2 Cubic Splines

• Splines are interpolations that are generally a sequence of functions that span sequential data intervals, demanding continuity and differentiability at the boundaries between intervals. It is usual to require continuity in the second derivative also, forcing consideration of cubic polynomials $f_i(x)$. Such **cubic splines**, are the most popular choice of splines and interpolative techniques, being common in image enhancement and graphics applications.

$*$ Hence, we seek a fit to a set of points (x_i, y_i) with cubic functions $f_i(x)$ defined on $n-1$ intervals $x_i \le x \le x_{i+1}$, $i = 1, 2, \ldots n-1$. Obviously,

$$\begin{aligned} f_{i-1}(x_i) &= f_i(x_i) = y_i \quad , \quad i = 2, \ldots, n-1 \ , \\ f'_{i-1}(x_i) &= f'_i(x_i) \equiv y'_i \quad , \quad i = 2, \ldots, n-1 \ , \\ f''_{i-1}(x_i) &= f''_i(x_i) \equiv y''_i \quad , \quad i = 2, \ldots, n-1 \ . \end{aligned} \tag{12}$$

The derivatives y'_i and y''_i are clearly unknowns, and byproducts of the spline fit, *i.e., they do not necessarily match the derivative of* $y = f(x)$.

$*$ Considering the second derivative of the cubic splines leads to linear interpolations in f''_i. Setting

$$r_i = \frac{x - x_i}{\delta_i} \quad , \quad \delta_i = x_{i+1} - x_i \ , \tag{13}$$

so that $0 \le r_i \le 1$. It is trivial to determine the interpolation

$$f''_i(x) = y''_i(1 - r_i) + y''_{i+1} r_i \quad , \quad i = 1, \ldots, n-1 \ . \tag{14}$$

Twice integrating this with respect to x and selecting constants to guarantee continuity of the f_i yields cubics with only y''_i as unknowns:

$$f_i(x) = y_i(1 - r_i) + y_{i+1} r_i - \frac{\delta_i^2}{6} r_i(1 - r_i) \left[(2 - r_i) y''_i + (1 + r_i) y''_{i+1} \right] \ , \tag{15}$$

for $i = 1, \ldots, n-1$. Hence, we have satisfied continuity of the spline and its second derivative at the data points, but not that of the first derivatives.

This is easily done by differentiating Eq. (15) and imposing $f'_{i-1}(x_i) = f'_i(x_i)$. The result is a sequence of recurrence relations:

$$\begin{aligned}(x_i - x_{i-1})y''_{i-1} + 2(x_{i+1} - x_{i-1})y''_i + (x_{i+1} - x_i)y''_{i+1} \\ = \; 6\Big[\frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}}\Big] \quad .\end{aligned} \tag{16}$$

• This set of $n-2$ simultaneous equations can be solved for the y''_i by matrix techniques, or by using Mathematica's `Solve[]` function.

$*$ It should be noted that we are short two equations, and these pertain to arbitrary choices of endpoint conditions at x_1 and x_n. One common choice is setting $y''_1 = 0 = y''_n$, generating the so-called *natural spline*. The choice can be tailored to the problem at hand.

Example 2: Consider a cubic spline fit to the function $sin(x)/x$ on $0 < x < 2\pi$. Divide into two equal intervals $(0, \pi)$ and $(\pi, 2\pi)$. Hence, we trivially have $y_1 = 1$, $y_2 = 0$, $y_3 = 0$, and Mathematica can be used to quickly derive

$$y''_1 = -\frac{1}{3} \quad , \quad y''_2 = \frac{2}{\pi^2} \quad , \quad y''_3 = -\frac{1}{2\pi^2} \quad , \tag{17}$$

using the `D[y, x]` and `Limit[y, x->0]` operations on $sin(x)/x$. In the cubic spline scheme, y''_2 is an unknown, and does not have to satisfy Eq. (17). Neither are y''_1 and y''_3 so constrained; we set them according to Eq. (17) given our freedom with the endpoints (i.e. this is not a natural spline)

