Iterative Methods
MANUSCRIPT

William F. Barnes
[1]

March 6, 2024

[1]Copyright © 2014-2024 by William F. Barnes. All rights reserved. Unauthorized retention, duplication, distribution, or modification is not permitted.

Contents

Chapter 1

Iterative Methods

1 Matrix Tools

1.1 Linear Systems Review

The linear system $A\vec{x} = \vec{b}$ is ubiquitous in multivariate systems, and rears its face in a majority of numerical analysis applications.

We'll restrict analysis to systems that are can actually be solved, which is to say the system is determined by n equations and n unknowns, corresponding to a square matrix A with n rows and n columns.

Calculating the Inverse

Often, we're interested in 'solving' for $\vec{x}$, which requires having the inverse of A, namely A^{-1} such that

$$\left(A^{-1}A\right)\vec{x} = \vec{x} = A^{-1}\vec{b}\,.$$

Pursuing a popular technique, define the sub-matrix S_{jk} that removes the jth row and the kth column from the original matrix A. (The width and height of S are each one less than those of A.) Define the matrix minor M_{jk} as the determinant of S_{jk}.

Also, let there be another matrix B that relates to A via

$$B_{jk} = (-1)^{j+k}\,M_{kj}\,.$$

That is, the jkth component of the matrix B is equal to a constant times the entire determinant of the S_{kj} sub-matrix.

The reason for defining B this way is that the product AB equals the determinant of A multiplied by the identity matrix:

$$AB = (\det A)\,I$$

Multiply A^{-1} into each side to find a tight formula for the inverse:

$$A^{-1} = \frac{1}{\det A}B$$

Calculating the RREF

A second way for determining the components x_j involves the augmented matrix

$$A|b = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \cdots & A_{1n} & b_1 \\ A_{21} & A_{22} & A_{23} & \cdots & A_{2n} & b_2 \\ A_{31} & A_{32} & A_{33} & \cdots & A_{3n} & b_3 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ A_{n1} & A_{n2} & A_{n3} & \cdots & A_{nn} & b_n \end{bmatrix},$$

which 'tacks on' the vector $\vec{b}$ to the right side of A so the final object has n rows and $n+1$ columns.

The augmented matrix permits three row operations: (i) exchange of two rows, (ii) multiply a row by a scalar, (iii) replace a row by the sum of itself and another row.

Using row operations, it's possible to bring the augmented matrix $A|b$ into upper triangular form:

$$A|b = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \cdots & A_{1n} & b_1 \\ 0 & A_{22} & A_{23} & \cdots & A_{2n} & b_2 \\ 0 & 0 & A_{33} & \cdots & A_{3n} & b_3 \\ 0 & 0 & 0 & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & A_{nn} & b_n \end{bmatrix}$$

Keep in mind the components A_{jk}, b_j are not the same between t.he original and its transformed version.

Starting with the upper triangular form, a the matrix A is reduced again by similar steps to bring about the lower triangular form. This reduces A to having only diagonal entries:

$$A|b = \begin{bmatrix} A_{11} & 0 & 0 & 0 & 0 & b_1 \\ 0 & A_{22} & 0 & 0 & 0 & b_2 \\ 0 & 0 & A_{33} & 0 & 0 & b_3 \\ 0 & 0 & 0 & \cdots & 0 & \cdots \\ 0 & 0 & 0 & 0 & A_{nn} & b_n \end{bmatrix}$$

Finally, attain the row-reduced echelon form by normalizing the jth row by A_{jj} to arrive at the form $I|x$, where I is the identity matrix.

To summarize, the process

$$A|b \;\rightarrow\; I|x$$

exposes the components of $\vec{x}$ as the right-most column of the augmented matrix $I|x$. This, of course, is Gaussian elimination.

Gauss-Jordan Elimination

Remarkably, row operations can also be used to calculate the inverse of a matrix. For an $n \times n$ matrix A, it turns out that the augmented matrix $A|I$, having n rows and $2n$ columns, may undergo the same treatment as $A|b$. That is, perform row operations to attain the transformation:

$$A|I \;\rightarrow\; I|A^{-1}$$

This is Gauss-Jordan elimination.

1.2 Pseudocode

One type of notation we'll employ here is *pseudocode*, which attempts to bridge a computer algorithm to human readability.

Pseudocode is read sequentially (top to bottom), paying particular mind to variables, conditions, and so on, in order to understand what a computer would do if the code were implemented in a proper language. Instructions that are indented are contained in a function, loop, or conditional. Instructions that end with an underscore (_) are continued to the next line.

For example, the contents of a matrix A can be replaced by the contents of another matrix B by the following pseudocode:

```
# Requires matrix B
# Returns matrix A

function Equate(B)
  for each j in rows of B
    for each k in columns of B
      A(j, k) = B(j, k)
  return A
```

This process requires matrices A and B to be of the same dimension. Note how the 'inner' loop goes over the columns of B and the 'outer' loop goes by row, which is analogous to reading text on a page. Of course, there are many more ways to replace the j,kth component of matrix B, such as running the loops backwards.

