Chapter 10: Computational Tools

April 2026

Outline

1. Introduction
2. Approximating a Function
3. Root Finding
4. Optimization
5. Discretizing an AR(1) Process
6. Dynamic Programming via Value Function Iteration
7. Solving Dynamic Models by Linearization
8. Summary

Introduction

Motivation

Lucas (1980): "Our task as I see it. . . is to write a FORTRAN program that will accept specific economic policy rules as 'input' and will generate as 'output' statistics describing the operating characteristics of time series we care about, which are predicted to result from these policies."

Why computational tools?

- Many macroeconomic models do **not** have analytical solutions
- To compute equilibria and simulate time series

This chapter's toolkit:

- Function approximation, root finding, optimization
- Discretization of stochastic processes
- Dynamic programming via value function iteration
- Linearization/log-linearization of rational expectations models

How the Tools Fit Together

- Sections 10.1–10.4 are **building blocks**
- Section 10.5 combines them to solve a dynamic programming problem via value function iteration
- Section 10.6 presents a separate local (perturbation) method

In practice, off-the-shelf software performs these routines (one rarely codes them from scratch). But understanding *how* the methods work is crucial for:

- Choosing the right method for a given problem
- Diagnosing problems when an algorithm fails

Approximating a Function

The Problem

We often need to store a function in the computer.

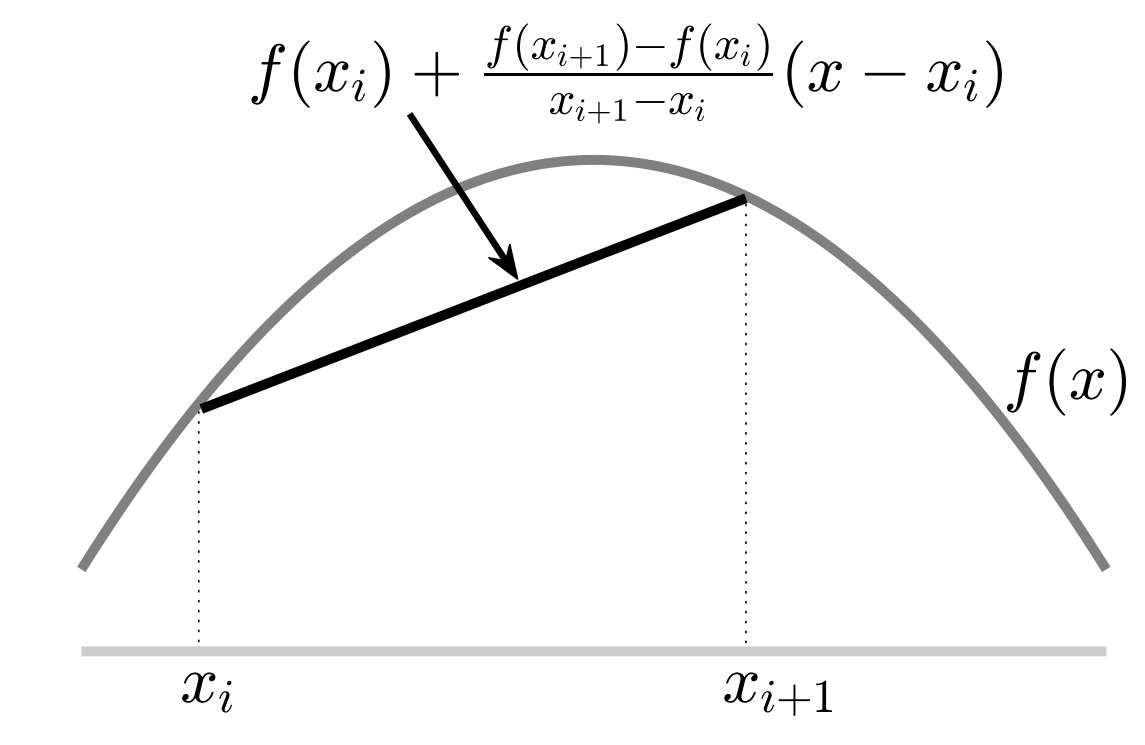
- Easy cases: $f(x) = ax^2 + bx + c$ summarized by 3 numbers
- Hard cases: value functions and policy functions from Bellman equations rarely have analytical expressions
- Storing f at each point on $[\underline{x}, \overline{x}]$ would in principle require infinite numbers

$\Rightarrow$ Need **approximation methods** to store an approximate version with finite memory.

Two popular approaches:

1. **Interpolation**: store values on grid points and interpolate between them
2. **Approximation by known functions**: weighted sums of basis functions

Linear Interpolation



Set grid points $\{x_1, x_2, \ldots, x_n\}$ with $x_1 = \underline{x}$ and $x_n = \overline{x}$. Store values $f(x_i)$.

Linear Interpolation

To approximate $f(x)$:

1. Find i such that $x \in [x_i, x_{i+1})$
2. Compute

$$\hat{f}(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i)$$

Advantage: uses only local information, robust to behavior elsewhere.

Disadvantages:

- Large error if f is highly nonlinear
- Approximated function is not differentiable at grid points

Cubic Spline Interpolation

Interpolate with a cubic function instead of linear.

Between x_i and x_{i+1}: interpolate by cubic function with the information of $f(x_i)$, $f(x_{i+1})$, $f''(x_i)$, $f''(x_{i+1})$

Key properties:

- Second derivatives computed automatically from continuity conditions + boundary conditions
- Still only need to store $\{f(x_i)\}$
- Approximated function is **twice continuously differentiable** (important in economics)

Downside vs. linear interpolation: computing $f''(x_i)$ uses information from *all* $f(x_j)$. So in principle, $f(x_j)$ can affect the approximation in $[x_i, x_{i+1})$ even when x_j is far from x_i.

