Table of Contents

Essential numerical tools and perturbation analysis (2.b)

Day 2: Differentiation

Pablo Winant

African Econometric Society Workshop 14/12/23

Differentiation Flavours

Several approaches are available to differentiate functions:

1. Manual
2. Finite Differences
3. Symbolic Differentiation
4. Automatic Differentiation

Lots of packages

Finite Differences

- Choose small $\epsilon > 0$, typically $\sqrt{machine\ eps}$
- Forward Difference scheme:
  - $f'(x) \approx \frac{f(x+\epsilon)-f(x)}{\epsilon}$
  - precision: $o(\epsilon)$
  - bonus: if $f(x+\epsilon)$ unavailable, one can compute $f(x) - f(x-\epsilon)$ instead (Backward)
- Central Difference scheme:
  - $f'(x) \approx \frac{f(x+\epsilon)-f(x-\epsilon)}{2\epsilon}$
  - average of forward and backward
  - precision: $o(\epsilon^2)$

Finite Differences: Higher order

- Central formula:

$$f''(x) \approx \frac{f'(x) - f'(x-\epsilon)}{\epsilon} \approx \frac{(f(x+\epsilon)) - f(x)) - (f(x) - f(x-\epsilon))}{\epsilon^2}$$
$$= \frac{f(x+\epsilon) - 2f(x) + f(x-\epsilon)}{\epsilon^2}$$

- precision: $o(\epsilon)$
- Generalizes to higher order but becomes more and more innacurate

Exercise 1

Use finite differences to compute the derivative of function `sin(x)`

```
1 Enter cell code...
```

Try packages *FiniteDiff.jl* or *FiniteDifferences.jl*

```
1 Enter cell code...
```

Symbolic Differentiation

- manipulate the tree of algebraic expressions
    - implements various simplification rules
- requires mathematical expression
- can produce mathematical insights
- sometimes inaccurate:
    - cf: $\left(\frac{1+u(x)}{1+v(x)}\right)^{100}$

Julia Packages:

- Lots of packages
- *FiniteDiff.jl, FiniteDifferences.jl, SparseDiffTools.jl*
  - careful implementation of finite diff
- *Calculus.jl*:
  - pure julia
  - finite difference
  - symbolic calculation
- *SymEngine.jl*
  - fast symbolic calculation
- *Symbolics.jl*
  - fast, pure Julia
  - less complete than SymEngine

Exercise 2

Use `Symbolics.jl` to differentiate the expression $sqrt(sin(x))$

```
using Symbolics
```

```
Enter cell code...
```

Automatic Differentiation

- does not provide mathematical insights but solves the other problems
- can differentiate any piece of code
- two flavours
  - forward accumulation
  - reverse accumulation

Simple example

Say we want to calculate the differential of the function

```
function f(x::Float64)
    a = x + 1
    b = x^2
    c = sin(a) + a + b
end
```

By following simple differentiation rules, it can be *automatically* rewritten as:

```
function f(x::Float64)

    # x is an argument
    x_dx = 1.0

    a = x + 1
    a_dx = x_dx

    b = x^2
    b_dx = 2*x*x_dx

    t = sin(a)
    t_x = cos(a)*a_dx

    c = t + a + b
    c_x = t_dx + a_dx + b_dx

    return (c, c_x)
end
```

That is the **forward accumulation mode**.

```
## Compatible with control flow

```

```
using ForwardDiff: Dual
```

```
Dual{Nothing}(0.6930471905599447,0.0)
```

```
let
    x = Dual(1.0, 1.0)
    a = 0.5*x
    b = sum([(x)^i/i*(-1)^(i+1) for i=1:5000])
    # compare with log(1+x)
end
```

```
Dual{Nothing}(2.718281828459045,2.718281828459045,1.0)
```

```
let
    #generalizes nicely to gradient computations

    x = Dual(1.0, 1.0, 0.0)
    y = Dual(1.0, 0.0, 1.0)
    exp(x) + log(y)

end
```

```
#Example with jacobian
```

Technical remark

- autodiff libraries, use special types and operator overloading to perform operations (like Dual numbers)
- this relies on Julia duck-typing ability
  - so don't specify too precisely type arguments for functions you want to autodiff
- This works:

```
using ForwardDiff
f(x) = [x[1] + x[2], x[1]*x[2]]
ForwardDiff.jacobian(f, [0.4, 0.1])
```

- This doesn't:

```
using ForwardDiff
g(x::Vector{Float64}) = [x[1] + x[2], x[1]*x[2]]
ForwardDiff.jacobian(g, [0.4, 0.1])
```

Forward vs Reverse Accumulation Mode

- Forward Accumulation mode: isomorphic to dual number calculation
  - compute tree with values and derivatives at the same time
  - efficient for $f: R^n \rightarrow R^m$, with $n << m$
    - (keeps lots of empty gradients when $n >> m$)
- Reverse Accumulation / Back Propagation
  - efficient for $f: R^n \rightarrow R^m$, with $m << n$
  - requires data storage (to keep intermediate values)
  - graph / example
  - Very good for machine learning:
    - e.g. $\nabla_\theta F(x;\theta)$ where F can be an objective

Libraries for AutoDiff

- See JuliaDiff
  - *ForwardDiff.jl*

    - *ReverseDiff.jl*
- *Zygote.jl*
- Deep learning framework:
    - higher order diff w.r.t. any vector -> tensor operations
    - Flux.jl, MXNet.jl, Tensorflow.jl
- Other libraries like *NLsolve* or *Optim.jl* rely on on the former libraries to perform automatic differentiation automatically.

```
using NLsolve
```

```
fun! (generic function with 1 method)
```

```
function fun!(F, x)
    F[1] = (x[1]+3)*(x[2]^3-7)+18
    F[2] = sin(x[2]*exp(x[1])-1)
end
```

```
Results of Nonlinear Solver Algorithm
 * Algorithm: Trust-region with dogleg and autoscaling
 * Starting Point: [0.1, 0.2]
 * Zero: [3.7695438451406987e-13, 1.0000000000009226]
 * Inf-norm of residuals: 0.000000
 * Iterations: 5
 * Convergence: true
   * |x - x'| < 0.0e+00: false
   * |f(x)| < 1.0e-08: true
 * Function Calls (f): 6
 * Jacobian Calls (df/dx): 6
```

```
nlsolve(fun!, [0.1, 0.2], autodiff = :forward)
```