1.3 Augmenting

For two matrices A and B having the same number of rows but no restriction of the number of columns, the augmented matrix $A|B$ is attained by the following:

```
# Requires A, B (same no. of rows)
# Returns AB

function Augment(A, B)
  r = rows in A or B
  c1 = number of columns in A
  c2 = number of columns in B
  for j from 1 to r
    for k from 1 to c1
      AB(j, k) = A(j, k)
    for k from 1 to c2
      AB(j, k + c1) = B(j, k)
  return AB
```

1.4 Matrix Multiplication

A matrix A having J rows and N columns multiplies into another matrix B with N rows and K columns to give a new matrix C with J rows and K columns. The jkth component of the resulting matrix is given by:

$$C_{jk} = \sum_{n=1}^{N} A_{jn} B_{nk}$$

Of course, there are $j \times k$ calculations like this to do, which is a dauntless endeavor for a computer. As pseudocode, the whole matrix C is can calculated by:

```
# Requires A size J*N
# Requires B size N*K
# Returns C size JK

function Multiply(A, B)
  for each j in rows of A
    for each k in columns of B
      for each n in columns of A
        C(j, k) = C(j, k) _
                  + A(j, n) * B(n, k)
  return C
```

1.5 Sub-Matrix

For a matrix A, the sub-matrix S_{jk} that removes the jth row and kth column is established by the following.

```
# Requires matrix A size J*K
# Requires Row j, Column k
# Returns matrix S size (J-1)(K-1)

function SubMatrix(A, j, k)
  x = 0
  for each u in rows of A
    if (u <> j)
      y += 1
      x = 0
      for each v in columns of A
        if (v <> k)
          x += 1
          S(x, y) = A(u, v)
  return S
```

1.6 Determinant

For the square matrix

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} ,$$

the determinant of A is

$$\det A = A_{11}A_{22} - A_{12}A_{21} .$$

The most trivial matrix has $A = A_{11}$, in which case $\det A = A_{11}$.

For an $n \times n$ matrix, the determinant can be written from a recursive equation:

$$\det A = \sum_{\substack{j \le n \\ k=1}}^{n} (-1)^{j+k} A_{jk} M_{jk}$$

Recall that M_{jk} is the determinant of the matrix minor S_{jk}, hence the recursive nature of the above.

The matrix determinant calculation is represented in the pseudocode below. Importantly, note that certain environments have trouble making perfect sense of $(-1)^{j+k}$. For this, the instruction `if (k MOD 2) = 0` is used instead to achieve the same ends, where `MOD` is the 'modulus' operator.

```
# Requires matrix A size n^2
# Returns Det(A)

function Determinant(A)
  n = side(A)
  if n = 1: det = A(1, 1)
  if n = 2: det = A(1, 1) * A(2, 2) _
                - A(1, 2) * A(2, 1)
  if n > 2
    define matrix S size (n-1)x(n-1)
    for k from 1 to n
      S = SubMatrix(A, 1, k)
      m = Det(S)
      if (k MOD 2) = 0
        m = -m
      det = det + A(1, k) * m
  return det
```

1.7 Inverse Matrix

For an invertible matrix A, recall that the inverse A^{-1} is calculated by

$$A^{-1} = \frac{1}{\det A} B ,$$

where the components of B depend on the sub-matrix minors M_{kj} as detailed above.

The pseudocode below steps through the inverse calculation and stores the result as `Inv`. Take note of the `MOD` operator performing a similar role as it does in the determinant calculation.

```
# Requires matrix A size n^2
# Returns inverse of A

function Inverse(A)
  define matrix S size (n-1)x(n-1)
  detA = Determinant(A) # nonzero
  for each j in rows of A
    for each k in columns of A
      S = Submatrix(A, j, k)
      m = Determinant(S)
      if ((j + k) MOD 2) = 1
        m = -1 * m
      Inv(j, k) = m / detA
  return Inv
```

1.8 Augmented Upper Triangular

The augmented matrix $A|b$ with n rows and $n+1$ columns can be reduced to the upper triangular form (same dimensions) by the pseudocode below.

```
# Requires Ab size n*m
# Returns upper triangular

function UT(Ab)
  n = number of rows in Ab
  m = number of columns in Ab
  for j from 1 to n
    for k from (j+1) to n
      f = Ab(k, j) / Ab(j, j)
      for p from 1 to m
        Ab(k, p) = Ab(k, p) _
                   - f * Ab(j, p)
  return Ab
```

For a bonus, this method also works for the $A|I \to I \to A^{-1}$ system with n rows and $2n$ columns. The structures b and I are interchangeable.

1.9 Augmented Lower Triangular

Modifying the previous example with surgical edits to the loop limits and step direction give a way to compute the lower triangular form of $A|b$ or $A|I$:

```
# Requires Ab size n*m
# Returns lower triangular

function LT(Ab)
  n = number of rows in Ab
  m = number of columns in Ab
  for j from n to 2 step -1
    for k from (j-1) to 1 step -1
      f = Ab(k, j) / Ab(j, j)
      for p from 1 to m
        Ab(k, p) = Ab(k, p) _
                   - f * Ab(j, p)
  return Ab
```

Note that in the pursuit of $A|I \to I|A^{-1}$, the upper triangular and lower triangular calculations can be done in any order.

1.10 RREF Matrix

The augmented system $A|B$ can be wrestled into RREF format by the pseudocode that follows:

```
# Requires Ab size n*m
# Returns R(REF)

function RREF(AB)
  n = number of rows in AB
  m = number of columns in AB
  rref = LT(UT(AB))
  # Normalize:
  for j from 1 to n
    for k from n + 1 to m
      R(j, k) = R(j, k) / R(j, j)
    R(j, j) = R(j, j) / R(j, j) # = 1
  return rref
```

2 Approximating Integrals

2.1 Riemann Sums

The predecessor to the notion of the integral and the fundamental theorem of calculus is the Riemann sum, which relates the endpoint values of a function $f(x)$ to a sum over the function's slope by

$$f(x_n) - f(x_0) \approx S = \sum_{j=0}^{n-1} f'\left(x_j^*\right) \Delta x \,,$$

where

$$\Delta x = \frac{x_n - x_0}{n} \,,$$

and x_j^* is any x-value within $[x_j, x_{j+1}]$, and

$$x_j = x_0 + j\Delta x \; .$$

The dimensionless integer j is the index, and n is the total number of bins in the sum.

