Workshop on AI, Machine Learning for Economists
5-7: Solving Dynamic Equilibrium Models–VFI, Perturbation, Projection and Beyond

Zhigang Feng

January 14, 2026

Optimal Growth Model via Perturbation

Solving Functional Equations

- We are interested in solving functional equations of the form:

$$H(d, x, \theta) \;=\; 0,$$

  where d is an **unknown decision rule** we want to determine as a function of (x, θ).

- **Perturbation approach:**
  - We look for a solution $d^n(x, \theta)$ expanded in a series (e.g., polynomial, power series) around a known point (x_0, θ_0).
  - Example of a polynomial expansion:

$$d^n(x, \theta) \;=\; \sum_{i=0}^{n} \alpha_i \, (x - x_0)^i,$$

    where $\{\alpha_i\}$ depend on θ or are further expanded in terms of θ.
  - Coefficients $\{\alpha_i\}$ are determined by enforcing $H(d^n(x, \theta), x, \theta) = 0$ *to the desired order.*

Role of the Implicit Function Theorem

- **Implicit Function Theorem (IFT):**
  - Suppose we have $H(d, x, \theta) = 0$ and at some point (d_0, x_0, θ_0), the partial derivative $\frac{\partial H}{\partial d}(d_0, x_0, \theta_0) \neq 0$.
  - Then there exists a neighborhood of (x_0, θ_0) where we can solve for d **as a function** of (x, θ):

$$d = f(x, \theta).$$

  - In other words, the condition $\frac{\partial H}{\partial d} \neq 0$ ensures *local solvability*.
- **Interpretation for Perturbation:**
  - Once the IFT guarantees $d = f(x, \theta)$ near (x_0, θ_0), we can expand f in a power series or other functional approximation.
  - This is the core idea of "perturbation," where we assume small deviations from (x_0, θ_0) and match terms order-by-order.

Why the Derivatives Matter

- **Two different derivatives:**
  1. $\dfrac{\partial H}{\partial d}$ at (d_0, x_0, θ_0).
     - Must be nonzero for the IFT to hold.
     - Ensures we can *locally* treat d as an explicit function of (x, θ).
  2. $\dfrac{\partial}{\partial (x,\theta)} \Big[H(d(x,\theta), x, \theta) \Big]$.
     - After we know $d = f(x,\theta)$ solves $H(d, x, \theta) = 0$, the function

       $$G(x,\theta) \;=\; H\big(f(x,\theta),\, x,\, \theta\big)$$

       is identically zero for all (x, θ) (in the neighborhood).
     - Hence, $\frac{\partial G}{\partial x} = \frac{\partial G}{\partial \theta} = 0$, because $G \equiv 0$.
- **No contradiction:**
  - $\frac{\partial H}{\partial d} \neq 0$ is about *solvability in the d-direction*.
  - $\frac{\partial G}{\partial (x,\theta)} = 0$ occurs because $G(x,\theta) \equiv 0$ once we *have* that solution $d = f(x,\theta)$.

Chain Rule and Matching Conditions

- Define $G(x,\theta) := H(f(x,\theta),x,\theta)$. Then $G(x,\theta) \equiv 0$.
- By the chain rule:

$$\frac{\partial G}{\partial x}(x,\theta) \;=\; \frac{\partial H}{\partial d}\big(f(x,\theta),x,\theta\big)\,\frac{\partial f(x,\theta)}{\partial x} \;+\; \frac{\partial H}{\partial x}\big(f(x,\theta),x,\theta\big).$$

$$\frac{\partial G}{\partial \theta}(x,\theta) \;=\; \frac{\partial H}{\partial d}\big(f(x,\theta),x,\theta\big)\,\frac{\partial f(x,\theta)}{\partial \theta} \;+\; \frac{\partial H}{\partial \theta}\big(f(x,\theta),x,\theta\big).$$

- Since $G \equiv 0$, these partial derivatives must be zero:

$$\frac{\partial G}{\partial x}(x,\theta) = 0, \quad \frac{\partial G}{\partial \theta}(x,\theta) = 0.$$

- **Matching terms in a perturbation series:**
  - Expand $f(x,\theta)$ in a series around (x_0,θ_0).
  - Plug into $G(x,\theta)$ and match coefficients at each order to enforce these derivatives vanish.
  - Solve iteratively for the unknown coefficients of the series.

- When there is inequality constraints associated with Kuhn-Tuker conditions, we can do the following transformation.
- Consider the complementarity condition:

$$a \geq 0, \quad h \geq 0, \quad \text{and} \quad ah = 0, \tag{1}$$

where a and h are non-negative variables, and at least one of them must be zero.

- The Fischer-Burmeister function for this condition is given by:

$$\Psi^{FB}(a, h) = a + h - \sqrt{a^2 + h^2} = 0. \tag{2}$$

This function behaves differently in various regions of the(a, h) space:

- When $a > 0$ and $h = 0$, we have $\Psi^{FB}(a, 0) = a - a = 0$.
- When $a = 0$ and $h > 0$, we have $\Psi^{FB}(0, h) = h - h = 0$.
- When $a = 0$ and $h = 0$, we have $\Psi^{FB}(0, 0) = 0 - 0 = 0$.

- In all these cases, the Fischer-Burmeister function satisfies the complementarity condition. However, for $a > 0$ and $h > 0$, the function yields a non-zero value, $\Psi^{FB}(a, h) < 0$, due to the Cauchy-Schwarz inequality.

Stochastic neoclassical growth model

$$\max \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \{\log c_t\}$$
$$c_t + k_{t+1} = e^{z_t} k_t^{\alpha}, \forall t > 0$$
$$z_t = \rho z_{t-1} + \sigma \epsilon_t, \epsilon_t \sim N(0,1)$$

Equilibrium conditions:

$$\frac{1}{c_t} = \beta \mathbb{E}_t \left\{ \frac{1}{c_{t+1}} \alpha e^{z_{t+1}} k_{t+1}^{\alpha-1} \right\}$$
$$c_t + k_{t+1} = e^{z_t} k_t^{\alpha}$$
$$z_t = \rho z_{t-1} + \sigma \epsilon_t$$

Solution and steady state

- Exact solution (found by "guess and verify"):

$$c_t = (1 - \alpha\beta)e^{z_t}k_t^{\alpha}$$
$$k_{t+1} = \alpha\beta e^{z_t}k_t^{\alpha}$$

- Steady state is also easy to find:

$$k = (\alpha\beta)^{\frac{1}{1-\alpha}}$$
$$c = (\alpha\beta)^{\frac{\alpha}{1-\alpha}} - (\alpha\beta)^{\frac{1}{1-\alpha}}$$
$$z = 0$$

The goal

- We are searching for decision rules:

$$d = \begin{cases} c_t = c(k_t, z_t) \\ k_{t+1} = k(k_t, z_t) \end{cases}$$

- Then we have:

$$\begin{aligned} \frac{1}{c(k_t, z_t)} &= \beta \mathbb{E}_t \left\{ \frac{1}{c(k(k_t, z_t))} \alpha e^{\rho z_t + \sigma \epsilon_{t+1}} k(k_t, z_t)^{\alpha - 1} \right\} \\ c(k_t, z_t) + k(k_t, z_t) &= e^{z_t} k_t^{\alpha} \\ z_t &= \rho z_{t-1} + \sigma \epsilon_t \end{aligned}$$

- This is a system of functional equations.

A perturbation solution

- Rewrite the problem in terms of perturbation parameter λ.
- Different possibilities for λ. let try

$$z_t = \rho z_{t-1} + \lambda \sigma \epsilon_t$$

  - when $\lambda = 1$, the original model.
  - when $\lambda = 0$, deterministic case.
- Now we are searching for the decision rules:

$$c_t = c(k_t, z_t; \lambda)$$
$$k_{t+1} = k(k_t, z_t; \lambda)$$

Taylor's theorem

- We will build a local approximation around $(k, 0; 0)$.
- Given equilibrium conditions:

$$\frac{1}{c(k_t, z_t)} = \beta \mathbb{E}_t \left\{ \frac{1}{c(k(k_t, z_t))} \alpha e^{\rho z_t + \sigma \epsilon_{t+1}} k(k_t, z_t)^{\alpha - 1} \right\}$$
$$c(k_t, z_t) + k(k_t, z_t) = e^{z_t} k_t^{\alpha}$$

  We will take derivatives with respect to k_t, z_t, and λ and evaluate them around $(k, 0; 0)$.
- Apply Taylor's theorem and a version of the implicit-function theorem.

Why Perturb Around $\lambda = 0$?

- **Idea of perturbation:**
  - Expand the solution $\big(c(k_t, z_t; \lambda),\ k(k_t, z_t; \lambda)\big)$ around the $\lambda = 0$ case (the deterministic steady state).
  - Then treat λ as a "small" parameter controlling the shock size.
  - For $\lambda \neq 0$, capture the effect of uncertainty in a Taylor expansion.
- **Benefit:** By expanding around $\lambda = 0$, we see how a small amount of uncertainty changes policy *relative* to the deterministic solution.

Taylor's theorem

$$
\begin{aligned}
c_t = c\left(k_t, z_t; 1\right)|_{k,0,0} &= c\left(k, 0; 0\right) \\
&+ c_k\left(k, 0; 0\right)\left(k_t - k\right) + c_z\left(k, 0; 0\right) z_t + c_\lambda\left(k, 0; 0\right) \\
&+ \frac{1}{2} c_{kk}\left(k, 0; 0\right)\left(k_t - k\right)^2 + \frac{1}{2} c_{kz}\left(k, 0; 0\right)\left(k_t - k\right) z_t + \frac{1}{2} c_{k\lambda}\left(k, 0; 0\right)\left(k_t - k\right) \\
&+ \frac{1}{2} c_{zk}\left(k, 0; 0\right) z_t\left(k_t - k\right) + \frac{1}{2} c_{zz}\left(k, 0; 0\right) z_t^2 + \frac{1}{2} c_{z\lambda}\left(k, 0; 0\right) z_t \\
&+ \frac{1}{2} c_{\lambda k}\left(k, 0; 0\right)\left(k_t - k\right) + \frac{1}{2} c_{\lambda z}\left(k, 0; 0\right) \lambda z_t + \frac{1}{2} c_{\lambda^2}\left(k, 0; 0\right) \\
&+ ...
\end{aligned}
$$

Taylor's theorem

$$
\begin{aligned}
k_{t+1} = & \, k\left(k_t, z_t; 1\right)|_{k,0,0} = k\left(k, 0; 0\right) \\
& + k_k\left(k, 0; 0\right)\left(k_t - k\right) + k_z\left(k, 0; 0\right) z_t + k_\lambda\left(k, 0; 0\right) \\
& + \frac{1}{2} k_{kk}\left(k, 0; 0\right)\left(k_t - k\right)^2 + \frac{1}{2} k_{kz}\left(k, 0; 0\right)\left(k_t - k\right) z_t + \frac{1}{2} k_{k\lambda}\left(k, 0; 0\right)\left(k_t - k\right) \\
& + \frac{1}{2} k_{zk}\left(k, 0; 0\right) z_t\left(k_t - k\right) + \frac{1}{2} k_{zz}\left(k, 0; 0\right) z_t^2 + \frac{1}{2} k_{z\lambda}\left(k, 0; 0\right) z_t \\
& + \frac{1}{2} k_{\lambda k}\left(k, 0; 0\right)\left(k_t - k\right) + \frac{1}{2} k_{\lambda z}\left(k, 0; 0\right) z_t + \frac{1}{2} k_{\lambda^2}\left(k, 0; 0\right) \\
& + ...
\end{aligned}
$$

Defining $F(k_t, z_t; \lambda) \;=\; 0$

- We rewrite the equilibrium conditions in a compact form:

$$F(k_t, z_t; \lambda) \;=\; \begin{pmatrix} \dfrac{1}{c(k_t, z_t; \lambda)} \;-\; \beta\, \dfrac{\alpha\, e^{\rho\, z_t + \lambda\, \sigma\, \epsilon_{t+1}}\, \Big(k(k_t, z_t; \lambda)\Big)^{\alpha - 1}}{c\Big(k\big(k_t, z_t; \lambda\big), \rho\, z_t + \lambda\, \sigma\, \epsilon_{t+1}; \lambda\Big)} \\ c(k_t, z_t; \lambda) + k(k_t, z_t; \lambda) \;-\; e^{z_t}\, k_t^{\alpha} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

- Equivalently, define

$$F(k_t, z_t; \lambda) \;=\; H\Big(c_t,\; c_{t+1},\; k_t,\; k_{t+1},\; z_t;\; \lambda\Big),$$

  where

$$H\Big(c_t, c_{t+1}, k_t, k_{t+1}, z_t; \lambda\Big) = H\Big(\, c(k_t, z_t; \lambda),\; c\big(k(k_t, z_t; \lambda),\; z_{t+1}; \lambda\big),\; k_t,\; k(k_t, z_t; \lambda),\; z_t;\; \lambda\Big).$$

- Denote H_i as the partial derivative of H with respect to the i component and drop the evaluation at the steady state of the functions when we do not need it.

