# 10. Algorithmic differentiation and the Newton method

# Last time

- Linear regression and intro to machine learning
- Derivatives
- Algorithmic differentiation

# Today

- Tips from Problem Set 3
- Review of algorithmic differentiation
- Newton method for finding roots
- Higher dimensions

# Tips from Problem Set 3

# Style

- Spaces around `=` and operators, and after comma
- Blank lines separating conceptually different blocks of code
- Function names: *no* capitals; types: each word capitalised
- Use names of enums instead of converting to `Int`
- Do not use abbreviated names like `sta` or `stt` for `status`
- e.g. `possible_locations` instead of `locpos`

# Style II

- If calculating a Boolean condition, *don't* do e.g.

```
if a < 0
    return true
else
    return false
end
```

- Just do `return a < 0`
- Don't use "magic numbers" like `dynamics!(L, 0.70, 0.01, new_list_agents_2, 100, 100)`
- Give those `0.70` and `100` a *name*

# Julia tips

- Don't need `Pkg.add` each time – once only to install the package
- Do need `using` each time
- If have numerical values, try to avoid `if` looping over all the different values
- Subtypes of `AbstractWalker2D` are not necessarily mutable

## PS3 Q.6

- `initialize` function is “irrelevant” to computational complexity
- Run once so cost “amortised” if run simulation for a long time
- Expensive part is looping over all walkers to look for collisions
- So store locations of walkers in a `Dict` or `Matrix`
- `Matrix` is twice as fast (?)
- But need to keep this data structure updated as walkers move

# Algorithmic differentiation

- Recall: Want to calculate derivatives exactly and automatically
- By following rules for combining derivatives
- E.g. $(f \cdot g)'(a) = f(a)g'(a) + f'(a)g(a)$
- For each function $f$ need pair $(f(a), f'(a))$

# Implementation

- Defined new "dual number" type:

```
struct Dual
    value::Float64
    deriv::Float64
end

f = Dual(3, 4)
```

- `f` is an object representing a function $f$ with $(f(a), f'(a)) = (3, 4)$
- Usual *not* to explicitly represent evaluation point $a$ in dual numbers

# Arithmetic

- Define getters

```
val(f::Dual) = f.value
der(f::Dual) = f.deriv
```

- And arithmetic operations on that type:

```
import Base: *

*(f::Dual, g::Dual) = Dual(val(f) * val(g),
                           val(f) * der(g), der(f) * val(g))
```

# Meaning of dual number

- Approximation of function near given point $a$
- `Dual(c, d)` is function that looks like $c + \epsilon d$
- $\epsilon = x - a$ is distance from $a$
- Represents function $f$ with $f(a) = c$ and $f'(a) = d$
- As the pair $(f(a), f'(a))$

# Interpretation of dual number

- “Where you are and how fast you’re moving”
- Tangent line
- Polynomial of order 1 (affine function)

# Applying functions to dual numbers

- Suppose $g(x)$ is a given function
- What happens if apply $g$ to a dual number $c + d\epsilon$?
- $g(c + d\epsilon) = g(c) + \epsilon \cdot g'(c) \cdot d$
- In particular,

$$g(a + \epsilon) = g(a) + \epsilon g'(a)$$

- So pass in $a + \epsilon$, i.e. `Dual(a, 1)`, to calculate derivative
- Derivative is coefficient of $\epsilon$

# Functions of dual numbers

- Suspose `f = Dual(c, d)` represents function $f(x)$ near $a$
- Then e.g. `sin(f)` should represent $g(x) = \sin(f(x))$ near $a$
- Value $g(a) = \sin(f(a))$
- Derivative $g'(a) = \sin'(f(a)) \cdot f'(a)$ by chain rule
- Code:

```
Base.sin(f::Dual) = Dual(sin(val(f)),
                         cos(val(f)) * der(f))
```

# Chain rule

- Suppose $g(x)$ is a given function
- What happens if apply $g$ to a dual number $c + d\epsilon$ representing $f$?
- Have $c = f(a)$ and $d = f'(a)$
- So $g(c + d\epsilon) = g(f(a)) + \epsilon\, g'(f(a)) \cdot f'(a)$
- Chain rule is *automatically encoded* in derivative of $g$

# Finding roots using the Newton method

# Roots

- Often want to solve $f(x) = 0$ for **nonlinear** function $f$
- **Root**: Solution $x^*$ where $f(x^*) = 0$ (or **zero**)
- General nonlinear equation $f(x) = 0$ cannot be solved exactly
- E.g. polynomials of degree $\geq 5$
- But we still want to find roots (numerically)!
- How could we do this?

# Iterative methods

- Idea: Look for a discrete-time recurrence $x_{n+1} = \alpha(x_n)$
- Start from initial guess $x_0$
- Want sequence $x_0$, $x_1$, ... with $x_n \to x^*$ as $n \to \infty$.
- Many possible choices of algorithms $\alpha$
- We will look at **Newton method** – uses derivative $f'$
- How?

