Bar-Ilan University
Estimation of DP Models
March, 2017

Moshe Buchinsky
Department of Economics
UCLA

Lecture Note 5

Appoximation and Interpolation Methods

Part II

Shape preserving approximations

In this section, we will see an approximation method that preserves the shape of the function we want to approximate.

This method was proposed by Schumaker [1983] and essentially amounts to exploit some information on both the level and the slope of the function to be approximated to build a smooth approximation.

We will deal with two situations.

The first one – Hermite interpolation – assumes that we have information on both the level and the slope of the function to approximate.

The second one – that uses Lagrange data – assumes that no information on the slope of the function is available.

Both method was originally developed using quadratic splines.

Hermite interpolation

This method assumes that we have information on both the level and the slope of the function to be approximated.

Assume we want to approximate the function F on the interval $[x_i, x_2]$ and we know $y_i = F(x_i)$ and $z_i = F'(x_i), \quad i = 1, 2.$

Schumaker proposes to build a quadratic function $S(x)$ on $[x_1, x_2]$ that satisfies

$$S(x_i) = y_i \text{ and } S'(x_i) = z_i \text{ for } i = 1, 2.$$

Schumaker establishes first that

Lemma 1 *If*

$$\frac{z_1 + z_2}{2} = \frac{y_2 - y_1}{x_2 - x_1}$$

then the quadratic form

$$S(x) = y_1 + z_1(x - x_1) + \frac{z_2 - z_1}{2(x_2 - x_1)}(x - x_1)^2$$

satisfies $S(x_i) = y_i$ *and* $S'(x_i) = z_i$ *for* $i = 1, 2$.

The construction of this function is rather appealing. If z_1 and z_2 have the same sign then $S'(x)$ has the same sign as z_1 and $z_2 2$ over $[x_1, x_2]$:

$$S'(x) = z_1 + \frac{(z_2 - z_1)}{(x_2 - x_1)}(x - x_1).$$

Hence, if F is monotonically increasing (decreasing) on the interval $[x_1, x_2]$, so is $S(x)$. Further, $z_1 > z_2 \quad (z_1 < z_2)$ indicates concavity (convexity), which $S(x)$ satisfies as $S''(x) = (z_2 - z_1)/(x_2 - x_1) < 0 \quad (> 0)$.

However, the conditions stated by this lemma are extremely stringent and do not usually apply, such that we have to adapt the procedure.

This may be done by adding a node between x_1 and x_2 and construct another spline that satisfies the lemma.

Lemma 2 *For every $x^* \in (x_1, x_2)$ there exist a unique quadratic spline that solves*

$$S(x_i) = y_i \quad \textit{and} \quad S'(x_i) = z_i \quad \textit{for} \quad i = 1, 2$$

with a node at x^*. This spline is given by

$$S(x) = \begin{cases} \alpha_{01} + \alpha_{11}(x - x_1) + \alpha_{21} + \alpha_{11}(x - x_1)^2 & \textit{for } x \in [x_1, x^*] \\ \alpha_{02} + \alpha_{12}(x - x^*) + \alpha_{21} + \alpha_{22}(x - x^*)^2 & \textit{for } x \in [x^*, x_x] \end{cases}$$

where

$$\begin{array}{lll} \alpha_{01} = y_1 & \alpha_{11} = z_1 & \alpha_{21} = \frac{\overline{z} - z_1}{2(x^* - x_1)} \\ \alpha_{02} = y_1 + \frac{\overline{z} + z_1}{2}(x^* - x_1) & \alpha_{12} = \overline{z} & \alpha_{22} = \frac{z_2 - \overline{z}}{2(x_x - x^*)} \end{array}$$

where $\overline{z} = \frac{2(y_2 - y_1) - (z_1(x^* - x_1) + z_2(x_x - x^*))}{x_2 - x_1}$.

If the latter lemma fully characterized the quadratic spline, it gives no information on x^* which therefore remains to be selected.

x^* will be set such that the spline matches the desired shape properties.