Approximation by Known Functions

Approximate with a weighted sum of basis functions:

$$\hat{f}(x) = \sum_{i=1}^{n} a_i \phi_i(x)$$

- Choose n basis functions $\phi_i(x)$ (polynomials, logs, trig functions, ...)
- Evaluate f at m points $\{x_1, \ldots, x_m\}$ and solve/fit for a_i
- $m = n$: unique solution (generically)
- $m > n$: cannot solve exactly; use as a regression

Popular choice: **Chebyshev polynomials**

- Efficient representation of f
- Well-established methods for choosing evaluation points

Downside: global information. Approximation in one region affected by behavior of f and ϕ_i in other regions.

Root Finding

The Problem

Find a scalar x satisfying:

$$f(x) = 0$$

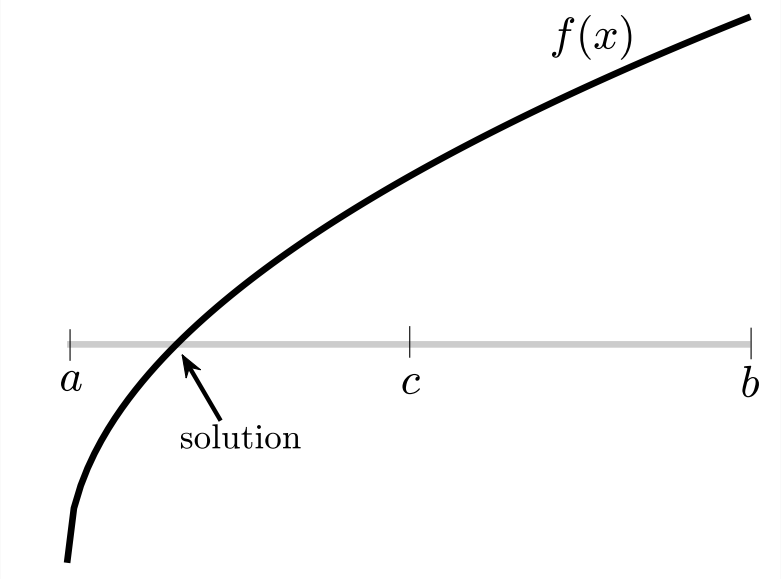
Multi-dimensional cases are harder, but typically rely on similar principles.

Brute-force grid search: Create $\{x_1, \ldots, x_n\}$ between x and $\overline{x}$, evaluate $f(x_i)$, pick the x_i with $f(x_i)$ closest to zero.

- Always works for large n
- But extremely slow $\Rightarrow$ rarely the best choice

Two smarter methods: **bisection** and **Newton–Raphson**.

Bisection Method



Steps:

1. **Bracketing**: Find a and $b > a$ with $\text{sign}(f(a)) \neq \text{sign}(f(b))$.
   - If f is continuous, a solution exists in $[a, b]$
2. Let $c = (a + b)/2$. Evaluate $f(c)$:
   - If $\text{sign}(f(c)) = \text{sign}(f(a))$: solution in $[c, b]$, update $a = c$
   - Otherwise: solution in $[a, c]$, update $b = c$
3. Repeat until $b - a < \varepsilon$ (tolerance), check $f(a) \approx 0$, return a

Bisection: Properties

Benefits:

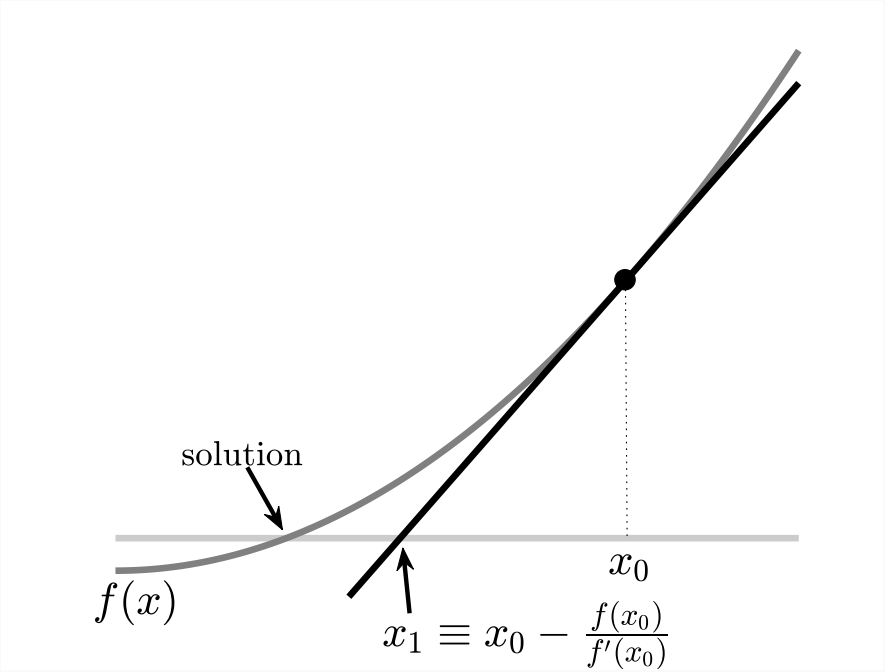
- Does **not** require differentiability of f
- Always finds *one* solution if f is continuous (even if multiple roots)
- Always terminates in a finite number of steps
- Each iteration halves the interval $\Rightarrow$ predictable speed
- Even terminates when no solution exists (but then $f(a) \not\approx 0$)

Speed comparison (solve $f(x) = 0$ on $[0, 1]$ with accuracy 10^{-3}):

- Brute-force grid search: ≈ 1000 evaluations
- Bisection: only ≈ 10 evaluations (since $(1/2)^{10} = 1/1024$)

Robust and useful despite simplicity.

