Interpolation

Key References:

1. Judd, K. Numerical Methods in Economics, Cambridge: MIT Press. Chapter 6.
2. Press, W. et. al. Numerical Recipes in Fortran 77, Cambridge: Cambridge University Press. Chapter 3 and 5

- We sometimes know the numerical value of a function $f(x)$ at a set of points $\{x_i\}_{i=1}^N$ where $x_1 < x_2 < ... < x_N$, but we don't have an analytic expression for $f(x)$ that lets us calculate its value at any arbitrary point.
- In relation to what we have done already:
  - Basic problem

$$V(a) = \max_{a' \geq \underline{a}} u(y + a - qa') + \beta V(a')$$

  - Discrete grid on a : Let D be the grid points. Problem is

$$\begin{aligned} V_D(a) &= \max_{a'_D \in D} u(y + a - qa'_D) + \beta V_D(a'_D) \\ s.t. a &\in D. \end{aligned}$$

  - Interpolation. Problem is

$$\begin{aligned} V_I(a) &= \max_{a' \geq \underline{a}} u(y + a - qa'_I) + \beta V_I(a'_I) \\ s.t. a &\in D. \end{aligned}$$

    Even though $V_I(a'_I)$ is piecewise linear, $u(y + a - qa'_I) + \beta V_I(a'_I)$ will have the concave shape in the last figure since $u(y + a - qa'_I)$ induces strict concavity.

1 Background

- We need a metric or norm to measure "closeness" when thinking about approximation.

1. L_p norm: $\|g\|_p = \left[\int_a^b |g(x)|^p dx\right]^{1/p}$, $p \in [1, \infty)$.Note that L_2 is mean squared error.
2. sup norm: $\|g\|_\infty = \sup_{x \in [a,b]} |g(x)|$

Theorem (Weierstrass Approximation) If $f \in C([a, b])$,then for all $\varepsilon > 0$,there exists a polynomial $p(x)$ such that $\|f - p\|_\infty < \varepsilon$. Hence there is a sequence of polynomials p_n such that $p_n \to f$ uniformly on $[a, b]$.

2 Piecewise linear interpolation

- Suppose we know $\{y_i = f(x_i)\}_{i=1}^N$ at some discrete set of points $\{x_i\}_{i=1}^N$. For instance, this could be the value function $v^j(k_i)$ at iteration j at capital grid points.

- We construct a function $\ell(x)$ such that $\ell(x_i) = y_i$, $i = 1, ..., N$ and on each interval $[x_i, x_{i+1}]$ for $i = 1, ..., N-1$, the function is linear

$$\ell_{[x_i, x_{i+1}]}(x) = A_i(x) y_i + (1 - A_i(x))\, y_{i+1}, \tag{1}$$

  where the weights A_i measure the relative distance between the point x and the grid points x_i and x_{i+1} given by $A_i(x) = \left(\frac{x_{i+1} - x}{h}\right)$ where $h = x_{i+1} - x_i$.

- Suppose $f(x) = x^2$ and $\{x_i\}_{i=1}^N = \{0, 1, 2\}$. Then $y_1 = 0$, $y_2 = 1$, and $y_3 = 4$.

  - For $x \in [0, 1]$,

$$\ell_{[0,1]}(x) = \left(\frac{1 - x}{1}\right) \cdot 0 + \left(\frac{x - 0}{1}\right) \cdot 1 = x.$$

  - For $x \in [1, 2]$,

$$\begin{aligned} \ell_{[1,2]}(x) &= \left(\frac{2 - x}{1}\right) \cdot 1 + \left(\frac{x - 1}{1}\right) \cdot 4 \\ &= 2 - x + 4x - 4 \\ &= -2 + 3x \end{aligned}$$

  - See Figure entitled Linear Interpolation for $y = x^2$.

3 Cubic Splines

- We construct a function $s(x)$ such that $s(x_i) = y_i$, $i = 1, \ldots, N$ and on each interval $[x_i, x_{i+1}]$ for $i = 1, ..., N-1$, the function is a cubic

$$s_{[x_i, x_{i+1}]}(x) = a_i + b_i x + c_i x^2 + d_i x^3. \tag{2}$$

  So we have $N - 1$ intervals and N data points.

- How do we find those coefficients? Since we have 4 coefficients for each of the $N - 1$ intervals, we have $4(N - 1)$ unknowns a_i, b_i, c_i, d_i for $i = 1, \ldots, N - 1$.

  - The interpolating conditions at the endpoints and continuity at the interior nodes will provide us with $2(N - 1)$ equations

* $y_1 = s_{[x_1,x_2]}(x_1)$ for $i = 1$,
* $s_{[x_{i-1},x_i]}(x_i) = y_i = s_{[x_i,x_{i+1}]}(x_i)$ for $i = 2, ..., N-1$,
* and $y_N = s_{[x_{N-1},x_N]}(x_N)$ for $i = N$

– or:

$$\begin{aligned} y_i &= a_{i-1} + b_{i-1}x_i + c_{i-1}x_i^2 + d_{i-1}x_i^3, \quad i = 2, \ldots, N \qquad (3) \\ y_i &= a_i + b_i x_i + c_i x_i^2 + d_i x_i^3, \quad i = 1, \ldots, N-1. \qquad (4) \end{aligned}$$

– Continuity of the first derivative on the interior nodes gives $(N-2)$ (i.e. the interior nodes is why it is -2) equations $s'_{[x_{i-1},x_i]}(x_i) = s'_{[x_i,x_{i+1}]}(x_i)$ or:

$$b_{i-1} + 2c_{i-1}x_i + 3d_{i-1}x_i^2 = b_i + 2c_i x_i + 3d_i x_i^2, \quad i = 2, \ldots, N-1 \quad (5)$$

– Continuity of the second derivative on the interior nodes (i.e. the interior nodes is why it is -2) gives $(N-2)$ equations $s''_{[x_{i-1},x_i]}(x_i) = s''_{[x_i,x_{i+1}]}(x_i)$ or:

$$2c_{i-1} + 6d_{i-1}x_i = 2c_i + 6d_i x_i, \quad i = 2, \ldots, N-1 \qquad (6)$$

– Since we have $2(N-1)+2(N-2)$ equations but $4(N-1)$ unknowns, we still need two more conditions to be able to solve the system.

