ENGR 21
Computer Engineering Fundamentals

Instructor: Emad Masroor

Lecture 18
Thursday, Nov 6, 2025

Multi-dimensional Optimization with Constraints

$$\text{Minimize } f(\vec{x}) = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

when $n = 2$ or 3,
$x_1 = x$
$x_2 = y$
$x_3 = z$

scalar function of an n-dimensional vector
OR
n scalars

$$\text{subject to} \begin{cases} g(\vec{x}) = 0 \\ h(\vec{x}) \leq 0 \end{cases}$$

Constraints

Multi-dimensional Optimization with Constraints

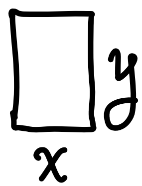
Maximize

$$\cancel{\text{Minimize}}\ f(\vec{x})$$

$$\text{subject to } \begin{cases} g(\vec{x}) = 0 \\ h(\vec{x}) \leq 0 \end{cases}$$

Find a rectangle with perimeter 20 having the largest area (out of all possible □'s with perimeter 20.

$f(x,y) = xy$ ← Maximize this.



Constraint : Perimeter must be 20.

$$2x + 2y = 20 \Rightarrow \underbrace{2x+2y-20}_{g(x,y)} = 0$$

$g(x,y) = 0$ ← Satisfy this constraint

Let $f^*(x,y) \equiv f(x,y) + \lambda \cdot [g(x,y)]^2$

λ: Lagrange multiplier
it's some scalar

Maximize f^*, not f

Multi-dimensional Optimization with Constraints

Maximize f^* with respect to x, y and λ.

$$\frac{\partial f^*}{\partial x} = 0$$

$$\frac{\partial f^*}{\partial y} = 0$$

$$\frac{\partial f^*}{\partial \lambda} = 0$$

if derivatives of f^* are linear in x, y, λ this can be solved as

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} x \\ y \\ \lambda \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

in general, optimization problems are nonlinear.

Note: $f(\vec{x}) = \vec{b}$ is hard. — in 1-d, we did this. "Root finding"

$A\vec{x} = \vec{b}$ is easier — can do for n-dimensions

Multi-dimensional Optimization with Constraints

$$\text{Minimize } f(\vec{x})$$

$$\text{subject to } \begin{cases} g(\vec{x}) = 0 \\ h(\vec{x}) \leq 0 \end{cases}$$

Strategy:

$$f^*(\vec{x}) = f(\vec{x}) + \lambda_g \left[g(\vec{x})\right]^2 + \lambda_h \left[\max\{0, h(\vec{x})\}\right]^2$$

Example: multi-dim. optim. with constr.

Find the rectangle with perimeter 20 having the largest area.

* : there is a more intelligent way.

Numerical Technique: Naive* n-dimensional optimization

Successively carry out 1-d optimizations in **n** directions.

e.g. if n = 3 design variables, your 3 directions could be:

$$\left\{ \underbrace{\begin{bmatrix}0\\0\\1\end{bmatrix}}_{\vec{V_1}}, \underbrace{\begin{bmatrix}0\\1\\0\end{bmatrix}}_{\vec{V_2}}, \underbrace{\begin{bmatrix}1\\0\\0\end{bmatrix}}_{\vec{V_3}} \right\}$$

Cycle through the n directions as many times as needed.

$$\text{Minimize } f(\vec{x})$$

$$\vec{x} = [x_1, x_2, ..., x_n]$$

$$\{\vec{v_1}, \vec{v_2}, ..., \vec{v_n}\}$$

1-d optimizations along specified directions:

Guess

$\overrightarrow{x_0} \rightarrow \overrightarrow{x_1} \rightarrow \overrightarrow{x_2} \rightarrow \dots$

Optimize along $\overrightarrow{v_1}$

(1-d optimize)

Find α_{min} that minimizes $f(\overrightarrow{x_0} + \alpha \overrightarrow{v_1})$

using

that α

Increment guess in direction $\overrightarrow{v_1}$ using step size α

$$\overrightarrow{x_1} = \overrightarrow{x_0} + \alpha_{min} \overrightarrow{v_1}$$

Optimize along $\overrightarrow{v_2}$

Using any 1-d optimization technique

1-d optimizations along specified directions:

$$\begin{tikzcd}
\vec{x}_0 \arrow[r, "\vec{v}_1"] & \vec{x}_1 \arrow[r, "\vec{v}_2"] & \vec{x}_2 \arrow[r, "\vec{v}_3"] & \vec{x}_3 \arrow[r, "\vec{v}_1"] & \vec{x}_4 \arrow[d, "\vec{v}_2"] \\
 & \cdots & \vec{x}_7 \arrow[l] & \vec{x}_6 \arrow[l, "\vec{v}_1"'] & \vec{x}_5 \arrow[l, "\vec{v}_3"']
\end{tikzcd}$$

say $n = 3$

Iterative method.

