Week 3 — Unconstrained Optimisation II

Gradient Descent & Newton's Method

Dr Juan Zurita

Optimisation & Mathematical Methods for Economics
The University of Edinburgh · School of Economics

Gradient Descent

Algorithm: $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k)$

- $\alpha > 0$: step size (learning rate)
- Follows the steepest descent direction
- **Linear** convergence rate
- Simple but can be slow on ill-conditioned problems

Newton's Method

Uses second-order information:
$\mathbf{x}_{k+1} = \mathbf{x}_k - [H_f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)$

- **Quadratic** convergence near optimum
- Each step is more expensive (Hessian computation)
- May diverge if started far from optimum

Quasi-Newton: BFGS

Idea: approximate the inverse Hessian from gradient differences.

- No need to compute or invert the Hessian
- **Superlinear** convergence
- L-BFGS: limited-memory variant for large problems
- The workhorse of smooth optimisation in practice

In Python: `scipy.optimize.minimize(method='BFGS')`

Convergence Comparison

Method	Convergence	Cost per step
Gradient descent	Linear	$O(n)$
Newton	Quadratic	$O(n^3)$
BFGS	Superlinear	$O(n^2)$

Choose based on problem size and smoothness.

Summary

- GD: simple, slow; Newton: fast, expensive; BFGS: best of both
- Step size matters — too large diverges, too small crawls
- `scipy.optimize` provides production-quality implementations
- **Next week:** constrained optimisation with Lagrange multipliers