First-Order Derivatives of F

- Let $F(k_t, z_t; \lambda) = \begin{pmatrix} F^1 \\ F^2 \end{pmatrix}$. At the steady state $(k, 0)$ and $\lambda = 0$, all partials must vanish:

$$F_k(k, 0; 0) = \frac{\partial F}{\partial k}(k, 0; 0) = 0, \quad F_z(k, 0; 0) = 0, \quad F_\lambda(k, 0; 0) = 0.$$

- Concretely, each derivative is a combination of the partials of c and k. For example:

$$F_k(k, 0; 0) = H_1\, c_k \;+\; H_2\,(\dots) \;+\; H_3 \;+\; H_4\, k_k \;=\; 0,$$

  and similarly for F_z, F_λ.

- Because $F \equiv 0$ for *all* (k_t, z_t, λ) in a neighborhood of the steady state, all these partial derivatives must be zero as well (the function is identically zero).

First-order approximation

- We take derivatives of $F(k_t, z_t; \lambda)$ around k, 0, and 0.
- Because $F(k_t, z_t; \lambda)$ must be equal to zero for any possible values of k_t , z_t , and λ, the derivatives of any order of F must also be zero.
  - Suppose we have a function $H(d, x, \theta) = 0$, which defines d implicitly as a function of x and θ. The implicit function theorem states that if H is continuously differentiable and the derivative of H with respect to d is nonzero at a particular point (say, d_0, x_0, θ_0), then in some neighborhood of (x_0, θ_0), there exists a function $d = f(x, \theta)$ that satisfies $H(d, x, \theta) = 0$. In other words, we can express d explicitly as a function of x and θ near the point (x_0, θ_0).
  - Furthermore, the function $H(d(x, \theta)) = 0$ is equal to zero for all values of x, θ, then it is a constant function (equal to zero). The derivative of a constant function is zero, since the rate of change is zero: $H_x = H_\theta = 0$.
- Hence:

$$
\begin{aligned}
F_k(k, 0; 0) &= H_1 c_k + H_2 c_k k_k + H_3 + H_4 k_k = 0 \\
F_z(k, 0; 0) &= H_1 c_z + H_2 \left(c_k k_z + c_z \rho\right) + H_4 k_z + H_5 = 0 \\
F_\lambda(k, 0; 0) &= H_1 c_\lambda + H_2 \left(c_k k_\lambda + c_\lambda\right) + H_4 k_\lambda + H_6 = 0
\end{aligned}
$$

First-Order Approximation: Six Unknowns, but Two Disappear (1/2)

- **Six unknowns at linear order.**
  - When we expand the policy functions $\{\, c(k_t, z_t; \lambda),\ k(k_t, z_t; \lambda)\}$ around $(k, 0, \lambda = 0)$, we introduce the partial derivatives:
$$\big\{\, c_k,\ c_z,\ c_\lambda,\ k_k,\ k_z,\ k_\lambda \big\}.$$
  - Thus, *a priori*, we have 6 unknown coefficients.
- **The condition on λ-derivatives:**
$$F_\lambda(k, 0; 0) \ = \ H_1\, c_\lambda \ + \ H_2 \left(c_k\, k_\lambda + c_\lambda\right) \ + \ H_4\, k_\lambda \ + \ H_6 \ = \ 0.$$
  - This is *linear* in $\{c_\lambda, k_\lambda\}$.
  - If it is also *homogeneous* (no constant term), then the only solution is $c_\lambda = k_\lambda = 0$.
- **Implication:** $c_\lambda = k_\lambda = 0 \ \Rightarrow$ certainty equivalence at first order (no precautionary effect).

Why $H_6 = 0$ and the System is Homogeneous (2/2)

- **Meaning of H_6:**
  - H_6 is the term in F_λ that might act like a constant (i.e., it does not multiply c_λ or k_λ).
  - Often arises from the partial derivative of the equilibrium condition w.r.t. λ at $(k, 0, \lambda = 0)$.
- **Why H_6 typically evaluates to zero at the steady state:**
  - At $\lambda = 0$, the model is deterministic with $\mathbb{E}[\epsilon_{t+1}] = 0$.
  - A "small" shock has no *linear* effect on the system at the steady state, so the first-order term in λ drops out.
  - This leaves no free constant in the λ-derivative equation.
- **Consequence:**
  - The equation becomes *fully homogeneous* in $\{c_\lambda,\ k_\lambda\}$.
  - The unique non-degenerate solution is $c_\lambda = 0, k_\lambda = 0$.
  - $\implies$ *Certainty equivalence* at linear order.

Why Exactly Four Equations Remain? (Role of $F = 0$)

- Recall F is a *vector* of two equilibrium conditions:

$$F(k_t, z_t; \lambda) \;=\; \begin{pmatrix} F^1 \\ F^2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

- When we differentiate around $(k, 0; 0)$,

$$F_k(k, 0; 0) = 0 \quad \Longrightarrow \quad F_k^1 = 0 \;\text{ and }\; F_k^2 = 0,$$

$$F_z(k, 0; 0) = 0 \quad \Longrightarrow \quad F_z^1 = 0 \;\text{ and }\; F_z^2 = 0.$$

  - Each derivative $(F_k^1, F_k^2, F_z^1, F_z^2)$ is one scalar equation.
  - Hence, 4 scalar equations in total.

- After discarding c_λ, k_λ (both zero), we have exactly four remaining unknowns:

$$c_k,\; c_z,\; k_k,\; k_z.$$

- So these 4 equations solve for those 4 partial derivatives.

Blanchard–Kahn (1980)

The Blanchard–Kahn conditions

Objective: Provide explicit conditions ensuring existence and uniqueness of solutions to linear rational expectation models.

Consider a linearized rational expectations model expressed as:

$$\begin{aligned} Ax_{t+1|t} &= Bx_t + Cz_t, \\ z_{t+1} &= \rho z_t + \epsilon_{t+1}, \quad \epsilon_{t+1} \sim N(0, \sigma^2), \end{aligned}$$

where x_t includes state and control variables, and z_t is the exogenous state.

Key Idea: Eigenvalue decomposition of system matrices determines stability and uniqueness of equilibrium.

Applying Blanchard–Kahn to the Stochastic Growth Model

Recall the linearized stochastic growth model around the steady state $(k, c, z) = (k^*, c^*, 0)$:

$$\begin{aligned}
\hat{k}_{t+1} &= a_{11}\hat{k}_t + a_{12}z_t + a_{13}\hat{c}_t, \\
0 &= a_{21}\hat{k}_t + a_{22}z_t + a_{23}\hat{c}_t, \\
z_{t+1} &= \rho z_t + \sigma\epsilon_{t+1}, \quad \epsilon_{t+1} \sim N(0,1),
\end{aligned}$$

where hats $(\hat{k}_t, \hat{c}_t)$ denote deviations from steady state.

Express in matrix form:

$$\begin{pmatrix} \hat{k}_{t+1} \\ 0 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{pmatrix} \begin{pmatrix} \hat{k}_t \\ \hat{c}_t \end{pmatrix} + \begin{pmatrix} a_{12} \\ a_{22} \end{pmatrix} z_t$$

Eigenvalues and Stability (Stochastic Growth Model)

Define the transition matrix:

$$A = \begin{pmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{pmatrix}$$

Compute eigenvalues λ_1, λ_2 of matrix A:

- Stability requires $|\lambda| < 1$.
- Unstable eigenvalues ($|\lambda| > 1$) indicate explosive dynamics.

Eigenvalues and Stability (Stochastic Growth Model)

The eigenvalues determine the dynamic behavior of the model:

- Predetermined variables (capital, k_t) must evolve smoothly.
- Control variables (consumption, c_t) must adjust instantly to ensure bounded paths.

For this model:

- One predetermined state variable: k_t.
- One non-predetermined control variable: c_t.

Blanchard–Kahn criterion: Exactly one unstable eigenvalue is required for uniqueness and stability.

Interpreting Blanchard–Kahn Conditions

Scenarios based on eigenvalues:

1. **Uniqueness:** Exactly one eigenvalue outside the unit circle.
2. **Indeterminacy:** No eigenvalues outside the unit circle (multiple solutions exist).
3. **Non-existence:** More eigenvalues outside the unit circle than control variables (no stable solution exists).

Economic intuition:

- Capital (k_t), a predetermined variable, cannot adjust immediately.
- Consumption (c_t), a jump variable, adjusts instantly to avoid explosive dynamics.
- One unstable eigenvalue ensures exactly one control variable adjustment, guaranteeing a unique equilibrium path.

Summary: Blanchard–Kahn in practice (Growth Model)

Practical Steps:

- Compute steady state (k^*, c^*).
- Linearize Euler equation and resource constraint.
- Formulate the system in matrix form.
- Compute eigenvalues of the system matrix.
- Verify eigenvalue condition (exactly one eigenvalue outside unit circle).

Result: Explicit condition ensuring the existence and uniqueness of the equilibrium solution.

Higher Order

Second-order approximation

- We take second-order derivatives of $F(k_t, z_t; \lambda)$ around $(k, 0, 0)$:

$$
\begin{aligned}
F_{kk}(k, 0; 0) &= 0 \\
F_{kz}(k, 0; 0) &= 0 \\
F_{k\lambda}(k, 0; 0) &= 0 \\
F_{zz}(k, 0; 0) &= 0 \\
F_{z\lambda}(k, 0; 0) &= 0 \\
F_{\lambda\lambda}(k, 0; 0) &= 0
\end{aligned}
$$

- We substitute the coefficients that we already know. A linear system of 12 equations on 12 unknowns.
- We have the term $\frac{1}{2}c_{\lambda\lambda}(k, 0, 0)$, which captures precautionary behavior.
- We can continue the iteration for as long as we want. And the procedure is recursive!