Eq. (16) then yields

$$\pi y''_1 + 4\pi y''_2 + \pi y''_3 \; = \; \frac{6}{\pi} \; \Rightarrow \; y''_2 \; = \; \frac{1}{12} + \frac{13}{8\pi^2} \quad , \tag{18}$$

a 22% difference from the value in Eq. (17). The spline functions are then

$$\begin{aligned} f_1(x) &= \frac{(\pi - x)}{144\,\pi^3}\left(144\,\pi^2 - 39\,\pi\,x + 14\,\pi^3\,x - 39\,x^2 - 10\,\pi^2\,x^2\right) \\ f_2(x) &= \frac{(\pi - x)\,(2\,\pi - x)}{144\pi^3}\left(117\,\pi + 6\,\pi^3 - 51\,x - 2\,\pi^2\,x\right) \end{aligned} \tag{19}$$

and are plotted in Figure 2, indicating significant inaccuracy of the spline fit; reducing the interval size will improve the fit accordingly.

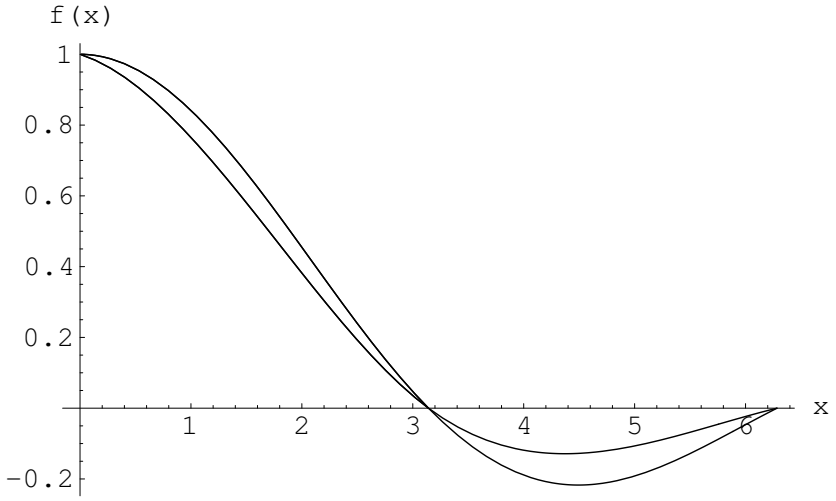

Figure 2: A comparison of $\sin(x)/x$ and the cubic spline fit of Example 2.

$*$ Observe that the first derivatives y_1', y_2', y_3' clearly do not match those of the function at the spline interval boundaries.

$*$ The formalism presented indicates that splines are extremely suitable for functions that are purely numerical in nature, i.e. have been generated as solutions of differential or integral equations, or Monte Carlo or other numerical simulations. *Higher order n splines generally imply better fits.*

3 Least Squares Fitting

This technique borrow heavily on statistics, since it is widely used in fitting experimental/observational data with scatter; it is not treated in this course. The interested reader can look at Section 14-7 of Mathews & Walker. The Mathematica function for this is `Fit[` *data*`,` *functions*`,` *variables* `]`.

4 Padé Approximations

• Functions can often possess significant ranges of values over the domains of interest, so sometimes it is expedient to use a different form of approximation, namely *a ratio of polynomials*. Such a rational function *fit* (technically not an interpolation) to $f(x)$ is called a *Padé Approximation*:

$$f(x) \;\approx\; f_{\rm P}(x) \;\equiv\; \frac{P_n(x)}{Q_m(x)} \quad , \tag{20}$$

where

$$P_n(x) \;=\; \sum_{k=0}^{n} p_k x^k \quad , \quad Q_m(x) \;=\; 1 + \sum_{k=1}^{m} q_k x^k \;\; . \tag{21}$$

The values of the coefficients can be determined by using a power series expansion for $f(x)$:

$$f(x) \;=\; \sum_{k=0}^{\infty} f_k x^k \quad . \tag{22}$$

It follows that

$$\left(\sum_{k=0}^{\infty} f_k x^k \right) \left[1 + \sum_{k=1}^{m} q_k x^k \right] \;\approx\; \sum_{k=0}^{n} p_k x^k \quad . \tag{23}$$

Equating coefficients for various powers of x^j yields the requisite number of simultaneous linear equations that can be solved by either matrix methods, or an algebra package such as `Mathematica`. We illustrate by example.

