AI505, Optimization – Exercise Sheet 01

2026-02-17

ℹ Solutions included.

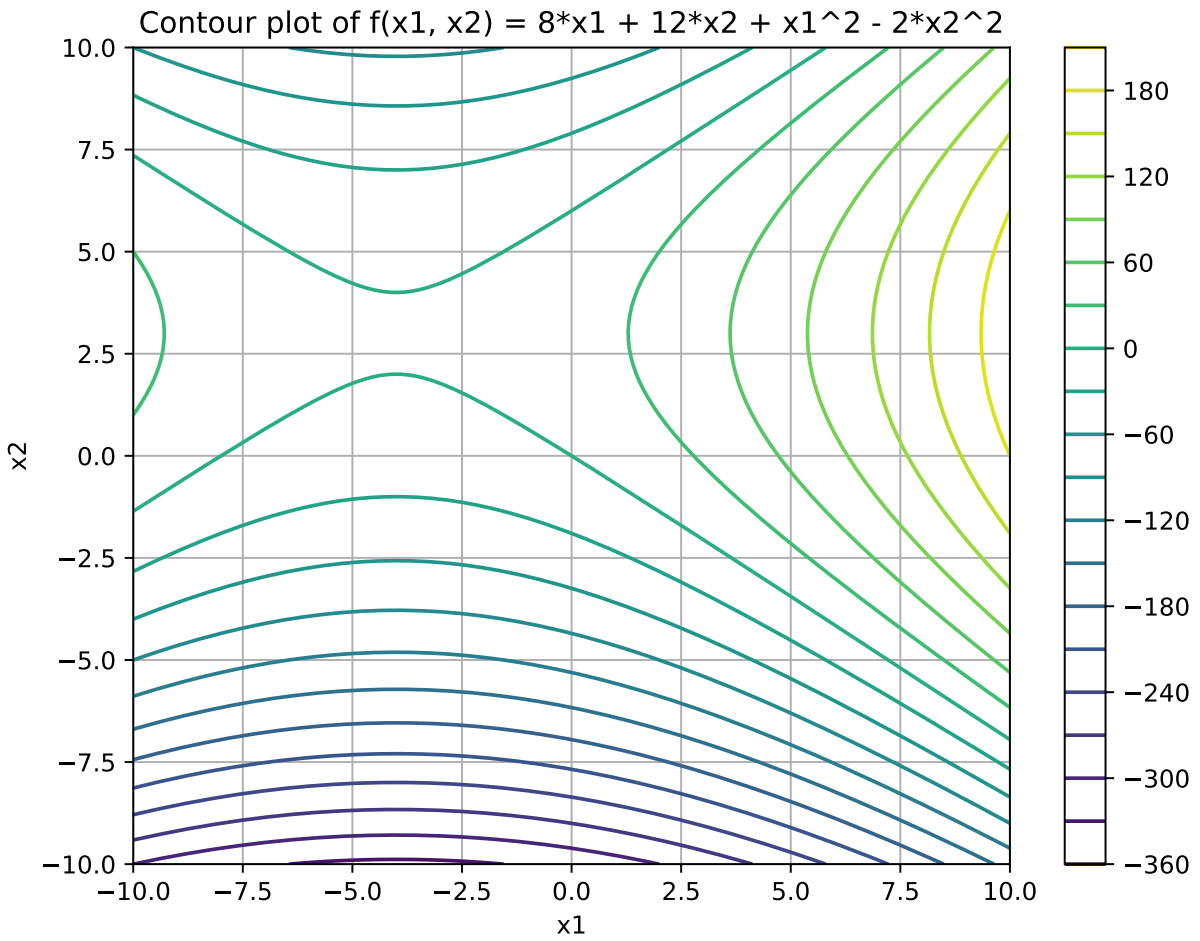
Exercises with the symbol $^+$ are to be done at home before the exercise class. Exercises with the symbol * will be tackled in class. The remaining exercises are left for self training after the exercise class. Some exercises are from the text book and the number is reported in parenthesis. They have the solution at the end of the book.