First note that if z_1 and z_2 are both positive (negative), then $S(x)$ is monotone if and only if $z_1 \overline{z} \geq 0 \quad (\leq 0)$which is actually equivalent to

$$2(y - y_1) \gtreqless (x^* - x_1) z_1 + (x_2 - x^*) z_2 \quad \text{if } z_1, z_2 \gtreqless 0.$$

This essentially deals with the monotonicity problem, and we now have to tackle the question of curvature. To do so, we compute the slope of the secant line between x_1 and x_2

$$\Delta = \frac{y_2 - y_1}{x_2 - x_1}.$$

Then, if $(z_2 - \Delta)(z_1 - \Delta) \geq 0$, this indicates the presence of an inflexion point in the interval $[x_1, x_2]$ such that the interpolant cannot be neither convex nor concave.

Conversely, if $|z_2 - \Delta| < |z_1 - \Delta|$ and x^* satisfies

$$x_1 < x^* \leq \overline{x} \equiv x_1 + \frac{2(x_2 - x_1)(z_2 - \Delta)}{(z_2 - z_1)}$$

then $S(x)$, as described in the latter lemma, is convex (concave) if $z_1 < z_2 \quad (z_1 > z_2)$

Further, if $z_1 z_2 > 0$ it is also monotone.

If, on the contrary, $|z_2 - \Delta| > |z_1 - \Delta|$ and x^* satisfies

$$\underline{x} \equiv x_2 + \frac{2\left(x_2 - x_1\right)\left(z_1 - \Delta\right)}{\left(z_2 - z_1\right)} \leq x^* < x_2$$

then $S\left(x\right)$, as described in the latter lemma, is convex (concave) if $z_1 < z_2 \quad \left(z_1 > z_2\right)$.

This therefore endows us with a range of values for x^* that will insure that shape properties will be preserved.

1. Check if lemma 1 is satisfied. If so set $x^* = x_2$ and set $S(x)$ as in lemma 2. Then STOP else go to 2.

2. Compute $\Delta = y_2 - y_1/x_2 - x_1$

3. if $(z_1 - \Delta)(z_2 - \Delta) \geq 0$ set $x^* = (x_1 + x_2)/2$ and STOP else go to 4.

4. if $|z_1 - \Delta| < |z_2 - \Delta| \geq 0$ set $x^* = (x_1 + \overline{x})/2$ and STOP else go to 5.

5. if $|z_1 - \Delta| \geq |z_2 - \Delta| \geq 0$ set $x^* = (x_2 + \underline{x})/2$ and STOP.

We have then in hand a value for x^* for $[x_1, x_2]$.

We then apply it to each sub-interval to get $x_i^* \in [x_i, x_{i+1}]$ and then solve the general interpolation problem as explained in lemma 2.

Note here that everything assumes that with have Hermite data in hand – i.e., $\{x_i, y_i, \dot{z}_i : i = 0, .., n\}$. However, the knowledge of the slope is usually not the rule and we therefore have to adapt the algorithm to such situations.

Unknown slope: back to Lagrange interpolation

Assume now that we do not have any data for the slope of the function, that is we are only endowed with Lagrange data $\{x_i, y_i : i = 0, .., n\}$.

In such a case, we just have to add the needed information – an estimate of the slope of the function– and proceed exactly as in Hermite interpolation.

Schumaker proposes the following procedure to get $\{\dot{z}_i : i = 0, .., n\}$. Compute

$$L_i = \left[(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2\right]^{1/2}$$

and

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

for $i = 1, ..., n - 1$.

Then $\dot{z}_i : i = 0, ..., n$ can be recovered as

$$z_i = \begin{cases} \frac{L_{i-1}\Delta_{i-1} + L_i\Delta_i}{L_{i-1}+L_i} & \text{if } \Delta_{i-1}\Delta_i > 0 \\ 0 & \Delta_{i-1}\Delta_i \leq 0 \end{cases} \qquad i = 2, ..., n-1$$

and

$$z_1 = -\frac{3\Delta_1 - z_2}{2} \text{ and } z_n = -\frac{3\Delta n - 1 - s_{n-1}}{2}.$$

Then, we just apply exactly the same procedure as described in the previous section.

Up to now, all methods we have been studying are unidimensional whereas most of the model we deal with in economics involve more than 1 variable. We therefore need to extend the analysis to higher dimensional problems.

Multidimensional approximations

Computing a multidimensional approximation to a function may be quite cumbersome and even impossible in some cases.

