Computational Tools for Macroeconomics using MATLAB

Week 5 – Solving Nonlinear Equations in Economics

Cristiano Cantore

Sapienza University of Rome

Learning Outcomes

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

1. Understand the difference between analytical and numerical solutions.
2. Implement root-finding algorithms (bisection, Newton–Raphson) in MATLAB.
3. Apply numerical methods to economic equilibrium problems.
4. Check for convergence and interpret stopping criteria.
5. Compare performance of different root-finding approaches.

Roadmap of Week 5

- Motivation: Why do we need numerical solutions?
- Bisection method – safe but slow.
- Newton–Raphson method – fast but risky.
- Applying both to economic equilibrium problems.
- Comparing speed and accuracy of convergence.

Motivation: Analytical vs. Numerical Solutions

- Many macroeconomic models involve **nonlinear equations** that cannot be solved analytically.
- Examples:
  - Equilibrium in goods and labor markets
  - First-order conditions from utility maximization
  - Steady-state equations in growth models
- Numerical methods allow us to find **approximate solutions** where algebra fails.
- This week: we study **root-finding algorithms** for $f(x) = 0$.

Examples of Nonlinear Equations in Economics

- **Consumption-saving problem:**

$$u'(c_t) = \beta(1 + r)u'(c_{t+1})$$

Nonlinear in c_t, r, and model parameters.

- **Labor–leisure choice:**

$$w(1 - l)^{-\sigma} = \lambda$$

Requires solving for l^* numerically.

- **Market equilibrium:**

$$D(p) - S(p) = 0$$

Often nonlinear in p.

MATLAB

- MATLAB provides a clean environment for **numerical iteration and visualization**.
- We can easily:
  - Write custom algorithms (bisection, Newton–Raphson)
  - Visualize convergence paths and errors
  - Compare performance of different methods
- We will code everything from scratch before using `fzero()` or `fsolve()`.

The Bisection Method: Intuition

- Goal: find x^* such that $f(x^*) = 0$.
- Start from two points a and b such that $f(a)$ and $f(b)$ have opposite signs.
- By the Intermediate Value Theorem:

$$f(a)f(b) < 0 \Rightarrow \exists\, x^* \in [a, b] \text{ with } f(x^*) = 0.$$

- The algorithm repeatedly halves the interval:

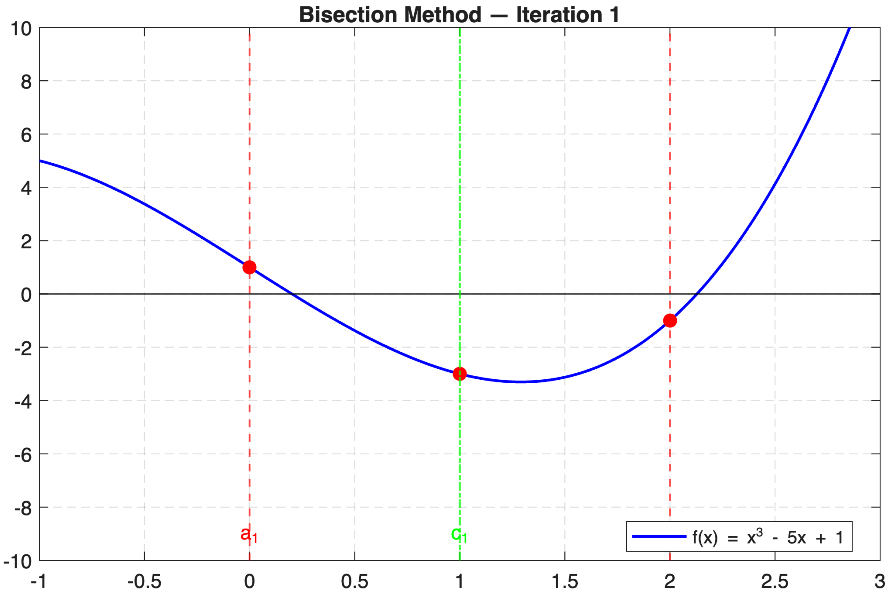
$$c = \frac{a+b}{2}.$$

- At each step, we replace one endpoint with c.

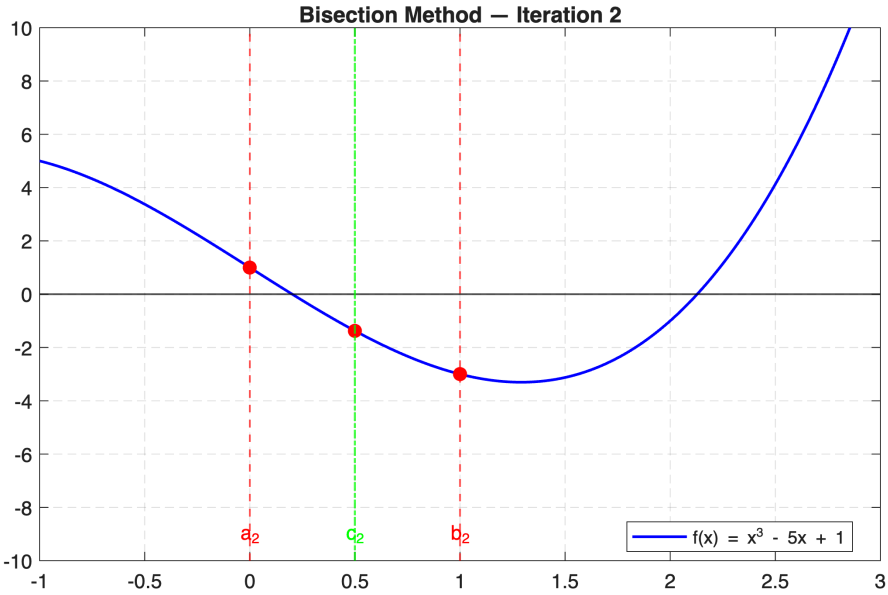
Graphical Idea

- Start with $f(a)$ and $f(b)$ of opposite signs.
- Compute midpoint c.
- Keep the subinterval that contains the sign change.
- Repeat until $|b - a| < \varepsilon$.