Newton-Raphson Method



Idea: Approximate f locally by its tangent line

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

Setting the RHS to zero:

$$\hat{x} = x_0 - \frac{f(x_0)}{f'(x_0)}$$

If f is linear, $\hat{x}$ delivers the solution in one step.

Newton-Raphson: Algorithm and Trade-offs

Procedure:

1. Initial guess x_0
2. Update: $x_1 = x_0 - f(x_0)/f'(x_0)$
3. Check $|f(x_1)| < \varepsilon$. If not, iterate until $|f(x_i)| < \varepsilon$

Benefits: Much faster than bisection when f is close to linear or x_0 is close to the true solution.

Downsides:

- Requires differentiability: need $f'(x)$ analytically or numerically
- May fail to find the solution if f is not well-behaved

Numerical derivative (central difference):

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Choose $h \approx 10^{-5}$ (not too small—machine epsilon $\approx 10^{-16}$).

Optimization

The Problem

Maximize a function $F(x)$. Similarly to root finding, grid search works but is slow.

Focus on the **one-dimensional case**. For multi-dimensional, there are various specialized methods. One can also optimize one variable at a time:

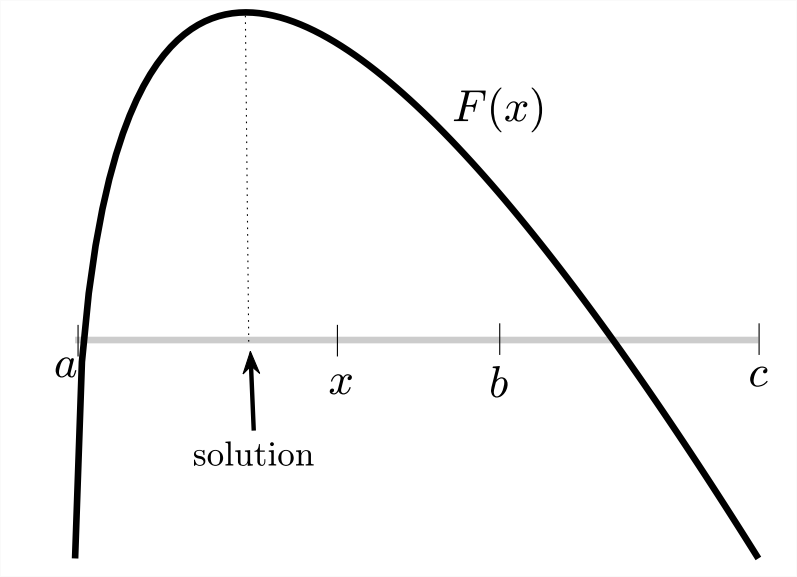
Example: $\max F(x, y)$

- Fix x, maximize over y: get $y^*(x)$
- Maximize $F(x, y^*(x))$ over x (one-dimensional)

Two methods below:

- **Golden-section search**: derivative-free
- **Newton's method**: uses first and second derivatives

Golden-Section Search



Steps:

1. Find $a < b < c$ with $F(a) < F(b)$ and $F(c) < F(b)$
   - A (local) maximum exists in $[a, c]$
   - If this fails, likely a corner solution
2. Take the longer segment (say $[a, b]$) and place $x \in [a, b]$ with $(b - x)/(b - a) = \omega^*$, where $\omega^* = (3 - \sqrt{5})/2 \approx 0.38197$
   - If $F(x) > F(b)$: new bracket (a, x, b)
   - Else: new bracket (x, b, c)
3. Repeat until bracket size $< \varepsilon$

Golden-Section Search: Properties

The constant $\omega^* = (3 - \sqrt{5})/2 \approx 0.38197$. The ratio $(1 - \omega^*)/\omega^* \approx 1.61803$ is the **golden ratio**.

Why this ω^*? Starting from $(c - b)/(c - a) = \omega^*$ and following the procedure, each iteration removes an ω^* fraction of the segment.

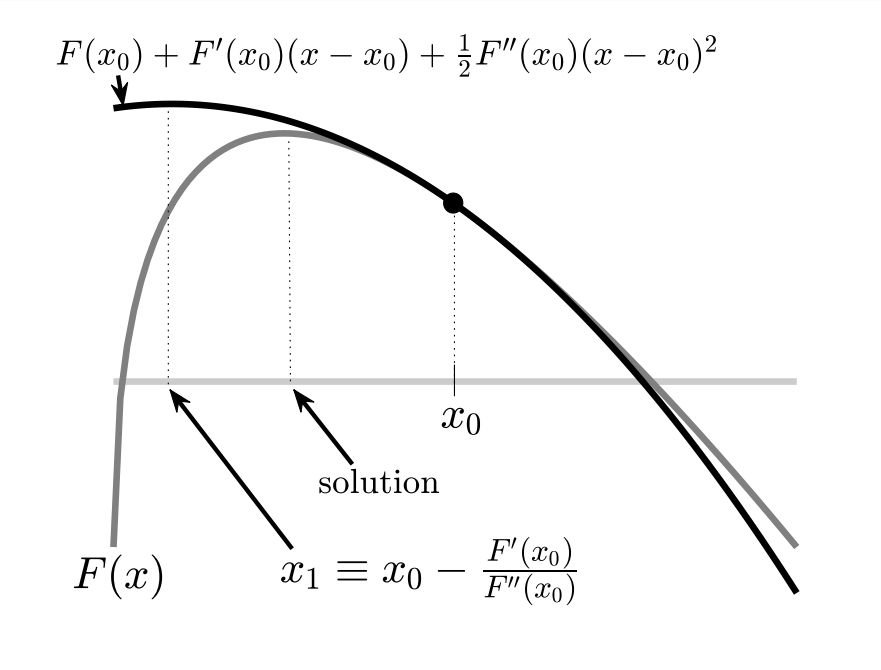
After n iterations, remaining bracket length is $(1 - \omega^*)^n (c - b)$.