To understand the problem, let us restate an example provided by Judd [1998].

Consider we have data points $\{P_1, P_2, P_3, P_4\} = \{(1,0), (-1,0), (0,1), (0,-1)\}$ in $\mathbb{R}^2$ and the corresponding data $z_i = F(P_i), \quad i = 1, ..., 4$.

Assume now that we want to construct the approximation of function F using a linear combination of $\{1, x, y, xy\}$ defined as

$$G(x,y) = a + bx + cy + dxy$$

such that $G(x_i, y_i) = z_i$.

Finding a, b, c, d amounts to solve the linear system

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix}$$

which is not feasible as the matrix is not full rank.

This example reveals two potential problems:

1. Approximation in higher dimensional systems involves cross-product and therefore poses the problem of the selection of polynomial basis to be used for approximation,

2. More important is the selection of the grid of nodes used to evaluate the function to compute the approximation.

We now investigate these issues, by first considering the simplest way to attack the question – namely considering *tensor product bases* – and then moving to a second way of dealing with this problem – considering *complete polynomials*. In each case, we explain how Chebychev approximations can be obtained.

Tensor product bases

The idea here is to use the tensor product of univariate functions to form a basis of multivariate functions. In order to better understand this point, let us consider that we want to approximate a function $F : \mathbb{R}^2 \longrightarrow \mathbb{R}$ using simple univariate monomials up to order 2: $\mathcal{X} = \left\{1, x, x^2\right\}$ and $\mathcal{Y} = \left\{1, y, y^2\right\}$. The tensor product basis is given by

$$\left\{1, x, y, xy, x^2, y^2, x^2y, xy^2, x^2y^2\right\}$$

i.e., all possible 2-terms products of elements belonging to $\mathcal{X}$ and $\mathcal{Y}$.

We are now in position to define the n-*fold tensor product* basis for functions of n variables $\{x_1, ..., x_i, ..., x_n\}$.

Definition 1 *Given a basis for n functions of the single variable x_i : $\mathcal{P}_i = \left\{p_i^k(x_i)\right\}_{k=0}^{k_i}$ then the tensor product basis is given by*

$$\mathcal{B} = \left\{ \prod_{k_1=0}^{k_1} \ldots \prod_{k_n=0}^{kn} .p_1^{k_1}(x_1) \ldots p_n^{k_n}(x_n) \right\}.$$

An important problem with this type of tensor product basis is their size. For example, considering a m-dimensional space with polynomials of order n, we already get $(n+1)^m$ terms!

This exponential growth in the number of terms makes it particularly costly to use this type of basis, as soon as the number of terms or the number of nodes is high.

Nevertheless, it will often be satisfactory or sufficient for low enough polynomials (in practice $n = 2$!) Therefore, one often relies on less computationally costly basis.

Complete polynomials

As aforementioned, tensor product bases grow exponentially as the dimension of the problem increases, complete polynomials have the great advantage of growing only polynomially as the dimension increases.

From an intuitive point of view, complete polynomials bases take products of order lower than a priori given κ into account, ignoring higher terms of higher degrees.

Definition 2 *For $\kappa \in \mathbb{N}$ given, the complete set of polynomials of total degree κ in n variables is given by*

$$\mathcal{B}^c = \left\{ x_1^{k_1} \times ... \times x_n^{k_n} : k_1, ..., k_n \geq 0, \sum_{i=1}^{n} k_i \leq \kappa \right\}.$$

To see this more clearly, let us consider the example developed in the previous section $\left(\mathcal{X} = \left\{1, x, x^2\right\} \text{ and } \mathcal{Y} = \left\{1, y, y^2\right\}\right)$ and let us assume that $\kappa = 2$.

