Lecture 15

Interpolation II

1 Introduction

In the previous lecture we focused primarily on polynomial interpolation of a set of *n* points. A difficulty we observed is that when *n* is large, our polynomial has to be of high order, namely $n-1$. Unfortunately, high-order polynomials tend to suffer from "wiggle," and this limits their practical usefulness for interpolation. In this lecture we will explore how we can use polynomials of moderate order to achieve smooth interpolations while avoiding the problems associate with high-order polynomials.

2 Piecewise polynomial interpolation – Hermite splines

What we've previously called linear interpolation is more precisely *piecewise-linear interpolation*. We don't interpolate the entire set of points with a single line. Instead, we use different line segments over different intervals between sample points (Fig. 1). The complete interpolation is built by "tying together" these lines. In fact the sample points where lines join or "tie" together, $(x_i, y_i), i=2,3,\ldots,x_{n-1}$, are appropriately called "knots."

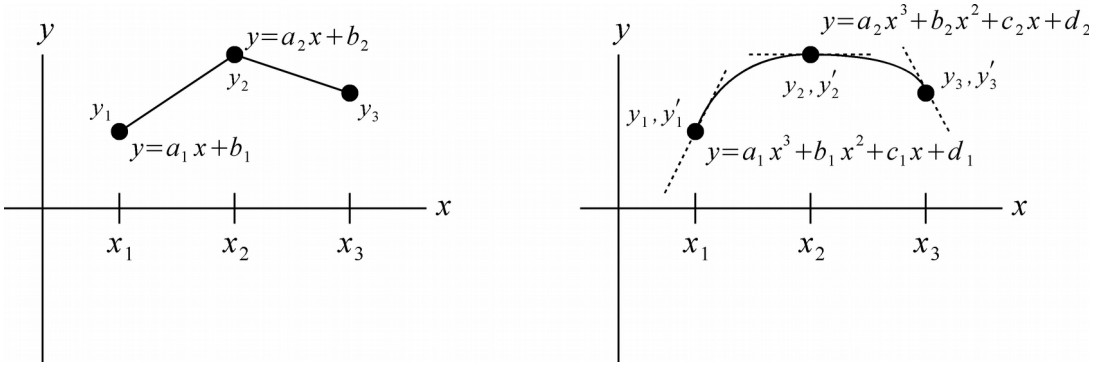

Fig. 1: Piecewise interpolation: linear (left) and cubic (right). The sample points at which pieces join (or "tie together") are called "knots."

The entire interpolation function is described by

$$y=f(x)=a_i x+b_i \text{ if } x_i \le x < x_{i+1}$$

and the a_i, b_i values are determined by the conditions

$$y_i = a_i x_i + b_i$$
$$y_{i+1} = a_i x_{i+1} + b_i$$

We can express a linear function in different forms, one of which might be more convenient for determining coefficients. We could write the i^{th} segment in the form of a 1st order Taylor series expanded about the point x_i,

$$y = y_i + y_i'(x - x_i)$$

This trivially satisfies $y = y_i$ when $x = x_i$. For $x = x_{i+1}$ the requirement

$$y_{i+1} = y_i + y_i'(x_{i+1} - x_i)$$

provides the value the y_i' value

$$y_i' = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Alternately, the Lagrange interpolating polynomial can be written by inspection as

$$y = y_i \frac{x - x_{i+1}}{x_i - x_{i+1}} + y_{i+1} \frac{x - x_i}{x_{i+1} - x_i}$$

We now extend this idea of piecewise interpolation to polynomials. While a piecewise linear interpolation is continuous, the derivative is clearly *not* continuous at the sample points. Suppose now that for each x_i we know both the function value y_i and the function slope y_i'. Let's build a piecewise polynomial interpolation that has the specified function and slope values at the knots. These polynomial pieces are known as *splines*. This term comes from the practice of bending strips of wood or plastic to form smooth curves, a technique often used in ship building and pre-computer-era drafting.

For each segment we have four equations to satisfy, the two endpoint function values and the two endpoint slope values. Our interpolation function must therefore have four unknown coefficients. Since a 3rd order polynomial has four coefficients we write (Fig. 1)

$$f(x) = a_i x^3 + b_i x^2 + c_i x + d_i \text{ if } x_i \le x < x_{i+1}$$

In each interval we have four unknowns a_i, b_i, c_i, d_i satisfying four equations

$$\begin{aligned} y_i &= a_i x_i^3 + b_i x_i^2 + c_i x_i + d_i \\ y_i' &= 3 a_i x_i^2 + 2 b_i x_i + c_i \\ y_{i+1} &= a_i x_{i+1}^3 + b_i x_{i+1}^2 + c_i x_{i+1} + d_{i+1} \\ y_{i+1}' &= 3 a_i x_{i+1}^2 + 2 b_i x_{i+1} + c_i \end{aligned} \tag{1}$$

We could solve these four equations in four unknowns directly. Instead we'll apply some bookkeeping to obtain a bit "cleaner" approach.

