a vector of probabilities of initially being in each of the states, then $\boldsymbol{A}\boldsymbol{p}$ is the vector of probabilities of being

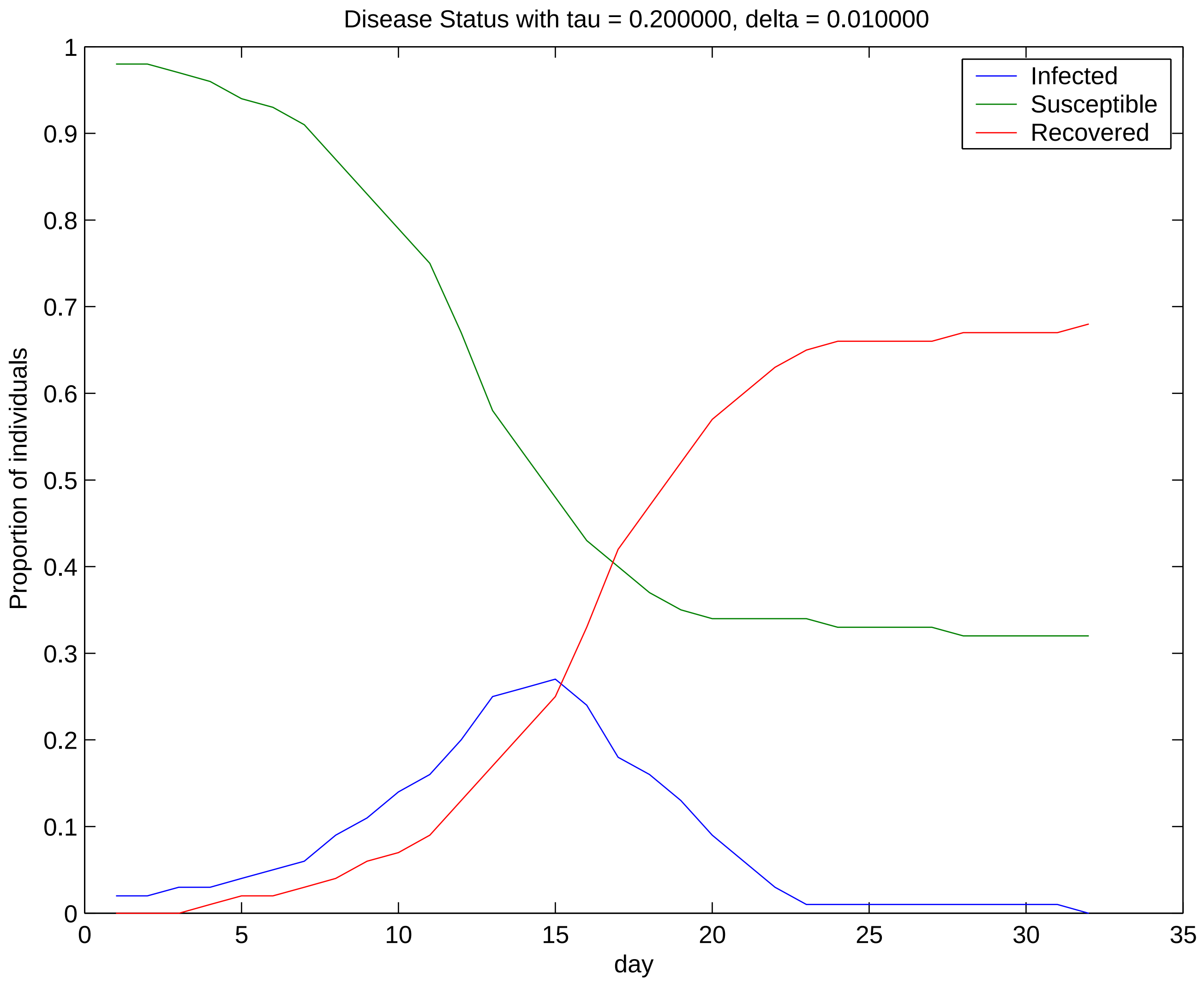


Figure 19.2. *Proportion of individuals infected by day in a* 10×10 *grid of hospital beds, with infection rate* $\tau = 0.2$ *and mobility rate* $\delta = 0.01$.

in them at time 1.

(c) If $\boldsymbol{A}$ is a dense matrix, then computing $\boldsymbol{A}(\boldsymbol{A}\boldsymbol{e}_1)$ costs $2s^2$ multiplications, where s is the number of states. Computing $(\boldsymbol{A}^2)\boldsymbol{e}_1$ costs $s^3 + s^2$ multiplications, and this is quite a bit more when s is large. (We should also take advantage of the zeros in $\boldsymbol{A}$ and avoid multiplying by them. If we do this for our matrix, $\boldsymbol{A}$ has 21 nonzero elements while $\boldsymbol{A}^2$ has 23, so again it takes more multiplications to form $(\boldsymbol{A}^2)\boldsymbol{e}_1$ than to form $\boldsymbol{A}(\boldsymbol{A}\boldsymbol{e}_1)$. We should also note that the product $\boldsymbol{A}\boldsymbol{e}_1$ is just the first

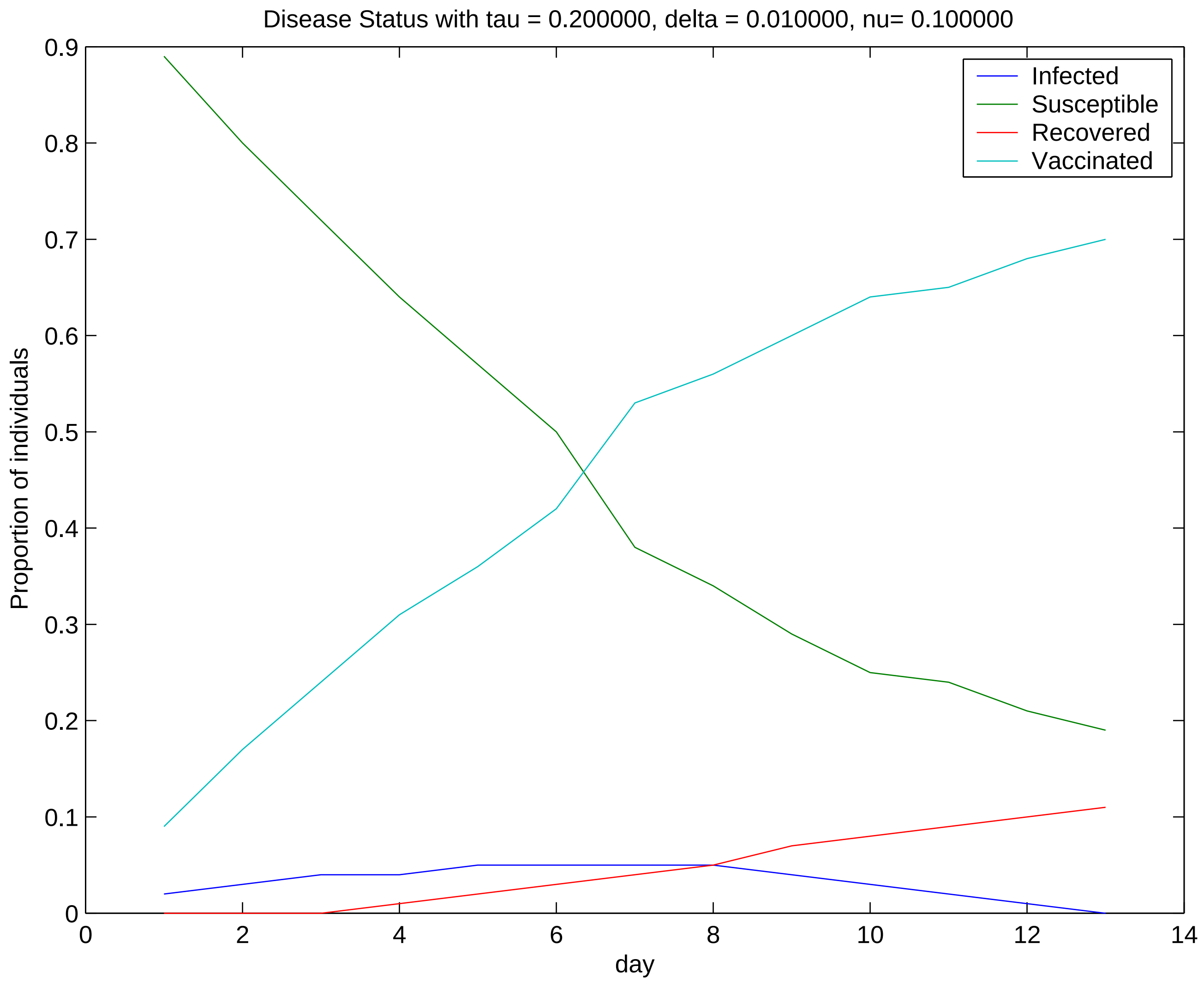


Figure 19.3. *Proportion of individuals infected by day in a* 10×10 *grid of hospital beds, with infection rate* $\tau = 0.2$*, mobility rate* $\delta = 0.01$*, and vaccination rate* $\nu = 0.1$.

column of $\boldsymbol{A}$, so it could be computed without multiplications.)

(d) In this experiment, it took 280 Monte Carlo simulations to get 2 digits of accuracy. Asking for 3 digits raises the number of trials into the ten thousands, since the variance is high relative to this threshold.

(e) There is only one path to state Q, corresponding to a single infection, and the product of the probabilities of transitions along this path are $(1-\tau)^4$. There are

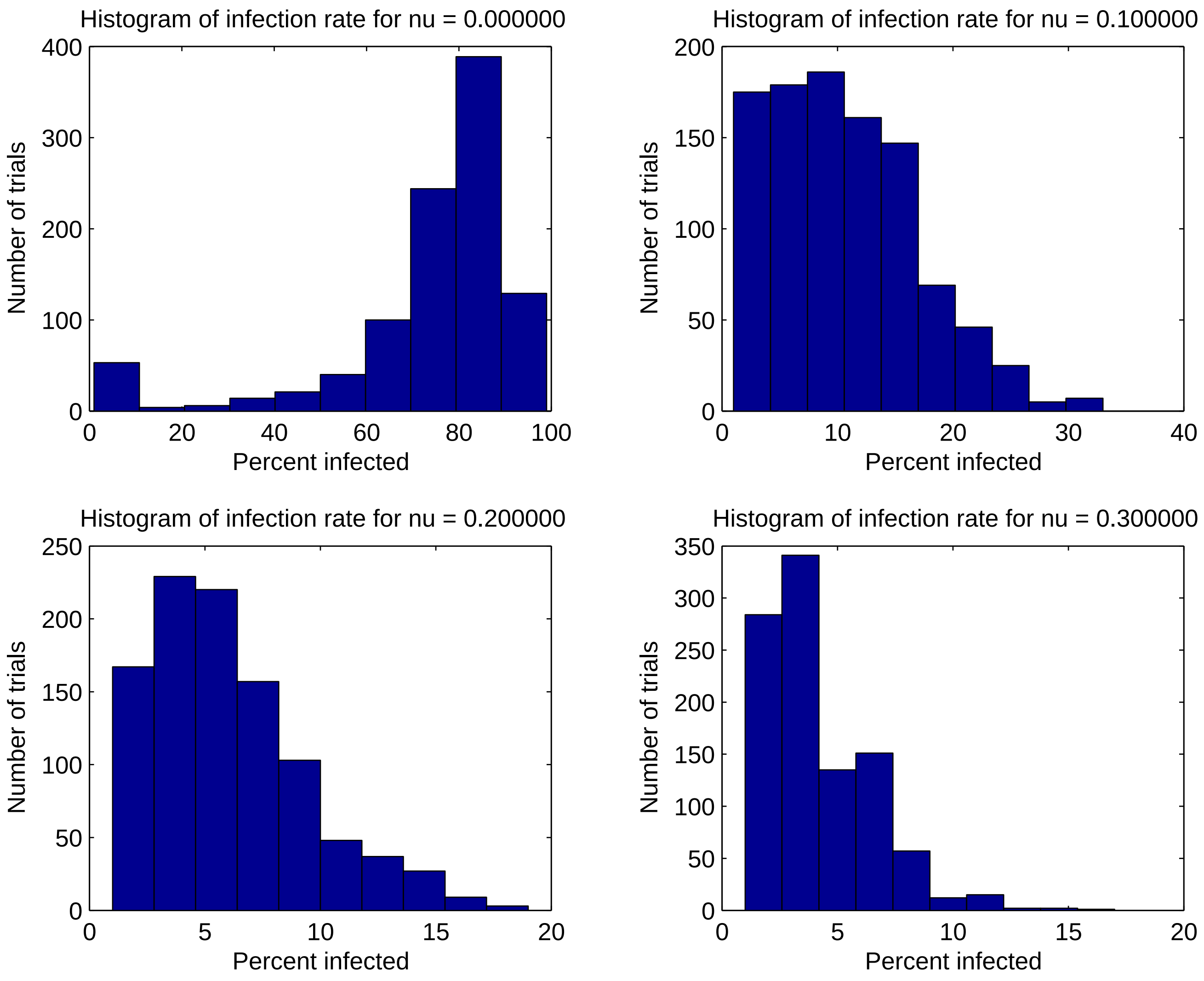


Figure 19.4. *Results of* 1000 *trials for a* 10×10 *grid of hospital beds, with infection rate* $\tau = 0.2$ *and vaccination rate* ν*, with* ν *varying.*

2 paths to state S, and summing the product of the probabilities along the paths gives $(\tau(1-\tau)^2 + \tau(1-\tau)^3)$. The probability of reaching state P is the same, so the probability of 2 infections is twice this number. Similarly, the probability of reaching state R, corresponding to 3 infections, is $\tau^2 + 2\tau^2(1-\tau) + (1-\tau)^2\tau^2$. The probabilities of reaching states P, Q, R, and S sum to one, since these are the only possible outcomes.

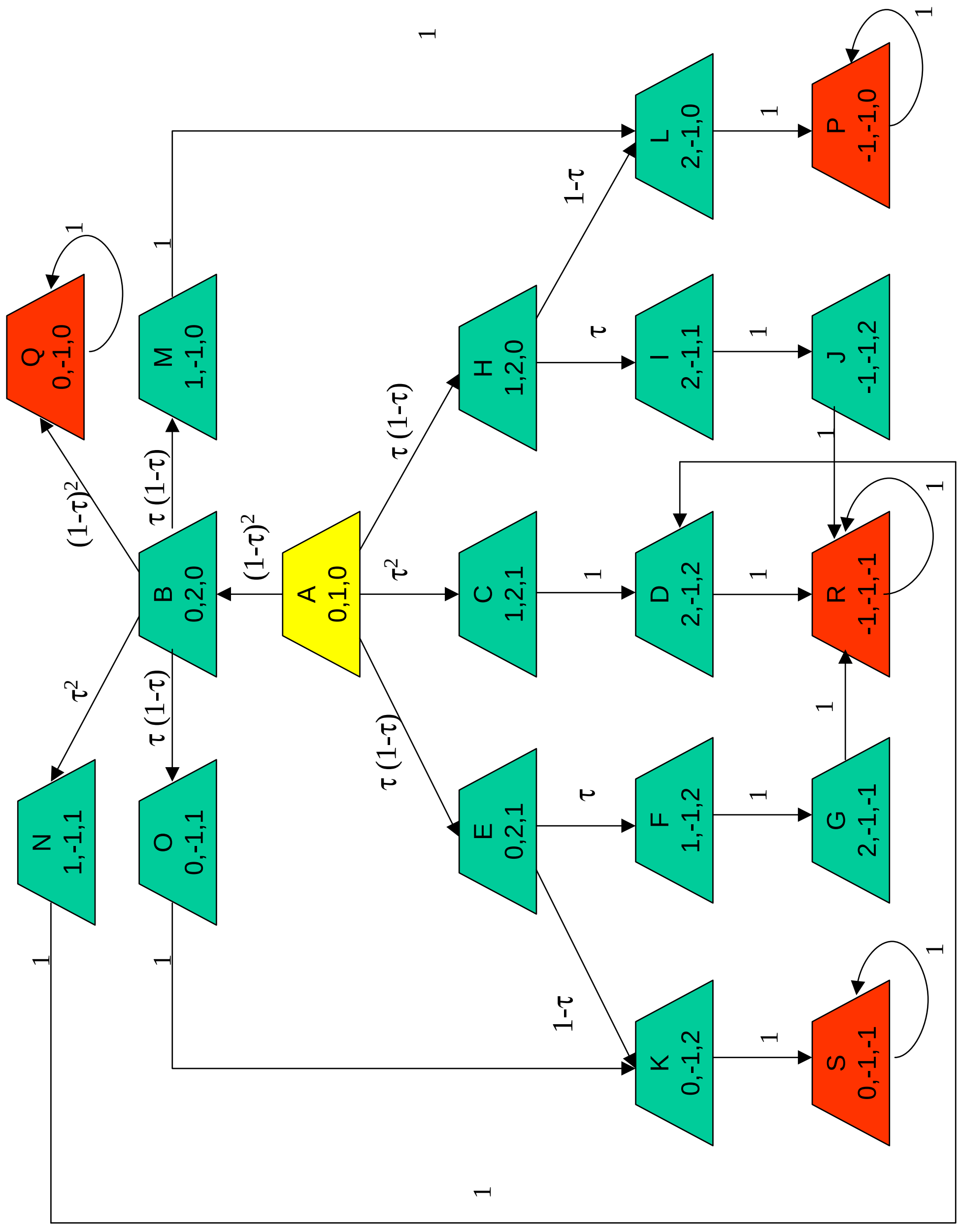


Figure 19.5. *Transition probabilities for the Markov chain model of the epidemic.*

CHALLENGE 19.7.

- The probabilities are clearly nonnegative and sum to 1.
- Note that the jth component of $\boldsymbol{e}^T \boldsymbol{A}$ is the sum of the elements in column j, and this is 1, so $\boldsymbol{e}^T \boldsymbol{A} = \boldsymbol{e}^T$.
- Therefore, $\boldsymbol{e}^T \boldsymbol{A} = 1\boldsymbol{e}^T$, and this means that the vector $\boldsymbol{e}^T$ is unchanged in direction when multiplied on the right by $\boldsymbol{A}$. This is the definition of a left eigenvector of $\boldsymbol{A}$, and the eigenvalue is 1.
- Apply the Gerschgorin circle theorem to $\boldsymbol{A}^T$, which has the same eigenvalues as $\boldsymbol{A}$. If the main diagonal element of $\boldsymbol{A}^T$ is $0 < \alpha < 1$, then the off-diagonal elements are nonnegative and sum to $1-\alpha$. Therefore, the Gerschgorin circle is centered at α with radius $1-\alpha$. This circle touches the unit circle at the point $(1, 0)$ but lies inside of it. The eigenvalues lie in the union of the Gerschgorin circles, so all eigenvalues lie inside the unit circle.
- If $\boldsymbol{A}$ were irreducible then the eigenvalue at 1 would be simple; see, for example, [1].
- Let the eigensystem of $\boldsymbol{A}$ be defined by $\boldsymbol{A}\boldsymbol{u}_j = \lambda_j \boldsymbol{u}_j$, and let

  $$\boldsymbol{e}_1 = \sum_{j=1}^{n} \alpha_j \boldsymbol{u}_j,$$

  where $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_4$ are a basis for the eigenspace corresponding to the eigenvalue 1. Then verify that

  $$\boldsymbol{A}^k \boldsymbol{e}_1 = \sum_{j=1}^{n} \alpha_j \lambda_j^k \boldsymbol{u}_j.$$

  Since $\lambda_j^k \to 0$ as $k \to \infty$ except for the eigenvalue 1, we see that

  $$\boldsymbol{A}^k \boldsymbol{e}_1 \to \alpha_1 \boldsymbol{u}_1 + \alpha_2 \boldsymbol{u}_2 + \alpha_3 \boldsymbol{u}_3 + \alpha_4 \boldsymbol{u}_4.$$

- Therefore, we converge to a multiple of the stationary vector.