In this case, we end up with a complete polynomials basis of the type

$$\mathcal{B}^c = \left\{1, x, y, x^2, y^2, xy\right\} = \mathcal{B} \diagdown \left\{xy^2, x^2y, x^2y^2\right\}.$$

Note that we have actually already encountered this type of basis, as this is typically what is done by Taylor's theorem for many dimensions

$$\begin{aligned} F(x) &\simeq F(x^*) + \sum_{i=1}^{n} \frac{\partial F}{\partial x_i}(x^*)(x_i - x_{i^*}) \\ &\vdots \\ &+ \frac{1}{k!} \sum_{i_i=1}^{n} \dots \sum_{i_k=1}^{n} \frac{\partial F}{\partial x_{i_1} \dots \partial x_{i_k}}(x^*)\left(x_{i_1} - x_{i_1}^*\right)\left(x_{i_k} - x_{i_k}^*\right) \end{aligned}$$

For instance, considering the Taylor expansion to the 2-dimensional function $F(x,y)$ around (x^*,y^*) we get

$$\begin{aligned} F(x,y) \simeq\ & F(x^*,y^*) + F_x(x^*,y^*)(x-x^*) + F_y(x^*,y^*)(y-y^*) \\ & + \frac{1}{2}\{F_{xx}(x^*,y^*)(x-x^*)^2 + 2F_{xy}(x^*,y^*)(x-x^*)(y-y^*) \\ & + F_{yy}(x^*,y^*)(y-y^*)^2\} \end{aligned}$$

which rewrites

$$F(x,y) = \alpha_0 + \alpha_1 x + \alpha_2 y + \alpha_3 x^2 + \alpha_4 y^2 + \alpha_5 xy$$

such that the implicit polynomial basis is the complete polynomials basis of order 2 with 2 variables.

The key difference between tensor product bases and complete polynomials bases lies essentially in the rate at which the size of the basis increases.

As aforementioned, tensor product bases grow exponentially while complete polynomials bases only grow polynomially. This reduces the computational cost of approximation.

But *what do we loose using complete polynomials rather than tensor product bases?*

From a theoretical point of view, Taylor's theorem gives us the answer: Nothing! Indeed, Taylor's theorem indicates that the element in $\mathcal{B}^c$ delivers a approximation in the neighborhood of x^* at exhibits an asymptotic degree of convergence equal to k.

The n-fold tensor product, $\mathcal{B}$, can deliver only a kth degree of convergence as it does not contains all terms of degree $k+1$.

In other words, complete polynomials and tensor product bases deliver the same degree of asymptotic convergence and therefore complete polynomials based approximation yields an as good level of accuracy as tensor product based approximations.

Once we have chosen a basis, we can proceed to approximation. For example, we may use Chebychev approximation in higher dimensional problems. Judd [1998] reports the algorithm for this problem.

As we will see, it takes advantage of a very nice feature of orthogonal polynomials: they inherit their orthogonality property even if we extend them to higher dimensions.

Let us then assume we want to compute the Chebychev approximation of a 2-dimensional function $F(x, y)$ over the interval $[a_x; b_x] \times [a_y; b_y]$ and let us assume – to keep things simple for a while – that we use a tensor product basis.

Then, the algorithm is as follows:

1. Choose a polynomial order for $x\,(n_x)$ and $y\,(n_y)$

2. Compute $m_x \geq n_x + 1$ and $m_y \geq n_y + 1$ Chebychev interpolation nodes on $[-1; 1]$

$$z_k^x = \cos\left(\frac{2k-1}{2m_x}\pi\right), \quad k = 1, ..., m_x$$

and

$$z_k^y = \cos\left(\frac{2k-1}{2m_y}\pi\right), \quad k = 1, ..., m_y$$

3. Adjust the nodes to fit in both interval

$$x_k = a_x + (1 + z_k^x)\left(\frac{b_x - a_x}{2}\right), \quad k = 1, ..., m_x$$

and

$$y_k = a_y + \left(1 + z_k^y\right)\left(\frac{b_y - a_y}{2}\right), \quad k = 1, ..., m_y$$

4. Evaluate the function F at each node to form

$$\Omega \equiv \{\omega_{kl} = F(x_k, y_l) : k = 1, ..., m_x; l = 1, ..., m_y\}$$

5. Compute the $(n_. + 1)\times(n_y + 1)$ Chebychev coefficients α_{ij}, $i = 0, ..., n_x$, $j = 0, ...n_y$, as

$$\alpha_{ij} = \frac{\sum_{k=1}^{m_x}\sum_{l=1}^{m_y}\omega_{kl}T_i^x\left(z_k^x\right)T_j^y\left(z_l^y\right)}{\left(\sum_{k=1}^{m_x}T_i^x\left(z_k^x\right)^2\right)\left(\sum_{l=1}^{m_y}T_j^y\left(z_l^y\right)^2\right)}$$

which may be simply obtained in this case as

$$\alpha = \frac{T^x\left(z^x\right)'\Omega T^y\left(z^y\right)}{\left\|T^x\left(z^x\right)\right\|^2 \times \left\|T^y\left(z^y\right)\right\|^2}$$

