COMPUTING EQUILIBRIUM WITH HETEROGENEOUS AGENTS AND AGGREGATE UNCERTAINTY (BASED ON KRUEGER AND KUBLER, 2004)

Daniel Harenberg

daniel.harenberg@gmx.de

University of Mannheim

Econ 714, 28.11.06

What this is about

- Macro models with heterogenous agents and aggregate uncertainty, for example:
  - Stochastic shock to production
  - Idiosyncratic shock to productivity
  - Alternatively: overlapping generations
- Distribution of assets as state variable
- Approximate law of motion as in Krusell and Smith (1998)
- Multidimensional interpolation of policy functions in general computationally infeasible
- Krueger and Kubler method feasible up to 20 dimensions

Outline

1. Foreword: Interpolating with Chebychev polynomials
2. Problems in multidimensional interpolation
3. Sparse grids and Smolyak's algorithm
4. Implementation

Chebychev Interpolation: Motivation

Why use polynomials for interpolation?

Nice properties of Chebychev polynomials:

- Easy to calculate coefficients
- Relatively cheap evaluation
- (Nearly) minimizes maximum error of approximation among polynomials (near-minimax, see Judd (1998))
- Simple construction of derivatives and integrals
- Chebychev regression, Chebychev economization

Drawback: Approximated function must be smooth (C^1)

Chebychev Interpolation: Motivation

Why use polynomials for interpolation?

Nice properties of Chebychev polynomials:

- Easy to calculate coefficients
- Relatively cheap evaluation
- (Nearly) minimizes maximum error of approximation among polynomials (near-minimax, see Judd (1998))
- Simple construction of derivatives and integrals
- Chebychev regression, Chebychev economization

Drawback: Approximated function must be smooth (C^1)

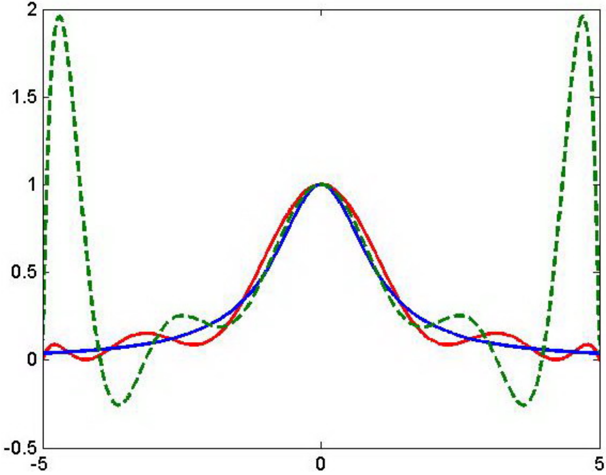
Chebychev Interpolation: Motivation

Why use polynomials for interpolation?

Nice properties of Chebychev polynomials:

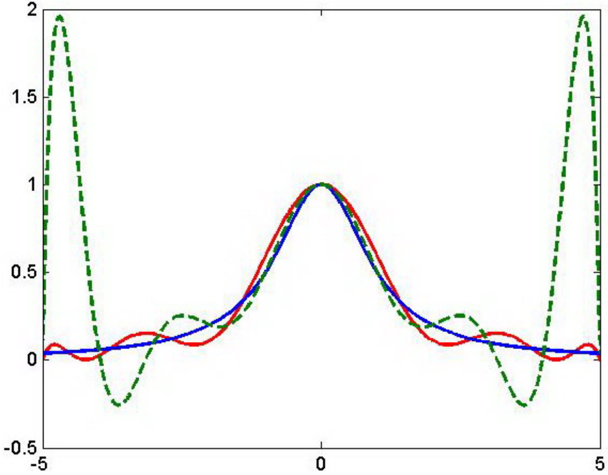
- Easy to calculate coefficients
- Relatively cheap evaluation
- (Nearly) minimizes maximum error of approximation among polynomials (near-minimax, see Judd (1998))
- Simple construction of derivatives and integrals
- Chebychev regression, Chebychev economization

Drawback: Approximated function must be smooth (C^1)



Chebychev zeros: $z_i = -\cos\left(\frac{(2i-1)\pi}{2n}\right), \quad i = 1, \dots, n$

