Real-world Multi-agent Systems

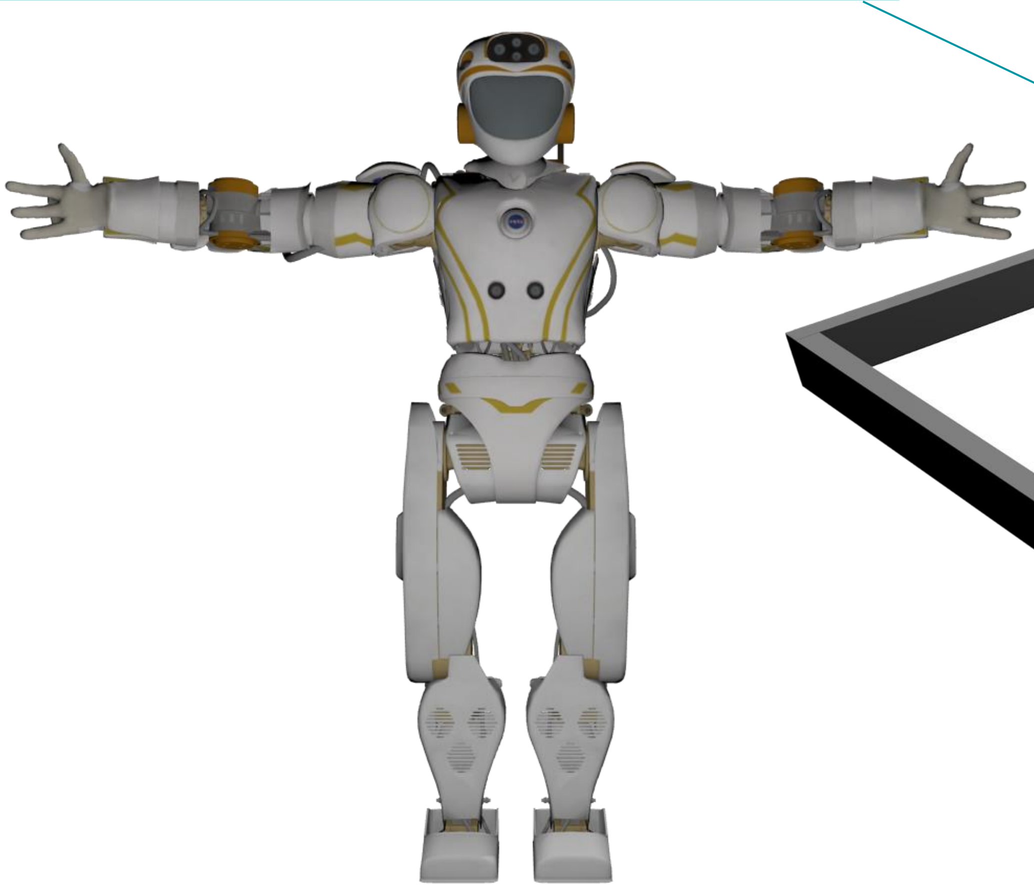
Advanced Optimization

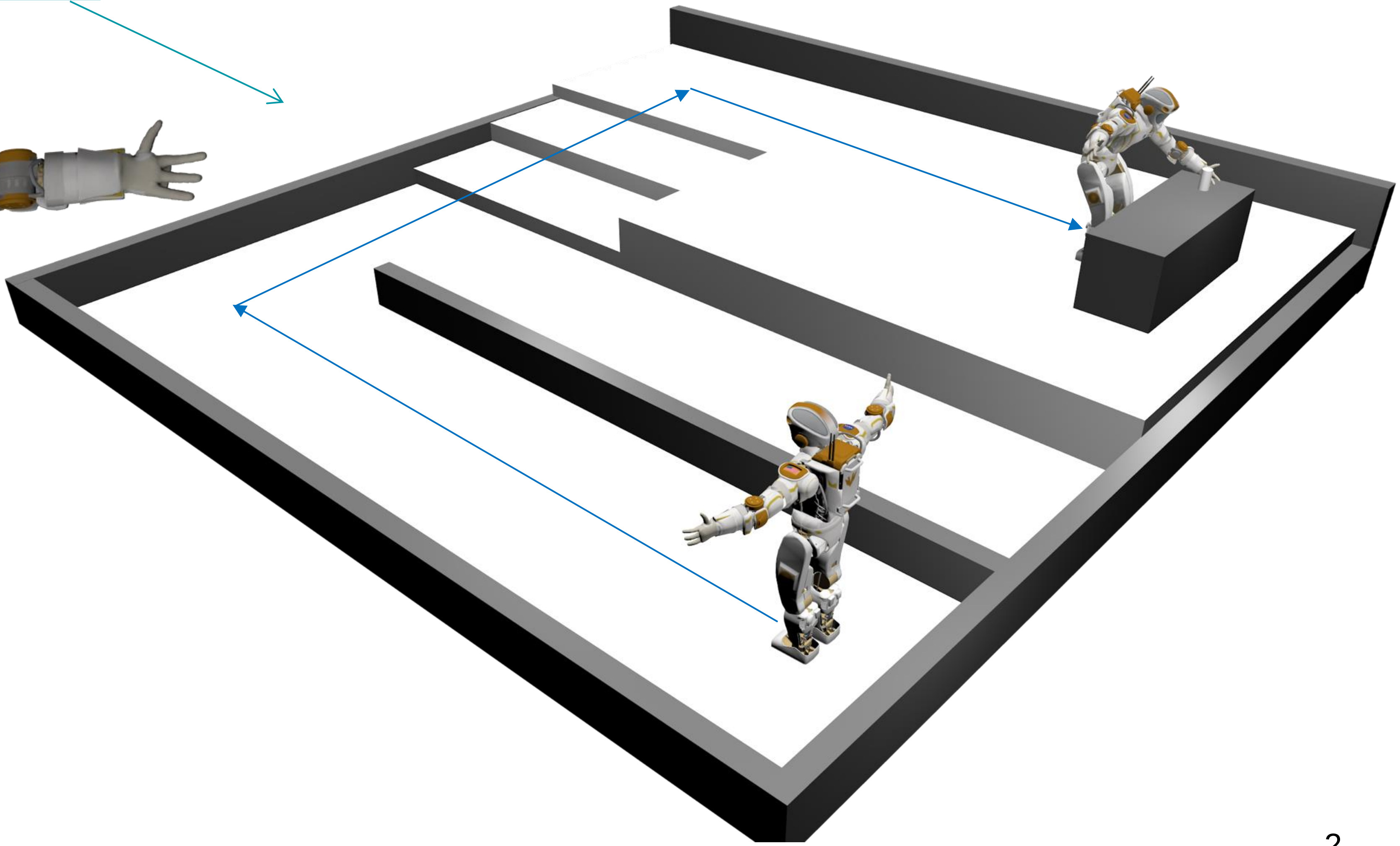
Akin Delibasi
Lecturer in Distributed Systems
Computer Science
University College London, UK

Optimisation?

Optimisation question:
What is the best for doing so?

Motion Planning answers: How do I get there?





