6. Root finding in higher dimensions

Last time

- Convergence of iterative methods
- Newton method

Goals for today

- Feedback from problem set 1
- Solving **systems of nonlinear equations**
- Review of derivatives for functions $\mathbf{f}: \mathbb{R}^n \to \mathbb{R}^n$
- Newton in higher dimensions

Feedback from problem set 1

Getting help

- **Please ask for help if something doesn’t work**
- Especially for stupid technical issues
- And also for “how do I do this in Julia” questions
- And if you’re stuck after thinking for a while
- After class, office hours, TA, Piazza, email

Logistics

- Easiest for graders: submit a single PDF file
- Produce PDF from Jupyter:
  - `File -> Print Preview`
  - `Print` and save to PDF
- Submit Jupyter notebook too just in case
- **Check** the PDF and make sure that everything came out OK; fix it if it didn't!

Logistics II

- When interactive display is requested, add an extra cell with a representative plot that displays in the PDF
- If WebIO doesn't work, follow instructions in `installation.md`

Equations

- Equations should be LaTeXed in Jupyter notebook (preferably)
- Enclose in `$...$` (inline) or `$$...$$` (displayed)
- Or in separate LaTeX file
- Or written by hand on paper or tablet and included into PDF file
- Please do not write equations in plain text since they are unreadable

Tips

- When derivations and operation counting are asked for, assume that they refer to the general algorithm, as a function of n (some measure of size of problem)
- Comparisons should be done via plots, not only verbal descriptions
- Always comment on what your code shows; e.g. what does it **mean** if you get an `OverflowError`? Don't just show that it happens, comment on it.
- Please explicitly label your problem numbers with headings (e.g. `# Exercise 1` in Jupyter)
- In the future a notebook version of the psets will be available

Solving systems of nonlinear equations

- How can we solve a **system** of nonlinear equations
- e.g. 2 equations in 2 unknowns:

$$f(x, y) := x^2 + y^2 - 3 = 0$$
$$g(x, y) := \left(\frac{x}{2}\right)^2 + (y - 0.5)^2 - 1 = 0$$

- How can we solve a **system** of nonlinear equations
- e.g. 2 equations in 2 unknowns:

$$f(x, y) := x^2 + y^2 - 3 = 0$$
$$g(x, y) := \left(\frac{x}{2}\right)^2 + (y - 0.5)^2 - 1 = 0$$

- What does solution set of $f(x, y) = 0$ look like?

Systems of equations II

- $f(x, y) = 0$ usually gives a **curve**
- Called a **level set**: set $\{(x, y) : f(x, y) = c\}$
- How to draw
- Want *joint* roots $f(x^*, y^*) = g(x^*, y^*) = 0$
- **Intersection** points of curves: significantly harder than 1D

Systems of equations II

- $f(x, y) = 0$ usually gives a **curve**
- Called a **level set**: set $\{(x, y) : f(x, y) = c\}$
- How to draw
- Want *joint* roots $f(x^*, y^*) = g(x^*, y^*) = 0$
- **Intersection** points of curves: significantly harder than 1D
- Multidimensional bisection or interval arithmetic

Vector form

- Rewrite system of equations into vector form:
- Write $f_1 = f; f_2 = g; x_1 = x; x_2 = y$
- Get system $f_i(x_1, \ldots, x_n) = 0$

Vector form

- Rewrite system of equations into vector form:
- Write $f_1 = f; f_2 = g; x_1 = x; x_2 = y$
- Get system $f_i(x_1, \dots, x_n) = 0$
- Write vectors $\mathbf{x} = (x_1, \dots, x_n)$ and $\mathbf{f} = (f_1, \dots, f_n)$
- Vector form: $\mathbf{f}(\mathbf{x}) = \mathbf{0}$

Multidimensional Newton

- Idea: Apply same technique as for 1D Newton

Multidimensional Newton

- Idea: Apply same technique as for 1D Newton
- Initial guess $\mathbf{x}_0$; try to solve $\mathbf{f}(\mathbf{x}_n + \boldsymbol{\delta}) = \mathbf{0}$