[1] Richard Varga, *Matrix Iterative Analysis*, Prentice Hall, Englewood Cliffs, NJ, 1962.

Unit V

SOLUTIONS: Solution of Differential Equations

Chapter 20

Solutions: Solution of Ordinary Differential Equations

CHALLENGE 20.1. (Partial Solution) See the programs on the website.

CHALLENGE 20.2. We need the real parts of all eigenvalues to be negative. This means $4 - t^2 < 0$ and $-t < 0$, so the equation is stable when $t > 2$.

CHALLENGE 20.3. The polynomial is

$$p(t) = y_n + (t - t_n) f_n,$$

so we compute

$$p(t_{n+1}) = y_n + (t_{n+1} - t_n) f_n.$$

CHALLENGE 20.4. The true solution is $y(t) = t$. We compute:

t_n	Euler approximation
0	0
0.1	$0 + 1 * .1 = 0.1$
0.2	$.1 + 1 * .1 = 0.2$
...	...
1.0	$.9 + 1 * .1 = 1.0$

Euler's method is exact for this problem.

CHALLENGE 20.5. Since $f(t, y) = -y$, the backward Euler formula is

$$y_{n+1} = y_n + h_n f(t_{n+1}, y_{n+1}) = y_n - h_n y_{n+1}.$$

Therefore,

$$(1 + h_n)y_{n+1} = y_n,$$

so

$$y_{n+1} = \frac{1}{1 + h_n} y_n.$$

We compute:

t_n	y_n	$y(t_n)$
0	1	1
0.1	$1/1.1 = 0.9091$	0.9048
0.2	$(1/1.1)^2 = 0.8264$	0.8187
0.3	$(1/1.1)^3 = 0.7513$	0.7408

CHALLENGE 20.6. Rearranging, we get

$$(1 + ha/2)y_{n+1} = (1 - ha/2)y_n,$$

so

$$y_{n+1} = \frac{1 - ha/2}{1 + ha/2} y_n.$$

Apply Taylor series expansion to the differential equation to get

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + hy'(t_n) + \frac{h^2}{2} y''(\xi) \\ &= y(t_n) - hay(t_n) + \frac{h^2}{2} y''(\xi) \\ &= (1 - ha)y(t_n) + \frac{h^2}{2} y''(\xi), \end{aligned}$$

where ξ is a point between t_n and t_{n+1}. Let $e_n = y_n - y(t_n)$, and subtract our two expressions to obtain

$$e_{n+1} = \frac{1 - ha/2}{1 + ha/2} e_n - (1 - ha - \frac{1 - ha/2}{1 + ha/2}) y(t) - \frac{h^2}{2} y''(\xi)$$

Now, since

$$(1 - ha - \frac{1 - ha/2}{1 + ha/2}) y(t) = -\frac{h^2 a^2}{2 + ha} y(t) = -\frac{h^2}{2 + ha} y''(t),$$

we see that

$$e_{n+1} = \frac{1 - ha/2}{1 + ha/2} e_n + \frac{h^2}{2 + ha} y''(t) + \frac{h^2}{2} y''(\xi).$$

The last two terms can be combined and represent an effective local error. Therefore, the global error is magnified if $|(1-ha/2)/(1+ha/2)| > 1$. Conversely, the method is stable when

$$\left|\frac{1-ha/2}{1+ha/2}\right| < 1,$$

which holds for all $h > 0$.

CHALLENGE 20.7. Recall that Euler's method is

$$y_{n+1} = y_n + hf(t_n, y_n),$$

and backward Euler is

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}).$$

$$\begin{aligned}
P &: y = 1 + .1(1^2) = 1.1,\\
E &: f = (1.1)^2 - .5 = 0.71,\\
C &: y = 1 + .1 * .71 = 1.071,\\
E &: f = (1.071)^2 - 0.5.
\end{aligned}$$

The predicted value is quite close to the corrected value; this is an indication that the stepsize is small enough to obtain some accuracy in the computed solution.

CHALLENGE 20.8. $f(t,y) = 10y^2 - 20$.
P: $y^P = y(0) + .1f(0, y(0)) = 1 + .1(-10) = 0$.
E: $f^P = f(.1, y^P) = 10 * 0 - 20 = -20$.
C: $y^C = y(0) + .1f^P = 1 - 2 = -1$.
E: $f^C = f(.1, y^C) = 10 - 20 = -10$.

Note that the predicted and corrected values are quite different, so neither can be trusted; we should reduce the stepsize and recompute. The true value is $y(.1) \approx -0.69$.

CHALLENGE 20.9. Suppose $\widehat{\boldsymbol{y}}$ is the result of the predictor and $\tilde{\boldsymbol{y}}$ is the result of the corrector. Assuming $\tilde{\boldsymbol{y}}$ is much more accurate than $\widehat{\boldsymbol{y}}$,

$$\|\widehat{\boldsymbol{y}} - \boldsymbol{y}_{true}\| \approx \|\widehat{\boldsymbol{y}} - \tilde{\boldsymbol{y}}\| \equiv \delta$$

If $\delta > \tau$, reduce h and retake the step:

- perhaps $h = h/2$.
- perhaps $h = h/2^p$ where, since we need $\delta 2^{-5p} \approx \tau$, we define $p = (\log\delta - \log\tau)/(5\log 2)$.

CHALLENGE 20.10. We know that if our old values are correct,

$$y_{n+1}^P - y(t_{n+1}) = \frac{3h^4}{8} y^{(4)}(\eta).$$
$$y_{n+1}^C - y(t_{n+1}) = -\frac{h^4}{24} y^{(4)}(\nu).$$

Subtracting, we obtain

$$y_{n+1}^P - y_{n+1}^C = \frac{3h^4}{8} y^{(4)}(\eta) - (-\frac{h^4}{24} y^{(4)}(\nu)),$$

where η, ν are in the interval containing y_{n+1}^P, y_{n+1}^C, and the true value. Since $3/8 + 1/24 = 10/24$, the error in the corrector can be estimated as $\epsilon = |y_{n+1}^P - y_{n+1}^C|/10$. Now, if $\epsilon > \tau$, we might reduce h by a factor of 2 and retake the step. If $\epsilon << \tau$, we might double h in preparation for the next step (expecting that the local error might increase by a factor of 2^4).

CHALLENGE 20.11. No answer provided.

CHALLENGE 20.12.

$$\begin{aligned}
\boldsymbol{y}'(t) &= \boldsymbol{D}\nabla_{\boldsymbol{y}} H(\boldsymbol{y})(t) \\
&= \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} y_1(t) + y_1(t)\, y_2^2(t)\, y_3^2(t)\, y_4^2(t) \\ y_2(t) + y_1^2(t)\, y_2(t)\, y_3^2(t)\, y_4^2(t) \\ y_3(t) + y_1^2(t)\, y_2^2(t)\, y_3(t)\, y_4^2(t) \\ y_4(t) + y_1^2(t)\, y_2^2(t)\, y_3^2(t)\, y_4(t) \end{bmatrix} \\
&= \begin{bmatrix} y_2(t) + y_1^2(t)\, y_2(t)\, y_3^2(t)\, y_4^2(t) \\ -(y_1(t) + y_1(t)\, y_2^2(t)\, y_3^2(t)\, y_4^2(t)) \\ y_4(t) + y_1^2(t)\, y_2^2(t)\, y_3^2(t)\, y_4(t) \\ -(y_3(t) + y_1^2(t)\, y_2^2(t)\, y_3(t)\, y_4^2(t)) \end{bmatrix}.
\end{aligned}$$

CHALLENGE 20.13.

$$\boldsymbol{y}(t) = \begin{bmatrix} u(t) \\ v(t) \\ w(t) \end{bmatrix}, \boldsymbol{M}(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \ \boldsymbol{A}(t) = \begin{bmatrix} 7 & -6 & 0 \\ 4 & -2 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \ \boldsymbol{f}(t) = \begin{bmatrix} 4t \\ 0 \\ -24 \end{bmatrix}.$$

CHALLENGE 20.14. We calculate

$$\boldsymbol{y}(t) = \begin{bmatrix} \boldsymbol{u}(t) \\ \boldsymbol{p}(t) \end{bmatrix}, \ \boldsymbol{M} = \begin{bmatrix} \boldsymbol{I}_{n_u} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}, \ \boldsymbol{A} = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{B} \\ \boldsymbol{B}^T & \boldsymbol{0} \end{bmatrix}, \ \boldsymbol{f}(t) = \boldsymbol{0}.$$

Therefore,

$$\boldsymbol{P}_1 = \begin{bmatrix} I_{n_u} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ -\boldsymbol{A} & -\boldsymbol{B} & \boldsymbol{I}_{n_u} & \boldsymbol{0} \\ -\boldsymbol{B}^T & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix},$$

$$\boldsymbol{N}_1 = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{B} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{B}^T & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix},$$

so $\operatorname{rank}(\boldsymbol{P}_1) = 2(n_u + n_p) - 2n_p$. Therefore we take $n_a = 2n_p$. We need a basis for the nullspace of $\boldsymbol{P}_1^T$, and we can take, for example,

$$\boldsymbol{Z}^T = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I}_{n_p} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{B}^T & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}_{n_p} \end{bmatrix}.$$

Now we calculate

$$\widehat{\boldsymbol{N}}_1 = \begin{bmatrix} \boldsymbol{B}^T & \boldsymbol{0} \\ \boldsymbol{B}^T\boldsymbol{C} & \boldsymbol{B}^T\boldsymbol{B} \end{bmatrix},$$

so we can take, for example,

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{X} \\ -(\boldsymbol{B}^T\boldsymbol{B})^{-1}\boldsymbol{B}^T\boldsymbol{C}\boldsymbol{X} \end{bmatrix},$$

which has $n_d = n_u - n_p$ columns. Then

$$\boldsymbol{M}\boldsymbol{T} = \begin{bmatrix} \boldsymbol{I}_{n_u} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{X} \\ -(\boldsymbol{B}^T\boldsymbol{B})^{-1}\boldsymbol{B}^T\boldsymbol{C}\boldsymbol{X} \end{bmatrix} = \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{0} \end{bmatrix},$$

so we can take

$$\boldsymbol{W}^T = [\ \boldsymbol{X}^T, \ \ \boldsymbol{0}\ ],$$

which makes $\boldsymbol{W}^T\boldsymbol{M}\boldsymbol{T} = \boldsymbol{X}^T\boldsymbol{X}$ which has rank $n_d = n_u - n_p$, as desired.

All the matrices are constant with respect to time, so the differentiability assumptions are satisfied.

CHALLENGE 20.15.

- Using the notation of the pointer, we let $a(t) = 1 > 0$, $b(t) = 8.125\pi \cot((1 + t)\pi/8)$, $c(t) = \pi^2 > 0$, and $f(t) = -3\pi^2$. These are all smooth functions on $[0, 1]$.
- Since $c(t) = \pi^2/2 > 0$ and
$$\int_0^1 [f(t)]^2 \mathrm{d}t = \frac{\pi^4}{4},$$
the solution exists and is unique.
- Since $f(t) < 0$, the Maximum Principle tells us that
$$\max_{t\in[0,1]} u(t) \leq \max(-2.0761, -2.2929, 0) = 0.$$
- Letting $v(t) = -3$, we see that
$$-v''(t) + 8.125\pi \cot((1 + t)\pi/8)v'(t) + \pi^2 v(t) = -3\pi^2$$
and $v(0) = v(1) = -3$. Therefore the Monotonicity Theorem says that $u(t) \geq v(t)$ for $t \in [0, 1]$.
- Therefore we conclude $-3 \leq u(t) \leq 0$ for $t \in [0, 1]$.

Note on how I constructed the problem: The true solution to the problem is $u(t) = \cos((1 + t)\pi/8) - 3$, which does indeed have the properties we proved about it. But we can obtain a lot of information about the solution (as illustrated in this problem) without ever evaluating it!

CHALLENGE 20.16. Let $y_{(1)}(t) = a(t)$, $y_{(2)}(t) = a'(t)$. Then our system is

$$\boldsymbol{y}'(t) = \begin{bmatrix} y_{(2)}(t) \\ y_{(1)}^2(t) - 5y_{(2)}(t) \end{bmatrix}.$$

```
function [t,y,z] = solvebvp()

z = fzero(@fvalue,2);
% {\tt now the solution can be obtained by using ode45 with
% {\tt initial conditions [5,z].}}\STATE{\% {\tt For example,
[t,y] = ode45(@yprime,[0:.1:1],[5 z]);

% End of solvebvp
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function yp = yprime(t,y)
yp = [y(2); y(1)^2 - 5 * y(2)];

% End of yprime

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function f = fvalue(z)

[t,y] = ode45(@yprime,[0 1],[5,z]);
f = y(end,1)-2;

% End of fvalue

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CHALLENGE 20.17.

```
function [t,y,z] = solvebvp()

z = fzero(@evalshoot,[-1,1]);

[t,y] = ode45(@yprime,[0,1],[1,z]);

% The true solution is ...

utrue = cos(pi*t/2) + t.^2;

plot(t,y(:,1),t,utrue)
legend('Computed solution','True solution')
xlabel('t')
ylabel('u')

% end of solvebvp

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function f = evalshoot(z)

% Given a value for y(2) at time t=0, see how close
% y(2) is to b_desired at t=1.

b_desired = 1;
[t,y] = ode45(@yprime,[0,1],[1,z]);
f = y(end,1)-b_desired;

% end of evalshoot

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function yp = yprime(t,y)

yp = [ y(2);   -(pi/2)^2*y(1)+(pi/2)^2*t.^2 + 2];

% end of yprime

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CHALLENGE 20.18. No answer provided.

CHALLENGE 20.19. (Partial Solution.) We use Taylor series expansions to derive the second formula:

$$u(t+h) = u(t) + hu'(t) + \frac{h^2}{2}u''(t) + \frac{h^3}{6}u'''(t) + \frac{h^4}{24}u''''(\eta_1),\ \eta_1 \in [t, t+h],$$
$$u(t-h) = u(t) - hu'(t) + \frac{h^2}{2}u''(t) - \frac{h^3}{6}u'''(t) + \frac{h^4}{24}u''''(\eta_2),\ \eta_2 \in [t-h, t].$$

Adding, we obtain

$$u(t+h) + u(t-h) = 2u(t) + h^2u''(t) + \frac{h^4}{24}\left[u''''(\eta_1) + u''''(\eta_2)\right].$$

Using the Mean Value Theorm on the last term and solving for $u''(t)$ gives

$$\Rightarrow u''(t) = \frac{u(t-h) - 2u(t) + u(t+h)}{h^2} \underbrace{-\frac{h^4}{h^2 \cdot 24}2u''''(\hat{\eta})}_{O(h^2)},\ \hat{\eta} \in [t-h, t+h].$$

CHALLENGE 20.20. Let u_j approximate $u(jh)$. Then

$$\begin{aligned} u_0 &= 2 \\ \frac{u_{j-1} - 2u_j + u_{j+1}}{h^2} &= \frac{u_{j+1} - u_{j-1}}{2h} + 6u_j \\ u_5 &= 3 \end{aligned}$$

where $j = 1, 2, 3, 4$.

CHALLENGE 20.21. For $j = 1, \ldots, 99$,

$$\boldsymbol{F}_j(u) = \frac{u_{j-1} - 2u_j + u_{j+1}}{h^2} - \frac{u_{j+1} - u_{j-1}}{2h} + jhu_j - e^{u_j} \,,$$

where $u_0 = 1$ and $u_{100} = 0$.

Chapter 21

Solutions: Case Study: More Models of Infection: It's Epidemic

CHALLENGE 21.1. Sample programs are given on the website. The results are shown in Figure 21.1. 95.3% of the population becomes infected.

CHALLENGE 21.2. The results are shown in Figure 21.1 and, as expected, are indistinguishable from those of Model 1.

CHALLENGE 21.3. The results are shown in Figure 21.2. 94.3% of the population becomes infected, slightly less than in the first models, and the epidemic dies out in roughly half the time.

CHALLENGE 21.4. Let's use subscripts x to denote partial derivatives with respect to x, so that $I_{xx}(t,x,y) = \partial^2 I(t,x,y)/\partial x^2$.

(a) Since Taylor series expansion yields

$$I(t)_{i-1,j} = I(t,x,y) - hI_x(t,x,y) + \frac{h^2}{2}I_{xx}(t,x,y) - \frac{h^3}{6}I_{xxx}(t,x,y) + O(h^4)\,,$$
$$I(t)_{i+1,j} = I(t,x,y) + hI_x(t,x,y) + \frac{h^2}{2}I_{xx}(t,x,y) + \frac{h^3}{6}I_{xxx}(t,x,y) + O(h^4)\,,$$

we see that

$$\frac{I(t)_{i-1,j} - 2I(t)_{ij} + I(t)_{i+1,j}}{h^2} = \frac{h^2 I_{xx}(t,x,y) + O(h^4)}{h^2} = I_{xx}(t,x,y) + O(h^2)\,.$$

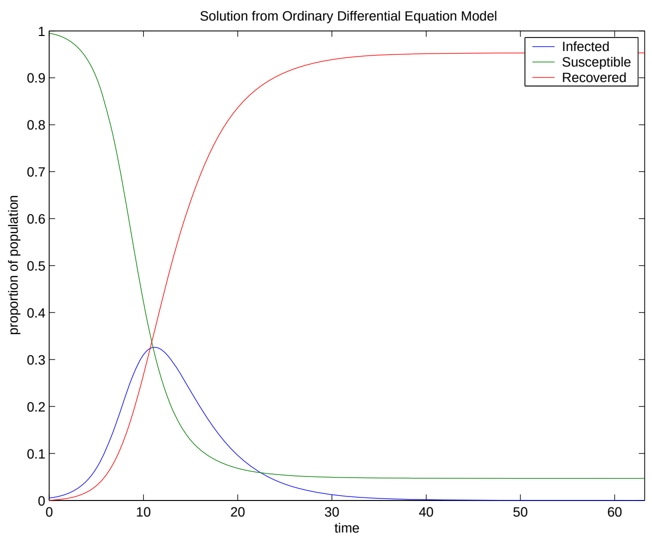


Figure 21.1. *Proportion of individuals infected by the epidemic from the* ODE *Model 1 or the* DAE *Model 2.*

(b) The matrix $\boldsymbol{A}$ can be expressed as

$$\boldsymbol{A} = \boldsymbol{T} \otimes \boldsymbol{I} + \boldsymbol{I} \otimes \boldsymbol{T},$$

where

$$\boldsymbol{T} = \frac{1}{h^2} \begin{bmatrix} -2 & 2 & & & & \\ 1 & -2 & 1 & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & 1 & -2 & 1 \\ & & & & 2 & -2 \end{bmatrix},$$

and $\boldsymbol{T}$ and the identity matrix $\boldsymbol{I}$ are matrices of dimension $n \times n$. (The notation $\boldsymbol{C} \otimes \boldsymbol{D}$ denotes the matrix whose (i, j)-th block is $c_{ij}\boldsymbol{D}$. The MATLAB command to form this matrix is `kron(C,D)`, which means Kronecker product of $\boldsymbol{C}$ and $\boldsymbol{D}$. See Chapter 6.)

CHALLENGE 21.5. The results of (a) are given in Figure 21.3, and those for (b) are given in Figure 21.4. The infection rate without vaccination is 95.3% (very similar to Model 1) while with vaccination it drops to 38.9%. Vaccination also significantly shortens the duration of the epidemic.