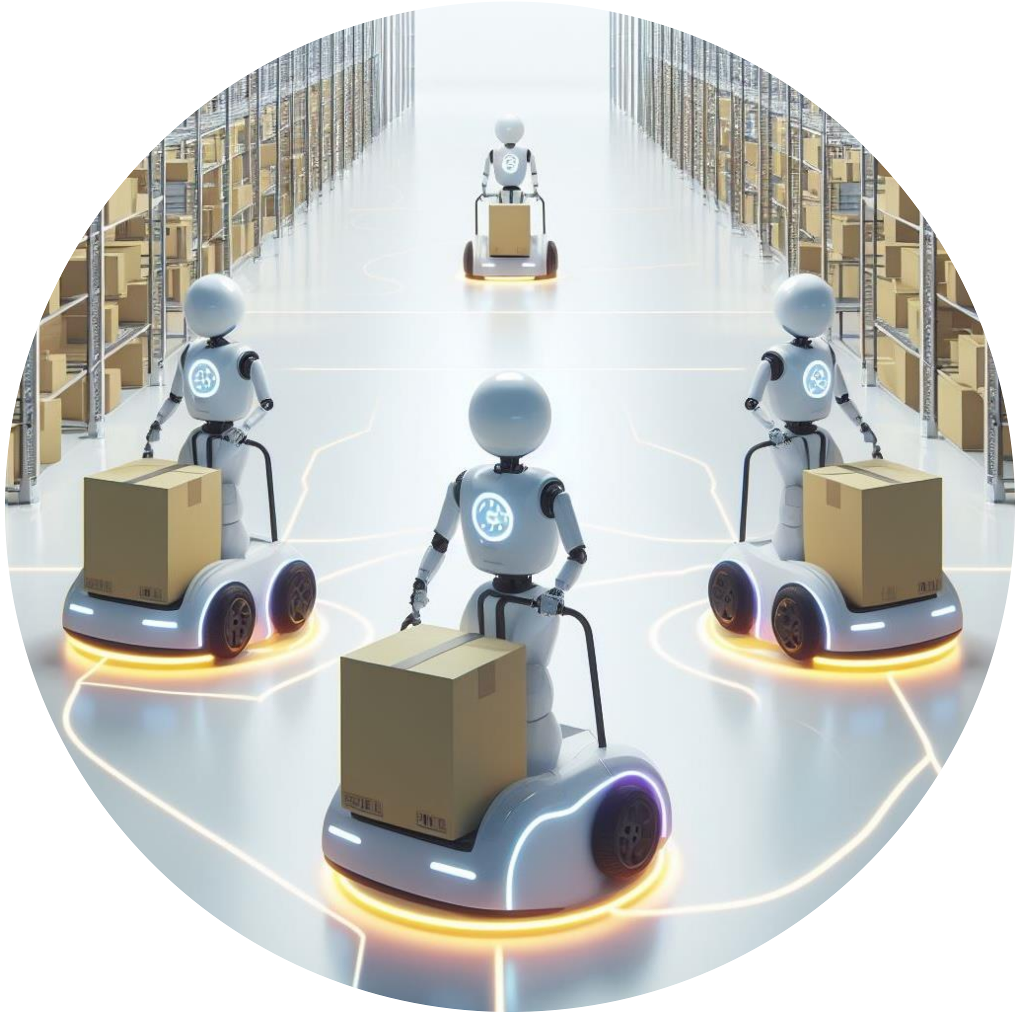
Content

- Concept of optimization: what, why, and how?
- Linear & unconstrained optimization
  - ❑ Least Square (LS) optimization
- Gradient-based optimization
- Non-linear numerical optimization (more generic)

What is Optimization?

Optimization: in simple terms, to find the *best* solution for a particular problem.

It can be used for signal/data processing, motion planning, identifying models, robot control, multi-agent cooperation and more – finding the best, feasible solution for these systems to operate/perform.



Concept and Definition

Minimization or maximization the value of a cost function ***f*** (also written as ***J***).

The function ***f*** is called an **objective function**. Usually, the **objective function** refers to the norm of errors, so we also use the name **cost function**, followed by an exchangeable notion of ***J*** as well.

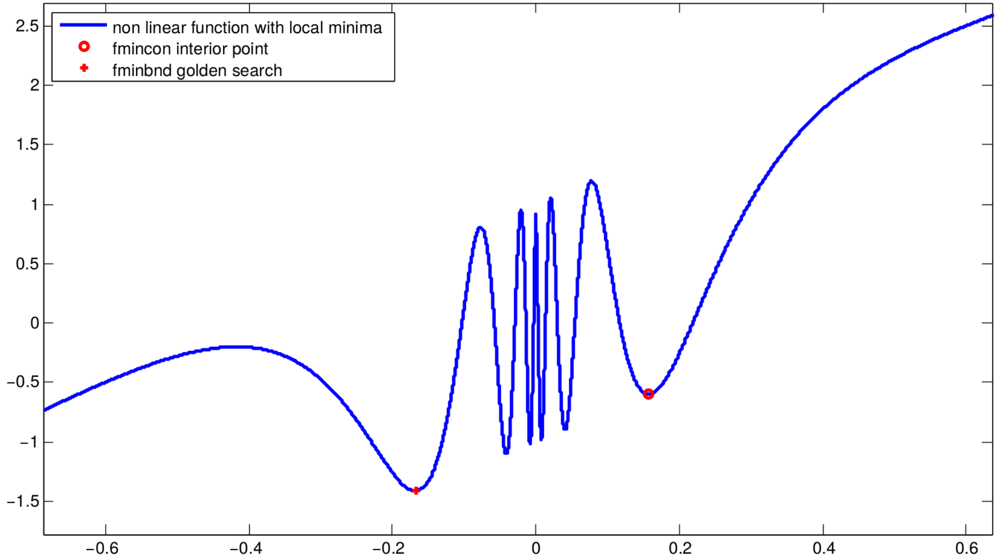
Minimization or maximization depends on the formulation of ***f*** :

- Maximize rewards;
- Minimize penalties and errors.

Definition

- **Cost Function / Objective Function**: The goal is to either **maximize** or **minimize** this function, often denoted as ***f***(***x***), or ***J***(***x***).
- **Decision Variables**: **variables** (usually it is a set ***x***) that can be adjusted to achieve the optimal solution.

- **Optima Types**:
  - ❑ Local Optimum: A solution better than neighboring solutions but not necessarily the best.
  - ❑ Global Optimum: The best possible solution across the entire domain.



Types and Scope

Broadly, we can consider two types: discrete optimization and continuous optimization.

Here, we only introduce **continuous optimization**, where the variables are allowed to take on any value within a range of values, usually **real numbers**.

Out of scope: Discrete optimization, where some or all of the variables may be binary, integer, or more abstract objects drawn from sets with finitely many elements, eg mixed-integer programming (MIP).

Unconstrained optimization

Unconstrained optimization

Unconstrained optimization considers the problem of minimizing an objective function ***f*** that depends on real variables with no restrictions on their values.

Mathematically, let $\boldsymbol{x} \in \boldsymbol{R}^n$ be a n-dimensional vector (n≥1);

and let $\boldsymbol{f}(\boldsymbol{R}^n) \to \boldsymbol{R}$ be a smooth function.

Unconstrained optimization problem is simply to find $\boldsymbol{x} \in \boldsymbol{R}^n$ such that

$$\operatorname{argmin}\ \boldsymbol{f}(\boldsymbol{x})$$

Often it is also practical to replace the constraints of an optimization problem with penalized terms in the objective function, and to solve the problem faster as an unconstrained problem. (see our demo code later).

Least Square (LS) optimization

- Least Squares (LS) optimization:
  - ❑ An unconstrained minimization
  - ❑ A linear optimization method
  - ❑ Objective function in a quadratic form
  - ❑ Model parameters and its representation follow linear algebra
  - ❑ Solvable using linear algebra techniques

- **Objective**: Least Squares method is to minimize the **sum of the squared errors (SSE)**, ie the differences (residuals) between observed values (data points) and values estimated by a model.

Least Square (LS) optimization

Suppose we want to find the value of **x** that minimizes

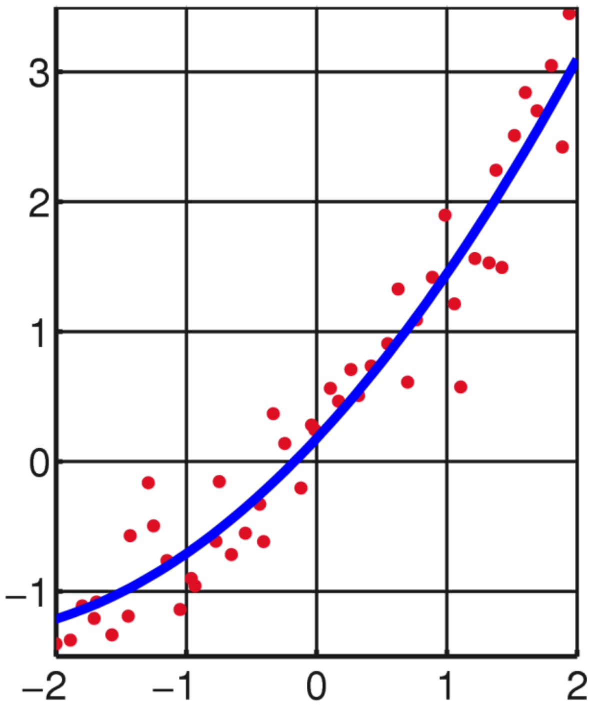
$$\boldsymbol{f}(\boldsymbol{x}) = ||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||^2$$

Let $\boldsymbol{A}\boldsymbol{x} = y$, then a more intuitive form of this minimization is

$$\boldsymbol{f}(\boldsymbol{x}) = \sum (y_i - b_i)^2$$

A*x=y is what you estimate, ***b*** is the measured value.

How to interpret this?



