# Calculating Derivative

Asst. Prof. Dr.-Ing. Sudchai Boonto

October 7, 2024

Department of Control System and Instrumentation Engineering
King Mongkut's Unniversity of Technology Thonburi
Thailand

# Objective

At the end of this chapter you should be able to:

- Describe, implement, and Calculating the Derivative.
- Finite Derivative
- Automatic Differentiation

# Introduction

Three derivative calculating techniques are used in the nonlinear optimization algorithm.

- **Finite Differentiation:** Based on Taylor's theorem, this technique observes the change in function values concerning the small perturbations of the unknown near a given point $x$. Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ with respect to the $i$th variable $x_i$. The central-difference formula can approximate the derivative.

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\mathbf{x} + \epsilon \mathbf{e}_i) - f(\mathbf{x} - \epsilon \mathbf{e}_i)}{2\epsilon},$$

  where $\epsilon$ is a small positive scalar and $\mathbf{e}_i$ is the $i$th unit vector, that is, the vector whose elements are all 0 except for the 1 in the $i$th position.

- **Automatic Differentiation:** This technique uses computer code to evaluate the function that can be extracted into a composition of elementary arithmetic operations to which we can apply the chain rule. The automatic derivative (AD) is often used in machine learning and AI fields.

# Introduction

- **Symbolic Differentiation:** In this technique, the algebraic specification for the function $f$ is manipulated by symbolic manipulation tools to produce new algebraic expressions for each component of the gradient.

# Finite Differentiation

A popular formula for approximating the partial derivative $\partial f/\partial x_i$ at a given point $\mathbf{x}$ is the *forward-difference*, or *one-sided-difference*, approximation, defined as

$$\frac{\partial f}{\partial x_i}(\mathbf{x}) \approx \frac{f(\mathbf{x} + \epsilon \mathbf{e}_i) - f(\mathbf{x})}{\epsilon}$$

The gradient can be built up by simply applying the formula for $i = 1, 2, \ldots, n$. This process requires evaluation of $f$ at the point $x$ as well as the $n$ perturbed points $x + \epsilon \mathbf{e}_i$, $i = 1, 2, \ldots, n$: a total of $(n + 1)$ points.
The Taylor's theorem

$$f(\mathbf{x} + \mathbf{p}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{p} + \frac{1}{2} \mathbf{p}^T \nabla^2 f(\mathbf{x} + \mathbf{p}) \mathbf{p},$$

If we choose $L$ to be a bound on the size of $\|\nabla^2 f(\cdot)\|$ in the region of interest. The last term is bounded by $(L/2)\|p\|^2$, so that

$$\|f(\mathbf{x} + \mathbf{p}) - f(\mathbf{x}) - \nabla f(\mathbf{x})^T \mathbf{p}\| \leq \frac{L}{2} \|\mathbf{p}\|^2$$

# Finite Differentiation

For $\mathbf{p} = \epsilon \mathbf{e}_i$, we have that $\nabla f(\mathbf{x})^T \mathbf{p} = \nabla f(\mathbf{x})^T \mathbf{e}_i = \frac{\partial f}{\partial x_i}$, then

$$\frac{\partial f}{\partial x_i}(\mathbf{x}) = \frac{f(\mathbf{x} + \epsilon \mathbf{e}_i) - f(\mathbf{x})}{\epsilon} + \delta_\epsilon, \text{ where } |\delta_\epsilon| \leq \frac{L}{2}\epsilon$$