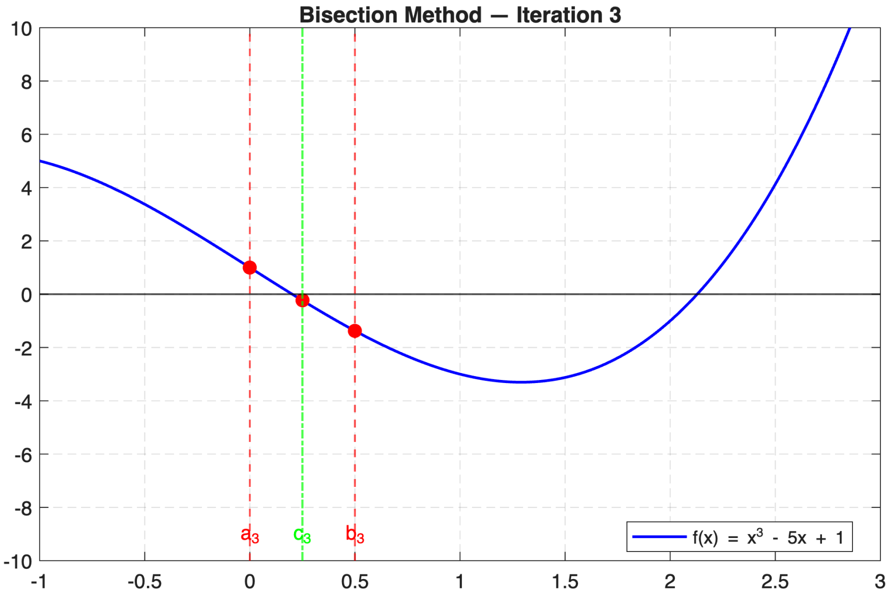
Graphical Idea



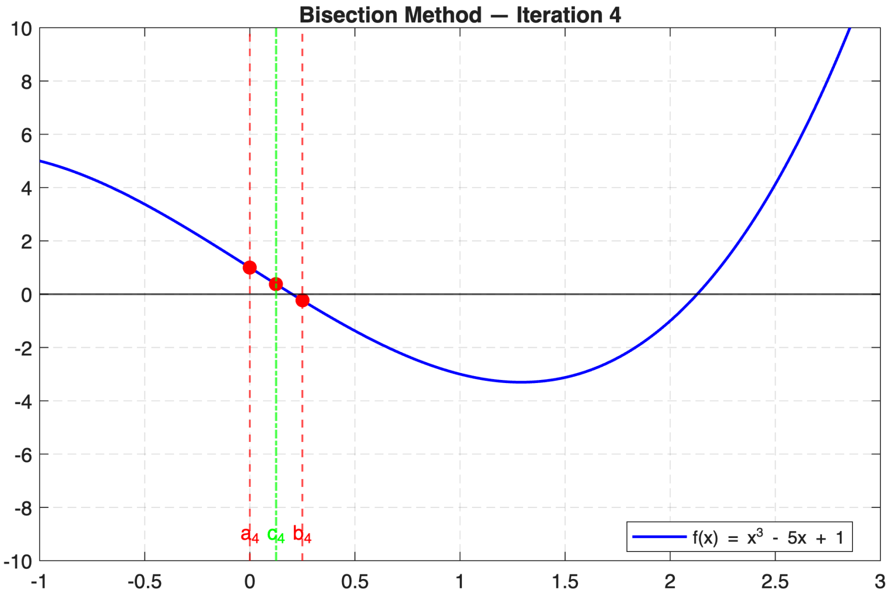
Graphical Idea



Graphical Idea



Graphical Idea



Algorithm Steps

1. Choose a, b with $f(a)f(b) < 0$.
2. Compute midpoint $c = (a + b)/2$.
3. If $f(c) = 0$ or $|b - a| < \varepsilon$, stop.
4. If $f(a)f(c) < 0$, set $b = c$; else $a = c$.
5. Repeat until convergence.

Bisection Method in MATLAB

```
% Define function
f = @(x) x.^3 - 5*x + 1;

% Initial bracket
a = 0; b = 2;
tol = 1e-4; maxit = 100;

for k = 1:maxit
    c = (a + b)/2;
    if abs(f(c)) < tol
        break
    elseif f(a)*f(c) < 0
        b = c;
    else
        a = c;
    end
end
```

Convergence and Features

- **Guaranteed convergence** if f continuous and sign change exists.
- **Linear convergence**: error shrinks roughly by half each iteration.

$$|x_{t+1} - x^*| \approx \frac{1}{2}|x_t - x^*|$$

- **Stopping criterion:**

$$|f(c)| < \varepsilon \quad \text{or} \quad |b - a| < \varepsilon$$

- Works even if $f'(x)$ unknown or discontinuous.

Example: Market Equilibrium

- Supply and demand functions:

$$D(p) = 20p^{-1.5}, \qquad S(p) = 2 + 3p.$$

- Equilibrium: $f(p) = D(p) - S(p) = 0$.
- Use bisection method to find p^*.

```
f = @(p) 20*p.^(-1.5) - (2 + 3*p);
a = 0.5; b = 5;
tol = 1e-4; maxit = 100;
```

Newton–Raphson: Idea

- Goal: solve $f(x) = 0$ by linearizing f around current guess x_t.
- Tangent-line update:

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}.$$

- **Intuition:** follow the tangent at $(x_t, f(x_t))$ down to the x-axis.
- **Pros:** very fast (quadratic) when close to the root.
- **Cons:** needs f' and a decent initial guess; can diverge.

Derivation via Taylor Expansion

- First-order Taylor around x_t: $f(x) \approx f(x_t) + f'(x_t)(x - x_t)$.
- Set $f(x_{t+1}) \approx 0$:

$$0 \approx f(x_t) + f'(x_t)\,(x_{t+1} - x_t) \;\Rightarrow\; x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}.$$

- Works for scalars and extends naturally to systems (Jacobian).

Algorithm (Pseudocode)

1. Choose initial guess x_0, tolerance ε, max iterations K.
2. For $t = 0, 1, \ldots, K$:
    2.1 Evaluate $f(x_t)$ and $f'(x_t)$.
    2.2 If $|f(x_t)| < \varepsilon$ (or $|x_t - x_{t-1}| < \varepsilon$), **stop**.
    2.3 Update $x_{t+1} = x_t - \dfrac{f(x_t)}{f'(x_t)}$.
    2.4 (Optional) *damping*: $x_{t+1} = x_t - \alpha \dfrac{f(x_t)}{f'(x_t)}$, $0 < \alpha \leq 1$.

Newton–Raphson in MATLAB (Scalar)

```
% f and derivative
f  = @(x) x.^3 - 5*x + 1;
fp = @(x) 3*x.^2 - 5;

x  = 1.5;                        % initial guess
tol = 1e-8; maxit = 50;

for it = 1:maxit
    fx  = f(x);
    fpx = fp(x);
    if abs(fx) < tol, break; end
    if abs(fpx) < 1e-12, warning('Derivative near zero'); break;
    x = x - fx/fpx;              % update (alpha=1)
end

fprintf('Root ~ %.10f (it=%d, |f|=%.1e)\n', x, it, abs(f(x)));
```

Newton with Damping (Safeguard)

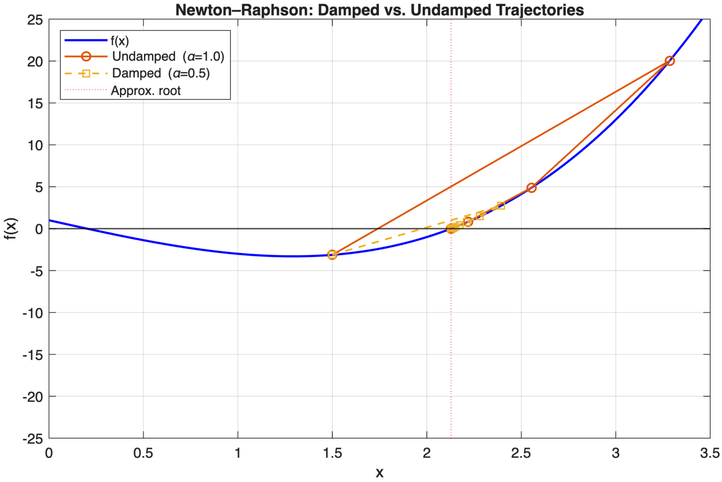
```
alpha = 0.5;                    % step size in (0,1] to stabilize
x = 1.5; tol = 1e-8; maxit = 50;

for it = 1:maxit
    fx  = f(x); fpx = fp(x);
    if abs(fx) < tol, break; end
    if abs(fpx) < 1e-12, warning('flat slope'); break; end
    xnew = x - alpha * fx / fpx;
    if ~isfinite(xnew), warning('bad update'); break; end
    x = xnew;
    fprintf('Root ~ %.10f (it=%d, |f|=%.1e)\n', x, it, abs(f(x))

end
```

