Computational Tools for Macroeconomics using MATLAB

Week 6 – Optimization & Calibration

Cristiano Cantore

Sapienza University of Rome

Learning Outcomes

By the end of this week, you will be able to:

1. Understand the difference between unconstrained and constrained optimization.
2. Use MATLAB's built-in optimization functions (‘fminsearch‘, ‘fminunc‘, ‘fmincon‘).
3. Implement simple search methods (e.g., golden section search) manually.
4. Calibrate model parameters to match economic targets.
5. Assess goodness of fit for calibration exercises.

Optimization in Economics

- Agents choose variables to **maximize utility** or **minimize costs**.
- Typical examples:
  - Household: $\max_{c,l} U(c, l)$ subject to budget/time constraint.
  - Firm: $\max_{K,L} \Pi(K, L)$ given production function.
- Optimization $\Rightarrow$ equilibrium conditions (first-order conditions).

Unconstrained vs. Constrained Problems

Unconstrained:

$$\max_x f(x)$$

$$\Rightarrow f'(x^*) = 0$$

Constrained:

$$\max_x f(x) \quad \text{s.t. } g(x) = 0,\ h(x) \leq 0$$

$\Rightarrow$ use Lagrange multipliers.

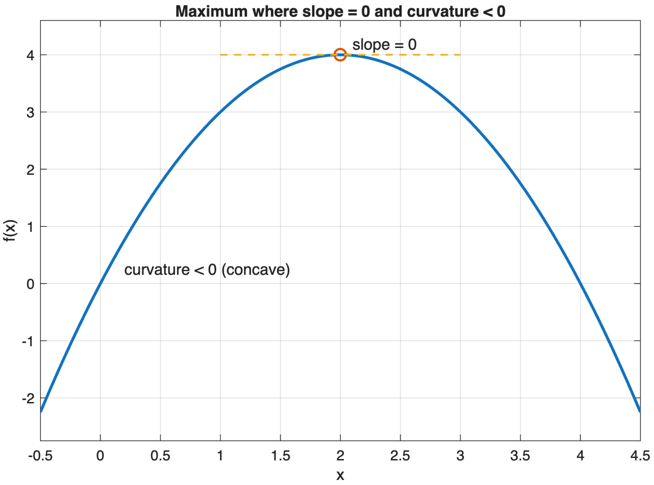
Economic Interpretation

Constraints represent scarce resources, time, or budgets.

Link to Previous Topics

- Week 5: root-finding $\Rightarrow$ find $f(x) = 0$.
- Optimization $\Rightarrow$ find $f'(x) = 0$ (and check curvature).
- Both rely on **iterative numerical methods**.
- Same logic, different purpose: locating maxima/minima instead of roots.

Visualizing Optimization



Maximum where slope $= 0$ *and curvature* < 0.

Unconstrained Optimization in Practice

- In many economic problems, we maximize or minimize a smooth function $f(x)$.
- If derivatives are available $\Rightarrow$ analytical solution.
- Otherwise: use **search algorithms**.
- MATLAB offers built-in routines:
  - `fminsearch`: derivative-free (Nelder–Mead simplex).
  - `fminunc`: uses gradients (if available).

Example: Quadratic Function

- Let $f(x) = -(x-2)^2 + 4$
- Theoretical maximum: $x^* = 2, f(x^*) = 4$
- We'll find it numerically.

Key idea

Search iteratively for x that maximizes $f(x)$ or minimizes $-f(x)$.

Manual (Grid) Search

```
% Define function
f = @(x) -(x-2).^2 + 4;

% Define grid of points
x = linspace(0, 4, 100);

% Evaluate and find maximum
y = f(x);
[~, idx] = max(y);

% Display result
x_star = x(idx)
y_star = y(idx)
```

Pros and Cons

+ Simple to understand. **–** Inefficient and imprecise (depends on grid density).