Least Square (LS) optimization

So, given

$$\boldsymbol{f}(\boldsymbol{x}) = ||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||^2$$

For minimizing ***f***(**x**), ideally we would like to have

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \ (\boldsymbol{f}(\boldsymbol{x}) = 0)$$

$$\boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

Least Square (LS) optimization

In order to have $\boldsymbol{x} = \boldsymbol{A}^{-\mathbf{1}}\boldsymbol{b}$ (eg $\boldsymbol{A}^{-\mathbf{1}}$) to be valid, there are several assumptions (recap of linear algebra):

1. $\boldsymbol{A}$ is a $n \times n$ square matrix
2. $\boldsymbol{A}$ has full rank, ie $\operatorname{rank}(\boldsymbol{A}) = n$
3. $\boldsymbol{A}$ is nonsingular (its determinant is *Not* 0, $\det(\boldsymbol{A}) \neq 0$

However, in most cases, the assumption of $\boldsymbol{A}$ is **not** valid. Usually, $\boldsymbol{A}$ is a $m \times n$ matrix, ie $\boldsymbol{A} \in M(m, n)$ and $(m > n)$. $\boldsymbol{A}$ is a rectangular matrix, not a square matrix.

Least Square (LS) optimization

So, given

$$\boldsymbol{f}(\boldsymbol{x}) = ||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||^2$$

we have specialized linear algebra algorithms that can solve $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$ problem efficiently by *pseudo inverse*:

$$\boldsymbol{x} = \boldsymbol{A}^{\dagger}\boldsymbol{b}$$

Least Square (LS) optimization

So, the smallest-norm solution to the unconstrained least squares problem

$$\boldsymbol{f}(\boldsymbol{x}) = ||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||^2$$

is:

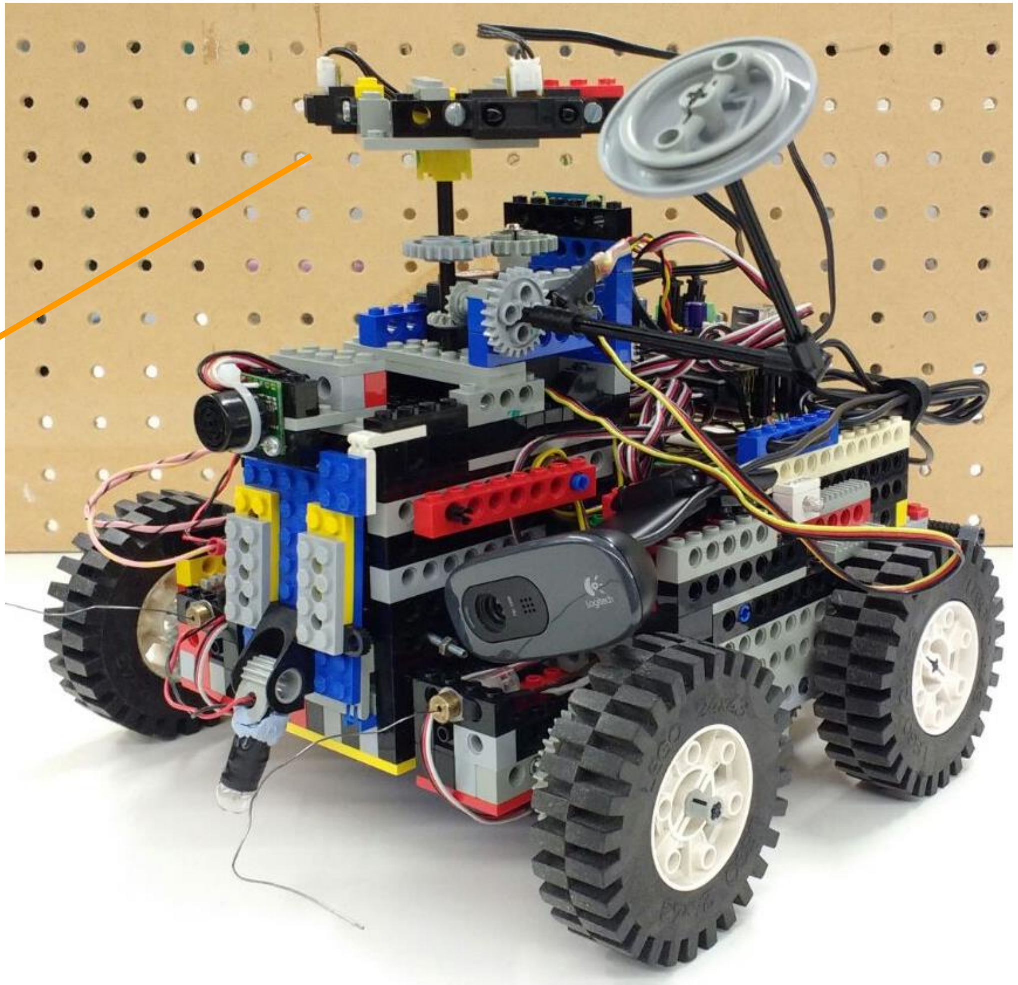
$$\boldsymbol{x} = \boldsymbol{A}^{\dagger}\boldsymbol{b}$$

where $\boldsymbol{A}^{\dagger}$ is the left pseudoinverse

$$\boldsymbol{x} = [(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T] \cdot \boldsymbol{b}$$

Application in real practice

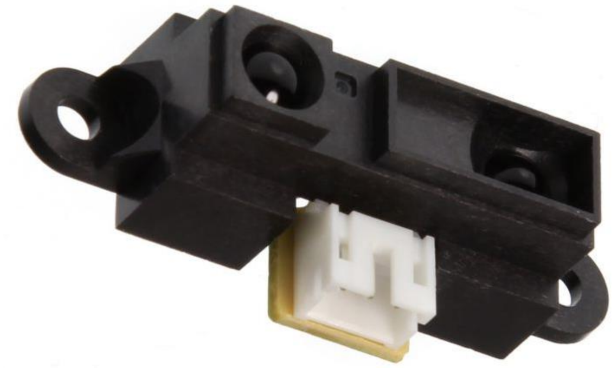
More accurate estimation of distance from sensor readings, using real measurement and data to “model” the relation between sensor reading and distance.

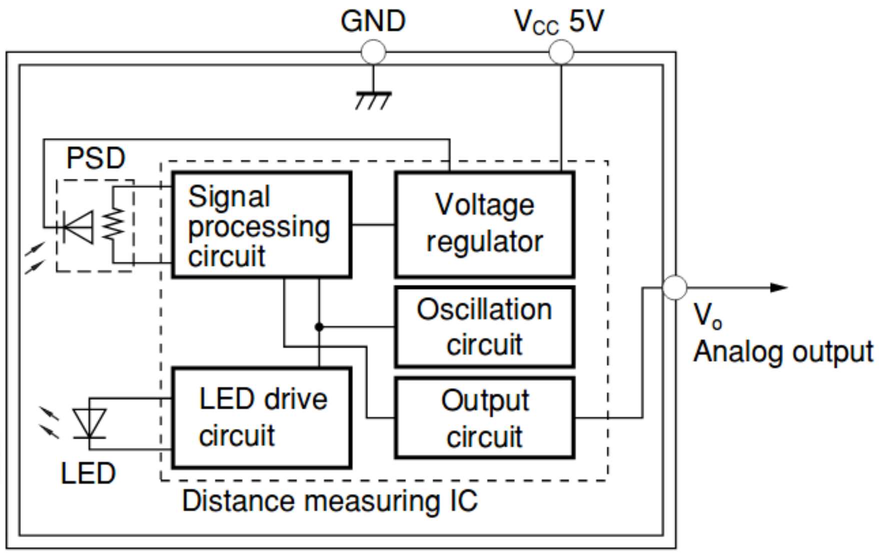


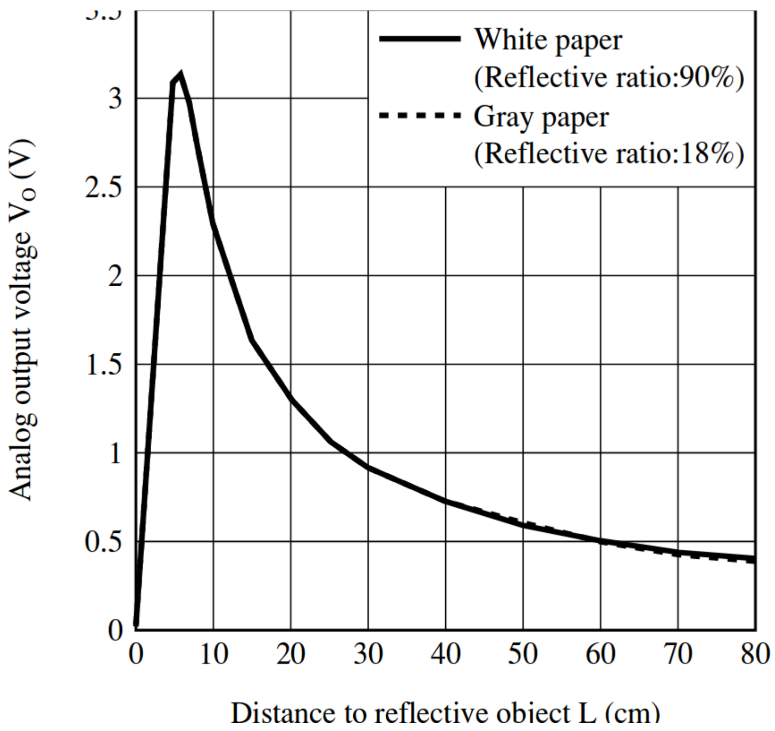
Application in real practice

Strength of reflection is inverse proportional to the distance.