Exercise 1 $^+$ Python

Show that the function $f(x) = 8x_1 + 12x_2 + x_1^2 - 2x_2^2$ has only one stationary point, and that it is neither a maximum or minimum, but a saddle point (or inflection point). Plot the contour lines of f in Python (see slides 17, 18 of the tutorial material Part 3).

Solution

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the function
def f(x1, x2):
    return 8 * x1 + 12 * x2 + x1**2 - 2 * x2**2


# Define the grid for plotting
x1 = np.linspace(-10, 10, 400)
x2 = np.linspace(-10, 10, 400)
X1, X2 = np.meshgrid(x1, x2)
Z = f(X1, X2)

# Create the contour plot
plt.figure(figsize=(8, 6))
contour = plt.contour(X1, X2, Z, levels=20, cmap="viridis")
plt.colorbar(contour)
plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Contour plot of f(x1, x2) = 8*x1 + 12*x2 + x1^2 - 2*x2^2")
plt.grid(True)
plt.savefig("contour.png")
```



Exercise 2 +

Write the second-order Taylor expansion for the function $\cos(1/x)$ around a nonzero point x, and the third-order Taylor expansion of $\cos(x)$ around any point x. Evaluate the second expansion for the specific case of $x = 1$.

Solution

Let $a \neq 0$, then the Taylor expansion of $\cos(1/x)$ at $x = a$ is given by:

$$\cos(1/(x-a)) = \cos(1/a) + \frac{(x-a)\sin(1/a)}{a^2} - \frac{(x-a)^2(2a\sin(1/a) + \cos(1/a)}{2a^4} + O((x-a)^3)$$

The Taylor expansion of $\cos(x)$ around $x = 1$ is given by:

$$\cos(x-1) = \cos(1) - (x-1)\sin(1) - \frac{1}{2}(x-1)^2\cos(1) + \frac{1}{6}(x-1)^3\sin(1) + O((x-1)^4)$$

Exercise 3

Suppose that $f(\boldsymbol{x}) = \boldsymbol{x}^T Q\boldsymbol{x}$, where Q is an $n \times n$ *symmetric positive semidefinite* matrix. Show using the definition of convex functions, that $f(\boldsymbol{x})$ is convex on the domain $\mathbb{R}^n$. Hint: It may be convenient to prove the following equivalent inequality: $f(\boldsymbol{y}+\alpha(\boldsymbol{x}-\boldsymbol{y}))-\alpha f(\boldsymbol{x})-(1-\alpha)f(\boldsymbol{y}) \leq 0$ for all $\alpha \in [0,1]$ and all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

Solution

Since Q is positive semidefinite, the function $f(\boldsymbol{x})$ is a quadratic form (see the slides $\boldsymbol{x}^T A\boldsymbol{x}$ to confirm this) which is convex. However, we are asked to prove it using the definition, not just citing properties of quadratic forms.
We have the following function:

$$f(\boldsymbol{x}) := \boldsymbol{x}^\top Q\boldsymbol{x}, \qquad \boldsymbol{x} \in \mathbb{R}^n,$$

where $Q \in \mathbb{R}^{n\times n}$ is *positive semidefinite* and symmetric. That is:

$$z^\top Qz \geq 0 \quad \text{for all } z.$$

We prove that f is convex on $\mathbb{R}^n$ using the definition:

$$f(\boldsymbol{y}+\alpha(\boldsymbol{x}-\boldsymbol{y}))-\alpha f(\boldsymbol{x})-(1-\alpha)f(\boldsymbol{y}) \leq 0$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and all $\alpha \in [0,1]$.
For the first term on the left-hand side, we have

$$\boldsymbol{z} := \boldsymbol{y}+\alpha(\boldsymbol{x}-\boldsymbol{y}) = (1-\alpha)\boldsymbol{y}+\alpha\boldsymbol{x}.$$

Then

$$\begin{aligned} f(\boldsymbol{z}) &= \boldsymbol{z}^\top Q\boldsymbol{z} \\ &= \big((1-\alpha)\boldsymbol{y}+\alpha\boldsymbol{x}\big)^\top Q\big((1-\alpha)\boldsymbol{y}+\alpha\boldsymbol{x}\big). \end{aligned}$$

Using symmetry of Q,

$$f(z) = (1-\alpha)^2\boldsymbol{y}^\top Qy + 2\alpha(1-\alpha)\boldsymbol{x}^\top Q\boldsymbol{y} + \alpha^2\boldsymbol{x}^\top Q\boldsymbol{x}.$$

For the second part of the left-hand side, the affine combination: $\alpha f(\boldsymbol{x})+(1-\alpha)f(\boldsymbol{y})$, we have:

$$\alpha f(\boldsymbol{x})+(1-\alpha)f(\boldsymbol{y}) = \alpha\boldsymbol{x}^\top Q\boldsymbol{x}+(1-\alpha)\boldsymbol{y}^\top Q\boldsymbol{y}.$$

Putting the terms together, we have:

$$\begin{aligned} &f(\boldsymbol{z})-\alpha f(\boldsymbol{x})-(1-\alpha)f(\boldsymbol{y}) \\ &= \alpha^2\boldsymbol{x}^\top Q\boldsymbol{x}-\alpha\boldsymbol{x}^\top Q\boldsymbol{x}+(1-\alpha)^2\boldsymbol{y}^\top Q\boldsymbol{y}-(1-\alpha)\boldsymbol{y}^\top Q\boldsymbol{y} \\ &\quad + 2\alpha(1-\alpha)\boldsymbol{x}^\top Q\boldsymbol{y}. \end{aligned}$$

Factoring terms yields

$$\begin{aligned} &= -\alpha(1-\alpha)\boldsymbol{x}^\top Q\boldsymbol{x}-\alpha(1-\alpha)\boldsymbol{y}^\top Q\boldsymbol{y}+2\alpha(1-\alpha)\boldsymbol{x}^\top Q\boldsymbol{y} \\ &= \alpha(1-\alpha)\Big(-\boldsymbol{x}^\top Q\boldsymbol{x}-\boldsymbol{y}^\top Q\boldsymbol{y}+2\boldsymbol{x}^\top Q\boldsymbol{y}\Big). \end{aligned}$$

Observe that

$$\boldsymbol{x}^\top Q\boldsymbol{x}+\boldsymbol{y}^\top Q\boldsymbol{y}-2\boldsymbol{x}^\top Q\boldsymbol{y} = (\boldsymbol{x}-\boldsymbol{y})^\top Q(\boldsymbol{x}-\boldsymbol{y}).$$

Therefore,

$$f(\boldsymbol{z})-\alpha f(\boldsymbol{x})-(1-\alpha)f(\boldsymbol{y}) = -\alpha(1-\alpha)(\boldsymbol{x}-\boldsymbol{y})^\top Q(\boldsymbol{x}-\boldsymbol{y}).$$

Since $\alpha(1-\alpha) \geq 0$ for all $\alpha \in [0,1]$ and Q is positive semidefinite, we have that

$$(\boldsymbol{x}-\boldsymbol{y})^\top Q(\boldsymbol{x}-\boldsymbol{y}) \geq 0,$$

we conclude that

$$f(\boldsymbol{y}+\alpha(\boldsymbol{x}-\boldsymbol{y}))-\alpha f(\boldsymbol{x})-(1-\alpha)f(\boldsymbol{y})\leq 0.$$

Thus,

$$f(\boldsymbol{x})=\boldsymbol{x}^\top Q\boldsymbol{x}$$

is convex on $\mathbb{R}^n$. If Q is positive definite, the inequality is strict for $\boldsymbol{x}\neq\boldsymbol{y}$ and $\alpha\in(0,1)$, so f is strictly convex.