Solving the Dynamic Problem via Projection

Model Setup

- The economic model consists of:
  1. **Households**: Choose consumption c_t and labor l_t to maximize utility.
  2. **Firms**: Produce output using capital k_t, labor l_t, and productivity z_t.
  3. **Aggregate Conditions**: Include resource constraints and the law of motion for capital.
- The equilibrium system is characterized by the following equations:

$$\text{(Euler Equation)} \quad \frac{1}{c_t} = \beta\,\mathbb{E}_t\left[\frac{1}{c_{t+1}}\left(1+\alpha\,k_{t+1}^{\alpha-1}(e^{z_t}l_t)^{1-\alpha}-\delta\right)\right],$$

$$\text{(Labor Supply)} \quad \psi\,\frac{c_t}{1-l_t} = (1-\alpha)\,k_{t+1}^{\alpha}(e^{z_t}l_t)^{-\alpha}\frac{1}{l_t},$$

$$\text{(Resource Constraint)} \quad c_t + k_{t+1} = k_t^{\alpha}(e^{z_t}l_t)^{1-\alpha} + (1-\delta)k_t,$$

$$\text{(Productivity Shock)} \quad z_t = \rho\,z_{t-1} + \epsilon_t.$$

- **Objective:** Find a *decision rule* $d(x)$ that satisfies the system:

$$H(d) = 0, \quad \text{where } d(x) \text{ represents optimal choices, e.g., } k_{t+1} = d(k_t, z_t), \quad l_t = \ldots$$

Projection: The Core Concept

- **Key idea**: Approximate the unknown function $d(x)$ by restricting it to a finite-dimensional subspace of functions.

$$d^n(x;\theta) \;=\; \sum_{i=0}^{n} \theta_i \, \psi_i(x),$$

  where

  - $\{\psi_i(x)\}$ is a chosen *basis* (e.g., polynomials, splines, orthogonal polynomials).
  - $\{\theta_i\}$ are the unknown coefficients to be solved for.
  - Instead of using fixed basis functions $\psi_i(x)$, we can also approximate $d(x)$ using a neural network:

$$d^n(x;\theta) = \mathcal{N}(x;\theta),$$

    where $\mathcal{N}(x;\theta)$ is a deep neural network with trainable weights θ.

- The term **"projection"** arises because we:
  1. *Project* the infinite-dimensional problem onto a finite-dimensional space.
  2. *Enforce* that the residual $R(x;\theta) = H(d^n(x;\theta))$ is "small" or zero in that finite-dimensional sense.

Why "Projection"? Mathematical Intuition

- The true solution $d(x)$ might belong to a large function space (e.g., C^1(Continuously Differentiable Functions), L^2 (Square-Integrable Functions), etc.).
- By picking a $\{\psi_i(x)\}$ basis, we reduce the problem to $\theta = (\theta_0, \ldots, \theta_n)$, a **finite** number of parameters.
- We then choose θ to ensure the equation $H(d^n)$ is satisfied in a *projected sense*, for example:
  - **Collocation**: $R(x_j; \theta) = 0$ at selected points $\{x_j\}$.
  - **Galerkin / Orthogonality**: $\langle R(\cdot; \theta), \psi_i(\cdot) \rangle = 0$ for $i = 0, \ldots, n$.
  - **Least Squares**: Minimize $\|R(\cdot; \theta)\|$ over the domain.
- Projected means that we are not enforcing $H(d^n) = 0$ everywhere in the (potentially infinite) domain. Instead, we ensure the **error or residual is small or zero** in a finite set of conditions (points, inner products, norms). This allows us to handle otherwise *infinite-dimensional* problems using only a finite number of parameters.
- This approach yields a *global* solution (valid over the entire domain of x) provided the basis and projection scheme are suitably chosen.

Projection Method: Pros and Cons

- **Pros**:
  - A **global** solution valid over the entire state space (not just near a point).
  - Flexible in handling highly nonlinear structures, since you're not linearizing around one point.
- **Cons**:
  - Potentially **high computational cost**, especially in multidimensional problems ($\Rightarrow$ "curse of dimensionality").
  - Need to choose a suitable set of basis functions $\{\psi_i\}$ and a robust method to handle the residual.

Projection vs. Perturbation

- **Perturbation Method**:
  - *Local Expansion*: Expand around a known point (e.g., steady state) using a Taylor or polynomial series.
  - *Speed and Simplicity*:
    - **Fast to compute** and easy to implement.
    - Valid primarily **near the expansion point**.
  - *High-Dimensional Feasibility*:
    - Typically more **tractable** for large state spaces, since only local information is used.
    - Accuracy **degrades** as you move away from the expansion point.
- **Projection Method**:
  - *Global Approximation*:
    - Approximates $d(x)$ over the **entire domain** using basis functions.
    - No assumption of being close to a particular point.
  - *Residual Minimization or Orthogonality*:
    - Uses techniques like **Galerkin**, **Collocation**, or **Least Squares**.
    - Ensures the **residual** is small or zero **globally**.
  - *Computational Cost*:
    - May suffer from the **curse of dimensionality** in large-scale problems.
    - Generally more **accurate** over wide ranges of the state space.

Algorithm

- Define $n+1$ known linearly independent functions $\psi_i : \Psi \to \mathbb{R}^m$, where $n < \infty$, where $\psi_0(\cdot)$,..., $\psi_n(\cdot)$ are the basis functions .
- Define a vector of coefficients $\theta = [\theta_0, \theta_1, ..., \theta_n]$.
- Define a combiniation of the basis functions and the θ's

$$d^n(\cdot;\theta) = \sum_{i=0}^{n} \theta_i \psi_i(\cdot)$$

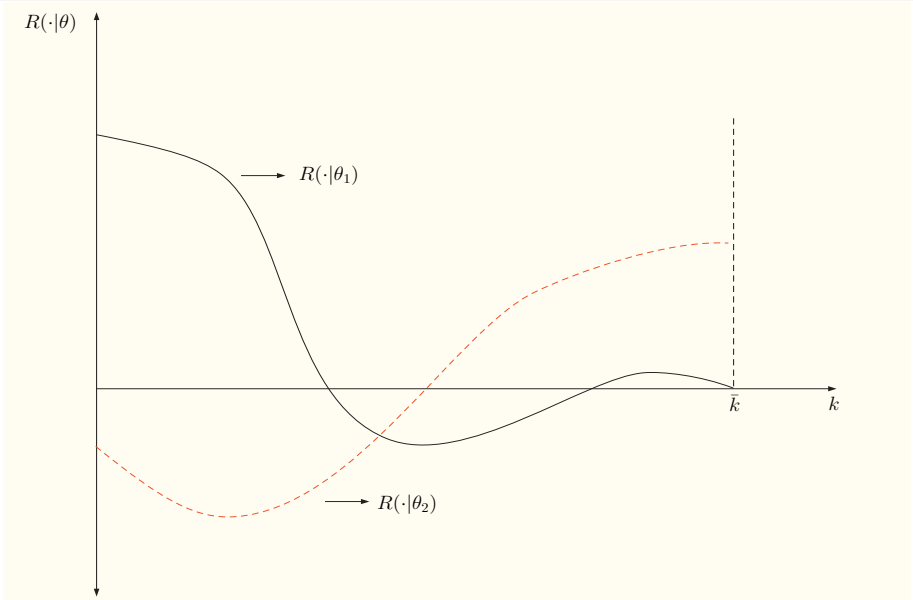
- Plug $d^n(\cdot;\theta)$ into $H()$ to find the residual equation:

$$R(\cdot;\theta) = H\left(d^n(\cdot;\theta)\right)$$

- Find $\hat{\theta}$ that minimize the residual equation under some metrics:

$$\hat{\theta} = \arg\max_{\theta} \rho\left[R(\cdot;\theta)\right]$$

Algorithm



The Choice of Basis Functions

Two issues

- We need to decide:
  - Which basis we use?
    1. Pick a global basis $\Rightarrow$ spectral methods.
    2. Pick a local basis $\Rightarrow$ finite elements methods.
  - How do we "project"?
- Different choices in 1 and 2 will result in slightly different projection methods.
- We can mix projection and perturbation methods in mixed approaches.

Spectral methods

- Main reference: Judd (1992).
- Spectral techniques use basis functions that are nonzero and smooth almost everywhere in Ψ.
- Advantages: simplicity.
- Disadvantages: difficult to capture local behavior
  - Gibbs phenomenon: the oscillatory behavior of the Fourier series of a piecewise continuously differentiable periodic function around a jump discontinuity.
- Let us start first with unidimensional problems (trick with discretization).

Gibbs Phenomenon in Polynomial Approximation

- **Definition:** Oscillations arise when approximating functions near discontinuities (or sharp corners) with polynomials or Fourier series.
- **In Macroeconomics:**
  - Policy/value functions may have kinks or binding constraints (e.g., credit constraints, irreversibility) that lead to non-smoothness.
  - Polynomial expansions (e.g., Chebyshev) can overshoot near these points, causing large local errors.
- **Implications:**
  - Increasing polynomial order does not eliminate the oscillations, only confines them closer to the discontinuity.
  - Piecewise approaches or neural network approximations can better handle sharp changes.
- **How DNNs Can Help:**
  - Neural networks use adaptive, learned basis functions, which provide better local approximation near discontinuities and reduce oscillatory artifacts.

Polynomials

$1, x, x^2, x^3, ...$

- Simple and intuitive.
- If Ω is the space of bounded measurable functions on a compact set, the Stone-Weierstrass theorem assures completeness in the L^1 norm.
  - every continuous function defined on a closed interval $[a, b]$ can be uniformly approximated as closely as desired by a polynomial function.
- Problems:
  - (Nearly) multicollinearity. Compare the graph of x10 with x11. The LS problem of fitting a polynomial of degree 6 to a function (the Hilbert Matrix) is a popular test of numerical accuracy since it maximizes rounding errors!
  - Monomials vary considerably in size, leading to scaling problems and accumulation of numerical errors.
- We want an orthogonal basis. Why?

Orthogonal Basis

- In a **function space**, two functions are **orthogonal** if their **inner product** equals zero.
- An **orthogonal basis** is beneficial for **projection methods** in **linear algebra**, **signal processing**, and **data analysis** due to several key advantages:
- **Simplicity**:
  - **Orthogonal vectors** simplify computations.
  - In an **orthogonal basis**, the **coordinates** of a vector are simply its **projections** onto the basis vectors.
  - **No need** to solve a system of equations to determine the coordinates.
- **Independence**:
  - **Orthogonal vectors** are **linearly independent**.
  - Each basis vector captures a **unique dimension** of the data.

Orthogonal Basis (Continued)

- **Numerical Stability**:
  - Computations with **orthogonal vectors** are **more stable** and **less prone to large errors** caused by small input variations.
- **Normalization**:
  - If the **orthogonal basis** is also **normalized** (*i.e.*, each vector has **length 1**), it becomes an **orthonormal basis**.
  - In an **orthonormal basis**, the **dot product** of a vector with a basis vector directly gives the **coordinate** in that direction.
- **Interpretability**:
  - **Orthogonal bases** improve interpretation in applications like **Principal Component Analysis (PCA)**.
  - **PCA transforms data** into a new **coordinate system** where the basis vectors (**principal components**) represent **directions of maximum variance**.
  - Each **principal component** is **orthogonal** (**uncorrelated**) to the others.

Projections in an Orthogonal Basis

- **Key Idea:** In an **orthogonal basis** $\{v_1, v_2, \ldots, v_n\}$, any vector x can be expressed as:

$$x = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n.$$

  The coefficients c_i are found using projections onto the basis vectors.