Example 3: Consider fitting the function $f(x) = 1/\cosh(x)$ on an interval $0 \le x$. Our goal is to obtain an approximation $P_2(x)/Q_4(x)$, where only even powers of x appear due to the even nature of $f(x)$ about $x = 0$.

The series that contribute to Eq. (23) are easily obtained using the Mathematica operation `Series[` *function*`, {`$x, x_0, nmax$`} ]`. Specifically, the sequence of `Mathematica` code

```
ser1 = Simplify[ Normal[ Series[ (1 + p2 x^2)
      - (1 + q2 x^2 + q4 x^4)/Cosh[x], {x,0,6}] ] ] ;
Solve[ {Coefficient[ser1, x^2]==0, Coefficient[ser1, x^4]==0,
      Coefficient[ser1, x^6]==0}, {p2,q2,q4}]
```

can be used to establish the series expansion, extract the relevant coefficients, and then solve the collected simultaneous equations. The result is

$$f(x) \;\approx\; f_{\rm P}(x) \;=\; \frac{P_2(x)}{Q_4(x)} \;\equiv\; \frac{1 - x^2/30}{1 + 7x^2/15 + x^4/40} \quad . \tag{24}$$

Using `fP[x_] := (1 - x^2/30)/(1 + 7 x^2/15 + x^4/40)`, the accuracy of this Padé approximation can be elucidated by plotting it:

```
Plot[ {1/Cosh[x], fP[x], (1/Cosh[x]-fP[x])/fP[x]}, {x, 0, 4.8},
          AxesLabel -> {"x", "f[x], fP[x], Error"} ]
```

While this generally appears accurate for $x \leq 3$, it still fades to only 50% accuracy at $x = 5$; improved accuracy would be facilitated by retaining higher orders in the polynomials, or by factoring out the exponential dependence.

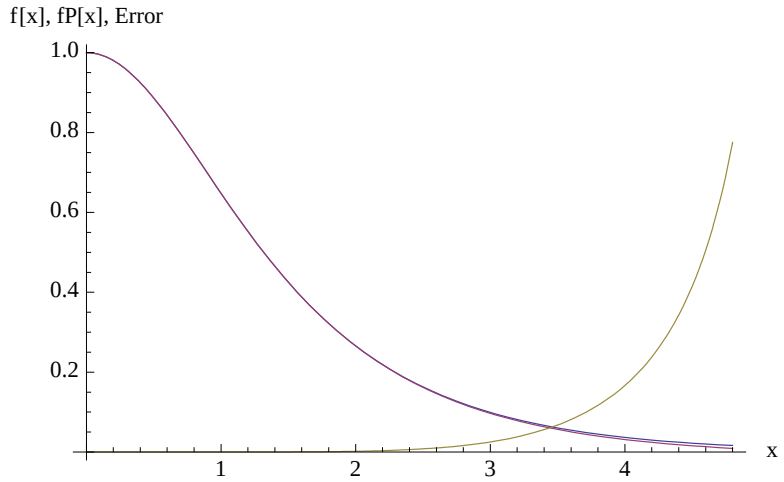

Figure 3: A comparison of $1/\cosh(x)$ [red] and the Padé approximation [blue] of Example 3, which fails for $x \gtrsim 4$ as indicated by the fractional error curve [green-orange].

Extracting the exponential can lead to the Padé approximation

$$f(x) \;\approx\; \frac{e^{-x}\,(1 + 2x/3 + 2x^2/15)}{1 - x/3 + 7x^2/15 - 2x^3/15 + x^4/45} \quad , \tag{25}$$

which is marginally better than Eq. (24) when $x \lesssim 5$, but substantially better (though not excellent) at larger x.

5 Root Solving

• An important tool in the numerical arsenal concerns the numerical solution of algebraic or transcendental equations (*in one variable*) to ascertain their roots, viz.:

$$f(x) \;=\; 0 \quad . \tag{26}$$

There are two broadly-applicable algorithms that will be highlighted here: the simple **bisection technique**, and the **Newton-Raphson method**, a more elegant, but more delicate approach.

∗ Root-solving in more than one dimension is a complicated and awkward business. For analytic functions, in two dimensions it can sometimes be facilitated by using complex variables.