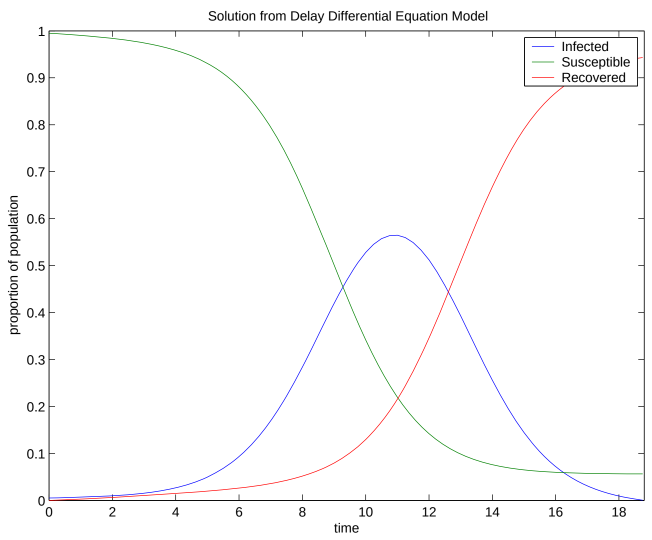


Figure 21.2. *Proportion of individuals infected by the epidemic from the* DDE *Model 3.*

CHALLENGE 21.6. No answer provided.

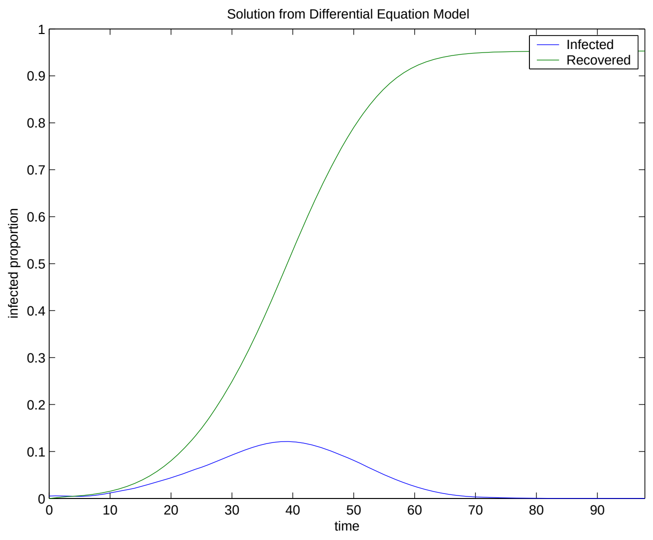


Figure 21.3. *Proportion of individuals infected by the epidemic from the differential equation of Model 5a.*

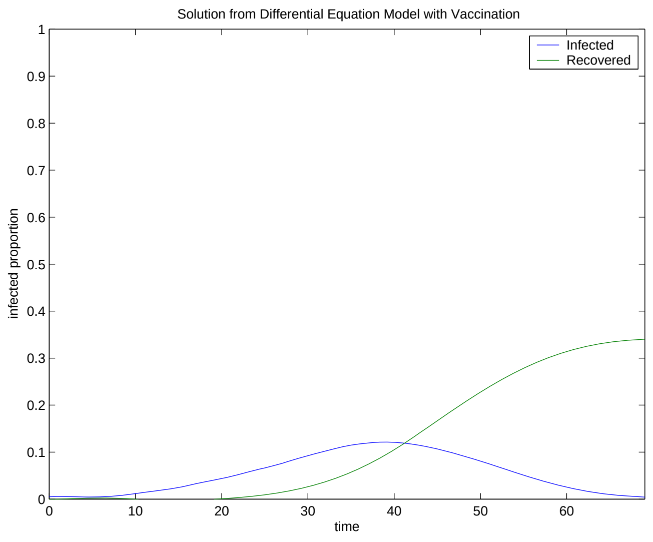


Figure 21.4. *Proportion of individuals infected by the epidemic from the differential equation of Model 5b, including vaccinations.*

Chapter 22

Solutions: Case Study: Robot Control: Swinging Like a Pendulum

(coauthored by Yalin E. Sagduyu)

CHALLENGE 22.1. Under the transformation, equation (1) becomes

$$\begin{bmatrix} 1 & 0 \\ c & m\ell \end{bmatrix} \begin{bmatrix} y'_{(1)}(t) \\ y'_{(2)}(t) \end{bmatrix} = \begin{bmatrix} y_{(2)}(t) \\ -mg\sin(y_{(1)}(t)) \end{bmatrix},$$

or

$$\begin{bmatrix} y'_{(1)}(t) \\ y'_{(2)}(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -c/(m\ell) & 1/(m\ell) \end{bmatrix} \begin{bmatrix} y_{(2)}(t) \\ -mg\sin(y_{(1)}(t)) \end{bmatrix}.$$

Replacing $\sin(y_{(1)}(t))$ by $y_{(1)}(t)$ gives the system

$$\boldsymbol{y}' = \begin{bmatrix} y'_{(1)}(t) \\ y'_{(2)}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -g/\ell & -c/(m\ell) \end{bmatrix} \begin{bmatrix} y_{(1)}(t) \\ y_{(2)}(t) \end{bmatrix} = \boldsymbol{A}\boldsymbol{y}.$$

The eigenvalues of the matrix $\boldsymbol{A}$ are the roots of $\det(\boldsymbol{A} - \lambda\boldsymbol{I}) = 0$, or the roots of $\lambda^2 + \lambda c/(m\ell) + g/\ell = 0$, and these are

$$\lambda_{1,2} = -\frac{c}{2m\ell} \pm \sqrt{\frac{c^2}{4m^2\ell^2} - \frac{g}{\ell}}.$$

For the undamped case, $c = 0$, so the real part of each eigenvalue is zero and the system is unstable. The real part of each eigenvalue is negative if $c > 0$, so in the damped case, the system is stable.

If $\lambda_1 \neq \lambda_2$, the eigenvectors of the matrix are

$$\begin{bmatrix} 1 \\ \lambda_1 \end{bmatrix}, \begin{bmatrix} 1 \\ \lambda_2 \end{bmatrix},$$

so the solution to the differential equation is

$$\boldsymbol{y}(t) = \alpha_1 \begin{bmatrix} 1 \\ \lambda_1 \end{bmatrix} e^{\lambda_1 t} + \alpha_2 \begin{bmatrix} 1 \\ \lambda_2 \end{bmatrix} e^{\lambda_2 t},$$

where α_1 and α_2 are constants determined by two additional conditions. If the discriminant satisfies $c^2/(4m^2\ell^2) - g/\ell > 0$, then the solution decays; otherwise it can have an oscillatory component in addition to a decaying one.

CHALLENGE 22.2. We note that $v(0,0) = 0$, and it is easy to see that $v > 0$ for all other values of its arguments.

We differentiate:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} v(\boldsymbol{y}(t)) &= \frac{g \sin\theta(t)}{\ell} \frac{\mathrm{d}\theta(t)}{\mathrm{d}t} + \frac{\mathrm{d}\theta(t)}{\mathrm{d}t} \frac{\mathrm{d}^2\theta(t)}{\mathrm{d}t^2} \\ &= \frac{g \sin\theta(t)}{\ell} \frac{\mathrm{d}\theta(t)}{\mathrm{d}t} - \frac{\mathrm{d}\theta(t)}{\mathrm{d}t} \frac{1}{m\ell} (c \frac{\mathrm{d}\theta(t)}{\mathrm{d}t} + mg \sin(\theta(t))) \\ &= \frac{-c}{m\ell} \left(\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} \right)^2 \le 0\,. \end{aligned}$$

Therefore, we can now conclude that the point $\theta = 0, \mathrm{d}\theta/\mathrm{d}t = 0$ is stable for both the damped $(c > 0)$ and undamped $(c = 0)$ cases. For the undamped case, $\mathrm{d}v(\boldsymbol{y}(t))/\mathrm{d}t$ is identically zero, and we cannot conclude that we have asymptotic stability. For the damped case, we note that the set defined by $\mathrm{d}v(\boldsymbol{y}(t))/\mathrm{d}t = 0$ contains all points $(\theta, \mathrm{d}\theta/\mathrm{d}t = 0)$, and the only invariant set is the one containing the single point $(0,0)$ so this point is asymptotically stable.

CHALLENGE 22.3. From Challenge 1, we see that

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ -g/\ell & -c/(m\ell) \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 0 \\ 1/(m\ell) \end{bmatrix}.$$

Our dimensions are $n = 2$, $m = 1$, so the controllability matrix is

$$\begin{bmatrix} \boldsymbol{B} & \boldsymbol{AB} \end{bmatrix} = \begin{bmatrix} 0 & 1/(m\ell) \\ 1/(m\ell) & -c/(m\ell)^2 \end{bmatrix}.$$

This matrix has rank 2, independent of c, so the system is controllable.

CHALLENGE 22.4. See the program `problem4.m` on the website. The results are shown in Figures 22.1 and 22.2. The models for the undamped undriven pendulum quickly show a phase difference in their results, while the damped undriven pendulum results are quite similar. For the driven pendulum, the linear and nonlinear results differ more as the angle θ_f gets bigger, and the linear models do not converge to θ_f.

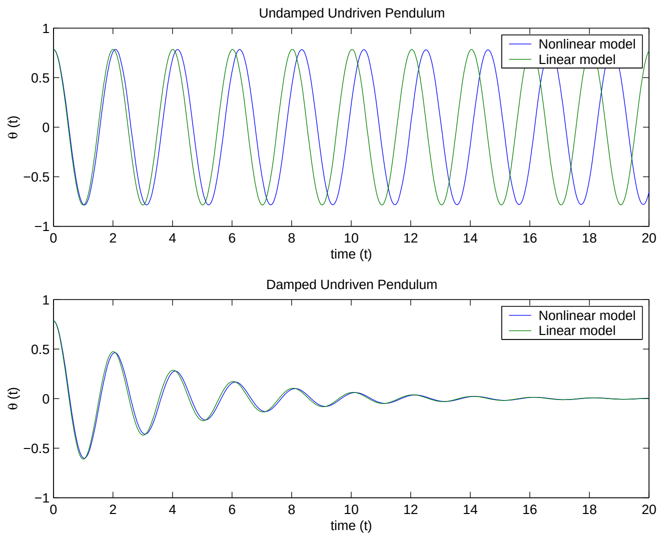


Figure 22.1. *The linear and nonlinear undriven models.*

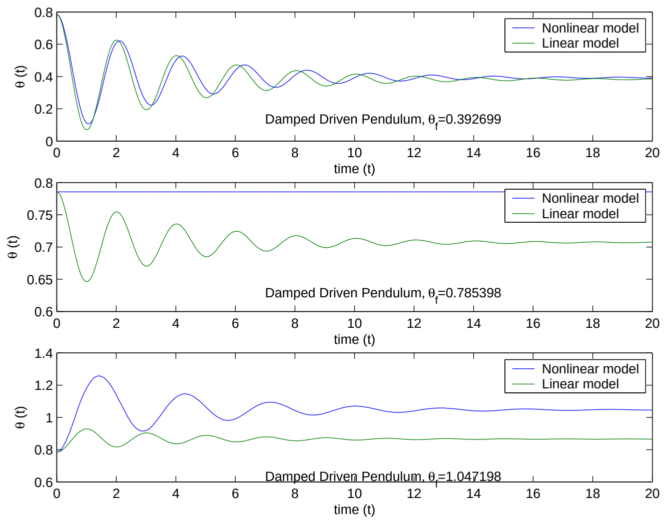


Figure 22.2. *The linear and nonlinear driven models.*

CHALLENGE 22.5. See the program `problem5.m` on the website. The $\theta(t)$ results for the original solution, shooting method, and finite difference method differ by at most 0.004.

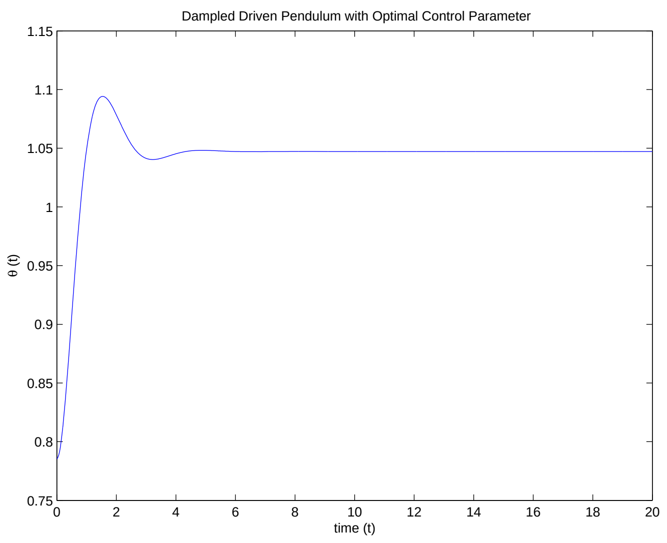


Figure 22.3. *The path of the robot arm with optimal control.*

CHALLENGE 22.6. See the program `problem6.m` on the website. The energy function returns the energy as specified above plus a large positive *penalty term* in case the parameter is unsuccessful; the penalty keeps the minimizer from choosing an unsuccessful parameter. For $b = -1.7859$, the total energy consumed is about 43.14 Joules. The motion of the pendulum is shown in Figure 22.3.

Note that it is always a good idea to sketch the function to be minimized to see if the reported solution is reasonable.

Chapter 23

Solutions: Case Study: Finite Differences and Finite Elements: Getting to Know You

CHALLENGE 23.1.

$$\frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix},$$

where $h = 1/5$, $u_j \approx u(jh)$, and $f_j = f(jh)$.

CHALLENGE 23.2. Documentation is posted on the website for the program `finitediff2.m` of Problem 3, which is very similar to `finitediff1.m` but more useful. Please remember that if you use a program like `finitediff1.m` and fail to include the name of the program's author, or at least a reference to the website from which you obtained it, it is plagiarism. Similarly, your implementation of `finitediff2.m` should probably include a statement like, "Derived from finitediff1.m by Dianne O'Leary."

CHALLENGE 23.3. See `finitediff2.m` on the website.

CHALLENGE 23.4.

(a) First notice that if α and β are constants and v and z are functions of x, then

$$a(u, \alpha v + \beta z) = \alpha a(u, v) + \beta a(u, z),$$

since we can compute the integral of a sum as the sum of the integrals and then move the constants outside the integrals. Therefore,

$$\begin{aligned}
a(u_h, v_h) &= a(u_h, \sum_{j=1}^{M-2} v_j \phi_j) \\
&= \sum_{j=1}^{M-2} v_j a(u_h, \phi_j) \\
&= \sum_{j=1}^{M-2} v_j (f, \phi_j) \\
&= (f, \sum_{j=1}^{M-2} v_j \phi_j) \\
&= (f, v_h).
\end{aligned}$$

(b) We compute

$$\begin{aligned}
a(\phi_j, \phi_j) &= \int_0^1 (\phi_j'(t))^2 \mathrm{d}t \\
&= \int_{(j-1)h}^{(j+1)h} (\phi_j'(t))^2 \mathrm{d}t \\
&= 2\int_{(j-1)h}^{jh} \frac{1}{h^2} \mathrm{d}t \\
&= \frac{2}{h},
\end{aligned}$$

and

$$\begin{aligned}
a(\phi_j, \phi_{j+1}) &= \int_0^1 \phi_j'(t)\phi_{j+1}'(t) \mathrm{d}t \\
&= \int_{jh}^{(j+1)h} \phi_j'(t)\phi_{j+1}'(t) \mathrm{d}t \\
&= \int_{jh}^{(j+1)h} \frac{(-1)}{h}\frac{1}{h} \mathrm{d}t \\
&= -\frac{1}{h}.
\end{aligned}$$

So our system becomes

$$\frac{1}{h}\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}$$

where u_j is the coefficient of ϕ_j in the representation of u_h and

$$f_j = \int_0^1 f(t)\phi_j(t)\mathrm{d}t = \int_{(j-1)h}^{(j+1)h} f(t)\phi_j(t)\mathrm{d}t,$$

which is h times a weighted average of f over the jth interval. The only difference between the finite difference system and this system is that we have replaced point samples of f by average values. Note that if $a(t)$ is not constant, then the systems look even more different.

CHALLENGE 23.5. See the program posted on the website.

CHALLENGE 23.6. See the program posted on the website.

CHALLENGE 23.7. Here are the results, in dull tables with interesting entries:

PROBLEM 1
Using coefficient functions $a(1)$ and $c(1)$ with true solution $u(1)$
Infinity norm of the error at the grid points
for various methods and numbers of interior grid points M

$M =$	9	99	999
1st order finite difference	2.1541e-03	2.1662e-05	2.1662e-07
2nd order finite difference	2.1541e-03	2.1662e-05	2.1662e-07
Linear finite elements	1.3389e-13	1.4544e-14	1.4033e-13
Quadratic finite elements	3.1004e-05	3.5682e-09	3.6271e-13

PROBLEM 2
Using coefficient functions $a(1)$ and $c(2)$ with true solution $u(1)$
Infinity norm of the error at the grid points
for various methods and numbers of interior grid points M

$M =$	9	99	999
1st order finite difference	1.7931e-03	1.8008e-05	1.8009e-07
2nd order finite difference	1.7931e-03	1.8008e-05	1.8009e-07
Linear finite elements	6.1283e-04	6.1378e-06	6.1368e-08
Quadratic finite elements	2.7279e-05	3.5164e-09	1.7416e-12

PROBLEM 3

Using coefficient functions $a(1)$ and $c(3)$ with true solution $u(1)$
Infinity norm of the error at the grid points
for various methods and numbers of interior grid points M

$M =$	9	99	999
1st order finite difference	1.9405e-03	1.9529e-05	1.9530e-07
2nd order finite difference	1.9405e-03	1.9529e-05	1.9530e-07
Linear finite elements	4.3912e-04	4.3908e-06	4.3906e-08
Quadratic finite elements	2.8745e-05	3.5282e-09	3.6134e-13

PROBLEM 4
Using coefficient functions $a(2)$ and $c(1)$ with true solution $u(1)$
Infinity norm of the error at the grid points
for various methods and numbers of interior grid points M

$M =$	9	99	999
1st order finite difference	1.5788e-02	1.8705e-03	1.8979e-04
2nd order finite difference	3.8465e-03	3.8751e-05	3.8752e-07
Linear finite elements	1.3904e-03	1.3930e-05	1.3930e-07
Quadratic finite elements	1.6287e-04	1.9539e-08	1.9897e-12

PROBLEM 5
Using coefficient functions $a(3)$ and $c(1)$ with true solution $u(1)$
Infinity norm of the error at the grid points
for various methods and numbers of interior grid points M

$M =$	9	99	999
1st order finite difference	1.1858e-02	1.4780e-03	1.5065e-04
2nd order finite difference	3.6018e-03	3.6454e-05	3.6467e-07
Linear finite elements	8.3148e-04	8.2486e-06	1.2200e-06
Quadratic finite elements	1.0981e-04	1.6801e-06	2.5858e-06

PROBLEM 6
Using coefficient functions $a(1)$ and $c(1)$ with true solution $u(2)$
Infinity norm of the error at the grid points
for various methods and numbers of interior grid points M

$M =$	9	99	999
1st order finite difference	8.9200e-02	9.5538e-02	9.6120e-02
2nd order finite difference	8.9200e-02	9.5538e-02	9.6120e-02
Linear finite elements	8.6564e-02	9.5219e-02	9.6086e-02
Quadratic finite elements	8.6570e-02	9.5224e-02	9.6088e-02

PROBLEM 7
Using coefficient functions $a(1)$ and $c(1)$ with true solution $u(3)$

Infinity norm of the error at the grid points
for various methods and numbers of interior grid points M

$M =$	9	99	999
1st order finite difference	1.5702e-01	1.6571e-01	1.6632e-01
2nd order finite difference	1.5702e-01	1.6571e-01	1.6632e-01
Linear finite elements	1.4974e-01	1.6472e-01	1.6622e-01
Quadratic finite elements	1.4975e-01	1.6472e-01	1.6622e-01

Discussion:

Clearly, the finite difference methods are easier to program and therefore are almost always used when x is a single variable. Finite elements become useful, though, when x has 2 or more components and the shape of the domain is nontrivial.

