Least Squares Methods
What is SPH ?

A Special Talk to CML Students

Noboru Kikuchi

February 1 & 7, 2000

Least Squares Method

Discrete Case

$$\min_{c_k} \frac{1}{2} \sum_{i=1}^{n+1} \left(f_i - \sum_{k=1}^{m+1} c_k \phi_k \left(x_i \right) \right)^2$$

x_i = smpling points

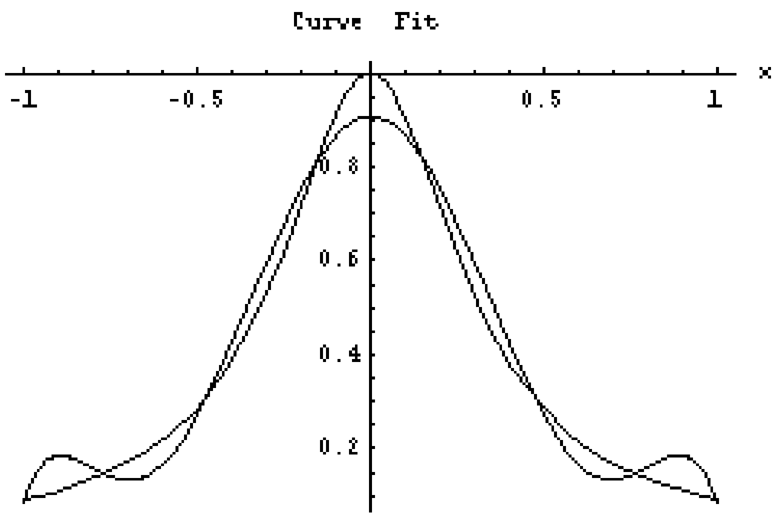
f_i = data

$\phi_i(x)$ = basis functions

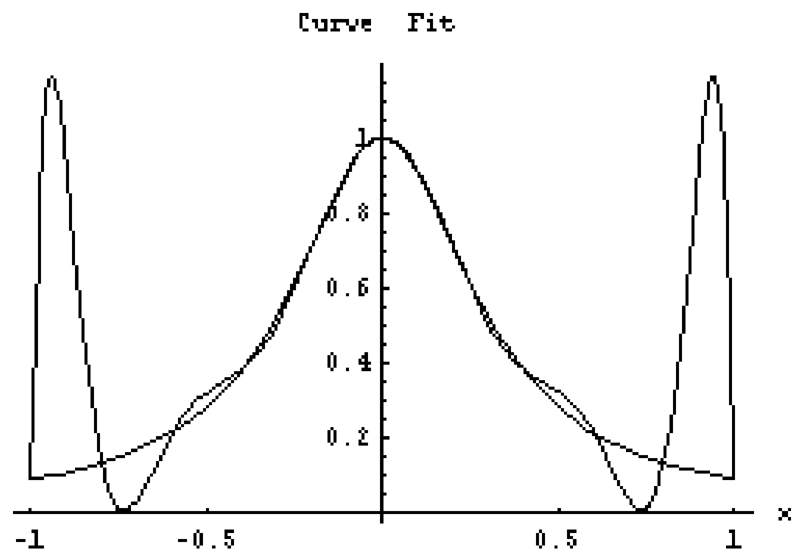
Example : Curve Fitting

11 sampling points

$$f_i = \frac{1}{1+10x_i^{\ 2}} \ , \ x_i = -1 + 2\frac{i-1}{n} \ , n = 10$$



6th order polynomial



14th order polynomial

Remarks

Degree of polynomials is small, then the values need not be Recovered by the least squares

Degree of polynomials is large, it becomes interpolation

When the degree is larger than the number of sampling, then Use pinv or PseudoInverse instead of inv or Inverse

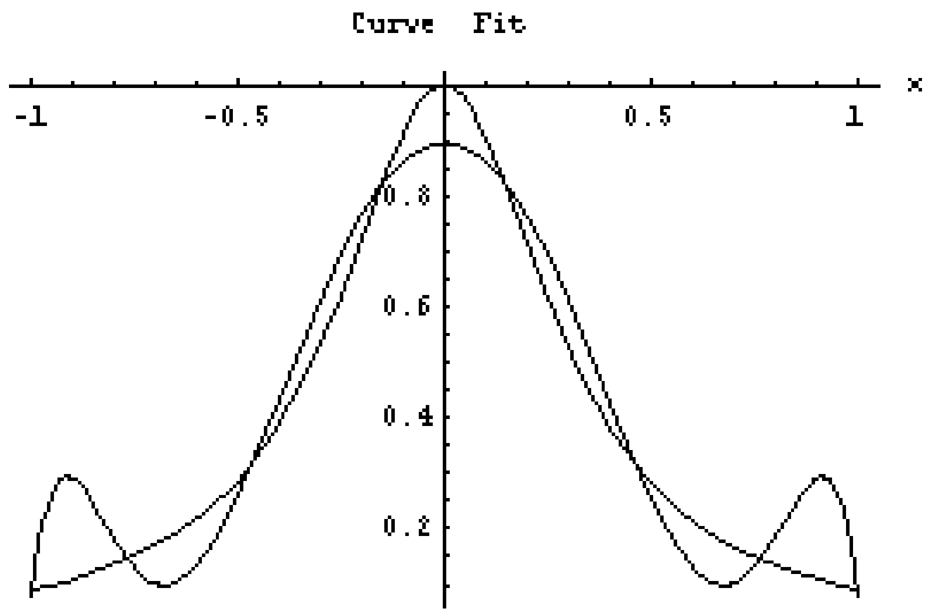
Weighted Least Squares

$$\min_{c_i} \frac{1}{2} \sum_{i=1}^{n+1} \left\{ \sum_{j=1}^{n+1} w_{ij} \left(f_j - \sum_{k=1}^{m+1} c_k \phi_k \left(x_j \right) \right) \right\}^2$$

$$w_{ij} = w_i \left(x_j \right)$$

$$w_i \left(x \right) = \exp\left(-\alpha \left| x - x_i \right|^{\beta} \right)$$