Voltage is *inverse proportional* to the distance.

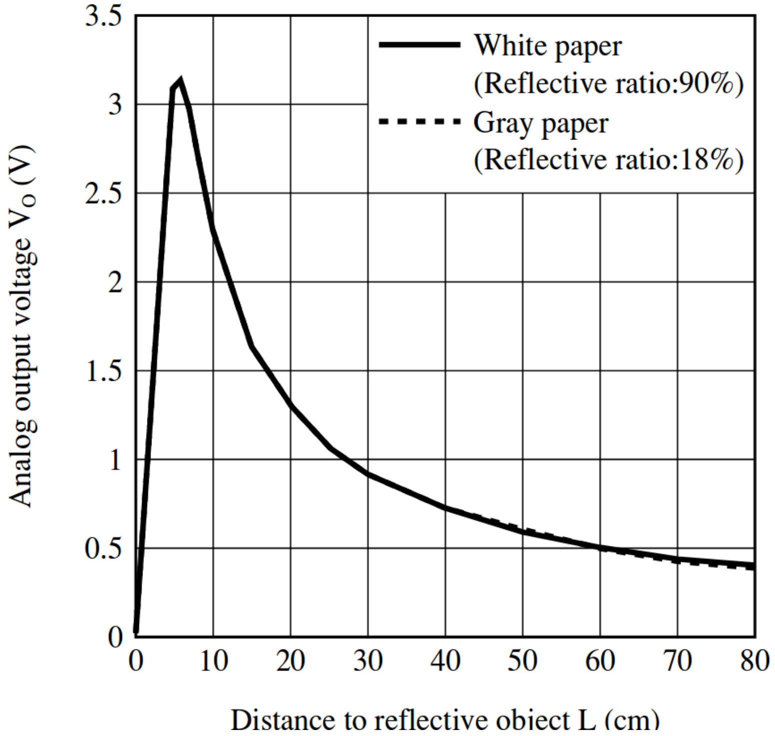


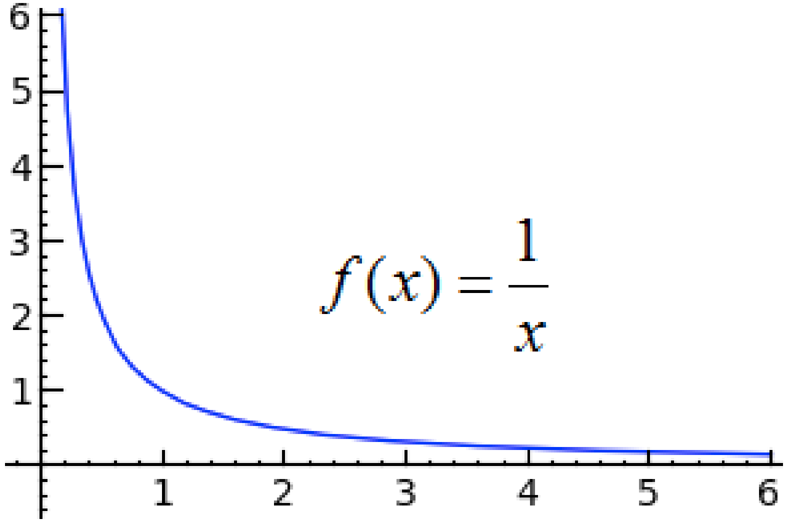


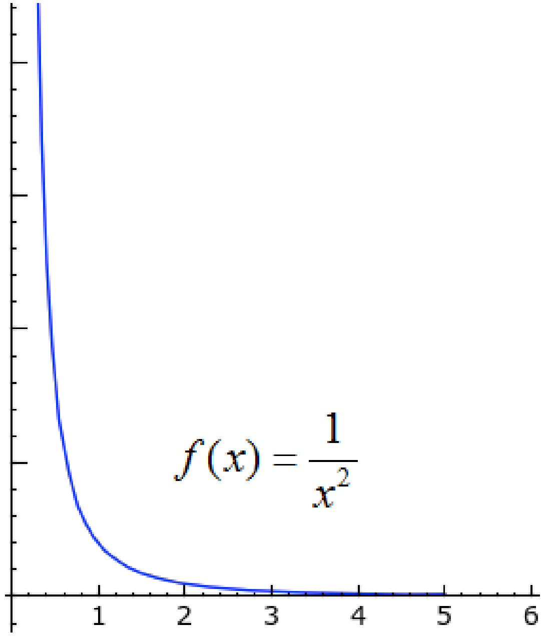


Correlating distance and the voltage

Voltage is *inverse proportional* to the distance. Some basic functions that reflect the inverse proportional relation.







Correlating distance and the voltage

Given a voltage measurement, we want to calculate the distance.

Define x as the voltage (scalar), y as the distance (scalar), voltage-distance has *inverse proportional* relation, use polynomial to correlate the relation as:

$$y = a_0 + a_1 x^{-1} + a_2 x^{-2} + a_3 x^{-3} + a_4 x_1^{-4}$$

Stack data from multiple measurements into matrix form as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} 1, x_1^{-1}, x_1^{-2}, x_1^{-3}, x_1^{-4} \\ 1, x_2^{-1}, x_2^{-2}, x_2^{-3}, x_2^{-4} \\ \vdots \\ 1, x_k^{-1}, x_k^{-2}, x_k^{-3}, x_k^{-4} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

Correlating distance and the voltage

$$\underbrace{\begin{bmatrix} 1, x_1^{-1}, x_1^{-2}, x_1^{-3}, x_1^{-4} \\ 1, x_2^{-1}, x_2^{-2}, x_2^{-3}, x_2^{-4} \\ \vdots \\ 1, x_k^{-1}, x_k^{-2}, x_k^{-3}, x_k^{-4} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix}}_{b}$$