- In theory the smaller $\epsilon$, the more accuracy. Unfortunately, this expression ignores the roundoff errors that are introduced when the function $f$ is evaluated on the real computer, in floating-point arithmetic.
- the quantity $\mathbf{u}$ known as *unit roundoff* is crucial. $\mathbf{u}$ is about $1.1 \times 10^{-16}$ in double precision IEEE floating-point arithmetic.
- The computed values of $f(\mathbf{x})$ and $f(\mathbf{x} + \epsilon \mathbf{e}_i)$ are related to the exact values in the following way:

$$|\text{comp}(f(\mathbf{x})) - f(\mathbf{x})| \leq \mathbf{u} L_f,$$
$$|\text{comp}(f(\mathbf{x} - \epsilon \mathbf{e}_i)) - f(\mathbf{x} - \epsilon \mathbf{e}_i)| \leq \mathbf{u} L_f,$$

# Finite Differentiation

Then the approximation derivative is bounded by

$$\frac{L}{2}\epsilon + 2\mathbf{u}\frac{L_f}{\epsilon} \text{ and}$$

$$\frac{d}{d\epsilon}\left(\frac{L}{2}\epsilon + 2\mathbf{u}\frac{L_f}{\epsilon}\right) = \frac{L\epsilon^2 - \left(\frac{L}{2}\epsilon^2 + 2\mathbf{u}L_f\right)}{\epsilon^2} = 0$$

$$\epsilon^2 = \frac{4L_f}{L}\mathbf{u}$$

The best choice of $\epsilon$ is

$$\epsilon = \sqrt{\mathbf{u}}$$

# Finite Differentiation

A more accurate approximation to the derivative can be obtained by using the *central difference* formula, defined as

$$\frac{\partial f}{\partial x_i}(\mathbf{x}) \approx \frac{f(\mathbf{x}+\epsilon\mathbf{e}_i) - f(\mathbf{x}-\epsilon\mathbf{e}_i)}{2\epsilon)}$$

It is twice as expensive, wince we need to evaluate $f$ at the points $\mathbf{x}$ and $\mathbf{x} \pm \epsilon\mathbf{e}_i, i = 1, 2, \ldots, n$ : a total of $2n + 1$ points. I can be proved as

$$f(\mathbf{x}+\epsilon\mathbf{e}_i) = f(\mathbf{x}) + \epsilon\frac{\partial f}{\partial x_i} + \frac{1}{2}\epsilon^2\frac{\partial^2 f}{\partial x_i^2} + \mathcal{O}(\epsilon^3)$$
$$f(\mathbf{x}-\epsilon\mathbf{e}_i) = f(\mathbf{x}) - \epsilon\frac{\partial f}{\partial x_i} + \frac{1}{2}\epsilon^2\frac{\partial^2 f}{\partial x_i^2} + \mathcal{O}(\epsilon^3)$$

Subtracting the second equation from the first and dividing by $2\epsilon$, we get

$$\frac{\partial f}{\partial x_i}(\mathbf{x}) = \frac{f(\mathbf{x}+\epsilon\mathbf{e}_i) - f(\mathbf{x}-\epsilon\mathbf{e}_i)}{2\epsilon)} + \mathcal{O}(\epsilon^2), \text{ where } \epsilon = \mathbf{u}^{\frac{1}{3}}$$

# Finite Differentiation: example

Consider the following function with two variables and two functions of interest:

$$f(\mathbf{x}) = \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} x_1 x_2 + \sin x_1 \\ x_1 x_2 + x_2^2 \end{bmatrix}$$

We can differentiate this symbolically to obtain exact reference values:

$$\frac{\partial f}{\partial x} = \begin{bmatrix} x_2 + \cos x_1 & x_1 \\ x_2 & x_1 + 2x_2 \end{bmatrix}$$

We evaluate this at $x = \left(\frac{\pi}{4}, 2\right)$, which yields

$$\frac{\partial f}{\partial x} = \begin{bmatrix} 2.707 & 0.785 \\ 2.00 & 4.785 \end{bmatrix}$$