Golden Section Search (1D)

- Efficient way to locate a maximum/minimum in one dimension.
- Does not require derivatives.
- Iteratively shrinks interval $[a, b]$ until length $< \varepsilon$.

$$x_1 = b - \phi(b - a), \quad x_2 = a + \phi(b - a), \quad \phi = 0.618$$

$$\text{If } f(x_1) < f(x_2) \Rightarrow a = x_1, \text{ else } b = x_2$$

Why $\phi = 0.618$?

- The golden-section search keeps the same proportion each time the interval $[a, b]$ is reduced.
- The ratio between the whole interval and the larger subinterval equals the ratio between the larger and the smaller:

$$\frac{b-a}{b-x_1} = \frac{b-x_1}{x_1-a} = r$$

- Solving $r^2 = r + 1$ gives $r = \frac{1+\sqrt{5}}{2} \approx 1.618$
- The inverse of this number is:

$$\phi = \frac{1}{r} = \frac{\sqrt{5}-1}{2} \approx 0.618$$

Intuition

ϕ keeps the search interval shrinking in constant proportion, reusing previous function evaluations efficiently.

Example: Golden Section Search

```
f = @(x) -(x-2).^2 + 4;
a = 0; b = 4; tol = 1e-4;
phi = (sqrt(5)-1)/2;

while (b - a) > tol
    x1 = b - phi*(b - a);
    x2 = a + phi*(b - a);
    if f(x1) < f(x2)
        a = x1;
    else
        b = x2;
    end
end
x_star = (a + b)/2;
f_star = f(x_star);
```

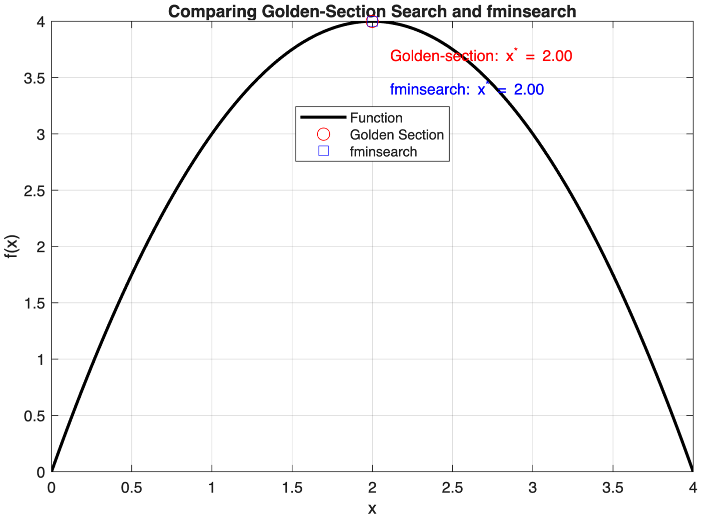
Built-In Solver: fminsearch

```
f = @(x) -(x-2).^2 + 4;
x0 = 1;                              % starting guess
[x_star, fval] = fminsearch(@(x) -f(x), x0);
fprintf('Optimum: x = %.4f, f(x) = %.4f\n', x_star, -fval);
```

Key Points

- `fminsearch` minimizes by default → use $-f(x)$ for maximization.
- Works without gradients.
- Sensitive to starting values in multimodal problems.

Comparing Search Methods



- Both methods find $x^* = 2$.
- `fminsearch` is faster, but golden section is more robust.

Why Constrained Optimization?

- In economics, agents face **constraints**:
  - Consumers: budget or time.
  - Firms: technology or capacity.
- Typical problem:

$$\max_x f(x) \quad \text{s.t. } g(x) = 0, \; h(x) \leq 0$$

- Constraints define the **feasible set**.
- Solution requires balancing objectives and constraints.

Lagrange Method (Concept)

- Combine objective and constraints:

$$\mathcal{L}(x,\lambda) = f(x) + \lambda[b - g(x)]$$

- First-order conditions:

$$\frac{\partial \mathcal{L}}{\partial x} = 0, \quad \frac{\partial \mathcal{L}}{\partial \lambda} = 0$$

- λ (Lagrange multiplier) = shadow value of relaxing constraint.

