

# Probabilistic Robotics Course

## Differentiation

**Giorgio Grisetti**

`grisetti@dis.uniroma1.it`

Dept of Computer Control and Management Engineering  
Sapienza University of Rome

# Outline

- Computing Derivatives
  - Analytical
  - Numerical Differentiation
  - Automatic Differentiation

# Approaching the problem

We want to develop a KF based algorithm to track the position of Orazio as it moves

The inputs of our algorithms will be

- velocity measurements
- landmark measurements

The prior knowledge about the map is represented by the location of each landmark in the world

# Derivatives

In many cases one has to compute derivatives of complicated multivariate functions

Computing them by hand is

- tedious
- error prone
- requires a lot of time

Computers like  
doing boring stuff

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} \quad \mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{x}) \end{pmatrix}$$
$$\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_m} \end{pmatrix}$$

# Numerical Differentiation

Use the definition of derivative

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Given a set of perturbation vectors

$$\epsilon_1 = \begin{pmatrix} \epsilon \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \epsilon_2 = \begin{pmatrix} 0 \\ \epsilon \\ 0 \\ \vdots \end{pmatrix} \dots, \epsilon_m = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \epsilon \end{pmatrix}$$

Symmetry around  
linearization point  
leads to lower  
numerical errors

we compute the  $i^{\text{th}}$  column by the following

$$\frac{\partial \mathbf{f}}{\partial x_i} \simeq \frac{\mathbf{f}(\mathbf{x} + \epsilon_i) - \mathbf{f}(\mathbf{x} - \epsilon_i)}{2\epsilon}$$

$$\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_m} \end{pmatrix}$$

Each column requires  
evaluating  $\mathbf{f}$  2 times

# Numerical Differentiation

Choosing epsilon might be non trivial

- too small leads to machine precision errors
- too large poor derivative
- computation might be lowered

However

- easy to implement
- most of the times it works well
- can be used to check your hand-computed derivatives

# Automatic Differentiation

Can we get the computer giving us the exact value of the derivative at a point, **without** computing the derivatives analytically?

Derivatives are mechanic!

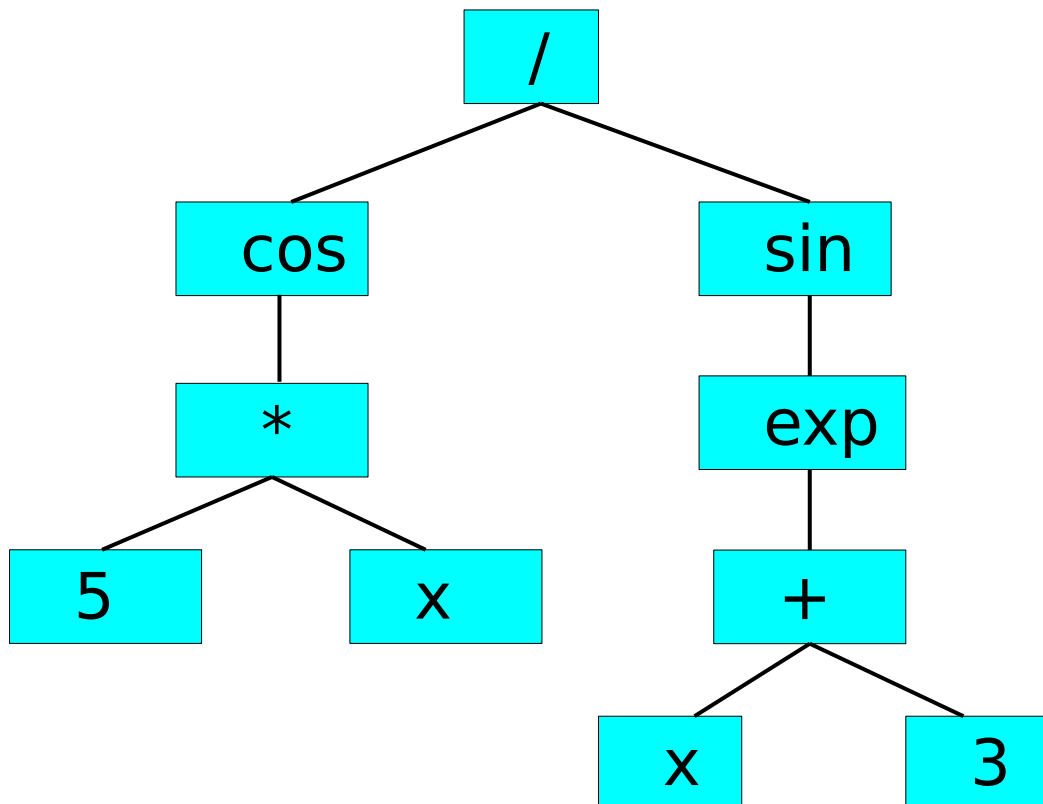
Can't we get the computer **evaluating** them for us?

# Parsing Tree

Consider the following expression

$$\cos(5*x)/(\sin(\exp(x+3)))$$

Its parsing tree looks like that





# Chain Rule

To **evaluate** the derivative of a nested function

$$f(g(x))$$

in a point  $\check{x}$  we need to know

- The **formula** of the derivative of  $f$

$$\frac{\partial f(y)}{\partial y} = f'(y)$$

- The **value** of the argument of  $f$

$$u = g(\check{x})$$

- The **value** of the derivative of the argument

$$u' = g'(\check{x})$$

**Chain rule** tells us that

$$\frac{\partial f(g(x))}{\partial x} \Big|_{x=\check{x}} = f'(u)u'$$

# Parsing Tree for Derivatives

For a generic function, we can use the parsing tree to compute

- the value of a function
- the value of the derivative of the function

We need to replace the basic type (float/double) with a pair  $(u, u')$ , and redefine the operators consistently

# Atoms and Unary Functions

## Atoms

- The variable used for differentiation becomes a pair  $[x, 1]$
- All constants become a pair  $[c_i, 0]$

## Transcendental functions

- $\sin([u, u']) = [\sin(u), \cos(u) * u']$
- $\cos([u, u']) = [\cos(u), -\sin(u) * u']$
- $\exp([u, u']) = [\exp(u), \exp(u) * u']$
- ....

# Binary Functions (operators)

Sum, Subtraction, Multiplications and Division are implemented by applying the derivative rules on their arguments

- $[u, u'] + [v, v'] = [u + v, u' + v']$
- $[u, u'] - [v, v'] = [u - v, u' - v']$
- $[u, u'] * [v, v'] = [uv, u'v + v'u]$
- $[u, u'] / [v, v'] = [uv, (u'v - v'u) / v^2]$

# Template will do

Have a look at the **autodiff** example on the course page.

It's a minimal c++ implementation, that can be embedded in Eigen.