6. Compute the approximation as

$$G(x,y) = \sum_{i=0}^{n_x} \sum_{j=0}^{n_y} \alpha_{ij} T_i^x \left(2\frac{x - a_x}{b_x - a_x} - 1 \right) T_j^y \left(2\frac{y - a_y}{b_y - a_y} - 1 \right)$$

which may also be obtained as

$$G(x,y) = T^x \left(2\frac{x - a_x}{b_x - a_x} - 1 \right) \alpha T^y \left(2\frac{y - a_y}{b_y - a_y} - 1 \right)'.$$

As an illustration of the algorithm we compute the approximation of the CES function

$$F(x, y) = [x^{\rho} + y^{\rho}]^{\frac{1}{\rho}}$$

on the $[0.01; 2] \times [0.01; 2]$ interval for $\rho = 0.75$.

We used 5-th order polynomials for both x and y and 20 nodes for both x and y, such that there are 400 possible interpolation nodes. Applying the algorithm we just described, we get the matrix of coefficients reported in table 7.

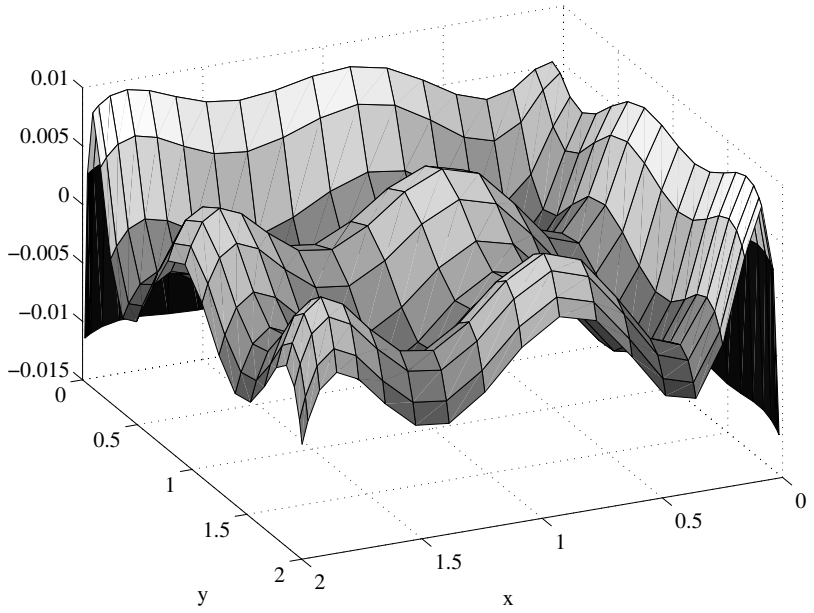
As can be seen from the table, most of the coefficients are close to zero as soon as they involve the cross-product of higher order terms, such that using a complete polynomial basis would yield the same efficiency at a lower computational cost.

Figure 10 reports the graph of the residuals for the approximation.

Table 7: Matrix of Chebychev coefficients (tensor product basis)

$k_x \backslash k_y$	0	1	2	3	4	5
0	2.4251	1.2744	-0.0582	0.0217	-0.0104	0.0057
1	1.2744	0.2030	-0.0366	0.0124	-0.0055	0.0029
2	-0.0582	-0.0366	0.0094	-0.0037	0.0018	-0.0009
3	0.0217	0.0124	-0.0037	0.0016	-0.0008	0.0005
4	-0.0104	-0.0055	0.0018	-0.0008	0.0004	-0.0003
5	0.0057	0.0029	-0.0009	0.0005	-0.0003	0.0002