Benefits:

- Does not require differentiability of F
- Terminates in pre-set number of iterations (like bisection)

Removes $\sim$1/3 of the interval per iteration (less efficient than bisection's 1/2), but robust.

Newton's Method for Optimization



Idea: Approximate F with a quadratic near x_0:

$$F(x) \approx F(x_0) + F'(x_0)(x - x_0) + \frac{1}{2}F''(x_0)(x - x_0)^2$$

Take the FOC of the RHS and set equal to zero:

$$\hat{x} = x_0 - \frac{F'(x_0)}{F''(x_0)}$$

If F is quadratic, $\hat{x}$ delivers the solution in one step.

Newton's Method: Algorithm and Properties

Procedure:

1. Initial guess x_0
2. Update: $x_1 = x_0 - F'(x_0)/F''(x_0)$
3. If $|x_1 - x_0| < \varepsilon$, stop; otherwise iterate

Benefits: Much faster than golden-section search if F is close to quadratic.

Downsides:

- Requires F' and F''
- Not guaranteed to find the maximum (good starting guess helps)

Connection: Root Finding and Optimization

Maximizing F typically requires solving the FOC $F'(x) = 0$, so optimization is equivalent to finding the root of F'.

Newton for optimization:

$$x_{i+1} = x_i - \frac{F'(x_i)}{F''(x_i)}$$

Newton–Raphson applied to $F'(x) = 0$: same iteration!

Golden-section vs. bisection on FOC:

- Golden-section removes $\sim 1/3$ of interval; bisection removes $1/2$
- Golden-section benefits:
  - Does not require F to be differentiable
  - Each step aims at maximization (not minimum)

Practice: Use Newton when F is smooth and derivatives are available. Use golden-section when robustness is more important.

Example 1: Labor-Leisure Choice

Consumer solves:

$$\max_{c,\ell} u(c,\ell) \quad \text{subject to} \quad c = f(\ell) + x$$

with $u(c,\ell) = \ln(c) - \omega\ell^{\gamma}/\gamma$, $\omega > 0$, $\gamma > 0$, and $f(\ell) = \ell^{\alpha}$, $\alpha \in (0,1)$.

Problem reduces to maximizing

$$\ln(\ell^{\alpha} + x) - \frac{\omega}{\gamma}\ell^{\gamma}$$

or solving the FOC

$$\frac{\alpha\ell^{\alpha-1}}{\ell^{\alpha} + x} - \omega\ell^{\gamma-1} = 0$$

Parameters: $\alpha = 1/3$, $x = 0.5$, $\omega = 1$, $\gamma = 2$.

Three MATLAB codes: `Ex1_bisection.m`, `Ex1_newton.m`, `Ex1_GS.m`.

Discretizing an AR(1) Process

The Problem

Many macro models use an AR(1) shock:

$$z_{t+1} = \rho z_t + \varepsilon_{t+1}, \qquad \rho \in (0,1)$$

with $\mathbb{E}_t[\varepsilon_{t+1}] = 0$, $\text{Var}[\varepsilon_{t+1}] = \sigma_\varepsilon^2$, and $\Pr[\varepsilon \leq u] = F(u/\sigma_\varepsilon)$.

Unconditional variance:

$$\sigma_z^2 = \frac{1}{1-\rho^2}\sigma_\varepsilon^2$$

Goal: Approximate $\{z_t\}$ by a Markov chain with:

- A finite set of grid points $\{\bar{z}^1, \ldots, \bar{z}^N\}$
- Transition matrix with elements π_{ij} (probability of $i \to j$)

Two popular methods: **Tauchen method** and **Rouwenhorst method**.

The Tauchen Method

Tauchen (1986):

1. Create equally spaced grid $\{\bar{z}^1, \ldots, \bar{z}^N\}$ with distance ω
   - Common choice: $\bar{z}^1 = -m\sigma_z$, $\bar{z}^N = m\sigma_z$
   - m controls how wide the grid is (e.g., $m = 3$ covers 3 SDs)
2. Set probabilities

$$\pi_{ij} = \begin{cases} F\left(\dfrac{\bar{z}^1 - \rho\bar{z}^i + \omega/2}{\sigma_\varepsilon}\right) & j = 1 \\ F\left(\dfrac{\bar{z}^j - \rho\bar{z}^i + \omega/2}{\sigma_\varepsilon}\right) - F\left(\dfrac{\bar{z}^j - \rho\bar{z}^i - \omega/2}{\sigma_\varepsilon}\right) & j = 2, \ldots, N-1 \\ 1 - F\left(\dfrac{\bar{z}^N - \rho\bar{z}^i - \omega/2}{\sigma_\varepsilon}\right) & j = N \end{cases}$$

Tauchen: Intuition

For $i \in \{1, \ldots, N\}$ and $j \in \{2, \ldots, N-1\}$:

- Start at $\bar{z}^i$
- Compute the probability that the AR(1) next-period value falls in $[\bar{z}^j - \omega/2, \bar{z}^j + \omega/2]$
- Assign this probability to π_{ij}

End points ($j = 1$ and $j = N$): similar logic, but account for the support of z extending beyond the grid.

Concern: Tauchen can be inaccurate when ρ is close to 1. A problem because many macro processes (income, productivity) are very persistent.

The Rouwenhorst Method

Rouwenhorst (1995): better approximation for high persistence.

Equally spaced grid, with $\bar{z}^1 = -m\sigma_z$, $\bar{z}^N = m\sigma_z$, $m = \sqrt{N-1}$.