Exercise 4

Suppose that f is a convex function. Show that the set of global minimizers of f is a convex set.

Solution

We assume to have optimal points x_i from the set of global optimizers $i=1..n$. Let $x^*=\sum_i\lambda_i x_i$, λ in $[0,1]$, $\sum_i\lambda=1$ be any convex combination of them. Because f is convex we have that:

$$f\left(\sum_i\lambda_i x_i\right)\leq\sum_i\lambda_i f(x_i)$$

Since x_i are optimal solutions, $f(x_i)=z$ and there does not exist any other point x' with $f(x')<z$. Using the fact that $\sum_i\lambda=1$ then:

$$f\left(\sum_i\lambda_i x_i\right)\leq\sum_i\lambda_i f(x_i)=z$$

and since no other point can be better we necessarily have that $f(\sum_i\lambda_i x_i)=z$ and, hence, that any convex combination like x^* is optimal itself. The set of optimal solutions is therefore convex.

Exercise 5 *

Consider the function $f(x_1,x_2)=(x_1+x_2^2)^2$. At the point $\boldsymbol{x}_0=[1,0]$ we consider the search direction $\boldsymbol{p}=[-1,1]$. Show that $\boldsymbol{p}$ is a descent direction and find all minimizers of the problem $\min_\alpha f(\boldsymbol{x}_0+\alpha\boldsymbol{p})$.

Solution

Substituting $\boldsymbol{x}_0$ and $\boldsymbol{p}$ we want to solve:

$$\min_\alpha f\left(\begin{bmatrix}1\\0\end{bmatrix}+\alpha\begin{bmatrix}-1\\1\end{bmatrix}\right)=\min_\alpha f\left(\begin{bmatrix}1-\alpha\\\alpha\end{bmatrix}\right)=\min_\alpha\,(1-\alpha+\alpha^2)^2$$

We look for the points where the necessary conditions of local optimality are satisfied:

$$f'(\boldsymbol{x}_0,\boldsymbol{p},\alpha)=2(1-\alpha+\alpha^2)(-1+2\alpha)=(4\alpha-2)(\alpha^2-\alpha+1)$$

$$f''(\boldsymbol{x}_0,\boldsymbol{p},\alpha)=4\alpha^2-4\alpha+(2\alpha-1)(4\alpha-2)+4$$

The first derivative is zero at the only real number: $\alpha_1=1/2$. The second term gives a complex number:

$$\alpha_{2,3}=\frac{1\pm\sqrt{1-4}}{2}$$

In $\alpha_1 = 1/2$, the second derivative is 3 hence positive. The point α is therefore a local minimum. If the function is convex the point is also a global minimum.
The function is convex if its second derivative is always larger or equal to zero, that is:

$$4\alpha^2 - 4\alpha + (2\alpha - 1)(4\alpha - 2) + 4 \geq 0$$

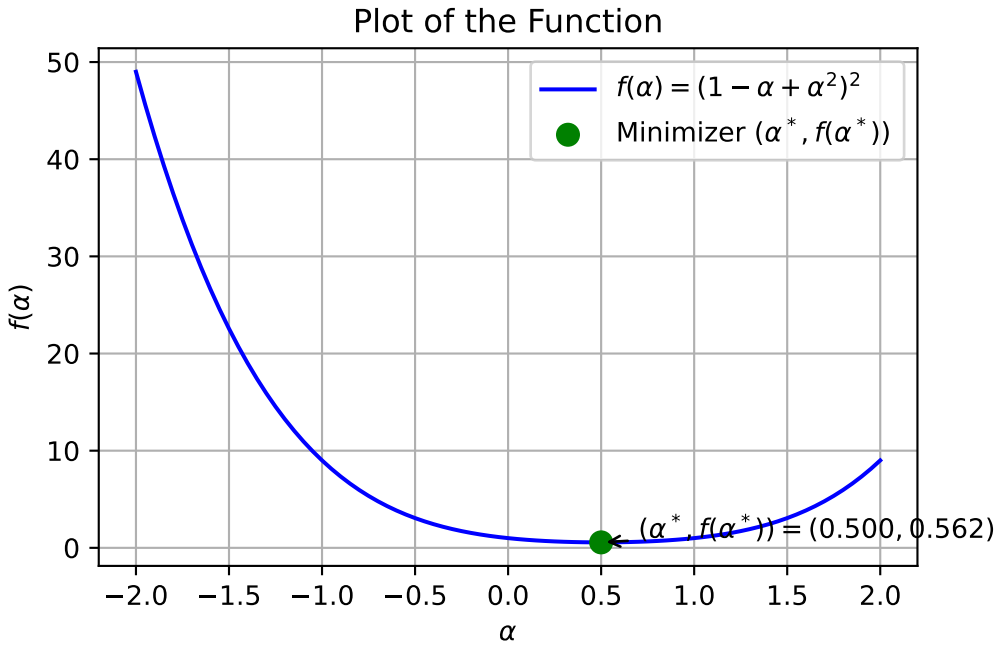
or equivalently

$$2(6\alpha^2 - 2\alpha + 3) \geq 0$$

The second factor does not have any real root and it is positive in α_1, hence it is always positive.
Since $\alpha = 1/2$ is the unique global minimizer for this problem, it is clear that $f(x_0 + 0.5p) < f(x_0)$, (substituting $\alpha = 0$) and hence p is a descent direction.
We could have carried out the analysis in Python as follows:

```python
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp

def f(alpha):
    return (1 - alpha + alpha**2) ** 2

# Define the range of alpha values
alpha_values = np.arange(-2, 2 + 0.05, 0.05)
f_values = f(alpha_values)

# Compute minimizer
min_idx = np.argmin(f_values)
alpha_star = alpha_values[min_idx]
f_star = f_values[min_idx]

# Plot the function
plt.plot(
    alpha_values, f_values, label=r"$f(\alpha) = (1 - \alpha + \alpha^2)^2$", color="b"
)

plt.plot(alpha_star, f_star, marker="o", color="g", markersize=8,
         linestyle="None", label=r"Minimizer $(\alpha^*, f(\alpha^*))$")
plt.annotate(                                                    # Add values of minimizer
    fr"$(\alpha^*, f(\alpha^*))=({alpha_star:.3f}, {f_star:.3f})$",
    xy=(alpha_star, f_star),
    xytext=(alpha_star + 0.2, f_star + 0.2),
    arrowprops=dict(arrowstyle="->"),
)
plt.xlabel(r"$\alpha$")
plt.ylabel(r"$f(\alpha)$")
plt.title("Plot of the Function")
plt.legend()
plt.grid()
plt.savefig("function_plot.png")

# Compute the derivatives using sympy
alpha=sp.symbols('alpha')
f_sym = (1 - alpha + alpha**2)**2
deriv_f = sp.diff(f_sym, alpha)
second_deriv_f = sp.diff(deriv_f, alpha)
print("Derivative of f(alpha):", deriv_f)
print("Second derivative of f(alpha):", second_deriv_f)
```

