# Differentiation options

$h = 1e-6 = 10^{-6}$

1. Numerical differentiation $\rightarrow \frac{\partial f(x)}{\partial x} = \frac{f(x+h) - f(x)}{h}$
2. Symbolic differentiation (SD) $\rightarrow$ sympy, matlab
3. Automatic differentiation (AD)
   - A. Forward mode differentiation
   - B. Reverse mode differentiation

$x_1 \exp(-(x_1^2 + x_2^2))$ $\rightarrow$ Symbolic Diff $\rightarrow$ $\frac{\partial f(x)}{\partial x}$ in math symbol

```
x1 = np.array(1)
x2 = np.array(2)
f = x1 ** np.exp(-x1 ** 2 - x2 ** 2)
```

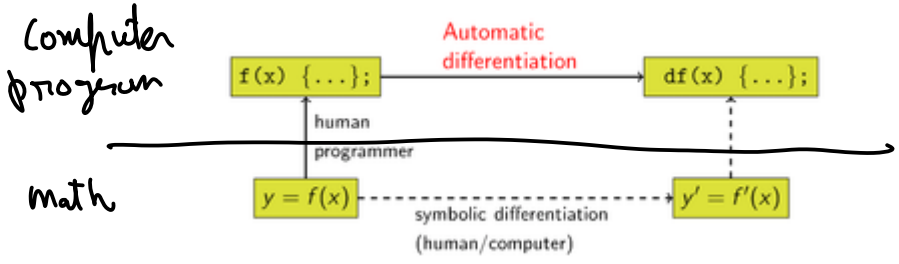
AD $\rightarrow$ Computer code for computing the derivative

$\left.\frac{\partial f}{\partial x_1}\right|_{x_1=1}$ $\left.\frac{\partial f}{\partial x_2}\right|_{x_2=2}$

# 1. Numerical differentiation

# 2. Symbolic differentiation

# 3. Automatic differentiation



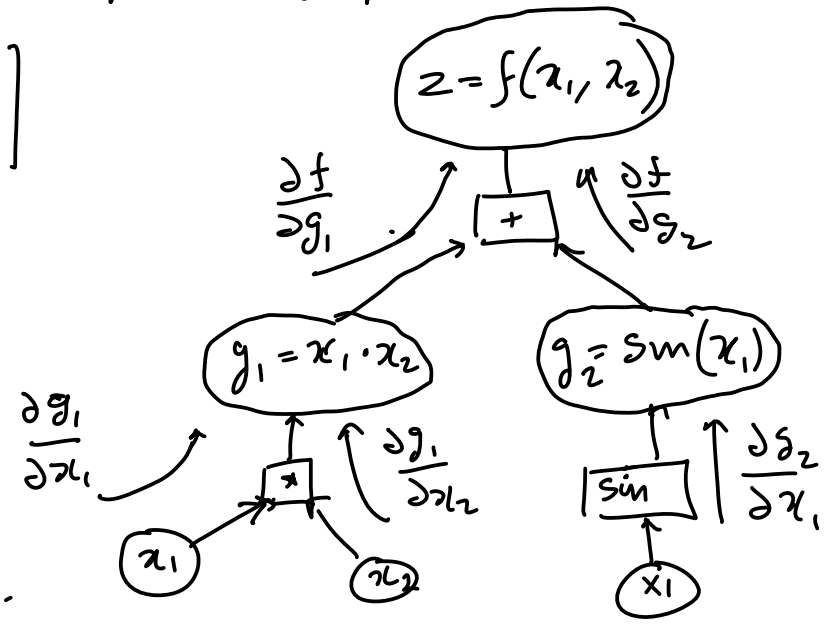
3.A Forward mode

Example:

$$z = f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$

Every mathematical expression can be written as computational graph

$$z = f(x_1, x_2) = \left[ (x_1 \cdot x_2) + (\sin(x_1)) \right]$$




$$f = g_1 + g_2$$

$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial g_1}\frac{\partial g_1}{\partial x_1} + \frac{\partial f}{\partial g_2}\frac{\partial g_2}{\partial x_1}$$

$$\frac{\partial g_1}{\partial x_1} = \frac{\partial (x_1 \cdot x_2)}{\partial x_1}$$

$$\frac{\partial g_2}{\partial x_1} = \frac{\partial (\sin(x_1))}{\partial x_1}$$