To simplify analysis over an arbitrary interval $x_i \le x < x_{i+1}$ it's a good idea to form the *normalized variable*

$$u = \frac{x - x_i}{x_{i+1} - x_i} = \frac{x - x_i}{h_i} \tag{2}$$

As *x* varies over $x_i \le x \le x_{i+1}$, *u* varies over $0 \le u \le 1$. Note that

$$\frac{du}{dx} = \frac{1}{h_i}$$

so

$$y' = \frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} = \frac{1}{h_i}\frac{dy}{du}$$

or

$$\frac{dy}{du} = h_i y'$$

We now write

$$\begin{aligned} y &= a + b u + c u^2 + d u^3 \\ \frac{dy}{du} &= b + 2 c u + 3 d u^2 \end{aligned}$$

Our system of equations is now (evaluating the above equations at $u = 0,1$)

$$\begin{aligned} y_i &= a \\ h_i y'_i &= b \\ y_{i+1} &= a + b + c + d \\ h_i y'_{i+1} &= b + 2 c + 3 d \end{aligned} \tag{3}$$

The simplification from (1) is significant. These can easily be solved to give

$$\begin{aligned} a &= y_i \\ b &= h_i y'_i \\ c &= -3 y_i - 2 h_i y'_i + 3 y_{i+1} - h_i y'_{i+1} \\ d &= 2 y_i + h_i y'_i - 2 y_{i+1} + h_i y'_{i+1} \end{aligned}$$

and the interpolating cubic is

$$y = y_i + h_i y'_i u + (-3 y_i - 2 h_i y'_i + 3 y_{i+1} - h_i y'_{i+1}) u^2 + (2 y_i + h_i y'_i - 2 y_{i+1} + h_i y'_{i+1}) u^3$$

It can be convenient to separate out the various y terms to obtain

$$y = y_i (1 - 3 u^2 + 2 u^3) + h_i y'_i (u - 2 u^2 + u^3) + y_{i+1} (3 u^2 - 2 u^3) + h_i y'_{i+1} (-u^2 + u^3)$$

A bit of factoring puts this in a more compact form

$$y = (1-u)^2 [y_i (1 + 2u) + h_i y'_i u] + u^2 [y_{i+1} (3 - 2u) - h_i y'_{i+1} (1 - u)]$$

The result is the so-called *Hermite spline* interpolation algorithm.

Hermite spline interpolation

Given $(x_1, y_1, y'_1), (x_2, y_2, y'_2), \ldots, (x_n, y_n, y'_n)$, with increasing x values

For a value x

Find x_i such that $x_i < x < x_{i+1}$

Set $h_i = x_{i+1} - x_i$ and $u = (x - x_i)/h_i$

Calculate $y = (1-u)^2 [y_i (1 + 2u) + h_i y'_i u] + u^2 [y_{i+1} (3 - 2u) - h_i y'_{i+1} (1 - u)]$

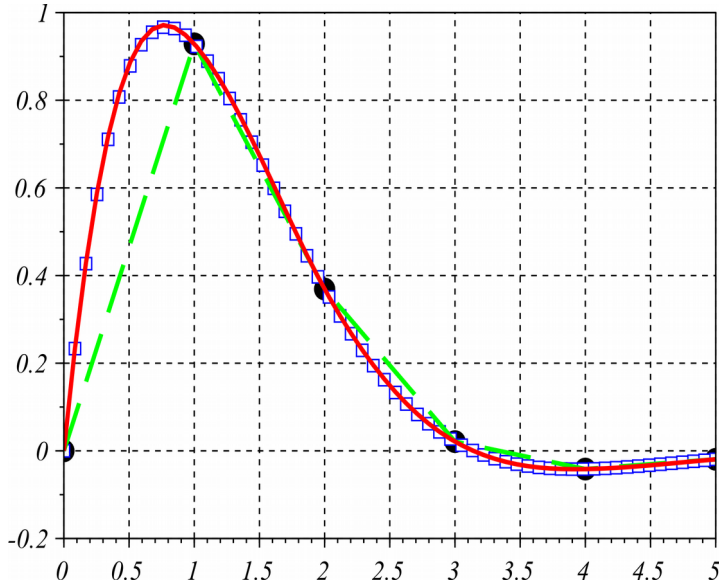

Fig. 2: $y=3e^{-x}\sin x$ *. solid circles: sample points, squares: function values, dashed line: linear interpolation, solid line: Hermite-spline interpolation. Derivative values were estimated numerically.*