```
Derivative of f(alpha): (4*alpha - 2)*(alpha**2 - alpha + 1)
Second derivative of f(alpha): 4*alpha**2 - 4*alpha + (2*alpha - 1)*(4*alpha - 2) + 4
```



Exercise 6 +

Consider the case of a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$. The matrix $J(x)$ of dimension $n \times m$ of first derivatives for this function is defined as follows:

$$J(\boldsymbol{x}) = \left[\frac{\partial}{\partial x_i} f_j\right]_{\substack{i=1..n \\ j=1..m}}.$$

Write the forward-difference calculations needed to compute $J(\boldsymbol{x})$ at a given point $\boldsymbol{x}$.

Solution

For each component i of the vector function $f(\boldsymbol{x})$ we have a gradient vector in the corresponding column of $J(\boldsymbol{x})$. Each of the partial derivatives can be approximated by forward difference as follows:

$$\frac{\partial f_i(\boldsymbol{x})}{\partial x_j} \approx \frac{f_i(\boldsymbol{x} + h\boldsymbol{e}_j) - f_i(\boldsymbol{x})}{h}$$

$$J(\boldsymbol{x}) \approx \begin{bmatrix} \frac{f_1(\boldsymbol{x}+h\boldsymbol{e}_1)-f_1(\boldsymbol{x})}{h} & \frac{f_2(\boldsymbol{x}+h\boldsymbol{e}_1)-f_2(\boldsymbol{x})}{h} & \cdots & \frac{f_m(\boldsymbol{x}+h\boldsymbol{e}_1)-f_m(\boldsymbol{x})}{h} \\ \frac{f_1(\boldsymbol{x}+h\boldsymbol{e}_2)-f_1(\boldsymbol{x})}{h} & \frac{f_2(\boldsymbol{x}+h\boldsymbol{e}_2)-f_2(\boldsymbol{x})}{h} & \cdots & \frac{f_m(\boldsymbol{x}+h\boldsymbol{e}_2)-f_m(\boldsymbol{x})}{h} \\ \vdots & & \ddots & \\ \frac{f_1(\boldsymbol{x}+h\boldsymbol{e}_n)-f_1(\boldsymbol{x})}{h} & \frac{f_2(\boldsymbol{x}+h\boldsymbol{e}_n)-f_2(\boldsymbol{x})}{h} & \cdots & \frac{f_m(\boldsymbol{x}+h\boldsymbol{e}_n)-f_m(\boldsymbol{x})}{h} \end{bmatrix}$$

Note that the calculation of this matrix requires only $n + 1$ function evaluations and not $n \times m$ as one could naively think. This is because the same function evaluation can be used to compute the partial derivatives of all the components of f with respect to the same variable x_j. That is, the function $f(\boldsymbol{x})$ is computed once overall and the function $f(\boldsymbol{x} + h\boldsymbol{e}_j)$ is computed once for each direction e_j (hence once for every row of the matrix).
This matrix is called Jacobian matrix of f at $\boldsymbol{x}$. Often, for notational convenience, it is preferred to work with the transpose of his matrix, which has dimensions $m \times n$.

Exercise 7 + (2.1)

Adopt the forward difference method to approximate the Hessian of $f(x)$ using its gradient, $\nabla f(x)$.

Exercise 8 (2.6)

Combine the forward and backward difference methods to obtain a difference method for estimating the second-order derivative of a function f at x using three function evaluations.

Exercise 9 Python (2.3)

Implement in Python a finite difference method and the complex step method and compute the gradient of $f(x) = \ln x + e^x + 1/x$ for a point x close to zero. What term dominates in the expression?

Exercise 10 * (2.5)

Draw the computational graph for $f(x, y) = \sin(x+y^2)$. Use the computational graph with forward accumulation to compute $\frac{\partial f}{\partial y}$ at $(x, y) = (1, 1)$. Label the intermediate values and partial derivatives as they are propagated through the graph.

Solution

We will also solve the reverse accumulation for this question.
Forward accumulation Since we would like to find the partial derivative of the function with respect to y, evaluated at $(1, 1)$, we will compute at every node $\dot{c}_j := \dfrac{\partial c_j}{\partial y}$, where the final step will give us $\frac{\partial f}{\partial y}$, evaluated at $(1, 1)$. Thus, initially, $\dot{x} = \dfrac{\partial x}{\partial y} = 0$ and $\dot{y} = 1$. Then we go through the graph as follows: (a) $c_1 = y^2$, $\dot{c}_1 = 2y$, (b) $c_2 = c_1 + x$, $\dot{c}_2 = \dot{c}_1 + \dot{x}$, (c) $c_3 = \sin(c_2)$, $\dot{c}_3 = \cos(c_2).\dot{c}_2$. Since c_3 is the function output, $\dot{c}_3$, gives us the desired derivative.