Construct Markov matrix Π_N **recursively**: **For** $N = 2$:

$$\Pi_2 = \begin{pmatrix} p & 1-p \\ 1-p & p \end{pmatrix}$$

For $N \geq 3$: construct the $N \times N$ matrix ($\mathbf{0}$ is $(N-1) \times 1$ zero matrix)

$$\Pi_N = p \begin{pmatrix} \Pi_{N-1} & \mathbf{0} \\ \mathbf{0}' & 0 \end{pmatrix} + (1-p) \begin{pmatrix} \mathbf{0} & \Pi_{N-1} \\ 0 & \mathbf{0}' \end{pmatrix}$$
$$+p \begin{pmatrix} \mathbf{0}' & 0 \\ \Pi_{N-1} & \mathbf{0} \end{pmatrix} + (1-p) \begin{pmatrix} 0 & \mathbf{0}' \\ \mathbf{0} & \Pi_{N-1} \end{pmatrix}$$

Then divide all but the top and bottom rows by two (so each row sums to 1).

Rouwenhorst: Properties

Set $p = (1+\rho)/2$ and $m = \sqrt{N-1}$. Then:

- Unconditional mean and variance of the Markov chain match those of the original AR(1)
- Uses only information about mean and variance
- Invariant distribution converges to a normal distribution as $N \to \infty$
- Especially recommended when ε_{t+1} is normal
- **Particularly powerful when ρ is close to 1**

See Kopecky and Suen (2010) for further exposition and characterizations.

Example 2: Approximating an AR(1)

Consider the AR(1):

$$y_{t+1} = 0.95\, y_t + \varepsilon_{t+1}$$

where $\varepsilon_{t+1} \sim N(0, \sigma^2)$ with $\sigma = 0.2$.

MATLAB codes:

- `Ex2_tauchen.m`: approximates using 10-state Markov chain via Tauchen
- `Ex2_rouwenhorst.m`: same task via Rouwenhorst

With $\rho = 0.95$ (highly persistent), Rouwenhorst is expected to perform better.

Dynamic Programming via Value Function Iteration

The Bellman Equation

Consider a Bellman equation of the form

$$V(k) = \max_{k'} \{F(k, k') + \beta V(k')\}$$

- k: state variable; k': next-period state
- $F(k, k')$: return function for the current period
- $\beta \in (0, 1)$: discount factor

Define the mapping T:

$$TV_i(k) = \max_{k'} \{F(k, k') + \beta V_i(k')\}$$

Given a function V_i, TV_i is a new function.

Value Function Iteration: Algorithm

Contraction mapping theorem (Chapter 4): starting from any V_0 and iterating $V_{i+1} = TV_i$, $V_i \to V$ (the true value function) as $i \to \infty$.

Algorithm:

1. Fix a grid on values of k
2. Start from an arbitrary $V_0(k)$ represented on the grid
3. For each k on the grid, solve $\max_{k'} F(k, k') + \beta V_0(k')$
    - May involve interpolating V_0 off the grid
4. Let $V_1(k)$ be the maximized value. Repeat using $V_1, V_2, \ldots$ until $V_i \approx V_{i+1}$

Pros: Extremely robust. The theorem guarantees convergence.

Cons: Can be slow, especially when β is close to 1. Other (faster) methods exist.

Deterministic Case: Cake-Eating Problem

Infinitely lived consumer with initial cake size a_0:

$$\max \sum_{t=0}^{\infty} \beta^t \sqrt{c_t} \quad \text{subject to} \quad a_{t+1} = a_t - c_t$$

Bellman equation:

$$V(a) = \max_{a'} \sqrt{a - a'} + \beta V(a')$$

- Natural grid: $[0, a_0]$
- $a_1 = 0$, $a_N = a_0$, N grid points
- Initial guess: $V_0(a^j) = 0$ for all j (zero vector)

Cake-Eating: Algorithm Details

Step 3 (the maximization): Two options.

Option A (grid search): Restrict a' to grid points.

- Pick the grid point a^n (with $a^n \leq a^j$) that maximizes $\sqrt{a^j - a^n} + \beta V_0(a^n)$
- Simple but coarse

Option B (off-grid optimization): Allow a' anywhere in $[0, a^j]$.

- Evaluate $V_0(a')$ via interpolation
- If V_0 is differentiable: use FOC + root finding
- Or use golden-section search
- The contraction mapping guarantees concavity here $\Rightarrow$ FOC works

Termination: Stop when $\max_j |V_{i+1}(a^j) - V_i(a^j)| < \varepsilon$.

Example 3: Deterministic DP

Cake-eating problem with $\beta = 0.98$ and $a_0 = 10$.

Provided codes conduct VFI and simulate the first 100 periods to plot:

- Time series of leftover cake a_t
- Time series of consumption c_t
- Value function $V(a)$
- Policy function $a'(a)$

- `Ex3_gridsearch.m`: restricts a' to grid points
- `Ex3_GS.m`: allows a' off the grid (golden-section search + interpolation)

Stochastic Case: Cake-Eating with Shocks

At the beginning of each period, an additional cake z arrives ("high" z_H or "low" z_L), following a Markov chain with $\pi_{zz'}$.