# Finite Differentiation: Matlab code

```
% esp is a spacing of floating point numbers.
epw = sqrt(eps)
epc = eps^(1/3)

f1 = @(x) x(1)*x(2) + sin(x(1)); f2 = @(x) x(1)*x(2) + x(2)^2;
f = {f1, f2}

x = [pi/4; 2]; e = eye(2);
nablafw = zeros(2,2); nablafc = zeros(2,2);

for i = 1:2
    for j = 1:2
        nablafw(j,i) = (f{j}(x + epw*e(:,i)) - f{j}(x))/epw;
        nablafc(j,i) = (f{j}(x + epc*e(:,i)) - f{j}(x - epc*e(:,i)))...
                /(2*epc);
    end
end

nablafc
nablafw
```

# Automatic Differentiation

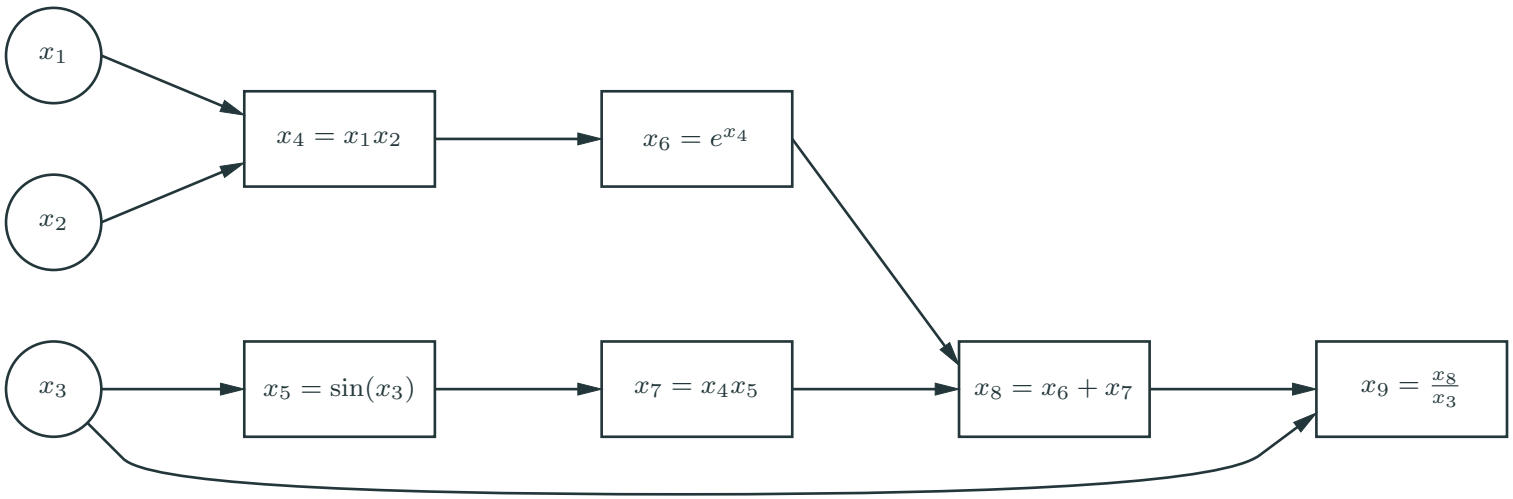
Automatic differentiation (AD) is the generic name for techniques that use the computational repreentation of a function to produce analytic values for the derivatives.

- AD techniques are founded on the observation that any function is evaluated by performing a sequence of simple elementary operations involving just one or two arguments at a time.
- Two-argument operations include addition, multiplication, division, and the power operation $a^b$.
- Single-argument operations include the trigonometric, exponential, and logarithmic function: $\sin x$, $\cos x$, $e^x$, $\log x$ etc.
- Using *chain rule*, if $h$ is a function of the vector $y \in \mathbb{R}^m$, which is in turn a function of the vector $x \in \mathbb{R}^n$, the derivative of $h$ with respect to $x$ is

$$\nabla_x h(y(x)) = \frac{\partial h}{\partial y_1} \nabla y_1(x) + \cdots + \frac{\partial h}{\partial y_m} \nabla y_m(x) = \sum_{i=1}^{m} \frac{\partial h}{\partial y_i} \nabla y_i(x)$$