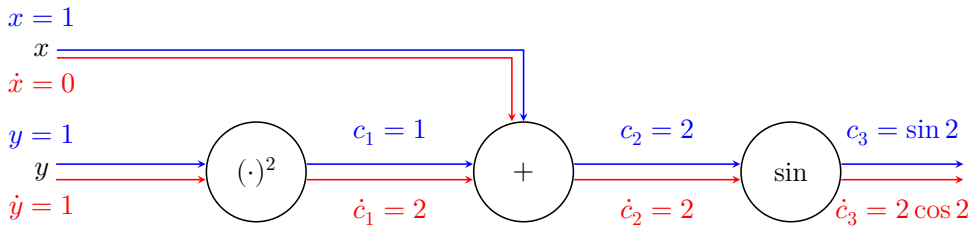


Figure 1: Forward Accumulation Computational Graph

Reverse accumulation Now, if we were to look at reverse accumulation, we still need the forward pass similar to above, but only of the values alone and not their gradients. To compute the gradients we need to work with the quantity that can be defined as $\bar{v} := \dfrac{\partial c_3}{\partial v}$. We do this because c_3 is the function output and we *backpropagate* the gradients at each node. Initially $\bar{c}_3 = 1$, by applying the definition and our goal is to compute $\bar{y}$, (we essentially get $\bar{x}$ along the way as well). Now we start from the output node and apply the following computational steps: (a) $\bar{c}_2 = \dfrac{\partial c_3}{\partial c_2} = \cos(c_2)$, (b) $\bar{c}_1 = \dfrac{\partial c_3}{\partial c_1} = \dfrac{\partial c_3}{\partial c_2}\dfrac{\partial c_2}{\partial c_1} = \bar{c}_2$ and in

similar vein $\bar{x} = \dfrac{\partial c_3}{\partial x} = \dfrac{\partial c_3}{\partial c_2}\dfrac{\partial c_2}{\partial x} = \bar{c}_2$. Finally, (c) $\bar{y} = \dfrac{\partial c_3}{\partial y} = \dfrac{\partial c_3}{\partial c_1}\dfrac{\partial c_1}{\partial y} = \bar{c}_1.(2y) = 2\cos(2)$.
Note that in the reverse accumulation algorithm, if a node has multiple edges going out from it, during the forward phase, when backpropagating the gradients one must sum up the contribtuions over all paths to compute the gradient correctly.

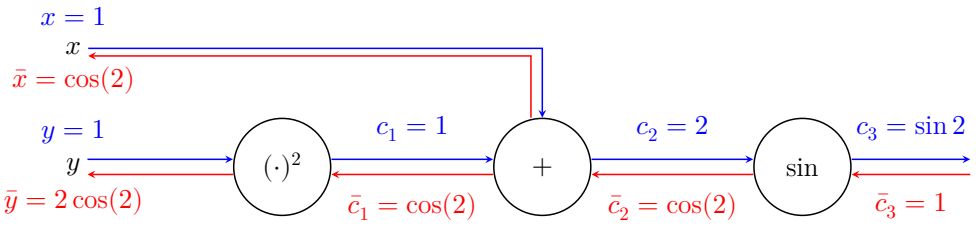


Figure 2: Reverse Accumulation Computational Graph

Exercise 11 * Python

Implement dual numbers in Python overriding the operators `+,-,*,/`. Test the implementation on the following operations:

- $\epsilon * \epsilon$
- $1/(1+\epsilon)$
- $(1+2\epsilon)*(3-4\epsilon)$

Calculate the forward accumulation of the dual numbers $a = 3 + 1\epsilon$ and $b = 2$ on the computational graph of $\log(a * b + \max(a, 2))$.

Solution

Dual numbers $D(a, b)$ can be writen as $a + b\epsilon$, where ϵ satisfies $\epsilon^2 = 0$, so we can drop all $O(\epsilon^2)$ terms. The four rules are:

- $(a + b\epsilon) \pm (c + d\epsilon) = (a \pm c) + (b \pm d)\epsilon$
- $(a + b\epsilon) \times (c + d\epsilon) = (ac) + (ad + bc)\epsilon$
- $(a + b\epsilon)/(c + d\epsilon) = (a/c) + (ad - bc)/c^2\epsilon$

As the other rules, the last one is *designed* such that the mulitpliers of ϵ implement the quotient rules of derivatives: Let $h(x) = \dfrac{f(x)}{g(x)}$, where both f and g are differentiable and $g(x) \neq 0$, the quotient rule states that the derivative of $h(x)$ is

$$h'(x) = \frac{f'(x)g(x) - f(x)g'(x)}{(g(x))^2}.$$

- $D(a, b) = a + b\varepsilon$
- We have $\varepsilon^2 = 0$ to recover function value and the first derivative.
- If f is a function of one variable, then $f(D(a, 1))$ will be $D(f(a), f'(a))$.
- If f is a function of two variables x, y, to get partial derivative with respect to x at some point $[x_0, y_0]$, do $f(D(x_0, 1), D(y_0, 0)) = D(f(x_0, y_0), \partial_x f(x_0, y_0))$
- $D(a, b) + D(c, d) = D(a + b, c + d)$
- $D(a, b) - D(c, d) = D(a - b, c - d)$

- $D(a,b) * D(c,d) = D(ac, ad + bc)$
- $D(a,b)/D(c,d) = D(a/c, (bc - ad)/c^2)$
- $\log(D(a,b)) = D(\log(a), b/a)$

$$\max\ (D(a,b),\ c) = \begin{cases} a + b\varepsilon, & a > c, \\ c + 0\varepsilon, & a \leq c, \end{cases}$$

The code below implements the aforementioned operations