Coefficients are the unknowns here

Map our formulation to the LS equation. Solution of pseudo inverse

$$x = A^{\dagger} b$$

Correlating distance and the voltage

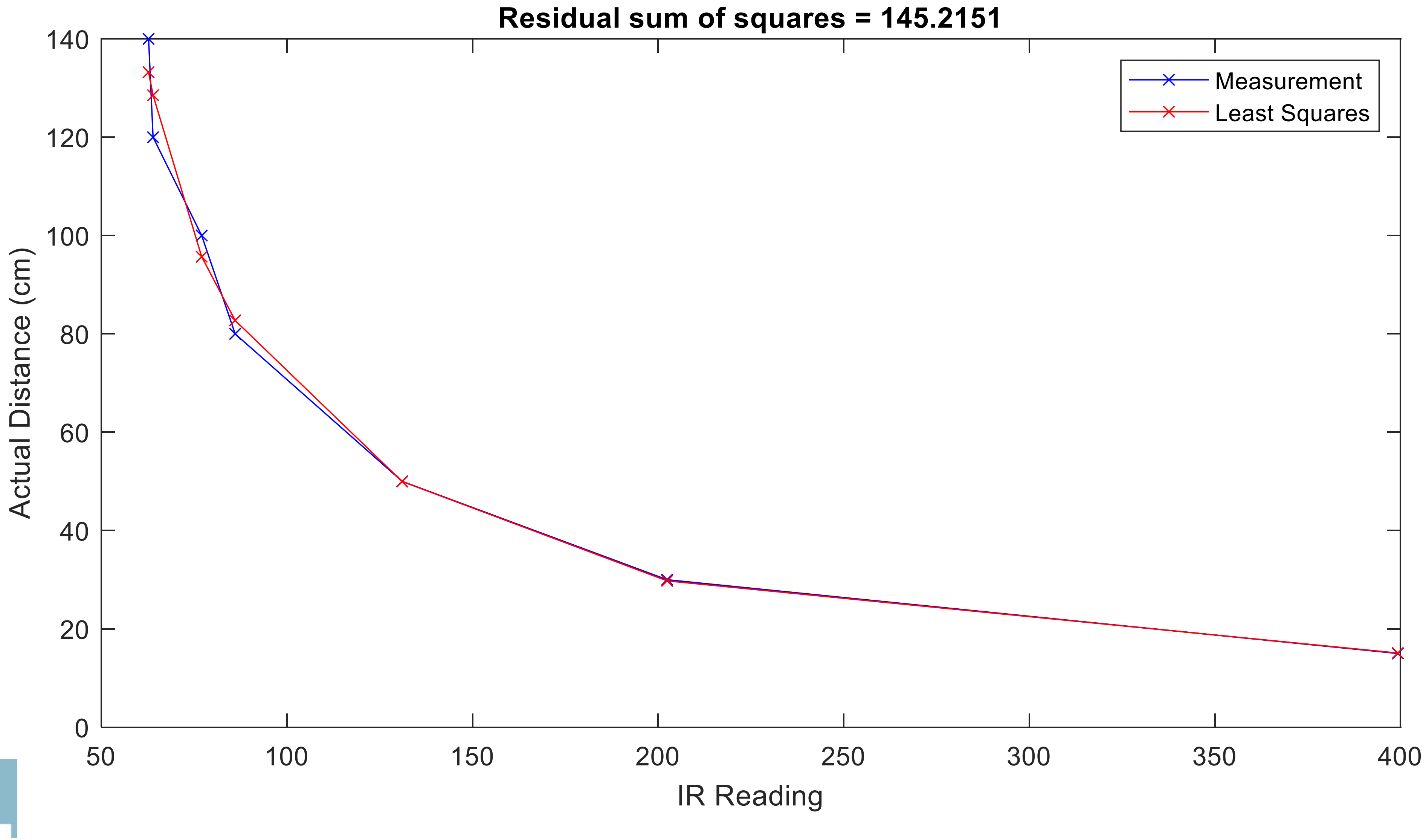
Solution of pseudo inverse:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 1, x_1^{-1}, x_1^{-2}, x_1^{-3}, x_1^{-4} \\ 1, x_2^{-1}, x_2^{-2}, x_2^{-3}, x_2^{-4} \\ \vdots \\ 1, x_k^{-1}, x_k^{-2}, x_k^{-3}, x_k^{-4} \end{bmatrix}^{\dagger} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix}$$

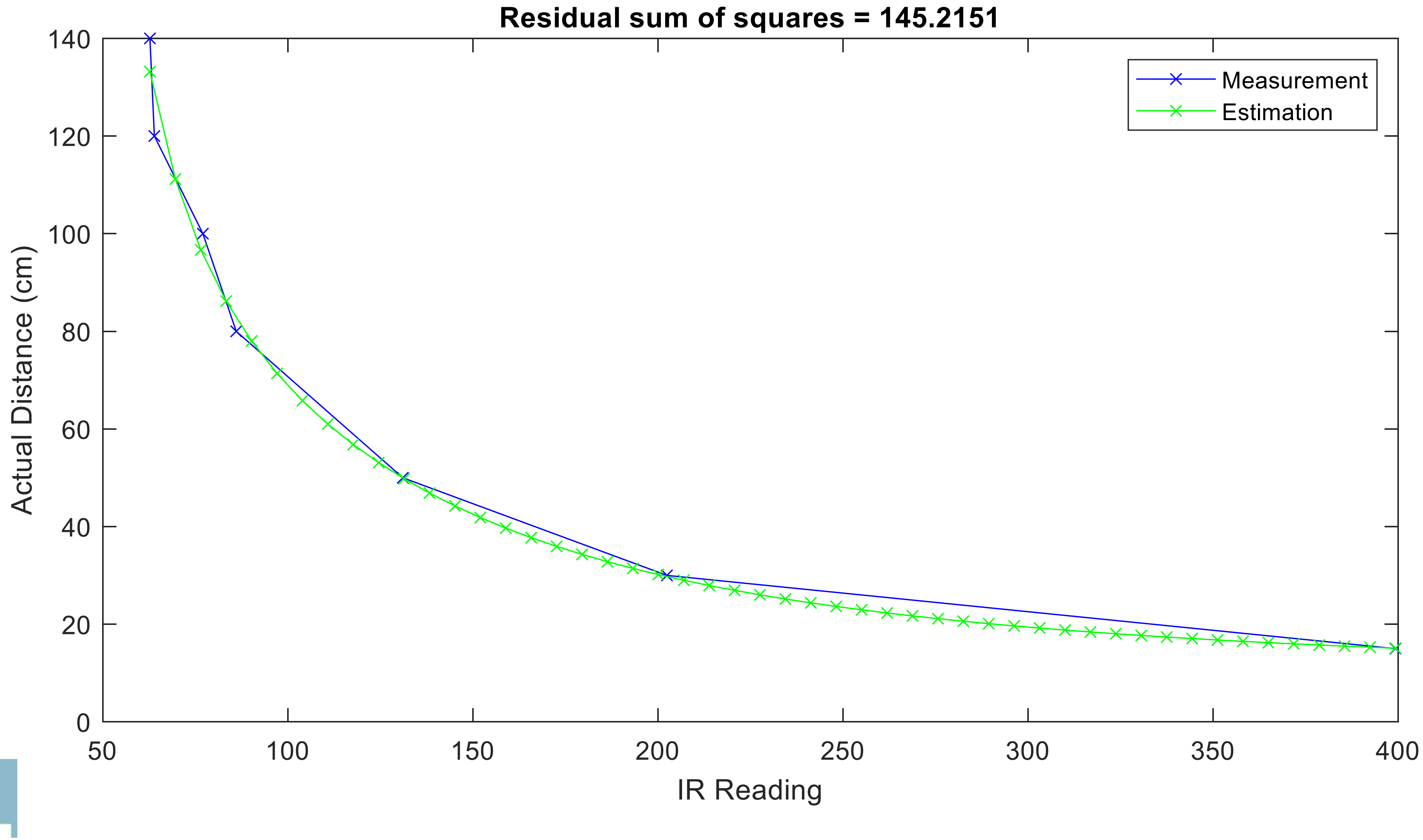
After **obtaining coefficients**, we can apply equation below for any new measured voltage to calculate the distance

$$y = a_0 + a_1 x^{-1} + a_2 x^{-2} + a_3 x^{-3} + a_4 x_1^{-4}$$

Results - from coarse samples to fine-grained estimation



Results - from coarse samples to fine-grained estimation



Summary of unconstrained optimization

Least Squares (LS) optimization is an unconstrained minimization, and a linear optimization method, which has analytic solutions.

Pros: analytic solution, fast computation, suitable for real-time implementation.

Cons: limited complexity.

Note: In real systems, physical quantities are bounded, ie systems are subject to *hard* physical constraints, because materials, energy and power are not infinite.

Quiz: what are the constraints in a physical system?