Result : Not Much Use



We can equally distribute the approximation error

Constrained Least Squares

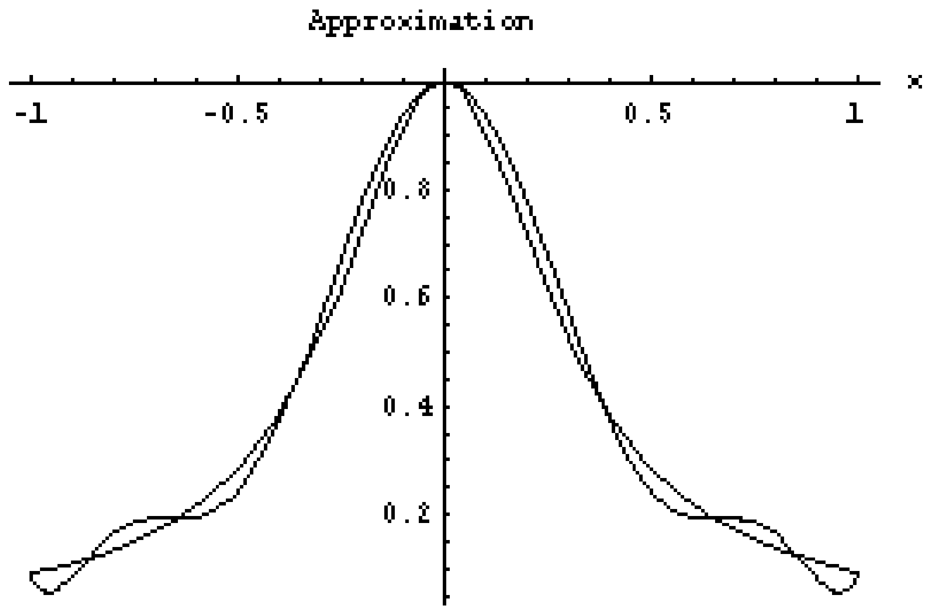
$$\max_{\lambda_j} \min_{c_k} \frac{1}{2} \sum_{i=1}^{n+1} \left(f_i - \sum_{k=1}^{m+1} c_k \phi_k \left(x_i \right) \right)^2 - \sum_{j=1}^{j_{\max}} \lambda_j \left(f_j - \sum_{k=1}^{m+1} c_k \phi_k \left(x_j \right) \right)$$

At x_j ' we should satisfy $f_j = \sum_{k=1}^{m+1} c_k \phi_k \left(x_j ' \right)$

for $j = 1, 2, ..., j_{\max}$

Appliction of Lagrange Multiplier Method

Result : Great



Least Squares Method

Continuous Case

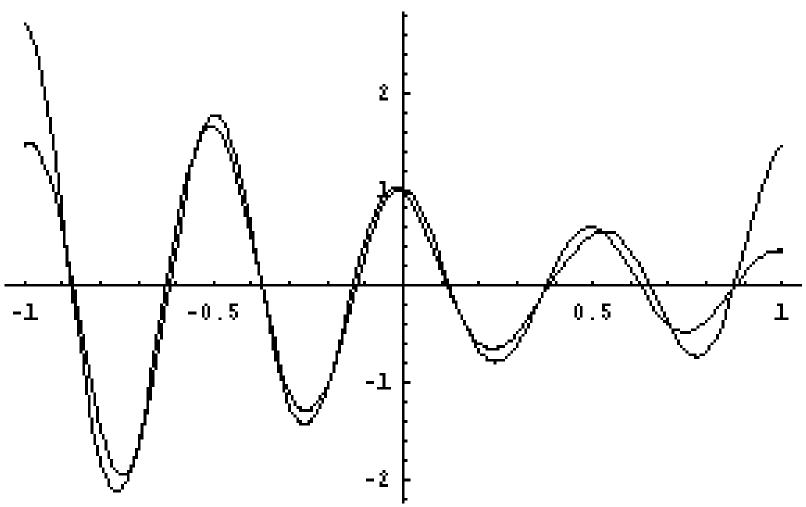
Without Constraint

$$\min_{c_i} \frac{1}{2} \left\| f - f_n \right\|^2 = \min_{c_i} \frac{1}{2} \left\| f - \sum_{i=1}^{n} c_i \phi_i(x) \right\|^2$$

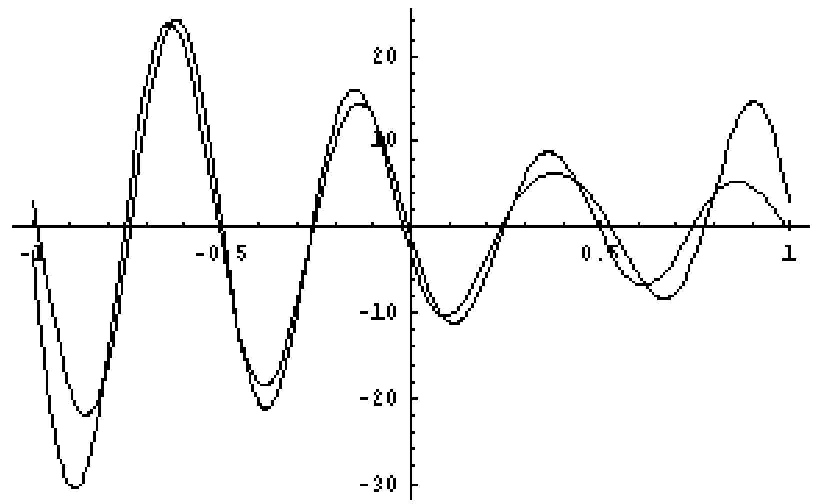
With Constraint

$$\max_{\lambda_j} \min_{c_i} \frac{1}{2} \left\| f - \sum_{i=1}^{n+1} c_i \phi_i(x) \right\|^2 - \sum_{j=1}^{m+1} \lambda_j \left(f(x_j') - \sum_{i=1}^{n+1} c_i \phi_i(x_j') \right)$$

Example



Function itself



Its First derivative

Discrete & Continuous

How to make continuous least squares from discrete data? Origin of SPH

$$\min_{a(x)} \frac{1}{2} \sum_{i=1}^{n+1} w_i(x)\left(f_i - a(x)\right)^2$$

Taking the first variation in a(x) yields

$$\left(\sum_{i=1}^{n+1} w_i(x)\right) a(x) = \sum_{i=1}^{n+1} w_i(x)\, f_i$$

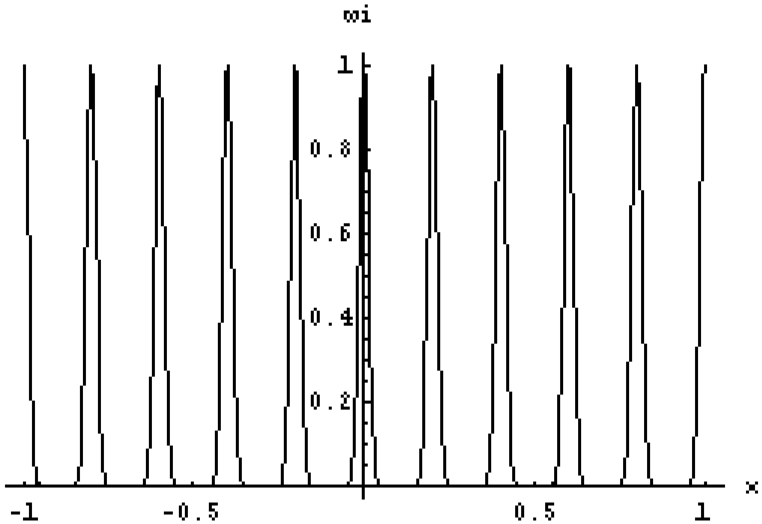
SPH with “1”

$$f\left(x\right)\approx\left\{f_1,\ldots,f_{n+1}\right\}\approx a\left(x\right)=\sum_{i=1}^{n+1} f_i\,\frac{w_i\left(x\right)}{\sum_{j=1}^{n+1} w_j\left(x\right)}$$