```
import math

class Dual:
    """
    A Dual number that holds two numbers: a value and a gradient.
    """
    #__.__ is used to override the respective operations in python.
    def __init__(self, val: float | int, grad: float | int):
        assert type(val) in {float, int}
        assert type(grad) in {float, int}
        self.v = val
        self.g = grad

    def __add__(self: "Dual", other: "Dual") -> "Dual":
        return Dual(self.v + other.v, self.g + other.g)

    def __mul__(self: "Dual", other: "Dual") -> "Dual":
        return Dual(self.v * other.v, self.v * other.g + self.g * other.v)

    def __sub__(self: "Dual", other: "Dual") -> "Dual":
        return Dual(self.v - other.v, self.g - other.g)

    def __truediv__(self: "Dual", other: "Dual") -> "Dual":
        assert other.v != 0, "Division by a Dual with zero real part is undefined."
        return Dual(
            self.v / other.v,
            (self.g * other.v - self.v * other.g) / (other.v**2)
        )

    def __repr__(self):
        """__repr__ is used to override str representation. print(.) calls this."""
        return "Dual(v=%.4f, g=%.4f)" % (self.v, self.g)
```

We also need to implement how elementary functions would treat dual numbers, in particular the derivative that they apply:

$$\frac{\partial \ln(x)}{\partial x} = \frac{x'}{x}$$

$$\frac{\partial \max(x,p)}{\partial x} = \begin{cases} 0 & \text{if } p > x \\ x' & \text{if } p < x \end{cases}$$

```
def log(a: "Dual") -> "Dual":
    return Dual(math.log(a.v), a.g / a.v)

def max(a: "Dual", b: int) -> "Dual":
    return Dual(a.v if a.v > b else b, 0 if b >= a.v else a.g)
```

```
# Qa
a = Dual(0, 1)
b = Dual(0, 1)
print(a * b)
```

```
Dual(v=0.0000, g=0.0000)
```

```
# Qb
a = Dual(1, 0)
b = Dual(1, 1)
print(a/b)
```

```
Dual(v=1.0000, g=-1.0000)
```

```
#Qc
a = Dual(1, 2)
b = Dual(3, -4)
print(a * b)
```

```
Dual(v=3.0000, g=2.0000)
```

Finally, for the forward accumulation of the dual numbers $a = 3 + 1\epsilon$ and $b = 2$ on the computational graph of $\log(a * b + \max(a, 2))$, we have:

```
a = Dual(3, 1)
b = Dual(2, 0)
print(log(a * b + max(a, 2)))
```

```
Dual(v=2.1972, g=0.3333)
```

We get the same result as the text book:

```
Dual(v=2.1972, g=0.3333)
```

Exercise 12 * Python

Read about nanograd and use it to compute by reverse accumulation the gradient of

$$f(x_1, x_2, x_3) = \max\left\{0, \frac{x_1 + (-x_2 x_3)^2}{x_2 x_3}\right\}.$$

```python
import sys
sys.path.append('sheet01/code') # Ensure Python can find the folder
from nanograd import Var

# Point where you want the gradient
x1_0, x2_0, x3_0 = 3.0, 5.0, 9.0

# Create Variables
x1 = Var(x1_0)
x2 = Var(x2_0)
x3 = Var(x3_0)

# Define f(x1,x2,x3)
d = x2 * x3
e = x1 + (-d) ** 2
f = (e / d).relu()     # scalar output

# Reverse accumulation
f.backward()

print("f(x1,x2,x3) =", f.v)
print("df/dx1 =", x1.grad)
print("df/dx2 =", x2.grad)
print("df/dx3 =", x3.grad)
```

```
f(x1,x2,x3) = 45.06666666666667
df/dx1 = 0.022222222222222223
df/dx2 = 8.986666666666668
df/dx3 = 4.992592592592593
```