$$\frac{\partial f}{\partial x_2} = \frac{\partial f}{\partial g_1}\frac{\partial g_1}{\partial x_2}$$

$$g_0 = x_1$$

$$\frac{\partial g_0}{\partial x_1} = 1$$

$$f = g_2(g_1(g_0(x)))$$

$$\frac{\partial f}{\partial x} = \left(\frac{\partial f}{\partial g_2}\left(\frac{\partial g_2}{\partial g_1}\left(\frac{\partial g_1}{\partial g_0}\cdot\frac{\partial g_0}{\partial x}\right)\right)\right)$$ Forward mode AD

$$\frac{\partial f}{\partial x} = \left(\left(\left(\frac{\partial f}{\partial g_2}\cdot\frac{\partial g_2}{\partial g_1}\right)\frac{\partial g_1}{\partial g_0}\right)\frac{\partial g_0}{\partial x}\right)$$ Reverse mode AD

$$f(x) = g(x) + h(x)$$

$$\frac{\partial f(x)}{\partial x} = \frac{\partial g(x)}{\partial x} + \frac{\partial h(x)}{\partial x}$$

known

$$\frac{\partial f}{\partial x} = \left(\frac{\partial f}{\partial g_2}\left(\frac{\partial g_2}{\partial g_1}\left(\frac{\partial g_1}{\partial g_0}\cdot\frac{\partial g_0}{\partial x}\right)\right)\right) + \cdots$$

$$f(x) = g(x)\cdot h(x)$$

$$\frac{\partial f}{\partial x} = \frac{\partial g(x)}{\partial x} h(x) + g(x)\frac{\partial h(x)}{\partial x}$$

$$f(x) = \sin(g(x)) \Rightarrow \frac{\partial f}{\partial x} = \cos(g(x))\frac{\partial g(x)}{\partial x}$$

Forward mode of AD $\quad f(x_1, x_2, \ldots x_n) \longmapsto (z_1, \ldots z_m)$

FM requires $n$-passes for computing all the derivatives of $f(\underline{x})$ $\quad \Big] n << m$