Gradient-Based Optimization

Optimization: gradient descent

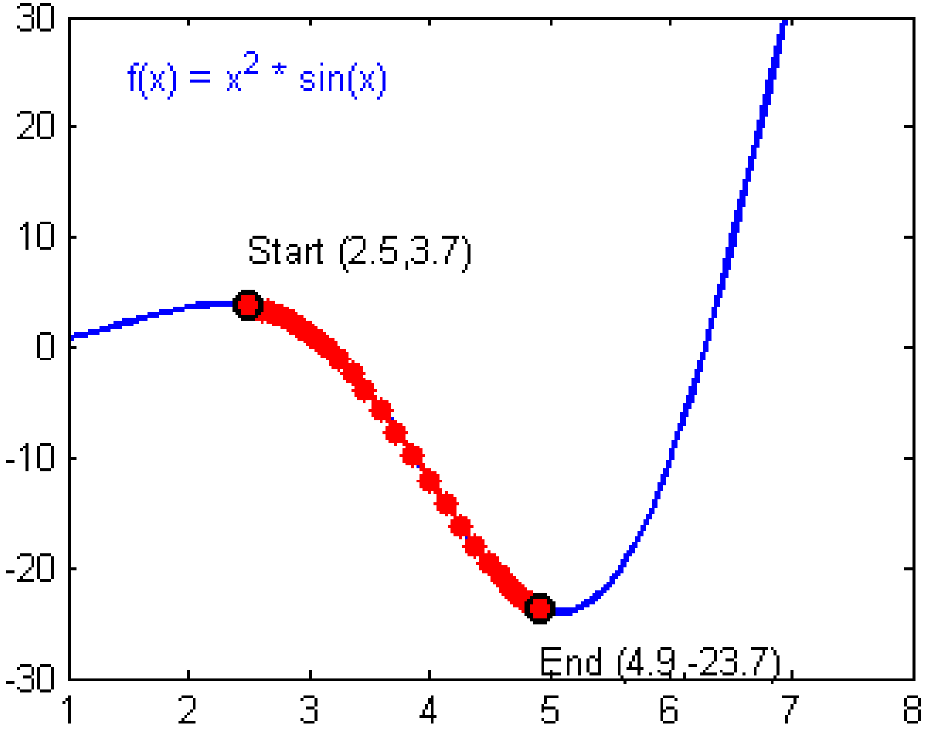
Gradient descent optimization: use the first-order iterative to find the *minimum* of a function. The derivatives of a function can be used to follow the function downhill to a minimum. This technique is called *gradient descent*.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{\alpha} \cdot \bigtriangledown \boldsymbol{f}(\boldsymbol{x}_k)$$

The *gradient descent* algorithm takes incremental steps proportional to the negative of the gradient of the function at the current point.

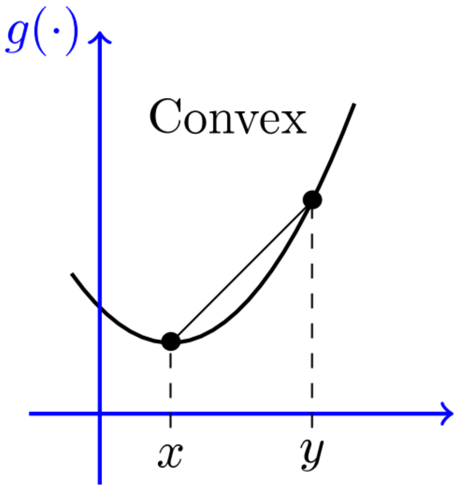
Similarly, *gradient ascent* is to find the local maxima, it takes steps proportional to the positive of the gradient.

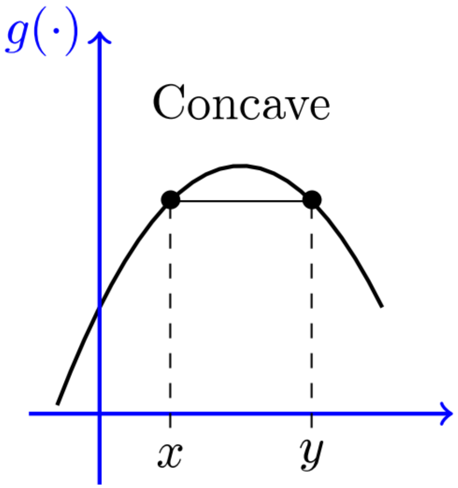
Descending with step coefficient 0.005 (iteration 50)

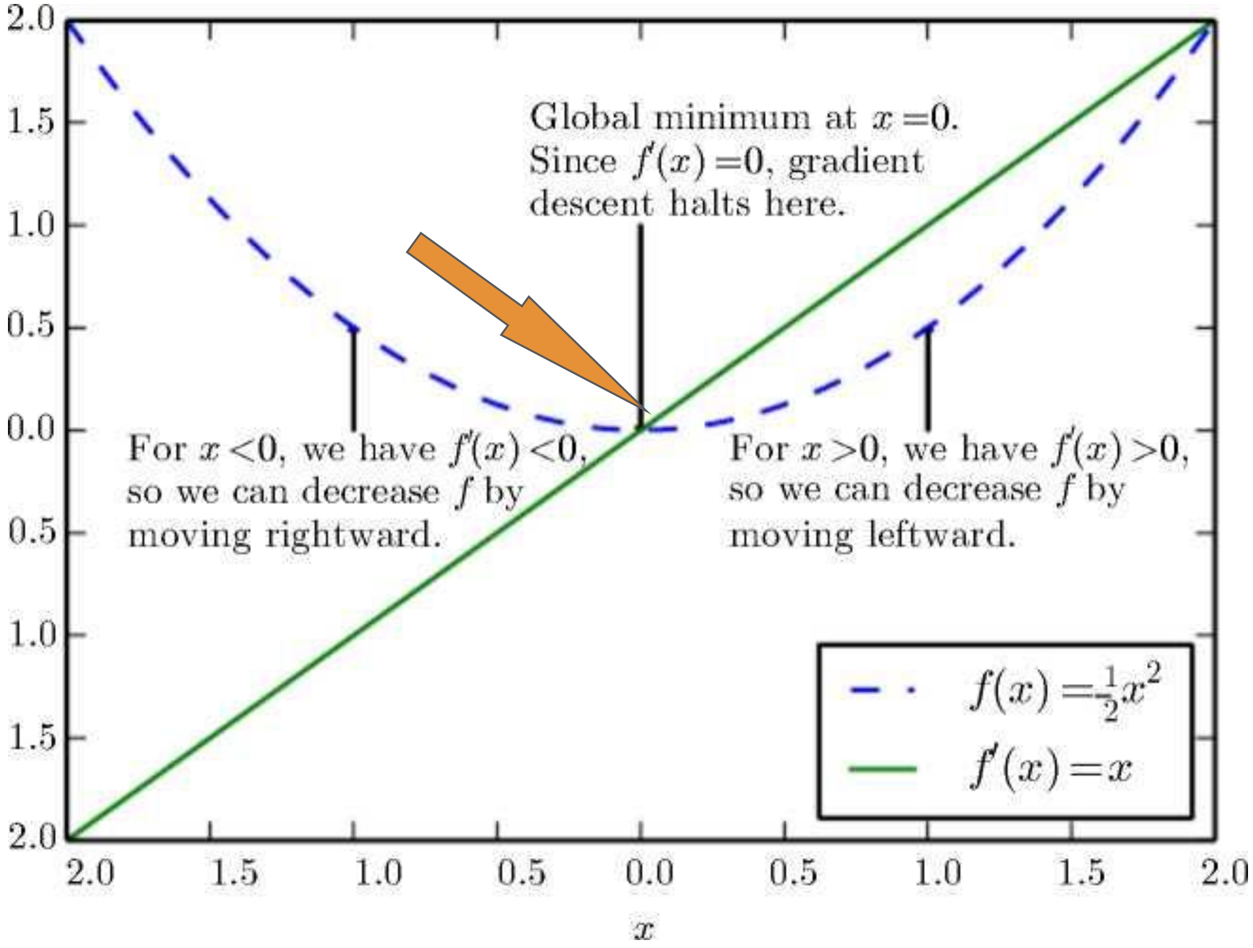


Gradient descent

$\triangledown \boldsymbol{f}(x) = 0$ is at $x = 0$, where convex or concave function $\boldsymbol{f}(x)$ reaches extrema.







Global minimum at $x = 0$. Since $f'(x) = 0$, gradient descent halts here.

For $x < 0$, we have $f'(x) < 0$, so we can decrease f by moving rightward.

For $x > 0$, we have $f'(x) > 0$, so we can decrease f by moving leftward.