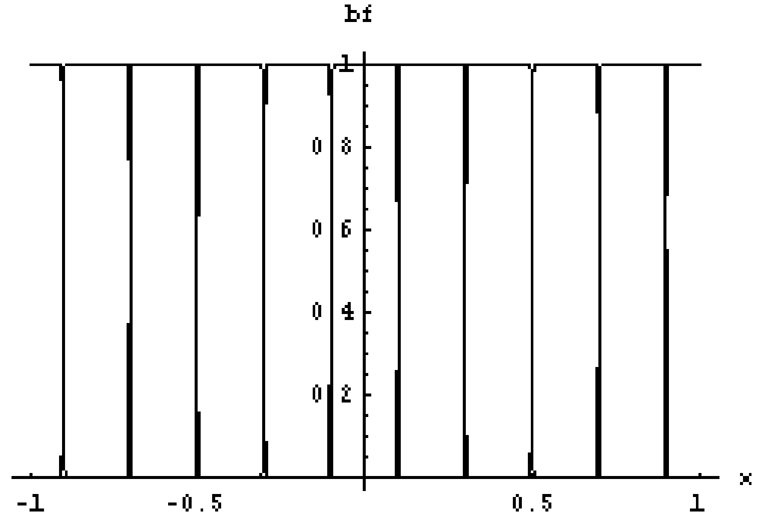
$$\phi_i\left(x\right)=\frac{w_i\left(x\right)}{\sum_{j=1}^{n+1} w_j\left(x\right)}\qquad,\qquad j=1,2,\ldots,n+1$$

Property of SPH Basis Functions

When the parameter α is very large

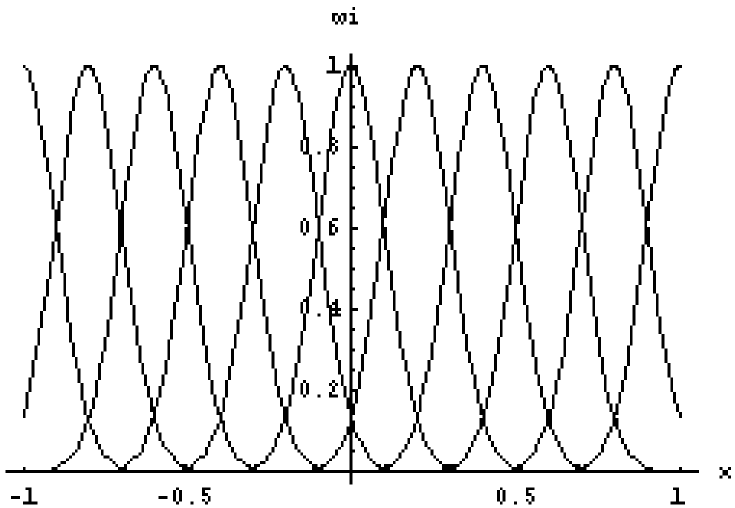


Functions $w_i(x)$

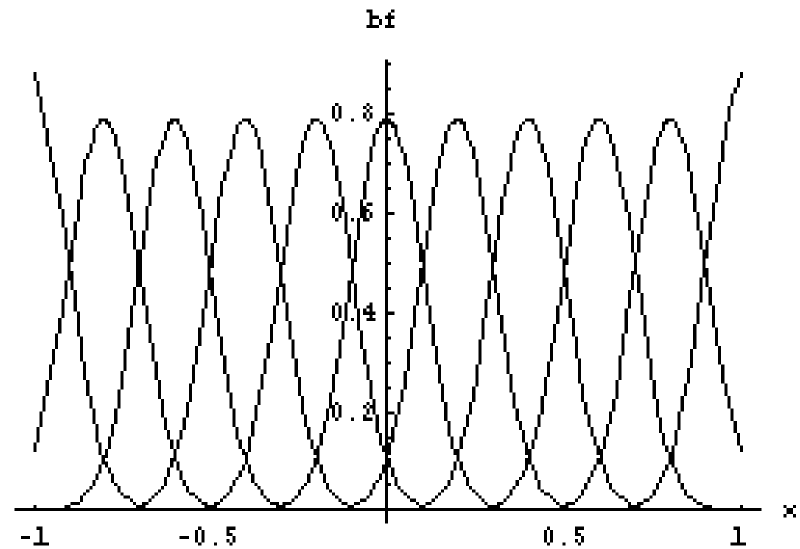


Basis Functions $\phi_i(x)$

When the parameter α is reasonably small, then
These basis functions are similar with fem