Since there is freedom in how x_j^* is chosen, there are three standard methods called the left sum, right sum, and midpoint sum:

$$x_j^* = \begin{cases} x_j & \text{Left sum} \\ x_{j+1} & \text{Right sum} \\ \left(x_j + x_{j+1}\right)/2 & \text{Midpoint sum} \end{cases}$$

Denoting S_{Left}, S_{Right} as the left and right sums respectively, the average of these yields the trapezoid rule:

$$S_{\text{Trap}} = \frac{1}{2}\left(S_{\text{Left}} + S_{\text{Right}}\right)$$

Of course, all Riemann sums are the same in the continuous limit, which is why the integral need not concern over left, right, mid, etc.

2.2 Simpson's Rule

For approximating the area under a function $f(x)$, an improvement over straight-line methods uses a quadratic function to estimate $f(x)$ at each step, known as *Simpson's rule.* To get started, propose a quadratic form

$$g(x) = Ax^2 + Bx + C \; ,$$

where the coefficients A, B, C depend on $f(x)$ in the neighborhood of x.

Now consider a point x_j somewhere in the region and write the definite integral

$$\int_{x_j-h}^{x_j+h} f(x)\,dx \approx \int_{x_j-h}^{x_j+h} g(x)\,dx = I(h) \; .$$

Without filling in the details yet, the result of such an integral is written $I(h)$, where $2h$ is the width of the integration domain. Substituting $g(x)$ into the above and turning the crank gives the form

$$I(h) = \frac{2h}{3}\left(A\left(3x_j^2 + h^2\right) + 3Bx_j + 3C\right) \; .$$

Meanwhile, examine a new quantity

$$J(h) = g(x_j - h) + 4g(x_j) + g(x_j + h) \; ,$$

which, after substituting $g(x)$, becomes

$$J(h) = \frac{3}{h} I(h) \; .$$

Evidently, the integral $I(h)$ is the same as the sum $J(h)$ up to a factor $3/h$:

$$\begin{aligned} \int_{x_j-h}^{x_j+h} f(x)\,dx &\approx \int_{x_j-h}^{x_j+h} g(x)\,dx \\ &= \frac{h}{3}\left(g(x_j-h) + 4g(x_j) + g(x_j+h)\right) \end{aligned}$$

Of course, this result only works in the neighborhood on a given x_j.

To apply this over a macroscopic interval, sum over all x_j in steps $2h$, and let $f(x)$ replace the function being evaluated. The integration region is given by

$$\frac{b-a}{n} = 2h \; ,$$

where a, b are the lower and upper limits, and n is the number of bins. The effective bin width is $2h$. In order to have $x_0 - h = a$ and $x_{n-1} + h = b$, the x_j are located via

$$\begin{aligned} x_j &= (a+h) + \frac{j}{n}(b-a) \\ x_j &= (a+h) + j(2h) \; . \end{aligned}$$

Assimilating these changes, the approximation becomes

$$\begin{aligned} &\int_a^b f(x)\,dx \approx \qquad (1.1) \\ &\quad \sum_{j=0}^{n-1} \frac{h}{3}\left(f(x_j-h) + 4f(x_j) + f(x_j+h)\right) \; , \end{aligned}$$

which we may take as a final answer.

Pseudocode

As pseudocode, the Equation (1.1) can be implemented shown in the box that follows. The area being approximated is under the function $f(x) = 4x - x^2$ in the region $0 \le x \le 4$ using 15 bins. Lines of code that are indented by two spaces are 'looped over'.

```
f(x) = 4 * x - x * x

a = 0   # lower limit
b = 4   # upper limit
n = 15  # bins
h = (b - a) / (2 * n)
# Initialize other variables to zero.

for j from 0 to n - 1
  xj = (a + h) + j * (2 * h)
  f1 = f(xj - h)
  f2 = f(xj)
  f3 = f(xj + h)
  simp += (h / 3) * (f1 + 4 * f2 + f3)
```

The approximation to the integral is held in the `simp` variable. If the above pseudocode were implemented in a suitable computation environment, one would find:

```
simp = 10.666666666666666
```

This result is indistinguishable from the exact answer to standard computation precision:

$$\int_0^4 \left(4x - x^2\right) = \frac{32}{3} = 10.666\overline{6}$$

Weighted Average Identity

Starting with Equation (1.1), for Simpson's rule identify $2h = \Delta x$ and keep simplifying:

$$\int_a^b f(x)\,dx \approx$$
$$\frac{1}{6}\sum_{j=0}^{n-1}\left(f\left(x_j - \frac{\Delta x}{2}\right) + f\left(x_j + \frac{\Delta x}{2}\right)\right)\Delta x$$
$$\frac{1}{6}\sum_{j=0}^{n-1} 4f(x_j)\,\Delta x\,.$$

The sum has been broken in two parts. The first consists of the left sum S_L and right sum S_R rules added together, which is twice the trapezoid rule S_T. The final sum involving $f(x_j)$ alone is identical to the midpoint sum S_M. Evidently, Simpson's rule is the weighted average of more elementary methods after all:

$$\int_a^b f(x)\,dx \approx \frac{1}{3}\left(S_T + 2S_M\right) \tag{1.2}$$

Setting up another program to approximate the left, right, and midpoint sums, we can also get the trapezoid sum and verify that Simpson's rule obeys the above identity. For an example problem, let us approximate the area under a curve we can verify by hand:

$$\int_{-2}^{3}\left(5x^2 - x\right)dx = \frac{335}{6} = 55.8333\overline{3}$$

Then, making appropriate changes to the above program, we have:

```
f(x) = 5 * x * x - x

a = -2  # lower limit
b = 3   # upper limit
n = 15  # bins
dx = (b - a) / n
# Initialize other variables to zero.

for j from 0 to n - 1
  xj = a + j * dx
  x1 = xj + dx
  xm = (xj + x1) / 2
  left += dx * f(xj)
  right += dx * f(x1)
  mid += dx * f(xm)

# Calculate trap and simp after loop
trap = (left + right) / 2
simp = (1 / 3) * (trap + 2 * mid)
```