# Example

Consider $f(\mathbf{x}) = \frac{(x_1 x_2 \sin x_3 + e^{x_1 x_2})}{x_3}$




$$x_4 = x_1 x_2, \quad x_5 = \sin(x_3), \quad x_6 = e^{x_4}$$
$$x_7 = x_4 x_5, \quad x_8 = x_6 + x_7, \quad x_9 = \frac{x_8}{x_3}$$

# Forward-mode AD

For a more convenient notation, we define a new variable that represents the total derivative of variable $i$ with respect to a fixed input $j$ as $\dot{x}_i = dx_i/dx_j$ and rewrite the chain rule as

$$\dot{x}_i = \sum_{k=j}^{i-1} \frac{\partial x_i}{\partial x_k} \dot{x}_k$$

$$\dot{x}_1 = 1, \quad \dot{x}_2 = \frac{\partial x_2}{\partial x_1}\dot{x}_1 \quad \dot{x}_3 = \frac{\partial x_3}{\partial x_1}\dot{x}_1 + \frac{\partial x_3}{\partial x_2}\dot{x}_2,$$

$$\dot{x}_4 = \frac{\partial x_4}{\partial x_1}\dot{x}_1 + \frac{\partial x_4}{\partial x_2}\dot{x}_2 + \frac{\partial x_4}{\partial x_3}\dot{x}_3, \quad \dot{x}_5 = \frac{\partial x_5}{\partial x_1}\dot{x}_1 + \frac{\partial x_5}{\partial x_2}\dot{x}_2 + \frac{\partial x_5}{\partial x_3}\dot{x}_3 + \frac{\partial x_5}{\partial x_4}\dot{x}_4$$

$$\dot{x}_6 = \frac{\partial x_6}{\partial x_1}\dot{x}_1 + \frac{\partial x_6}{\partial x_2}\dot{x}_2 + \frac{\partial x_6}{\partial x_3}\dot{x}_3 + \frac{\partial x_6}{\partial x_4}\dot{x}_4 + \frac{\partial x_6}{\partial x_5}\dot{x}_5$$

$$\vdots$$

$$\dot{x}_9 = \frac{df}{dx}$$

# Forward-mode AD

$$\dot{x}_1 = 1, \quad \dot{x}_2 = 0, \quad \dot{x}_3 = 0, \dot{x}_4 = x_2\dot{x}_1 + x_1\dot{x}_2 = x_2$$
$$\dot{x}_5 = \cos(x_3)\dot{x}_3 = 0, \quad \dot{x}_6 = e^{x_4}\dot{x}_4 = x_2 e^{x_4} = x_2 e^{x_1 x_2}$$
$$\dot{x}_7 = x_5\dot{x}_4 + x_4\dot{x}_5 = x_2 x_5,$$
$$\dot{x}_8 = 1\dot{x}_6 + 1\dot{x}_7 = x_2 e^{x_1 x_2} + x_2 x_5 = x_2 e^{x_1 x_2} + x_2 \sin(x_3)$$
$$\dot{x}_9 = \frac{\partial x_9}{\partial x_3}\dot{x}_3 + \frac{1}{x_2}\dot{x}_8 = \frac{x_2}{3}\left(e^{x_1 x_2} + \sin(x_3)\right) = \frac{df}{dx_1}$$

We can find $\frac{df}{dx_2}$ and $\frac{df}{dx_3}$ by setting $\dot{x}_2 = 1$ and $\dot{x}_3 = 1$ respectively.

$$\frac{df}{dx_2} = \frac{x_1}{x_3}\left(\sin x_3 + e^{x_1 x_2}\right)$$
$$\frac{df}{dx_3} = \frac{1}{x_3} x_1 x_2 \cos x_3 - \frac{1}{x_3^2}\left(e^{x_1 x_2} + x_1 x_2 \sin x_3\right)$$