- Terminate when $\vec{x}_{j+1}$ is not too different from $\vec{x}_j$
- Check residual

e.g. find the minimum of $f(\vec{x}) = f(x,y) = (x-1)^2 + (y+1)^2 + xy$

- Start from [-1,+2]
- Use the directions:
  - [0,1]
  - [1,0]

Illustrating Naive n-d optimization

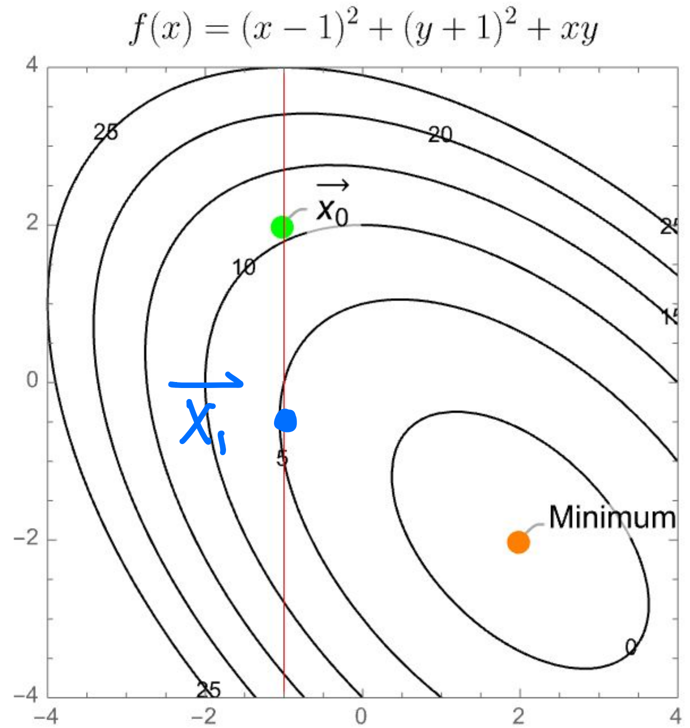
First, we'll do the direction $\vec{v}_1 \; \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

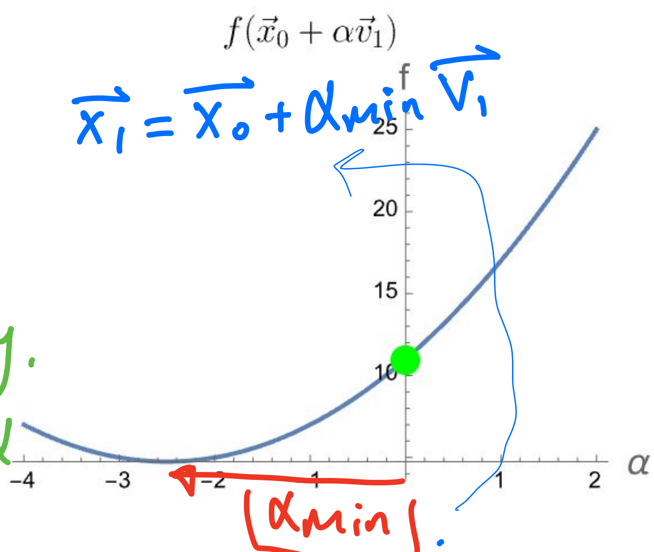
Then, direction $[1, 0] \; \vec{v}_2$

Calculate an expression for $f(x,y)$ along this particular straight line.

$$\boxed{f(\vec{x}_0 + \underbrace{\alpha}_{\text{independent variable}} \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{\text{chosen direction}})}$$