The bulk of the work in these methods is in function evaluations. We need $O(M)$ evaluations of a, c, and f in order to form each matrix. For finite differences, the constant is close to 1, but `quad` (the numerical integration routine) uses many function evaluations per call (on the order of 10), making formation of the finite element matrices about 10 times as expensive.

The experimental rate of convergence should be calculated as the $\log_{10}$ of the ratio of the successive errors (since we increase the number of grid points by a factor of 10 each time). There are several departures from the expected rate of convergence:

- `finitediff1` is expected to have a linear convergence rate ($r = 1$), but has $r = 2$ for the first three problems because $a' = 0$ and the approximation is the same as that in `finitediff2`.

- The quadratic finite element approximation has $r = 4$ on Test Problems 1-4, better than the $r = 3$ we might expect. This is called *superconvergence* and happens because we only measured the error at the grid points, whereas the $r = 3$ result was for the average value of the error over the entire interval.

- Linear finite elements give almost an exact answer to Test Problem 1 at the grid points (but not between the grid points). This occurs because our finite element equations demand that

$$a(u_h, \phi_j) = (u_h', \phi_j') = [-u_h(t_{j-1}) + 2u_h(t_j) - u_h(t_{j+1})]/h = (f, \phi),$$

  and our true solution also satisfies this relation.

- In Test Problem 5, the coefficient function a has a discontinuous derivative at $x = 1/3$. The matrix entries computed by the numerical integration routine are not very accurate, so the finite element methods appear to have slow convergence. This can be fixed by extra calls to `quad` so that it never tries to integrate across the discontinuity.

- The "solution" to Test Problem 6 has a discontinuous derivative, and the "solution" to Test Problem 7 is discontinuous. None of our methods compute good approximations, although all of them return a reasonable answer (See Figure 23.1) that could be mistaken for what we are looking for. The finite difference approximations lose accuracy because their error term depends on u''. The finite element equations were derived from the integrated (weak) formulation of our problem, and when we used integration by parts, we left off the boundary term that we would have gotten at $x = 2/3$, so our equations are wrong. This is a case of, "Be careful what you ask for."

- The entries in the finite element matrices are only approximations to the true values, due to inaccuracy in estimation of the integrals. This means that as the grid size is decreased, we need to reduce the tolerance that we send to `quad` in order to keep the matrix accurate enough.
- The theoretical convergence rate only holds down to the rounding level of the machine, so if we took even finer grids (much larger M), we would fail to see the expected rate.

On these simple 1-dimensional examples, we uncovered many pitfalls in naive use of finite differences and finite elements. Nevertheless, both methods are quite useful when used with care.

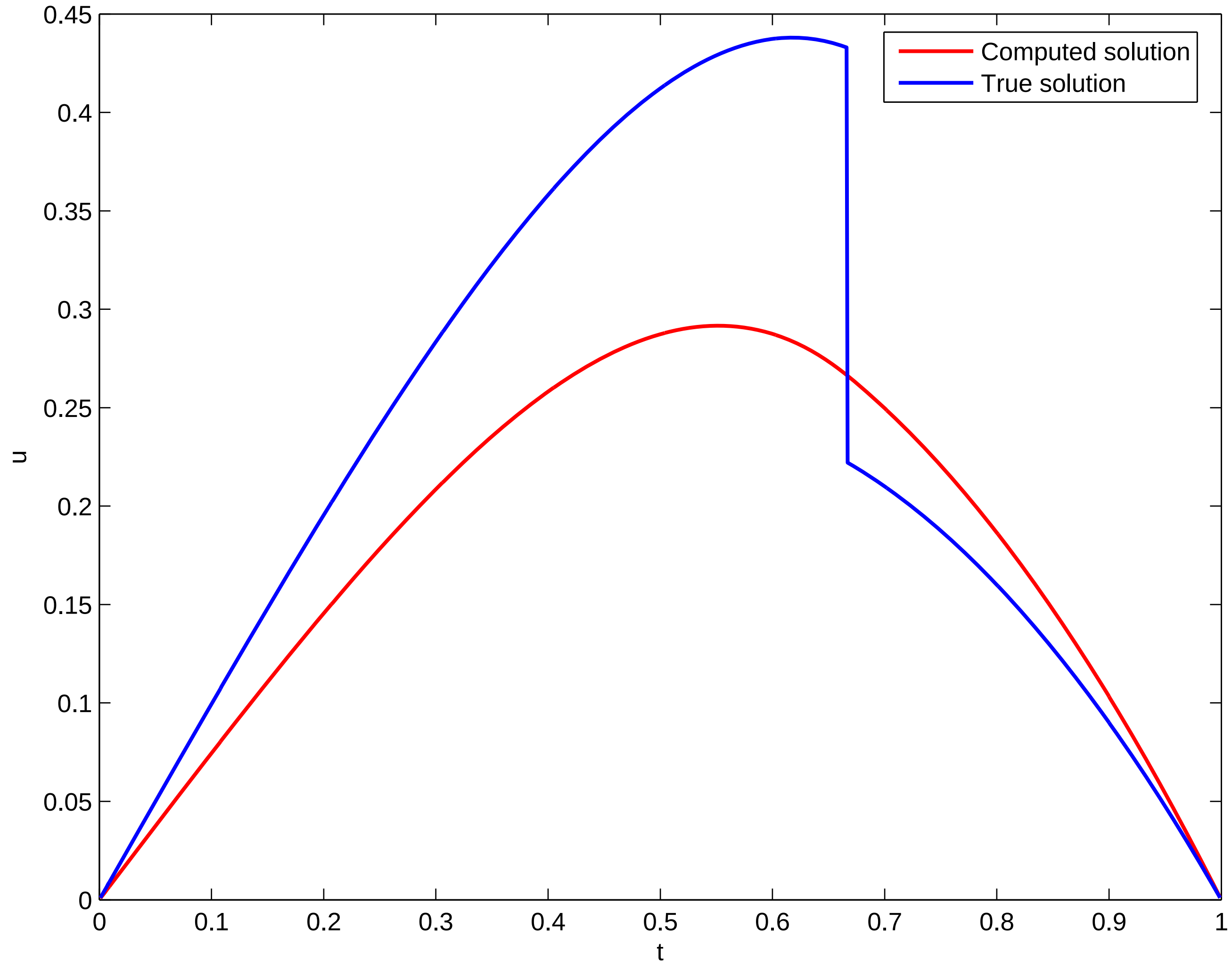


Figure 23.1. *The "solution" to the seventh test problem. We compute an accurate answer to a different problem.*

Unit VI

SOLUTIONS: Nonlinear Systems and Continuation Methods

Chapter 24

Solutions: Nonlinear Systems

CHALLENGE 24.1.

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{bmatrix} x^2y^3 + xy - 2 \\ 2xy^2 + x^2y + xy \end{bmatrix}$$

and

$$\boldsymbol{J}(\boldsymbol{x}) = \begin{bmatrix} 2xy^3 + y & 3x^2y^2 + x \\ 2y^2 + 2xy + y & 4xy + x^2 + x \end{bmatrix}.$$

```
x = [5;4];
for i=1:5,
    F = [x(1)^2*x(2)^3 + x(1)*x(2) - 2;
         2*x(1)*x(2)^2 + x(1)^2*x(2) + x(1)*x(2)];

    J = [2*x(1)*x(2)^3 + x(2), 3*x(1)^2*x(2)^2 + x(1);
         2*x(2)^2 + 2*x(1)*x(2) + x(2), 4*x(1)*x(2) + x(1)^2 + x(1)];

    p = - ( J \ F );
    x = x + p;
end
```

CHALLENGE 24.2.

- The first line does n^2 divisions. It would be better to add parentheses to drop this to n: `B = B + (y-B*s)*(s'/(s'*s))` .

- It is silly to refactor the matrix `B` each time, when it is just a rank-1 update of the previous matrix. Instead, update a decomposition or (less desirable) update the inverse using the techniques of Chapter 7.

CHALLENGE 24.3. For some vector $\boldsymbol{r}$, we need to form

$$(\boldsymbol{A} - \boldsymbol{Z}\boldsymbol{V}^T)^{-1}\boldsymbol{r} = \boldsymbol{A}^{-1}\boldsymbol{r} + \boldsymbol{A}^{-1}\boldsymbol{Z}(\boldsymbol{I} - \boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{Z})^{-1}\boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{r},$$

where $\boldsymbol{A} = \boldsymbol{B}^{(k)}$, $\boldsymbol{Z} = \boldsymbol{y} - \boldsymbol{B}^{(k)}\boldsymbol{s}$, and $\boldsymbol{V} = -\boldsymbol{s}/(\boldsymbol{s}^T\boldsymbol{s})$. Thus we need to

- form $\boldsymbol{t} = \boldsymbol{A}^{-1}\boldsymbol{r}$ and $\boldsymbol{u} = \boldsymbol{A}^{-1}\boldsymbol{Z}$, at a cost of $2p$ multiplications,
- form $\alpha = 1 - \boldsymbol{V}^T\boldsymbol{u}$ at a cost of n multiplications,
- form $\boldsymbol{w} = ((\boldsymbol{V}^T\boldsymbol{t})/\alpha)\boldsymbol{u}$ at a cost of $2n$ multiplications and 1 division
- add $\boldsymbol{t}$ and $\boldsymbol{w}$.

The total number of multiplications is $2p + 3n + 1$. Again, updating a matrix decomposition has many advantages over this approach!

CHALLENGE 24.4. (Partial Solution.)

(a) We compute the partial of $\rho_{\boldsymbol{a}}(\lambda, \boldsymbol{x})$ with respect to λ:

$$\boldsymbol{s} = \begin{bmatrix} x^2y^3 + xy - 2 - (x - a_1) \\ 2xy^2 + x^2y + xy - (y - a_2) \end{bmatrix}.$$

Then the Jacobian of ρ_a is the 2×3 matrix

$$\widehat{\boldsymbol{J}}(\lambda, \boldsymbol{x}) = \begin{bmatrix} \boldsymbol{s}, & (1-\lambda)\boldsymbol{I} + \lambda\boldsymbol{J}(\boldsymbol{x}) \end{bmatrix},$$

where $\boldsymbol{J}(\boldsymbol{x})$ is the matrix from Problem 1.

(b) In order for the function to be transversal to zero, the matrix $\widehat{\boldsymbol{J}}(\lambda, \boldsymbol{x})$ must be full rank (i.e., rank-2) at every point $\lambda \in [0, 1)$, $x, y \in (-\infty, \infty)$.

The matrix $\boldsymbol{J}(\boldsymbol{x})$ has two eigenvalues – call them α_1 and α_2. The matrix $\boldsymbol{K} = (1-\lambda)\boldsymbol{I} + \lambda\boldsymbol{J}(\boldsymbol{x})$ has eigenvalues $(1-\lambda) + \lambda\alpha_i$, so it is singular only if $\lambda = 1/(1-\alpha_1)$ or $\lambda = 1/(1-\alpha_2)$. Even if that happens, it is likely that the vector $\boldsymbol{s}$ will point in a different direction, making the rank of $\boldsymbol{J}(\lambda, \boldsymbol{x})$ equal to 2.

CHALLENGE 24.5. Using the Lagrange form of the interpolating polynomial, we can write

$$p(f) = L_1(f)t_1 + L_2(f)t_2 + L_3(f)t_3,$$

where

$$L_1(f) = \frac{(f-f_2)(f-f_3)}{(f_1-f_2)(f_1-f_3)},$$
$$L_2(f) = \frac{(f-f_1)(f-f_3)}{(f_2-f_1)(f_2-f_3)},$$
$$L_3(f) = \frac{(f-f_1)(f-f_2)}{(f_3-f_1)(f_3-f_2)}.$$

It is easy to verify that $L_j(f_j) = 1$ and $L_j(f_k) = 0$ if $j \neq k$. Therefore, $p(f_1) = t_1$, $p(f_2) = t_2$, and $p(f_3) = t_3$, as desired.

Now, we want to estimate the value of t so that $f(t) = 0$, and we take this estimate to be $p(0)$. We calculate:

```
L(1) = f(2)*f(3)/((f(1)-f(2))*(f(1)-f(3)));
L(2) = f(1)*f(3)/((f(2)-f(1))*(f(2)-f(3)));
L(3) = f(1)*f(2)/((f(3)-f(1))*(f(3)-f(2)));
testimated = L*t;
```

Chapter 25

Solutions: Case Study: Variable-Geometry Trusses: What's Your Angle?

CHALLENGE 25.1.

See Figures 25.1 – 25.4.

CHALLENGE 25.2.

See the programs on the website.

CHALLENGE 25.3.

See the programs on the website and the figure in the book. Compare your results with Arun, who found 8 solutions for Platform A, 8 for Platform B, 4 for Platform C, and 16 for Platform D.

The continuation method was much slower than `fsolve` and gave no more solutions. A trick called **homogenization** can be used to improve the homotopy. This involves replacing the variables by their inverses in order to take solutions that are at infinity and map them to zero. See [1,2,3] for more details.

1. V. Arun, *The Solution of Variable-Geometry Truss Problems Using New Homotopy Continuation Methods*, PhD thesis, Mechanical Engineering Department, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, September 1990.

2. V. Arun, C. F. Reinholtz, and L. T. Watson, Application of new homotopy continuation techniques to variable geometry trusses, *Trans. of the ASME*, 114:422–428, September 1992.

3. A. Morgan, *Solving Polynomial Systems Using Continuation For Engineering and Scientific Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

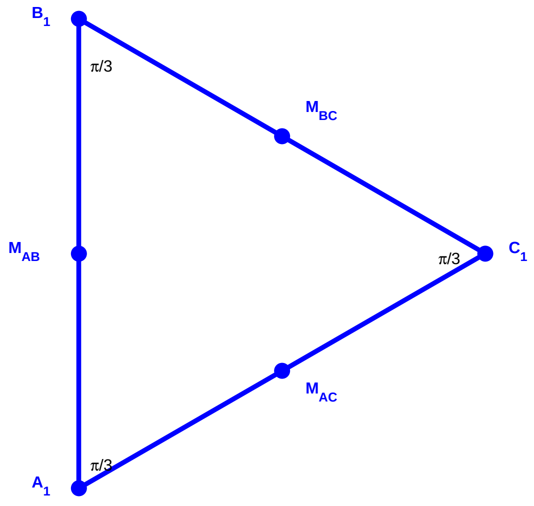


Figure 25.1. *The triangle at the base of the platform (in the xy-plane) and the midpoints of each side.*

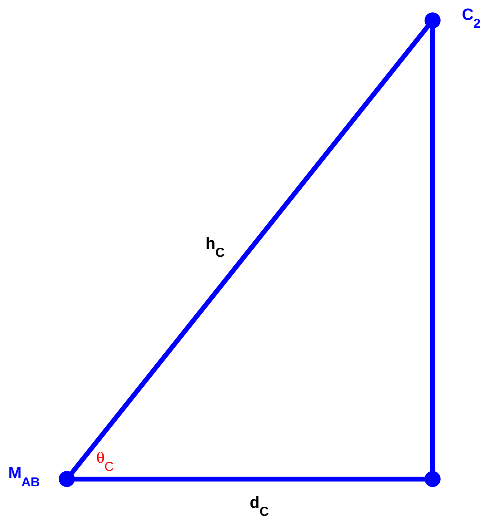


Figure 25.2. *The distance from M_{AB} (located in the xy-plane) to C_2 is h_C, and the side of the triangle with length $d_C = h_C \cos(\theta_C)$ points in the positive x-direction. Therefore, the coordinates of C_2 are those of $\mathbf{M}_{AB}$ plus d_C in the x-direction, 0 in the y-direction, and $h_C \sin(\theta_C)$ in the z-direction.*

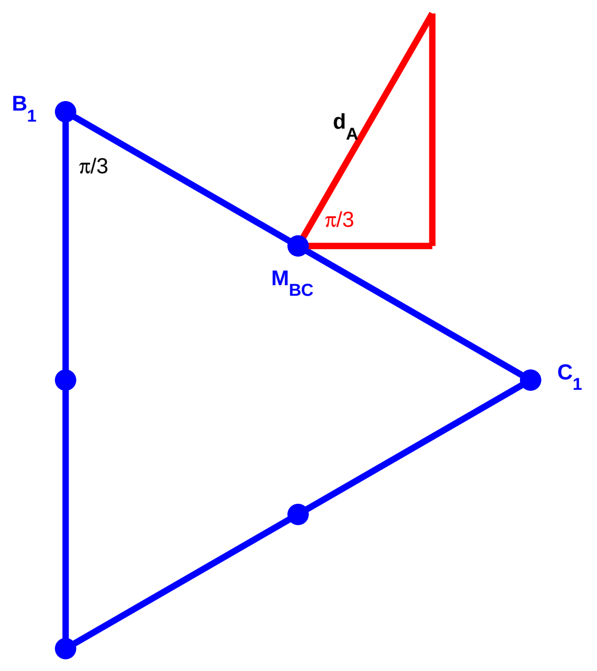


Figure 25.3. *The endpoint of the side labeled* d_A *determines the* x *and* y *coordinates of* A_2*. The length is* $d_A = h_A \cos(\theta_A)$*, so the* x*-displacement from* M_{BC} *is* $d_A \cos(\pi/3)$ *and the* y*-displacement is* $d_A \sin(\pi/3)$*. The* z*-displacement is* $h_A \sin(\theta_A)$*.*

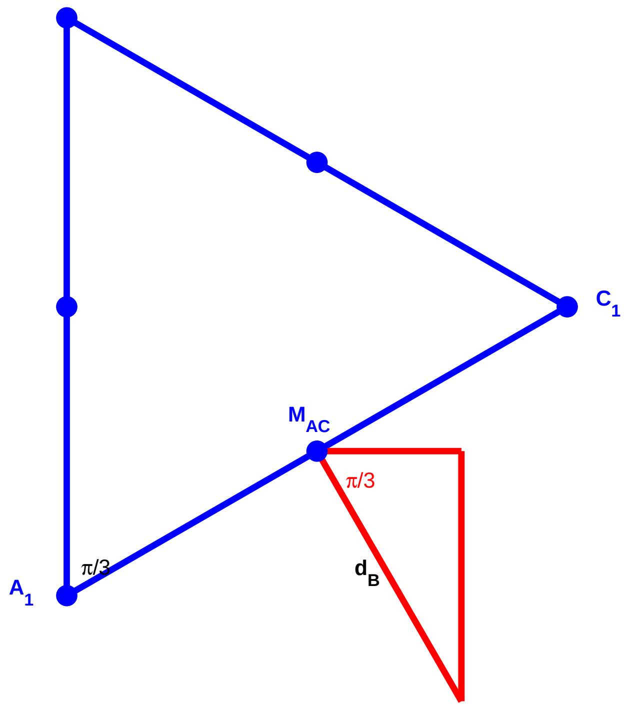


Figure 25.4. *The endpoint of the side labeled d_B determines the x and y coordinates of B_2. The length is $d_B = h_B \cos(\theta_B)$, so the x-displacement from M_{AC} is $d_B \cos(-\pi/3)$ and the y-displacement is $d_B \sin(-\pi/3)$. The z-displacement is $h_B \sin(\theta_B)$.*

Chapter 26

Solutions: Case Study: Beetles, Cannibalism, and Chaos: Analyzing a Dynamical System Model

CHALLENGE 26.1. The results are shown in Figure 26.1. When $\mu_A = 0.1$, the solution eventually settles into a cycle, oscillating between two different values: 18.7 and 321.6 larvae, 156.7 and 9.1 pupae, and 110.1 and 121.2 adults. Thus the population at 4 week intervals is constant. Note that the peak pupae population lags 2 weeks behind the peak larvae population, and that the oscillation of the adult population is small compared to the larvae and pupae.