Considering the entire interpolation algorithm as a function, we write $y=S(x)$. A Scilab function to perform Hermite spline interpolation is given in Appendix 1 as `interpHermite`, and an example of Hermite spline interpolation is shown in Fig. 2.

If we had samples of the form (x_i, y_i, y_i', y_i'') we could find 5th order interpolation polynomials for each interval and so on, in principle, for any number of known derivatives at each sample point. If we have function and derivative values up to $d^m y/dx^m$, the two endpoints of each interval will provide $2(m+1)$ equations. A polynomial with this many coefficients has order $n=2m+1$.

3 Cubic splines

If we know function and derivative values at *n* points, we can interpolate each interval with Hermite splines. Often, however, we only know the function values and not the derivative values. This provides only enough information to uniquely determine a piecewise-linear interpolation. But the smoothness of a piecewise-cubic interpolation is highly desirable, and we would like to find a way to keep that property even when we lack derivative information. We will refer to piecewise cubic interpolation without specific derivative values as *cubic splines.*

3.1 Smoothest Hermite spline interpolation

One approach would be to treat the y_i' as unknowns and find the values that optimize some desirable property of the curve. "Smoothness" is an intuitively appealing property to have. A smooth curve is one in which the slope does not change rapidly. A sudden change in slope produces a "kink" in the curve, which is about as "unsmooth" as you can get.

Therefore the second derivative – the rate of change of the slope – should be small. Let's write the integral of the square of the second derivative of $S(x)$ as a function of the unknown y_i'

values:

$$\phi(y_1', y_2', \ldots, y_n') = \int_{x_1}^{x_n} \left[S''(x) \right]^2 dx \tag{4}$$

Using the notation

$$S_i(u) = (1-u)^2 [y_i(1+2u) + h_i y_i' u] + u^2 [y_{i+1}(3-2u) - h_i y_{i+1}'(1-u)]$$

for u as given in (2), we write (4) as

$$\phi = \sum_{i=1}^{n-1} \frac{1}{h_i} \int_0^1 \left(\frac{d^2}{dt^2} S_i(u) \right)^2 du$$

For evenly space samples where $h_i = h$ we show in Appendix 1 that minimizing ϕ leads to the equations

$$y_2' + 2y_1' = \frac{3}{h}(y_2 - y_1) \ , \ y_n' + 2y_{n-1}' = \frac{3}{h}(y_n - y_{n-1}) \tag{5}$$

$$y_{i+1}' + 4y_i' + y_{i-1}' = \frac{3}{h}(y_{i+1} - y_{i-1}) \ \text{ for } \ i = 2,3,\ldots,n-1 \tag{6}$$

for the y_i' values. The case of nonuniform samples is similar but a bit messier because we have to keep track of different h values.

3.2 Continuity of second derivatives

Another approach is to require that not only the first, but also the second derivatives of the interpolation be continuous at the knots $x_2, x_3, \ldots, x_{n-1}$. There are $n-2$ knots, since the endpoints x_1, x_n are not knots (no other pieces connect there). Continuity of the second derivatives at the knots provide $n-2$ equations. For uniformly spaced samples these turn out to be (6). This tells us that the "smoothest" Hermite spline interpolation we derived previously also results in continuous second derivatives at all knots.

We then need two more equations to obtain a unique solution. So-called "natural" end conditions are obtained by setting $S''(x) = 0$ at the endpoints x_1, x_n, in other words, we let the interpolation "go straight" at both ends. This leads to equations (5). Therefore a cubic spline interpolation with natural end conditions is precisely the "optimally smooth" Hermite spline interpolation we derived above.

Another option is to specify the end-point slopes y_1', y_n'. This is called the "fixed-slope" end conditions. If we have a good estimate of these slopes then this makes sense. Otherwise the choice is arbitrary.