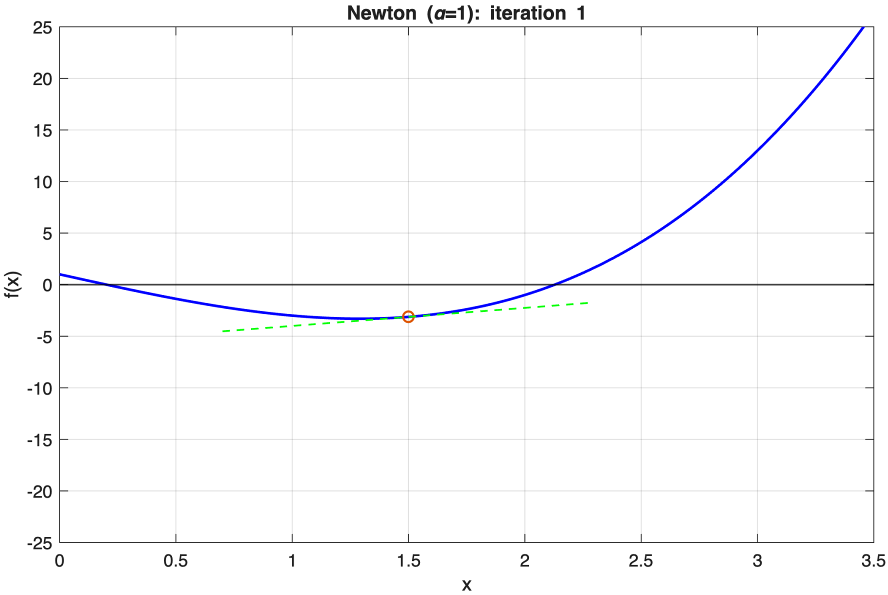
- **Why damping?** Avoids overshooting/divergence when slope is poor.
- Practical: backtracking (reduce α if $|f(x_{t+1})|$ not improving).

Geometric Illustration (Newton with/without damping)

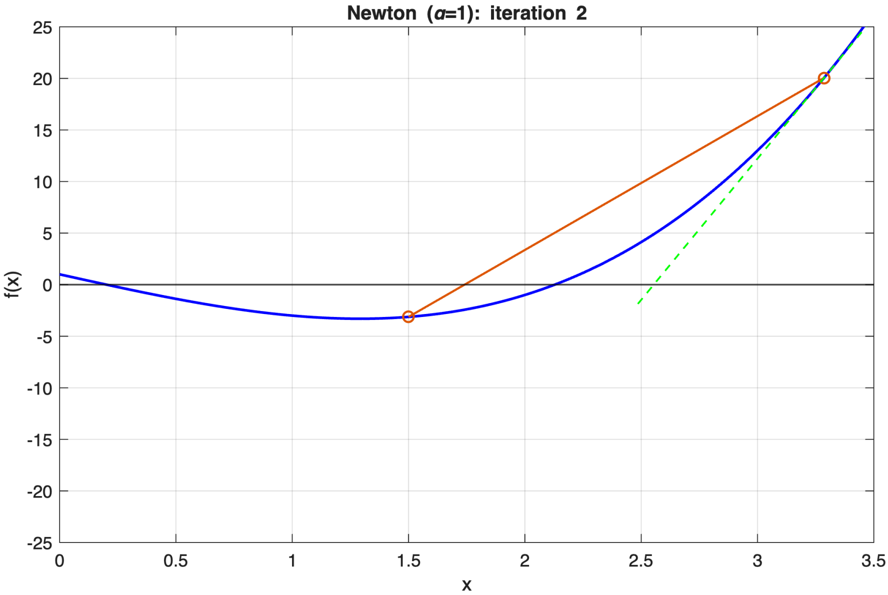


- At x_t, the tangent gives $x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$.
- **Undamped ($\alpha = 1$)**: large, efficient steps $\Rightarrow$ fast (quadratic) when close.
- **Damped ($0 < \alpha < 1$)**: smaller steps $\Rightarrow$ safer but slower (often linear far from root).

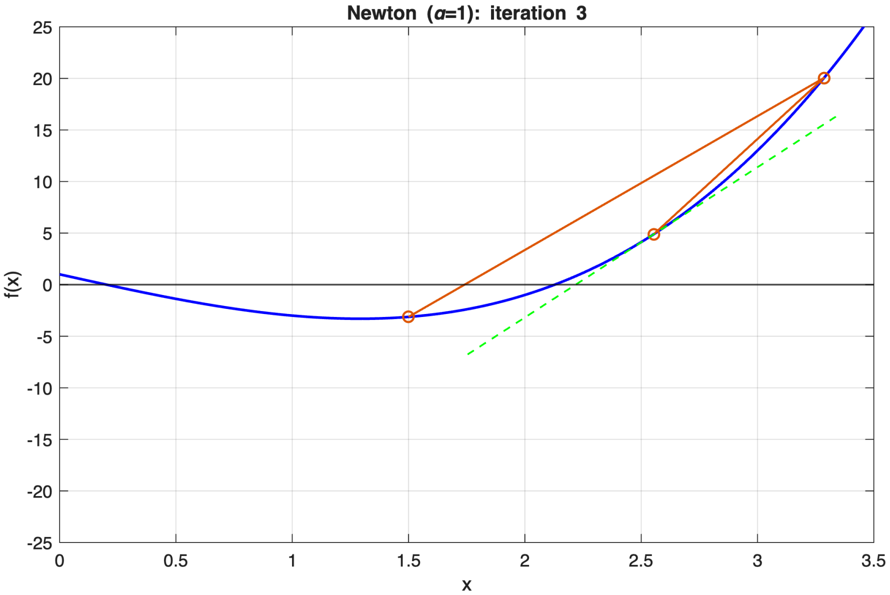
Newton (undamped, $\alpha = 1.0$) – Iteration 1