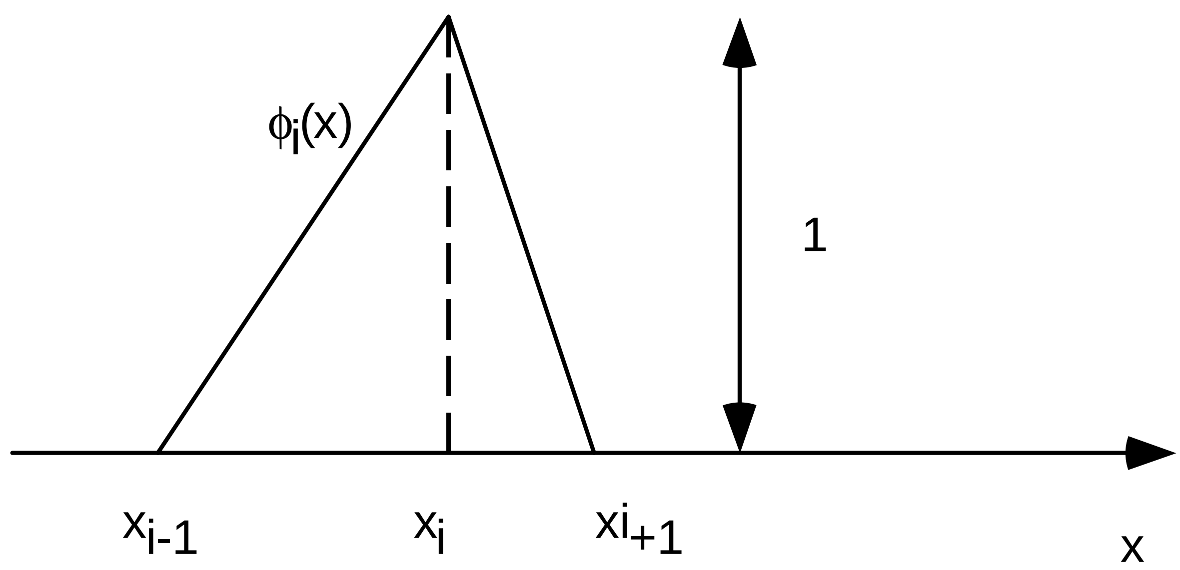


Functions $w_i(x)$



Basis Functions $\phi_i(x)$

Choice of FEM Shape Functions



Then the moving least squares implies piecewise linear Interpolation as in the finite element method.

Belytschko's EFG Method

$$f(x) \approx f_k(x) = \mathbf{a}(x)^T \mathbf{p}(x) \quad , \quad \mathbf{a}(x)^T = \{a_0(x), a_1(x), \ldots, a_k(x)\}$$

$$\mathbf{p}(x)^T = \{1, x, \ldots\ldots, x^k\}$$

Moving Least Squares Method

$$\min_{\mathbf{a}(x)} \frac{1}{2} \sum_{i=1}^{n+1} w(x_i - x)\left(f_i - \mathbf{a}(x)^T \mathbf{p}(x_i)\right)^2$$

$$w(x) = \exp\left(-\alpha x^{\beta}\right) \quad , \quad \alpha, \beta > 0$$

Necessary Condition

$$\sum_{i=1}^{n+1} \mathbf{p}(x_i) w(x_i - x) \mathbf{p}(x_i)^T \mathbf{a}(x) = \sum_{i=1}^{n+1} f_i w(x_i - x) \mathbf{p}(x_i)$$

$$\Leftrightarrow$$

$$\mathbf{a}(x) = \left[\sum_{j=1}^{n+1} \mathbf{p}(x_j) w(x_j - x) \mathbf{p}(x_j)^T \right]^{-1} \sum_{i=1}^{n+1} f_i w(x_i - x) \mathbf{p}(x_i)$$

Approximation

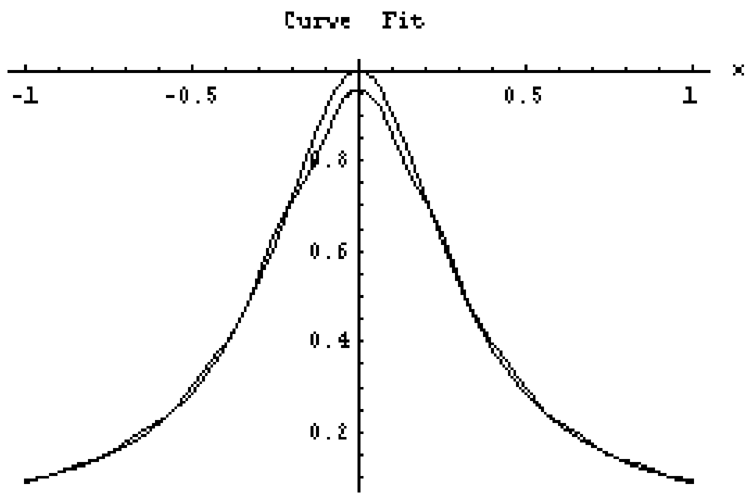
$$f_k(x) = \mathbf{p}(x)^T \mathbf{a}(x) = \mathbf{p}(x)^T \left[\sum_{i=1}^{n+1} \mathbf{p}(x_i) w(x_i - x) \mathbf{p}(x_i)^T \right]^{-1} \sum_{i=1}^{n+1} f_i w(x_i - x) \mathbf{p}(x_i)$$

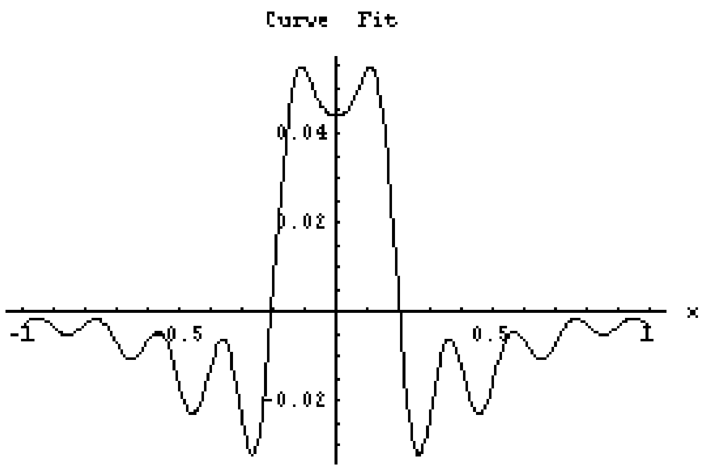
$$= \sum_{i=1}^{n+1} \left\{ \mathbf{p}(x)^T \left[\sum_{j=1}^{n+1} \mathbf{p}(x_j) w(x_j - x) \mathbf{p}(x_j)^T \right]^{-1} w(x_i - x) \mathbf{p}(x_i) \right\} f_i$$

Final Form of the Shape function

$$\phi_i(x) = \mathbf{p}(x)^T \left[\sum_{j=1}^{n+1} \mathbf{p}(x_j) w(x_j - x) \mathbf{p}(x_j)^T \right]^{-1} w(x_i - x) \mathbf{p}(x_i)$$