Finally we can choose the so-called "not-a-knot" conditions where we require the third derivative of the interpolation to be continuous at the first and last knots. At these knots, therefore, the cubic functions and the first, second and third derivatives are continuous. But cubics that agree in this manner are simply the same cubic; there is no other possibility. So what used to be the first and last knots are no longer knots, hence the name not-a-knot. For the uniformly sampled case these equations read

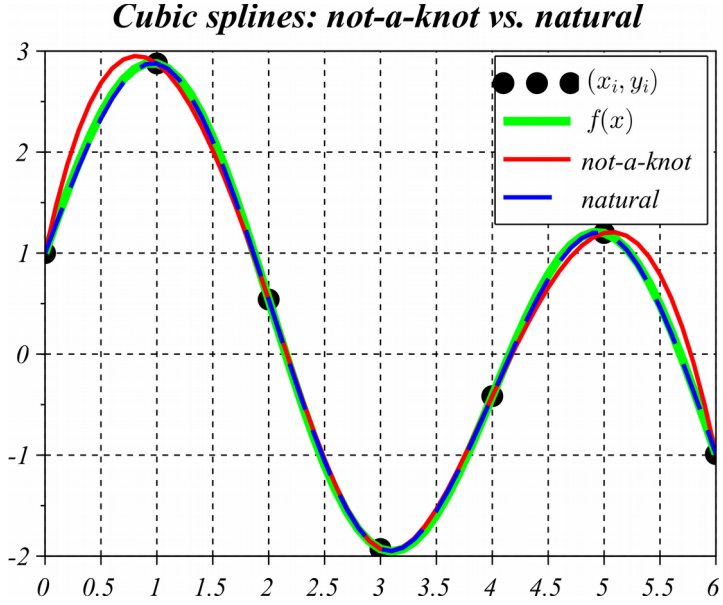

Fig. 3 A case where natural conditions produce a more accurate interpolation than not-a-knot conditions.

$$y'_1+4\,y'_2+y'_3=\frac{3}{h}(y_3-y_1) \quad \text{and} \quad y'_{n-2}+4\,y'_{n-1}+y'_n=\frac{3}{h}(y_n-y_{n-2}) \tag{7}$$

Which end conditions should we choose? The natural conditions are attractive because of their "maximally smooth" feature. However, in many cases the not-a-knot conditions provide a more accurate interpolation. It depends on the underlying function $f(x)$ (see Fig. 3 and Fig. 4). Actual functions are not necessarily as smooth as possible! "Common practice" is to use the not-a-knot conditions. In practice the two end conditions produce very similar results except, possibly in the first and last intervals.

3.3 Optimal smoothness of natural cubic spline interpolation

We've spent a lot of time working with cubic splines. We've shown that natural cubic splines are the smoothest-possible piecewise-cubic interpolation of any set of points. If smoothness is so desirable, why not try piecewise interpolation with even higher-order polynomials? It turns out that *no other interpolation is smoother than a natural cubic spline.* This rather remarkable result tells us that, as far as smoothness is concerned, cubic splines are the best we can do.

Suppose that $S(x)$ is the natural cubic spline interpolation of the *n* samples (x_i, y_i). Let $f(x)$ be *any* twice-differentiable function that also interpolates the samples (this could even be the original function from which the samples were drawn). Then one can show that

$$\int_{x_1}^{x_n}\left[S''(x)\right]^2 dx \le \int_{x_1}^{x_n}\left[f''(x)\right]^2 dx$$

This is the sense in which we can say that no function provides a "smoother" interpolation of a set of data points than does the natural cubic spline. However, as shown in Fig. 3 and Fig. 4, smoother is not necessarily better.

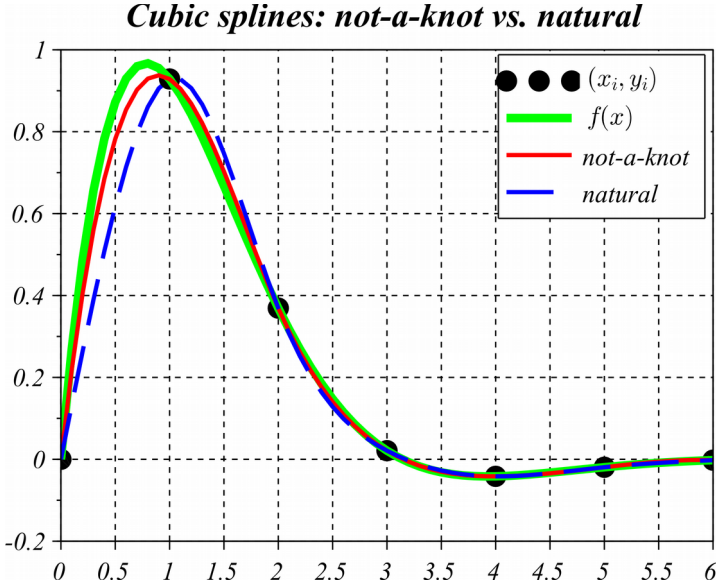

Fig. 4 A case where not-a-knot conditions produce a more accurate interpolation than natural conditions.