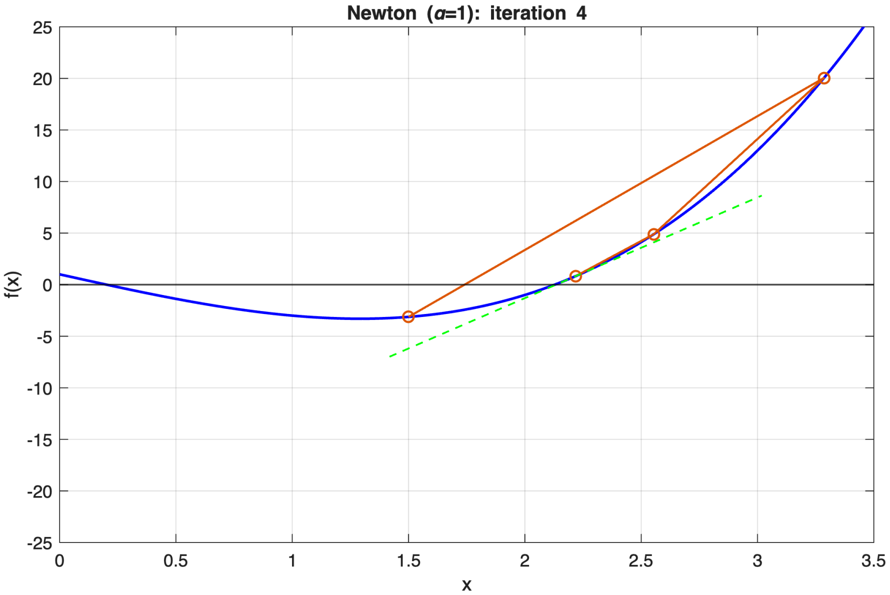
Newton (undamped, $\alpha = 1.0$) – Iteration 2



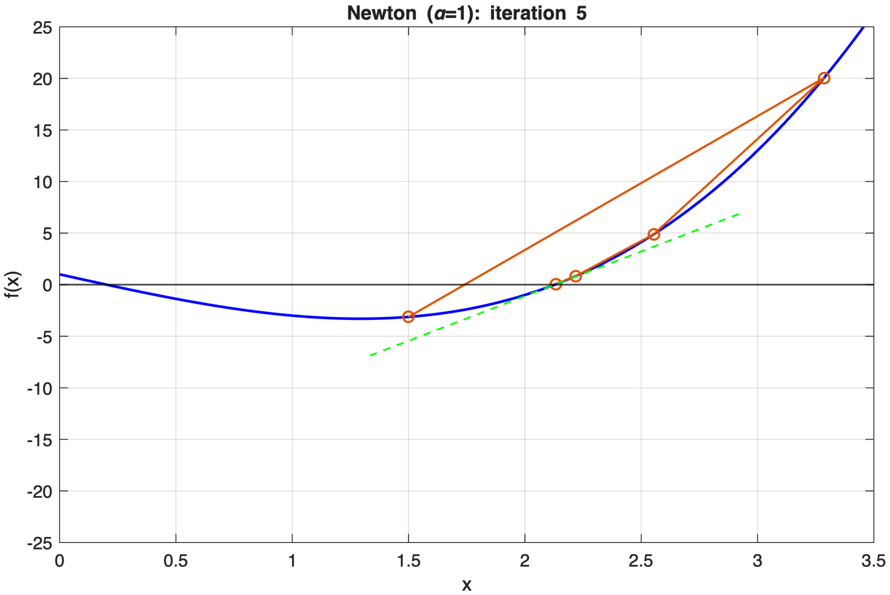
Newton (undamped, $\alpha = 1.0$) – Iteration 3



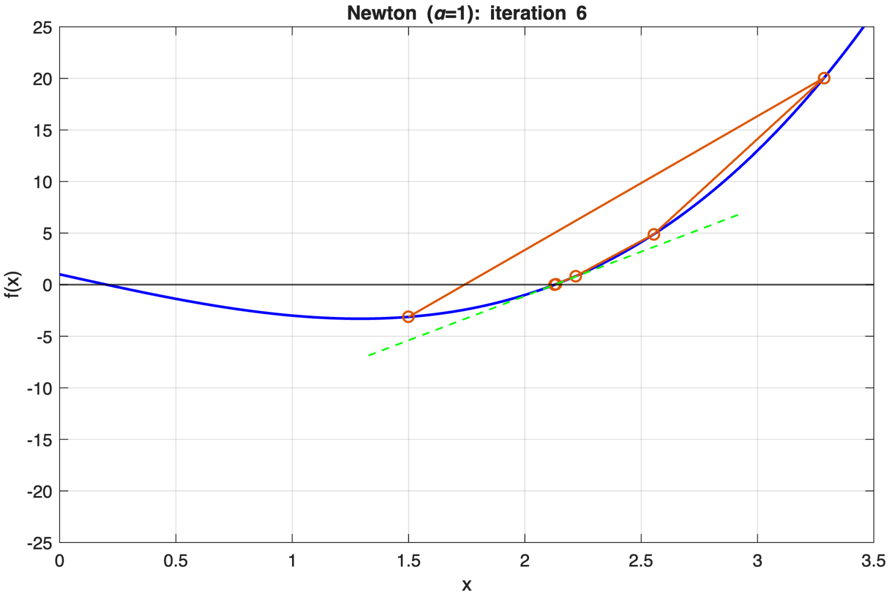
Newton (undamped, $\alpha = 1.0$) – Iteration 4



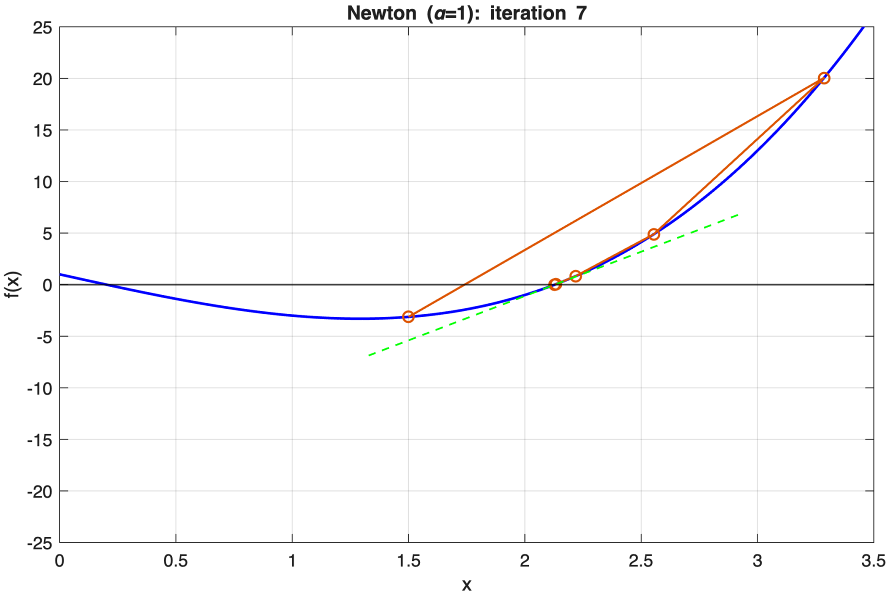
Newton (undamped, $\alpha = 1.0$) – Iteration 5