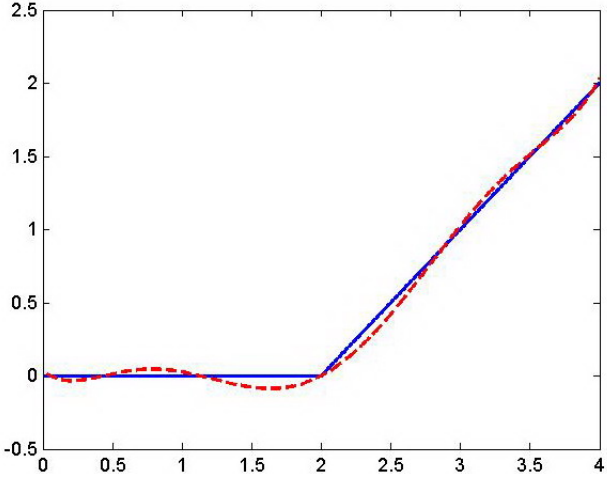
$x \in [-1, 1]$



Chebychev zeros: $z_i = -\cos\left(\frac{(2i-1)\pi}{2n}\right), \quad i = 1, \ldots, n$

$x \in [-1, 1]$

Chebychev Interpolation: Problems with kinks



Chebychev Interpolation: Where to look

- For formulae, algorithm and theoretical background, see appendix.
- Makoto: very good slides on Chebychev, theoretical background (projection methods, *wrm.pdf)
- Judd (1998): Regression, 2-dimensional interpolation
- Press et al. (1992) show derivatives, Fortran codes.
- Aruoba et al. (2006): compare algorithms for computing standard stochastic growth model, Fortran codes
- Implementation in Matlab

Multidimensional Interpolation: Problems

Generalize from 1-dimensional interpolation

$\Rightarrow$ Construction of grid by Tensor product

A) Linear interpolation:

- Bilinear interpolation (Fortran code from Press et al.)
- Simplicial interpolation (Judd)
- Not monotone, not smooth in general

B) Polynomial interpolation:

- Tensor product of one-dimensional monomials
- Curse of dimensionality: exp. growth of nodes & coeffs
- example: 20 generations, asset grid: 10 nodes

Multidimensional Interpolation: Problems

Generalize from 1-dimensional interpolation

$\Rightarrow$ Construction of grid by Tensor product

A) Linear interpolation:

- Bilinear interpolation (Fortran code from Press et al.)
- Simplicial interpolation (Judd)
- Not monotone, not smooth in general

B) Polynomial interpolation:

- Tensor product of one-dimensional monomials
- Curse of dimensionality: exp. growth of nodes & coeffs
- example: 20 generations, asset grid: 10 nodes

Multidimensional Interpolation: Problems

Generalize from 1-dimensional interpolation

$\Rightarrow$ Construction of grid by Tensor product

A) Linear interpolation:

- Bilinear interpolation (Fortran code from Press et al.)
- Simplicial interpolation (Judd)
- Not monotone, not smooth in general

B) Polynomial interpolation:

- Tensor product of one-dimensional monomials
- Curse of dimensionality: exp. growth of nodes & coeffs
- example: 20 generations, asset grid: 10 nodes

Multidimensional Interpolation: A solution

Identified 2 problems:

1. How to handle exponential growth of grid?
2. How to choose nodes and interpolators and combine them?

Krueger and Kubler propose:

1. Construct Sparse Grids.
2. Apply Smolyak's Algorithm to combine selected low-dimensional polynomials.

2 comments up front:

- Known in numerics and engineering, new to econ.
- Does not presuppose or exploit economic structure.

Multidimensional Interpolation: A solution

Identified 2 problems:

1. How to handle exponential growth of grid?
2. How to choose nodes and interpolators and combine them?

Krueger and Kubler propose:

1. Construct Sparse Grids.
2. Apply Smolyak's Algorithm to combine selected low-dimensional polynomials.

2 comments up front:

- Known in numerics and engineering, new to econ.
- Does not presuppose or exploit economic structure.