4 The interp1 function (Scilab/Matlab)

As for most numerical methods we study, Scilab and Matlab have built-in functions offering state-of-the-art implementations. The following command

```
yp = interp1(x,y,xp,str); //str = 'nearest' or 'linear' or 'spline'
```

Implements one-dimensional interpolation of either nearest-neighbor, linear or spline type. For spline interpolation not-a-knot end conditions are used. Here the vectors `x,y` are sample data, `xp` is the vector of x values where we want interpolations, and `yp` is the vector of interpolated

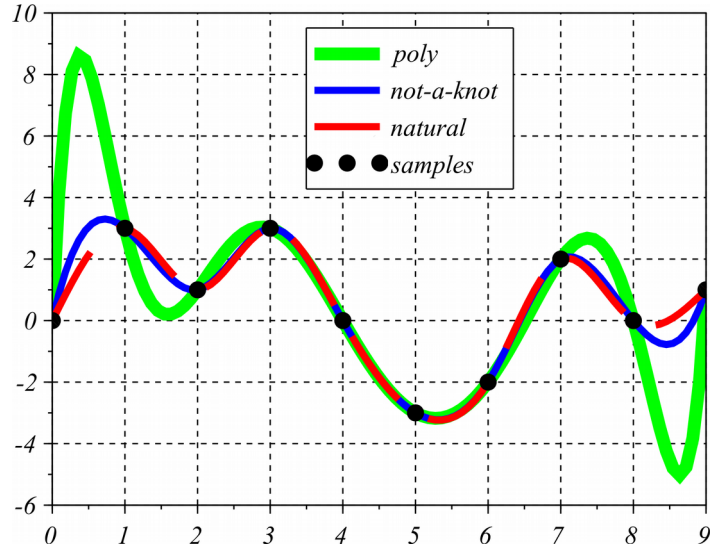

Fig. 5: Ten randomly selected points (dots). Thick (green) line: ninth-order polynomial. Thin (blue) line: not-a-knot spline. Thin dashed (red) line: natural spline.

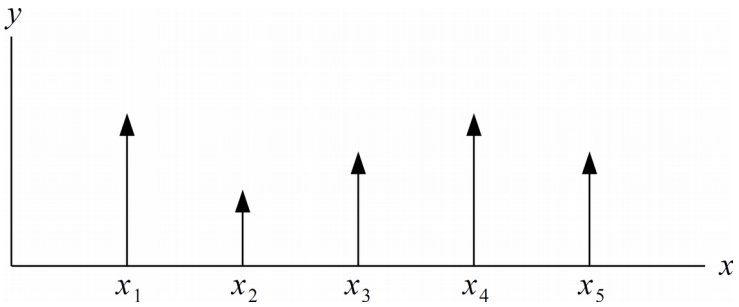

Fig. 6 Sample data represented as impulses or "poles" of height y_i .

values. This is sufficient for almost all one-dimensional interpolation needs.

With regards to "wiggle," the advantage of splines over high-order polynomials for interpolation is illustrated in Fig. 5. Notice too that the difference of the two spline end conditions is significant only in the first and last intervals. These data points were chosen at random so there is no actual underlying $f(x)$. However, the spline interpolations certainly appear more "realistic."

5 Lanczos (convolution) interpolation

The methods we have considered so far apply to an arbitrary set of samples (x_i, y_i) . If the x values are uniformly spaced, however, then some ideas from signal processing can be applied. We turn to that now. Let's suppose we have uniformly spaced samples with $x_i = x_1 + (i-1)h$. We can visualize our data as shown in Fig. 6.

Imagine an impulse or "pole" of height y_i erected vertically with its base "on the ground" at location x_i . For our purposes *convolution* is the process of replacing each impulse with a common *impulse response* function, centered at the impulse and scaled by the "height" y_i . The *triangle* function $\Lambda(x)$ is shown in Fig. 7.