For $\mu_A = 0.6$, the population eventually approaches a fixed point: 110.7 larvae, 54.0 pupae, and 42.3 adults.

In the third case, $\mu_A = 0.9$, there is no regular pattern for the solution, and it is called **chaotic**. The number of larvae varies between 18 and 242, the number of pupae between 8 and 117, and the number of adults between 9 and 94.

CHALLENGE 26.2. The results are shown in Figure 26.2. For the stable solutions, if the model is initialized with population values near A_{fixed}, L_{fixed}, and P_{fixed}, it will converge to these equilibrium values.

CHALLENGE 26.3. The bifurcation diagram is shown in Figure 26.3. The largest tested value of μ_A that gives a stable solution is 0.58. If the computation were performed in exact arithmetic, the graph would just be a plot of L_{fixed} vs. μ_A. When the solution is stable, rounding error in the computation produces a nearby point from which the iteration tends to return to the fixed point. When the solution is unstable, rounding error in the computation can cause the computed solution to drift away. Sometimes it produces a solution that oscillates between two values (for example, when $\mu_A = 0.72$) and sometimes the solution becomes chaotic or at least has a long cycle (for example, when $\mu_A = 0.94$).

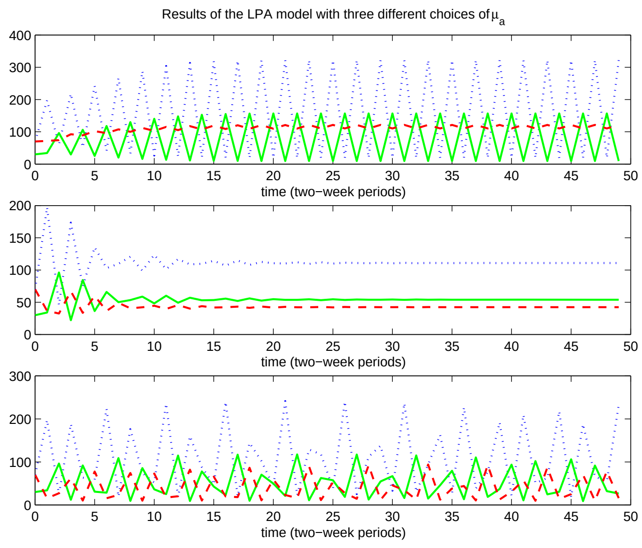


Figure 26.1. *Model predictions for* $b = 11.6772$, $\mu_L = 0.5129$, $c_{el} = 0.0093$, $c_{ea} = 0.0110$, $c_{pa} = 0.0178$, $L(0) = 70$, $P(0) = 30$, $A(0) = 70$, *and* $\mu_A = 0.1$ *(top),* 0.6 *(middle), and* 0.9 *(bottom). Number of larvae (blue dotted), pupae (green solid), and adults (red dashed).*

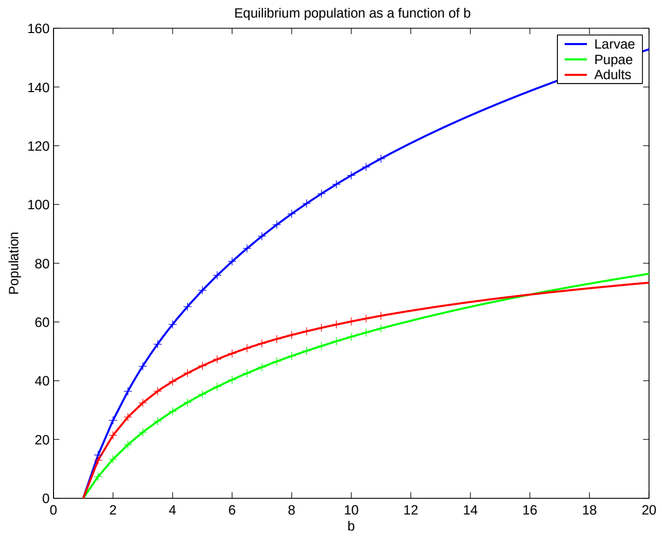


Figure 26.2. *Equilibrium populations for* $\mu_L = 0.5$, $\mu_A = 0.5$, $c_{el} = 0.01$, $c_{ea} = 0.01$, *and* $c_{pa} = 0.01$, $b = 1.0, 1.5, 2.0, \ldots, 20.0$. *Stable solutions are marked with plusses.*

colony	c_{el}	c_{ea}	c_{pa}	b	μ_L	μ_A	residual
new: 1	0.018664	0.008854	0.020690	5.58	0.144137	0.036097	5.04
old: 1	0.009800	0.017500	0.019800	23.36	0.472600	0.093400	17.19
new: 2	0.004212	0.013351	0.028541	6.77	0.587314	0.000005	7.25
old: 2	0.010500	0.008700	0.017400	11.24	0.501400	0.093000	14.24
new: 3	0.018904	0.006858	0.035082	6.47	0.288125	0.000062	4.37
old: 3	0.008000	0.004400	0.018000	5.34	0.508200	0.146800	4.66
new: 4	0.017520	0.012798	0.023705	6.79	0.284414	0.005774	6.47
old: 4	0.008000	0.006800	0.016200	7.20	0.564600	0.109900	7.42

Table 26.1. *Parameter estimates computed in Challenge 4 for our minimization function ("new") and that of Dennis et al. ("old").*

Colony	Norm of data vector	New residual	Old residual
Colony 1	33.55	5.04	17.19
Colony 2	33.70	7.25	14.24
Colony 3	33.44	4.37	4.66
Colony 4	33.68	6.47	7.42

Table 26.2. *Residual norms computed in Challenge 4 for our minimization function ("new") and that of Dennis et al. ("old").*

CHALLENGE 26.4. The bounds used were 0 and 1 for all parameters except b. The value of b was confined to the interval $[0.1, 9.0]$. The results are summarized in Tables 26.1 and 26.2

CHALLENGE 26.5. When the data is randomly perturbed, the estimate of b for the second colony ranges from 4.735941 to 6.831328.

A larger upper bound for b tended to cause the minimizer to converge to a local solution with a much larger residual.

There are many ways to measure sensitivity:

- We might ask how large a change we see in b when the data is perturbed a bit. This is a **forward error** result.
- We might ask how large a change we see in the residual when the value of b is perturbed a bit. This is a **backward error** result.

To estimate the forward error, I repeated the fit, after adding 50 samples of normally distributed error (mean 0, standard deviation 1) to the log of the counts. This is only an approximation to the error assumption usually made for counts,

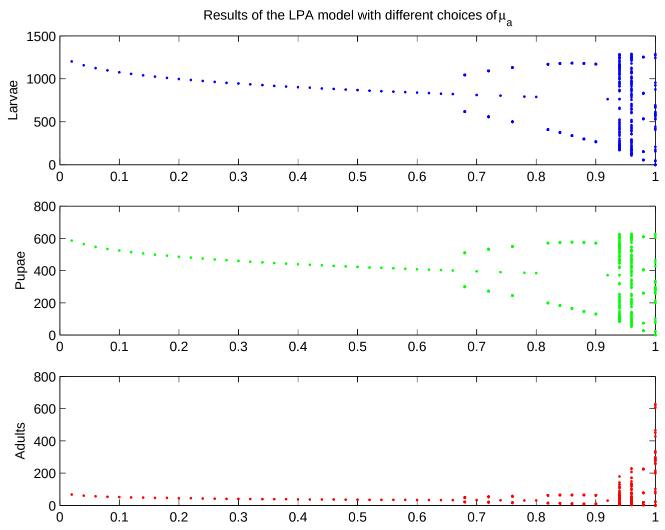


Figure 26.3. *The bifurcation diagram for the data in Problem 3.*

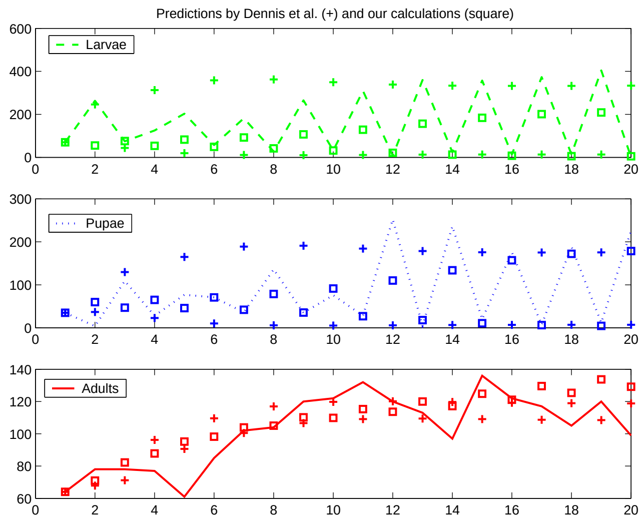


Figure 26.4. *Model predictions for Colony 1.*

Poisson error, but by using the log function in their minimization, the authors are assuming that this is how the error behaves. Even so, the estimates, shown in Figure 26.8, range from 1.00 to 9.00, quite a large change.

To estimate the backward error, I varied b, keeping the other parameters at their optimal values, and plotted the resulting residual vs. b in Figure 26.9. We see that the residual is not very sensitive to changes in b.

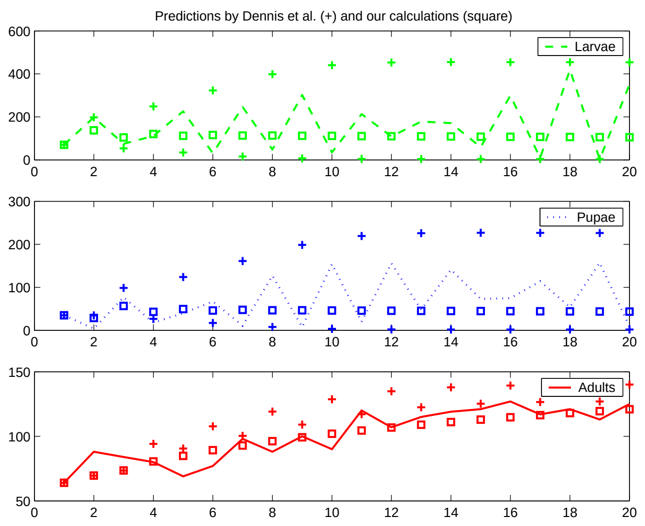


Figure 26.5. *Model predictions for Colony 2.*

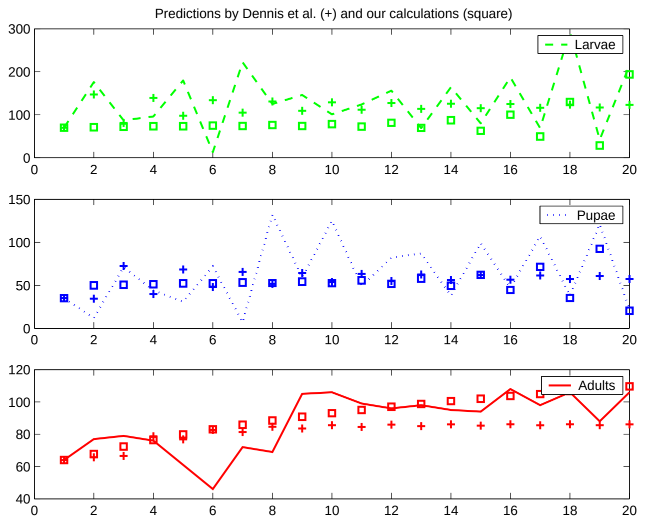


Figure 26.6. *Model predictions for Colony 3.*

Then I minimized the residual as a function of the 5 parameters remaining after setting b to fixed values. From Figure 26.10, we conclude that for any value of b between 1 and 50 we can obtain a residual norm within 10% of the computed minimum over all choices of b. This model seems to give no insight into the true value of b.

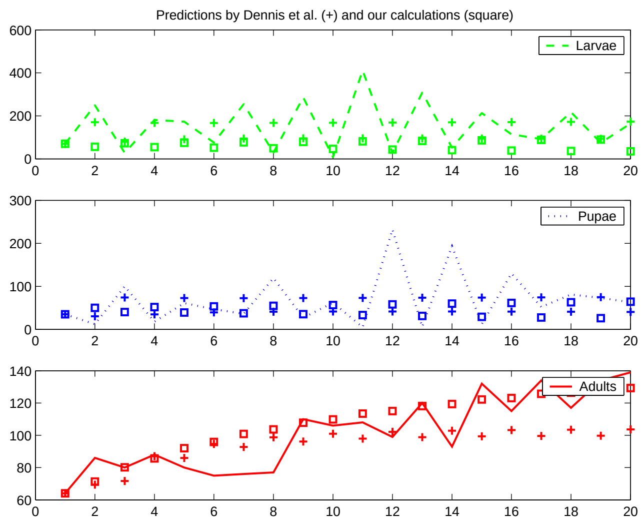


Figure 26.7. *Model predictions for Colony 4.*

But as a final attempt, I used a continuation algorithm, repeating the computations from Figure 26.10, but starting each minimization from the optimal point found for the previous value of b. The resulting residuals, shown in Figure 26.11, are much smaller, and the b value is somewhat better determined – probably between 5 and 10. Even more interesting, the fitted model finally gives a reasonable approximation of the data; see Figure 26.12.

To check the reliability of these estimates, it would be a good idea to repeat the experiment for the data for the other three colonies, and to repeat the least squares calculations using a variety of initial guesses.

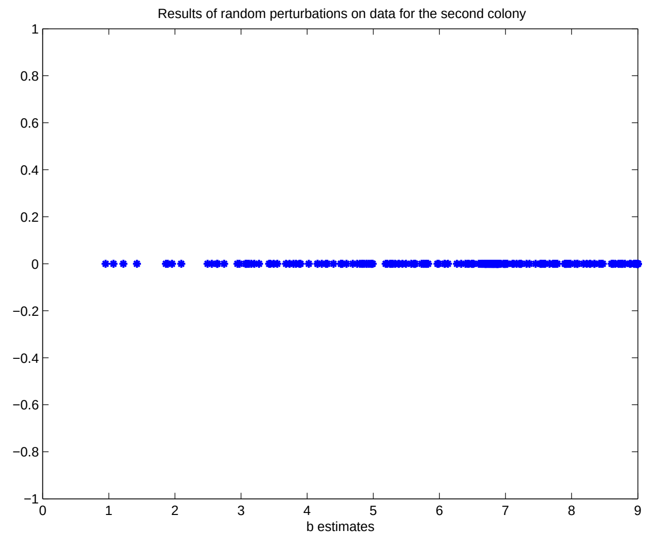


Figure 26.8. *Values of b computed for Colony 2 with 250 random perturbations of the log of the data, drawn from a normal distribution with mean 0 and standard deviation 1.*

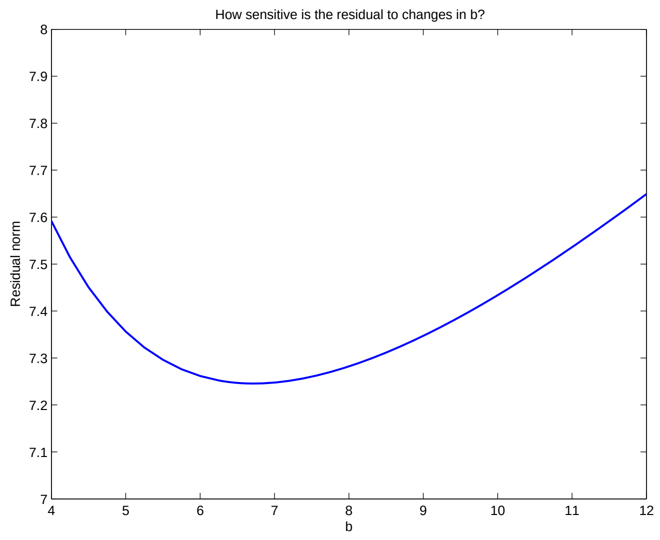


Figure 26.9. *Changes in the residual as b is changed for Colony 2, leaving the other parameters fixed.*

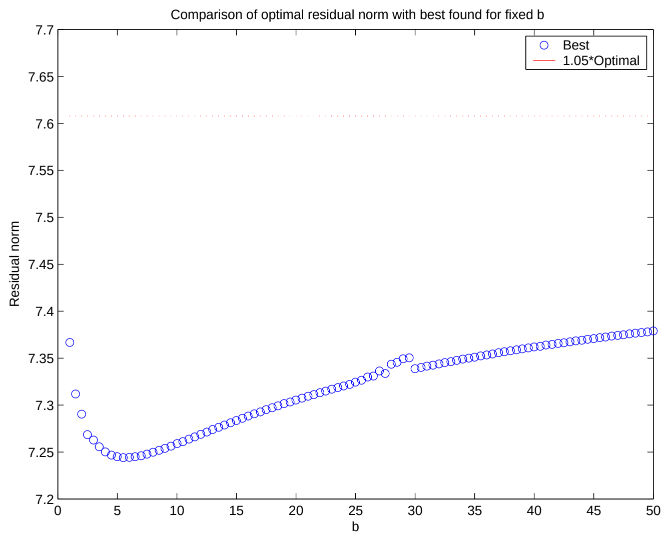


Figure 26.10. *Best (smallest) residuals for Colony 2 computed as a function of of the parameter b (blue circles) compared with the red dotted line, indicating a 10% increase over the minimal computed residual.*

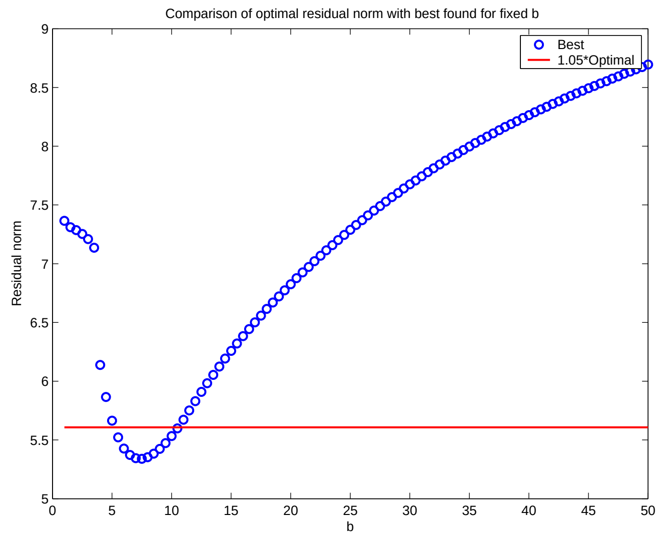


Figure 26.11. *Best (smallest) residuals for Colony 2 computed as a function of of the parameter b (blue circles) compared with the red dotted line, indicating a 10% increase over the minimal computed residual, using continuation.*

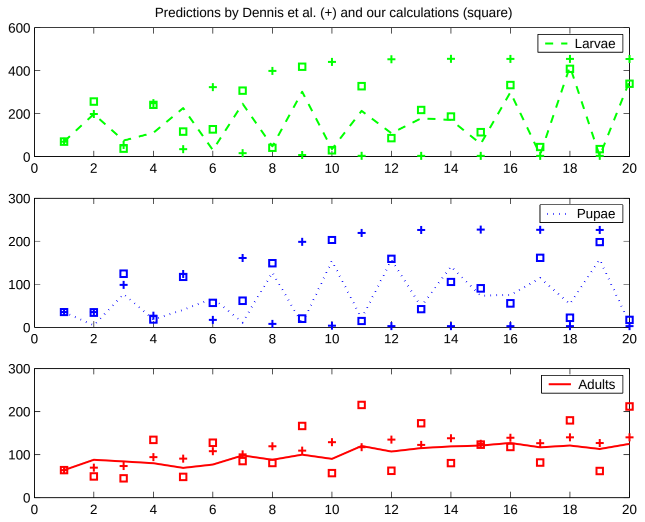


Figure 26.12. *Revised Model predictions for Colony 2, with parameters* $c_{el} = 0.008930$, $c_{ea} = c_{pa} = 0$, $b = 7.5$, $\mu_L = 0.515596$, $\mu_A = 0.776820$.