Reverse mode

RM $\quad m$-passes " $\quad \Big] n >> m$

Numerical gradient

$f(x_1, x_2) \longmapsto z_1$

$$\frac{\partial f}{\partial x_1} = \frac{f(x_1+h, x_2) - f(x_1, x_2)}{h}$$

$$\frac{\partial f}{\partial x_2} = \frac{f(x_1, x_2+h) - f(x_1, x_2)}{h}$$

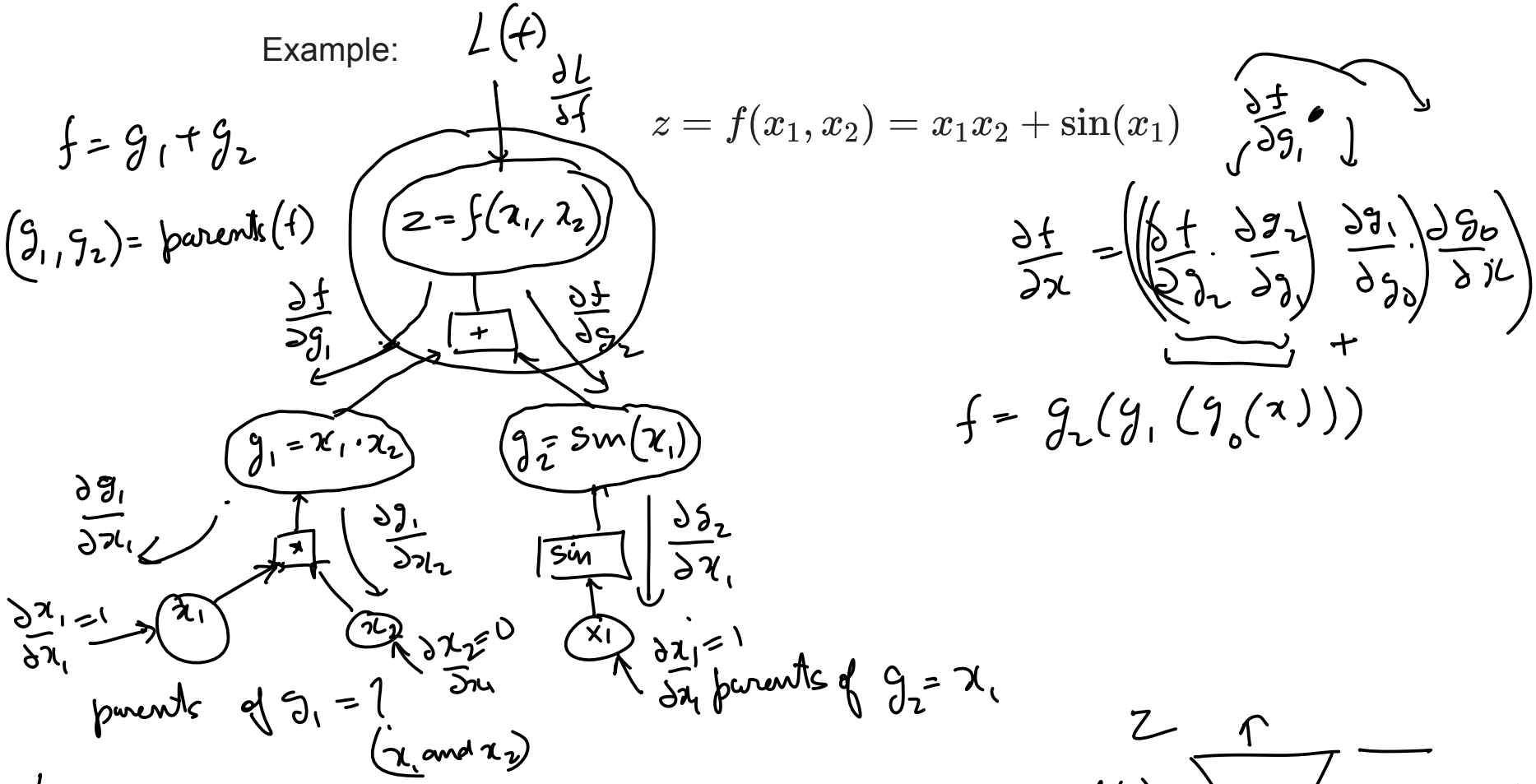
$f(x_1, \dots x_n) \longmapsto z_1$ $\quad n+1$ : NG requires $n+1$ passes

$f(x_1) \longmapsto (z_1, \dots z_n)$

$\cdot \frac{\partial f}{\partial x_1} = \frac{f(x_1+h) - f(x_1)}{h}$ 2 passes

3.B Reverse mode

Example:




$$\frac{\partial f}{\partial x} = \left(\left(\frac{\partial f}{\partial g_2}\cdot\frac{\partial g_2}{\partial g_1}\right)\frac{\partial g_1}{\partial g_0}\right)\frac{\partial g_0}{\partial x} + \cdots$$

$$f = g_2(g_1(g_0(x)))$$

$f(x) = g(x) + h(x)$

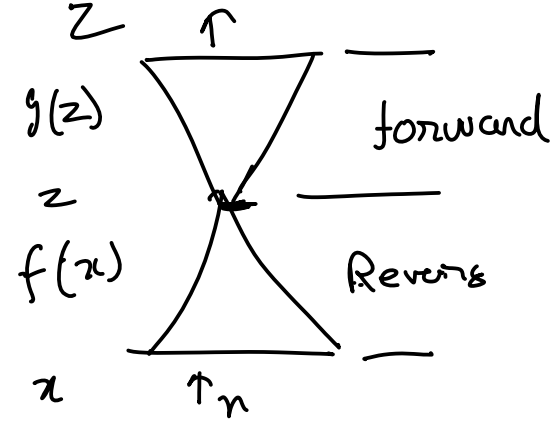
$\frac{\partial f}{\partial x} = \frac{\partial g}{\partial x} + \frac{\partial h}{\partial x} \Rightarrow$

grad add_derive($g(x), h(x)$)

$\text{grad} = \frac{\partial f}{\partial f}$

$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial x}$

$\Rightarrow \frac{\partial L}{\partial x} = \frac{\partial L}{\partial g}\frac{\partial g}{\partial x} + \frac{\partial L}{\partial h}\frac{\partial h}{\partial x}$