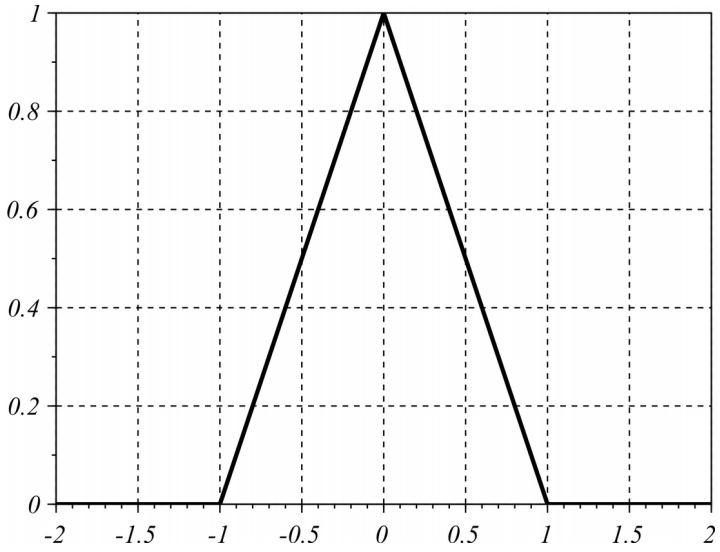

Fig. 7: The "triangle" function $\Lambda(x)$.

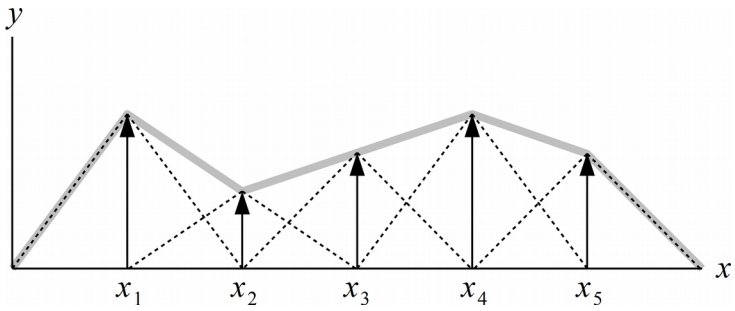

Fig. 8: Convolution of impulses with triangle function. Thick gray curve is sum of all triangle functions and interpolates the data points.

If we replace each impulse by a stretched version of the triangle function

$$\Lambda(x/h)=\begin{cases} 1-|x/h| & |x|\le h \\ 0 & |x|>h \end{cases}$$

scaled by y_i, then the sum of these

$$f(x)=\sum_{i=1}^{n} y_i \Lambda\left(\frac{x-x_i}{h}\right)$$

produces the interpolation shown in Fig. 8. We recognize this as the piecewise linear interpolation of the data with the addition of linear *extrapolations* at the two ends. This naturally leads us to wonder if using a different impulse response function might produce a "better" interpolation.

Thinking of x as time and y as the amplitude of an audio signal, there is a remarkable theorem

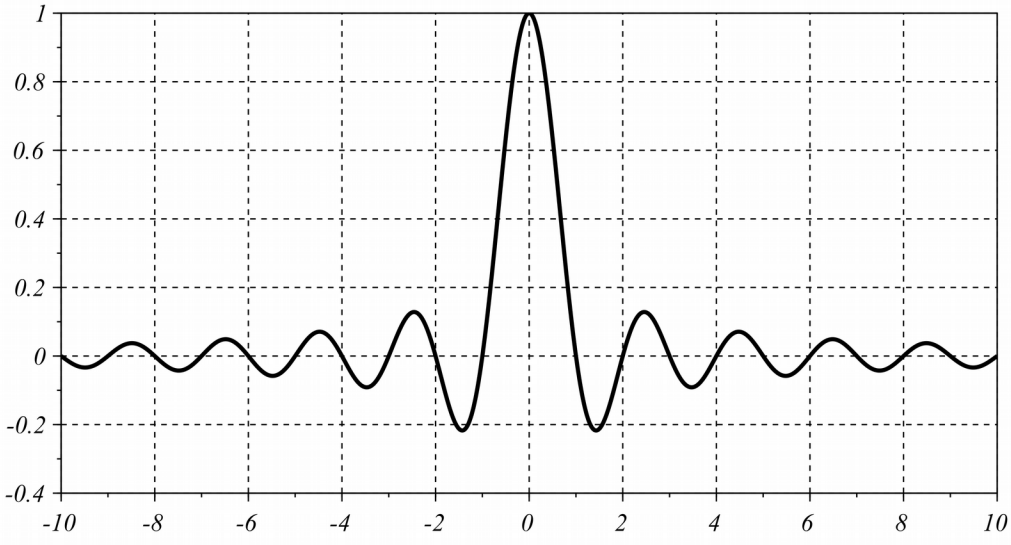

Fig. 9 The sinc function.