With the number of bins set to $n = 15$, the results of such a program turn out as:

```
left  = 52.96296296296295
right = 59.62962962962900
mid   = 55.60185185185184
trap  = 56.29629629629628
simp  = 55.83333333333332
```

Comparing each of these to the exact answer, we see the left sum underestimating, the right sum overestimating, and so on. Of course, the trailing digits in each may vary slightly, depending on the environment used. Most notably, Simpson's rule seems to get the answer (to this integral) to near-perfect precision.

3 Regression Analysis

Regression analysis is an attempt to derive meaningful patterns from numerical data. To introduce the subject, we'll explore the scenario of fitting a curve $y = f(x)$ to a set of given data points $\{(x_j, y_j)\}$ in various ways.

3.1 Linear Fit

Suppose we're provided with the following set of ordered pairs:

x_j	0.6	1.8	2.8	3.6	4.2	5.6
y_j	1.6	1.6	2.6	2.0	4.0	3.6

Take each pair (x_j, y_j) with $j = 1, 2, \dots, n$ as a point in the Cartesian plane. Further, suppose there was reason to believe that the pattern in the provided points is described by a straight line in the plane

$$y = mx + b \, ,$$

where the slope m and y-intercept b are unknown, and to be found using the data provided.

To advance on the problem, move all variables to one side, and consider n instances of the equation

$$h_j\left(m, b\right) = m x_j + b - y_j \, .$$

That is, h_j measures the vertical distance between the point $mx_j + b$ and y_j. If the approximate line passes directly through (x_j, y_j), then $h_j\left(m, b\right) = 0$ for that point.

As defined, $h_j\left(m, b\right)$ could be a positive or a negative value, which would mean negative errors cancel out positive ones. To avoid this, let us work with the square of the vertical distance represented by h_j and call this a new function $F_j\left(m, b\right)$:

$$F_j\left(m, b\right) = \left(m x_j + b - y_j\right)^2$$

The total vertical distance from each point (x_j, y_j) to the line $y = mx + b$ is the sum of all F_j:

$$F\left(m, b\right) = \sum_{j=1}^{n} F_j\left(m, b\right) = \sum_{j=1}^{n} \left(m x_j + b - y_j\right)^2$$

Now comes the new idea. The ideal m and b for the data provided should correspond to a minimum in $F\left(m, b\right)$. That is, set the partial derivatives of F with respect to these variables to zero, and the correct m, b are implicated. We then have

$$\frac{\partial F}{\partial m} = 0 = \sum_{j=1}^{n} 2 x_j \left(m x_j + b - y_j\right)$$
$$\frac{\partial F}{\partial b} = 0 = \sum_{j=1}^{n} 2 \left(m x_j + b - y_j\right) \, .$$

To keep the algebra contained, define the quantity

$$X^\alpha Y^\beta = \sum_{j=1}^{n} x_j^\alpha y_j^\beta \, ,$$

which isn't treated as regular algebraic variable, for instance $(X)(X) \neq X^2$, and $(X)(Y) \neq XY$. Note for this example we have α, β never exceeding one.

In terms of the sums X, Y, etc., the minimization of $F\left(m, b\right)$ gives a system of two equations with two unknowns:

$$0 = mX^2 + bX - XY$$
$$0 = mX + bn - Y$$

The solution is straightforward using matrix methods or traditional:

$$m = \frac{(n)(XY) - (X)(Y)}{(n)(X^2) - (X)(X)}$$
$$b = \frac{(Y)\left(X^2\right) - (X)(XY)}{(n)(X^2) - (X)(X)}$$

To finish the example on hand, use a calculator to find $X = 18.6$, $Y = 15.4$, $XY = 55.28$, $X^2 = 73.4$. The final answer is:

$$m \approx 0.479$$
$$b \approx 1.082$$

3.2 Exponential Fit

Consider another set of points $\{(x_j, y_j)\}$ in the Cartesian plane that would be best approximated by an exponential fit

$$y = A\, e^{mx} \, ,$$

where the scaling constant A and exponential parameter m are to be determined form the data given.

For this problem, let $A = \ln(b)$, where b is another constant, and the equation becomes $y = e^{mx+b}$. Take the natural log of both sides to find

$$\ln(y) = mx + b \, .$$

From here, the problem is completely analogous to the straight-line fit, except all y_j are substituted for $\ln(y_j)$. One b is known, reverse the logarithm to solve for A.

3.3 Polynomial Fit

For any set of of n total points (x_j, y_j) in the Cartesian plane, we can try a polynomial of order $m < n$ to fit the data:

$$y = A_0 + A_1 x + A_2 x^2 + \cdots + A_m x^m$$

Similar to the straight-line fit, define a vertical distance $h_j\left(\{A_m\}\right)$ such that

$$h_j = A_0 + A_1 x_j + A_2 x_j^2 + \cdots + A_m x_j^m - y_j \, .$$

Square this distance and sum over all n data points:

$$F\left(\{A_m\}\right) = \sum_{j=1}^{n} h_j^2 = \sum_{j=1}^{n}\left(A_0 + A_1 x_j + \cdots + A_m x_j^m - y_j\right)^2$$

The best-fitting polynomial is the one that that minimizes F with respect to all A_j simultaneously. Writing these out, one finds

$$\frac{\partial F}{\partial A_0} = 2\sum_{j=1}^{n}\left(A_0 + A_1 x_j + \cdots + A_m x_j^m - y_j\right)$$
$$\frac{\partial F}{\partial A_1} = 2\sum_{j=1}^{n} x_j\left(A_0 + A_1 x_j + \cdots + A_m x_j^m - y_j\right)$$
$$\frac{\partial F}{\partial A_2} = 2\sum_{j=1}^{n} x_j^2\left(A_0 + A_1 x_j + \cdots + A_m x_j^m - y_j\right) ,$$

availing the pattern

$$\frac{\partial F}{\partial A_k} = 2\sum_{j=1}^{n} x_j^k\left(A_0 + A_1 x_j + \cdots + A_m x_j^m - y_j\right) ,$$

for any $k \leq m$.