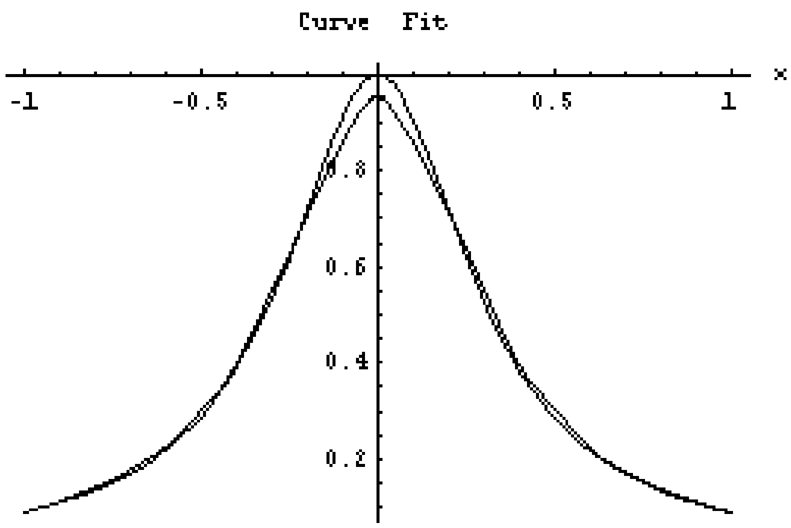
Application 1
Constant SPH

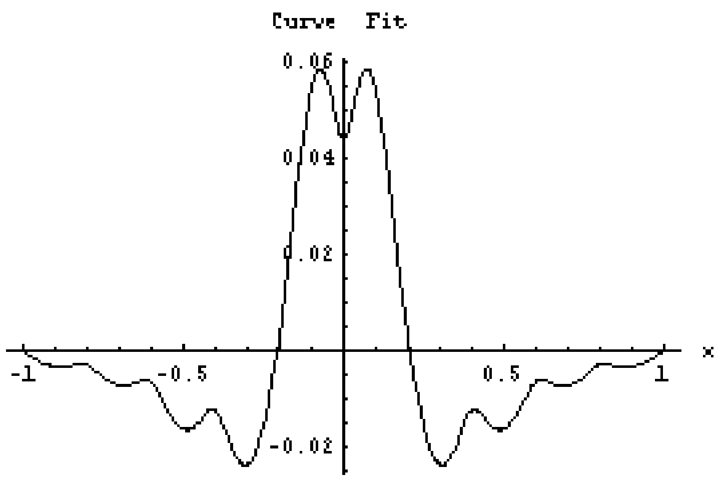




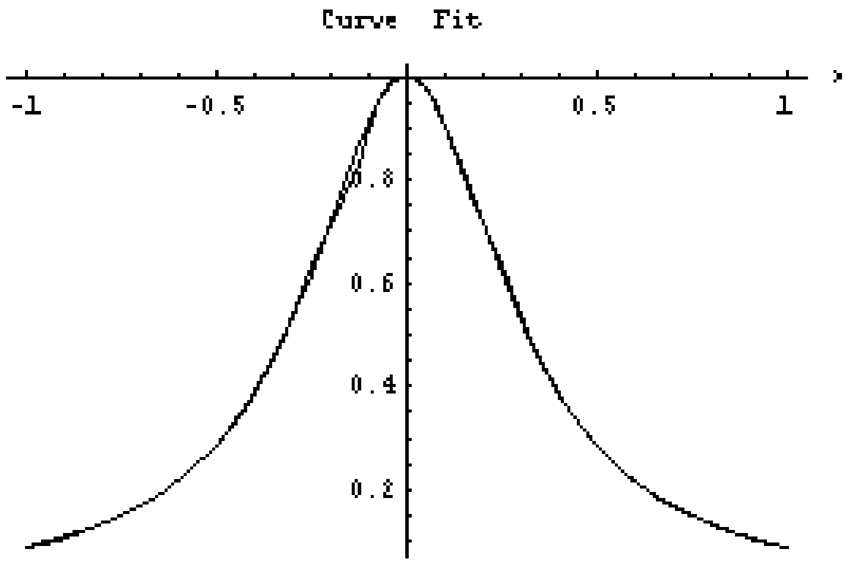
Application 2

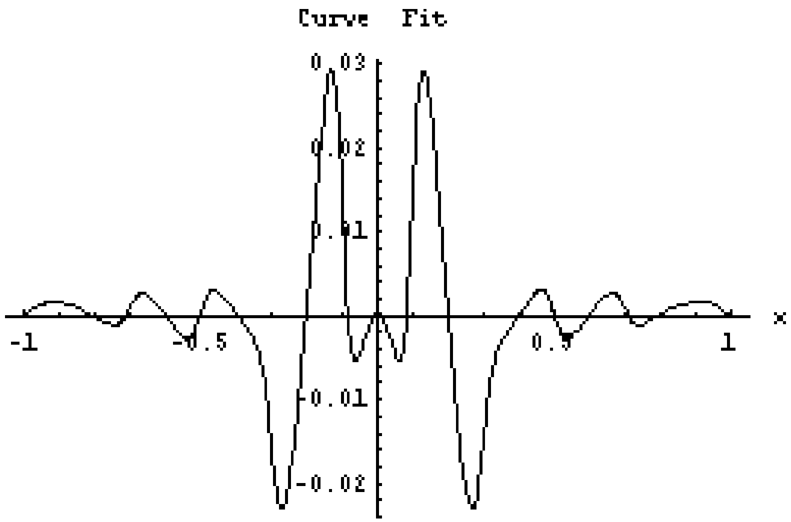
Linear Polynomial



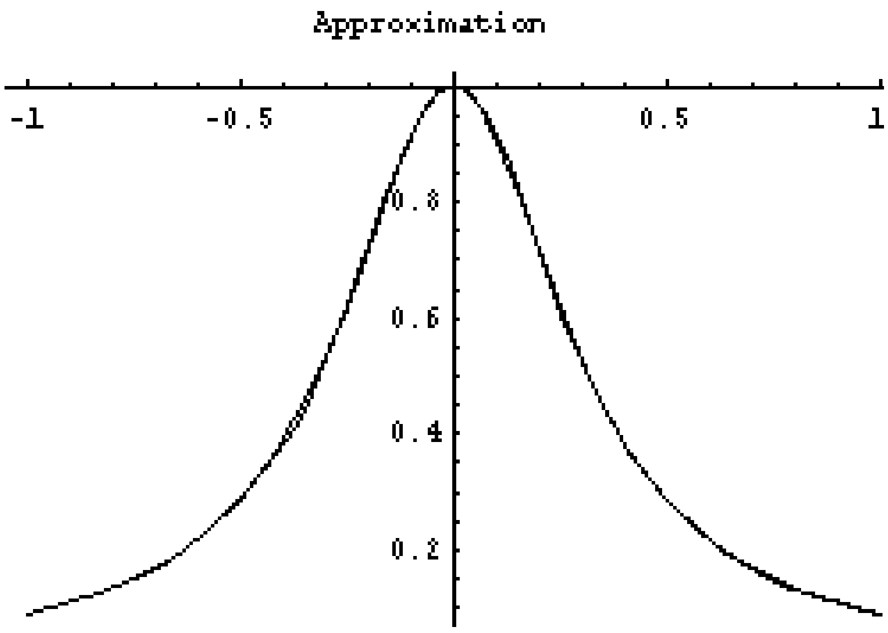


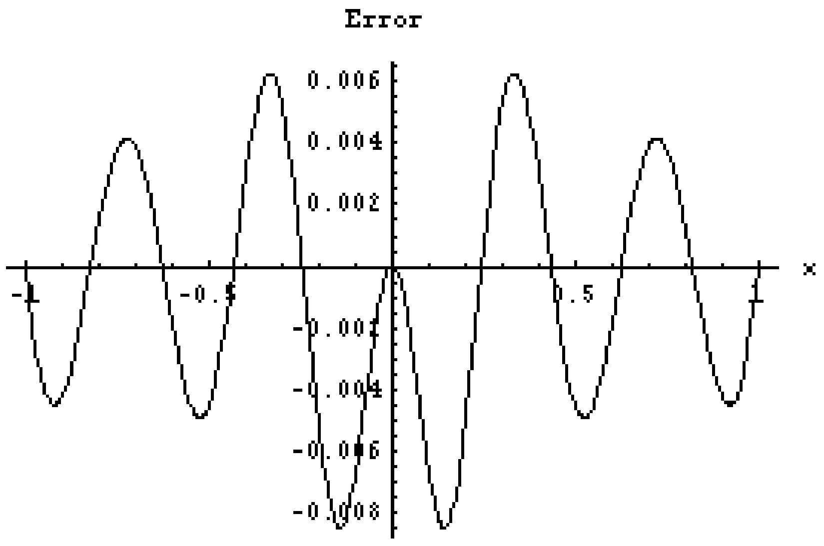