5.1 Numerical Bisection

• This is a "brute force" approach that is extremely robust. The only requirement is that the root be bracketed so as to lie in a well-defined, finite interval $x_1 \leq x \leq x_2$. For this to occur, we require that $f(x)$ switch sign *precisely once* in this interval, i.e. that

$$f(x_1) \times f(x_2) \;<\; 0 \quad , \tag{27}$$

together with the constraint that any sub-interval $[x_l,\, x_u]$ within $[x_1,\, x_2]$ that satisfies $f(x_l) \times f(x_u) > 0$ does not include the root.

∗ This implies that $[x_1,\, x_2]$ contains one root to Eq. (26) only.

∗ Practically, subject to these provisos, the bisection technique can tolerate local extrema and inflection points within $[x_1,\, x_2]$, and accordingly is *extremely robust.*

• The algorithm is to establish sequential sub-intervals $[x_i,\, x_{i+1}]$ by bisecting previous intervals, selecting $x_{i+1} = (x_i + x_{i-1})/2$, and then testing for

$$f(x_{i+1}) \times f(x_i) \begin{array}{c} > \\ < \end{array} 0 \quad \text{and} \quad f(x_{i+1}) \times f(x_{i-1}) \begin{array}{c} > \\ < \end{array} 0 \quad , \tag{28}$$

only one of which can be true. Then whichever test returns a positive product dictates the replacement of either x_i or x_{i-1} with an "update" x_{i+1} to maintain bracketing of the root.

Plot: Graphical representation of the rapid convergence

• Solution to arbitrary precision is guaranteed, and convergence to this is exponential, at a rate of around 2^{-n} in fractional error for n iterations.

∗ This implies accuracies of typically 0.1% with ten iterations.

• **Example 4:** Consider finding the first two positive roots of the Bessel function $f(x) = J_0(2\pi \sin x)$. The character of this function is revealed by the Mathematica command

```
Plot[ BesselJ[0, 2*Pi Sin[x]], {x, 0, Pi/2},
          AxesLabel -> {"x", "f[x]"}]
```

which yields the plot

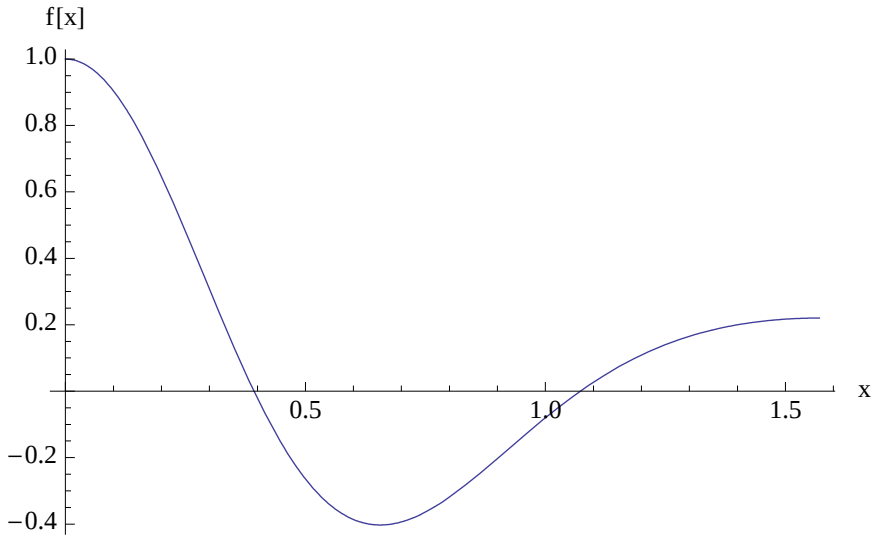

Figure 4: The ordinary Bessel function $f(x) = J_0(2\pi \sin x)$ over the interval $0 \le x \le \pi/2$, for the exposition on the bisection technique in Example 4.

From this it is clear that there are two roots in the $[0,\, \pi/2]$ interval, one in $[0,\, \pi/4]$, and one in $[\pi/4,\, \pi/2]$. This establishes the requisite bracketing.