* If we knew the true function, then we could use first derivatives $s'_{[x_1,x_2]}(x_1) = f'(x_1)$ and $s'_{[x_{N-1},x_N]}(x_N) = f'(x_N)$. But in general we do not know $f(x)$,so we can't calculate $f'(x)$.
* One thing people do is to assume $s'_{[x_1,x_2]}(x_1) = 0$ and $s'_{[x_{N-1},x_N]}(x_N) = 0$. It is evident the second assumption could be a problem for our $f(x) = x^2$ example.
* Another possibility is something called a hermite spline. In particular, we use the slopes of the secant lines over $[x_1, x_2]$ and $[x_{N-1}, x_N]$;that is, we choose $s(x)$ to make

$$\begin{aligned} s'_{[x_1,x_2]}(x_1) &= \frac{s_{[x_1,x_2]}(x_2) - s_{[x_1,x_2]}(x_1)}{x_2 - x_1} \qquad (7) \\ s'_{[x_{N-1},x_N]}(x_N) &= \frac{s_{[x_{N-1},x_N]}(x_N) - s_{[x_{N-1},x_N]}(x_{N-1})}{x_N - x_{N-1}}. \end{aligned}$$

– Now that we have $4(N-1)$ equations in $4(N-1)$ unknowns, you can just solve a system of equations in matlab.

- Suppose $f(x) = x^2$ and $\{x_i\}_{i=1}^N = \{0, 1, 2\}$ (i.e. $N = 3$). Then $y_1 = 0$, $y_2 = 1$, and $y_3 = 4$. For the cubic spline, we need $4(3-1) = 8$ coefficients $\{a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2\}$.

– From (3) we have

$$\begin{aligned} y_2 &= a_1 + b_1 x_2 + c_1 x_2^2 + d_1 x_2^3 \Longleftrightarrow 1 = a_1 + b_1 + c_1 + d_1. \\ y_3 &= a_2 + b_2 x_3 + c_2 x_3^2 + d_2 x_3^3 \Longleftrightarrow 4 = a_2 + 2b_2 + 4c_2 + 8d_2. \end{aligned}$$

– From (4) we have

$$\begin{aligned} y_1 &= a_1 + b_1 x_1 + c_1 x_1^2 + d_1 x_1^3 \Longleftrightarrow 0 = a_1 \\ y_2 &= a_2 + b_2 x_2 + c_2 x_2^2 + d_2 x_2^3 \Longleftrightarrow 1 = a_2 + b_2 + c_2 + d_2. \end{aligned}$$

– From (5) we have

$$b_1 + 2c_1 x_2 + 3d_1 x_2^2 = b_2 + 2c_2 x_2 + 3d_2 x_2^2 \Longleftrightarrow b_1 + 2c_1 + 3d_1 = b_2 + 2c_2 + 3d_2.$$

– From (6) we have

$$2c_1 + 6d_1 x_2 = 2c_2 + 6d_2 x_2 \Longleftrightarrow 2c_1 + 6d_1 = 2c_2 + 6d_2.$$

– From (7), using $s'_{[x_1,x_2]}(x) = b_1 + 2c_1 x + 3d_1 x^2$, we have

$$\begin{aligned} s'_{[0,1]}(0) &= \frac{y_2 - y_1}{1} \Longleftrightarrow b_1 = 1 \\ \text{and } s'_{[1,2]}(2) &= \frac{y_3 - y_2}{1} \Longleftrightarrow b_2 + 4c_2 + 12d_2 = 3 \end{aligned}$$

– See Figure entitled Cubic Interpolations for $y = x^2$.

- Since linear interpolation is a special case of the Cubic hermite spline, its mean square error must be weakly higher. See Figure entitled Linear vs. Cubic Interpolation.

4 Orthogonal Polynomials

- The Weierstrass theorem is conceptually valuable, but not important from a computational point of view because it uses Bernstein polynomials which converge slowly.[1]

[1] Let $f : [0,1] \to \mathbb{R}$. The *nth* **Bernstein polynomial** for f is defined to be

$$B_n(x; f) = \sum_{k=0}^{n} f\left(\frac{k}{n}\right) \left(\begin{array}{c} n \\ k \end{array} \right) x^k (1-x)^{n-k}$$

where the factorial is given by

$$\left(\begin{array}{c} n \\ k \end{array} \right) = \frac{n!}{k!(n-k)!}.$$

- Since the space of continuous functions is spanned by the monomials x^n, $n = 0, 1, ...$,it is natural to think of the monomials as a basis for the space of continuous functions. [2]

- But these monomials are very nearly linearly dependent, so higher n aren't very efficient at helping (like multicollinearity in regression where we don't get a good fit (i.e. high standard errors)). See figure entitled "Monomials".

- To avoid this problem, people consider an alternative basis that is orthogonal in $C([a, b])$. One can use the Gram-Schmidt algorithm to construct an orthonormal basis.

- A special case of orthogonal polynomials is the Chebyshev polynomials on $x \in [-1, 1]$ which can be expressed by the recursive formula

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), n \geq 1 \tag{8}$$

where $T_0(x) = 1$ and $T_1(x) = x$. Notice

$$\begin{aligned} T_2(x) &= 2x^2 - 1 \\ T_3(x) &= 2x\left(2x^2 - 1\right) - x = 4x^3 - 3x \\ &.... \end{aligned}$$

- The different spanning capabilities of monomials and Chebyshev polynomials on $[-1, 1]$ is evident by comparing the figure entitled "Chebyshev Polynomials" to the monomial one.

- Chebyshev approximation chooses grid points in order to get a good fit. Judd (p. 223) presents the following algorithm (6.2) where you choose m nodes and use them to construct a degree $n < m$ polynomial approximation of $f(x)$ on $[a, b]$ via the following steps:

1. Compute the $m \geq n + 1$ Chebyshev interpolation nodes on $[-1, 1]$:

$$z_k = -\cos\left(\frac{2k-1}{2m} \cdot \pi\right), k = 1, ..., m.$$

2. Adjust the nodes to the $[a, b]$ interval:

$$x_k = (z_k + 1)\left(\frac{b-a}{2}\right) + a, k = 1, ..., m.$$

[2] A basis B of a finite dimensional vector space V is a linearly independent subset of V that spans (or generates) V. A way to think about a basis is to think of the colors Red, Green, Blue as a basis and from there you can create combinations to make other colors. $B = \{v_1, ..., v_n\}$ is a basis if it satisfies the following properties: (1) the linear independence property (if $a_1v_1 + ... + a_nv_n = 0$, then $a_1 = ... = a_n = 0$; and (2) the spanning property (for every $x \in V$, it is possible to choose a unique set of coordinates $a_1, .., a_n$ such that $x = a_1v_1 + ... + a_nv_n$.