Unit VII

SOLUTIONS: Sparse Matrix Computations, with Application to Partial Differential Equations

Chapter 27

Solutions: Solving Sparse Linear Systems: Taking the Direct Approach

CHALLENGE 27.1.

(a) We notice that in Gauss elimination, we need only 5 row operations to zero elements in the lower triangle of the matrix, and the only row of the matrix that is changed is the last row. Since this row has no zeros, no new nonzeros can be produced.

(b) Since $\boldsymbol{P}^T\boldsymbol{P} = I$, we see that

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \iff \boldsymbol{P}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{P}\boldsymbol{b} \iff \boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^T(\boldsymbol{P}\boldsymbol{x}) = \boldsymbol{P}\boldsymbol{b},$$

and this verifies that the reordered system has the same solution as the original one.

CHALLENGE 27.2. For part (b), the important observation is that if element k is the first nonzero in row ℓ, then we start the elimination on row ℓ by a pivot operation with row k, after row k already has zeros in its first $k-1$ positions. Therefore, an induction argument shows that no new nonzeros can be created before the first nonzero in a row. A similar argument works for the columns. Part (a) is a special case of this.

CHALLENGE 27.3. The graph is shown in Figure 27.1. The given matrix is a permutation of a band matrix with bandwidth 2, and Reverse Cuthill-McKee was able to determine this and produce an optimal ordering. The reorderings and number of nonzeros in the Cholesky factor (nz($\boldsymbol{L}$)) are

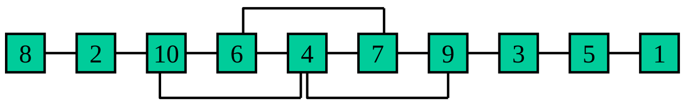


Figure 27.1. *The graph of the matrix of Problem 3.*

Method	Ordering	nz($\boldsymbol{L}$)
Original:	1 2 3 4 5 6 7 8 9 10	27
Reverse Cuthill-McKee:	1 5 3 9 7 4 6 10 2 8	22
Minimum degree:	2 8 10 6 1 3 5 9 4 7	24
Nested dissection(1 level):	8 2 10 6 4 9 3 5 1 7	25
Eigenpartition(1 level):	1 3 5 9 2 4 6 7 8 10	25

(Note that these answers are not unique, due to tiebreaking.) The resulting matrices and factors are shown in Figures 27.2-27.6.

CHALLENGE 27.4. Using a double precision word (2 words, or 8 bytes) as the unit of storage and seconds as the unit of time, here are the results:

Solving Laplace equation on circle sector with $n = 1208$			
Algorithm	storage	time	residual norm
Cholesky	660640	1.14e+00	4.04e-15
Cholesky, R-Cuthill-McKee	143575	7.21e-02	2.82e-15
Cholesky, minimum degree	92008	5.18e-02	1.96e-15
Cholesky, approx. mindeg	76912	1.70e-01	1.68e-15
Cholesky, eigenpartition	90232	4.59e+00	1.86e-15

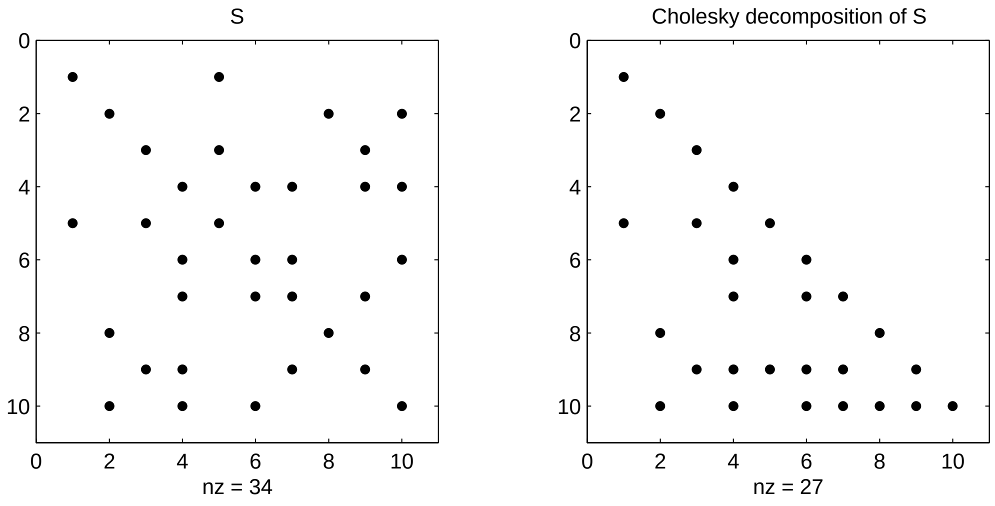


Figure 27.2. *Results of using original ordering.*

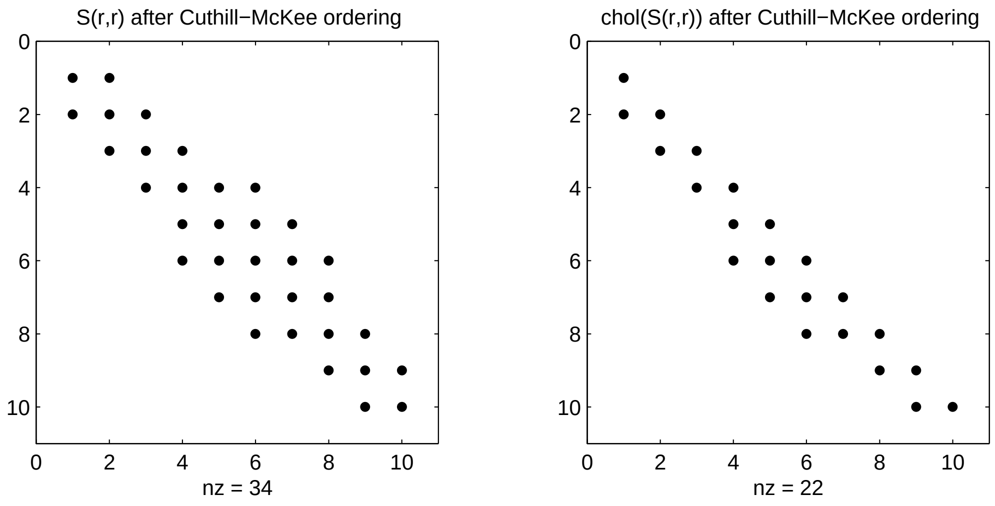


Figure 27.3. *Results of reordering using reverse Cuthill-McKee.*

Solving Laplace equation on circle sector with $n = 4931$			
Algorithm	storage	time	residual norm
Cholesky	6204481	3.21e+01	7.73e-15
Cholesky, R-Cuthill-McKee	1113694	7.08e-01	5.30e-15
Cholesky, minimum degree	486751	2.78e-01	2.85e-15
Cholesky, approx. mindeg	444109	2.34e-01	2.81e-15

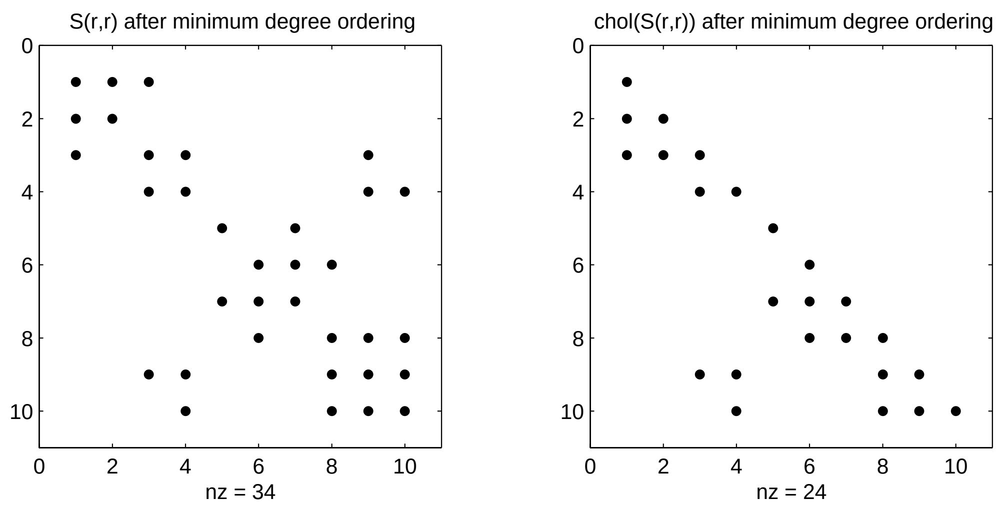


Figure 27.4. *Results of reordering using minimum degree.*

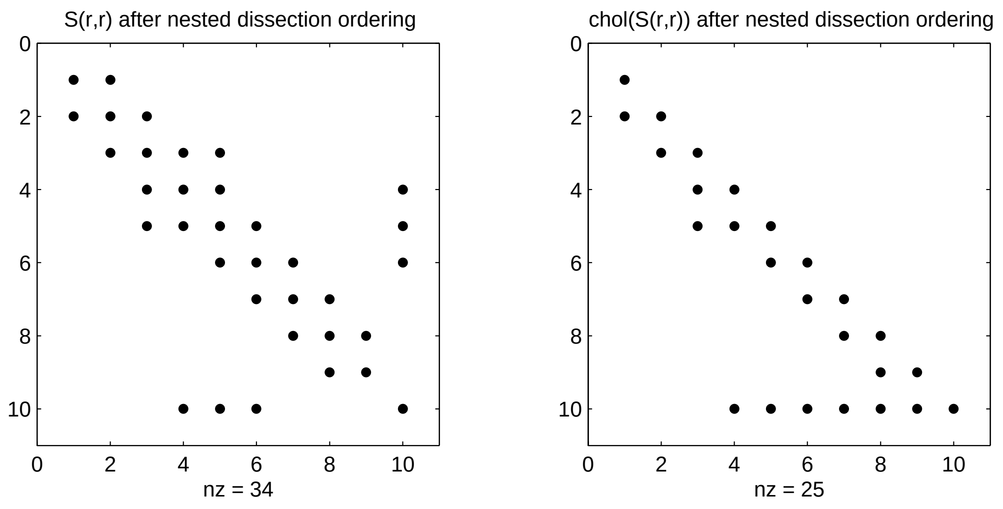


Figure 27.5. *Results of reordering using nested dissection.*

(There were too many recursions in eigenpartition method `specnd` from the Mesh Partitioning and Graph Separator Toolbox of Gilbert and Teng `http://www.cerfacs.fr/algor/Softs/MESHPART/`.)

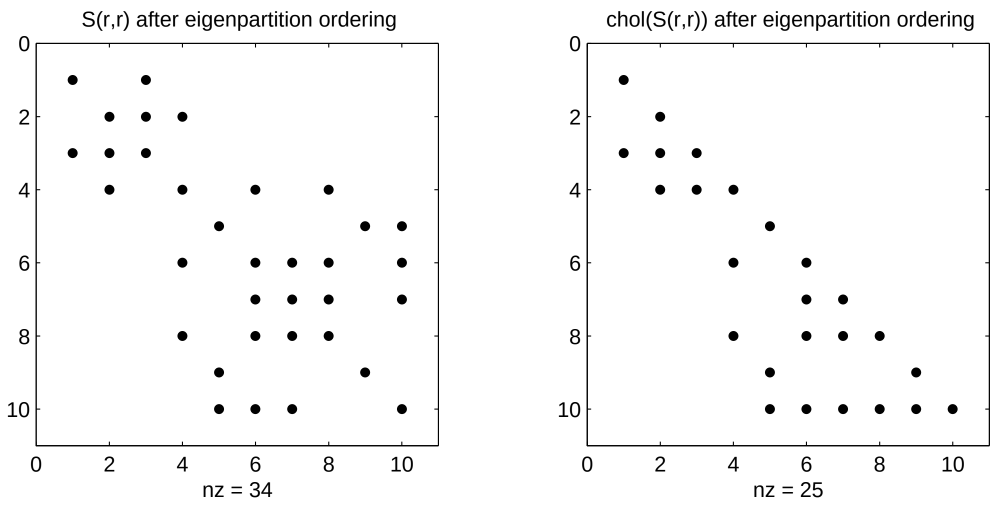


Figure 27.6. *Results of reordering using eigenpartitioning.*

Solving Laplace equation on box, with $n = 15625$			
Algorithm	storage	time	residual norm
Cholesky	28565072	1.02e+02	6.98e-14
Cholesky, R-Cuthill-McKee	16773590	3.79e+01	6.10e-14
Cholesky, minimum degree	8796896	4.08e+01	4.39e-14
Cholesky, approx. mindeg	7549652	3.08e+01	3.66e-14

(There were too many recursions in eigenpartition method `specnd`.)

All algorithms produced solutions with small residual norm. On each problem, the approximate minimum degree algorithm gave factors requiring the lowest storage, preserving sparsity the best, and on the last two problems, it used the least time as well. (Note that local storage used within MATLAB's `symrcm`, `symmmd`, `symamd`, and the toolbox `specnd` was not counted in this tabulation.) It is quite expensive to compute the eigenpartition ordering, and this method should only be used if the matrices will be used multiple times so that the cost can be amortized. To complete this study, it would be important to try different values of n, to determine the rate of increase of the storage and time as n increased.

To judge performance, several hardware parameters are significant, including computer (Sun Blade 1000 Model 1750), processor (Sun UltraSPARC-III), clock speed (750 MHz), and amount of RAM (1 Gbyte). The software specifications of importance include the operating system (Solaris 8) and the MATLAB version (6.5.1). Benchmarking is a difficult task, depending on the choice of hardware, software, and test problems, and our results on this problem should certainly raise more questions than they answer.

Chapter 28

Solutions: Iterative Methods for Linear Systems

CHALLENGE 28.1. See Solution to Challenge 6.

CHALLENGE 28.2. See the answer to Challenge 6.

CHALLENGE 28.3. See the answer to Challenge 6.

CHALLENGE 28.4. Consider our stationary iterative method

$$\boldsymbol{M}\boldsymbol{x}^{(k+1)} = \boldsymbol{N}\boldsymbol{x}^{(k)} + \boldsymbol{b}$$

or

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{M}^{-1}\boldsymbol{N}\boldsymbol{x}^{(k)} + \boldsymbol{M}^{-1}\boldsymbol{b}.$$

Manipulating these equations a bit, we get

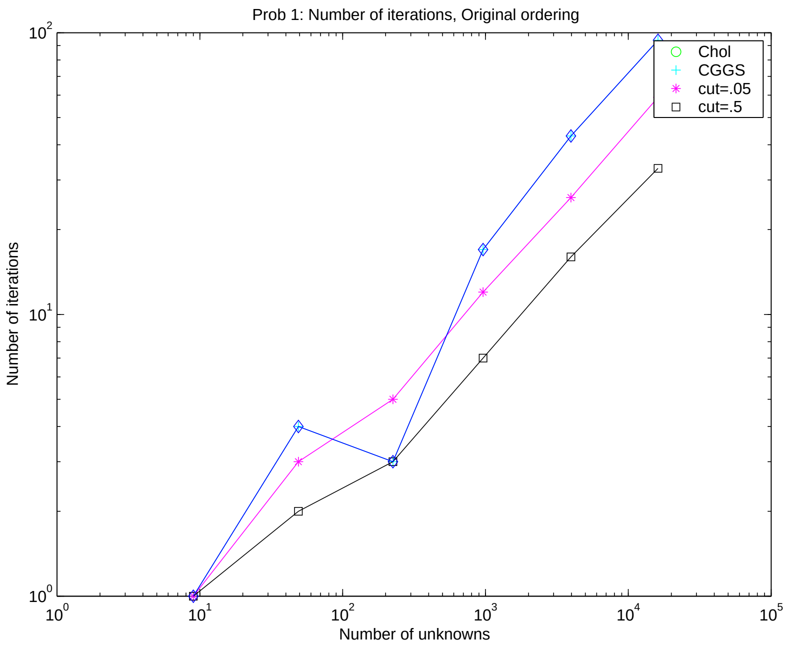
$$\begin{aligned}\boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + (\boldsymbol{M}^{-1}\boldsymbol{N} - \boldsymbol{I})\boldsymbol{x}^{(k)} + \boldsymbol{M}^{-1}\boldsymbol{b}\\ &= \boldsymbol{x}^{(k)} + \boldsymbol{M}^{-1}(\boldsymbol{N} - \boldsymbol{M})\boldsymbol{x}^{(k)} + \boldsymbol{M}^{-1}\boldsymbol{b}\\ &= \boldsymbol{x}^{(k)} + \boldsymbol{M}^{-1}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)})\\ &= \boldsymbol{x}^{(k)} + \boldsymbol{M}^{-1}\boldsymbol{r}^{(k)}.\end{aligned}$$

CHALLENGE 28.5. See the answer to Challenge 6.

CHALLENGE 28.6. The solution to these five problems is given on the website in `solution20.m`. The results for the square domain are shown in Figures 28.1 and 28.2. Gauss-Seidel took too many iterations to be competitive. The parameter `cut` is the drop-tolerance for the incomplete Cholesky decomposition. The AMD-Cholesky decomposition was the fastest algorithm for this problem, but it required 5.4 times the storage of CG and 2.6 times the storage of the PCG algorithm with incomplete Cholesky preconditioner for the problem of size 16129. Without reordering, Cholesky was slow and very demanding of storage, requiring almost 30 million double-precision words for the largest problem (almost 70 times as much as for the AMD reordering).

Gauss-Seidel took a large amount of time per iteration. This is an artifact of the implementation, since it is a bit tricky to get MATLAB to avoid working with the zero elements when accessing a sparse matrix row-by-row. Challenge: look at the program in `gauss_seidel.m` and try to speed it up. A better version is provided in the solution to Challenge 32.4.

Results for the domain with the circle cut out were similar; see Figures 28.3 and 28.4.