Each derivative is zero on the left, and the universal factor of 2 drops out. Not forgetting to distribute the x_j^k term into each sum, all of the above information is best written in matrix notation. In particular, we have

$$M = \begin{bmatrix} n & X & X^2 & \cdots & X^m \\ X & X^2 & X^3 & \cdots & X^{m+1} \\ X^2 & X^3 & X^4 & \cdots & X^{m+2} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ X^m & X^{m+1} & X^{m+2} & \cdots & X^{2m} \end{bmatrix} ,$$

such that

$$M \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ \cdots \\ A_m \end{bmatrix} = \begin{bmatrix} Y \\ YX \\ YX^2 \\ \cdots \\ YX^m \end{bmatrix} .$$

There is a lot of information to juggle with if you're insane enough to do this by hand. Regardless, system can be solved by finding the row-reduced echelon form of:

$$\begin{bmatrix} n & X & X^2 & \cdots & X^m & Y \\ X & X^2 & X^3 & \cdots & X^{m+1} & YX \\ X^2 & X^3 & X^4 & \cdots & X^{m+2} & YX^2 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ X^m & X^{m+1} & X^{m+2} & \cdots & X^{2m} & YX^m \end{bmatrix}$$

4 Interpolation

In regression analysis, a set of n total data points $\{(x_j, y_j)\}$, leads to a best-fit polynomial (or other curve) that passes near, but not necessarily through each data point. By a more powerful process called *interpolation*, it's possible to find a curve that passes through each data point. With some effort, such a curve can be made continuous and smooth in its domain.

4.1 Rectangle Approximation

The crudest interpolation of the provided data points is the rectangular approximation, which draws a horizontal line for all $n-1$ points spanning from x_j to x_{j+1} such that

$$f_0\left(x_j \leq x < x_{j+1}\right) = y_j$$

Of course, we can also draw lines to the left instead of the right by a shift of index:

$$f_0\left(x_j \leq x < x_{j+1}\right) = y_{j+1}$$

4.2 Connect the Dots

A slightly more informative approximation to the provided data points is the linear interpolation, which is a fancy name for connect the dots. By standard straight line methods, successive data points are connect by lines given by

$$f_1(x) = y_j + \left(x - x_j\right)\left(\frac{y_{j+1} - y_j}{x_{j+1} - x_j}\right) .$$

This has the appearance of a first terms of a Taylor approximation and also the form $y = mx + b$. All of these are equivalent to linear order.

There is another way to express the line $f_1(x)$ that appears mighty peculiar at first:

$$f_1(x) = y_j\left(\frac{x_{j+1} - x}{x_{j+1} - x_j}\right) + y_{j+1}\left(\frac{x - x_j}{x_{j+1} - x_j}\right)$$

Make sure the two expressions for $f_1(x)$ are equivalent.

4.3 Quadratic Interpolation

The linear interpolation can be improved by including an x^2-like term in $f(x)$, giving a quadratic interpolation in terms of undermined coefficients:

$$f_2(x) = A_0 + A_1 x + A_2 x^2$$

One way to find the unknown coefficients is to pick three consecutive points, such as (x_j, y_j) with

$j = 0, 1, 2$. This generates three equations and three unknowns, which can be solved by standard means.

For a different approach to the problem, let us recite a trick named after Lagrange, which extends the 'peculiar' straight line method written above. The quadratic approximation is called the *Lagrange integrating polynomial*, and is given by

$$f_2(x) = y_0 L_0(x) + y_1 L_1(x) + y_2 L_2(x) \ ,$$

where the $L_j(x)$ are called the *Lagrange interpolating basis functions*:

$$\begin{aligned} L_0 &= \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} \\ L_1 &= \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} \\ L_2 &= \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} \end{aligned}$$

This can be straightforwardly reconciled with a traditional attack on the problem. In the original function $f_2(x)$, observe that A_2 is exactly one half of the second derivative of the function.

From ordinary calculus, we know the second derivative can be expressed via

$$f''(x) = \lim_{h\to 0} \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} \ .$$

This formula is a bit oversimplifying in the sense that h doesn't necessarily equal any given pair $x_j - x_k$, never mind the limit.

Proceed by differentiating $f_2(x)$ twice (only the x^2 term survives). To keep the algebra sane, define

$$\Delta x_{jk} = x_j - x_k \ ,$$

and we have

$$\frac{1}{2} f_2''(x) = \frac{y_0}{\Delta x_{01}\Delta x_{02}} + \frac{y_1}{\Delta x_{10}\Delta x_{12}} + \frac{y_2}{\Delta x_{20}\Delta x_{21}} \ .$$

Establish a common denominator and rewrite:

$$\frac{1}{2} f_2''(x) = \frac{y_0\Delta x_{12} - y_1\Delta x_{02} + y_2\Delta x_{01}}{\Delta x_{01}\Delta x_{02}\Delta x_{12}}$$

This is technically as far as the comparison can go. To an approximation though, we can have

$$h = x_1 - x_0 = x_2 - x_1 = \frac{x_2 - x_0}{2}$$

and the two second derivative formulas agree.