# Newton(–Raphson) method

- Draw picture
- Start from $(x_0, f(x_0))$
- Follow tangent line down
- Intersect it with $x$-axis to give new guess $x_1$
- Repeat

# Derivation of Newton method

- Want to find $x_1 = x_0 + \delta$ (defines $\delta$)
- Solve $f(x_1) = f(x_0 + \delta) = 0$
- Still nonlinear so *linearize*:
- Taylor expand to linear order in $\delta$:

$$f(x_0 + \delta) \simeq f(x_0) + \delta f'(x_0)$$

- Have

$$f(x_1) = f(x_0 + \delta) \simeq f(x_0) + \delta f'(x_0)$$

- So replace $f(x_1) = 0$ with approximation

$$f(x_0) + \delta f'(x_0) \simeq 0$$

- Gives $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$
- In general get recurrence

$$\boxed{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}}$$

# Implementation

- Implement:

```
function newton(f, f', x0, N=20)
    x = x0
    xs = [x0]

    for i in 1:N
        x = x - f(x) / f'(x)
        push!(xs, x)
    end

    return xs
end
```

# Convergence

- Newton method *does not always converge*
- But when it does, it converges “fast” – how fast?
- Is there a difference if use numerical or exact derivative?
- Numerical derivatives give something like secant method
- Newton is “better” but each step may be more expensive

# Optimization

- Can use Newton or related methods to find minima
- How?
- What does this need?

# Optimization II

- Solve $f'(x) = 0$
- So need derivatives of the derivative, i.e. *2nd derivatives*
- Can also calculate automatically

# Higher dimensions

- What happens for higher-dimensional functions
- E.g. $f(x, y) = x^2 + y^2 - 1$
- Generalise approach from 1D
- Pass in dual numbers with *same* $\epsilon$:
- Set $x = a + c\epsilon$ and $y = b + d\epsilon$
- Calculate

$$f(a + c\epsilon, b + d\epsilon)$$

## Partial derivatives

- Calculate
$$f(a + c\epsilon, b + d\epsilon)$$
- Expand:
$$f(a,b) + c\epsilon\frac{\partial f}{\partial x}(a,b) + d\epsilon\frac{\partial f}{\partial y}(a,b) + O(\epsilon^2)$$
- Coefficient of $\epsilon$ is
$$c\frac{\partial f}{\partial x}(a,b) + d\frac{\partial f}{\partial y}(a,b)$$
- Derivatives evaluated at $(a,b)$
- What is this derivative?

# Directional derivative

- Have

$$c\frac{\partial f}{\partial x} + d\frac{\partial f}{\partial y}$$

- This is

$$\nabla f(a, b) \cdot \mathbf{v}$$

- Where $\mathbf{v} = (c, d)$
- **Directional derivative** in direction **v**
- How calculate this? How calculate partial derivatives?

# Calculating directional derivatives

- `f(Dual(a, v₁), Dual(b, v₂))` calculates $\nabla f(a, b) \cdot \mathbf{v}$!
- $\mathbf{v} = (1, 0)$ gives $\partial f / \partial x$
- $\mathbf{v} = (0, 1)$ gives $\partial f / \partial y$

# Jacobian

- For $f : \mathbb{R}^2 \to \mathbb{R}^2$, have $f = (f_1, f_2)$ with

$$f_i(\mathbf{a} + \epsilon \mathbf{v}) = f_i(\mathbf{a}) + \epsilon \nabla f_i(\mathbf{a}) \cdot \mathbf{v}$$

- So

$$f(\mathbf{a} + \epsilon \mathbf{v}) = f(\mathbf{a}) + \epsilon \, Df(\mathbf{a}) \cdot \mathbf{v}$$

- $Df(\mathbf{a})$ is **Jacobian matrix** – matrix of partial derivatives $\frac{\partial f_i}{\partial x_j}$
- Coefficient of $\epsilon$ is Jacobian–vector product

## Newton in higher dimensions

- Generalise argument for Newton method from 1D to higher dimensions:
- Look for root of $f(\mathbf{x}) = \mathbf{0}$.
- Initial guess $\mathbf{x}_0$
- Let $\mathbf{x}_1 = \mathbf{x}_0 + \boldsymbol{\delta}$; want to find $\boldsymbol{\delta}$

- Want $f(\mathbf{x}_1) = f(\mathbf{x}_0 + \boldsymbol{\delta}) = 0$
- Approximate:

$$f(\mathbf{x}_0) + \mathsf{J} \cdot \boldsymbol{\delta} \simeq \mathbf{0}$$

- Where $\mathsf{J} := Df(\mathbf{a})$
- So need to solve *system of linear equations*

$$\mathsf{J} \cdot \boldsymbol{\delta} = -f(\mathbf{a})$$

# Linear algebra in Julia

- Given matrix $\mathsf{A}$ and vector $\mathbf{b}$
- Solve $\mathsf{A} \cdot \mathbf{x} = \mathbf{b}$ for vector $\mathbf{x}$:

```
x = A \ b
```

- `\` is a kind of “division”

# Review

- Automatic differentiation
- Derivatives of higher-dimensional functions
- Application to root finding: Newton method