3. Evaluate f at the approximation nodes:

$$y_k = f(x_k), k = 1, ..., m.$$

4. Compute Chebyshev coefficients, $c_i, i = 0, ...,$ where T_i is given by (8):

$$c_i = \frac{\sum_{k=1}^{m} y_k T_i(z_k)}{\sum_{k=1}^{m} T_i(z_k)^2}$$

to arrive at the approximation for $f(x), x \in [a, b]$ given by

$$\widehat{f}(x) = \sum_{i=0}^{n} c_i T_i \left(2 \cdot \frac{x-a}{b-a} - 1 \right).$$

5 Bilinear Interpolation

- Now consider two dimensions. In this case, you can think of one of the dimensions being along the grid $\{x_i\}_{i=1}^{N}$ and another being along the grid $\{z_j\}_{j=1}^{M}$, forming an $N \times M$ matrix of grid points (x_i, z_j) in a "Cartesian Mesh".
- For any arbitrary point (x, z), let the functional values be given by $y = f(x, z)$ and let the associated matrix of functional values on the grid be given by the $N \times M$ matrix $y_{i,j} = f(x_i, z_j)$.
- We want to approximate, by interpolation, the function f at some untabulated point (x, z).
- An important concept is that of a grid rectangle (just the 2 dimensional analogue of an interval in the 1 dimensional case) in which the point (x, z) falls. That is, the four tabulated points that surround the desired interior point starting with the lower left and moving counterclockwise numbering the points y^1, y^2, y^3, y^4. See Figure 3.6.1a from the Numerical Recipes Book.[3]
  - If $x \in [x_i, x_{i+1}]$ and $z \in [z_j, z_{j+1}]$ implicitly defines i and j, then
  - $y^1 = y_{i,j}$
  - $y^2 = y_{i+1,j}$
  - $y^3 = y_{i+1,j+1}$
  - $y^4 = y_{i,j+1}$
- Then the equation of the surface over the grid rectangle using bilinear interpolation is given by

$$b_{y^1,y^2,y^3,y^4}(x, z) = (1-t)\cdot(1-u)\cdot y^1 + t\cdot(1-u)\cdot y^2 + t\cdot u\cdot y^3 + (1-t)\cdot u\cdot y^4 \quad (9)$$

[3] The table in Figure 3.6.1b is for bicubic interpolation.

where

$$\begin{aligned} t &\equiv \frac{x - x_i}{x_{i+1} - x_i} \\ u &\equiv \frac{z - z_j}{z_{j+1} - z_j} \end{aligned}$$

so that t and u lie between 0 and 1. See Figure entitled Bilinear Interpolation.

- One way to think of this is that for a given $z \in \{z_j, z_{j+1}\}$, b is a linear interpolation of the type in section 2.
  - Specifically, suppose $z = z_j$, then $u = 0$,

$$b_{y^1,y^2,y^3,y^4}(x, z_j) = \left[1 - \left(\frac{x - x_i}{x_{i+1} - x_i}\right)\right] \cdot y^1 + \left(\frac{x - x_i}{x_{i+1} - x_i}\right) \cdot y^2 \quad (10)$$

    where

$$\begin{aligned} y^1 &= f(x_i, z_j) \\ y^2 &= f(x_{i+1}, z_j). \end{aligned}$$

    Notice that (10) is just like (1) where the first term is like $1 - A$ and the second like A.
  - We can always rearrange (10) to yield

$$\begin{aligned} b_{y^1,y^2,y^3,y^4}(x, z_j) &= \left[y^1 - \left(\frac{x_i}{x_{i+1} - x_i}\right) \cdot \left[y^2 - y^1\right]\right] + \left(\frac{y^2 - y^1}{x_{i+1} - x_i}\right) \cdot x \\ &= b + mx \end{aligned}$$

6 Application to Dynamic Programming

6.1 One dimensional Grid algorithm

- Suppose we have a set of grid points $X = \{x_i\}_{i=1}^N$ where $x_1 < ... < x_i < ... < x_N$.
- Suppose we have an $N \times 1$ vector $v^j(x_i)$ at iteration j (this is just like the $\{y_i\}_{i=1}^N$ from before).
- For each x_i, we are trying to solve for v^{j+1} given by

$$v^{j+1}(x_i) = \max_{x' \in X} u(x_i - x') + \beta v^j(x'). \quad (11)$$

- There are obvious errors induced by restricting $x' \in X$. In particular, in the last figure, from x_i the agent chooses $x' = x_k$ since it yields higher utility than $x' = x_{k+1}$ but the agent would rather choose $x' \in (x_k, x_{k+1})$, which motivates why we might choose to do linear interpolation.

- The first algorithm we used to solve this problem (not necessarily the fastest way, but how you did it in the first problem set) is:
  - While $\sup_x |v^j(x) - v^{j-1}(x)| > \varepsilon$, do
    - for $i = 1, ..., N$,
    - evaluate the $N \times 1$ vector (just the value of the rhs of (11) evaluated at each of the x' on the grid):

$$V_i^{j+1} = \begin{bmatrix} u(x_i - x_1) + \beta v^j(x_1) \\ \dots \\ u(x_i - x_k) + \beta v^j(x_k) \\ \\ u(x_i - x_N) + \beta v^j(x_N) \end{bmatrix}$$

      To be clear, this is just a vector of numbers for each grid point x_i.
    - use a min function to choose the row (i.e. number) of the above vector which minimizes $-V_i^{j+1}$ for the given grid point x_i and call that $v^{j+1}(x_i)$. We do this for each x_i (which is why we say for $i = 1, ..., N$).