Higher Orders

The Lagrange interpolating basis functions readily generalize to higher orders. With the products

$$\begin{aligned} Q_n(x) &= \prod_{j\neq n} (x - x_j) \\ R_n(x) &= \prod_{j\neq n} (x_n - x_j) \ , \end{aligned}$$

the nth function is

$$L_n(x) = \frac{Q_n(x)}{R_n(x)} \ .$$

In therms of he Lagrange interpolating basis functions, the corresponding $y_n(x)$ is

$$y_n(x) = \sum_{j=0}^{n} y_j L_j(x) \ .$$

4.4 Three Roads Problem

In the Cartesian plane, consider two parabola-shaped 'roads' described by

$$\begin{aligned} y_1(x) &= -\frac{x^2}{4} - 1 \\ y_2(x) &= \frac{x^2}{4} + 1 \ . \end{aligned}$$

Also, suppose it's our job to propose a new road $f(x)$ connecting the point $(-2, -2)$ on $y_1(x)$ to another point $(1, 5/4)$ on $y_2(x)$.

Straight Line Approximation

The easiest solution to write down, which happens to also be the shortest road connecting the given points, is a straight line

$$f_1(x) = mx + b \ .$$

Using the information provided, it follows that the line connecting the points is specified via

$$\begin{aligned} f(-2) &= y_1(-2) \\ f(1) &= y_2(1) \ , \end{aligned}$$

and solved by:

$$\begin{aligned} m &= \frac{5/4 - (-2)}{1 - (-2)} = \frac{13}{12} \\ b &= \frac{1}{6} \end{aligned}$$

However, the instantaneous derivative of each $y_{1,2}(x)$ tells us

$$\left(\frac{d}{dx}y_1(x)\right)\bigg|_{-2} = 1$$
$$\left(\frac{d}{dx}y_2(x)\right)\bigg|_{1} = \frac{1}{2}\,,$$

and neither is equal to m. This means that the straight line approximation induces a break in the smoothness of the ride at each transition.

Cubic Approximation

Trying a slightly more versatile candidate, consider the order-three approximation

$$f_2(x) = A_0 + A_1 x + A_2 x^2\,,$$

where each A_j is an unknown coefficient, three in total. However, we've already discerned that (at least) four equations govern the system. There are one too few unknowns on hand.

The next best thing to do is guess a cubic equation:

$$f_3(x) = A_0 + A_1 x + A_2 x^2 + A_3 x^3$$

With the cubic approximation, we can find all unknown coefficients by requiring $f(p) = y_{1,2}(x)$ and $f'(x) = y'_{1,2}(x)$ where the roads intersect.

Note that whatever $f(x)$ is doing for $x < -2$ or $x > 1$ doesn't quite matter. This is why adding a cubic term, which undoubtedly changes the global shape of $f(x)$, happens to provide extra tuning in the domain $-2 \le x \le 1$.

Quintic Approximation

Two more equations can be extracted from the information provided, namely the second derivative of each road $y_{1,2}(x)$ at the points of transition:

$$\left(\frac{d^2}{dx^2}y_1(x)\right)\bigg|_{-2} = \frac{-1}{2}$$
$$\left(\frac{d^2}{dx^2}y_2(x)\right)\bigg|_{1} = \frac{1}{2}\,,$$

Including two more equations justifies introducing two more unknowns, so we may as well use an order-five polynomial

$$f_5(x) = \sum_{j=0}^{5} A_j x^j$$

having six unknowns A_j. The first two derivatives of $f_5(x)$ are easy to jot down:

$$\frac{d}{dx}f_5(x) = \sum_{j=1}^{5} j A_j x^{j-1}$$
$$\frac{d^2}{dx^2}f_5(x) = \sum_{j=2}^{5} j(j-1) A_j x^{j-2}$$

Of course, the second derivative of $f_5(x)$ at the respective endpoints is $-1/2$, $1/2$.

N Equations and Unknowns

To keep things general, let the given endpoints be represented by $(x_{1,2}, y_{1,2})$. Let the first and second derivatives of $y_{1,2}(x)$ at the endpoints be denoted $y'_{1,2}(x)$, $y''_{1,2}(x)$, respectively.

Note too that most of the curves $y_{1,2}(x)$ don't play into the solution. The relevant information is the location of each endpoints and the derivative(s). Working all of this out, the foll information of the problem is specified in the augmented matrix

$$\begin{bmatrix}
1 & x_1 & x_1^2 & x_1^3 & x_1^4 & x_1^5 & y_1 \\
1 & x_2 & x_2^2 & x_2^3 & x_2^4 & x_2^5 & y_2 \\
0 & 1 & 2x_1 & 3x_1^2 & 4x_1^3 & 5x_1^4 & y_1' \\
0 & 1 & 2x_2 & 3x_2^2 & 4x_2^3 & 5x_2^4 & y_2' \\
0 & 0 & 2 & 3\cdot 2x_1^2 & 4\cdot 3x_1^3 & 5\cdot 4x_1^3 & y_1'' \\
0 & 0 & 2 & 3\cdot 2x_2^2 & 4\cdot 3x_2^3 & 5\cdot 4x_2^3 & y_2''
\end{bmatrix}.$$

Solution

To finish the example on hand with the data provided, the above becomes

$$\begin{bmatrix}
1 & -2 & 4 & -8 & 16 & -32 & -2 \\
1 & 1 & 1 & 1 & 1 & 1 & 5/4 \\
0 & 1 & -4 & 12 & -32 & 80 & 1 \\
0 & 1 & 2 & 3 & 4 & 5 & 1/2 \\
0 & 0 & 2 & -12 & 48 & -160 & -1/2 \\
0 & 0 & 2 & 6 & 12 & 20 & 1/2
\end{bmatrix},$$

with corresponding RREF:

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 43/81 \\
0 & 1 & 0 & 0 & 0 & 0 & 94/81 \\
0 & 0 & 1 & 0 & 0 & 0 & -149/324 \\
0 & 0 & 0 & 1 & 0 & 0 & -23/162 \\
0 & 0 & 0 & 0 & 1 & 0 & 19/162 \\
0 & 0 & 0 & 0 & 0 & 1 & 7/162
\end{bmatrix},$$

exposing the coefficients $\{A_j\}$. All three roads are plotted together in Figure 1.1.