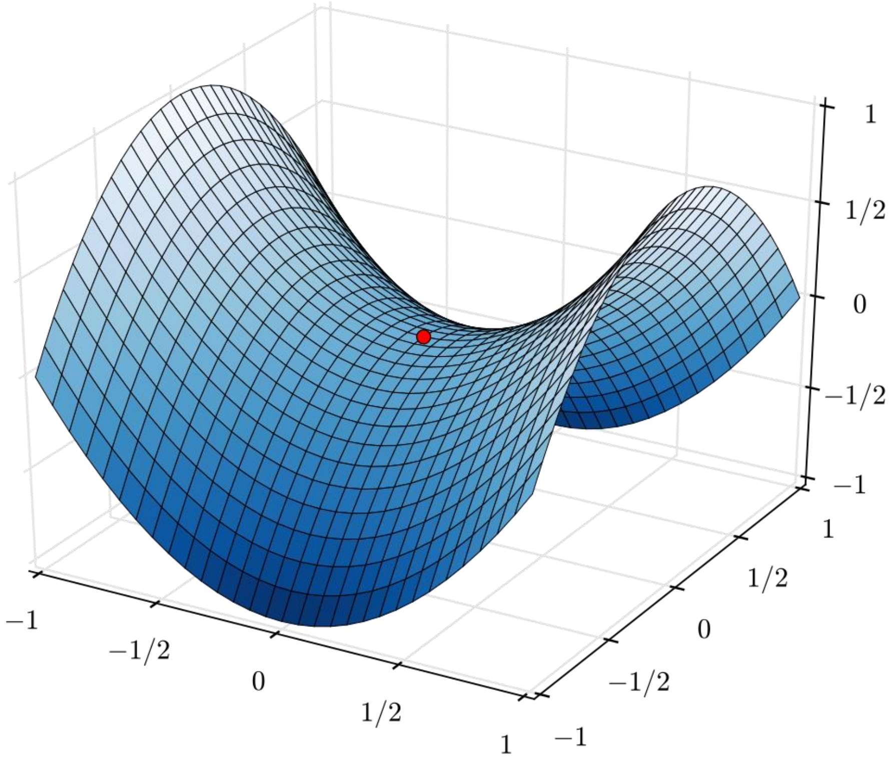
Picture source: Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

Saddle point

$\bigtriangledown \boldsymbol{f}(x) = 0 \Rightarrow \boldsymbol{f}(x)$ at extrema is not necessarily true when $\boldsymbol{f}(x)$ becomes more complicated.

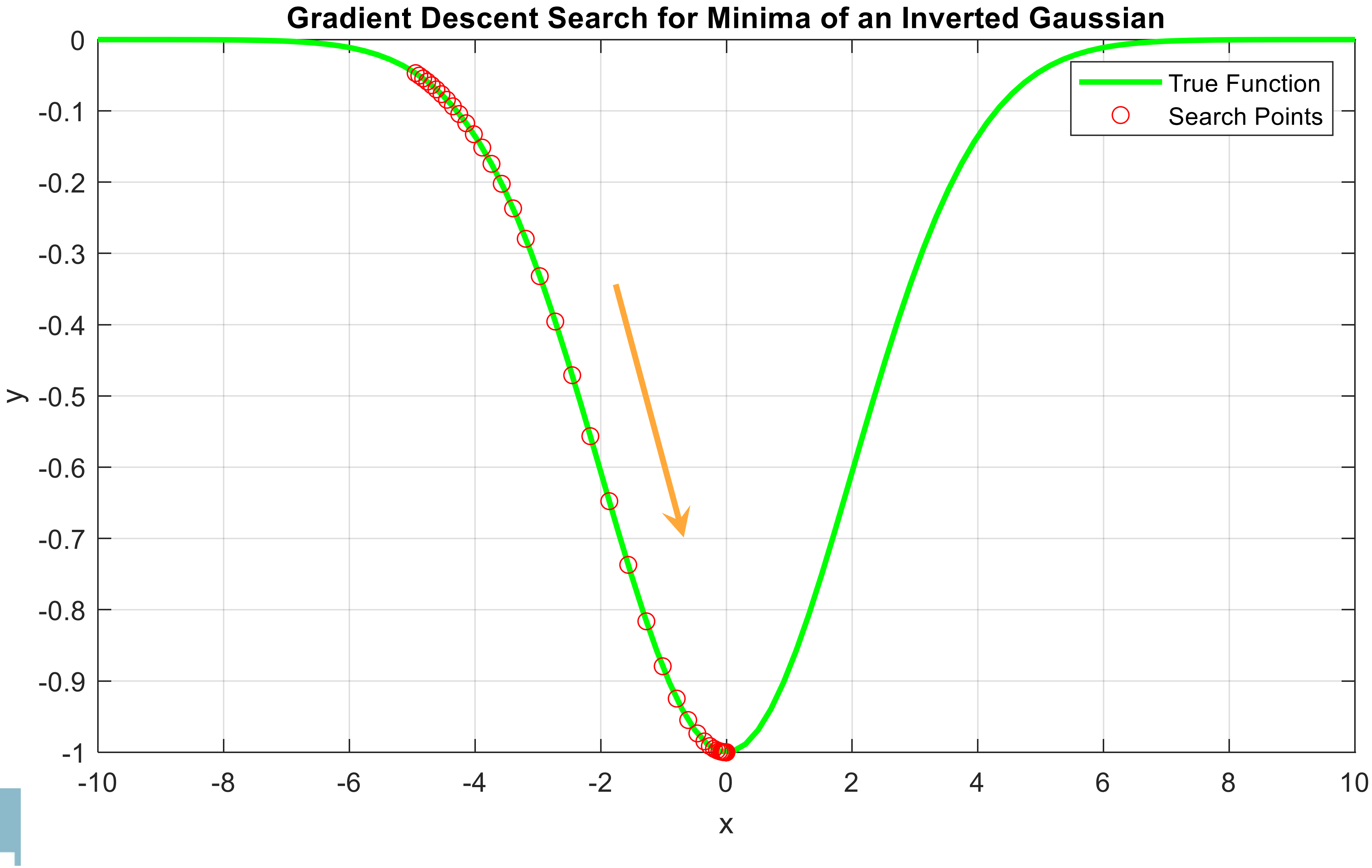
Saddle point

A point on the surface where the derivatives) of orthogonal function are zero, ie *stationary point*.

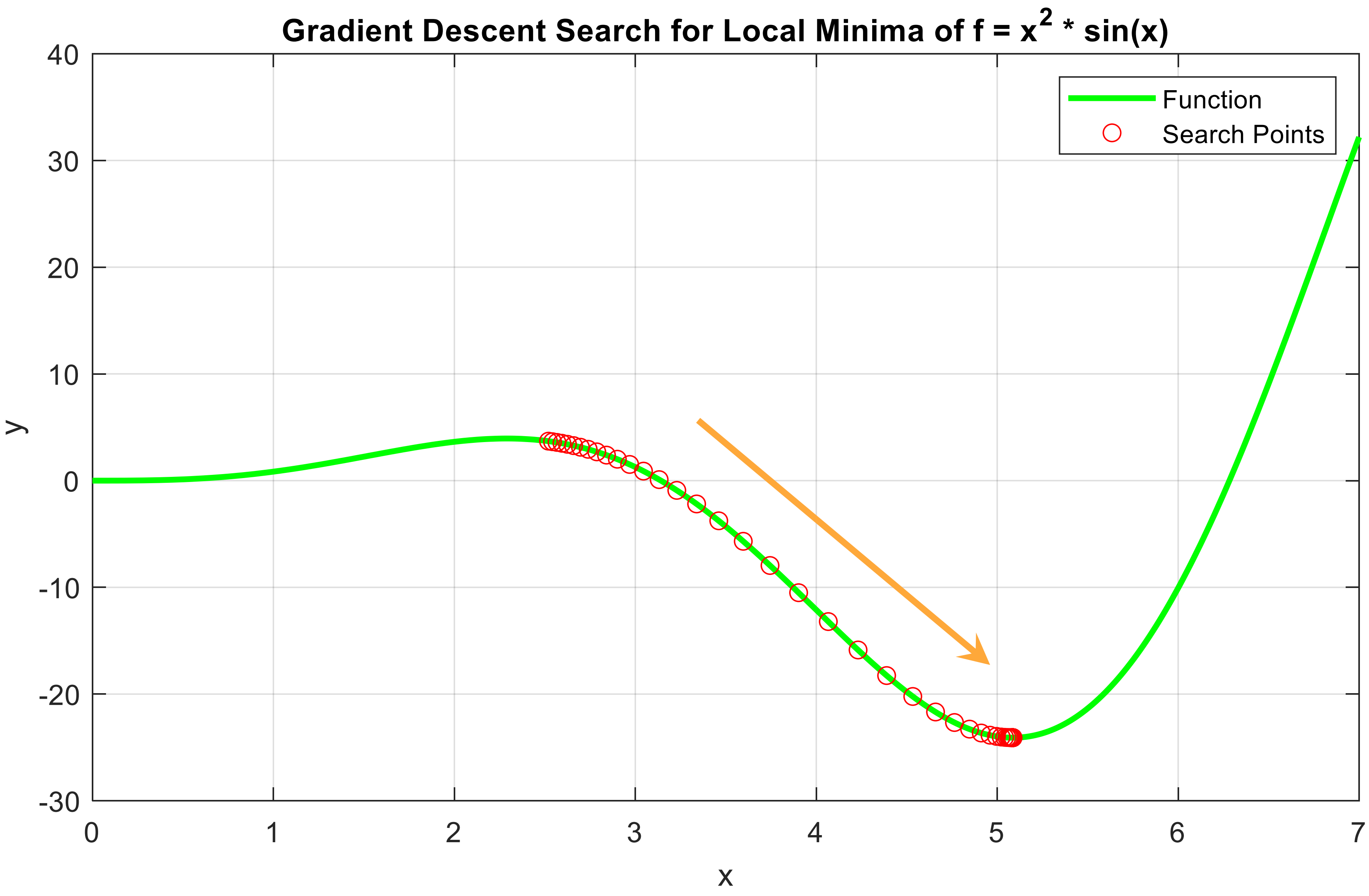


A saddle point (in red) on the function $z=x^2-y^2$
(Picture source: wikipedia)

