Discrete Wiener Interpolation

TM #12
J. P. Lewis

Computer Graphics Laboratory
New York Institute of Technology

Wiener interpolation differs from polynomial interpolation approaches in that it is based on the expected correlation of the data. Wiener interpolation of discrete data is simple, requiring only the solution of a matrix equation. This note describes two derivations for discrete Wiener interpolation.

Some advantages of Wiener interpolation are:

- The data can be arbitrarily spaced.
- The algorithm applies without modification to multi-dimensional data.
- The interpolation can be made local or global to the extent desired. This is achieved by adjusting the correlation function so that points beyond a desired distance have a negligible correlation.
- The interpolation can be as smooth as desired, for example an analytic correlation function will result in an analytic interpolated curve or surface.
- The interpolation need not be "smooth", for example, the correlation can be negative at certain distances, oscillatory, or (in several dimensions) have directional preferences.
- The algorithm provides an error or confidence level associated with each point on the interpolated surface.
- The algorithm is optimal by a particular criterion (below) which may or may not be relevant.

Some disadvantages of Wiener interpolation:

- It requires knowing or inventing the correlation function. While this may arise naturally from the problem in some cases, in other cases it would require interactive access to the parameters of some predefined correlation models to be "natural".

- It requires inverting a matrix whose size is the number of significantly correlated data points. This can be a practical problem, depending on the available matrix software. The correlation functions which are desirable for smooth interpolation (e.g. Gaussian correlation) are also harder to invert; this is the problem of "singular processes" in Wiener interpolation theory [1]. In my experience a local area of only about 10 data points can be accurately interpolated with a standard Gaussian elimination routine. More sophisticated matrix inversion routines would eliminate this problem. In particular, for singular processes pseudoinverses apparently avoid the matrix inversion difficulties [2].

Terminology

Symbols used below:

x value of a stochastic process at time t

$\hat{x}$ estimate of x

x_j observed values of the process at times or locations t_j

The derivations require two concepts from probability:

- The correlation of two values is the expectation of their product, $\mathbf{E}[xy]$. The autocorrelation or autocovariance function is the correlation of pairs of points from a process:
$$C(t_1, t_2) = \mathbf{E}\left\{x(t_1)x(t_2)\right\}$$
For a stationary process this expectation is a function only of the distance between the two points: $C(\tau) = \mathbf{E}[x(t)x(t+\tau)]$. The variance is the value of the autocorrelation function at zero: $\mathrm{var}(x) = C(0)$. (Auto)covariance usually refers to the correlation of a process whose mean is removed and (usually) whose variance is normalized to be one. There are differences in the terminology, so "Correlation function" will mean the autocovariance function of a normalized process here.

- Expectation behaves as a linear operator, so any factor or term which is known can be moved "outside" the expectation. For example, assuming a and b are known,
$$\mathbf{E}\left\{ax + b\right\} = a\mathbf{E}[x] + b$$
Also, the order of differentiation and expectation can be interchanged, etc.

Definition

Wiener interpolation estimates the value of the process at a particular location as a weighted sum of the observed values at some number of other locations:

$$\hat{x} = \sum a_j x_j \tag{1}$$

The weights a_j are chosen to minimize the expected squared difference or error between the estimate and the value of the "real" process at the same location:

$$\mathbf{E}\left\{(x - \hat{x})^2\right\} \tag{2}$$

The reference to the "real" process in (2) seems troublesome because the real process may be unknowable at the particular location, but since it is the *expected* error which is minimized, this reference disappears in the solution.

Wiener interpolation is optimal among linear interpolation schemes in that it minimizes the expected squared error (2). When the data have jointly Gaussian probability distributions (and thus are indistinguishable from a realization of a Gaussian stochastic process), Wiener interpolation is also optimal among nonlinear interpolation schemes.

Derivation #1

The first derivation uses the "orthogonality principle": the squared error of a linear estimator is minimum when the error is 'orthogonal' in expectation to all of the known data, with 'orthogonal' meaning that the expectation of the product of the data and the error is zero:

$$\mathbf{E}\left\{(x - \hat{x})x_k\right\} = 0 \text{ for all } j$$

Substituting $\hat{x}$ from (1),

$$\mathbf{E}\left\{(x - \sum a_j x_j)x_k\right\} = 0 \tag{3}$$

$$\mathbf{E}\left\{xx_k - \sum a_j x_j x_k\right\} = 0$$

The expectation of xx_k is the correlation $C(t - t_k)$, and likewise for $x_j x_k$:

$$C(t - t_k) = \sum a_j C(t_j - t_k)$$

or

$$\mathbf{Ca} = \mathbf{c} \tag{4}$$

This is a matrix equation which can be solved for the coefficients a_j. The coefficients depend on the positions of the data x_j though the correlation function, but not on the actual data values; the values appear in the interpolation (1) though. Also, (4) does not directly involve the dimensionality of the data. The only difference for multi-dimensional data is that the correlation is a function of several arguments: $\mathbf{E}[PQ] = C(x_p - x_q, y_p - y_q, z_p - z_q, \ldots)$.