In [58]:

```
import numpy as np
class ForwardDiff:
    def __init__(self, value, grad=None):
        self.value = value
        self.grad = np.zeros_like(value) if grad is None else grad


    def __add__(self, other):
        cls = type(self)
        other = other if isinstance(other, cls) else cls(other)
        out = cls(self.value + other.value,
                  self.grad + other.grad)
        return out
    __radd__ = __add__

    def __repr__(self):
        return f"{self.__class__.__name__}(data={self.value}, grad={sel
x = ForwardDiff(2, 1)
y = ForwardDiff(3, 0)

f = x + y
f
```

Out[58]:

```
ForwardDiff(data=5, grad=1)
```

In [63]:

```
oldFD = ForwardDiff # Bad practice: do not do it
class ForwardDiff(oldFD):
    def __mul__(self, other):
        cls = type(self)
        other = other if isinstance(other, cls) else cls(other)
        out = cls(self.value * other.value,
                  other.value * self.grad+
                   self.value * other.grad)
        return out

    __rmul__ = __mul__

x = ForwardDiff(2, 0)
y = ForwardDiff(3, 1)

f1 = x * y
f2 = 2*x + 3*y + x*y
f1, f2
```

Out[63]:

```
(ForwardDiff(data=6, grad=2), ForwardDiff(data=19, grad=5))
```

In [66]:

```
oldFD = ForwardDiff # Bad practice: do not do it
class ForwardDiff(oldFD):
    def log(self):
        cls = type(self)
        return cls(np.log(self.value),
                   1/self.value * self.grad)
    def exp(self):
        cls = type(self)
        out_val = np.exp(self.value)
        return cls(out_val,
                   out_val * self.grad)

    def sin(self):
        cls = type(self)
        return cls(np.sin(self.value),
                   np.cos(self.value) * self.grad)

    def cos(self):
        cls = type(self)
        return cls(np.cos(self.value),
                   -np.sin(self.value) * self.grad)

    def __pow__(self, other):
        cls = type(self)
        other = other if isinstance(other, cls) else cls(other)
        return (self.log() * other).exp()

    def __neg__(self): # -self
        return self * -1
```

```
    def __sub__(self, other): # self - other
        return self + (-other)

    def __truediv__(self, other): # self / other
        return self * other**-1

    def __rtruediv__(self, other): # other / self
        return other * self**-1



x = ForwardDiff(2, 1)
y = ForwardDiff(3, 0)

f = x**y
f
```

Out[66]: ForwardDiff(data=7.999999999999999, grad=11.999999999999998)

In [104]:

```
import numpy as np
def add_vjp(a, b, grad):
    return grad, grad


def no_parents_vjp(grad):
    return (grad,)

class ReverseDiff:
    def __init__(self, value, parents=(), op='', vjp=no_parents_vjp):
        self.value = value
        self.parents = parents
        self.op = op
        self.vjp = vjp
        self.grad = None

    def backward(self, grad):
        self.grad = grad
        op_args = [p.value for p in self.parents]
        grads = self.vjp(*op_args, grad)
        for g, p in zip(grads, self.parents):
            p.backward(g)

    def __add__(self, other):
        cls = type(self)
        other = other if isinstance(other, cls) else cls(other)
        out = cls(self.value + other.value,
                  parents=(self, other),
                  op='+',
                  vjp=add_vjp)
```

```
        return out

    __radd__ = __add__

    def __repr__(self):
        cls = type(self)
        return f"{cls.__name__}(value={self.value}, parents={self.parer

x = ReverseDiff(2)
y = ReverseDiff(3)

f = x + y + 3
f.backward(1)
f
x.grad, y.grad
```

Out[104]: (1, 1)

In [109]:

```
oldRD = ReverseDiff # Bad practice: do not do it

def mul_vjp(a, b, grad):
    return grad * b, grad * a

class ReverseDiff(oldRD):
    def __mul__(self, other):
        cls = type(self)
        other = other if isinstance(other, cls) else cls(other)
        out = cls(self.value * other.value,
                  parents=(self, other),
                  op='*',
                  vjp=mul_vjp)
        return out

    __rmul__ = __mul__

x = ReverseDiff(2)
y = ReverseDiff(3)

f1 = 5*x + 7* y
f1.backward(1)
x.grad, y.grad
```

Out[109]:

```
(5, 7)
```

In [110]:
```
f2 = x*y
f2.backward(1)
x.grad, y.grad
```

Out[110]:
```
(3, 2)
```

# Computational complexity