Multidimensional Interpolation: A solution

Identified 2 problems:

1. How to handle exponential growth of grid?
2. How to choose nodes and interpolators and combine them?

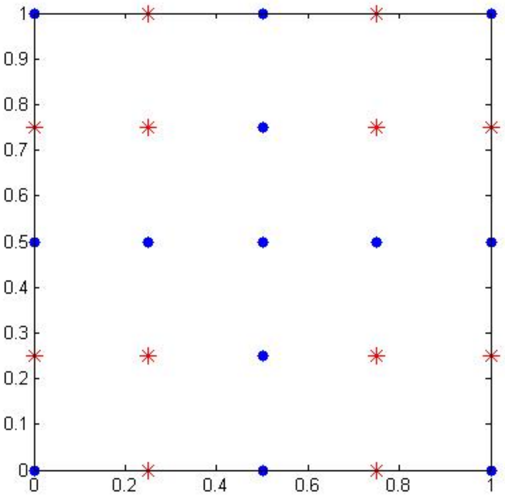
Krueger and Kubler propose:

1. Construct Sparse Grids.
2. Apply Smolyak's Algorithm to combine selected low-dimensional polynomials.

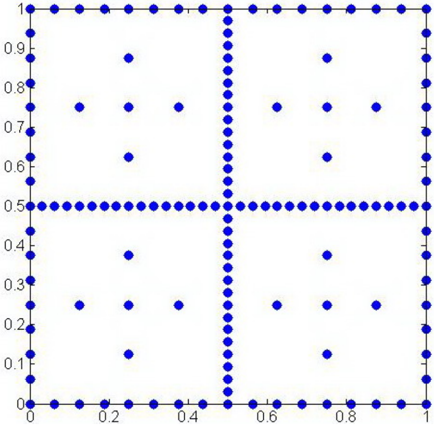
2 comments up front:

- Known in numerics and engineering, new to econ.
- Does not presuppose or exploit economic structure.

Tensor vs. sparse grid

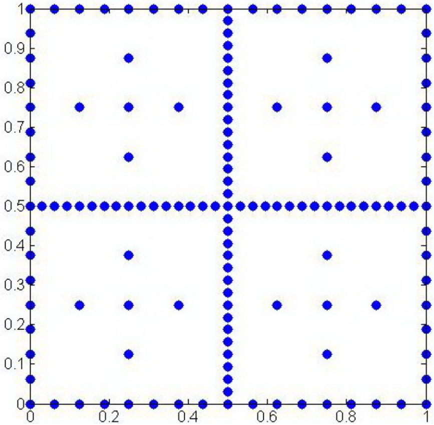


Higher degree sparse grid (q=7, d=2)



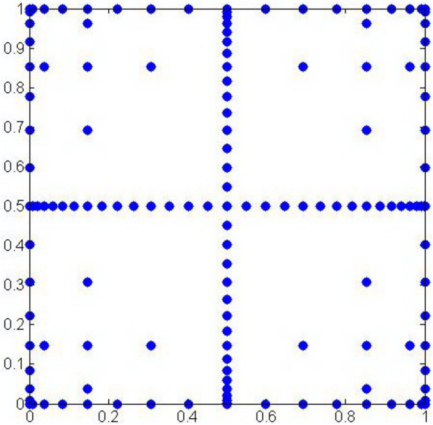
$$\mathcal{H}_{q,d} = \bigcup_{q-d+1 \le |\mathbf{i}| \le q} \left(\mathcal{G}^{i_1} \times \cdots \times \mathcal{G}^{i_d} \right)$$

Higher degree sparse grid (q=7, d=2)



$$\mathcal{H}_{q,d} = \bigcup_{q-d+1 \leq |\mathbf{i}| \leq q} \left(\mathcal{G}^{i_1} \times \cdots \times \mathcal{G}^{i_d} \right)$$

Sparse grid (q=7, d=2) of Chebychev extrema



$$\mathcal{H}_{q,d} = \bigcup_{q-d+1 \le |\mathbf{i}| \le q} \left(\mathcal{G}^{i_1} \times \cdots \times \mathcal{G}^{i_d} \right)$$