Application 3
quadratic polynomial





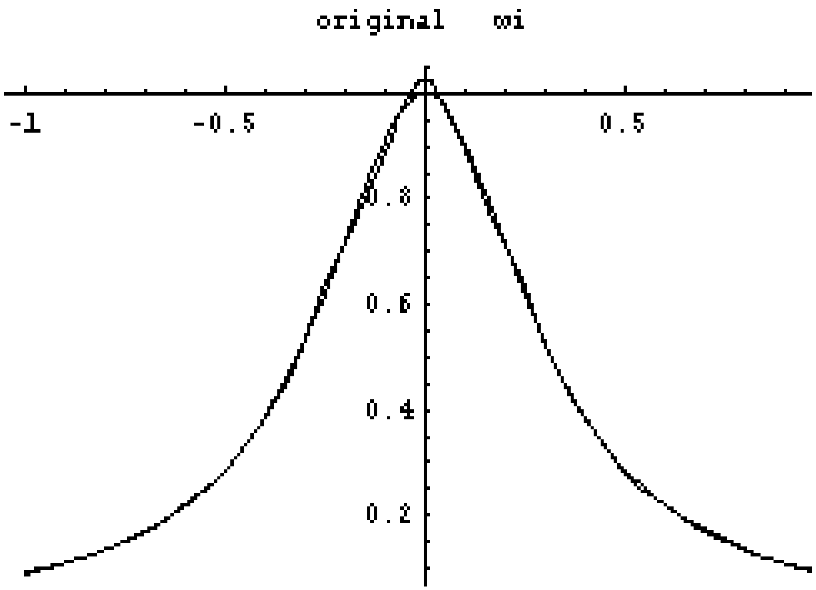
Constant SPH with Constraint

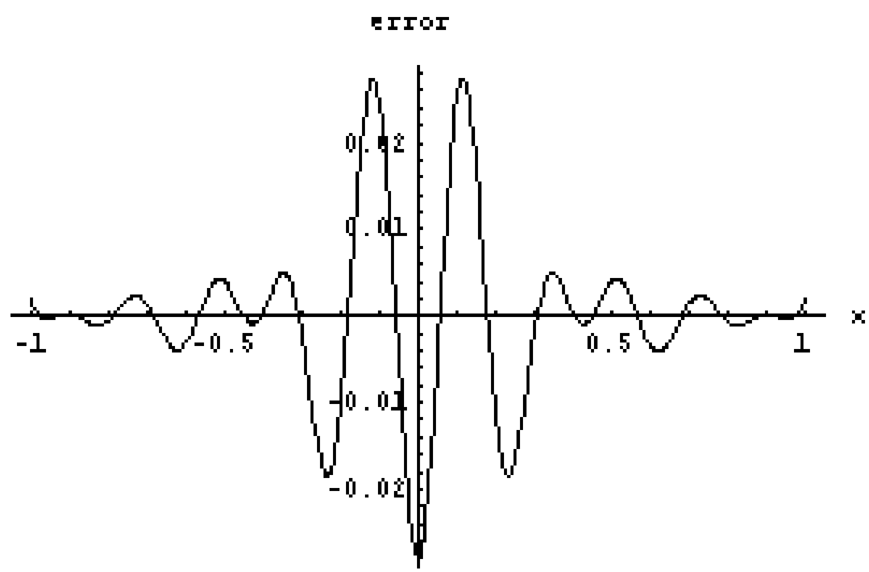




However !

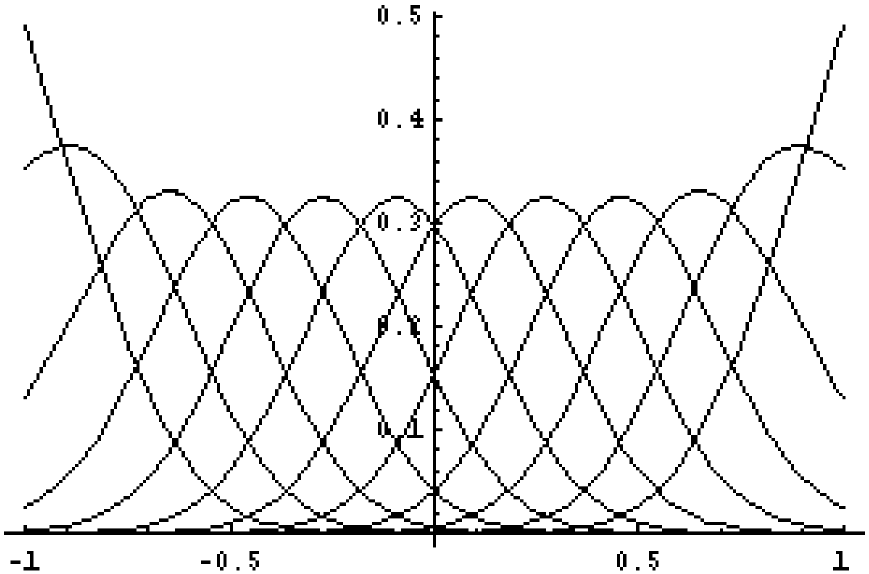
Using $f(x) \approx \sum_{i=1}^{n+1} c_i w_i(x)$ directly





Do we need extra calculation for its approximation ?