Interpretation

How much utility or profit increases if the constraint is loosened by one unit.

Example: Utility Maximization

- Preferences: $U(c, l) = c^{\alpha}(1 - l)^{1-\alpha}$
- Budget constraint: $c = wl + (1 + r)a_0 - T$
- Choose $l \in [0, 1]$ to maximize $U(c, l)$
- Parameters:

$$\alpha = 0.3, \quad w = 1, \quad r = 0.02, \quad a_0 = .05, \quad T = 0.1$$

Using `fmincon` in MATLAB

```
% Parameters
alpha = 0.3; w = 1; r = 0.02; a0 = .05; T = 0.1;

% Objective (negative utility for minimization)
U = @(l) -((w*l + (1+r)*a0 - T).^alpha .* (1 - l).^(1 - alpha)

% Bounds and initial guess
l0 = 0.5; lb = 0; ub = 1;

% Solve
l_star = fmincon(U, l0, [], [], [], [], lb, ub);

fprintf('Optimal␣labor␣supply:␣%.4f\n', l_star);
```

Inspecting the Solution

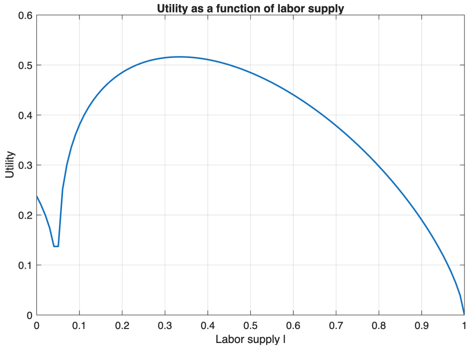
- Solver returns l^* and optionally λ^*.
- Check:
  - Is $0 < l^* < 1$ (interior)?
  - Does $U(l)$ decrease outside this range?
- Plot objective for visual confirmation.

Interpretation

At optimum, marginal benefit of leisure equals marginal cost of working.

Plotting the Objective Function

```
l = linspace(0,1,100);
U_vals = -U(l);                % convert back to positive utilit
plot(l, U_vals, 'LineWidth',1.5)
xlabel('Labor␣supply␣l'); ylabel('Utility');
title('Utility␣as␣a␣function␣of␣labor␣supply');
grid on
```

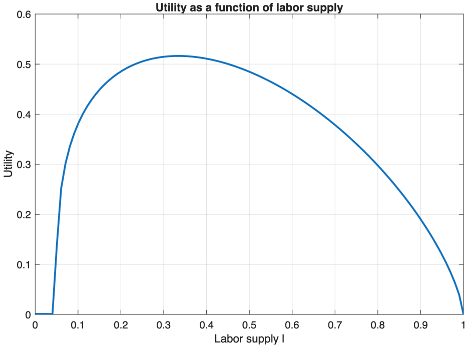


Plotting the Objective Function

- Why the initial dip?

Plotting the Objective Function

- Why the initial dip?
- Because of plotting for negative values of C!



Common Issues in Constrained Problems

- Poor starting values $\Rightarrow$ local minima.
- Infeasible initial guesses.
- Flat regions or discontinuities.
- Incorrect constraint specification.

Practical Tips

Always:

- Test the objective over a grid.
- Check constraint satisfaction.
- Verify solution by plotting.

Calibration as Optimization

- Economic models include parameters (θ) that must be chosen.
- **Calibration:** pick θ so that model moments match data moments.

$$\min_{\theta} \sum_{i} \left[m_i^{model}(\theta) - m_i^{data} \right]^2$$

- Equivalent to an optimization problem:

$$\theta^* = \arg\min_{\theta} \; \text{loss}(\theta)$$

- Key question: how to define the loss function and choose solver?