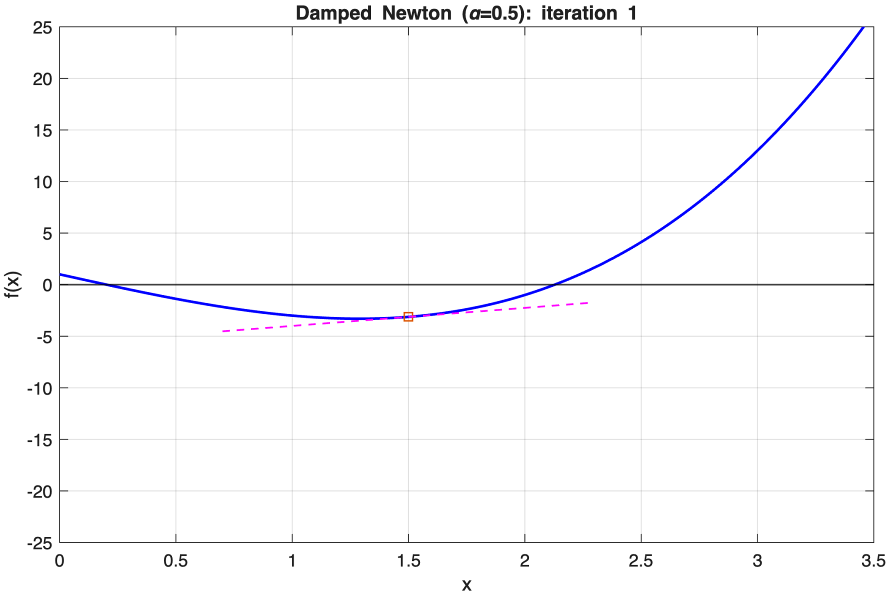
Newton (undamped, $\alpha = 1.0$) – Iteration 6



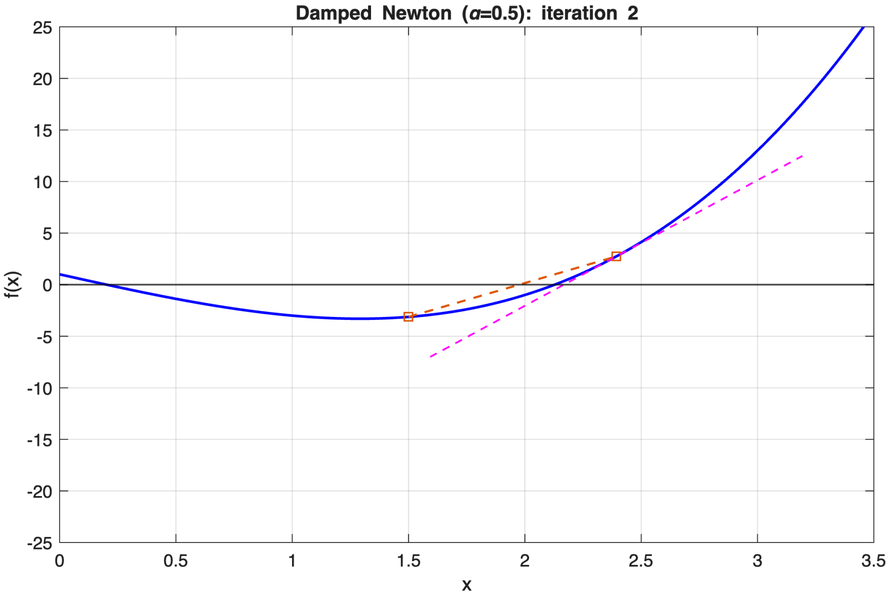
Newton (undamped, $\alpha = 1.0$) – Iteration 7



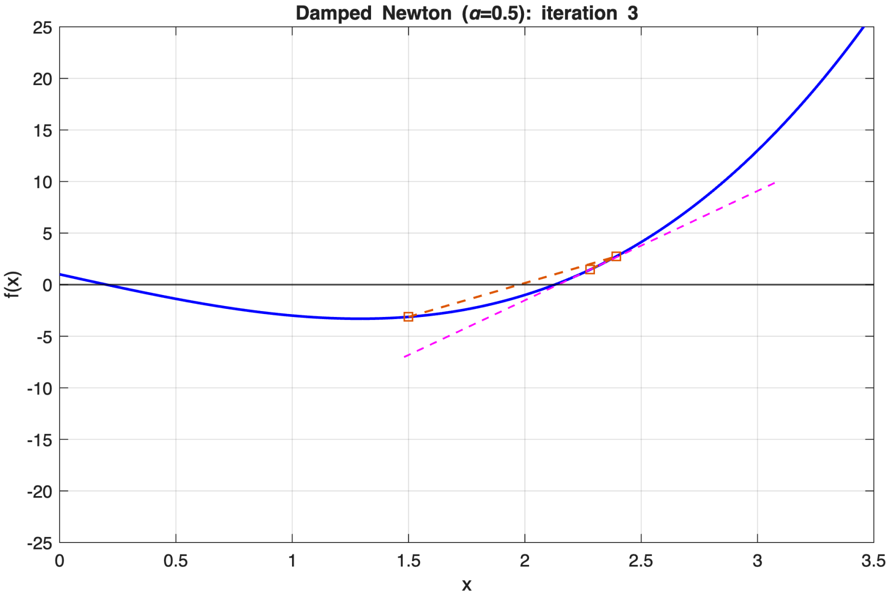
Newton (damped, $\alpha = 0.5$) – Iteration 1



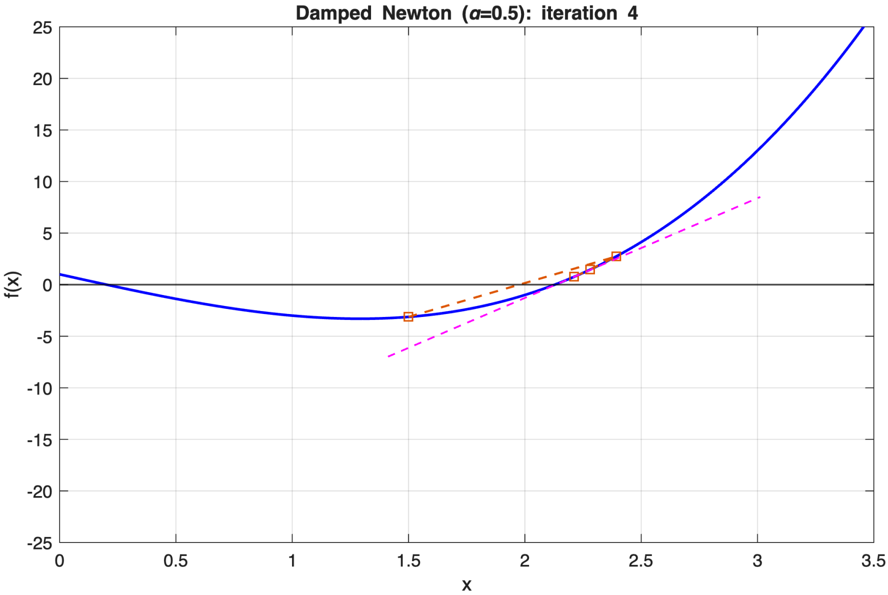
Newton (damped, $\alpha = 0.5$) – Iteration 2



