Sinc interpolation

Herman Jaramillo

February 15, 2012

1 Introduction

These notes are an effort to understand the function `mksinc.c` in the Center for Wave Phenomena (CWP) library from the Seismic Un*x (SU) package. I list the function upfront because everything done here is done in order to explain this function:

```
 lena,hjaramil $ pwd
/home/hjaramil/SU/src/cwp/lib
 lena,hjaramil $ cat mksinc.c
/* Copyright (c) Colorado School of Mines, 2011.*/
/* All rights reserved.                       */

/********************** self documentation **********************/
/*****************************************************************************
MKSINC - Compute least-squares optimal sinc interpolation coefficients.

mksinc          Compute least-squares optimal sinc interpolation coefficients.

******************************************************************************
Function Prototype:
void mksinc (float d, int lsinc, float sinc[]);

******************************************************************************
Input:
d               fractional distance to interpolation point; 0.0<=d<=1.0
lsinc           length of sinc approximation; lsinc%2==0 and lsinc<=20

Output:
sinc            array[lsinc] containing interpolation coefficients

******************************************************************************
Notes:
The coefficients are a least-squares-best approximation to the ideal
sinc function for frequencies from zero up to a computed maximum
frequency.  For a given interpolator length, lsinc, mksinc computes
the maximum frequency, fmax (expressed as a fraction of the nyquist
```

```
frequency), using the following empirically derived relation (from
a Western Geophysical Technical Memorandum by Ken Larner):

        fmax = min(0.066+0.265*log(lsinc),1.0)

Note that fmax increases as lsinc increases, up to a maximum of 1.0.
Use the coefficients to interpolate a uniformly-sampled function y(i)
as follows:

            lsinc-1
    y(i+d) =  sum  sinc[j]*y(i+j+1-lsinc/2)
              j=0

Interpolation error is greatest for d=0.5, but for frequencies less
than fmax, the error should be less than 1.0 percent.

*****************************************************************************
Author:  Dave Hale, Colorado School of Mines, 06/02/89
*****************************************************************************/
/*************** end self doc *******************************/

#include "cwp.h"

void mksinc (float d, int lsinc, float sinc[])
/*****************************************************************************
Compute least-squares optimal sinc interpolation coefficients.
******************************************************************************
Input:
d               fractional distance to interpolation point; 0.0<=d<=1.0
lsinc           length of sinc approximation; lsinc%2==0 and lsinc<=20

Output:
sinc            array[lsinc] containing interpolation coefficients
******************************************************************************
Notes:
The coefficients are a least-squares-best approximation to the ideal
sinc function for frequencies from zero up to a computed maximum
frequency.  For a given interpolator length, lsinc, mksinc computes
the maximum frequency, fmax (expressed as a fraction of the nyquist
frequency), using the following empirically derived relation (from
a Western Geophysical Technical Memorandum by Ken Larner):

        fmax = min(0.066+0.265*log(lsinc),1.0)

Note that fmax increases as lsinc increases, up to a maximum of 1.0.
Use the coefficients to interpolate a uniformly-sampled function y(i)
as follows:

            lsinc-1
    y(i+d) =  sum  sinc[j]*y(i+j+1-lsinc/2)
              j=0

Interpolation error is greatest for d=0.5, but for frequencies less
than fmax, the error should be less than 1.0 percent.
******************************************************************************
Author:  Dave Hale, Colorado School of Mines, 06/02/89
******************************************************************************/
```

```
int j;
double s[20],a[20],c[20],work[20],fmax;

/* compute auto-correlation and cross-correlation arrays */
fmax = 0.066+0.265*log((double)lsinc);
fmax = (fmax<1.0)?fmax:1.0;
for (j=0; j<lsinc; j++)
        a[j] = dsinc(fmax*j);
        c[j] = dsinc(fmax*(lsinc/2-j-1+d));


/* solve symmetric Toeplitz system for the sinc approximation */
stoepd(lsinc,a,c,s,work);
for (j=0; j<lsinc; j++)
        sinc[j] = s[j];
```

2 The algorithm design

2.1 Basic Theory of Interpolation

The sinc interpolation is the natural interpolation for band–limited data (such as seismic data for example). If data are perfecly band–limited, then it can be exactly written in the frequency domain as the product of their spectrum whith a box. The inverse Fourier transform of a box is a sinc function. So, in the time (or spatial) domain the band–limited function is a convolution of a discrete represention of the function with a sinc function. A perfect box in a frequency domain is an infinite domain sinc function in the time domain. If we want to have a finite duration wavelet in the time (or spatial) domain, we have to do some compromises. We could window the sinc function (Gaussian, Haning, Hamming, Kaiser, etc.) The WFI package picks to use the 2 to 20 by 2 sinc interpolation from the CWP library (program `mksinc.c`) by David Hale. Dave Hale's sinc interpolation fits the sinc coefficients to the ideal sinc by using least squares approach. I will try to explain Hale's algorithm here.