Example: Targeting Steady-State Labor

- Utility: $U(c, l) = c^{\alpha}(1 - l)^{1-\alpha}$
- Budget: $c = wl + (1 + r)a_0 - T$
- Goal: choose α so that steady-state labor $l^* = 0.3$
- Steps:
  1. For a given α, compute $l^*(\alpha)$ using `fmincon`.
  2. Define loss function: $(l^*(\alpha) - 0.3)^2$
  3. Minimize loss with `fminsearch`.

Implementation in MATLAB

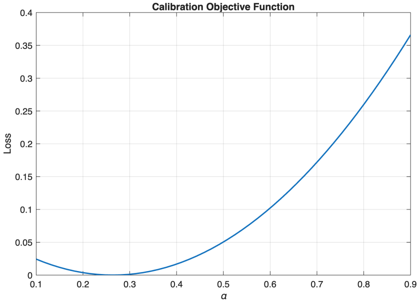
```
target = 0.3;

% Objective: difference between model and target
loss = @(alpha) (solve_lstar(alpha) - target)^2;

% Initial guess
alpha0 = 0.4;

% Minimize loss
alpha_star = fminsearch(loss, alpha0);

fprintf('Optimal␣alpha␣=␣%.4f\n', alpha_star);
```

Note

Here, `solve_lstar(alpha)` is a user-defined function that computes the optimal l^* for given α (using fmincon).

Visualizing the Loss Function

```
alpha = linspace(0.1,0.9,50);
for i = 1:length(alpha)
    L(i) = loss(alpha(i));
end
plot(alpha, L, 'LineWidth', 1.5)
xlabel('\alpha'); ylabel('Loss');
title('Calibration␣Objective␣Functi
grid on
```



Assessing Calibration Fit

- After calibration, verify:
  - Does $l^*(\alpha^*) \approx 0.3$?
  - Is the loss near zero?
  - Are results robust to initial guess?
- Visual check: plot model vs. data target.

Good Practice

Report both α^* and the resulting steady-state value l^*.

Economic Interpretation

- Higher α → stronger preference for consumption → higher labor supply.
- Calibration ensures model reproduces realistic steady-state behavior.
- Same principle applies to larger models:
  - Solow model: match savings rate s to observed growth.
  - RBC model: choose β, σ, α to match targets.

Takeaways

- Calibration = numerical optimization over parameters.
- Choice of objective function (loss) is crucial.
- Built-in solvers (`fminsearch`, `fmincon`) simplify the process.
- Always visualize and check fit.

Challenge: Calibration as Optimization

- Apply what we learned on optimization to an **economic calibration**.
- We will calibrate the **Solow model** savings rate s.
- Objective: make the model reproduce an observed (synthetic) output path y_t^{data}.

$$\min_s \sum_t \left[y_t^{model}(s) - y_t^{data} \right]^2$$

- Two parts:
  1. **In class:** manual grid search + optional `fminsearch`.
  2. **At home:** full calibration comparing several solvers.

The Solow Growth Model: Intuition

- A simple macro model describing how the economy accumulates capital over time.
- Output is produced using capital K_t and technology A_t:

$$Y_t = A_t K_t^{\alpha}, \quad 0 < \alpha < 1$$

- A fixed share of output, s, is saved and invested each period:

$$I_t = sY_t$$

- Capital depreciates at rate δ:

$$K_{t+1} = (1 - \delta)K_t + I_t$$

- The key parameter s (savings rate) determines how fast the economy grows.

Simulating the Solow Model

- Given parameters (α, δ, n, g) and initial values (K_0, A_0), we can simulate the model over time.

$$
\begin{aligned}
Y_t &= K_t^{\alpha} A_t^{1-\alpha} \\
K_{t+1} &= (1-\delta)K_t + sY_t \\
A_{t+1} &= (1+g)A_t
\end{aligned}
$$

- By varying s, we change the entire output path $\{Y_t\}$.
- Calibration: choose s so that simulated $\{Y_t^{model}(s)\}$ matches observed $\{Y_t^{data}\}$.