Consumer maximizes:

$$\mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t \sqrt{c_t} \right] \quad \text{subject to} \quad a_{t+1} = a_t - c_t + z_t$$

Bellman equation:

$$V(a, z) = \max_{a'} \sqrt{a - a' + z} + \beta[\pi_{zH} V(a', H) + (1 - \pi_{zH}) V(a', L)]$$

Differences from deterministic case:

- Natural upper bound of grid is *not* a_0 (cake can grow)
- Work with two vectors $V_i(a^j, H)$ and $V_i(a^j, L)$

Stochastic Neoclassical Growth Model: Setup

Representative consumer's utility:

$$\mathbb{E}_0\left[\sum_{t=0}^{\infty}\beta^t\big((1-\phi)\log(C_t)+\phi\log(1-H_t)\big)\right]$$

Resource constraint:

$$K_{t+1}+C_t=\exp(z_t)K_t^{\alpha}H_t^{1-\alpha}+(1-\delta)K_t$$

Productivity shock:

$$z_{t+1}=\rho z_t+\varepsilon_{t+1},\qquad \varepsilon_{t+1}\sim \text{i.i.d.},\ \ \mathbb{E}[\varepsilon]=0,\ \ \text{SD}=\sigma$$

- K_t: predetermined
- C_t, H_t: non-predetermined (chosen within period)
- $\beta,\phi,\alpha,\delta,\rho\in(0,1)$, $\sigma>0$

Stochastic NGM: Bellman Equation

$$
\begin{aligned}
&V(K,z) \\
&\quad = \max_{K',H} (1-\phi)\log\big(\exp(z)K^{\alpha}H^{1-\alpha} + (1-\delta)K - K'\big) \\
&\qquad\qquad +\phi\log(1-H) + \beta\mathbb{E}[V(K',z')|z]
\end{aligned}
$$

Two differences from cake-eating:

1. Domain of z is a real line $\Rightarrow$ discretize z_t using Tauchen or Rouwenhorst
2. Planner chooses a second variable H

Strategy for H: Solve the inner problem first.

$$\max_{H} (1-\phi)\log\big(\exp(z)K^{\alpha}H^{1-\alpha} + (1-\delta)K - K'\big) + \phi\log(1-H)$$

gives $H(z,K,K')$ (numerical, not analytical).

Plug $H(z,K,K')$ back into the Bellman equation and proceed as in the cake-eating problem.

Example 4: Stochastic NGM (Brock-Mirman)

Calibration: two-state Markov for $Z_t = \exp(z_t)$

- $Z \in \{0.985, 1.015\}$; $\pi_{HH} = \pi_{LL} = 0.95$, $\pi_{LH} = \pi_{HL} = 0.05$
- $\beta = 0.99$, $\alpha = 0.36$, $\delta = 0.025$
- ϕ chosen so that steady-state $H = 1/3$

Codes:

- `Ex4_gridsearch.m`: restricts K' to grid; uses bisection for inner H
- `Ex4_GS.m`: K' off grid (uses subroutines `Ex4_location.m`, `Ex4_labor.m`)

Simulation: 3000 periods for Y_t, C_t, H_t, $I_t = K_{t+1} - (1-\delta)K_t$, Z_t. After dropping first 100: log, HP-filter, compute standard deviations and correlations with Y_t.

Solving Dynamic Models by Linearization

Global vs. Local Methods

Global methods (previous section):

- Construct grid covering the relevant state space
- Approximate the solution at all grid points

Local methods (this section):

- Choose one point (typically the deterministic steady state)
- Examine how the solution changes *locally* around that point
- **Perturbation method**: approximate with simple functions (polynomials)
- This chapter: **linear** (log-linear) approximation only

When is the local approximation valid?

- Small deviations from steady state
- E.g., small AR(1) shocks around a long-run steady state

Why Use Linearization?

Speed: Much faster than global methods, especially with many variables.

Curse of dimensionality: Global methods (Example 4) become infeasible when the number of endogenous state variables is large.

Matrix algebra: Well-established methods for solving large linear systems.

Certainty-equivalence (affine property of expectations operator):

- Can solve the stochastic model as if it were deterministic with expected values
- Dynamics are effectively the same whether shocks are stochastic or deterministic

Predetermined vs. Non-Predetermined Variables

When the model is nonlinear, first **log-linearize**. Then work with a system of linear equations in log-deviations from the steady state.

Predetermined variables: values already determined when entering period t.

- Example: capital stock K_t in the growth model

Non-predetermined (jump) variables: values determined within period t.

- Example: consumption C_t in the growth model

Goal of "solving the model": Express non-predetermined and future predetermined variables as explicit functions of current predetermined variables and exogenous shocks.

Log-Linearizing the Stochastic NGM

Euler equation:

$$\frac{1}{C_t} = \mathbb{E}_t\left[\beta\left(1 + \alpha \exp(z_{t+1}) K_{t+1}^{\alpha-1} H_{t+1}^{1-\alpha} - \delta\right)\frac{1}{C_{t+1}}\right]$$

Labor-supply condition:

$$\frac{1-\phi}{C_t}(1-\alpha)\exp(z_t)K_t^{\alpha}H_t^{-\alpha} = \frac{\phi}{1-H_t}$$

Notation: Lower-case letter is log-deviation from steady state:

$$x_t \equiv \log(X_t/\bar{X})$$

where $\bar{X}$ is the steady-state value.

Log-Linearized System

Resource constraint:

$$\bar{K}k_{t+1} + \bar{C}c_t = \bar{K}^{\alpha}\bar{H}^{1-\alpha}(z_t + \alpha k_t + (1-\alpha)h_t) + (1-\delta)\bar{K}k_t$$

Euler equation:

$$-c_t = \mathbb{E}_t\big[\beta\alpha\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}(z_{t+1} + (\alpha-1)k_{t+1} + (1-\alpha)h_{t+1}) - c_{t+1}\big]$$

Labor supply:

$$h_t = \theta(z_t + \alpha k_t - c_t), \qquad \theta \equiv \frac{1-\bar{H}}{\bar{H}(1-\alpha)+\alpha}$$

Strategy: Use the labor supply equation to eliminate h_t from the other two equations.