This is NOT a function of x, y. It's a func. of α





Illustrating Naive n-d optimization

$$f([x, y]) = (x-1)^2 + (y+1)^2 + xy$$

Minimize $f(\boldsymbol{x_0} + \alpha \boldsymbol{v_1})$

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

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

$$= (-1-1)^2 + (2+\alpha+1)^2 + (-1+0)(2+\alpha)$$

$$= 4 + \alpha^2 + 6\alpha + 9 - 2 - \alpha$$

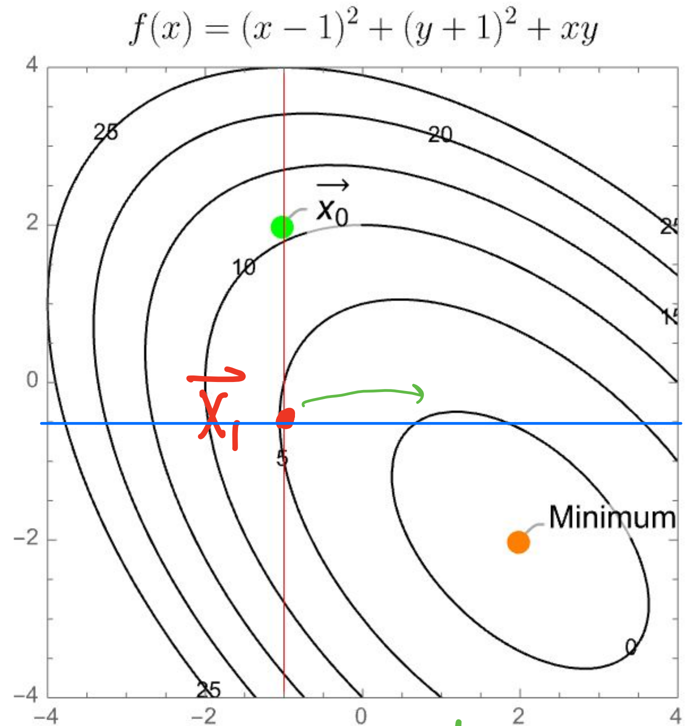
$$= \alpha^2 + 5\alpha + 11$$

Minimize $\alpha^2 + 5\alpha + 11$

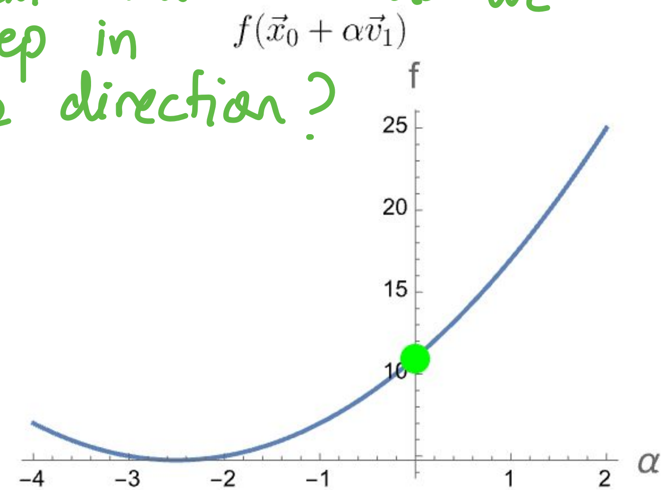
$$\implies \alpha_{\min} = -5/2$$

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \alpha_{\min}\boldsymbol{v}_1$$

$$= \begin{bmatrix}-1\\2\end{bmatrix} - \frac{5}{2}\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}-1\\-0.5\end{bmatrix}$$



Next: how much do we step in $\vec{v}_2$ direction?



Powell's Method

Powell's method is one of many n-dimensional optimization methods used today.

method : ***str or callable, optional***

Type of solver. Should be one of

- 'Nelder-Mead' (see here)
- 'Powell' (see here)
- 'CG' (see here)
- 'BFGS' (see here)
- 'Newton-CG' (see here)
- 'L-BFGS-B' (see here)
- 'TNC' (see here)
- 'COBYLA' (see here)
- 'COBYQA' (see here)
- 'SLSQP' (see here)
- 'trust-constr'(see here)
- 'dogleg' (see here)
- 'trust-ncg' (see here)
- 'trust-exact' (see here)
- 'trust-krylov' (see here)
- custom - a callable object, see below for description.

If not given, chosen to be one of `BFGS`, `L-BFGS-B`, `SLSQP`, or not the problem has constraints or bounds.

`scipy.optimize.`
`minimize`

`minimize(fun, x0, args=(), method=None, jac=None, hess=None, hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None)`

Minimization of scalar function of one or more variables. [source]

Parameters:

fun : ***callable***

The objective function to be minimized.

```
fun(x, *args) -> float
```

where `x` is a 1-D array with shape (n,) and `args` is a tuple of the fixed parameters needed to completely specify the function.

x0 : ***ndarray, shape (n,)***

Initial guess. Array of real elements of size (n,), where `n` is the number of independent variables.

args : ***tuple, optional***

Extra arguments passed to the objective function and its derivatives (*fun*, *jac* and *hess* functions).

method : ***str or callable, optional***

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html