Key Idea

Optimization links theory (the Solow model) with data via parameter estimation.

Part 1 — In-Class Challenge (15 min)

Goal: Find the savings rate s that minimizes the distance between Solow model output and data.

1. Start with the provided starter file:

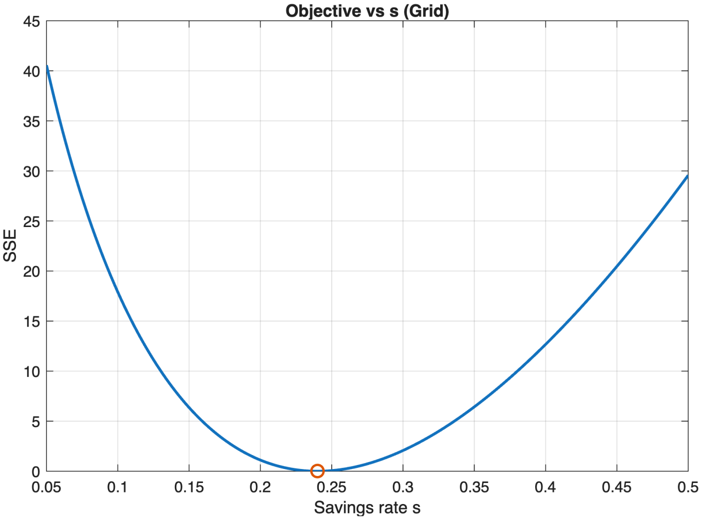
```
week6_challenge_starter.m
```

2. It uses the helper function:

```
solow_simulate(s, params, T)
```

3. Steps to implement:
   * Define the objective function: $\text{SSE}(s) = \sum_t \left(Y_t^{model}(s) - Y_t^{data}\right)^2$
   * Create a grid for $s \in [0.05, 0.5]$
   * Compute $\text{SSE}(s)$ for each value
   * Find and mark s^*
   * Plot the objective: SSE vs s
4. **Bonus:** use `fminsearch` (with re-parameterization to enforce $0 < s < 1$) to automate the search.

Expected Output (Visualization)



The objective (SSE) reaches its minimum near the true $s = 0.24$.

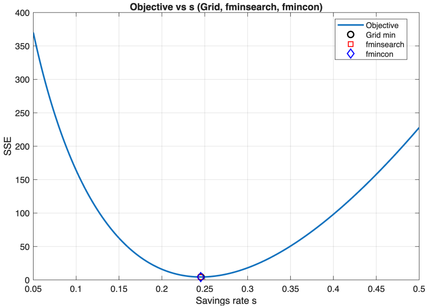
Part 2 — Homework: Solow Model Calibration

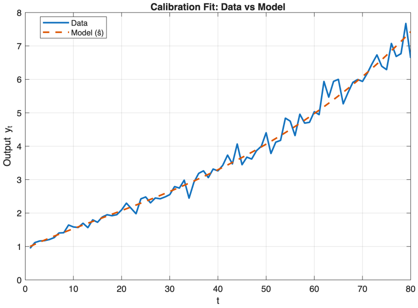
Goal: Compare manual search and built-in MATLAB optimization tools.

1. Calibrate s to match an observed or simulated y_t^{data}.
2. Compare results from:
   * Manual grid search
   * `fminsearch` (with re-parameterization to enforce $0 < s < 1$)
   * `fmincon` (with bounds $0 \leq s \leq 1$)
3. Produce and save:
   * A figure showing the objective function SSE vs s
   * A figure showing the fit of the model to the data

Homework Deliverables & Guidelines

- Submit your MATLAB code and generated figures.
- Include a short comment (2–3 lines) comparing methods:
  - Which solver is faster?
  - Which is more robust?
  - Do they give similar s^*?
- Your figures should look like the examples below (if you use the same calibration as in the challenge):





Save all figures in the `Figures/` folder.