- **Mathematical Justification:**
  - Taking the **inner product** of both sides with v_i:

$$\langle x, v_i \rangle = \sum_{j=1}^{n} c_j \langle v_j, v_i \rangle.$$

  - Since the basis is **orthogonal**, all terms vanish except when $j = i$:

$$\langle x, v_i \rangle = c_i \langle v_i, v_i \rangle.$$

  - Solving for c_i:

$$c_i = \frac{\langle x, v_i \rangle}{\|v_i\|^2}.$$

  - If the basis is **orthonormal** ($\|v_i\| = 1$), this simplifies to:

$$c_i = \langle x, v_i \rangle.$$

Projections in an Orthogonal Basis

- **Example in $\mathbb{R}^2$:**
  - Let $x = (3, 4)$, and use the orthonormal basis:

$$v_1 = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \right), \quad v_2 = \left(\frac{-1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \right).$$

  - The coordinates are:

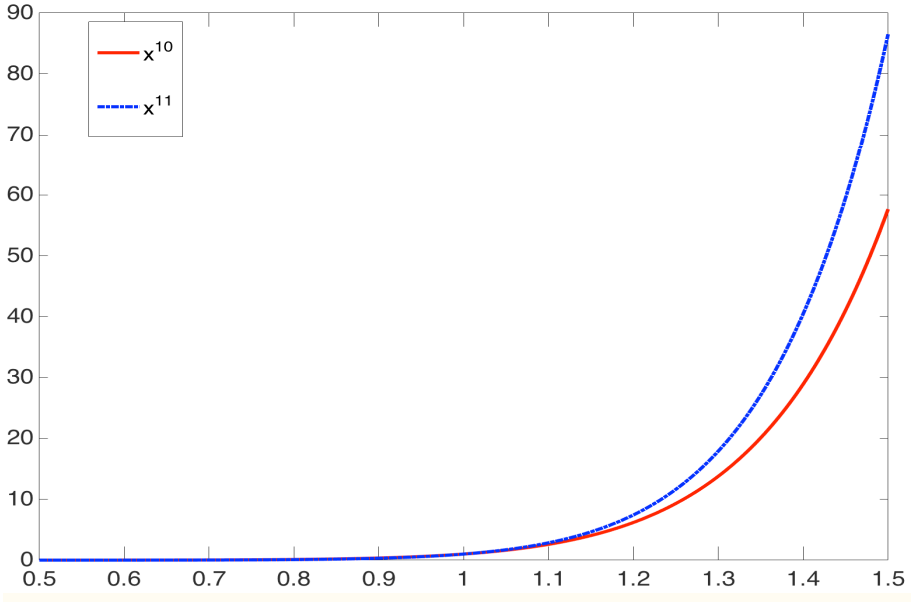
$$c_1 = \langle x, v_1 \rangle = \frac{3}{\sqrt{2}} + \frac{4}{\sqrt{2}} = \frac{7}{\sqrt{2}}.$$

$$c_2 = \langle x, v_2 \rangle = \frac{-3}{\sqrt{2}} + \frac{4}{\sqrt{2}} = \frac{1}{\sqrt{2}}.$$

  - Thus, x can be written as:

$$x = \frac{7}{\sqrt{2}} v_1 + \frac{1}{\sqrt{2}} v_2.$$

Polynomials



Trigonometric series

$\frac{1}{\sqrt{2\pi}}, \frac{\cos x}{\sqrt{2\pi}}, \frac{\sin x}{\sqrt{2\pi}}, \ldots \frac{\cos kx}{\sqrt{2\pi}}, \frac{\sin kx}{\sqrt{2\pi}}, \ldots$

- Periodic functions.
- However economic problems are generally not periodic.
- Periodic approximations to nonperiodic functions suffer from the Gibbs phenomenon.
  - the rate of convergence to the true solution as $n \to \infty$ is only $O(n)$.

Chebyshev Polynomials

Orthogonal Polynomials

- A **flexible class** of orthogonal polynomials belonging to the **Jacobi (or hypergeometric) family**.
- The **Jacobi polynomial** of degree n, $P_n^{\alpha,\beta}(x)$, for parameters $\alpha, \beta > -1$, is defined by the **orthogonality condition**:

$$\int_{-1}^{1} (1-x)^{\alpha}(1+x)^{\beta} P_n^{\alpha,\beta}(x) P_m^{\alpha,\beta}(x)\, dx = 0, \quad \text{for } m \neq n.$$

- **Explanation of Conditions:**
  - The **weight function** $(1-x)^{\alpha}(1+x)^{\beta}$ ensures that different Jacobi polynomials remain **orthogonal** over the interval $[-1, 1]$.
  - The condition $\alpha, \beta > -1$ guarantees that the integral is well-defined, preventing singularities in the weight function.
- **Two important cases of Jacobi polynomials:**
  - **Chebyshev Polynomials**: $\alpha = \beta = -\frac{1}{2}$ - Used in approximation theory and numerical methods.
  - **Legendre Polynomials**: $\alpha = \beta = 0$ - Commonly used in solving Laplace's equation and spherical harmonics.

Chebyshev polynomials

- One of the most common tools of applied mathematics:
  - Chebyshev and Fourier Spectral Methods, John P. Boyd (2001).
  - A Practical Guide to Pseudospectral Methods, Bengt Fornberg (1998).
- Advantages of Chebyshev Polynomials:
  - Numerous simple close-form expressions are available.
  - They are more robust than their alternatives for interpolation.
  - They are bounded between $[-1, 1]$.
  - They are smooth functions.
  - The move between the coefficients of a Chebyshev expansion of a function and the values of the function at the Chebyshev nodes quickly performed by the cosine transform.
  - Several theorems bound the errors for Chebyshev polynomials interpolations.

Chebyshev polynomials

- Recursive definition:

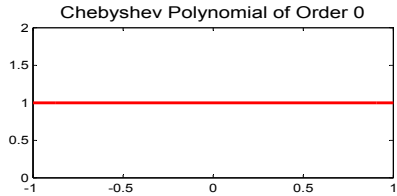
$$T_0(x) = 1$$
$$T_1(x) = x$$
$$...$$
$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

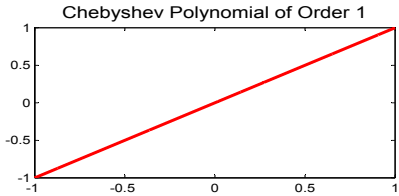
- The first few polynomials are then $1, x, 2x^2 - 1, 4x^3 - 3x, 8x^4 - 8x^2 + 1$, etc...
- The n zeros of the polynominal $T_n(x_k) = 0$ are given by

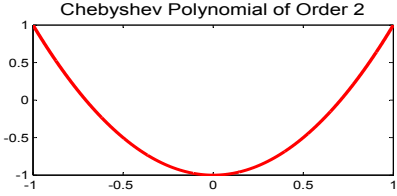
$$x_k = \cos \frac{2k-1}{2n}\pi, k = 1, ..., n$$

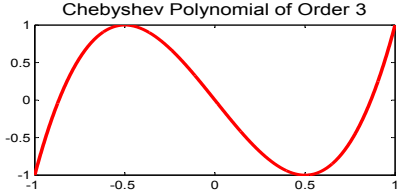
- Notice that this zeros are clustered quadratically towards ± 1.

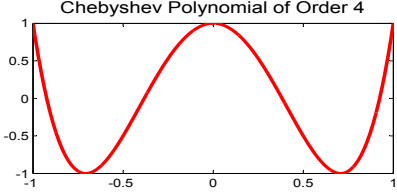
Chebyshev polynomials

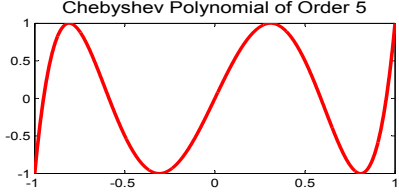












Chebyshev polynomials

$$T_n(x) = \begin{cases} \cos\left(n \arccos x\right) & \text{for } |x| \leq 1 \\ \frac{\left(x-\sqrt{x^2-1}\right)^n + \left(x+\sqrt{x^2-1}\right)^n}{2} & \text{for } |x| \geq 1 \end{cases}$$

$$= \begin{cases} \cos\left(n \arccos x\right) & \text{for } -1 \leq x \leq 1 \\ \cosh\left(n \operatorname{arcosh} x\right) & \text{for } 1 \leq x \\ (-1)^n \cosh\left(n \operatorname{arcosh}(-x)\right) & \text{for } x \leq -1 \end{cases}$$

$$T_n(x) = n \sum_{k=0}^{n} (-2)^k \frac{(n+k-1)!}{(n-k)!(2k)!} (1-x)^k, \text{for } n > 0$$

Chebyshev polynomials

- The domain of the Chebyshev polynomials is $[-1, 1]$. Since our state space is, in general, different, we use a linear translation from $[a, b]$ into $[-1, 1]$:

$$2\frac{x-a}{b-a} - 1$$

- Chebyshev polynomials are orthogonal with respect to the weight function:

$$\frac{1}{\sqrt{1-x^2}}$$

- Chebyshev Interpolation Theorem: If an approximating function is exact at the roots of the nth order Chebyshev polynomial then, as $n \to \infty$, the approximation error becomes arbitrarily small.

Multidimensional problems

- Chebyshev polynomials are defined on $[-1, 1]$.
- However, most problems in economics are multidimensional.
- How do we generalize the basis?
- Curse of dimensionality.

Tensors and Chebyshev Polynomial Approximation

- **Goal:** Approximate a function $F : [-1, 1]^d \to \mathbb{R}$ using **Chebyshev polynomials**.
- The **Chebyshev polynomial** of degree j, denoted $T_j(x)$, is defined recursively:

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{j+1}(x) = 2xT_j(x) - T_{j-1}(x).$$

- We approximate $F(x)$ using a **tensor product of Chebyshev polynomials** of degree κ:

$$\hat{F}(x) = \sum_{n_1=0}^{\kappa} \cdots \sum_{n_d=0}^{\kappa} \theta_{n_1,\ldots,n_d} T_{n_1}(x_1) \cdots T_{n_d}(x_d).$$

- **Key Property:** If the **one-dimensional Chebyshev basis** is orthogonal under a given norm, the **tensor product basis remains orthogonal** under the corresponding **product norm**.

Curse of Dimensionality in Tensor Approximation

- The number of basis terms **grows exponentially** with the number of dimensions d.
- At maximum polynomial degree κ, we must consider terms of the form:

$$x_1^{\kappa} x_2^{\kappa} \cdots x_d^{\kappa}.$$

- **Total number of terms:**

$$(\kappa + 1)^d.$$

- This exponential growth leads to **computational challenges**, making high-dimensional function approximation infeasible without efficient techniques such as:
  - **Sparse grid methods**
  - **Low-rank tensor decompositions**
  - **Adaptive basis selection**

Complete Homogeneous Symmetric Polynomials

- **Challenge:** The tensor approximation includes too many terms, leading to high computational costs.
- **Solution:** Reduce the number of elements in the tensor while maintaining accuracy and avoiding significant numerical degradation.
- **Judd and Gaspar (1997) Approach:** Use **complete homogeneous symmetric polynomials**, defined as:

$$P_k^d \equiv \left\{ x_1^{i_1} \cdots x_d^{i_d} \mid \sum_{j=1}^{d} i_j \leq k, \quad 0 \leq i_1, \ldots, i_d \right\}.$$

Complete homogeneous symmetric polynomials

- **Intuition:**
  - Instead of allowing each variable to have an independent degree up to κ, we restrict the sum of exponents across all variables to be at most k.
  - This ensures that only **low-complexity interactions** between variables are retained while eliminating many high-order terms.
  - The approach is similar to selecting only smoother, lower-order interactions that contribute most to function approximation.
- **Advantage:**
  - The number of terms is **much smaller** compared to the full tensor expansion.
  - Avoids evaluating terms of order $d\kappa$, significantly reducing computational cost.
- **Disadvantage:**
  - The number of elements is still large, especially for high d and k.

Smolyak's Algorithm

Smolyak's algorithm

- Smolyak's algorithm is a method used for the approximation of multivariate functions. It is an adaptive sparse grid interpolation method. The main goal of the algorithm is to approximate a function of multiple variables with a manageable amount of computational resources.
- The standard method of approximating a function of d variables would be to create a grid with N points in each dimension, resulting in N^d total points. As the dimensionality (d) increases, the number of grid points increases exponentially, leading to a "curse of dimensionality".
- Smolyak's algorithm alleviates this problem by creating a sparse grid, which reduces the number of points required for the approximation. The idea is to select a subset of the full grid that balances accuracy and computational tractability.

Smolyak's algorithm

- Here are the main steps of the Smolyak algorithm:
  1. Choose univariate basis functions and associated quadrature nodes and weights.
  2. Define the level of approximation.
  3. Compute the so-called Smolyak grid, which is a subset of the full tensor product grid.
  4. Compute the function values at the points of the Smolyak grid.
  5. Based on these function values, compute the coefficients for the basis functions using the Smolyak formula.