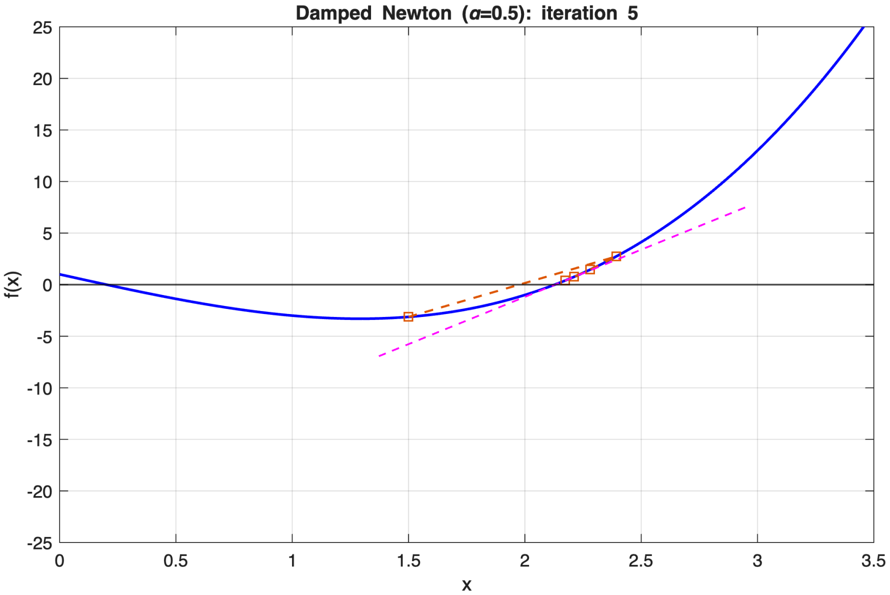
Newton (damped, $\alpha = 0.5$) – Iteration 3



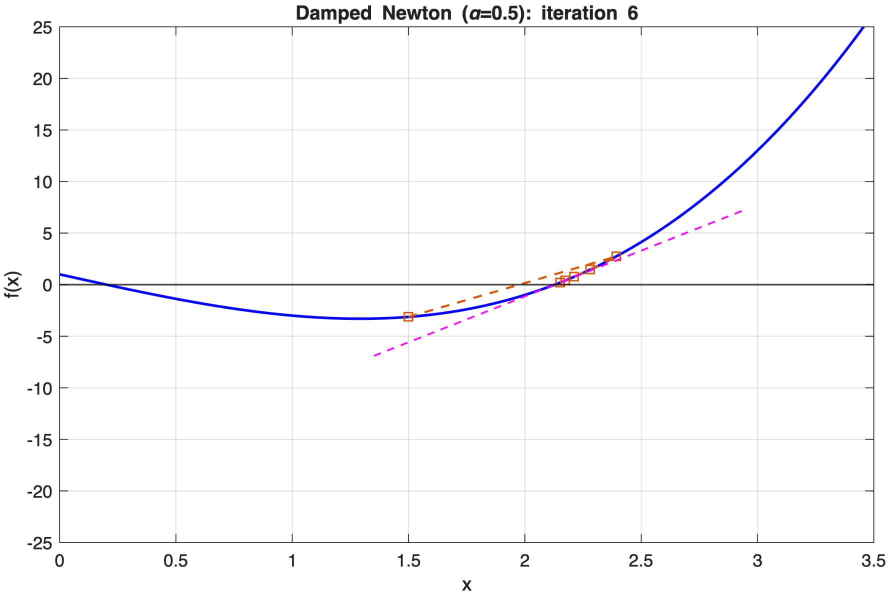
Newton (damped, $\alpha = 0.5$) – Iteration 4



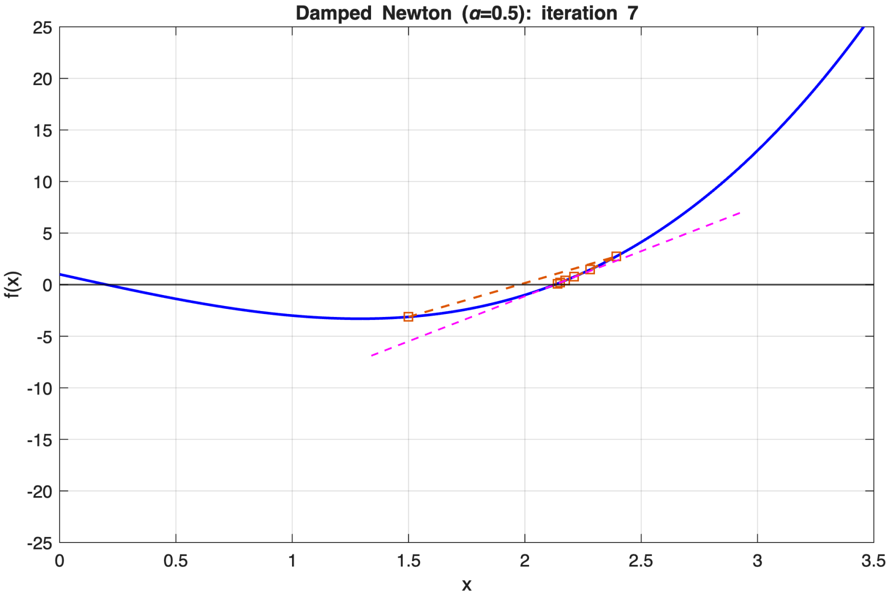
Newton (damped, $\alpha = 0.5$) – Iteration 5



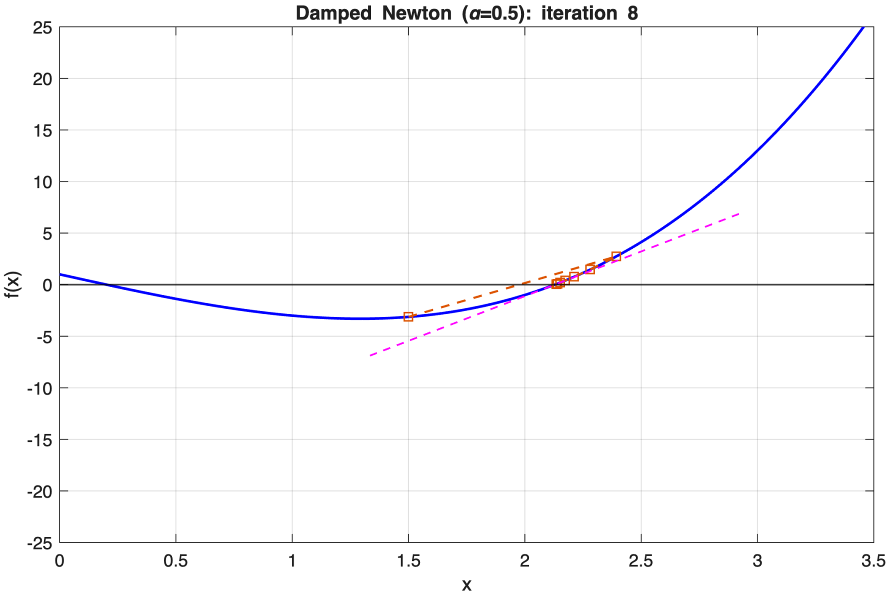
Newton (damped, $\alpha = 0.5$) – Iteration 6