6.2 One dimensional linear interpolation algorithm

- Suppose we have a set of grid points $X = \{x_i\}_{i=1}^N$ where $x_1 < ... < x_i < ... < x_N$.
- Suppose we have a piecewise linear function $v^j(x)$ at iteration j. Specifically, the function is defined by $N - 1$ intervals for which we have an intercept and a slope:

interval	value $v^j(x)$
$x \in [x_1, x_2]$	$a_{1,2}^j + b_{1,2}^j x$
...	
$x \in [x_n, x_{n+1}]$	$a_{n,n+1}^j + b_{n,n+1}^j x$
...	
$x \in [x_{N-1}, x_N]$	$a_{N-1,N}^j + b_{N-1,N}^j x$

- For each x_i, we are trying to solve for v^{j+1} given by

$$v^{j+1}(x_i) = \max_{x' \in [x_1, x_N]} u(x_i - x') + \beta v^j(x'). \tag{12}$$

  The fact that $x' \in [x_1, x_N]$ rather than $x' \in X$ is the gain to doing interpolation.
- An algorithm to solve this problem (not necessarily the fastest way) is:
  - While $\sup_x |v^j(x) - v^{j-1}(x)| > \varepsilon$, do

* for $i = 1, ..., N$,
  - use fminsearch to minimize $-V_i^{j+1}$ where:

$$V_i^{j+1} = \begin{bmatrix} u(x_i - x') + \beta[a_{1,2}^j + b_{1,2}^j x'] \text{ for } x' \in [x_1, x_2] \\ \dots \\ u(x_i - x') + \beta[a_{k,k+1}^j + b_{k,k+1}^j x'] \text{ for } x' \in [x_k, x_{k+1}] \\ \dots \\ u(x_i - x') + \beta[a_{N-1,N}^j + b_{N-1,N}^j x'] \text{ for } x' \in [x_{N-1}, x_N] \end{bmatrix}$$

    call the minimized value $v^{j+1}(x_i)$. To be clear, this is just a vector of functions to be minimized by choice of $x' \in [x_1, x_N]$.
  - with the new $v^{j+1}(x_i)$, recompute $\{a_{i,i+1}^{j+1}, b_{i,i+1}^{j+1}\}_{i=1}^{N-1}$.

- How do we use the matlab function fminsearch? fminsearch(W,initial x_0', [],parameters(i.e. $\{a_{i,i+1}^j, b_{i,i+1}^j\}_{i=1}^{N-1}$)) where you create a matlab function in order to define the function $W = -V_i^{j+1}(x')$.

- For each x_i, we can create the following function $Wfun$ of the choice variable x' (in matlab) as:
  - function $W = Wfun(x')$
  - if $x' \in [x_1, x_2]$, then $W = -\left\{u(x_i - x') + \beta[a_{1,2}^j + b_{1,2}^j x']\right\}$
  - else if ....$x' \in [x_k, x_{k+1}]$, then $W = -\left\{u(x_i - x') + \beta[a_{k,k+1}^j + b_{k,k+1}^j x']\right\}$
  - else if $x' \in [x_{N-1}, x_N]$, then $W = -\left\{u(x_i - x') + \beta[a_{N-1,N}^j + b_{N-1,N}^j x']\right\}$
  - else $W = 1e + 25$

- The fminsearch routine will vary $x' \in [x_1, x_N]$ among the different intervals $\{[x_1, x_2], ..., [x_{N-1}, x_N]\}$ until it finds the value of x' which minimizes $Wfun(x')$.

6.2.1 Searching an Ordered Table

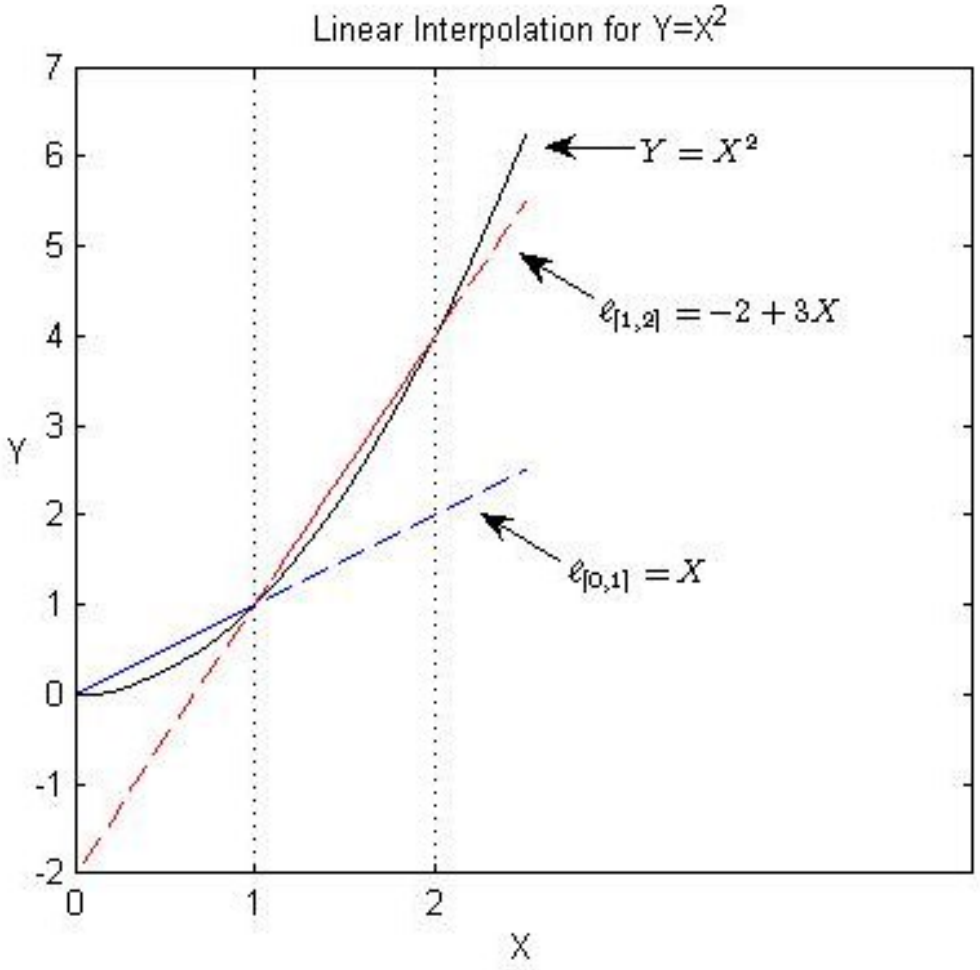
- The problem with the previous procedure is that it is slow - fminsearch will evaluate this function many times and conditional loops in matlab are slow.