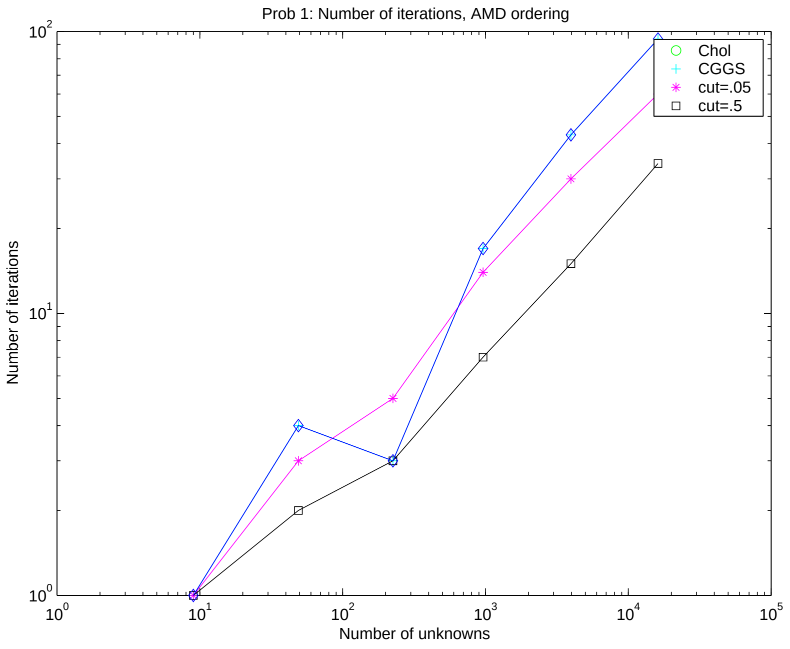
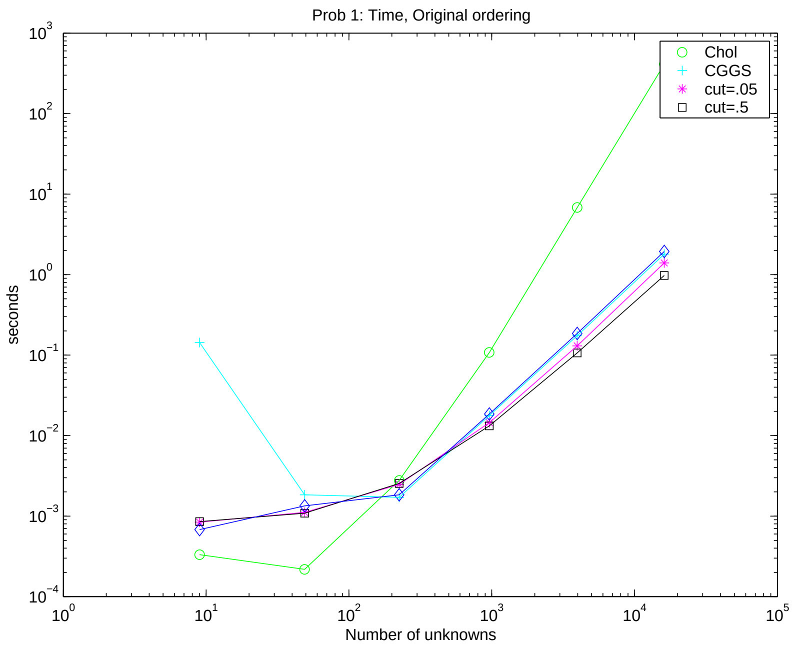


Figure 28.1. *Number of iterations for the various methods applied to the square domain.*



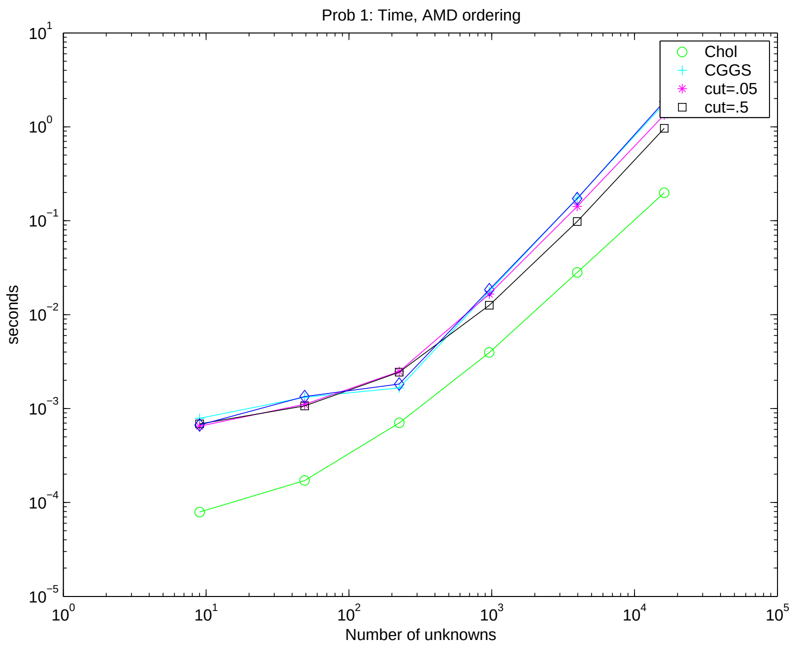
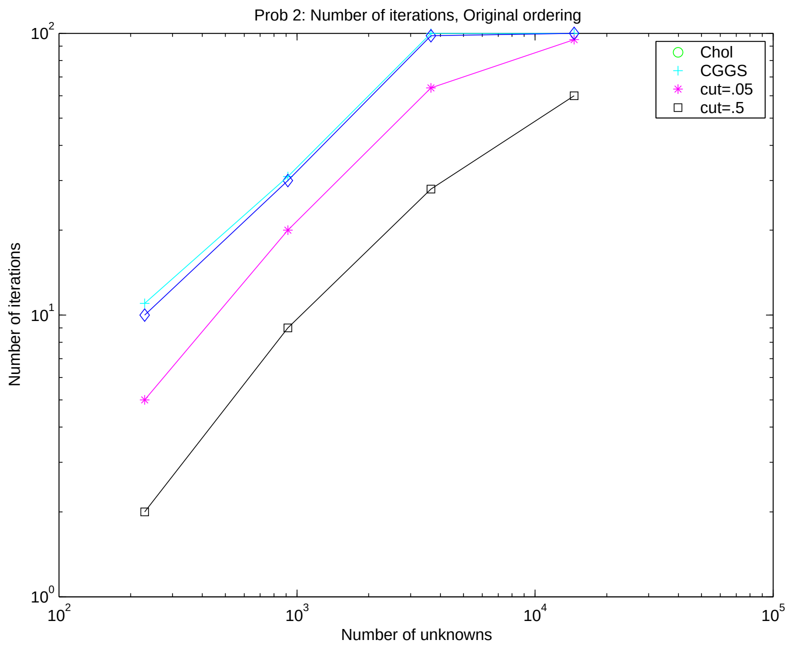


Figure 28.2. *Timings for the various methods applied to the square domain.*



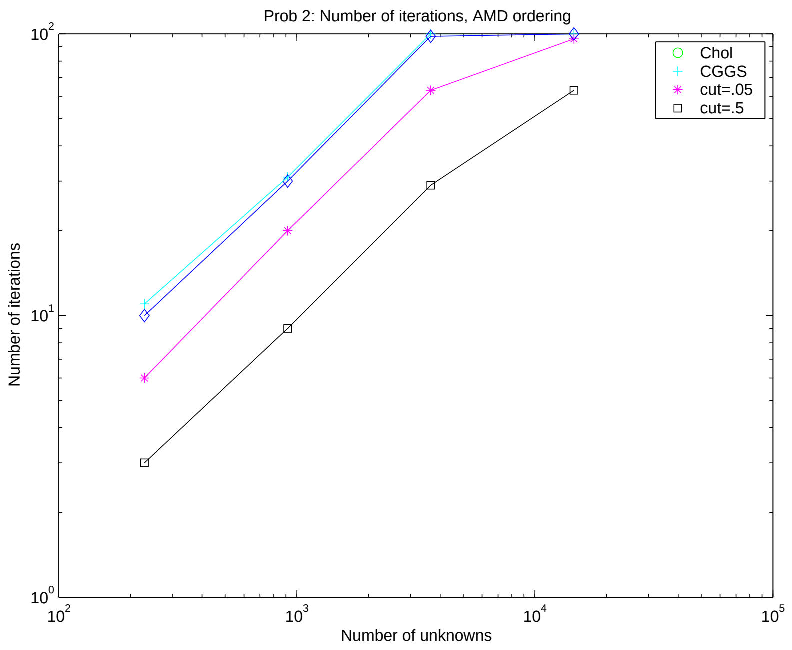
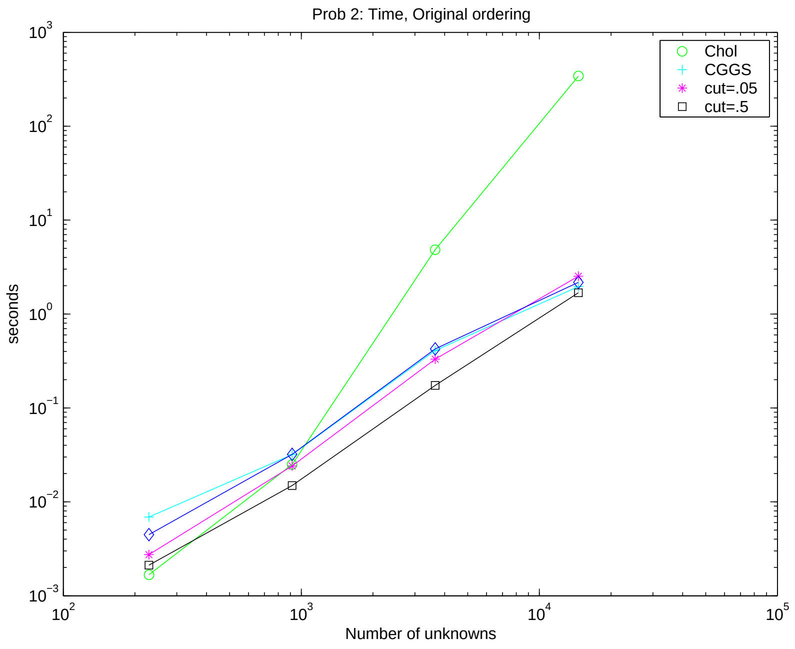


Figure 28.3. *Number of iterations for the various methods applied to the domain with the circle cut out.*



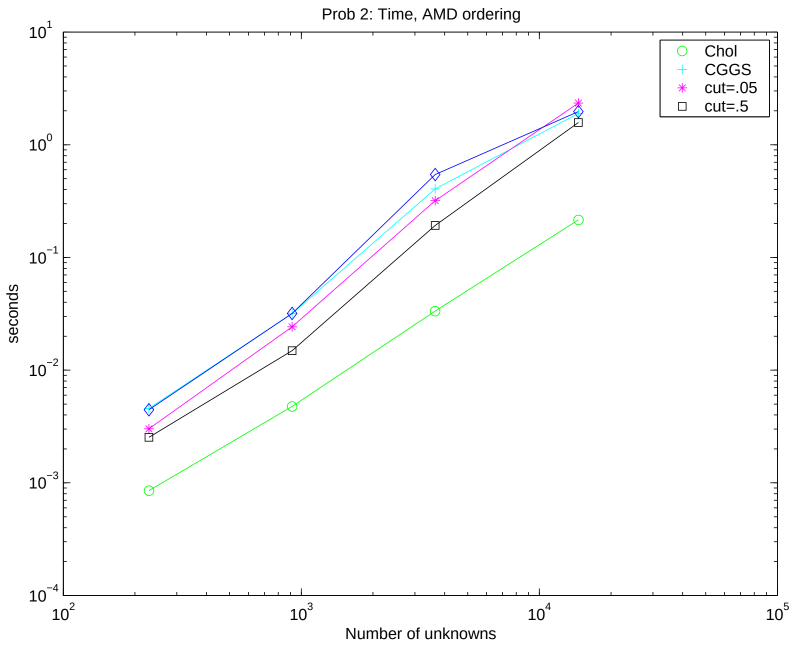


Figure 28.4. *Timings for the various methods applied to the domain with the circle cut out.*

Chapter 29

Solutions: Case Study: Elastoplastic Torsion: Twist and Stress

CHALLENGE 29.1. A sample MATLAB program is available on the website. We can estimate the error in $E(u)$ by computing estimates with finer and finer grids, using the finest one as an approximation to truth. We expect the error in the estimates to drop by a factor of 4 each time the mesh size is halved (since the error is proportional to h^2), and that is what we observe. The mesh of Figure 29.1 produces an energy estimate with estimated error less than 0.1; the resulting solution is shown in Figure 29.2.

CHALLENGE 29.2. We set up the Lagrangian function

$$L(x, y, \lambda) = (x - z_1)^2 + (y - z_2)^2 - \lambda \left(\left(\frac{x}{\alpha}\right)^2 + \left(\frac{y}{\beta}\right)^2 - 1 \right),$$

where the scalar λ is the Lagrange multiplier for the constraint. Setting the three partial derivatives to zero yields

$$\begin{aligned} 2(x - z_1) - 2\lambda \frac{x}{\alpha^2} &= 0, \\ 2(y - z_2) - 2\lambda \frac{y}{\beta^2} &= 0, \\ \left(\frac{x}{\alpha}\right)^2 + \left(\frac{y}{\beta}\right)^2 - 1 &= 0. \end{aligned}$$

We conclude that

$$x = \frac{\alpha^2 z_1}{\alpha^2 - \lambda}, \tag{29.1}$$

$$y = \frac{\beta^2 z_2}{\beta^2 - \lambda}, \tag{29.2}$$

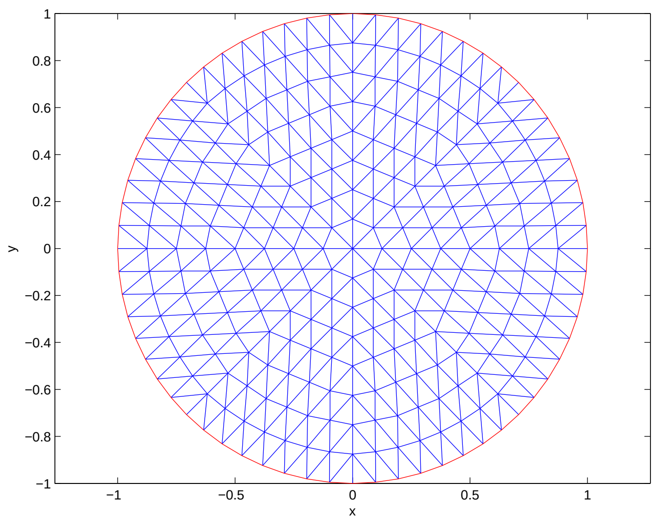


Figure 29.1. *Mesh used for a circular cross-section.*

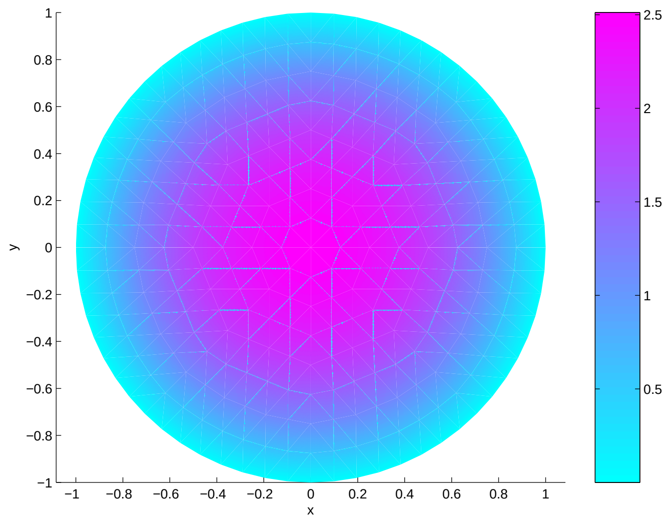


Figure 29.2. *Solution for the elastic model using a circular cross-section.*

as long as the denominators are nonzero. Since $|x| \le \alpha$ and $|y| \le \beta$, we conclude that the solution we seek has λ satisfying $0 \le \lambda \le \min(\alpha^2, \beta^2)$. So we can solve our problem by solving the nonlinear equation

$$f(\lambda) = \left(\frac{x}{\alpha}\right)^2 + \left(\frac{y}{\beta}\right)^2 - 1 = 0$$

using (29.1) and (29.2) to define $x(\lambda)$ and $y(\lambda)$.

These formulas fail when $z_1 = 0$ or $z_2 = 0$. There are two points to check, depending on whether it is shorter to move horizontally or vertically to the boundary. When $\boldsymbol{z} = \mathbf{0}$, for example, then the solution is either $(x, y) = (0, \beta)$ or $(\alpha, 0)$, depending on whether β or α is smaller. Full details are given in the sample program for Challenge 3 and also in a description by David Eberly [1].

CHALLENGE 29.3. A sample program appears on the website as `dist_to_ellipse.m`. The testing program plots the distances on a grid of points in the ellipse. Note that it is important to test points that are near zero. To validate the program, we might repeat the runs with various values of α and β, and also test the program for a point bfz outside the ellipse.

CHALLENGE 29.4. The results are shown in Figures 29.3 and 29.4, created with a program on the website. The meshes we used had the same refinement as that determined for the circular domain of Challenge 1. A sensitivity analysis should be done by refining the mesh once to see how much the solution changes in order to obtain an error estimate.

Note that it would be more computationally efficient to take advantage of the sequence of problems being solved by using the solution at the previous value of $\alpha\theta$ as an initial guess for the next value. See Chapter 24 for more information on such **continuation methods**.

[1] David Eberly, Distance from a Point to an Ellipse in 2D, Magic Software, Inc. www.magic-software.com/Documentation/DistanceEllipse2Ellipse2.pdf

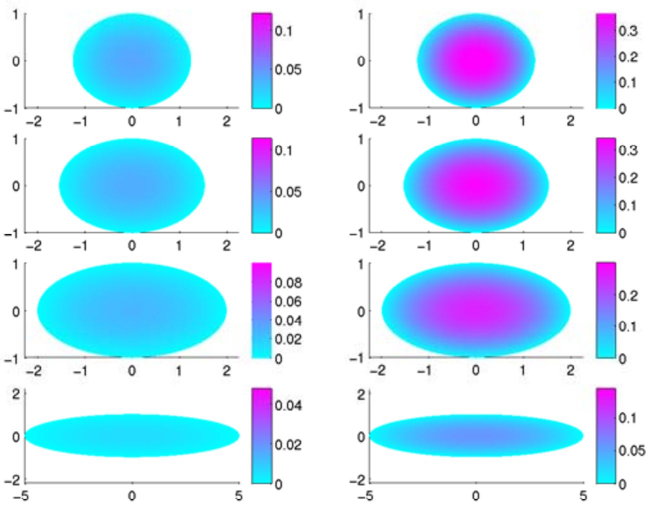


Figure 29.3. *Elasto-plastic solutions for various cross-sections. On the left, $\alpha\theta = 0.5$; on the right, $\alpha\theta = 1.0$.*

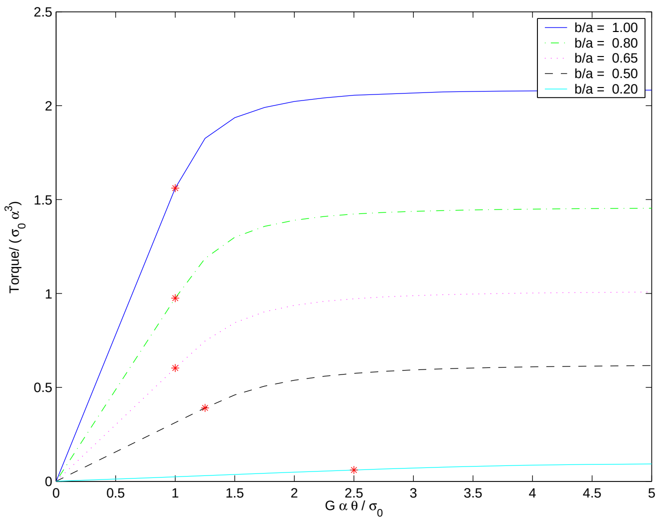


Figure 29.4. *Torque computed for various cross-sections as θ is increased. The red stars mark the boundary between elastic solutions and elastoplastic solutions.*

Chapter 30

Solutions: Case Study: Fast Solvers and Sylvester Equations: Both Sides Now

CHALLENGE 30.1. Equating the (j,k) element on each side of the equation

$$(\boldsymbol{B}_y \boldsymbol{U} + \boldsymbol{U}\boldsymbol{B}_x) = \boldsymbol{F},$$

we obtain

$$f(x_j, y_k) = \frac{1}{h^2}(-u(x_{j-1}, y_k)+2u(x_j, y_k)-u(x_{j+1}, y_k)-u(x_j, y_{k-1})+2u(x_j, y_k)-u(x_j, y_{k+1})),$$

which is the same as equation $(k-1)n + j$ of

$$(\boldsymbol{A}_x + \boldsymbol{A}_y)\boldsymbol{u} = \boldsymbol{f}.$$

CHALLENGE 30.2.