By sequentially setting `xmin = 0`, `xmax = Pi/4`, and `xmin = Pi/4`, `xmax= Pi/2`, the iteration can be performed on each interval using Mathematica:

```
While[ xmax-xmin > epsilon, xmid = (xmax+xmin)/2;
          If[ f[xmid] * f[xmin] >=0, xmin=xmid, xmax=xmid] ]
```

Here epsilon is the "tolerance," which is generally set to a small number; `epsilon = 0.0001` is chosen here. The resultant values of the roots depend slightly on the initial bracketing and the tolerance. One arrives at

$$x \;=\; 0.3927 \quad \text{and} \quad x \;=\; 1.0727 \quad , \tag{29}$$

the lower one being very close to $\pi/8$. These approximate roots can be checked with

```
FindRoot[f[x]==0, x, 0.4], FindRoot[f[x]==0, x, 1.0]
```

to yield $x \;=\; 0.39276$ and $x \;=\; 1.07281$, with slightly greater numerical precision.

• Note that there are more refined versions of the bisection algorithm, such as invoking Lagrange interpolation with linear, quadratic or cubic functions (the method of **regula falsi**), but in practice these only accelerate convergence marginally, and are considerably more involved concerning their coding.

5.2 The Newton-Raphson Technique

• This approach uses Newton's theory of differentiation to advantage in locating a root in an iterative process. It works best when a test value is proximate to the actual root, and the specified function $f(x)$ is well-behaved in the sense that there are *no extrema, inflection points or singularities* in the interval neighboring the root. This criterion amounts to bounds on $f'(x)$ and higher order derivates of $f(x)$. The Newton-Raphson method is more mathematically elegant than bisection, but its rate of convergence is only a modest improvement, and its coding is more involved.

The technique generates a sequence of estimates x_n for the root x_r (i.e. $f(x_r) = 0$) using a simple recurrence relation, upon which the definition of the derivative is based. Since $f'(x) \approx [f(x+h) - f(x)]/h$ for sufficiently small h, we can set $x \to x_n$ and $x + h = x_{n+1}$, i.e. $h = x_{n+1} - x_n$. Assuming $|f(x_{n+1})| \ll |f(x_n)|$, this approximation inverts to setting

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad . \tag{30}$$

This recurrence relation amounts to finding the x-axis intercept $x = x_{n+1}$ of the tangent to $f(x)$ at $x = x_n$:

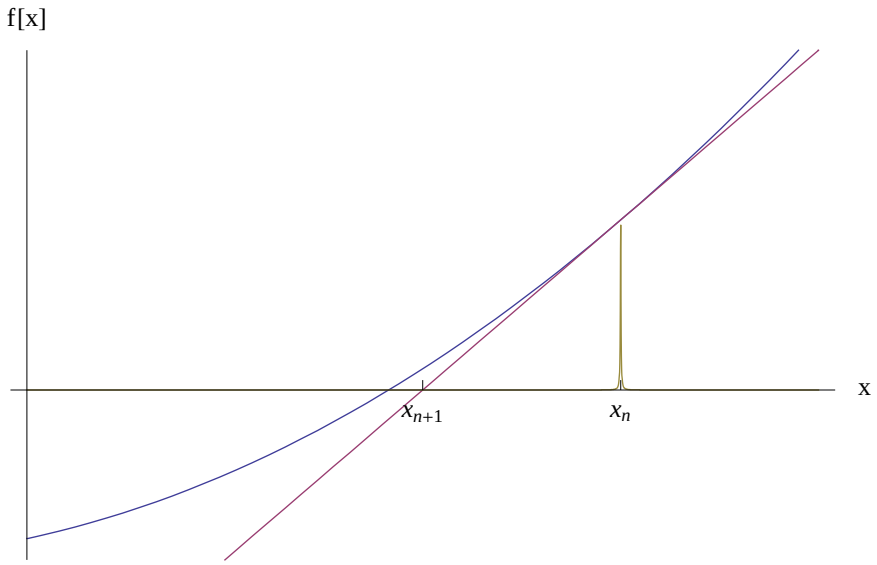

Figure 5: Illustration of Newton's method for obtaining an improved estimate x_{n+1} for the root of $f(x)$.

$*$ It is therefore clear that proximity of extrema ($f'(x_n) \approx 0$) or closely spaced roots will degrade the facility of this method.

$*$ The Newton-Raphson technique works well for known mathematical functions, but can also be viable for numerically-evaluated functions (e.g. multiple integrals) whose first derivative can be computed with precision.

$*$ The technique can also be adapted to analytic functions in the complex plane, where the bisection method is unworkable.