Figure 10: Residuals: Tensor product basis



MATLAB CODE: CHEBYCHEV COEFFICIENTS IN $\mathbb{R}^2$ (TENSOR PRODUCT BASIS)

```
rho = 0.75;
mx  = 20;
my  = 20;
nx  = 5;
ny  = 5;
ax  = 0.01;
bx  = 2;
ay  = 0.01;
by  = 2;
%
% Step 1
%
rx  = cos((2*[1:mx]'-1)*pi/(2*mx));
ry  = cos((2*[1:my]'-1)*pi/(2*my));
%
% Step 2
%
x   = (rx+1)*(bx-ax)/2+ax;
y   = (ry+1)*(by-ay)/2+ay;
%
% Step 3
%
Y   = zeros(mx,my);
for ix=1:mx;
   for iy=1:my;
      Y(ix,iy) = (x(ix)^rho+y(iy)^rho)^(1/rho);
   end
end
%
% Step 4
%
Xx  = [ones(mx,1) rx];
for i=3:nx+1;
  Xx= [Xx 2*rx.*Xx(:,i-1)-Xx(:,i-2)];
end Xy  = [ones(my,1) ry];
for i=3:ny+1;
  Xy= [Xy 2*ry.*Xy(:,i-1)-Xy(:,i-2)];
end
T2x = diag(Xx'*Xx);
T2y = diag(Xy'*Xy);
a   = (Xx'*Y*Xy)./(T2x*T2y');
```

If we now want to perform the same approximation using a complete polynomials basis, we just have to modify the algorithm to take into account the fact that when iterating on i and j we want to impose $i + j \leq \kappa$. Let us compute is for $\kappa = 5$. This implies that the basis will consists of

$$
\begin{aligned}
&1, T_1^x(.), T_1^y(.), T_2^x(), T_2^y(.), T_3^x(.), T_3^y(.), T_4^x(.), T_4^y(), T_5^x(.), T_5^y(.), \\
&T_1^x(.) T_1^y(.), T_1^x() T_2^y(.), T_1^x(.) T_3^y(), T_1^x(.) T_4^y(.), \\
&T_2^x(.) T_1^y(.), T_2^x() T_2^y(.), T_2^x(.) T_3^y(), \\
&T_3^x(.) T_1^y(.), T_3^x() T_2^y(.), \\
&T_4^x(.) T_1^y(.)
\end{aligned}
$$

Table 8: Matrix of Chebychev coefficients (Complete polynomials basis)

$k_x \backslash k_y$	0	1	2	3	4	5
0	2.4251	1.2744	-0.0582	0.0217	-0.0104	0.0057
1	1.2744	0.2030	-0.0366	0.0124	-0.0055	–
2	-0.0582	-0.0366	0.0094	-0.0037	–	–
3	0.0217	0.0124	-0.0037	–	–	–
4	-0.0104	-0.0055	–	–	–	–
5	0.0057	–	–	–	–	–

A first thing to note is that the coefficients that remain are the same as the one we got in the tensor product basis.

This should not be of any surprise as what we just find is just the expression of the Chebychev economization we already encountered in the unidimensional case and which is just the direct consequence of the orthogonality condition of Chebychev polynomials.

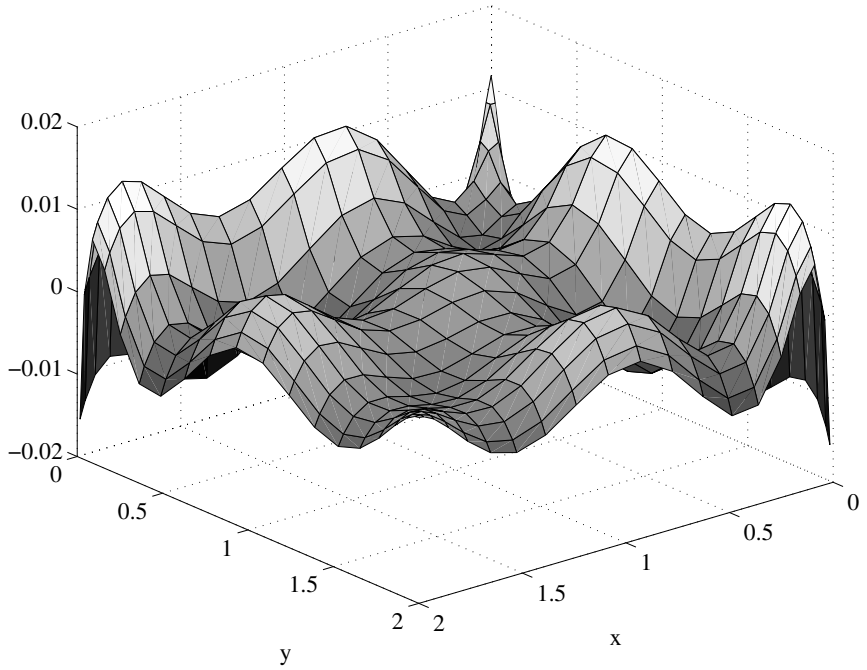
Figure 11 report the residuals from the approximation using the complete basis. As can be seen from the figure, this "constrained" approximation yields quantitatively similar results compared to the tensor product basis, therefore achieving almost the same accuracy while being less costly from a computational point of view.