I like to approach the iterpolation ideas through the theory of transforms. In general a transform has the form of

$$f(x) = \int_I K(x,y)\, c(y) dy. \tag{2.1}$$

Where the function f is represented as a combination (supperposition) of some basis functions (the Kernel $K(x,y)$) with some coefficients ($c(y)$) over

some index domain I. In basic linear algebra we say that a vector can be decomposed as the sum of its projections along a given basis. If the basis represented in the kenel $K(x, y)$ are orthonormal, the coefficients are easily extracted with inner products of the function $f(x)$ with the base vectors represented by $K(x, y)$.

We will slowly transform equation 2.1 to our final interpolation form. The first step is to acknowledge that our space along the variable x is discrete and rewrite 2.1 as

$$f(x_i) = \int_I K(x_i, y)\, c(y) dy. \tag{2.2}$$

The next step is to think that $f(x_i)$ is obtained as a weighted average (weight of $K(x_i, y)$) over some continuum ($c(y)$) representation of its discrete collection $f(x_j)$. That is, we want:

$$f(x_i) = \int_I K(x_i, y)\, f(y) dy. \tag{2.3}$$

This is the main postulate behind interpolation.

Such a representation exists and one example is to pick

$$K(x_i, y) = \delta(x_i - y). \tag{2.4}$$

The sifting property of the Diract delta distribution stablishes that

$$f(x_i) = \int_I \delta(x_i - y) f(y) dy, \tag{2.5}$$

where we assume that the domain of indices I is continuous and contains the point x_i. However this representation is uninteresting. We should transform it so that it could be useful in our applications. There are an infinite number of representations of the delta distribution and each of them yields an interpolation scheme. *

For example let us assume the representation

$$K(x, y) = \delta(x - y) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-\mathrm{i}k(x-y)} dk. \tag{2.6}$$

* The Sturm–Liouville problem yieds a wealth of orthogonal functions that span Dirac delta distributions. A few of them are: Legendre polynomials, Hermite polynomials, Chebyshev polynomials, Laguerre Polynomials, trigonometrical polynomials (the exponential functions that create the sinc interpolation shown here), and so on.

We plug 2.6 into equation 2.5 and find:

$$f(x_i) = \int_I f(y) \left\{ \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-\mathrm{i}k(x-y)} dk \right\} dy, \tag{2.7}$$

Here we draw more assumptions. We assume that the range of base functions limited. That is our domain of k is band–limited. We assume that f will be sampled at uniform intervals Δy and so the domain of wavenumbers is limited to the interval $[-\pi/\Delta y, \pi/\Delta y]$. So

$$K(x,y) = \frac{1}{2\pi} \int_{-\pi/\Delta y}^{\pi/\Delta y} e^{-\mathrm{i}k(x-y)} dk = \frac{\sin[\frac{\pi(x-y)}{\Delta y}]}{\pi(x-y)} = \frac{1}{\Delta y} \mathrm{sinc}\left(\frac{x-y}{\Delta y} \right). \tag{2.8}$$

and so

$$f(x_i) = \int_I f(y) \, \frac{1}{\Delta y} \mathrm{sinc}\left(\frac{x-y}{\Delta y} \right) dy \tag{2.9}$$

The next step is to asume that the wavenumber domain space is discrete and rewrite the integral as a sum over the samples with sampling rate Δy. That is:

$$f(x_i) = \sum_j \mathrm{sinc}\left(\frac{x_i - y_j}{\Delta y} \right) f(y_j). \tag{2.10}$$

Note here that Δy canceled since this is precisely the sampling interval chosen for the function f. That is, the y_j samples are uniformly distributed ($y_0 + j\,\Delta y$). We want to interpolate x_i. If x_i is a point in the y_k sequence, then $x_i - y_j = n_j\,\Delta y$ for some integer n_j and so when the point x_i coincides with the point y_j the sinc function will evaluate to 1 and in all the other cases to zero. So we get the identity $f(x_i) = f(y_j)$, with $(x_i = y_j)$. The interesting case, and that what sinc interpolation is about, is when the point x_i does not belong to the dataset with elements y_j.