due to Nyquist which says that provided: 1) the signal from which the audio samples were draw contains "frequency components" only within a limited range, and 2) the sample separation h is properly chosen, then a convolution interpolation using the *sinc* function (pronounced "sink") will *exactly* recreate the original function $f(x)$. Mathematically

$$y = f(x) = \sum_{i=-\infty}^{\infty} y_i \operatorname{sinc}\left(\frac{x - x_i}{h}\right)$$

The sinc function is

$$\operatorname{sinc}(x) = \frac{\sin \pi x}{\pi x}$$

and is plotted in Fig. 9. Note that $\operatorname{sinc} 0 = 1$ and $\operatorname{sinc} n = 0$ for n a non-zero integer. Unfortunately the sinc function extends to $x \to \pm\infty$, although the amplitude of the bumps drop off as $1/|x|$. If we are interpolating many points, we'll have to add a contribution from each point. A compromise proposed by Lanczos is to "window" the sinc function by another sinc function to produce the *Lanczos kernel*

$$L(x) = \begin{cases} \operatorname{sinc}(x) \operatorname{sinc}\left(\frac{x}{a}\right) & |x| \le a \\ 0 & |x| > a \end{cases}$$

where a is typically chosen to be a small integer (most often 2 or 3). This is plotted in Fig. 10. The Lanczos interpolation is

$$y = \sum_{j=i+1-a}^{i+a} y_j L\left(\frac{x - x_j}{h}\right)$$

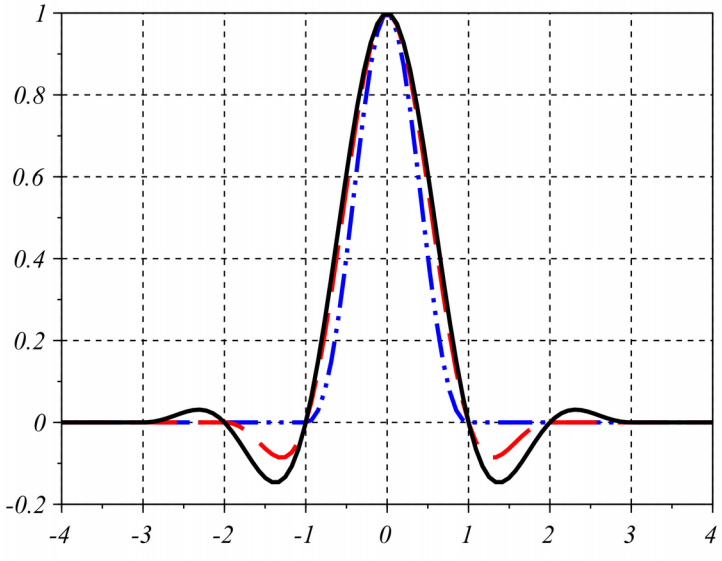

Fig. 10 The Lanczos kernel for $a = 1,2,3$.

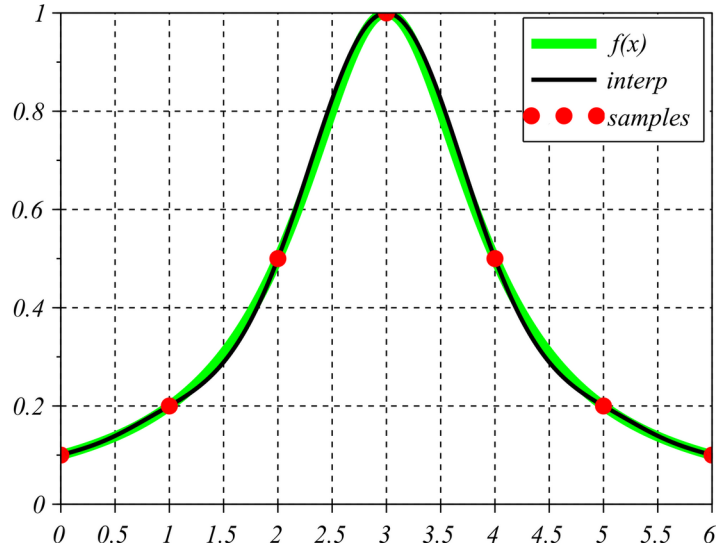

Fig. 11: Seven data points and Lanczos interpolation.

where *i* is the index such that $x_i \le x < x_{i+1}$. In this expression only the $2a$ nearest sample points contribute to the interpolation at a given value of *x*. An example of Lanczos interpolation (with $a=3$) is shown in .