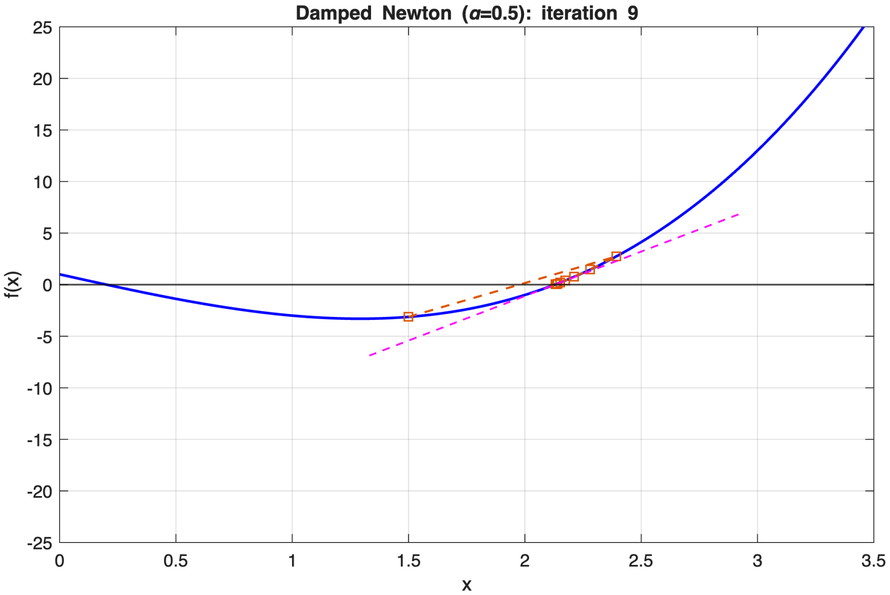
Newton (damped, $\alpha = 0.5$) – Iteration 7



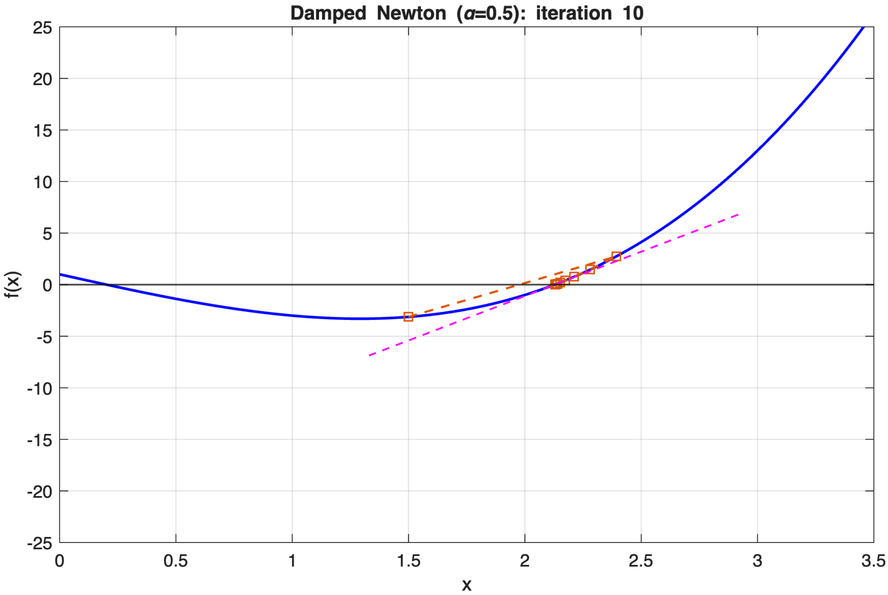
Newton (damped, $\alpha = 0.5$) – Iteration 8



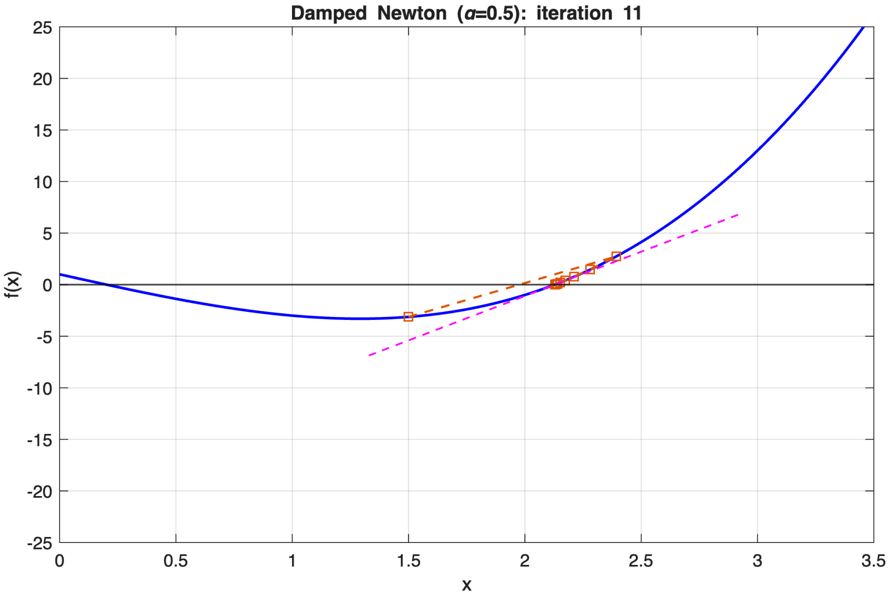
Newton (damped, $\alpha = 0.5$) – Iteration 9



Newton (damped, $\alpha = 0.5$) – Iteration 10



Newton (damped, $\alpha = 0.5$) – Iteration 11



Convergence Behavior

- **Quadratic** near the root: error roughly squares each step.
- Sensitive to initial guess; multiple roots $\Rightarrow$ different limits.
- Failure modes:
  - $f'(x_t) \approx 0$ (huge steps / NaN).
  - Jumps outside domain; non-smooth f.
  - Cycles if update keeps bouncing.
- Remedies: damping/backtracking, trust regions, switch to bisection if stuck.

Built-ins: `fzero`, `fsolve`

- `fzero(f,x0)`: scalar root finder (combines bracketing + Newton secant).
- `fsolve(F,x0)`: system of equations (requires Optimization Toolbox).
- Good practice: **code from scratch** first (learning), then validate with built-ins.

Examples

```
xstar = fzero(@(x) x.^3 - 5*x + 1, 1.5);

F = @(z) [ z(1)^2 + z(2) - 2;
           exp(z(1)) + z(2) - 3 ];
z0 = [1; 1];
zstar = fsolve(F, z0);
```

Newton: Practical Checklist

- Set **two** stopping rules: on $|f(x_t)|$ and $|x_t - x_{t-1}|$.
- Guard small derivatives: if $|f'(x_t)|$ tiny, damp or switch method.
- Keep iterates in domain (projection or backtracking).
- Limit step size and iterations; log diagnostics for debugging.
- If progress stalls, **fallback** to bisection on a bracket.

Bisection vs. Newton: A Comparison

- **Bisection:** Always converges, but slow (linear).
- **Newton:** Quadratic convergence near root, may diverge if guess is poor.
- **Trade-off:** Robustness vs. speed.
- Combine: start with bisection, switch to Newton once bracket small.

Systems of Equations and the Jacobian

For $F(x) = 0$, $x \in \mathbb{R}^n$, Newton's method becomes

$$x_{t+1} = x_t - J(x_t)^{-1} F(x_t)$$

- $J(x_t)$ = Jacobian matrix of partial derivatives.
- Use `fsolve(@(x)F(x),x0)` in MATLAB.

```
F = @(z) [z(1)^2 + z(2) - 2;
              exp(z(1)) + z(2) - 3];
z0 = [1;1];
zstar = fsolve(F,z0);
```

Numerical Derivatives (if f' unknown)

```
df = @(f,x,h) (f(x+h)-f(x-h)) / (2*h);
```

- Centered difference $\approx f'(x)$.
- Use small h (e.g. 10^{-6}) but beware of rounding errors.