Smolyak's algorithm

Intuition from multidimensional **Taylor-expansion**:

$$f(x) \approx f(x^0) \; + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(x^0)(x_i - x_i^0)$$

$$\vdots$$

$$+ \frac{1}{k!} \sum_{i_1=1}^{n} \cdots \sum_{i_k=1}^{n} \frac{\partial^k f}{\partial x_{i_1} \cdots \partial x_{i_k}}(x^0)(x_{i_1} - x_{i_1}^0) \cdots (x_{i_k} - x_{i_k}^0)$$

Formula for **Smolyak's algorithm**:

$$\hat{\mathcal{F}}_{q,d}(x) = \sum_{q-d+1 \le |\mathbf{i}| \le q} (-1)^{q-|\mathbf{i}|} \binom{d-1}{q-|\mathbf{i}|} \left(p^{i_1}(x_1) \cdots p^{i_d}(x_d) \right).$$

Smolyak's algorithm

Intuition from multidimensional **Taylor-expansion**:

$$f(x) \approx f(x^0) \; + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(x^0)(x_i - x_i^0)$$

$$\vdots$$

$$+ \frac{1}{k!} \sum_{i_1=1}^{n} \cdots \sum_{i_k=1}^{n} \frac{\partial^k f}{\partial x_{i_1} \cdots \partial x_{i_k}}(x^0)(x_{i_1} - x_{i_1}^0) \cdots (x_{i_k} - x_{i_k}^0)$$

Formula for **Smolyak's algorithm**:

$$\hat{\mathcal{F}}_{q,d}(x) = \sum_{q-d+1 \leq |\mathbf{i}| \leq q} (-1)^{q-|\mathbf{i}|} \binom{d-1}{q-|\mathbf{i}|} \left(p^{i_1}(x_1) \cdots p^{i_d}(x_d) \right).$$

Implementation: The Model of KK04

- OLG model with aggregate uncertainty
- Agent born at time $s = t - j + 1$
- Discrete shock z to productivity $\zeta(z)$ and depreciation $\delta(z)$
- Asset distribution is state variable

  $\Rightarrow$ one dimension for each generation

$$f(K, L, z) = \zeta(z) K^{\alpha} N^{1-\alpha} + K(1 - \delta(z)) \quad (1)$$

$$\{c_s, a_s\} \in \arg\max_{\tilde{c}_s, \tilde{a}_s} E_s \left[\sum_{j=1}^{J} \beta^{j-1} \frac{c_{j,t+j-1}^{1-\sigma}}{1-\sigma} \right] \quad (2)$$

$$a_{j,t+1} = R_t a_{j,t} + \vartheta_j w_t - c_{j,t} \quad (3)$$

Implementation: The Model of KK04

- OLG model with aggregate uncertainty
- Agent born at time $s = t - j + 1$
- Discrete shock z to productivity $\zeta(z)$ and depreciation $\delta(z)$
- Asset distribution is state variable

  $\Rightarrow$ one dimension for each generation

$$
\begin{aligned}
f(K, L, z) &= \zeta(z) K^{\alpha} N^{1-\alpha} + K(1 - \delta(z)) & (1)\\
\{c_s, a_s\} &\in \arg\max_{\tilde{c}_s, \tilde{a}_s} E_s \left[\sum_{j=1}^{J} \beta^{j-1} \frac{c_{j,t+j-1}^{1-\sigma}}{1-\sigma} \right] & (2)\\
a_{j,t+1} &= R_t a_{j,t} + \vartheta_j w_t - c_{j,t} & (3)
\end{aligned}
$$

Implementation: Solving the KK04-model

- Looking for policy function $\hat{a}_{j,z}(s;\theta)$ where θ is a vector of Chebychev coefficients
- Euler equations: $\forall\ j = 1, \ldots, J-1;\ \forall\ s \in\ \mathcal{H};\ \forall\ z$

$$
\begin{aligned}
u_c(\hat{c}_j(s,z;\theta)) &= \beta E_z R(\hat{s}', z') u_c(\hat{c}_{j+1}(\hat{s}', z';\theta)) \\
\text{where} \quad \hat{s}' &= \left(\hat{a}_{1,z}(s;\theta), \ldots, \hat{a}_{J-1,z}(s;\theta)\right)
\end{aligned}
$$