Derivation #2

The second derivation minimizes (2) by differentiating with respect to each a_k. Since (2) is a quadratic form (having no maxima), the identified extreme will be a minimum (intuitively, a squared difference (2) will not have maxima).

$$\frac{d}{da_k}\left[\mathbf{E}\left\{(x-\sum a_j x_j)^2\right\}\right] = \mathbf{E}\left[\frac{d}{da_k}(x-\sum a_j x_j)^2\right] = 0$$
$$2\mathbf{E}\left\{(x-\sum a_j x_j)\frac{d}{da_k}(x-\sum a_j x_j)\right\} = 0$$
$$\mathbf{E}\left\{(x-\sum a_j x_j)x_k\right\} = 0$$

which is (3).

Cost

From (4) and (1), the coefficients a_j are $\mathbf{a} = \mathbf{C}^{-1}\mathbf{c}$, and the estimate is $\hat{x} = \mathbf{x}^T\mathbf{C}^{-1}\mathbf{c}$. The vector $\mathbf{c}$ changes from point to point, but $\mathbf{x}^T\mathbf{C}^{-1}$ is constant for given data, so the per point estimation cost is a dot product of two vectors whose size is the number of data points. A C program using this approach is listed in the appendix.

Confidence

The interpolation coefficients a_j were found by minimizing the expected squared error (2). The resulting squared error itself can be used as a confidence measure for the interpolated points. For example, presumably the error variance would be high away from the data points if the data are very uncharacteristic of the chosen correlation function. The error at a particular point is found by expanding (2) and substituting $\mathbf{a} = \mathbf{C}^{-1}\mathbf{c}$:

$$\begin{aligned}
\mathbf{E}\left\{(x-\hat{x})^2\right\} &= \mathbf{E}\left\{(x-\sum a_j x_j)^2\right\} \\
&= \mathbf{E}\left\{x^2 - 2x\sum a_j x_j + \sum a_j x_j \sum a_k x_k\right\} \\
&= \text{var}(x) - 2\sum a_j C(t,t_j) + \sum a_j a_k C(t_j,t_k) \\
\text{(switching to matrix notation)} & \\
&= C(0) - 2\mathbf{a}^T\mathbf{c} + \mathbf{a}^T\mathbf{C}\mathbf{a} \\
&= C(0) - 2(\mathbf{C}^{-1}\mathbf{c})^T\mathbf{c} + (\mathbf{C}^{-1}\mathbf{c})^T\mathbf{C}\mathbf{C}^{-1}\mathbf{c} \\
&= C(0) - \mathbf{c}^T\mathbf{C}^{-1}\mathbf{c} \\
&= C(0) - \sum a_j C(t,t_j)
\end{aligned}$$

References

[1] Yaglob, A., *An Introduction to the Theory of Stationary Random Functions*, Dover, New York, 1973.

[2] Deutsch, R. *Estimation Theory*, Prentice-Hall, New Jersey, 1965.

Appendix

A C program for Wiener interpolation is listed in the following pages.

```
/* wiener interpolation program
 * zilla 25jul86
 */


/* wiener interpolation "control block" structure */

struct WINcb {
    int N;              /* number of data points */
    float *WK;          /* allocated coefficient array [N] */
    float (*R)();       /* correlation function */
};

/* given data points x[n],y[n] and correlation function R1(),
 * compute the constant vector x*inv(C) and return a
 * structure containing the precomputed vector.
 */

char *WINsetup(float x[],float y[],int n,float (*R1)())
{
    struct WINcb *wcb;
    register float *k,*wk;
    float det;
    register int i,j;
#   define K(j,i,n) k[(j)*(n)+(i)]

    wcb = (struct WINcb *)malloc(sizeof(struct WINcb));
    wcb->N = n;
    wcb->R = R1;
    wk = wcb->WK = (float *)malloc(n * sizeof(float));
    k = (float *)alloca(n * n * sizeof(float));

    /* correlation matrix in k */
```

```
    for( j=0; j < n; j++ ) {
        for( i=0; i < n; i++ ) {
            K(j,i,n) = (*R1)(x[j] - x[i]);
        }
    }

    /* invert k */
    minv(k,n,&det);

    /* precompute wk = x*inv(k) */
    for( j=0; j < n; j++ ) {
        wk[j] = 0.0;
        for( i=0; i < n; i++ ) {
            wk[j] += y[i] * K(j,i,n);
        }
    }

    return((char *)wcb);
} /*WINsetup*/


/* given control block and data ordinates xd[n],
 * interpolate the range x=minx..maxx into table f[len]
 */

void WINtab(struct WINcb *wcb,float xd[],float f[],int len,
            float minx,float maxx)
{
    register float x,y; float dx,flenm1;
    register float *wk;
    int t;
    register int i,n;
    float (*R)();

    dx = maxx - minx;
    flenm1 = (float)(len-1);
    wk = wcb->WK;
    n = wcb->N;
    R = wcb->R;

    for( t=0; t < len; t++ ) {
        x = minx + dx * ((float)t/flenm1);
        y = 0.0;
        for( i=0; i < n; i++ ) {
            y += wk[i] * (*R)(x - xd[i]);
        }
```

```
        f[t] = y;
    }
} /*tab*/


/* deallocate control block */

void WINfinit(struct WINcb *wcb)
{
    free((char *)wcb->WK);
    free((char *)wcb);
}
```