Economic Application: Labor Market Equilibrium

- Consider a representative household choosing hours $h \in (0, 1)$ to maximize utility in consumption c and leisure $\ell = 1 - h$.
- The utility function is given by $U(c) = V(\ell)$.
- **Budget constraint:**

$$c = (1 - \tau)wh + a - g,$$

  where w is the exogenous wage, τ the tax rate, a lump-sum transfers, and g government spending.
- **First-order condition:**

$$U_c(c)\,(1 - \tau)w = V_\ell(1 - h).$$

- This defines a nonlinear equation in h, which we will solve numerically.

Equilibrium Condition (Scalar Nonlinear Equation)

$$Z(h) \equiv U_c(c(h))\,(1-\tau)w \;-\; V_\ell(1-h) = 0,$$

where $c(h) = (1-\tau)wh + a - g$.

- The equilibrium labor supply h^* satisfies $Z(h^*) = 0$.
- Once h^* is found, compute $c^* = (1-\tau)wh^* + a - g$.
- This is a simple yet realistic example of solving an economic equilibrium as a root-finding problem.

Functional Forms for the Demo

- **Preferences:** CRRA in consumption and separable disutility of work:

$$U(c) = \frac{c^{1-\sigma} - 1}{1-\sigma}, \quad U_c(c) = c^{-\sigma}, \qquad V(\ell) = \chi \frac{\ell^{1+\phi}}{1+\phi}, \quad V_\ell(\ell) = \chi \ell^{\phi}.$$

- Substituting into the first-order condition:

$$Z(h) = \big((1-\tau)wh + a - g\big)^{-\sigma} (1-\tau)w - \chi (1-h)^{\phi}.$$

- We solve $Z(h) = 0$ for $h \in (0, 1)$ numerically.

Calibration

- Parameters ensuring a single interior root:

$$w = 1, \quad \tau = 0, \quad a = 0.10, \quad g = 0, \quad \sigma = 2, \quad \phi = 2, \quad \chi = 100.$$

- Interpretation:
  - High χ guarantees a meaningful disutility of work.
  - The root lies roughly around $h^* \approx 0.3$–0.5.
  - The equation can be solved with both Bisection and Damped Newton.

Optimal hours h^*

```
w=1; a=0.5; sigma=2; chi=3.7; phi=2; tau=0; g=0;
Z  = @(h) ((1-tau)*w*h + a-g).^(-sigma)...
.*(1-tau)*w - chi*(1-h).^phi;
dZ = @(h) -sigma*((1-tau)*w*h + a - g).^(-sigma-1)...
.*(1-tau)*w + chi*phi*(1-h).^(phi-1);

h = 0.5; alpha = 0.5; tol = 1e-8;
for it=1:60
    Zh = Z(h); if abs(Zh) < tol, break; end
    dZh = dZ(h); if abs(dZh) < 1e-10, break; end
    h = h - alpha * Zh/dZh;               % damped Newton
    h = min(max(h, 0+1e-8), 1-1e-8);   % project to (0,1)
end
fprintf('h* = %.6f, Z=%.1e, it=%d\n', h, Z(h), it);
```

Optimal hours h^*

- Interpret the results.

Week 5 Challenge — Labor Market Equilibrium

- Now calibrate $\tau = 0.2$ instead.
- Study equilibrium labor supply h^* from the condition

$$Z(h) = U_c(c(h))\,(1 - \tau)w - V_\ell(1 - h) = 0.$$

- Given the parameters above, solve numerically for h^*.
- Compare methods:
  1. Bisection on $[10^{-6}, 1 - 10^{-6}]$.
  2. Damped Newton ($\alpha = 0.5$) starting from $h_0 = 0.5$.
- Report h^*, number of iterations, and $|Z(h^*)|$ for each method.

Part 1 – Compare Bisection and Newton

```
% Parameters
w = 1; tau = 0.20; a = 0.50; g = 0;
sigma = 2; phi = 2; chi = 3.7;

% Function Z(h)
c = @(h) (1 - tau) * w .* h + a - g;
Z = @(h) c(h).^(-sigma) * (1 - tau) * w - chi * (1 - h).^phi;

% Solve with both methods and print results
```

- Verify convergence and the sign of $Z(h)$ near the root.
- Compare speed and accuracy.
- How does the tax rate affect the equilibrium labor supply?

Part 2 – Comparative Statics and Plots

- Loop over $\tau \in [0, 0.4]$ (e.g. 9 values).
- For each τ, solve $Z(h) = 0$ by bisection and record $h^*(\tau)$.
- Plot equilibrium hours as a function of τ.

Discussion – Interpreting Your Results

- Check that $Z(h^*) \approx 0$ for all solutions.
- Compare iteration counts between methods.
- Examine and interpret $h^*(\tau)$.

Week 5 Homework: Solving Nonlinear Equations

- The homework contains three exercises:
  1. Nonlinear IS–LM model (system of equations).
  2. Labor supply problem with varying preferences.
  3. Convergence speed comparison.

Exercise 1 — Nonlinear IS–LM Equilibrium

- Consider the IS–LM system:

$$\begin{aligned} Y &= C_0 + c(Y - T) + I_0 - \beta i^2 + G, \\ M/P &= kY - \lambda i. \end{aligned}$$

- Parameters: $C_0 = 50$, $c = 0.8$, $I_0 = 60$, $\beta = 2$, $T = 50$, $G = 100$, $M/P = 400$, $k = 0.5$, $\lambda = 20$.
- Solve for (Y^*, i^*) using `fsolve` (or your own Newton routine), starting from two guesses.
- Plot the IS and LM curves and verify that the equilibrium lies on the **positive-interest branch**.

Exercise 2 — Labor Supply and Risk Aversion

- Revisit the labor supply condition:

$$Z(h) = c(h)^{-\sigma}(1-\tau)w - \chi(1-h)^{\phi} = 0, \quad c(h) = (1-\tau)wh + a - g.$$

- Baseline: $w = 1$, $\tau = 0.2$, $a = 0.5$, $g = 0$, $\phi = 2$, $\chi = 3.7$.
- Vary $\sigma \in \{1, 2, 3, 4, 5\}$ and solve for $h^*(\sigma)$ using the **Bisection method**.
- Plot $h^*(\sigma)$ and interpret:
  - How does greater risk aversion affect labor supply?
  - Does h^* decline as σ rises?

Exercise 3 — Comparing Convergence Speed

- Using the baseline $Z(h)$, solve for h^* with:
  1. **Bisection** (full bracket, $[10^{-6}, 1-10^{-6}]$),
  2. **Damped Newton** ($\alpha = 0.5$) from initial guesses $h_0 = 0.2, 0.5, 0.8$.
- Record:

$$\text{Method, Starting guess, Iterations}, h^*, |Z(h^*)|.$$

- Discuss:
  - When does Newton outperform Bisection?
  - Under what conditions could it fail?

Submission Guidelines

- Submit one script named `week5_homework_solution.m` and any plots generated.
- Include:
  1. Clear comments for each exercise.
  2. Printed results for h^*, Y^*, i^*, iteration counts.
  3. Plots for IS–LM equilibrium and Labor Supply comparative statics.
- Deadline: before the Week 6 session.