- high-dimensional, nonlinear system of equations in θ
- High demands on nonlinear root finder (details)
- Simulate to get endogenous asset distribution

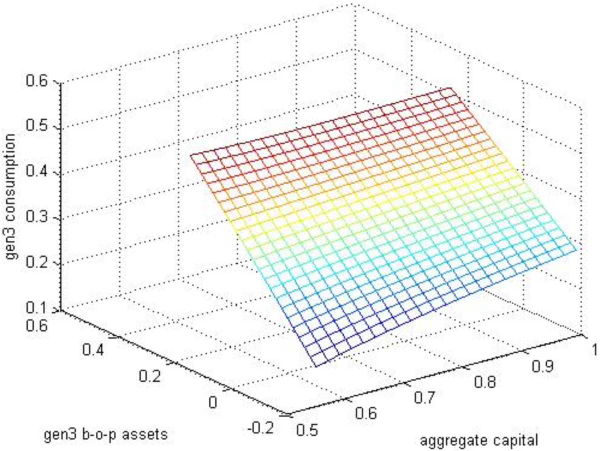
Implementation: Solving the KK04-model

- Looking for policy function $\hat{a}_{j,z}(s;\theta)$ where θ is a vector of Chebychev coefficients
- Euler equations: $\forall\ j = 1, \ldots, J-1;\ \forall\ s \in\ \mathcal{H};\ \forall\ z$

$$\begin{aligned} u_c(\hat{c}_j(s,z;\theta)) &= \beta E_z R(\hat{s}', z') u_c(\hat{c}_{j+1}(\hat{s}', z';\theta)) \\ \text{where} \quad \hat{s}' &= (\hat{a}_{1,z}(s;\theta), \ldots, \hat{a}_{J-1,z}(s;\theta)) \end{aligned}$$

- high-dimensional, nonlinear system of equations in θ
- High demands on nonlinear root finder (details)
- Simulate to get endogenous asset distribution

Equilibrium consumption policy of generation 3



$$\hat{c}_{3,z=1} = c_{3,1}(K_t, a_{t,2}, a_{t,3}, a_{t,5} \mid a_{t,2} = \bar{a}_2, a_{t,5} = \bar{a}_5)$$

Implementation: Coding the algorithm

- Krueger and Kubler used Fortran: about 1,5h for 20 generations, 30h for 30 generations
- Programming algorithm harder than it seems
- Pontus Rendahl, EUI: code not online anymore
- Andreas Klimke's Sparse Grid interpolation toolbox: Cave! (Problem with Euler equations)
- C++ code for Smolyak quadrature (e.g. Dynare++)

Thank you for your attention!

Implementation: Coding the algorithm

- Krueger and Kubler used Fortran: about 1,5h for 20 generations, 30h for 30 generations
- Programming algorithm harder than it seems
- Pontus Rendahl, EUI: code not online anymore
- Andreas Klimke's Sparse Grid interpolation toolbox: Cave! (Problem with Euler equations)
- C++ code for Smolyak quadrature (e.g. Dynare++)

Thank you for your attention!

Appendix contents

Appendix - Chebychev Interpolation formulae

Evaluation: $\hat{f}(x) = \sum_{i=0}^{n} \theta_i T_i(z)$, $z \in [1-, 1]$, $x \in [a, b]$

with $T_0 = 1$, $T_1 = z$, $T_{i+1}(z) = 2zT_i(z) - T_{i-1}(z)$

- Defined on [-1,1], scale to [a,b]: $x_i = (z_i + 1)\left(\frac{b-a}{2}\right) + a$
- As nodes, use Chebychev roots (see slide 5).
- Let m be number of interpolation nodes. For $m > n + 1$ we have Chebychev Regression.
- Then coefficients can be calculated as

$$\theta_j = \frac{2}{m} \sum_{i=1}^{m} T_j(z_i) f(z_i) \quad \left(= \frac{\sum_{i=1}^{m} T_j(z_i) f(z_i)}{\sum_{i=1}^{m} T_j(z_i)^2} \right)$$

Appendix - Chebychev Algorithm

Algorithm (Chebychev Regression, Judd (1998))