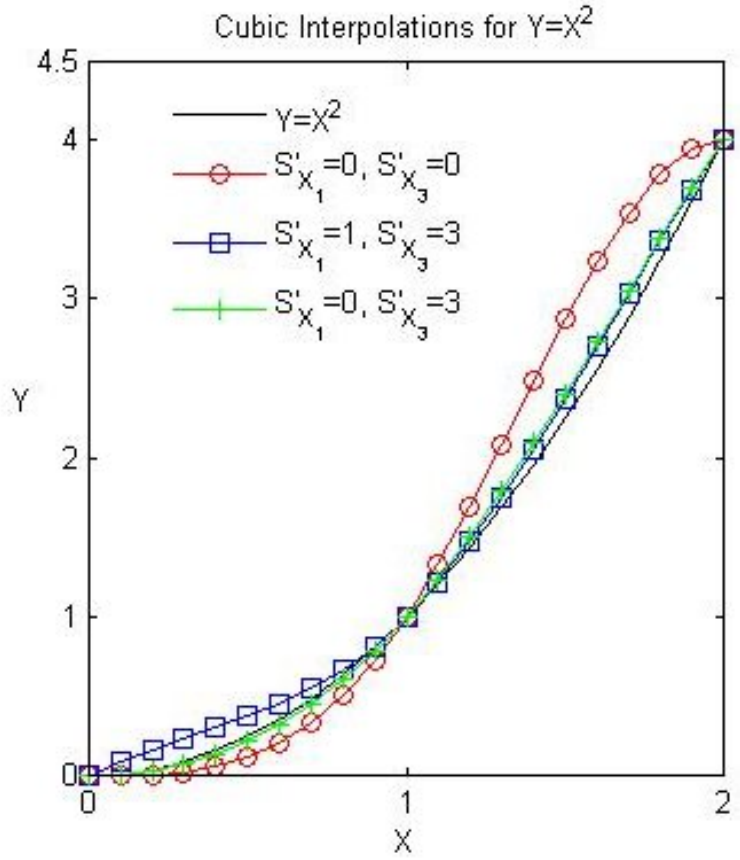
- If you're lazy, just use matlab's function $interp1$.

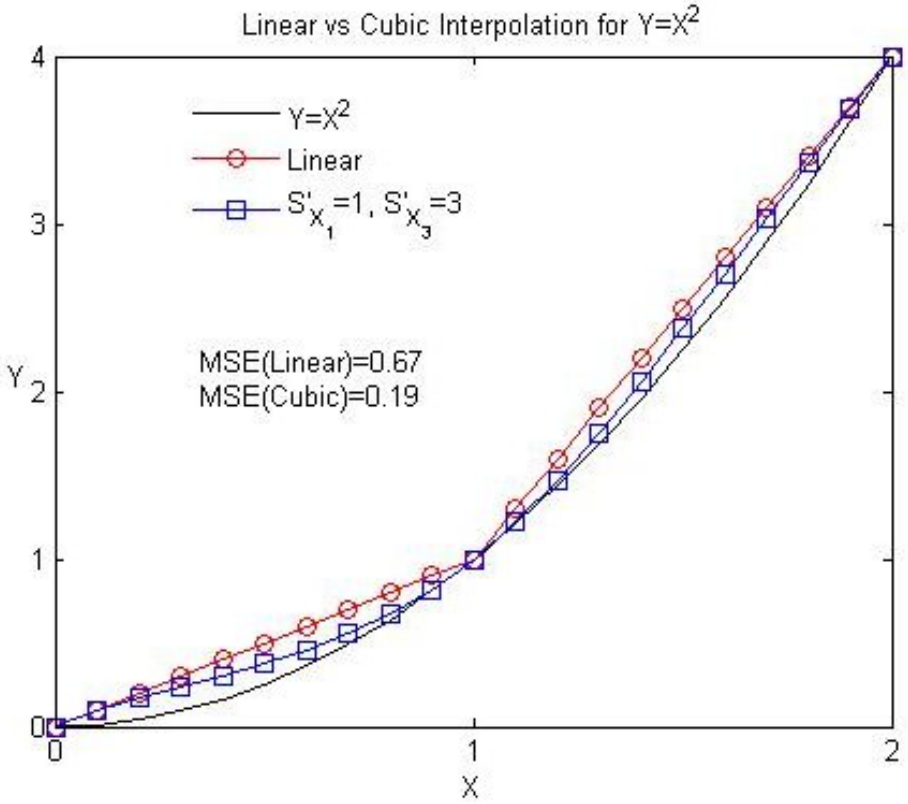
- The suggestion in Press, section 3.4 is to search an ordered table, which basically applies bisection and alters the function $Wfun$. Let's call the new function, $Wfunquick$.
  - function $W = Wfunquick(x)$
  - find which interval $[x_k, x_{k+1}]$ to which x belongs using bisection