Smolyak Algorithm: Constructing the Grid (Overview)

- **Goal:** Build a **sparse grid** in d dimensions that captures most of the function's behavior *without* using the full tensor product.
- **Key Idea:**
  - Use a **1D quadrature rule** (e.g., Gauss–Legendre, Clenshaw–Curtis) at multiple **levels** of refinement.
  - In each dimension, refine from a **coarse** to a **finer** set of points as needed.
  - Combine these 1D rules to form a **sparse multi-dimensional grid**.
- **Smolyak Principle:**
  - Rather than taking the **full tensor product** of all fine-level grids (which grows exponentially),
  - **Selectively mix** coarse levels in some dimensions with fine levels in others, ensuring the total "level" does not exceed a chosen threshold.
- **Why It Works:**
  - Many high-dimensional functions vary more significantly in certain directions or regions.
  - The sparse grid **adapts** to these variations, avoiding unnecessary points in less significant directions.
  - Reduces **computational complexity** dramatically compared to a full grid.

Smolyak Algorithm: Detailed Construction

1. **Choose 1D Quadrature Rules:**
   - For each level $\ell = 1, 2, \ldots, L$, pick a **1D node set** X_ℓ (e.g., Gauss–Legendre nodes).
   - The size of X_ℓ grows with ℓ (more points $\rightarrow$ finer approximation).
2. **Define the Level of Approximation $A(L, d)$:**
   - Let $(\ell_1, \ell_2, \ldots, \ell_d)$ be a **multi-index** where ℓ_i denotes the 1D approximation level in dimension i.
   - Only consider combinations where
   $$\ell_1 + \ell_2 + \cdots + \ell_d \leq L + (d - 1).$$
   - This **restricts** how many dimensions can be at a high level simultaneously.
3. **Construct the Sparse Grid $S(L, d)$:**
   - For each valid multi-index $(\ell_1, \ldots, \ell_d)$, form the **tensor product** of the 1D grids:
   $$X_{\ell_1} \times X_{\ell_2} \times \cdots \times X_{\ell_d}.$$
   - **Combine** these tensor products (merge and remove duplicates) to get the **sparse grid points**.
4. **Why This Saves Points:**
   - A **full grid** at level L in each dimension has $|X_L|^d$ points $\rightarrow$ **exponential in** d.
   - **Smolyak grid** uses only "partial refinement" across dimensions $\rightarrow$ **far fewer points**.
   - Often, **similar accuracy** but at a **fraction of the cost**.

What is a Smolyak "Level"? (1D Case)

- **1D Quadrature Rules:**
  - Think of a sequence of **increasingly refined** grids on $[-1, 1]$ (or another interval).
  - $\ell = 1, 2, 3, \ldots$ denotes the **level of refinement**.
  - Example (Clenshaw–Curtis):
    - Level 1 might have 2 nodes (endpoints).
    - Level 2 might have 3 nodes.
    - Level 3 might have 5 nodes, etc.
- **Interpretation of "Level" in 1D:**
  - Higher ℓ = more points = **finer approximation**.
  - $\ell = 1$ is very coarse, $\ell = 4$ is more precise, etc.

From 1D to d Dimensions — The Full Grid

- In d dimensions, a **full tensor grid** at level ℓ in each dimension takes the 1D grid of level ℓ and forms the **Cartesian product**:
$$X_\ell^{(1)} \times X_\ell^{(2)} \times \cdots \times X_\ell^{(d)}.$$
- **Number of points:**
$$\big(|X_\ell|\big)^d.$$
- **Problem:** Exponential growth in d. For large d, the number of points grows as:
$$(\kappa + 1)^d,$$
which quickly becomes impractical (**curse of dimensionality**).

Sparse Grid Intuition: Mixing Different Levels

- **Key Trick of Smolyak:**
  - Not every dimension needs to be at the highest refinement simultaneously.
  - We can have dimension 1 at level 3, dimension 2 at level 1, dimension 3 at level 2, etc.
- **Why This Helps:**
  - Many functions in high-dimension **don't vary equally** in each direction.
  - **Fewer points** overall if we only refine heavily where needed.
- **Multi-Index $\ell_1, \ell_2, \ldots, \ell_d$:**
  - Each ℓ_i is the 1D level in dimension i.
  - We'll **combine** these to form the final sparse grid.

Why the Inequality $\ell_1 + \ldots + \ell_d \leq L + (d-1)$?

- **Defining a Global "Level" L:**
  - L sets the **overall accuracy** we desire.
  - We allow each dimension's ℓ_i to vary from 1 up to L, but **not all** can be maxed at once.
- **Sum of Levels Restriction:**
  - $\ell_1 + \ell_2 + \cdots + \ell_d \ \leq \ L + (d-1)$.
  - This ensures the "combined refinement" across all dimensions **doesn't exceed** a certain threshold.
  - We basically **skip** combinations where every dimension is at a high level simultaneously.
- **Comparing to a Full Grid:**
  - **Full Grid:** $\ell_1 = \ell_2 = \cdots = \ell_d = L$. We get $(n_L)^d$ points.
  - **Smolyak Grid:** We only use certain $(\ell_1, \ell_2, \ldots, \ell_d)$ that satisfy the sum constraint. **Fewer combos** $\rightarrow$ **far fewer points**.

Intuitive Understanding of Smolyak and Its Accuracy

- **Selective Refinement:**
  - Imagine painting a landscape—only zoom in to add details where needed.
  - Smolyak refines some dimensions more than others, focusing on regions where the function varies the most.
- **Smart Grid Combination:**
  - Instead of building a dense grid over all dimensions, mix grids of different refinement levels.
  - This mix acts like assembling a puzzle—each piece adds detail without overwhelming the overall picture.
- **Balancing Efficiency and Accuracy:**
  - Fewer points are used, reducing computational cost while maintaining high-order accuracy.
  - The algorithm's design ensures that even with less data, essential features of the function are captured reliably.

2D Smolyak Example: Levels in Each Dimension

- **Goal:** Approximate a function $F(x_1, x_2)$ over a 2D domain, e.g. $[-1, 1] \times [-1, 1]$.
- **1D Grids: Levels 1 through L:**
  - For each dimension, we define *multiple* 1D grids of increasing **level** ℓ:

    $$X_1, \quad X_2, \quad \ldots, \quad X_L,$$

    where X_ℓ is the set of nodes at level ℓ.
  - Example (Clenshaw–Curtis):
    - $\ell = 1$: 2 nodes (coarse).
    - $\ell = 2$: 3 nodes (finer).
    - $\ell = 3$: 5 nodes, and so on.
- **Dimensional Perspective:**
  - Dimension 1 can be at level $\ell_1 \in \{1, \ldots, L\}$.
  - Dimension 2 can be at level $\ell_2 \in \{1, \ldots, L\}$.

2D Smolyak Construction: Tensor Products

- **Full Grid (for comparison):**
  - If we used a **single** 1D grid of level L in both dimensions, we get
    $$X_L^{(1)} \times X_L^{(2)} \quad \text{(Cartesian product)}.$$
  - #points $= (|X_L|)^2$.
  - **Problem:** For higher d, $(|X_L|)^d \Rightarrow$ exponential growth.
- **Sparse Grid (Smolyak):**
  - Consider *all* 1D levels $\ell_1 \in \{1, \ldots, L\}$ in dimension 1, and $\ell_2 \in \{1, \ldots, L\}$ in dimension 2.
  - Only **keep** pairs (ℓ_1, ℓ_2) where
    $$\ell_1 + \ell_2 \;\leq\; L \;+\; (2-1),$$
    i.e. $\ell_1 + \ell_2 \leq L + 1$.
  - For each **valid pair**, form the **tensor product grid**
    $$X_{\ell_1}^{(1)} \times X_{\ell_2}^{(2)},$$
    then **union** them to get the *sparse grid*.
- **Result:**
  - We get a **smaller set of points** than a single full grid of level L.
  - Accuracy $\approx$ full grid in many cases, but total points $\ll (|X_L|)^2$.

2D Visual Example (Schematic)

- **Full Grid (Level L):**
  - Typically looks like a dense $(|X_L| \times |X_L|)$ lattice across the 2D domain.
- **Sparse Grid (Smolyak):**
  - **Union** of multiple small grids:
$$\bigcup_{\substack{\ell_1=1,\dots,L \\ \ell_2=1,\dots,L \\ \ell_1+\ell_2 \leq L+1}} \left(X_{\ell_1}^{(1)} \times X_{\ell_2}^{(2)} \right).$$
  - Each smaller grid has $|X_{\ell_1}^{(1)}| \cdot |X_{\ell_2}^{(2)}|$ points.
  - Merged total is still **far fewer** than $(|X_L|)^2$.
- **Intuitive Benefit:**
  - Some pairs (ℓ_1, ℓ_2) might be $(1,3), (2,2), (3,1)$, etc.
  - We *cover* the domain **more adaptively**, refining certain axes more than others.

From 2D to n-Dimensions

- **In n-dimensions:**
  - Each dimension i has 1D grids $X_\ell^{(i)}$ for $\ell = 1, \ldots, L$.
  - We form multi-indices $(\ell_1, \ldots, \ell_n)$ with $\ell_i \in \{1, \ldots, L\}$.
- **The Smolyak Constraint:**

$$\ell_1 + \ell_2 + \cdots + \ell_n \;\; \leq \;\; L \; + \; (n-1).$$

  - Limits total "refinement budget" across all n dims.
  - Skips combos that refine every dimension to max level simultaneously.
- **Sparse Grid Construction:**

$$S(L,n) = \bigcup_{\substack{\ell_1=1,\ldots,L \\ \cdots \\ \ell_n=1,\ldots,L \\ \ell_1+\cdots+\ell_n \leq L+(n-1)}} \left(X_{\ell_1}^{(1)} \times \cdots \times X_{\ell_n}^{(n)} \right).$$

- **Comparison to Full Grid:**
  - **Full Grid (Level L)**: $\#\text{points} = (|X_L|)^n$.
  - **Smolyak Sparse Grid**: $\#\text{points} \approx O\big(|X_L| \cdot (\log |X_L|)^{n-1}\big)$ (roughly), much **smaller** for large n.

Why the Smolyak Construction Works

- **Adaptive Refinement Across Dimensions:**
  - Some dimensions are more "important" (function varies faster); Smolyak automatically refines those more.
  - Dimensions that are less significant stay at lower levels $\rightarrow$ fewer points.
- **Avoids Full-Grid Explosion:**
  - Full grid: exponential growth in points.
  - Sparse grid: partial combinations $\rightarrow$ **polynomial growth**.
- **Mathematical Guarantee:**
  - Smolyak **approximation error** is close to that of a full grid (spectral or polynomial accuracy) in many practical cases.
  - Often used for **high-dimensional integration and function approximation**.
- **Practical Implementation:**
  - We systematically build 1D rules from levels 1 to L.
  - Combine these rules according to the sum constraint $\ell_1 + \cdots + \ell_d \leq L + d - 1$.
  - Merge points (removing duplicates) to form the final **sparse grid**.

Smolyak's algorithm

- The final approximating function is then a linear combination of the basis functions, where the coefficients are determined by the Smolyak algorithm. By intelligently choosing which points to include in the approximation, the Smolyak algorithm can significantly reduce the computational burden of high-dimensional problems, while still maintaining a reasonable level of accuracy.
- The choice of the level of approximation, the basis functions, and their associated quadrature nodes and weights can significantly affect the accuracy and efficiency of the Smolyak algorithm.