Multidimensional Newton

- Idea: Apply same technique as for 1D Newton
- Initial guess $\mathbf{x}_0$; try to solve $\mathbf{f}(\mathbf{x}_n + \boldsymbol{\delta}) = \mathbf{0}$
- Need **Taylor expansion for higher dimensions**:

$$\mathbf{f}(\mathbf{a} + \boldsymbol{\delta}) = \mathbf{f}(\mathbf{a}) + \mathsf{D}\mathbf{f}(\mathbf{a}) \cdot \boldsymbol{\delta} + \mathcal{O}(\|\boldsymbol{\delta}\|^2)$$

Multidimensional Newton

- Idea: Apply same technique as for 1D Newton
- Initial guess $\mathbf{x}_0$; try to solve $\mathbf{f}(\mathbf{x}_n + \boldsymbol{\delta}) = \mathbf{0}$
- Need **Taylor expansion for higher dimensions**:

$$\mathbf{f}(\mathbf{a} + \boldsymbol{\delta}) = \mathbf{f}(\mathbf{a}) + \mathsf{D}\mathbf{f}(\mathbf{a}) \cdot \boldsymbol{\delta} + \mathcal{O}(\|\boldsymbol{\delta}\|^2)$$

- So $\mathbf{f}(\mathbf{x}_n + \boldsymbol{\delta}) \simeq \mathbf{f}(\mathbf{x}_n) + \mathsf{J}(\mathbf{x}_n) \cdot \boldsymbol{\delta}_n$

Multidimensional Newton

- Idea: Apply same technique as for 1D Newton
- Initial guess $\mathbf{x}_0$; try to solve $\mathbf{f}(\mathbf{x}_n + \boldsymbol{\delta}) = \mathbf{0}$
- Need **Taylor expansion for higher dimensions**:

$$\mathbf{f}(\mathbf{a} + \boldsymbol{\delta}) = \mathbf{f}(\mathbf{a}) + \mathsf{D}\mathbf{f}(\mathbf{a}) \cdot \boldsymbol{\delta} + \mathcal{O}(\|\boldsymbol{\delta}\|^2)$$

- So $\mathbf{f}(\mathbf{x}_n + \boldsymbol{\delta}) \simeq \mathbf{f}(\mathbf{x}_n) + \mathsf{J}(\mathbf{x}_n) \cdot \boldsymbol{\delta}_n$
- Need to solve $\boldsymbol{\delta}_n = -\mathsf{J}(\mathbf{x}_n)^{-1}\mathbf{f}(\mathbf{x}_n)$
- $\mathsf{J} := \mathsf{D}\mathbf{f}(\mathbf{x}_n)$ is **Jacobian matrix** of all partial derivatives

Multidimensional Newton II

- Reduced nonlinear system to **linear system**
- Mathematics: calculate **matrix inverse**

Multidimensional Newton II

- Reduced nonlinear system to **linear system**
- Mathematics: calculate **matrix inverse**
- Numerics: instead **solve linear system** (see later)

Multidimensional Newton II

- Reduced nonlinear system to **linear system**
- Mathematics: calculate **matrix inverse**
- Numerics: instead **solve linear system** (see later)
- For now, use Julia's linear system solver, written `\` ("backslash")
- "Magic" black box
- Type of "matrix division"

Solving linear systems in Julia

- To solve linear system $A \cdot \mathbf{x} = \mathbf{b}$ in Julia:

```
using LinearAlgebra   # standard library; no installation re

A = rand(2, 2)    # random matrix
b = rand(2)       # random vector

x = A \ b

residual = (A * x) - b
```

- `A * x` is standard matrix–vector multiplication
- `A \ b` is a **black box** that we will open up later in the course

Summary

- Proved convergence of iterative methods
- Viewed Newton method as a fixed-point iteration
- Secant method to avoid calculating derivative ("derivative-free")
- Newton method in higher dimensions
- Hints that we need **interpolation** and **linear algebra**