1. *Choose m interpolation nodes and the degree of polynomial approximation* $n < m$
2. *Compute* $m \geq n+1$ *nodes (roots) on* $[-1, 1]$*:*
   $z_i = -\cos\left(\frac{(2i-1)\pi}{2n}\right) \quad , \quad i = 1, \ldots, m.$
3. *Adjust to interval* $[a, b]$*:*
   $x_i = (z_i + 1)\left(\frac{b-a}{2}\right) + a \, , \quad i = 1, \ldots, m.$
4. *Evaluate f:* $y_i = f(x_i)$.
5. *Compute coefficients:* $\theta_j = \frac{2}{m}\sum_{i=1}^{m} T_j(z_i) y_i$

Approximation for $x \in [a, b]$: $\quad \hat{f}(x) = \sum_{i=0}^{n} \theta_i T_i\left(2\frac{x-a}{b-a} - 1\right)$

Appendix - Chebychev theoretical background

- Definition: $T_i(x) = \cos(i \cos^{-1} x)$.
- Expensive to compute, recursive formulation more efficient
- Family of orthogonal polynomials defined by

$$\int_a^b T_i(x)T_j(x)w(x)dx = 0 \ , \quad i \neq j \ ,$$

  where $w(x)$ is a weighting function. For Chebychev: $w(x) = \sqrt{(1 - x^2)}$.
- See Makotos slides on projection methods (Weighted Residual Methods, wrm.pdf), or Heer and Maußner (2005).
- Orthogonal polynomials belong to projection methods, with testing function the Dirac delta function.

Appendix - Tensor product

- If A and B are sets of functions their tensor product is

$$A \bigotimes B = \{\phi(x)\psi(y) | \phi \in A, \psi \in B\} .$$

- For certain cases also called Kronecker product.
- If x and y are vectors with 4 points in one dimension each, the Tensor grid is represented by

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \bigotimes \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} = \begin{bmatrix} (x_1, y_1) & (x_2, y_1) & (x_3, y_1) & (x_4, y_1) \\ (x_1, y_2) & (x_2, y_2) & (x_3, y_2) & (x_4, y_2) \\ (x_1, y_3) & (x_2, y_3) & (x_3, y_3) & (x_4, y_3) \\ (x_1, y_4) & (x_2, y_4) & (x_3, y_4) & (x_4, y_4) \end{bmatrix}$$

Appendix - Computational complexity of grid

Exponential and polynomial complexity

Let $\mathcal{H}_{q,d}$ denote the set of gridpoints depending on the number of dimensions d and the order of the interpolating polynomials q. Let $\nu(\mathcal{H}_{q,d})$ be a function returning the total number of nodes in the set. For given q and functions $g(q),\ h(q)$ the computational costs of computing the grid can be written as

(i) Exponential complexity: $\nu(\mathcal{H}_{q,d}) \in O(g(q)^d)$

(ii) Polynomial complexity: $\nu(\mathcal{H}_{q,d}) \in O(d^{h(q)})$

- See definition of big O notation on next slide.
- Slightly more loosely, this implies $\exists\ M\ :\ \frac{\nu(\mathcal{H}_{q,d})}{g(q)^d} \leq M$.
- Simply put: grid grows polynomially in dimension.

Appendix - Big O notation

Definition (Big O notation)

Let $f(x)$ and $g(x)$ be real functions.

$$
\begin{aligned}
& f(x) \in O(g(x)) \;\text{ as }\; x \to \infty \\
\Leftrightarrow \quad & \exists\, x_0,\ \exists\, M > 0 \;\text{ s. th. }\; |f(x)| \leq M|g(x)| \text{ for } x > x_0.
\end{aligned}
$$

We say that $f(x)$ is of order $g(x)$.