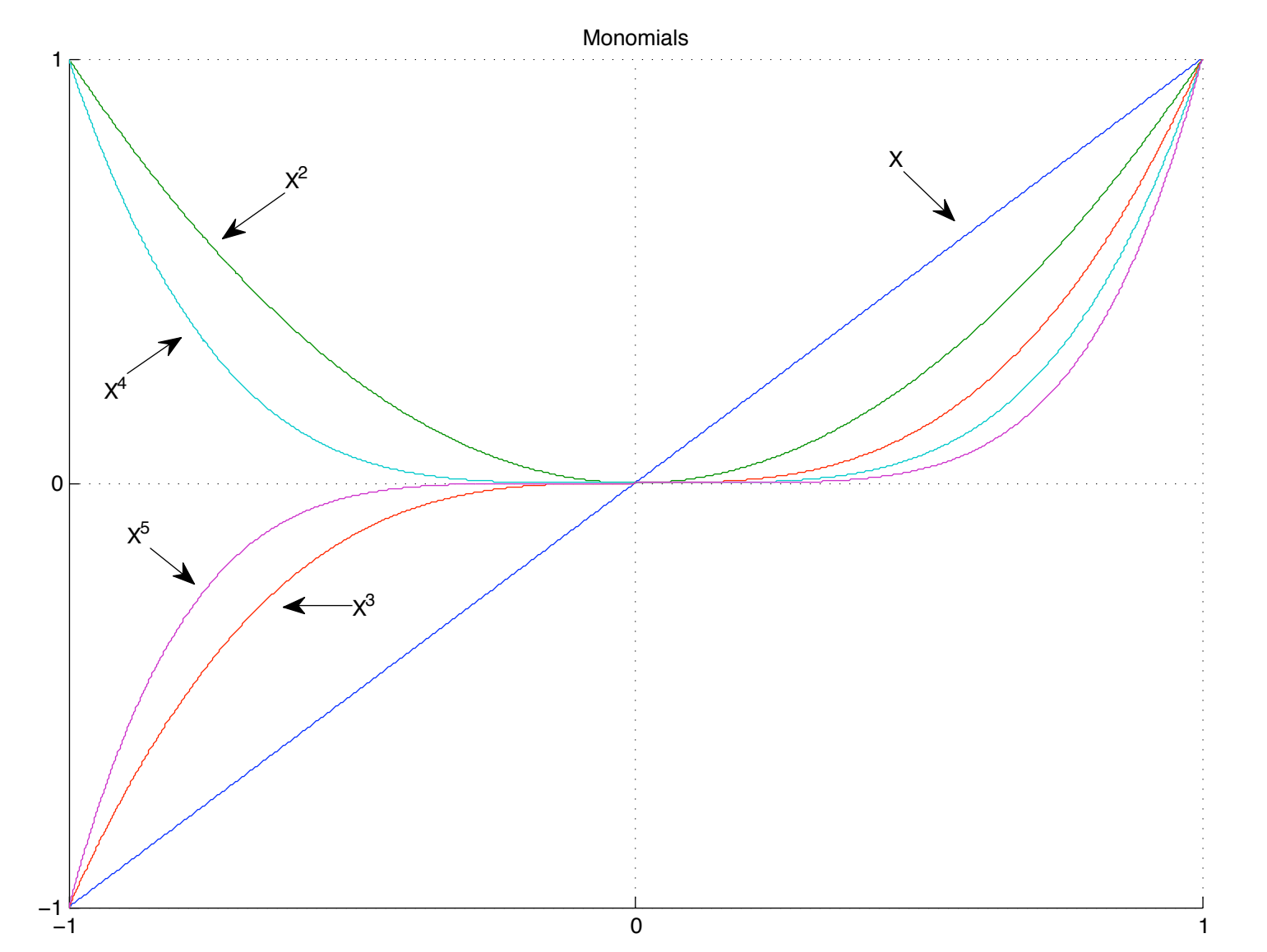
* let $x \in I^0 = [x^0_{\min}, x^0_{\max}]$ where $x^0_{\min} = x_1$ and $x^0_{\max} = x_N$
* if $x <= (x^0_{\min} + x^0_{\max})/2$ (or the closest grid point) then take $I^1 = [x^1_{\min}, x^1_{\max}]$ where $x^1_{\min} = x^0_{\min}$ and $x^1_{\max} = (x^0_{\min} + x^0_{\max})/2$ else $I^1 = [x^1_{\min}, x^1_{\max}]$ where $x^1_{\min} = (x^0_{\min} + x^0_{\max})/2$ and $x^1_{\max} = x^0_{\max}$.
* for $x \in I^{j-1}$, if $x <= (x^{j-1}_{\min} + x^{j-1}_{\max})/2$ (or the closest grid point) then take $I^j = [x^j_{\min}, x^j_{\max}]$ where $x^j_{\min} = x^{j-1}_{\min}$ and $x^j_{\max} = (x^{j-1}_{\min} + x^{j-1}_{\max})/2$ else $I^j = [x^j_{\min}, x^j_{\max}]$ where $x^j_{\min} = (x^{j-1}_{\min} + x^{j-1}_{\max})/2$ and $x^j_{\max} = x^{j-1}_{\max}$.
* do until $x_{\min}$ and $x_{\max}$ are adjacent (i.e. are the endpoints of one of the grid intervals).
* using this interval $[x_k, x_{k+1}]$, compute the A_k and B_k in (1) and define the function $W = -\left\{u(x_i - x') + \beta[A_k v^j(x_k) + B_k v^j(x_{k+1})]\right\}$.

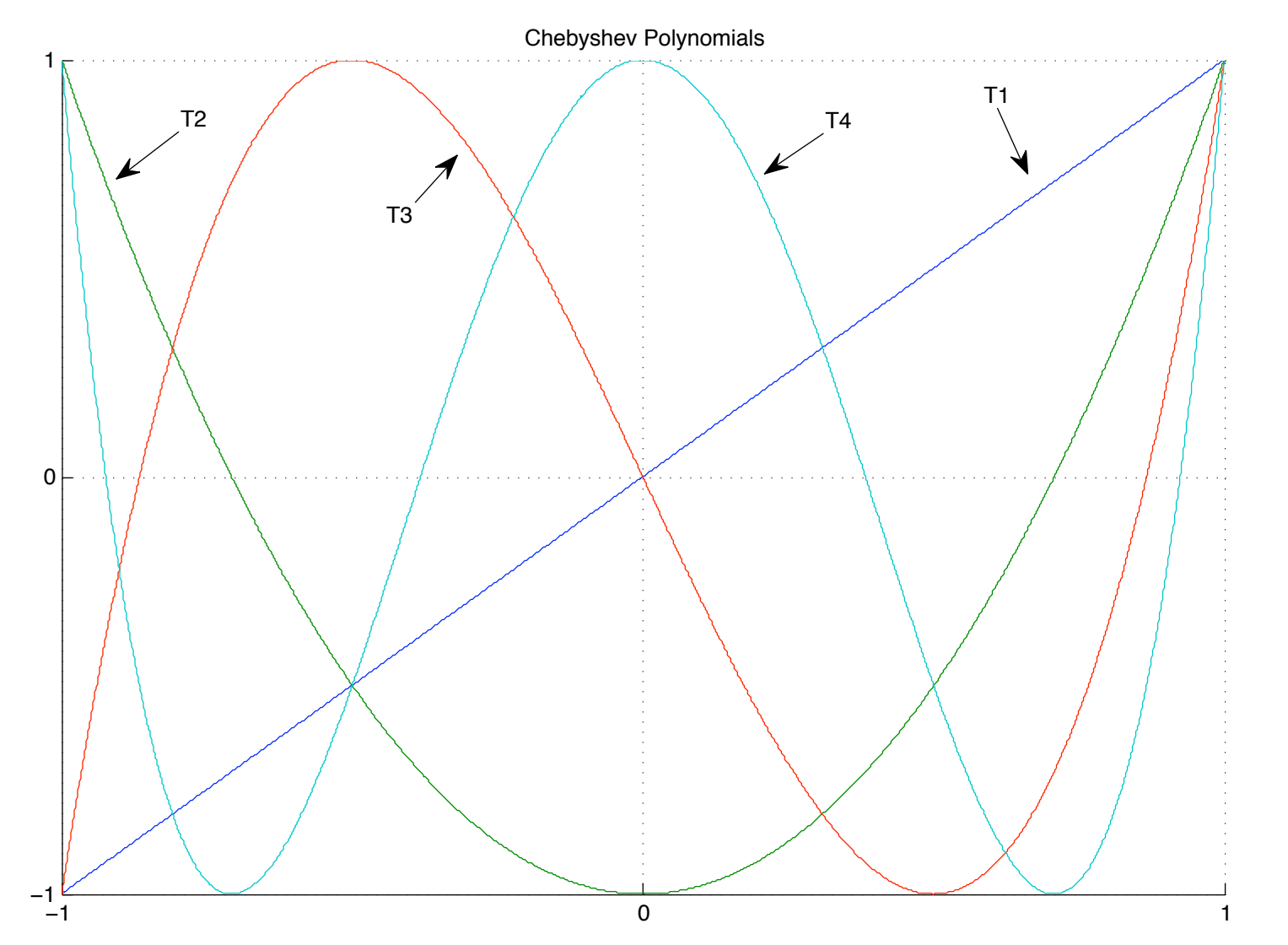
- This helps since the call by fminsearch will take much less time to evaluate whether a particular x' raises or lowers welfare, rather than computing all the intervals just to check x' in one interval (that is, bisection that finds the interval that x' lies in quickly).