Smolyak's algorithm

- Malin, B., Krueger, D., Kubler, F., 2011. "Solving the multi-country real business cycle model using a Smolyak-collocation method." J. Econ. Dyn. Control 35, 229–239.
- Define $m_1 = 1$ and $m_i = 2^{i-1} + 1$, $i = 2, ...$
- Define $G^i = \left\{x_1^i, ..., x_{m_i}^i\right\} \subset [-1, 1]$ as the set of the extrema of the Chebyshev polynomials

$$x_j^i = -\cos\left(\frac{\pi(j-1)}{m_i - 1}\right), \ j = 1, ..., m_i$$

with $G^1 = \{0\}$, and $G^i \subset G^{i+1}$,$\forall i = 1, 2, ...$ Note the extrema is defined as $\frac{dT_n(x)}{dx} = 0$.

- Example:

$$\begin{aligned}
&i = 1, m_1 = 1, G^1 = \{0\} \\
&i = 2, m_2 = 3, G^2 = \{-1, 0, 1\} \\
&i = 3, m_3 = 5, G^3 = \left\{-1, -\cos\left(\frac{\pi}{4}\right), 0, -\cos\left(\frac{3\pi}{4}\right), 1\right\}
\end{aligned}$$

Smolyak's algorithm

- For $q > d$, where d is the dimention of the problem. Define a sparse grid

$$\Phi(q,d) = \bigcup_{q-d+1 \leq |i| \leq q} \left(G^{i_1} \times ... \times G^{i_d}\right)$$

  where $|i| = i_1 + ... + i_d$.
- The number q defines the size of the grid and thus the precision of the approximation.

Smolyak's algorithm

- Example: $d = 3$, $\mu = 2$, and $q = d + \mu$

$$\Phi(q,d) = \Phi(5,2) = \underset{d \le |i| \le q}{\cup} \left(G^{i_1} \times ... \times G^{i_d} \right)$$

$$\begin{aligned} G^1 \times G^1 \times G^1, G^1 \times G^2 \times G^1, G^1 \times G^1 \times G^2 \\ G^1 \times G^3 \times G^1, G^1 \times G^1 \times G^3, G^1 \times G^2 \times G^2 \\ G^2 \times G^1 \times G^1, G^2 \times G^2 \times G^1, G^2 \times G^1 \times G^2 \\ G^3 \times G^1 \times G^1 \end{aligned}$$

- For $q > d$, we can approximate the smooth functions $f \in C^k([0,1]^d)$ as follows

$$\hat{f}(q,d)(x) = \sum_{d \le |i| \le q} (-1)^{q-|i|} \begin{pmatrix} d-1 \\ q-|i| \end{pmatrix} \left(U^{i_1} \bigotimes ... \bigotimes U^{i_d} \right)(x)$$

where $U^i = \sum_{j=1}^{m_i} \theta_j^i T_j(x^i)$.

Smolyak's algorithm

Tensor products of disjoint sets of unidimensional grid points for the two-dimensional case.

	$S_{i_1} \mid S_{i_2}$	$i_2 = 1$	$i_2 = 2$	$i_2 = 3$
		0	$0, -1, 1$	$0, -1, 1, \frac{-1}{\sqrt{2}}, \frac{1}{\sqrt{2}}$
$i_1 = 1$	0	$(0,0)$	$(0,0),(0,-1),(0,1)$	$(0,0),(0,-1),(0,1),\left(0,\frac{-1}{\sqrt{2}}\right),\left(0,\frac{1}{\sqrt{2}}\right)$
$i_1 = 2$	0 -1 1	$(0,0)$ $(-1,0)$ $(1,0)$	$(0,0),(0,-1),(0,1)$ $(-1,0),(-1,-1),(-1,1)$ $(1,0,),(1,-1),(1,1)$	$(0,0),(0,-1),(0,1),\left(0,\frac{-1}{\sqrt{2}}\right),\left(0,\frac{1}{\sqrt{2}}\right)$ $(-1,0),(-1,-1),(-1,1),\left(-1,\frac{-1}{\sqrt{2}}\right),\left(-1,\frac{1}{\sqrt{2}}\right)$ $(1,0),(1,-1),(1,1),\left(1,\frac{-1}{\sqrt{2}}\right),\left(1,\frac{1}{\sqrt{2}}\right)$
$i_1 = 3$	0 -1 1 $\frac{-1}{\sqrt{2}}$ $\frac{1}{\sqrt{2}}$	$(0,0)$ $(-1,0)$ $(1,0)$ $\left(\frac{-1}{\sqrt{2}},0\right)$ $\left(\frac{1}{\sqrt{2}},0\right)$	$(0,0),(0,-1),(0,1)$ $(-1,0),(-1,-1),(-1,1)$ $(1,0),(1,-1),(1,1)$ $\left(\frac{-1}{\sqrt{2}},0\right),\left(\frac{-1}{\sqrt{2}},-1\right),\left(\frac{-1}{\sqrt{2}},1\right)$ $\left(\frac{1}{\sqrt{2}},0\right),\left(\frac{1}{\sqrt{2}},-1\right),\left(\frac{1}{\sqrt{2}},1\right)$	$(0,0),(0,-1),(,1),\left(0,\frac{-1}{\sqrt{2}}\right),\left(0,\frac{1}{\sqrt{2}}\right)$ $(-1,0),(-1,-1),(-1,1),\left(-1,\frac{-1}{\sqrt{2}}\right),\left(-1,\frac{1}{\sqrt{2}}\right)$ $(1,0),(1,-1),(1,1),\left(1,\frac{-1}{\sqrt{2}}\right),\left(1,\frac{1}{\sqrt{2}}\right)$ $\left(\frac{-1}{\sqrt{2}},0\right),\left(\frac{-1}{\sqrt{2}},-1\right),\left(\frac{-1}{\sqrt{2}},1\right),\left(\frac{-1}{\sqrt{2}},\frac{-1}{\sqrt{2}}\right),\left(\frac{-1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)$ $\left(\frac{1}{\sqrt{2}},0\right),\left(\frac{1}{\sqrt{2}},-1\right),\left(\frac{1}{\sqrt{2}},1\right),\left(\frac{1}{\sqrt{2}},\frac{-1}{\sqrt{2}}\right),\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)$

Smolyak's algorithm

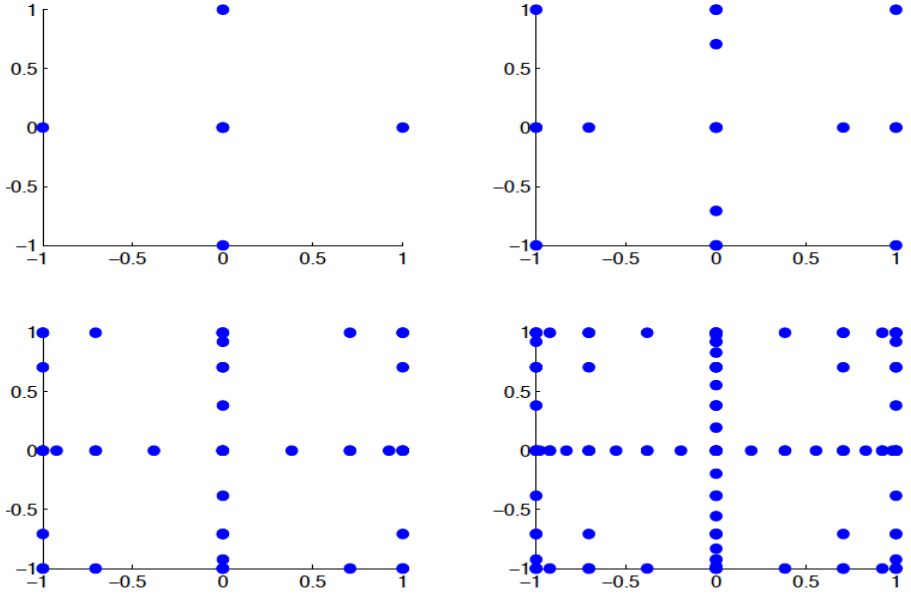
- If $\mu = 0$, then $2 \leq i_1 + i_2 \leq 2$. The only cell that satisfies this restriction is $i_1 = 1$ and $i_2 = 1$, so that the Smolyak grid has just one grid point.

$$\Phi(2,2) = \{(0,0)\}$$

- If $\mu = 1$, then $2 \leq i_1 + i_2 \leq 3$. There are three cell that satisfies this restriction:
  - $i_1 = 1$ and $i_2 = 1$;
  - $i_1 = 1$ and $i_2 = 2$;
  - $i_1 = 2$ and $i_2 = 1$
  - so that the Smolyak grid has five grid points.

$$\Phi(3,2) = \{(0,0), (-1,0), (1,0), (0,-1), (0,1)\}$$

Smolyak's algorithm



Smolyak's algorithm

Size of the grid for $q = d + 2$			
d	$2^{q-d} + 1$	points in $\Phi(q, d)$	$\left(2^{q-d} + 1\right)^d$
2	5	13	25
4	5	41	625
5	5	61	3125
12	5	313	244,140,625

Smolyak's algorithm

- To compute $\hat{f}(q,d)(x)$ one only needs to know function values at the "sparse grid"

$$\Phi(q,d) = \bigcup_{d \leq |i| \leq q} \left(G^{i_1} \times ... \times G^{i_d} \right)$$

- For the tensor product norm

$$||h||_k = \max \left\{ ||D^\alpha h||_{L_\infty} : \alpha \in \mathbb{N}_0^d, \alpha_i \leq k \right\}$$

the numerical error of $\hat{f}(q,d)(x)$ is bounded by

$$C_d n^{-k} \left(\log n \right)^{(d-1)(k+1)} ||f||_k$$

where n is the number of points in $\Phi(q,d)$.

Finite Elements Method

Finite Element Method (FEM)

- **Use Cases:**
  - **Complex Geometries** or boundaries in the model's domain.
  - State variables in economics or engineering with irregular shapes.
- **Discretization:**
  - Partition the domain (state space) into smaller **elements** (e.g., triangles, rectangles, tetrahedrons).
  - This **mesh** can adapt to curves, corners, or other complexities.
- **Basis Functions:**
  - On each element, choose **piecewise polynomial** basis functions.
  - Typically, these functions are **1 at a specific node** and **0 at all other nodes**, creating a **partition of unity**.
- **Formulation of System:**
  - Replace PDE derivatives with **finite-element approximations**.
  - Integrate (often using **integration by parts**) over each element.
  - Assemble the resulting equations into a **global algebraic system**.
- **Solution:**
  - Solve the system for **unknown coefficients** at each node.
  - Yields a **piecewise polynomial** approximation of the original PDE solution.

Finite Elements vs. Smolyak (I)

- **Basis Functions & Interpolation**
  - **FEM**: Uses **piecewise polynomial** basis defined on local elements.
    - The solution can differ **element-by-element**.
  - **Smolyak**: Generally uses **global** basis functions over the entire domain (e.g., **Chebyshev** polynomials or **splines**).
    - Relies on **sparse grid interpolation**, reducing the number of nodes.
- **Discretization & Grid**
  - **FEM**: Domain is divided into a **mesh** of elements (triangles, tetrahedrons, etc.).
    - Handles **irregular boundaries** very well.
    - Locally adaptive elements possible.
  - **Smolyak**: Constructs a **multi-dimensional grid** from a **tensor product** of 1D grids.
    - Uses **sparse grids** to avoid the curse of dimensionality.
    - May be less natural for highly **irregular domains**.

Finite Elements vs. Smolyak (II)

- **Computation & Accuracy**
  - **FEM**:
    - Very **efficient** for **complex geometries** or boundary conditions.
    - Widely used in **engineering and applied sciences**.
    - Can become **computationally heavy** in **high dimensions**.
  - **Smolyak**:
    - Designed to address the **curse of dimensionality** via **sparse grids**.
    - Highly **accurate** for many **high-dimensional** problems.
    - Less specialized for complex boundary shapes.
- **Key Trade-offs**
  - **FEM**:
    - Excellent for **low- to moderate-dimensional** PDEs with **irregular domains**.
    - Local **element-based** approximation.
  - **Smolyak**:
    - Optimal for **moderate- to high-dimensional** problems.
    - **Global** approximation with drastically **fewer grid points**.