# Reverse-mode AD

The *reverse mode* is also based on the chain rule but uses the alternative form:

$$\frac{dx_i}{dx_j} = \sum_{k=j+1}^{i} \frac{\partial x_k}{\partial x_j} \frac{dx_i}{dx_k},$$

where the summation happens in reverse (starts at $i$ and decrements to $j+1$). This is less intuitive than the forward chain rule, but it is equally valid.
We defined a more convenient notation for the variables that carry the total derivatives with a fixed $i$ as $\bar{x}_j = dx_i/dx_j$, which are sometimes called *adjoint* variables. Then we can rewrite the chain rule as

$$\bar{x}_j = \sum_{k=j+1}^{i} \frac{\partial x_k}{\partial x_j} \bar{x}_k$$

# Reverse-mode AD

$$f(\mathbf{x}) = \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} x_1 x_2 + \sin x_1 \\ x_1 x_2 + x_2^2 \end{bmatrix}, \mathbf{x} = \begin{bmatrix} \frac{\pi}{4}, 2 \end{bmatrix}$$

$$x_3 = x_1 x_2, \quad x_4 = \sin x_1, \quad x_5 = x_3 + x_4 = f_1$$

$$x_6 = x_2^2, \quad x_7 = x_3 + x_6 = f_2$$

We have

$$\bar{x}_7 = 1, \quad \bar{x}_6 = \frac{\partial x_7}{\partial x_6}\bar{x}_7 = \bar{x}_7 = 1, \quad \bar{x}_5 = \frac{\partial x_7}{\partial x_5}\bar{x}_7 + \frac{\partial x_6}{\partial x_5}\bar{x}_6 = 0$$

$$\bar{x}_4 = \frac{\partial x_5}{\partial x_4}\bar{x}_5 = 0, \qquad \bar{x}_3 = \frac{\partial x_7}{\partial x_3}\bar{x}_7 + \frac{\partial x_5}{\partial x_3}\bar{x}_5 = \bar{x}_7 + \bar{x}_5 = 1$$

$$\bar{x}_2 = \frac{\partial x_6}{\partial x_2}\bar{x}_6 + \frac{\partial x_3}{\partial x_2}\bar{x}_3 = 2x_2\bar{x}_6 + x_1\bar{x}_3 = 2x_2 + x1 = 4.785 = \frac{\partial f_2}{\partial x_2}$$

$$\bar{x}_1 = \frac{\partial x_4}{\partial x_1}\bar{x}_4 + \frac{\partial x_3}{\partial x_1}\bar{x}_3 = \cos x_1 \bar{x}_4 + x_2\bar{x}_3 = 2 = \frac{\partial f_2}{\partial x_1}$$

For $\frac{\partial f_1}{\partial x_1}$ and $\frac{\partial f_1}{\partial x_2}$, we can find by setting $\bar{x}_5 = 1$.

# Forward-mode or Reverse-mode AD

The difference between the forward and the reverse approaches is that:

- The forward mode computes the Jacobian column by column. Thus, the cost of the forward mode is proportional to the number of the input $n_x$
- The reverse mode computes the Jacobian row by row. The cost of the reverse mode is proportional to the number of the output function $n_f$.
- If we have more outputs (e.g., objective and constraints) than inputs (design variables), the forward mode is more efficient.
- If we have many more input than outputs, then the reverse mode is more efficient, as normally using in Machine Learning fields.

In `Julia`, there are packages like `ForwardDiff.jl` and `ReverseDiff.jl`. For Matlab, the deep learning toolbox contains functions to do the AD.

# Reference


1. Joaquim R. R. A. Martins, and Andrew Ning, "*Engineering Design Optimization*," Cambridge University Press, 2021.
2. Jorge Nocedal, and Stephen J. Wright, "*Numerical Optimization*," 2nd, Springer, 2026