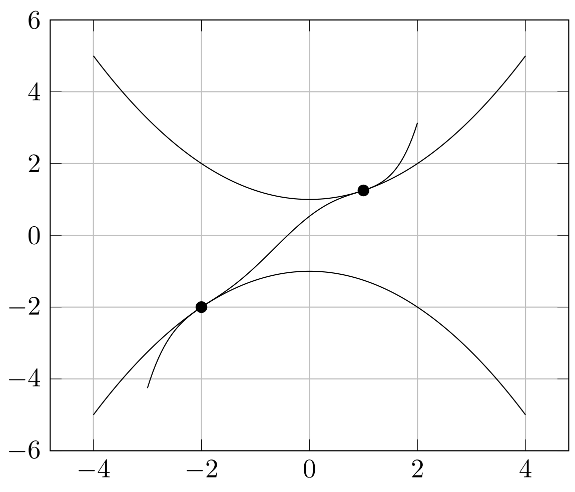

Figure 1.1: Three roads.

4.5 Spline

There is a handy method that combines polynomial interpolation with derivative matching. That is, if we have a set of n points $\{(x_j, y_j)\}$, where $j = 1, 2, 3, \ldots, n$, it's possible to come up with a curve that connects all of the points continuously and smoothly.

Cubic Approximation

In the domain $x_j < x < x_{j+1}$, a cubic interpolation going through the points x_j, x_{j+1} is assumed, taking generic form

$$f_j(x) = \sum_{j=0}^{3} A_j x^j \ .$$

Continuity Conditions

For continuity across the entire function f, we must have

$$\begin{aligned} f_{j-1}(x_j) &= f_j(x_j) \\ f_j(x_{j+1}) &= f_{j+1}(x_{j+1}) \ , \end{aligned}$$

which says the same thing twice. (Substitute $j \to j+1$ in the first equation to recover the second.)

For continuity in derivative of f, differentiate each of the above once with respect to x

$$\begin{aligned} f'_{j-1}(x_j) &= f'_j(x_j) \\ f'_j(x_{j+1}) &= f'_{j+1}(x_{j+1}) \ , \end{aligned}$$

and then once more:

$$\begin{aligned} f''_{j-1}(x_j) &= f''_j(x_j) \\ f''_j(x_{j+1}) &= f''_{j+1}(x_{j+1}) \ , \end{aligned}$$

If there are n total points provided, there must be $n-1$ polynomials in the total interpolation, with a total of $4(n-1)$ unknowns. However, only $4(n-2)$ equations come from continuity arguments, and we need four more.

At the first point $j = 1$ and the last point $j = n$, we set

$$\begin{aligned} f_1(x_1) &= y_1 \\ f_{n-1}(x_n) &= y_n \ . \end{aligned}$$

With the leftover freedom to impose two more equations, choose the second derivative to be zero at each endpoint:

$$\begin{aligned} f''_1(x_1) &= 0 \\ f''_{n-1}(x_n) &= 0 \end{aligned}$$

This setup is called the *natural spline.*

Second Derivative Continuity

This is enough to assemble the whole curve. Begin with the abbreviation

$$k_j = f''_{j-1}(x_j) = f''_j(x_j) \ .$$

Between neighboring k_j, k_{j+1}, the second derivative of $f''_j(x)$ is a straight line connecting the two. Express this line as a two-point Lagrange system:

$$f''_j(x) = k_j \left(\frac{x - x_{j+1}}{x_j - x_{j+1}}\right) + k_{j+1}\left(\frac{x - x_j}{x_{j+1} - x_j}\right)$$

Integrate the above in the x-variable once to get an equation for $f'_j(x)$

$$f'_j(x) = \int f''_j(x)\, dx + C \ ,$$

where C is an integration constant. Substituting the above and carrying out the integral, one finds

$$\begin{aligned} f'_j(x) = &\frac{k_j}{(x_j - x_{j+1})} \frac{(x - x_{j+1})^2}{2} \\ &+ \frac{k_{j+1}}{(x_{j+1} - x_j)} \frac{(x - x_j)^2}{2} + C \ . \end{aligned}$$

Integrate again to get an equation for $f_j(x)$

$$\begin{aligned} f_j(x) = &\frac{k_j}{(x_j - x_{j+1})} \frac{(x - x_{j+1})^3}{3 \cdot 2} \\ &+ \frac{k_{j+1}}{(x_{j+1} - x_j)} \frac{(x - x_j)^3}{3 \cdot 2} + Cx + D \ , \end{aligned}$$

where D is another integration constant.