At this point we will simplify equation 2.10 by assuming that $y_0 = 0$ and $\Delta y = 1$. We further assume that $x_i = i + d$ where $0 \leq d \leq 1$ and $y_j = j$. These assumptions should not affect our solution to the least square problem since the problem depends only on the fractional distance d and the number of coefficients used for the interpolation (not yet discussed here). With this, equation 2.10 turns out to be:

$$f(i+d) = \sum_{j=-\infty}^{\infty} \mathrm{sinc}(i+d-j) f(j). \tag{2.11}$$

2.2 The Least Squares Formulation

Now we are ready to start the formulation of the least squares problem. We want to find some set of kernel coefficients k_i such that we can approximate $f(i+d)$ with the formula

$$y(i+d) = \sum_{j=0}^{L} k_j(d)\, f[i+j+1-(L+1)/2]. \tag{2.12}$$

This looks like a complicated expression but it is because I want to stick to Dave Hale's formulation. Equation 2.12 corresponds to Hale's equation in the code documentation where:

Here	Hale
L	lsinc $-$ 1
$k_j(d)$	sinc[j]
$f[i+j+1-(L+1)/2]$	y(i + j + 1 $-$ lsinc/2).

(2.13)

Note the dependency of k from d. For each fractional d we can have a different set of L coefficients.

The problem will not change if we drop the variable i from equations 2.11 and 2.12. So, for simplication purposes, we rewrite these equations as

$$f(d) = \sum_{j=-\infty}^{\infty} \mathrm{sinc}(d-j) f(j). \tag{2.14}$$

$$y(d) = \sum_{j=1}^{L} k_j(d)\, f[j-L/2]. \tag{2.15}$$

where now $L = \mathrm{lsinc} - 2$ and we start the sum from 1 instead of from 0. From the frequency domain representation we find

$$\begin{aligned} E = f(d) - y(d) &= \frac{1}{2\pi} \int_{-\pi}^{\pi} F(\omega) \left(e^{\mathrm{i}\omega d} - \sum_j k_j e^{\mathrm{i}\,\omega(j-L/2)} \right) d\omega \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} F(\omega)\, e^{\mathrm{i}\omega d} \left(1 - \boldsymbol{k}^T \boldsymbol{x}\right) d\omega, \end{aligned} \tag{2.16}$$

where

$$\boldsymbol{k}(d) = \begin{pmatrix} k_1(d) \\ k_2(d) \\ . \\ . \\ k_L(d) \end{pmatrix} \tag{2.17}$$

and

$$\boldsymbol{x} = \begin{pmatrix} e^{\mathrm{i}\omega(1-d-L/2)} \\ e^{\mathrm{i}\omega(2-d-L/2)} \\ . \\ . \\ e^{\mathrm{i}\omega(L-d-L/2)} \end{pmatrix} \tag{2.18}$$

The objective is to minimize $|E|$ among all the possible L–tuples of $\boldsymbol{k}$s. Since $F(\omega)$ and $e^{\mathrm{i}\,\omega d}$ are independent of $\boldsymbol{k}$ the minimization is achieved by that of $1 - \boldsymbol{k}^T\,\boldsymbol{x}$. The least square problem is then

$$\boldsymbol{k}^T\boldsymbol{x} = 1. \tag{2.19}$$

Since $\boldsymbol{k}$ is the domain of minimization, then we rewrite 2.19 as

$$\boldsymbol{x}^T\boldsymbol{k} = 1. \tag{2.20}$$

This is one equation with L unknowns (the vector $\boldsymbol{k}$). The standard least square problem is done by building the normal equations. That is, by multiplying both sides of 2.20 by the conjugate transpose (the adjoint) of $\boldsymbol{x}^T$. If the matrix $\tilde{\boldsymbol{x}}\boldsymbol{x}^T$ is not singular (and it is in this case) then the system to solve is

$$\tilde{\boldsymbol{x}}\boldsymbol{x}^T\boldsymbol{k} = \tilde{\boldsymbol{x}}. \tag{2.21}$$

Let us define the matrix $A(\omega)$ as

$$A = \tilde{\boldsymbol{x}}\boldsymbol{x}^T \tag{2.22}$$

an entry of this matrix can be written as

$$A_{ij}(\omega) = \tilde{\boldsymbol{x}}_i\,\boldsymbol{x}_j = e^{-\mathrm{i}\omega(i-d-L/2)}\,e^{\mathrm{i}\omega(j-d-L/2)} = e^{\mathrm{i}\omega\,(j-i)}. \tag{2.23}$$

in the time domain (the d domain) the inverse Fourier transform of $A(\omega)$ provides

$$a_{ij}(d) = \text{sinc}(F\,(i-j)) \tag{2.24}$$

which is a Toeplitz matrix. Here F is the maximum frequency to use (a fraction of Nyquist). We have picked so far as the maximum frequency the Nyquist ($F = 1$), but Hale chosed to compute F following the formula:

$$F = \text{fmax} = \min(0.066 + 0.265 * \log(\text{lsinc}), 1.0). \tag{2.25}$$

He says that this equation is empirically derived from a Western Geophysical Technical Memorandum by Ken Larner. I would like to see that memorandum.