Structure

- Choose a basis for the policy functions in each element.
- Since the elements are small, a linear basis is often good enough:

$$g_i(k) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & \text{if } x \in [x_{i-1}, x_i] \\ \frac{x_{i-1} - x}{x_i - x_{i-1}} & if x \in [x_i, x_{i+1}] \\ 0 & elsewhere \end{cases}$$

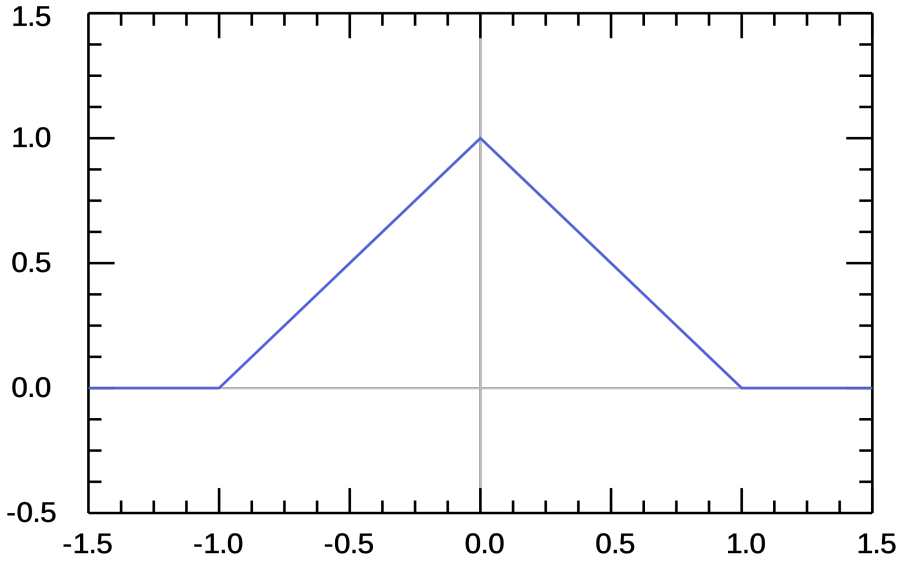
- Plug the policy function in the Equilibrium Conditions and find the unknown coefficients.
- Paste it together to ensure continuity.
- Advantages: we will need to invert an sparse matrix.
- When should be choose this strategy? speed of computation versus accuracy.

Why Does This Lead to a Sparse Matrix?

- **Local Support of Basis Functions:**
  - The chosen **linear basis functions** $g_i(x)$ are **piecewise-defined** and have **compact support**.
  - Each basis function is **nonzero only over a small interval** $[x_{i-1}, x_{i+1}]$ and **zero elsewhere**.
- **Sparse System of Equations:**
  - Substituting the policy function into the equilibrium conditions leads to a system of equations:

$$A\theta = b,$$

    where:
    - A is the **coefficient matrix** determined by the basis functions.
    - θ is the **vector of unknown coefficients**.
    - b is the **right-hand side vector** from the equilibrium conditions.
  - Due to the **local nature** of $g_i(x)$, each row in A **only involves a few neighboring coefficients**, making most entries **zero**.


1.5
1.0
0.5
0.0
-0.5
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5

Structure

Three different refinements

- h-refinement: subdivide each element into smaller elements to improve resolution uniformly over the domain.
- r-refinement: subdivide each element only in those regions where there are high nonlinearities.
- p-refinement: increase the order of the approximation in each element. If the order of the expansion is high enough, we will generate in that way an hybrid of finite and spectral methods knows as spectral elements.

To Determine the Coefficients for Basis Functions

Objective function

- Two issues:
  1. Which basis we use?
  2. How do we "project"?
- The most common answer to the second question is given by a weighted residual.
- That is why often projection methods are also called weighted residual methods.
- This set of techniques propose to get the residual close to 0 in the weighted integral sense.
- Given some weight functions $f_i : \Omega \to \mathbb{R}^m$

$$\rho\left(R(\cdot|\theta), 0\right) = \begin{cases} 0 & \text{if } \int_\Omega f_i(x)\mathrm{R}(\cdot|\theta)\mathrm{dx} = 0, i = 0, ..., n \\ 1 & \text{otherwise} \end{cases}$$

- Then the problem is to choose the θ that solve the system of equations

$$\int_\Omega f_i(x) R(\cdot|\theta) dx = 0, i = 0, ..., n$$

Objective function

- With the approximation of d, that solves $H(d) = 0$, by some functions g_i and the definition of some weight functions f_i, we have transform a rather intractable functional equation problem into the standard nonlinear equations system!
- The solution of this system can be found using standard methods, as a Newton for relatively small problems or a conjugate gradient for bigger ones.
- Issue: we have different choices for a weight function.

Weight function

- The intuition behind including a weight function in the projection method is to address the issues that may arise due to the irregular behavior of the function you are approximating over the domain of interest.
- For instance, the function could be changing rapidly in one part of the domain and slowly in other parts.
- By including a weight function, you can give more emphasis to the parts of the domain where the function is changing rapidly (or has higher variation). This makes the approximation more accurate in these regions.
- The weight function helps to handle heteroskedasticity in the function that you are approximating.

Weight function

- Basis Function as the Weight:
  - When you use the basis function as the weight, the idea is to make the weight variable depending on the point of the function you are approximating.
  - The benefit of using the basis functions themselves as the weights is that these basis functions are likely to capture the variations in the function over the domain, especially if the basis functions are chosen wisely based on the characteristics of the function being approximated.
  - Choosing the basis functions as weights ensures that the regions where the function changes rapidly get more weight and hence the approximation error in these regions is minimized. This choice can be particularly useful when the function has some local features that are best captured by the basis functions.

Weight function I: Least Squares

- $f_i(x) = \frac{\partial R(x|\theta)}{\partial \theta_i}$.
- This choice is motivated by the solution of the variational problem:

$$\min_{\theta} \int_{\Omega} R^2(\cdot|\theta) dx$$

with first-order condition:

$$\int_{\Omega} \frac{\partial R(x|\theta)}{\partial \theta_i} R(\cdot|\theta) dx = 0, i = 0, ..., n$$

- Variational problem is mathematically equivalent to a standard regression problem in econometrics.
- Least Squares always generates symmetric matrices, which are convenient theoretically (they simplify the proofs) and computationally (there are algorithms that exploit their structure to increase speed and decrease memory requirements).
- However, least squares may lead to ill-conditioning and systems of equations complicated to solve numerically.

Weight Function II: Subdomain Approach

- **Idea:** Instead of enforcing the residual condition over the entire domain Ω, we split the domain into smaller subdomains and enforce it locally.
- We divide Ω into n subdomains Ω_i and define the step functions:

$$f_i(x) = \begin{cases} 1 & \text{if } x \in \Omega_i \\ 0 & \text{otherwise.} \end{cases}$$

- This choice simplifies the weighted residual condition, leading to a system of equations:

$$\int_{\Omega_i} R(\cdot|\theta) dx = 0, \quad i = 0, \ldots, n.$$

- **Intuition:** - Each subdomain acts independently, meaning we solve smaller, localized problems rather than one large system. - This reduces computational effort and improves accuracy in regions where the function has more variation.

Weight function III: Moments

- Take $\{0, x, x^2, ..., x^{n-1}\}$ and compute the first n periods of the residual function:

$$\int_{\Omega_i} x^i R(\cdot|\theta) dx = 0, i = 0, ..., n$$

- This approach, widely used in engineering works well for a low n (2 or 3).
- However, for higher orders, its numerical performance is very low: high orders of x are highly collinear and arise serious rounding error problems.
- Hence, moments are to be avoided as weight functions.

Weight function IV: Collocation or pseudospectral

- Let $f_i(x) = \delta(x - x_i)$, where δ is the dirac delta function and x_i are the collocation points.
- This method implies that the residual function is zero at the n collocation points.
- Simple to compute since the integral only needs to be evaluated in one point. Specially attractive when dealing with strong nonlinearities.
- A systematic way to pick collocation points is to use a density function:

$$\mu_\gamma(x) = \frac{\Gamma(\frac{3}{2} - \gamma)\gamma}{(1 - x^2)^\gamma \sqrt{\pi}\Gamma(1 - \gamma)}$$

  and find the collocation points x_j as the solution to

$$\int_{-1}^{x_j} \mu_\gamma(x)dx = \frac{j}{n}$$

- For $\gamma = 0$, the density function implies equi-spaced points.

Intuition: $\mu_\gamma(x)$ and the Case $\gamma = 0$

- **Goal:** Systematically pick collocation points $\{x_j\}$ to optimize function approximation.
- We use a **density function** $\mu_\gamma(x)$ to **distribute** these points.
- **Effect of γ on Distribution:**
  - $\mu_\gamma(x) = \dfrac{\Gamma\left(\frac{3}{2} - \gamma\right)\gamma}{(1-x^2)^\gamma \sqrt{\pi}\,\Gamma(1-\gamma)}$, for $\gamma > 0$.
  - $(1-x^2)^{-\gamma} \Longrightarrow$ **more collocation points** near boundaries ($x = \pm 1$), ideal for **sharp edges**.
- $\gamma = 0$ **Case:**
  - Substituting $\gamma = 0$ into $\mu_\gamma(x)$ yields a **zero** function, not a valid PDF.
  - Hence, $\gamma = 0$ implies **equi-spaced** (uniform) collocation points, chosen **manually**.
- **Key Takeaways:**
  - $\gamma > 0$: Points cluster near ± 1.
  - $\gamma = 0$: Uniform spacing by a direct choice.
  - **Adaptive selection** of γ can improve accuracy in difficult regions.

Dirac Delta Function δ and Comparison

- **Dirac Delta $\delta(x - x_i)$:**
  - **Generalized function**, zero everywhere except at x_i.
  - Satisfies $\int_{-\infty}^{\infty} \delta(x - x_i)\, dx = 1$.
  - In **collocation (pseudospectral)** methods, used to enforce $R(x_i; \theta) = 0$.
- **Collocation vs. Subdomain**

Aspect	**Collocation**	**Subdomain**
Weight	$\delta(x - x_i)$	Step fn. $f_i(x)$
Residual	Zero at **pts.**	Zero over **subdoms.**
Eval.	**Pointwise** only	**Integrate** subdoms.
Cost	**Lower**	**Higher**
Best for	**Nonlinear**, spectral	**Smooth** PDE-like
Limit	Misses betw. pts.	Integration overhead

Weight function V: Orthogonal collocation

- Variation of the collocation method:
  - Basis functions are a set of orthogonal polynomials.
  - Collocation points given by the roots of the $n-th$ polynomial.
- Surprisingly good performance of orthogonal collocation methods.

Weight Function V: Orthogonal Collocation

- **What is Orthogonal Collocation?**
  - A variation of collocation where the **basis functions** are **orthogonal polynomials** (e.g., Legendre, Chebyshev).
  - **Collocation points** are chosen as the **roots (zeros) of the n-th order polynomial** in the basis.
- **How It Works:**
  1. Select an **orthogonal polynomial family** (e.g., Legendre, Chebyshev).
  2. Compute the **roots** of the n-th polynomial.
  3. Use these roots as **collocation points** to enforce residual conditions.
- **Why Use Polynomial Roots?**
  - They distribute collocation points optimally to minimize interpolation errors.
  - More points are placed near the endpoints, capturing rapid changes.
  - Avoids numerical instabilities common in equi-spaced collocation.
- **Advantages:**
  - **Higher accuracy**—converges exponentially for smooth functions.
  - **Efficient for solving differential equations** (spectral methods).
  - **Stable approximation**—avoids Runge's phenomenon.