The substitutions

$$\begin{aligned} C &= A - B \\ D &= -Ax_{j+1} + Bx_j \end{aligned}$$

makes the above be slightly easier to work with:

$$\begin{aligned} f_j(x) = &\frac{k_j}{(x_j - x_{j+1})} \frac{(x - x_{j+1})^3}{3 \cdot 2} \\ &- \frac{k_{j+1}}{(x_j - x_{j+1})} \frac{(x - x_j)^3}{3 \cdot 2} \\ &+ A(x - x_{j+1}) - B(x - x_j) \end{aligned}$$

Integration Constants

The integration constants need to be determined before moving on. Evaluate $f_j(x_j)$ to find

$$y_j = \frac{k_j}{(x_j - x_{j+1})} \frac{(x_j - x_{j+1})^3}{3 \cdot 2} + A(x_j - x_{j+1}) \ ,$$

or

$$A = \frac{y_j}{(x_j - x_{j+1})} - k_j \frac{(x_j - x_{j+1})}{3 \cdot 2} \ .$$

Evaluate $f_j(x_{j+1})$ to find

$$y_{j+1} = \frac{k_{j+1}}{(x_j - x_{j+1})} \frac{(x_j - x_{j+1})^3}{3 \cdot 2} + B(x_j - x_{j+1}) \ ,$$

or

$$B = \frac{y_{j+1}}{(x_j - x_{j+1})} - k_{j+1} \frac{(x_j - x_{j+1})}{3 \cdot 2} \ .$$

Putting the whole solution together:

$$\begin{aligned} &f_j(x) = \\ &\frac{k_j}{6}\left(\frac{(x - x_{j+1})^3}{(x_j - x_{j+1})} - (x - x_{j+1})(x_j - x_{j+1})\right) \\ &- \frac{k_{j+1}}{6}\left(\frac{(x - x_j)^3}{(x_j - x_{j+1})} - (x - x_{j+1})(x_j - x_{j+1})\right) \\ &+ \frac{y_j(x - x_{j+1}) - y_{j+1}(x - x_j)}{x_j - x_{j+1}} \end{aligned}$$

First Derivative Continuity

What remains is to determine the terms k_j in terms of $\{(x_j, y_j)\}$. Write out the derivatives $f_j'(x_j)$ and $f_{j-1}'(x_j)$ and set them equal (as agreed earlier). For an updated $f_j'(x)$, we have

$$\begin{aligned} &f_j'(x) = \\ &\quad \frac{k_j}{6}\left(\frac{3(x - x_{j+1})^2}{(x_j - x_{j+1})} - (x_j - x_{j+1})\right) \\ &\quad - \frac{k_{j+1}}{6}\left(\frac{3(x - x_j)^2}{(x_j - x_{j+1})} - (x_j - x_{j+1})\right) \\ &\quad + \frac{y_j - y_{j+1}}{x_j - x_{j+1}} \end{aligned}$$

and also, shifting index,

$$\begin{aligned} &f_{j-1}'(x) = \\ &\quad \frac{k_{j-1}}{6}\left(\frac{3(x - x_j)^2}{(x_{j-1} - x_j)} - (x_{j-1} - x_j)\right) \\ &\quad - \frac{k_j}{6}\left(\frac{3(x - x_{j-1})^2}{(x_{j-1} - x_j)} - (x_{j-1} - x_j)\right) \\ &\quad + \frac{y_{j-1} - y_j}{x_{j-1} - x_j} \ . \end{aligned}$$

For shorthand, define

$$\begin{aligned} \Delta x_{j+} &= x_j - x_{j+1} \\ \Delta x_{j-} &= x_j - x_{j-1} \end{aligned}$$

and similar for the y-variables. Then evaluate each derivative equation at x_j and equate the results to get the continuity equation

$$\begin{aligned} &k_{j+1}\Delta x_{j+} + 2k_j(\Delta x_{j+} - \Delta x_{j-}) \\ &\qquad - k_{j-1}\Delta x_{j-} = 6\left(\frac{\Delta y_{j-}}{\Delta x_{j-}} - \frac{\Delta y_{j+}}{\Delta x_{j+}}\right) \ . \end{aligned}$$

Creating a System

For yet another shorthand, define the coefficients

$$\begin{aligned} \alpha_j &= -\Delta x_{j-} \\ \beta_j &= 2(\Delta x_{j+} - \Delta x_{j-}) \\ \gamma_j &= \Delta x_{j+} \\ \delta_j &= 6\left(\frac{\Delta y_{j-}}{\Delta x_{j-}} - \frac{\Delta y_{j+}}{\Delta x_{j+}}\right) \end{aligned}$$

which are all known in terms of the provided points. Then, the above can be written

$$\alpha_j k_{j-1} + \beta_j k_j + \gamma_j k_{j+1} = \delta_j \ ,$$

which has three unknowns in general, and we already decided $k_1 = k_n = 0$. Thus only the cases $j = 2, 3, 4, \ldots, n-1$ need be written out.

Example n=5

Choosing a modest example with $n = 5$ points provided, the above becomes:

$$\begin{aligned} \alpha_2 0 + \beta_2 k_2 + \gamma_2 k_3 &= \delta_2 \\ \alpha_3 k_2 + \beta_3 k_3 + \gamma_3 k_4 &= \delta_3 \\ \alpha_4 k_3 + \beta_4 k_4 + \gamma_4 0 &= \delta_4 \end{aligned}$$

This is a system of three equations and three unknowns k_2, k_3, k_4, and the problem has been reduced to a regular $A\vec{x} = \vec{b}$-like system.

In general, the spline calculation leads to a hefty augmented matrix with $n-2$ rows and $n-1$ columns.

5 Newton's Method

In one dimension, Newton's method is a reliable means for estimating the roots of an equation $g(x)$, which is to say finding the x-value(s) that solve $g(x) = 0$.

The formula for Newton's method comes from a first-order approximation of $g(x)$, namely

$$g_1(x) = g(x_0) + g'(x_0)(x - x_0) \ .$$

Providing an initial guess x_0, Setting $g_1(x_1) = 0$ implicates a new x_1 that should be an an improvement over x_0. This can be continued recursively via the formula

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)} \ .$$