Eliminating h_t

After substitution, the system becomes

$$\begin{aligned}\bar{K}k_{t+1} = \bar{K}^{\alpha}\bar{H}^{1-\alpha}(1+\theta(1-\alpha))z_t \\ +\big(\bar{K}^{\alpha}\bar{H}^{1-\alpha}\alpha(1+\theta(1-\alpha)) + (1-\delta)\bar{K}\big)k_t \\ -\big(\bar{K}^{\alpha}\bar{H}^{1-\alpha}\theta(1-\alpha) + \bar{C}\big)c_t\end{aligned}$$

and (using $\mathbb{E}_t[z_{t+1}] = \rho z_t$):

$$\begin{aligned}\beta\alpha\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}(1-\alpha)(1-\theta\alpha)k_{t+1} + \big(\beta\alpha\theta(1-\alpha)\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}+1\big)\mathbb{E}_t[c_{t+1}] \\ = \beta(1+\theta(1-\alpha))\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}\rho z_t + c_t\end{aligned}$$

This is a system of linear equations in $(k_t, c_t, k_{t+1}, \mathbb{E}_t[c_{t+1}], z_t)$.

The Blanchard-Kahn Condition

Blanchard and Kahn (1980): Derived a condition for the uniqueness of the equilibrium in linear rational expectations models.

Purpose:

- Determines when a unique equilibrium exists
- Helps construct the solution

Before the general result, fix ideas with simple examples.

Example: One Non-Predetermined Variable

Consider

$$y_t = \phi y_{t+1}, \qquad \phi > 0$$

Rewrite as

$$y_{t+1} = \lambda y_t, \qquad \lambda = 1/\phi$$

Equilibrium condition: $\lim_{T\to\infty} y_{t+T}$ finite.

Applying repeatedly: $y_{t+T} = \lambda^T y_t$.

- **If** $\lambda > 1$: only $y_t = 0$ is consistent with finite limit. **Unique equilibrium**.
- **If** $\lambda \leq 1$: any y_t works. **Indeterminacy**.

Logic: To guarantee uniqueness, λ must be such that "wrong" y_t makes the economy blow up. Then the unique "right" y_t is the solution.

Example: Adding an Exogenous Shock

Now add an AR(1) shock:

$$\mathbb{E}_t[y_{t+1}] = \lambda y_t + z_t, \qquad z_{t+1} = \rho z_t + \varepsilon_{t+1}, \quad \rho \in (0,1)$$

Iterating forward (assuming $\lambda \neq \rho$):

$$\mathbb{E}_t[y_{t+T}] = \lambda^T \left[y_t + \frac{1 - (\rho/\lambda)^T}{\lambda - \rho} z_t \right]$$

If $\lambda > 1$: the only y_t that makes $\lim_{T\to\infty} \mathbb{E}_t[y_{t+T}]$ finite is

$$y_t = -\frac{1}{\lambda - \rho} z_t$$

If $\lambda \leq 1$: indeterminacy.

"Guess and Verify" Method

If uniqueness holds, one can also use the **method of undetermined coefficients**.

Guess: $y_t = \mathcal{A} z_t$ for unknown constant $\mathcal{A}$.

Plug into $\mathbb{E}_t[y_{t+1}] = \lambda y_t + z_t$:

$$\mathbb{E}_t[\mathcal{A} z_{t+1}] = \lambda \mathcal{A} z_t + z_t$$

Using $\mathbb{E}_t[z_{t+1}] = \rho z_t$:

$$(\mathcal{A}\rho - \mathcal{A}\lambda - 1) z_t = 0$$

Must hold for all $z_t \Rightarrow$

$$\mathcal{A} = -\frac{1}{\lambda - \rho}$$

General Setup

Model with m non-predetermined variables and $n + k$ predetermined variables:

- x_t: $n \times 1$ vector of endogenous predetermined variables
- y_t: $m \times 1$ vector of non-predetermined variables
- a_t: $k \times 1$ vector of exogenous predetermined variables

System form:

$$B \begin{pmatrix} x_{t+1} \\ \mathbb{E}_t[y_{t+1}] \end{pmatrix} = A \begin{pmatrix} x_t \\ y_t \end{pmatrix} + E a_t$$

where B and A are $(n + m) \times (n + m)$ matrices, and E is $(n + m) \times k$.

Each element of a_t follows an AR(1) process:

$$a^i_{t+1} = \rho a^i_t + \varepsilon^i_{t+1}$$

Stochastic NGM in Matrix Form

With $x_t = k_t$, $y_t = c_t$, $a_t = z_t$:

$$B \begin{pmatrix} k_{t+1} \\ \mathbb{E}_t[c_{t+1}] \end{pmatrix} = A \begin{pmatrix} k_t \\ c_t \end{pmatrix} + E z_t$$

where

$$B = \begin{pmatrix} \bar{K} & 0 \\ \beta\alpha\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}(1-\alpha)(1-\theta\alpha) & \beta\alpha\theta(1-\alpha)\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}+1 \end{pmatrix}$$

$$A = \begin{pmatrix} \bar{K}^{\alpha}\bar{H}^{1-\alpha}\alpha(1+\theta(1-\alpha))+(1-\delta)\bar{K} & -(\bar{K}^{\alpha}\bar{H}^{1-\alpha}\theta(1-\alpha)+\bar{C}) \\ 0 & 1 \end{pmatrix}$$

$$E = \begin{pmatrix} \bar{K}^{\alpha}\bar{H}^{1-\alpha}(1+\theta(1-\alpha)) \\ \beta(1+\theta(1-\alpha))\bar{K}^{\alpha-1}\bar{H}^{1-\alpha}\rho \end{pmatrix}$$

Jordan Decomposition

Suppose B is invertible. Then rewrite:

$$\begin{pmatrix} x_{t+1} \\ \mathbb{E}_t[y_{t+1}] \end{pmatrix} = F \begin{pmatrix} x_t \\ y_t \end{pmatrix} + Ga_t, \qquad F = B^{-1}A, \;\; G = B^{-1}E$$

Jordan decomposition: $F = HJH^{-1}$, where J is diagonal with eigenvalues of F, and H has corresponding eigenvectors.