SPH Basis Functions



Parameter alpha is small

Bezier Type Spline

Parameter alpha is large

B-spline Type Spline

Different Derivation

Recovering global polynomial form

$$\phi_i(x) = \{a_0(x) + x_i a_1(x)\} w_i(x) = \{1 \quad x_i\} w_i(x) \begin{Bmatrix} a_0(x) \\ a_1(x) \end{Bmatrix} \quad , \quad i = 1, \ldots, n$$

(Constant & Linear Recovering)

$$1 = \sum_{i=1}^{n} \phi_i(x) = \sum_{i=1}^{n} \{1 \quad x_i\} w_i(x) \begin{Bmatrix} a_0(x) \\ a_1(x) \end{Bmatrix}$$

and

$$x = \sum_{i=1}^{n} x_i \phi_i(x) = \sum_{i=1}^{n} \{x_i \quad x_i^{\ 2}\} w_i(x) \begin{Bmatrix} a_0(x) \\ a_1(x) \end{Bmatrix}$$

$$\begin{Bmatrix} a_0(x) \\ a_1(x) \end{Bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} w_i(x) & \sum_{i=1}^{n} x_i w_i(x) \\ \sum_{i=1}^{n} x_i w_i(x) & \sum_{i=1}^{n} x_i^{\,2} w_i(x) \end{bmatrix}^{-1} \begin{Bmatrix} 1 \\ x \end{Bmatrix} \quad \text{or} \quad \mathbf{a} = \mathbf{W}_x^{\;-1}\mathbf{p}_x$$

$$\phi_i(x) = \{1 \quad x_i\} w_i(x) \begin{Bmatrix} a_0(x) \\ a_1(x) \end{Bmatrix}$$

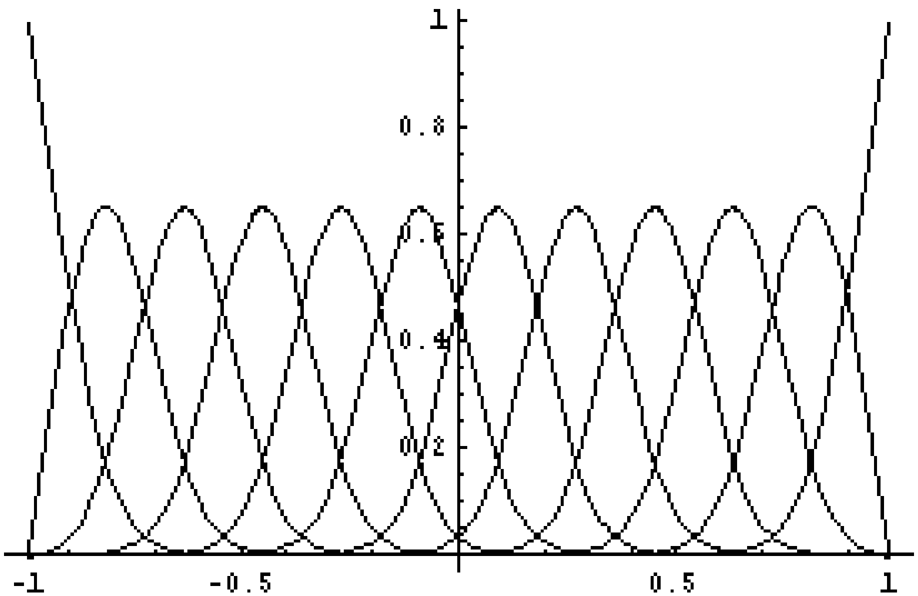
$$= \left(\{1 \quad x_i\} \begin{bmatrix} \sum_{i=1}^{n} w_i(x) & \sum_{i=1}^{n} x_i w_i(x) \\ \sum_{i=1}^{n} x_i w_i(x) & \sum_{i=1}^{n} x_i^{\,2} w_i(x) \end{bmatrix}^{-1} \begin{Bmatrix} 1 \\ x \end{Bmatrix} \right) w_i(x) \quad , \quad i = 1, \ldots, n$$

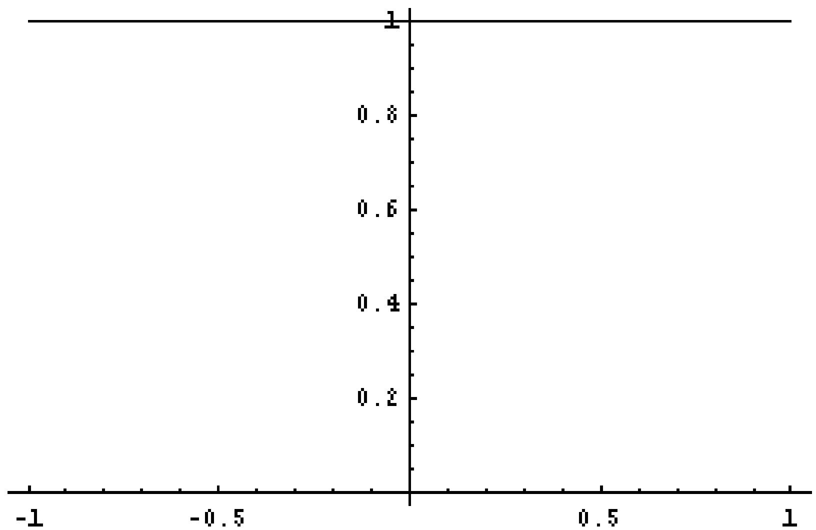
$$\begin{Bmatrix} 1 \\ x \\ \vdots \\ x^k \end{Bmatrix} = \begin{bmatrix} \sum_{j=1}^{n} w_j(x) & \sum_{j=1}^{n} x_j w_j(x) & \dots & \sum_{j=1}^{n} x_j{}^{k} w_j(x) \\ \sum_{j=1}^{n} x_j w_j(x) & \sum_{j=1}^{n} x_j{}^{2} w_j(x) & \dots & \sum_{j=1}^{n} x_j{}^{k+1} w_j(x) \\ \vdots & \vdots & & \vdots \\ \sum_{j=1}^{n} x_j{}^{k} w_j(x) & \sum_{j=1}^{n} x_j{}^{k+1} w_j(x) & \dots & \sum_{j=1}^{n} x_j{}^{2k} w_j(x) \end{bmatrix} \begin{Bmatrix} a_0(x) \\ a_1(x) \\ \vdots \\ a_k(x) \end{Bmatrix}$$