(a) Using MATLAB notation for subvectors, the algorithm is:

```
for $i = 1 : n$,
    for $j = 1 : n$,
        $U(i,j) = (C(i,j) - L(i, 1 : i-1) * U(1 : i-1, j)$
            $-U(i, 1 : j-1) * R(1 : j-1, j))/(L(i,i) + R(j,j))$
    end
end
```

The number of multiplications is

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(i-1+j-1) = n^2(n-1),$$

and the other operations are also easy to count.

(b) The algorithm fails if $L(i,i) + R(j,j) = 0$ for some value of i and j. The main diagonal elements of triangular matrices are the eigenvalues of the matrix, so it is necessary and sufficient that $\boldsymbol{L}$ and $-\boldsymbol{R}$ have no common eigenvalues.

(c) If $\boldsymbol{AU} + \boldsymbol{UB} = \boldsymbol{C}$, then

$$\boldsymbol{WLW}^* \boldsymbol{U} + \boldsymbol{UYRY}^* = \boldsymbol{C}.$$

Multiplying on the left by $\boldsymbol{W}^*$ and on the right by $\boldsymbol{Y}$, we obtain $\boldsymbol{L}\widehat{\boldsymbol{U}} + \widehat{\boldsymbol{U}}\boldsymbol{R} = \widehat{\boldsymbol{C}}$.

CHALLENGE 30.3. The algorithm of Challenge 2(a) reduces to $U(i,j) = F(i,j)/(L(i,i) + R(j,j))$ for $i,j = 1,\ldots,n$, which requires n^2 additions and divisions.

CHALLENGE 30.4.

(a) Recall the identities

$$\sin(a \pm b) = \sin a \cos b \pm \cos a \sin b.$$

If we form $\boldsymbol{B}_x$ times the jth column of $\boldsymbol{V}$, then the kth element is

$$\begin{aligned}
\frac{-v_{k-1,j} + 2v_{k,j} - v_{k+1,j}}{h^2} &= \frac{\alpha_j}{h^2}\left(-\sin\frac{(k-1)j\pi}{n+1} + 2\sin\frac{kj\pi}{n+1} - \sin\frac{(k+1)j\pi}{n+1}\right) \\
&= \frac{\alpha_j}{h^2}\left(-\sin\frac{kj\pi}{n+1}\cos\frac{j\pi}{n+1} + \cos\frac{kj\pi}{n+1}\sin\frac{j\pi}{n+1} + 2\sin\frac{kj\pi}{n+1}\right. \\
&\quad \left. -\sin\frac{kj\pi}{n+1}\cos\frac{j\pi}{n+1} - \cos\frac{kj\pi}{n+1}\sin\frac{j\pi}{n+1}\right) \\
&= \frac{\alpha_j}{h^2}\left(2 - 2\cos\frac{j\pi}{n+1}\right)\sin\frac{kj\pi}{n+1} \\
&= \frac{1}{h^2}\left(2 - 2\cos\frac{j\pi}{n+1}\right)v_{k,j} \\
&= \lambda_j v_{k,j}.
\end{aligned}$$

Stacking these elements we obtain $\boldsymbol{B}_x \boldsymbol{v}_j = \lambda_j \boldsymbol{v}_j$.

(b) This follows by writing the kth component of $\boldsymbol{Vy}$.

CHALLENGE 30.5. See the website for the programs. The results are shown in Figures 30.1 and 30.2. All of the algorithms give accurate results, but as n gets large, the efficiency of the fast algorithm of Challenge 4 becomes more apparent.

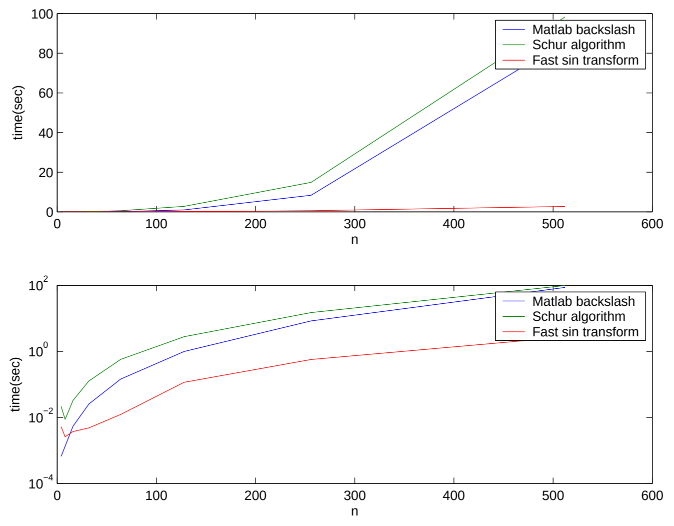


Figure 30.1. *The time (seconds on a Sun UltraSPARC-III with clock speed 750 MHz running MATLAB 6) taken by the three algorithms as a function of n. The bottom plot uses logscale to better display the times for the fast sine transform.*

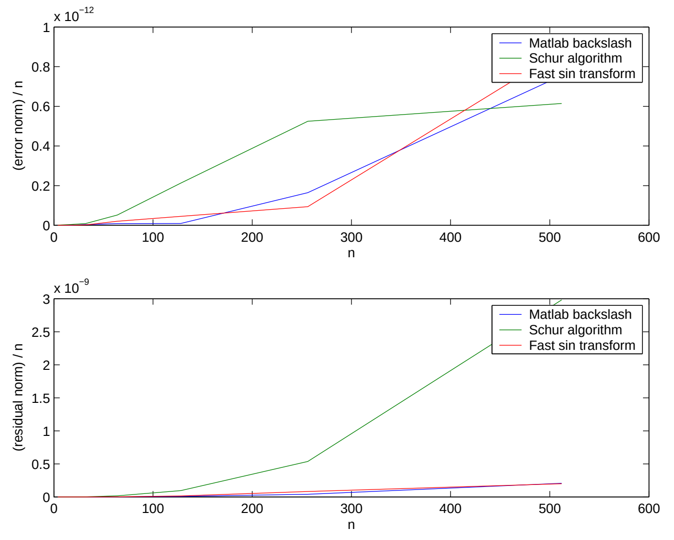


Figure 30.2. *The accuracy of the three algorithms as a function of n.*

Chapter 31

Solutions: Case Study: Eigenvalues: Valuable Principles

CHALLENGE 31.1. We can verify by direct computation that w_m satisfies the boundary condition and that $-\partial^2 w_{m\ell}/\partial x^2 - \partial^2 w_{m\ell}/\partial y^2$ is $(m^2+\ell^2)\pi^2/b^2$ times $w_{m\ell}$, so $\lambda_{m\ell} = (m^2+\ell^2)\pi^2/b^2$.

CHALLENGE 31.2.

(a) The eigenvalues are

$$\lambda_{jk} = \frac{j^2+k^2}{4}\pi^2$$

for $j, k = 1, 2, \ldots$. One expression for the eigenfunction is

$$v_{jk} = \sin(j\pi(x+1)/2)\sin(k\pi(y+1)/2).$$

This is not unique, since some of the eigenvalues are multiple. So, for example, $\lambda_{12} = \lambda_{21}$, and any function $av_{12} + bv_{21}$, for arbitrary scalars a and b, is an eigenfunction. Even for simple eigenvalues, the function v_{jj} can be multiplied by an arbitrary constant, positive or negative.

The first six v_{jk} are plotted in Figure 31.1, and it is an interesting exercise to describe them in words. Note that as the eigenvalue increases, the number of oscillations in the eigenfunction increases. In order to capture this behavior in a piecewise linear approximation, we need a finer mesh for the eigenfunctions corresponding to larger eigenvalues than we do for those corresponding to smaller eigenvalues.

(b) When using piecewise linear finite elements, the jth computed eigenvalue lies in an interval $[\lambda_j, \lambda_j + C_j h^2]$, where h is the mesh size used in the triangulation. This is observed in our computation using the program `problem1b.m`, found on the website. The error plots are shown in Figure 31.2. The horizontal axis is the

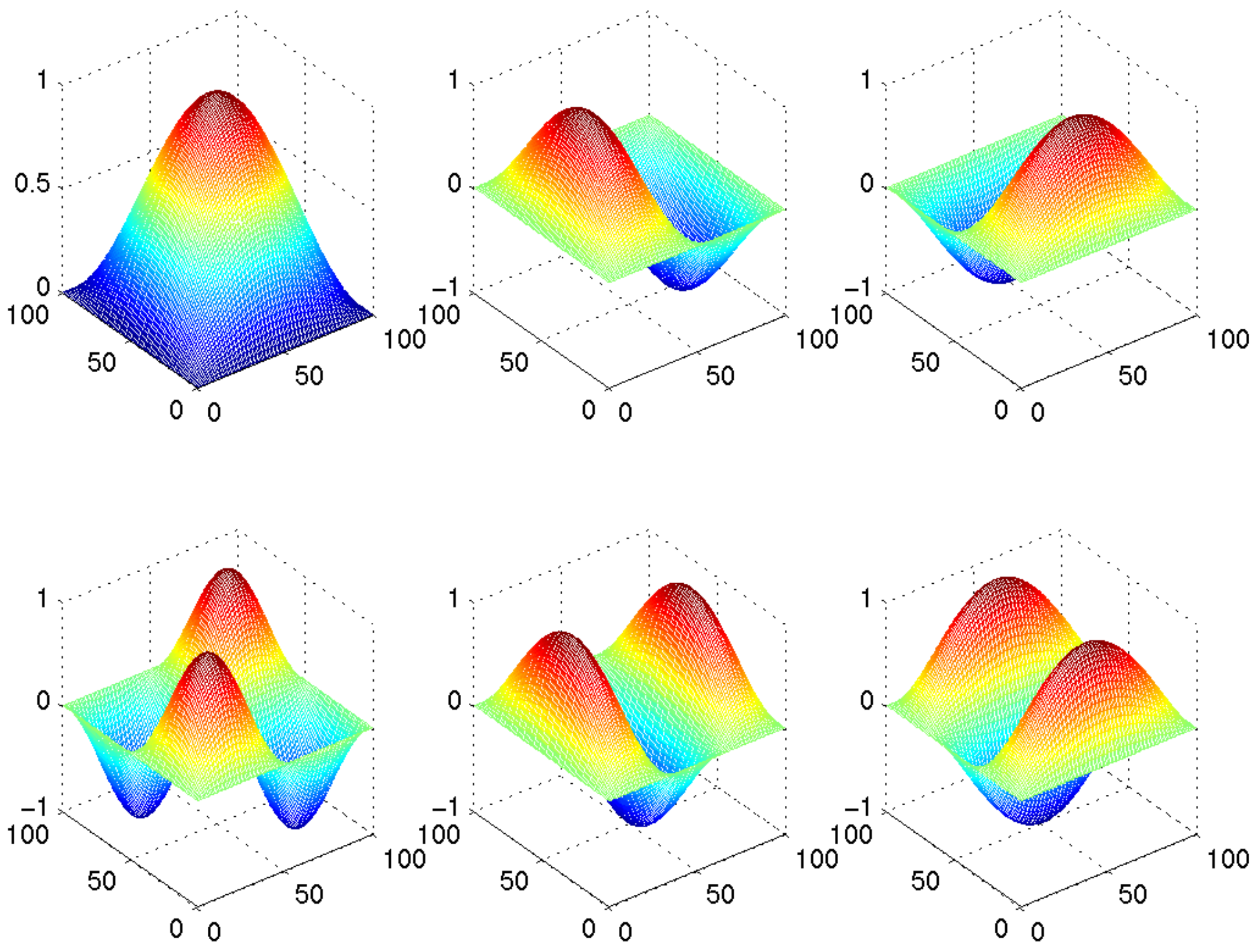


Figure 31.1. *Eigenfunctions corresponding to the eigenvalues* $\lambda = 4.9348, 12.3370, 12.3370$ *(top row) and* $\lambda = 19.7392, 24.6740, 24.6740$ *(bottom row).*

number of triangles, which is approximately proportional to $1/h^2$. The errors in the approximate eigenvalues are as follows:

λ_j	Mesh1	Mesh 2	Mesh 3	Mesh 4
$j = 1$	5.03e-02	1.27e-02	3.20e-03	8.02e-04
$j = 6$	1.29e+00	3.25e-01	8.15e-02	2.04e-02
$j = 11$	4.17e+00	1.04e+00	2.60e-01	6.50e-02
$j = 16$	8.54e+00	2.12e+00	5.28e-01	1.32e-01
$j = 21$	1.49e+01	3.67e+00	9.15e-01	2.29e-01

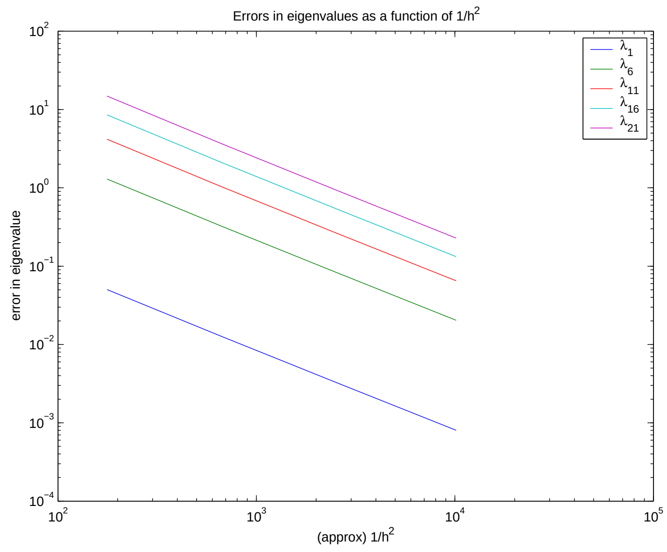


Figure 31.2. *The errors in the eigenvalue approximations.*

The error ratios are as follows:

$lambda_j$	Mesh 1 vs. 2	Mesh 2 vs. 3	Mesh 3 vs. 4
$j = 1$	3.95e+00	3.98e+00	3.99e+00
$j = 6$	3.98e+00	3.98e+00	3.99e+00
$j = 11$	4.02e+00	4.00e+00	4.00e+00
$j = 16$	4.04e+00	4.01e+00	4.00e+00
$j = 21$	4.05e+00	4.01e+00	4.00e+00

Therefore, the error is reduced by a factor of 4 as the side of each triangle is reduced by a factor of 2, so the error is $O(h^2)$, as expected, but the larger the eigenvalue, the finer the mesh necessary to achieve a given accuracy.

CHALLENGE 31.3.

(a) Suppose (for convenience of notation) that $\Omega \subset \mathcal{R}^2$. (Other dimensions are just as easy.) First we apply integration by parts (with zero boundary conditions) to see that if $w \neq 0$

$$\begin{aligned}(w, \mathcal{A}w) &= -\int\int_\Omega w\nabla \cdot (\boldsymbol{a}\nabla w)\mathrm{d}x\mathrm{d}y \\ &= \int\int_\Omega \nabla w \cdot (\boldsymbol{a}\nabla w)\mathrm{d}x\mathrm{d}y \\ &= \int\int_\Omega a_1(x,y)\left(\frac{\partial w}{\partial x}\right)^2 + a_2(x,y)\left(\frac{\partial w}{\partial y}\right)^2 \mathrm{d}x\mathrm{d}y \\ &\geq 0,\end{aligned}$$

since $a_1(\boldsymbol{x}), a_2(\boldsymbol{x}) > 0$.

Suppose $\mathcal{A}w = \lambda w$. Then

$$0 \leq (w, \mathcal{A}w) = \lambda(w, w),$$

so $\lambda \geq 0$.

(b) We know that

$$\lambda_1(\Omega) = \min_{w \neq 0} \frac{(w, \mathcal{A}w)}{(w, w)}$$

where the integrals are taken over Ω and w is constrained to be zero on the boundary of Ω. Suppose that the w that minimizes the function is $\tilde{w}$ and let's extend $\tilde{w}$ to make it zero over the part of $\tilde{\Omega}$ not contained in Ω. Then

$$\lambda_1(\tilde{\Omega}) = \min_{w \neq 0} \frac{(w, \mathcal{A}w)}{(w, w)} \leq \frac{(\tilde{w}, \mathcal{A}\tilde{w})}{(\tilde{w}, \tilde{w})} = \lambda_1(\Omega).$$

CHALLENGE 31.4. From Challenge 1, we know that the smallest eigenvalue for a square with dimension b is $2\pi^2/b$, so we want $b = \sqrt{2}/2$. Using MATLAB's PDE Toolbox interactively, we discover that $\alpha \approx 1.663$.

Chapter 32

Solutions: Multigrid Methods: Managing Massive Meshes

CHALLENGE 32.1. The V-cycle performs the following operations:

η_1 Gauss–Seidel iterations for $h = 1/16$:	$15\eta_1$ multiplications (by $h^2/2$)
$h = 1/16$ residual evaluation	16 multiplications
Multiplication by $R_{1/8}$	14 multiplications
η_1 Gauss–Seidel iterations for $h = 1/8$:	$7\eta_1$ multiplications
$h = 1/8$ residual evaluation	8 multiplications
Multiplication by $R_{1/4}$	6 multiplications
η_1 Gauss–Seidel iterations for $h = 1/4$:	$3\eta_1$ multiplications
$h = 1/4$ residual evaluation	4 multiplications
Multiplication by $R_{1/2}$	2 multiplications
Direct solution of the system for $h = 1/2$:	1 multiplication
Multiplication by $P_{1/2}$	2 multiplications
η_2 Gauss–Seidel iterations for $h = 1/4$:	$3\eta_2$ multiplications
Multiplication by $P_{1/4}$	6 multiplications
η_2 Gauss–Seidel iterations for $h = 1/8$:	$7\eta_2$ multiplications
Multiplication by $P_{1/8}$	14 multiplications
η_2 Gauss–Seidel iterations for $h = 1/16$:	$15\eta_2$ multiplications

The total cost is less than the cost of $2(\eta_1 + \eta_2)$ iterations, plus 2 residual calculations, all done on the $h = 1/16$ grid, plus 4 multiplications by $R_{1/8}$.

CHALLENGE 32.2. The right-hand sides have

$$15 + 7 + 3 + 1 = 16(1 + 1/2 + 1/4 + 1/8) - 4,$$

elements, which is less than twice the storage necessary for the right-hand side for the finest grid. The same is true for the solution vectors. Similarly, each matrix $\boldsymbol{A}_h$ has at most half of the number of nonzeros of the one for the next finer grid, so the total matrix storage is less than 2 times that for $\boldsymbol{A}_{1/16}$.

The matrices $\boldsymbol{P}_h$ can be stored as sparse matrices or, since we only need to form their products with vectors, we can just write a function to perform multiplication without explicitly storing them.

CHALLENGE 32.3. In Chapter 27 we saw that the fastest algorithm for the finest grid for `myproblem=1` was the AMD-Cholesky algorithm, which, on my computer, took about 0.2 seconds and storage about 5 times that for the matrix. The fastest iterative method, conjugate gradients with an incomplete Cholesky preconditioner, took 0.9 seconds. My implementation of multigrid for this problem took 4 iterations and 8.2 seconds. The virtue of multigrid, though, is if we want a finer grid, we will probably still get convergence in about 4 iterations, while the number of iterations of the other algorithms increases with h, so eventually multigrid will win.

CHALLENGE 32.4. The number of iterations remained 4 for $\kappa = 10$, 100, and -10, but for $\kappa = -100$, multigrid failed to converge. As noted in the challenge, a more complicated algorithm is necessary.

Note that the function `smooth.m` is a much faster implementation of Gauss–Seidel than that given in the solution to Challenge 28.6 in Chapter 28.