Order eigenvalues in ascending absolute value:
$|\lambda_1| < |\lambda_2| < \cdots < |\lambda_{n+m}|$.

Let $h =$ number of eigenvalues with $|\lambda_i| > 1$.

Blanchard-Kahn condition: The equilibrium is unique iff

$$h = m$$

That is, the number of eigenvalues outside the unit circle equals the number of non-predetermined variables.

Intuition for Blanchard-Kahn

Matrix multiplication combines **expansion/contraction** and **rotation**.

Eigenvalues tell us whether the multiplication expands ($|\lambda_i| > 1$) or contracts ($|\lambda_i| < 1$) in the direction of the corresponding eigenvector.

When $h = m$: We can choose a unique set of non-predetermined variables such that $(x_t', y_t')'$ has zero component in every "expanding" direction.

- Any other choice would cause the system to diverge
- So the unique non-exploding choice is the solution

When $h < m$: indeterminacy (multiple non-exploding paths).

When $h > m$: no solution (cannot eliminate all exploding directions with m free jump variables).

Constructing the Solution

Using the Jordan decomposition:

$$H^{-1}\begin{pmatrix} x_{t+1} \\ \mathbb{E}_t[y_{t+1}] \end{pmatrix} = JH^{-1}\begin{pmatrix} x_t \\ y_t \end{pmatrix} + H^{-1}Ga_t$$

Partition with n and m rows/columns:

$$H^{-1} = \begin{pmatrix} \tilde{H}_{11} & \tilde{H}_{12} \\ \tilde{H}_{21} & \tilde{H}_{22} \end{pmatrix}, \qquad J = \begin{pmatrix} J_1 & 0 \\ 0 & J_2 \end{pmatrix}, \qquad H^{-1}G = \begin{pmatrix} \Gamma_1 \\ \Gamma_2 \end{pmatrix}$$

Define transformed variables:

$$\begin{pmatrix} \tilde{x}_t \\ \tilde{y}_t \end{pmatrix} = H^{-1}\begin{pmatrix} x_t \\ y_t \end{pmatrix}$$

Then:

$$\begin{pmatrix} \tilde{x}_{t+1} \\ \mathbb{E}_t[\tilde{y}_{t+1}] \end{pmatrix} = \begin{pmatrix} J_1 & 0 \\ 0 & J_2 \end{pmatrix}\begin{pmatrix} \tilde{x}_t \\ \tilde{y}_t \end{pmatrix} + \begin{pmatrix} \Gamma_1 \\ \Gamma_2 \end{pmatrix} a_t$$

Solving the Transformed System

The last m rows are:

$$\mathbb{E}_t[\tilde{y}_{t+1}] = J_2 \tilde{y}_t + \Gamma_2 a_t$$

J_2 is diagonal with eigenvalues $|\lambda_i| > 1$. Apply the one-dimensional argument to each element:

$$\tilde{y}_t = \Lambda \Gamma_2 a_t$$

where Λ is $m \times m$ diagonal with elements $-1/(\lambda_i - \rho)$ for $i = n+1, \ldots, n+m$.

From the definition of $\tilde{y}_t$:

$$\Lambda \Gamma_2 a_t = \tilde{y}_t = \tilde{H}_{21} x_t + \tilde{H}_{22} y_t$$

The Final Solution

Solve for y_t:

$$y_t = -\tilde{H}_{22}^{-1}\tilde{H}_{21}x_t + \tilde{H}_{22}^{-1}\Lambda\Gamma_2 a_t$$

From the original system with $F = \begin{pmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{pmatrix}$:

$$\begin{aligned} x_{t+1} &= F_{11}x_t + F_{12}y_t + G_1 a_t \\ &= (F_{11} - F_{12}\tilde{H}_{22}^{-1}\tilde{H}_{21})x_t + (G_1 + F_{12}\tilde{H}_{22}^{-1}\Lambda\Gamma_2)a_t \end{aligned}$$

Done: y_t and x_{t+1} are both expressed as explicit linear functions of the predetermined state x_t and exogenous shock a_t.

Example 5: Solving Log-Linearized NGM

Stochastic neoclassical growth model with same parameters as Example 4, plus $\rho = 0.95$.

MATLAB code `Ex5_BK.m`:

- $J(1,1) = 0.9537$, $J(2,2) = 1.0592$
- Blanchard-Kahn satisfied ($h = 1$ eigenvalue outside unit circle; $m = 1$ jump variable)

Solution:

$$c_t = 0.5691\, k_t + 0.3920\, z_t$$

$$k_{t+1} = 0.9537\, k_t + 0.1132\, z_t$$

$$h_t = -0.2431\, k_t + 0.7070\, z_t$$

The code also plots log-linearized impulse response functions and computes HP-filtered statistics as in Example 4, with $\sigma = 0.007$.

Summary

Key Takeaways

1. **Function approximation**: Linear interpolation; cubic spline; Chebyshev polynomials
2. **Root finding**: Bisection; Newton-Raphson
3. **Optimization**: Golden-section search; Newton's method
4. **Root finding and optimization are connected**: optimizing $F \equiv$ finding roots of F'
5. **Discretizing AR(1)**: Tauchen method; Rouwenhorst method
6. **Value function iteration**: relies on contraction mapping theorem; robust but can be slow
7. **Linearization**: fast local solution; certainty equivalence; uses matrix algebra
8. **Blanchard–Kahn condition**: unique equilibrium iff number of eigenvalues outside unit circle $=$ number of non-predetermined variables