Weight function VI: Galerkin or Rayleigh-Ritz

- Define the weight function such that the residual is orthogonal to each of the basis functions.

$$\int_{\Omega} \phi_i(x) H \left(\sum_{i=0}^{n} \theta_i \phi_i(x) \right) dx = 0, i = 0, ..., n$$

- Galerkin is a highly accurate and robust but difficult to code.
- If the basis functions are complete over *the space of bounded measurable functions on a compact set*, then the Galerkin solution will converge pointwise to the true solution as $n \to \infty$:

$$\lim_{n \to \infty} \sum_{i=0}^{n} \theta_i \phi_i(x) = d(x)$$

- Experience suggests that a Galerkin approximation of order n is as accurate as a Pseudospectral $n+1$ or $n+2$ expansion.

Projection vs Neural Networks

The relationship between Projection Methods such as the Finite Element Method (FEM) and Neural Networks lies primarily in the general area of approximation theory. Both techniques are fundamentally ways of approximating complex functions or systems with simpler, more tractable representations.

- Function Approximation: Both neural networks and FEM aim to approximate complex functions. In FEM, the complex function (often a PDE) is approximated by piecewise polynomials. In contrast, a neural network approximates a function through multiple layers of nonlinear transformations. The fundamental idea is to capture the underlying pattern or mapping in the data.
- Optimization: Both approaches involve some form of optimization. In FEM, the coefficients of the polynomial basis functions are chosen to minimize the residual error in the equations. In neural networks, the weights and biases are updated iteratively to minimize a loss function that represents the discrepancy between the network's outputs and the true values.

Projection vs Neural Networks

- Representation Learning: Both methods can be seen as forms of representation learning. The objective of FEM is to represent a complex system (usually a continuous physical system) in terms of simpler elements (finite elements). Neural networks, especially deep learning, are also about representation learning, as they transform the raw input data through multiple layers of computation, thereby creating a hierarchy of learned features.
- Nonlinearity: Both methods are capable of dealing with nonlinearities. FEM can handle nonlinear PDEs by using nonlinear basis functions or by linearizing the equations through methods like the Newton-Raphson method. Neural networks inherently handle nonlinearities through the use of activation functions.

Projection vs Neural Networks

While both neural networks and projection methods like FEM aim to approximate complex functions, there are several important differences between the two, and in some cases, neural networks can offer distinct advantages:

- Learning from Data: The biggest advantage of neural networks is their ability to learn from data. This is particularly valuable in situations where the underlying mathematical model is unknown or very complex. In contrast, projection methods like FEM typically require a known mathematical model of the system under study.
- High-Dimensional & Complex Data: Neural networks excel at handling high-dimensional and complex data, such as images, audio, text, or large datasets with many features. This is a result of their hierarchical structure, where each layer of the network can learn increasingly complex features of the input data. FEM, on the other hand, can struggle with high-dimensional problems due to the so-called "curse of dimensionality".

Projection vs Neural Networks

- Nonlinearity: Both FEM and neural networks can handle nonlinearity, but neural networks do it more naturally via activation functions. FEM requires special treatment like the use of nonlinear basis functions or iterative linearization methods to handle nonlinear problems.
- Adaptive Learning: Neural networks can continually learn and update their parameters as new data is received. This makes them particularly useful in situations where the data changes over time, or where continual learning is needed. In contrast, FEM does not inherently have this adaptability; once the system is modeled and solved, changes require a new solution of the system.
- Universal Approximation: Neural networks have the property of being "universal approximators", meaning they can approximate any function given enough capacity (number of layers and neurons) and the right configuration. FEM approximations are tied to the specific choice of basis functions.

Basis Function vs Activation Function

- Basis Functions: In projection methods, basis functions are used to project a complex problem onto a simpler space where it can be solved more easily. These basis functions are usually fixed, well-defined mathematical functions like polynomials, and they are chosen beforehand based on the specific characteristics of the problem at hand (e.g., continuity requirements, boundary conditions). Basis functions are combined in a weighted sum to approximate the solution, with the weights often determined by solving a system of equations.

Basis Function vs Activation Function

- Activation Functions: In neural networks, activation functions are applied to the linear transformation of the input data at each neuron in the network. They introduce nonlinearity into the model, allowing the neural network to capture complex patterns in the data. Unlike basis functions, activation functions are not chosen to satisfy any particular physical criteria or boundary conditions, but rather are chosen based on their mathematical properties (like differentiability) and how well they work in practice for machine learning tasks. Common examples include the sigmoid, hyperbolic tangent, and ReLU functions.
- The connection between these two concepts lies in the fact that both are forms of basis functions in a broad sense: they transform the input space to potentially make a problem easier to solve. In neural networks, the layers of transformations can be thought of as learning an appropriate basis for representing the data. However, these are learned from the data during the training process and are not fixed beforehand.

Euler Equation Based Methods via DNN

Optimal Growth Model — Euler Equation

- Recall the **canonical optimal growth model** with capital k, consumption c, and discount factor β.
- The first-order (Euler) condition characterizing the optimum can be written:

$$u'(c) \;=\; \beta\left[u'(c')\big(1-\delta_k+f'(k')\big)\right],$$

  where

$$c = f(k) - k', \quad \text{and} \quad c' = f(k') - k''.$$

- In some cases, constraints (e.g., $k' \geq 0$) introduce *Kuhn–Tucker* conditions. One can convert the inequality to an equality using **Fischer–Burmeister** functions or, for more complex models, add **penalty terms** in the loss function.

Formulating the Euler Equation as a Learning Problem

- We want to find a **policy function** $g^k(\cdot)$ (e.g., next-period capital $k' = g^k(k)$) that satisfies the **Euler equation**:

$$u'(c_t) \;=\; \beta\left(1 - \delta_k + f'(k_{t+1})\right) u'(c_{t+1}), \quad \text{where } c_t = f(k_t) - g^k(k_t).$$

- We form an *expectation-based* loss function that penalizes deviations from the Euler equation:

$$\mathcal{L}(g^k) \;=\; \mathbb{E}_{k\sim d}\Big[\, u'\big(f(k) - g^k(k)\big) \;-\; \beta\, u'\big(f(k') - g^k(k')\big)\left(1 - \delta_k + f'(k')\right)\Big]^2.$$

  - Here $k' = g^k(k)$.
  - $d(k)$ represents the distribution of states (often the *stationary distribution* or a region of interest).

- In practice, we approximate this *expectation* by sampling states $\{k_i\}_{i=1}^N \sim d(k)$:

$$\mathcal{L}(g^k) \;\approx\; \frac{1}{N}\sum_{i=1}^{N}\Big[\, u'\big(f(k_i) - g^k(k_i)\big) \;-\; \beta\, u'\big(f(k_i') - g^k(k_i')\big)\left(1 - \delta_k + f'(k_i')\right)\Big]^2.$$

- Minimizing $\mathcal{L}(g^k)$ enforces that Euler equation residuals are *close to zero on average* (under the chosen distribution d).

Implementing the Loss Function $\mathcal{L}$

- $\mathcal{L}(g^k)$ measures the squared error of the Euler equation *residual*:

$$\text{residual}_i \;=\; u'\big(c_i\big) \;-\; \beta\, u'\big(c_i'\big)\big(1 - \delta_k + f'(k_i')\big).$$

- If there are *inequality constraints* (Kuhn–Tucker), we can:
  - Use the **Fischer–Burmeister function** to rewrite $\varphi(a, b) = a + b - \sqrt{a^2 + b^2}$ to handle $a \geq 0, b \geq 0, ab = 0$.
  - Or introduce **penalty terms** in $\mathcal{L}$ to enforce $k' \geq 0$, etc.
- The **gradient of** $\mathcal{L}$ w.r.t. the parameters of g^k is computed via backpropagation.
- We $\min_{g^k} \mathcal{L}(g^k)$ using $\gamma^{(j+1)} = \gamma^{(j)} - \alpha\, \nabla_\gamma \mathcal{L}(\gamma^{(j)})$.

Comparison with Collocation or Galerkin Methods

- **Collocation / Galerkin**:
  - Typically require choosing a *functional basis* (e.g., polynomials, Chebyshev polynomials) and *collocation points* or *weight functions*.
  - The Euler equation is enforced at these collocation points in a weighted sense (e.g., orthogonal projection).
- **DNN Approach**:
  - We approximate g^k with a flexible neural network Γ_γ *without* explicitly choosing a polynomial or spline basis.
  - Instead of a weight function for integration, we rely on *sampling* (often via Monte Carlo) to cover the relevant distribution of states.
  - The *loss function* is an *empirical* average of squared residuals, which implicitly gives uniform weight across sampled points (or can be extended to importance sampling if desired).
- **Key Difference**:
  - Collocation/Galerkin define a *weighted residual* in a functional space.
  - The DNN approach uses an *empirical average* of residuals over a sampled set of states. No explicit weight function is needed (unless we choose to incorporate one for emphasis).

Extensions and Future Directions

- **Penalty Methods for Complex Constraints**:
  - In heterogeneous-agent models or more intricate constraints, we can add penalty terms to the loss to handle inequality constraints without explicit projection or direct transformations.
  - This becomes crucial when we have, e.g., borrowing constraints, multiple asset classes, or piecewise-defined taxes.
- **Fischer–Burmeister vs. Penalty**:
  - Fischer–Burmeister is elegant for certain complementarity conditions, but can be less straightforward for large-scale problems.
  - Penalty methods are more general but may require careful tuning of penalty coefficients for stable training.
- **Broader Applicability**:
  - Once formulated as $\min \mathcal{L}(g^k)$, we can incorporate uncertainty, higher dimensions, or other state variables. The neural network's flexibility helps mitigate (though not fully eliminate) the curse of dimensionality.

Connection to Galerkin and Weighted Residual Methods

- **Projection methods** (Galerkin, Collocation, etc.) solve functional equations by enforcing that the *residual* (e.g., Euler equation deviation) is orthogonal to some function space or zero at selected points:
  - **Collocation:** Force the residual to be zero at chosen collocation points.
  - **Galerkin:** Require the residual to be orthogonal to a set of basis functions.
  - **Weighted Residual:** Require $\int w(x) \times \big[\text{residual}(x)\big]\, dx = 0$ for certain weighting functions $w(\cdot)$.
- In our **loss-based** approach:

$$\mathcal{L}(g^k) \;=\; \mathbb{E}_{k\sim d}\big[\text{residual}(k)\big]^2 \;=\; \int \big[\text{residual}(k)\big]^2 d(k)\, dk.$$

  - Minimizing this is akin to a **least-squares** weighted residual method:

$$\int w(k)\, \big[\text{residual}(k)\big]^2\, dk, \quad \text{where } w(k) = d(k).$$

- **Interpretation:**
  - If $d(k)$ is uniform over a grid, it resembles a *Collocation* method with equally weighted points.
  - If $d(k)$ represents the *stationary distribution*, we emphasize states that are visited often, making the solution accurate where it matters most in practice.
  - This approach is a general *weighted residual method* where the weighting function is chosen to

Exercise: Extending the Euler-Equation Approach

- Review the provided code that solves the canonical optimal growth model via the Euler-equation residual minimization approach:
  - `https://www.dropbox.com/scl/fi/9yjtwr36cvh6hwutjsfmu/opt_growth_nn_4.py?rlkey=svfkpiq7ei5d07336tvkkyk9m&dl=0`
- **Task:** Extend this code to incorporate a TFP (Total Factor Productivity) shock:
  - Allow $f(k)$ to become $A \cdot f(k)$, where A follows some exogenous stochastic process (e.g., AR(1)).
  - Modify the Euler equation and the sampling procedure to account for the stochastic shock in each period.
  - Ensure you sample both k and A from their joint distribution when forming the expectation.
- **Goal:** Observe how the policy function and the Euler equation residuals change when TFP is random.
- **Hint:**
  - You may need to track (k_t, A_t) as the *combined state*.
  - The updated Euler equation should include the expectation over next-period shocks.
  - Carefully adapt the loss function to reflect $\mathbb{E}_{k,A}\big[\cdot\big]$.