In the matlab code section, we just report the lines in step 4 that are affected by the adoption of the complete polynomials basis.

Matlab Code: Complete Polynomials Specificities

```
a=zeros(nx+1,ny+1);
for ix=1:nx+1;
   iy = 1;
   while ix+iy-2<=kappa
         a(ix,iy)=(Xx(:,ix)'*Y*Xy(:,iy))./(T2x(ix)*T2y(iy));
         iy=iy+1;
    end
end
```

Figure 11: Residuals: Complete polynomials basis



Finite element approximations

Finite element are extremely popular among engineers (especially in aeronautics).

This approach considers elements that are zero over most of the domain of approximation.

Although they are extremely powerful in the case of 2-dimensional problems, they are more difficult to implement in higher dimensions.

We therefore focus on the bi-dimensional case.

Bilinear approximations

A bilinear interpolation proposes to interpolate data linearly in both coordinate directions. Assume that we have the values of a function $F(x, y)$ at the four points

$$P_1 = (-1, -1) \qquad P_2 = (-1, 1)$$
$$P_3 = (1, -1) \qquad P_4 = (1, 1)$$

A cardinal interpolation basis on $[-1; 1] \times [-1; 1]$ is provided by the set of functions

$$b_1(x, y) = \frac{1}{4}(1 - x)(1 - y) \qquad b_2(x, y) = \frac{1}{4}(1 - x)(1 + y)$$
$$b_3(x, y) = \frac{1}{4}(1 + x)(1 - y) \qquad b_4(x, y) = \frac{1}{4}(1 + x)(1 + y)$$

All functions b_i are zero on all $P_j, i \neq j$, but on the point to which is associated the same index $(i = j)$. Therefore, an approximation of $F(x, y)$ on $[-1; 1] \times [-1; 1]$ is given by

$$\begin{aligned} & F(x, y) \\ \simeq \; & F(-1, -1)\, b_1(x, y)\, F(-1, 1)\, b_2(x, y)\, F(1, -1)\, b_3(x, y)\, F(1, 1)\, b_4(x, y). \end{aligned}$$

If we have data on $[a_x; b_x] \times [a_y; b_y]$, we use the linear transformation we have already encountered a great number of times

$$\left(2\frac{x - a_x}{b_x - a_x} - 1, 2\frac{y - a_y}{b_y - a_y} - 1 \right)$$

Then, if we have Lagrange data of the type $\{x_i, y_i, z_i : i = 1, ..., n\}$, we proceed as follows

1. Construct a grid of nodes for x and y;

2. Construct the interpolant over each square applying the previous scheme;

3. Piece all interpolant together.

Note that an important issue of piecewise interpolation is related to the continuity of the approximation: individual pieces must meet continuously at common edges.

In bilinear interpolation, this is insured by the fact that two interpolants overlap only at the edges of rectangles on which the approximation is a linear interpolant of 2 common end points.

This would not be insured if we were to construct bi-quadratic or bi-cubic interpolations for example.

Note that this type of interpolation scheme is typically what is done when a computer draw a 3d graph of a function.