Linear Interpolation for Y=X2
7
6
5
4
3
2
1
0
-1
-2
Y
0
1
2
X
$Y = X^2$
$\ell_{[1,2]} = -2 + 3X$
$\ell_{[0,1]} = X$


Cubic Interpolations for Y=X2
Y=X2
S'X1=0, S'X3=0
S'X1=1, S'X3=3
S'X1=0, S'X3=3
Y
X
0
1
2
3
4
4.5


Linear vs Cubic Interpolation for Y=X2
Y=X2
Linear
S'X1=1, S'X3=3
MSE(Linear)=0.67
MSE(Cubic)=0.19
Y
X
0
1
2
3
4


Monomials
1
0
−1
−1
0
1
x
x2
x3
x4
x5


Chebyshev Polynomials
T1
T2
T3
T4
1
0
−1
−1
0
1

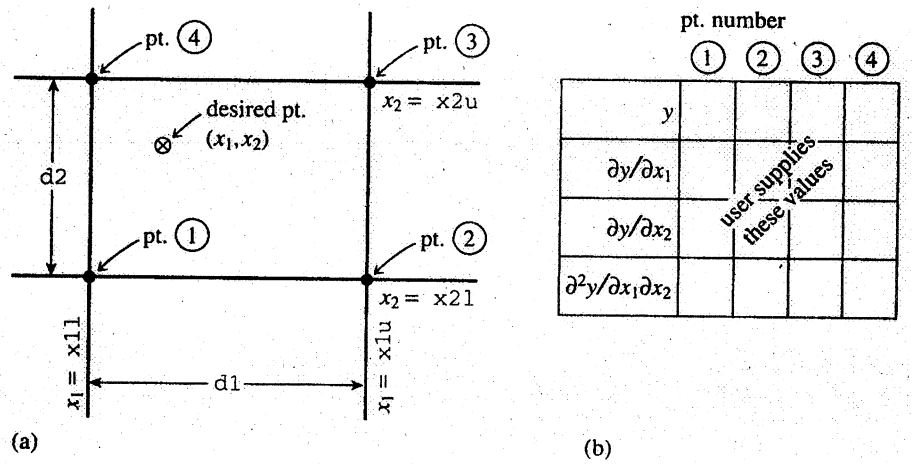
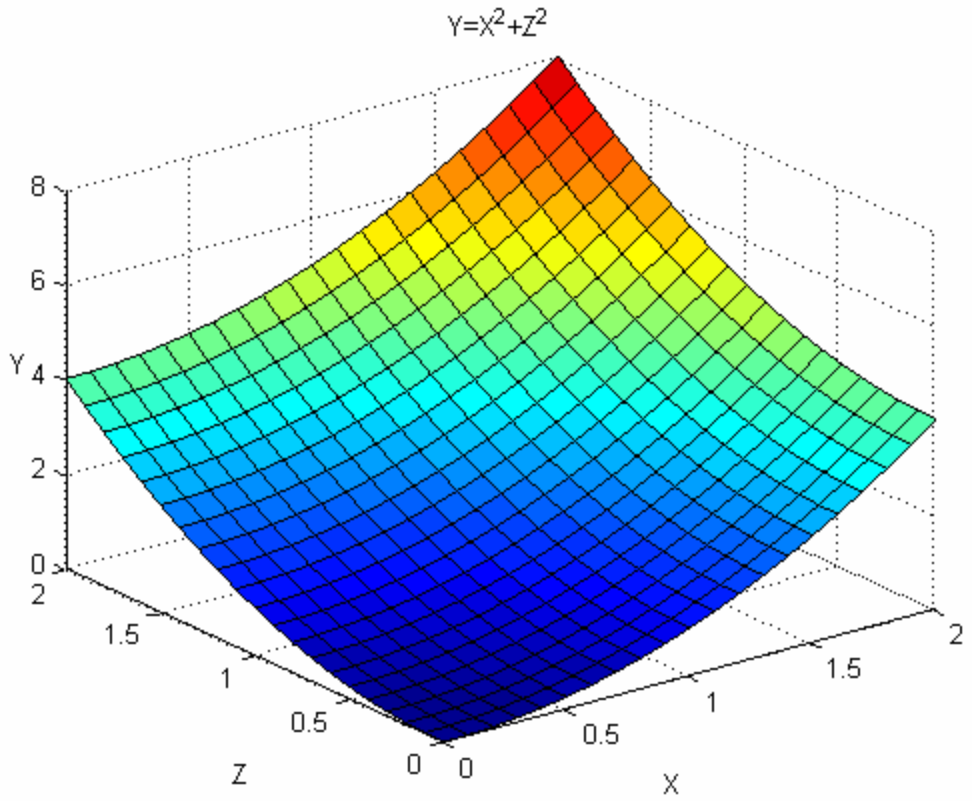
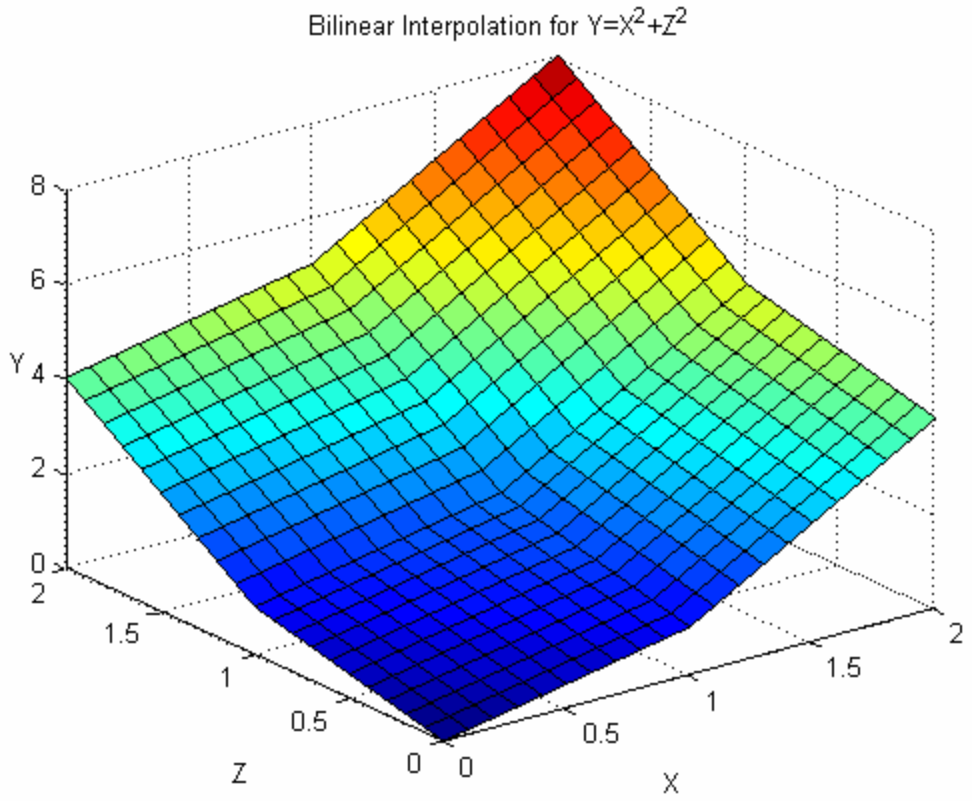