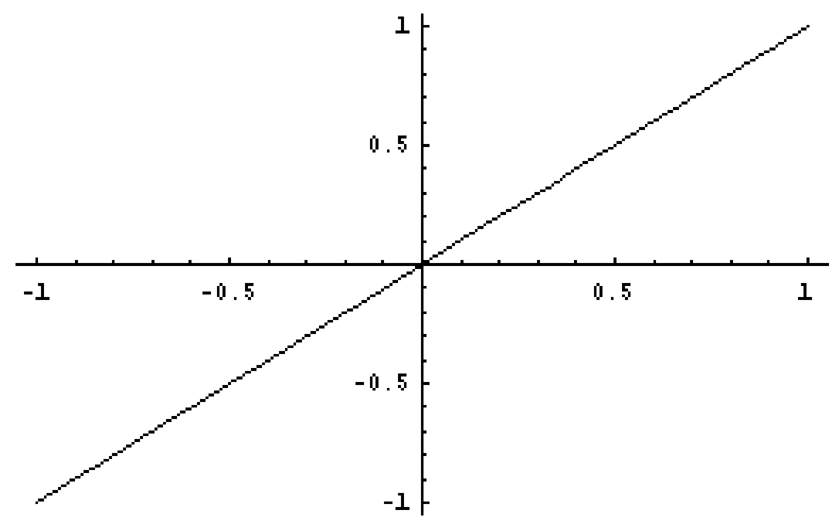
$$\phi_j(x)=\begin{Bmatrix}1 & x_j & \dots & x_j^{\ k}\end{Bmatrix}\begin{bmatrix}\sum_{j=1}^{n} w_j(x) & \sum_{j=1}^{n} x_j w_j(x) & \dots & \sum_{j=1}^{n} x_j^{\ k} w_j(x) \\ \sum_{j=1}^{n} x_j w_j(x) & \sum_{j=1}^{n} x_j^{\ 2} w_j(x) & \dots & \sum_{j=1}^{n} x_j^{\ k+1} w_j(x) \\ \vdots & \vdots & & \vdots \\ \sum_{j=1}^{n} x_j^{\ k} w_j(x) & \sum_{j=1}^{n} x_j^{\ k+1} w_j(x) & \dots & \sum_{j=1}^{n} x_j^{\ 2k} w_j(x)\end{bmatrix}^{-1}\begin{Bmatrix}1 \\ x \\ \vdots \\ x^k\end{Bmatrix} w_j(x)$$

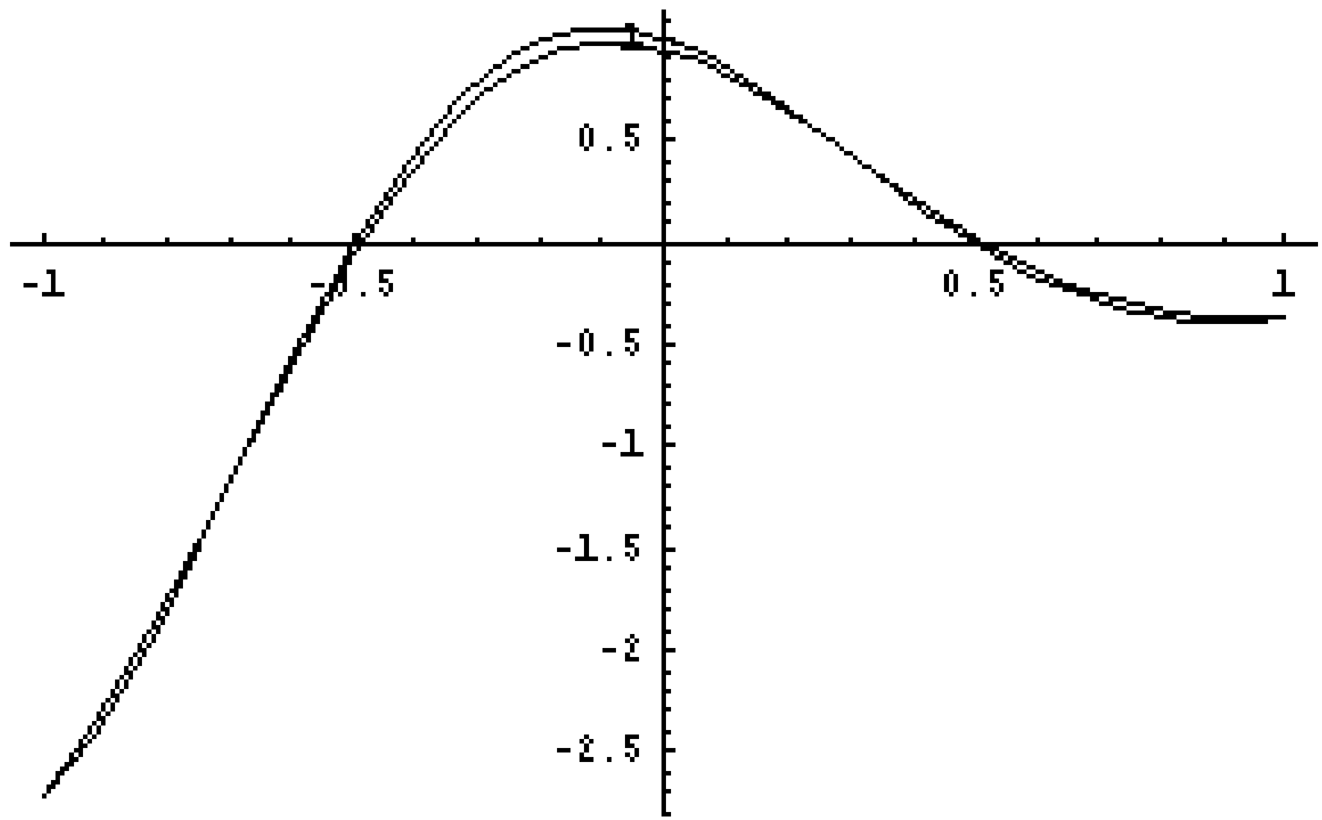
Application

Linear Recovering




1
0.8
0.6
0.4
0.2
-1
-0.5
0.5
1


1
0.5
-1
-0.5
0.5
1
-0.5
-1



-1
-0.5
0.5
1
0.5
-0.5
-1
-1.5
-2
-2.5

Nothing special is required

Simplest is the best !

Put more nodes in the discrete form.
This is the best choice.