6 Appendix 1 – Scilab code

6.1 Hermite spline interpolation

```
0001  ////////////////////////////////////////////////////////////////////
0002  // interpHermite.sci
0003  // 2014-06-25, Scott Hudson, for pedagogic purposes
0004  // Given n samples x(i),y(i),y1(i) in the column vectors x,y,y1
0005  // where y(i)=f(x(i)) and y1(i) is the derivative of f(x) at x(i),
0006  // interpolate at points xp using Hermite splines.
0007  // Note: x and xp values must be in ascending order.
0008  ////////////////////////////////////////////////////////////////////
0009  function yp=interpHermite(x, y, y1, xp)
0010    n = length(x);
0011    m = length(xp);
0012    yp = zeros(xp);
0013    i = 1; //start linear search at first element
0014    for j=1:m
0015      while (xp(j)>x(i+1)) //find j so that x(j)<=u(i)<=x(j+1)
0016        i = i+1;
0017      end
0018      h = x(i+1)-x(i);
0019      t = (xp(j)-x(i))/h;
0020      yp(j) = (t-1)^2*(y(i)*(2*t+1)+y1(i)*h*t) ..
0021        +t^2*(y(i+1)*(3-2*t)+y1(i+1)*h*(t-1));
0022    end
0023  endfunction
```

7 Appendix 2 – Smoothest Hermite spline interpolation

Assume we have *n* samples (x_i, y_i), and the *x* values are equally spaced $x_i = x_1 + (i-1)h$. Let

$$\phi = \sum_{i=1}^{n-1} \frac{1}{h} \int_0^1 \left(\frac{d^2}{dt^2} S_i(t) \right)^2 dt$$

where

$$S_i(t) = (t-1)^2 \left[y_i (2t+1) + y_i' h t \right] + t^2 \left[y_{i+1} (3-2t) + y_{i+1}' h (t-1) \right]$$

One can show that (a computer algebra program helps!)

$$\int_0^1 \left(\frac{d^2}{dt^2} S_i(t) \right)^2 dt = 12 (y_{i+1} - y_i)^2 - 12h \left[(y_{i+1}' + y_i')(y_{i+1} - y_i) \right] + 4h^2 \left((y_i')^2 + y_i' y_{i+1}' + (y_{i+1}')^2 \right)$$

For a knot $1 < i < n$, y_j' appears in S_{i-1} and S_i. Calling

$$4w = \frac{1}{h} \int_0^1 \left(\frac{d^2}{dt^2} S_{i-1}(t) \right)^2 dt + \frac{1}{h} \int_0^1 \left(\frac{d^2}{dt^2} S_i(t) \right)^2 dt$$

we have

$$\begin{aligned} w = \quad & 3 (y_i - y_{i-1})^2 - 3h \left[(y_i' + y_{j-1}')(y_i - y_{i-1}) \right] + h^2 \left((y_{i-1}')^2 + y_{i-1}' y_i' + (y_i')^2 \right) \\ & + 3 (y_{i+1} - y_i)^2 - 3h \left[(y_{i+1}' + y_i')(y_{i+1} - y_i) \right] + h^2 \left((y_i')^2 + y_i' y_{i+1}' + (y_{i+1}')^2 \right) \end{aligned}$$

Setting

$$\frac{\partial w}{\partial y_i'} = h^2 (y_{i+1}' + 4 y_i' + y_{i-1}') + 3h (y_{i-1} - y_{i+1}) = 0$$

we have

$$y_{i+1}' + 4 y_i' + y_{i-1}' = \frac{3}{h} (y_{i+1} - y_{i-1}) \tag{8}$$

For the first interval

$$\frac{\partial}{\partial y_1'} \left\{ 3 (y_2 - y_1)^2 - 3 h_1 \left[(y_2' + y_1')(y_2 - y_1) \right] + h_1^2 \left((y_1')^2 + y_1' y_2' + (y_2')^2 \right) \right\} = 0$$

gives us

$$y_2' + 2 y_1' = \frac{3}{h} (y_2 - y_1) \tag{9}$$

while for the last interval we find

$$y_n' + 2 y_{n-1}' = \frac{3}{h} (y_n - y_{n-1}) \tag{10}$$