The inverse Fourier transform for $\tilde{\boldsymbol{x}}$ yields

$$c_i(d) = \text{sinc}(F\,(i-d-L/2)) \tag{2.26}$$

and since in Hale's notation $L =$ lsinc-2, we can write

$$c_i(d) = \text{sinc}(F\,(i-d+1-\text{lsinc}/2)). \tag{2.27}$$

Now given that the function sinc is even we can write the previous equation as

$$c_j(d) = \tilde{\boldsymbol{x}}_j = \text{sinc}(F(\text{lsinc}/2-j-1+d)) \tag{2.28}$$

The reader can verify that $c_j = c[j]$ corresponds with the $c[j]$ in Hale's code and that $a_{ij}(d)$ corresponds with the Toeplitz matrix $a[j]$ in Hale's code.

3 The use of the algorithm to interpolate data

The code `intsinc8.c` in the same directory of `mksinc.c` shows examples of how to interpolate using the `mksinc.c` utility. A look up table for the interpolation is designed. In this particular case $L = 8$ (8 point sinc interpolator) and the number of fractional elements d is NTABLE=513. That is the unit is splitted into 513 fractions for the form $d = 1.0/j$ with $j = 1..512$ (the fraction d=0 and d=1 are considered as arrays with zeroes and a 1 in the middle.) For each fraction d an 8 point ($\boldsymbol{k}$) filter is computed so that equation 2.12 can be applied. Figure 1 shows the 513 graphs of 8 point each. As we move from left to right, the decimal points d move from 0 to 1 with increments of 1/512.

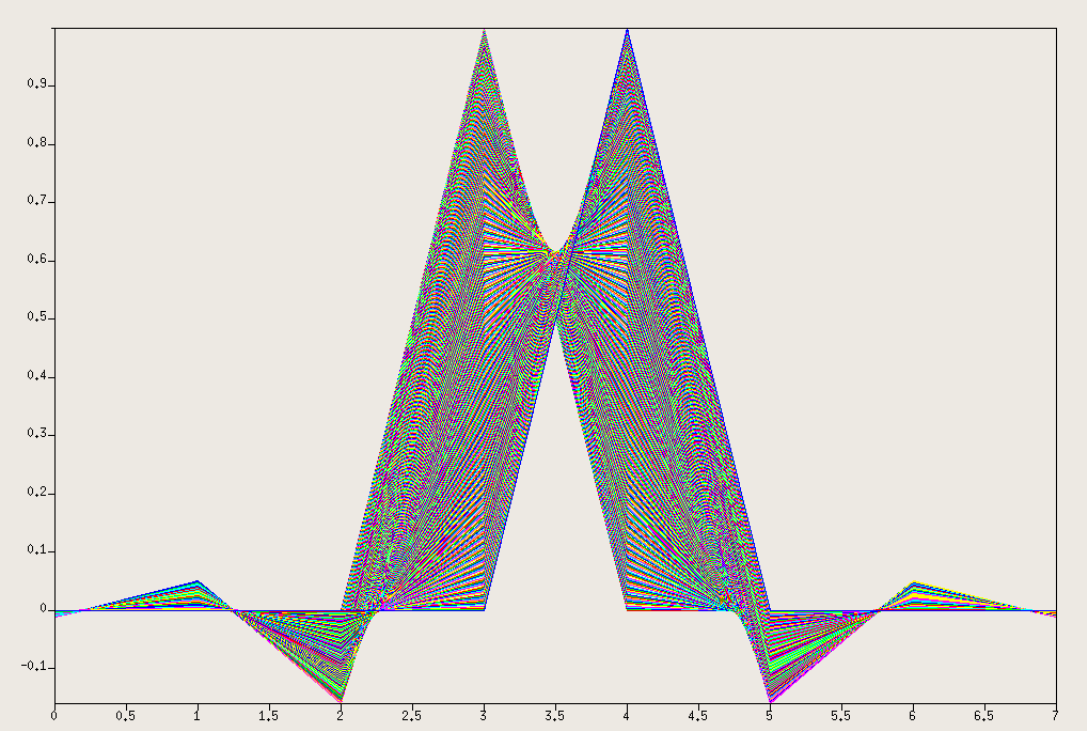

Figure 1: Plot of table of 513 stencils with 8 coefficients each, used to sinc interpolate