- Used in two senses:
  - (i) functional convergence
  - (ii) computational complexity
- In our setting, we need it to describe
  - (i) Convergence of the approximating to true function
  - (ii) Rate of growth of grid size (computational complexity)

Appendix - Smolyak details

$$\mathcal{H}_{q,d} = \bigcup_{q-d+1 \leq |\mathbf{i}| \leq q} \left(\mathcal{G}^{i_1} \times \cdots \times \mathcal{G}^{i_d} \right)$$

- Multi-index $\mathbf{i} \in \mathbb{N}^d$ with $|\mathbf{i}| = \sum_{l=1}^{d} i_l$
- Number of nodes in dimension i: $m_i = 2^{i-1} + 1$
- Nested Cheb *extrema*: $k_l^i = -\cos\left(\frac{\pi(k-1)}{m_i - 1}\right)$
- Recall that Binomial Coefficient defined as the number of ways that n objects can be chosen from k objects, regardless of order (speak "n choose k"): $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

$$\hat{\mathcal{F}}_{q,d}(x) = \sum_{q-d+1 \leq |\mathbf{i}| \leq q} (-1)^{q-|\mathbf{i}|} \binom{d-1}{q-|\mathbf{i}|} \left(p^{i_1}(x_1) \cdots p^{i_d}(x_d) \right).$$

Appendix - Smolyak details

$$\mathcal{H}_{q,d} = \bigcup_{q-d+1 \leq |\mathbf{i}| \leq q} \left(\mathcal{G}^{i_1} \times \cdots \times \mathcal{G}^{i_d} \right)$$

- Multi-index $\mathbf{i} \in \mathbb{N}^d$ with $|\mathbf{i}| = \sum_{l=1}^{d} i_l$
- Number of nodes in dimension i: $m_i = 2^{i-1} + 1$
- Nested Cheb *extrema*: $k_l^i = -\cos\left(\frac{\pi(k-1)}{m_i - 1}\right)$
- Recall that Binomial Coefficient defined as the number of ways that n objects can be chosen from k objects, regardless of order (speak "n choose k"): $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

$$\hat{\mathcal{F}}_{q,d}(x) = \sum_{q-d+1 \leq |\mathbf{i}| \leq q} (-1)^{q-|\mathbf{i}|} \binom{d-1}{q-|\mathbf{i}|} \left(p^{i_1}(x_1) \cdots p^{i_d}(x_d) \right).$$

Appendix - KK04 solution algorithm

Algorithm (Time iteration collocation)

i. *Guess coefficients* θ^0 *for initial* $\hat{a}^0 = \{\hat{a}^0_j\}_{j=1}^{J-1}$.

ii. *Given* θ^n *and thus* $\hat{a}^n$*, solve* $\forall\ j = 1, \ldots, J-1$, $\forall\ s \in \mathcal{H}$*, and* $\forall\ z$

$$\begin{aligned} u_c(c_j(a_{j,z}; s, z)) &= \beta E_z R(s', z') u_c(\hat{c}_{j+1}(s', z'; \theta^n)) \\ \text{where} \quad s' &= (a_{1,z}, \ldots, a_{J-1,z}) \\ c_j &= s_j R(s, z) + w(s, z) - a_{j,z} \end{aligned}$$

iii. *Compute new coefficients* θ^{n+1} *from optimal* $a_{j,z}$.

iv. *If* $\sup_{z,s\in\mathcal{H}} |\hat{a}^{n+1} - \hat{a}^n| < \tau$ *stop, else go to ii.*

References I

ARUOBA, S. B., J. FERNÁNDEZ-VILLAVERDE, AND J. F. RUBIO-RAMÍREZ (2006): "Comparing solution methods for dynamic equilibrium economies," *Journal of Economic Dynamics and Control*, 30, 2477–2508.

HEER, B. AND A. MAUSSNER (2005): *Dynamic General Equilibrium Modelling: Computational Methods and Applications*, Berlin: Springer.

JUDD, K. L. (1998): *Numerical Methods in Economics*, Cambridge, MA: The MIT Press, 2nd ed.

References II

KRUEGER, D. AND F. KUBLER (2004): “Computing Equilibrium in OLG Models with Stochastic Production,” *Journal of Economic Dynamics and Control*, 28, 1411–1436.

KRUSELL, P. AND A. A. SMITH, JR. (1998): “Income and Wealth Heterogeneity in the Macroeconomy,” *Journal of Political Economy*, 106, 867–896.

PRESS, W. H., B. P. FLANNERY, S. A. TEUKOLSKY, AND W. T. VETTERLING (1992): *Numerical Recipes in FORTRAN 77: The Art of Scientific Computing*, Cambridge: Cambridge University Press, 2nd ed.