In figure 12, we plot the residuals of the bilinear approximation of the CES function we approximated in the previous section, with 5 uniform intervals (6 nodes such that

$$x = \{0.010, 0.408, 0.806, 1.204, 1.602, 2.000\},$$

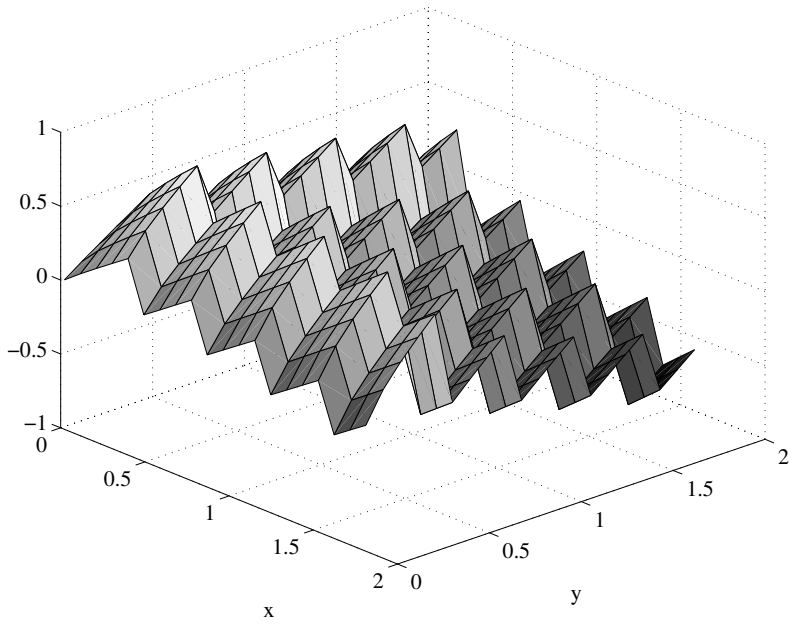
and

$$y = \{0.010, 0.408, 0.806, 1.204, 1.602, 2.000\}).$$

Like in the spline approximation procedure, the most difficult step once we have obtained an approximation is to determine the square the point for which we want an approximation belongs to.

We therefore face exactly the same type of problems.

Figure 12: Residuals of the bilinear approximation



Simplicial 2D linear interpolation

This method essentially amounts to consider triangles rather than rectangles as an approximation basis. The idea is then to build triangles in the x-y plane.

To do so, and assuming that the Lagrange data have already been transformed using the linear map described earlier, we set 3 points

$$P_1 = (0,0)\,, \quad P_2 = (0,1)\,, \quad P\,(1,0)$$

to which we associate 3 functions

$$b_1\,(x,y) = 1 - x - y, \quad b_2\,(x,y) = y, \quad b_3\,(x,y) = x$$

which are such that all functions b_i are zero on all $P_j, i \neq j$, but on the point to which is associated the same index $(i = j)$. b_1, b_2 and b_3 are the cardinal functions on P_1, P_2, P_3.

Let us now add the point $P_4 = (1,1)$, then we have the following cardinal functions for P_2, P_3, P_4 :

$$b_4(x,y) = 1 - x, \quad b_5(x,y) = 1 - y, \quad b_6(x,y) = x + y - 1.$$

Therefore, on the square P_1, P_2, P_3, P_4 the interpolant for F is given by

$$\begin{aligned} & G(x,y) \\ = & \begin{cases} F(0,0)(1-x-y) + F(0,1)\,y + F(1,0)\,x & \text{if } x+y \leq 1 \\ F(0,1)(1-x) + F(1,0)(1-y) + F(1,1)(x+y-1) & \text{if } x+y \geq 1 \end{cases} \end{aligned}$$

It should be clear to you that if these methods are pretty easy to implement in dimension 3, it becomes quite cumbersome in higher dimensions, and not that much is proposed in the literature, are most of these methods were designed by engineers and physicists that essentially have to deal with 2, 3 at most 4 dimensional problems.

We will see that this will be a limitation in a number of economic applications.

Bibliography

Hornik, K., M. Stinchcombe, and H. White, Multi-Layer Feedforward Net works are Universal Approximators, Neural Networks, 1989, 2, 359-366.

Judd, K.L., Numerical methods in economics, Cambridge, Massachussets: MIT Press, 1998.

Schumaker, L.L., On Shape-preseving Quadratic Spline Interpolation, SIAM Journal of Numerical Analysis, 1983, 20, 854-864.