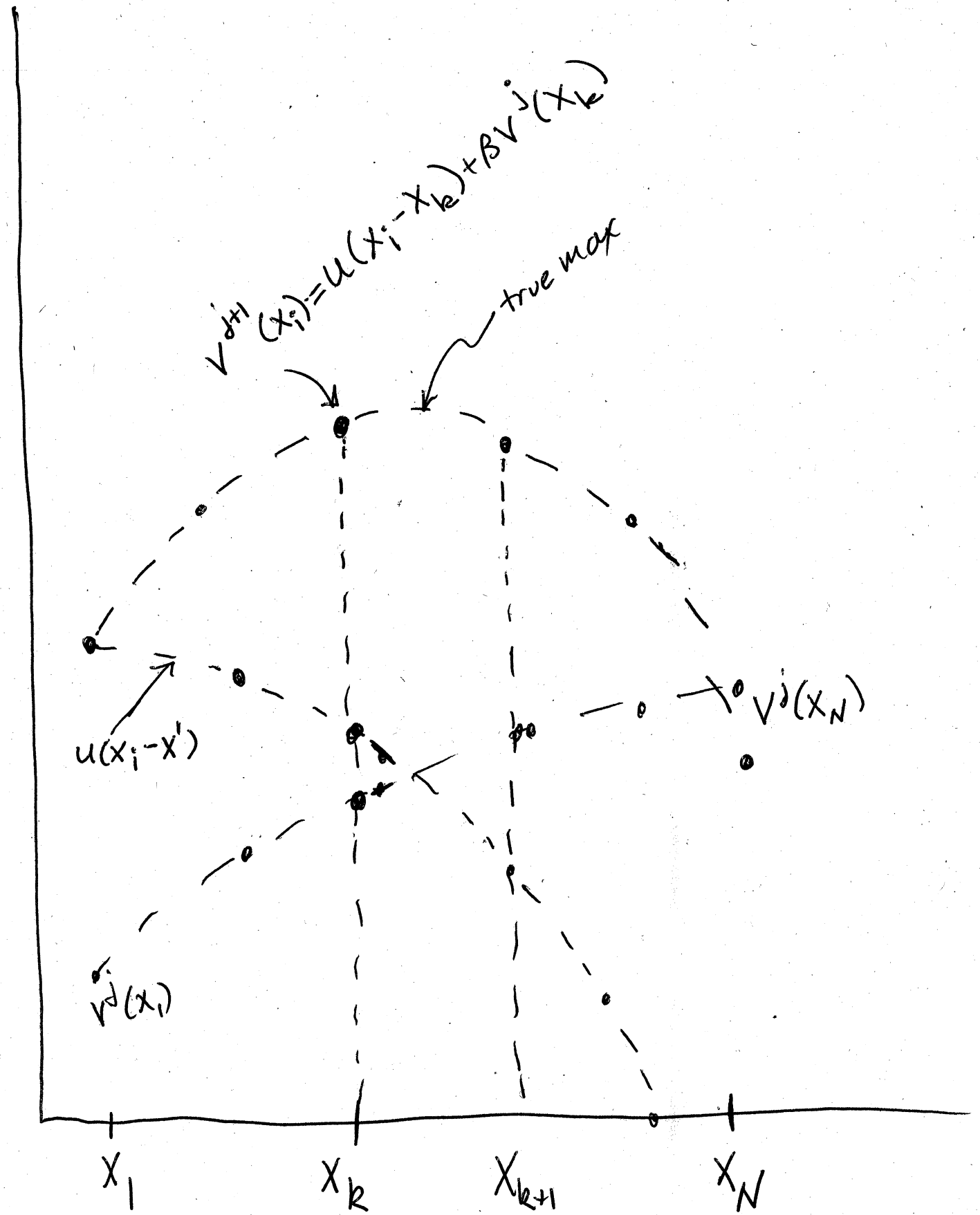


Figure 3.6.1. (a) Labeling of points used in the two-dimensional interpolation routines `bcuint` and `bcucof`. (b) For each of the four points in (a), the user supplies one function value, two first derivatives, and one cross-derivative, a total of 16 numbers.


Y=X^2+Z^2
8
6
Y 4
2
0
2
1.5
1
0.5
0
Z
0
0.5
1
1.5
2
X


Bilinear Interpolation for Y=X^2+Z^2
8
6
Y 4
2
0
2
1.5
1
0.5
0
Z
0
0.5
1
1.5
2
X


$V^{j+1}(x_i) = u(x_i - x_k) + \beta V^j(x_k)$
true max
$u(x_i - x')$
$V^j(x_N)$
$V^j(x_1)$
x_1
x_k
x_{k+1}
x_N