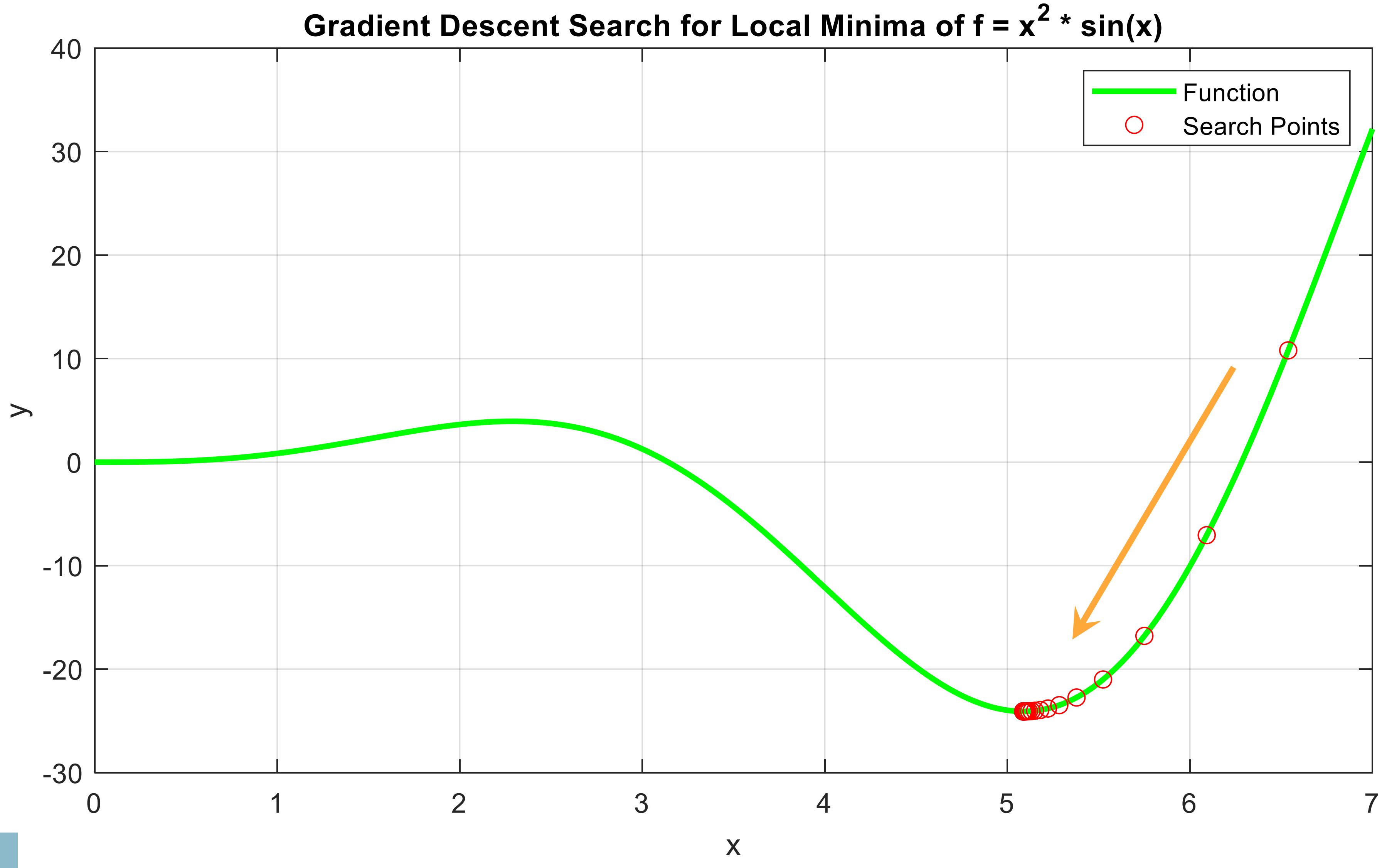
Example of gradient descent



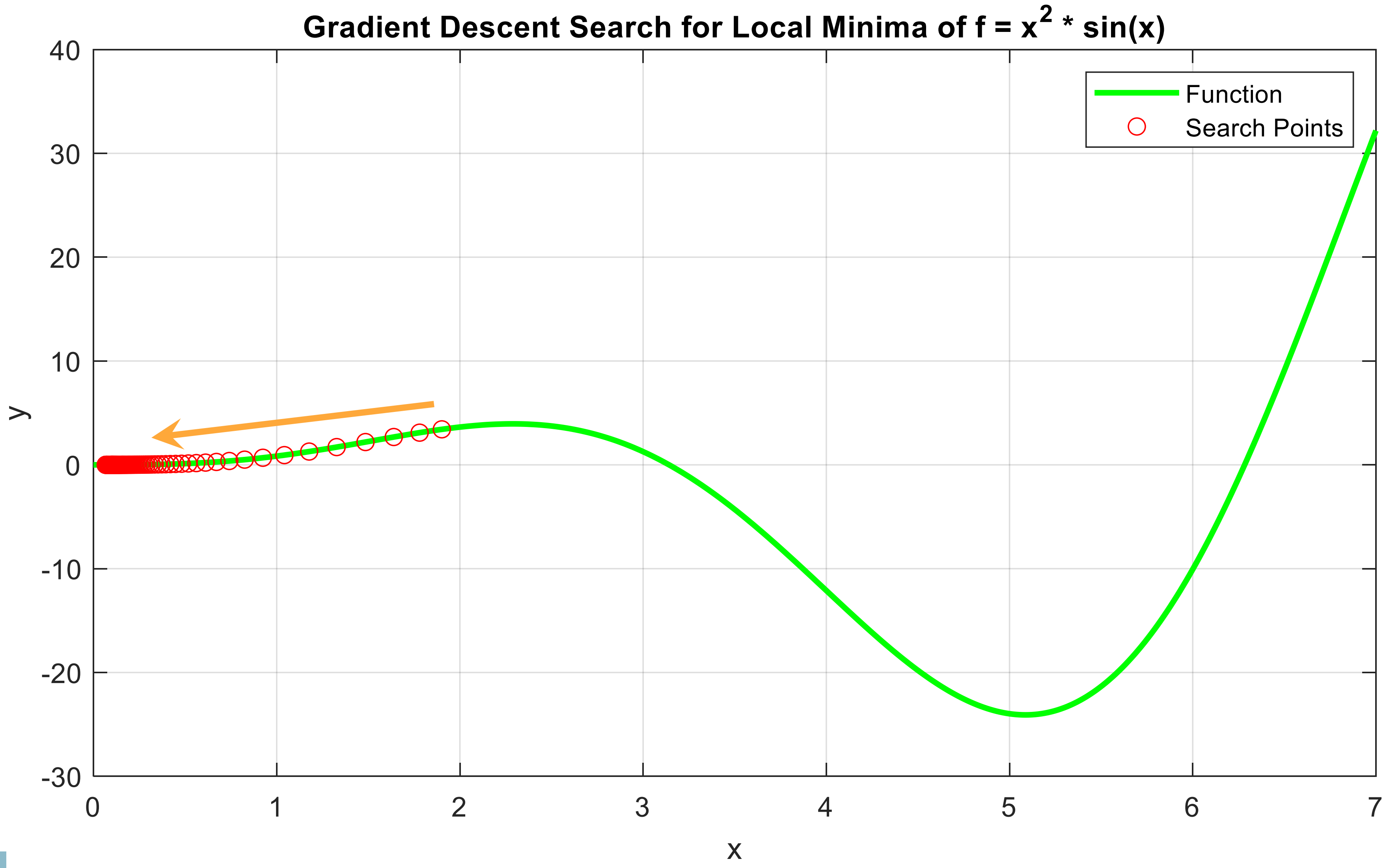
Example of gradient descent (Code demo)



Example of gradient descent

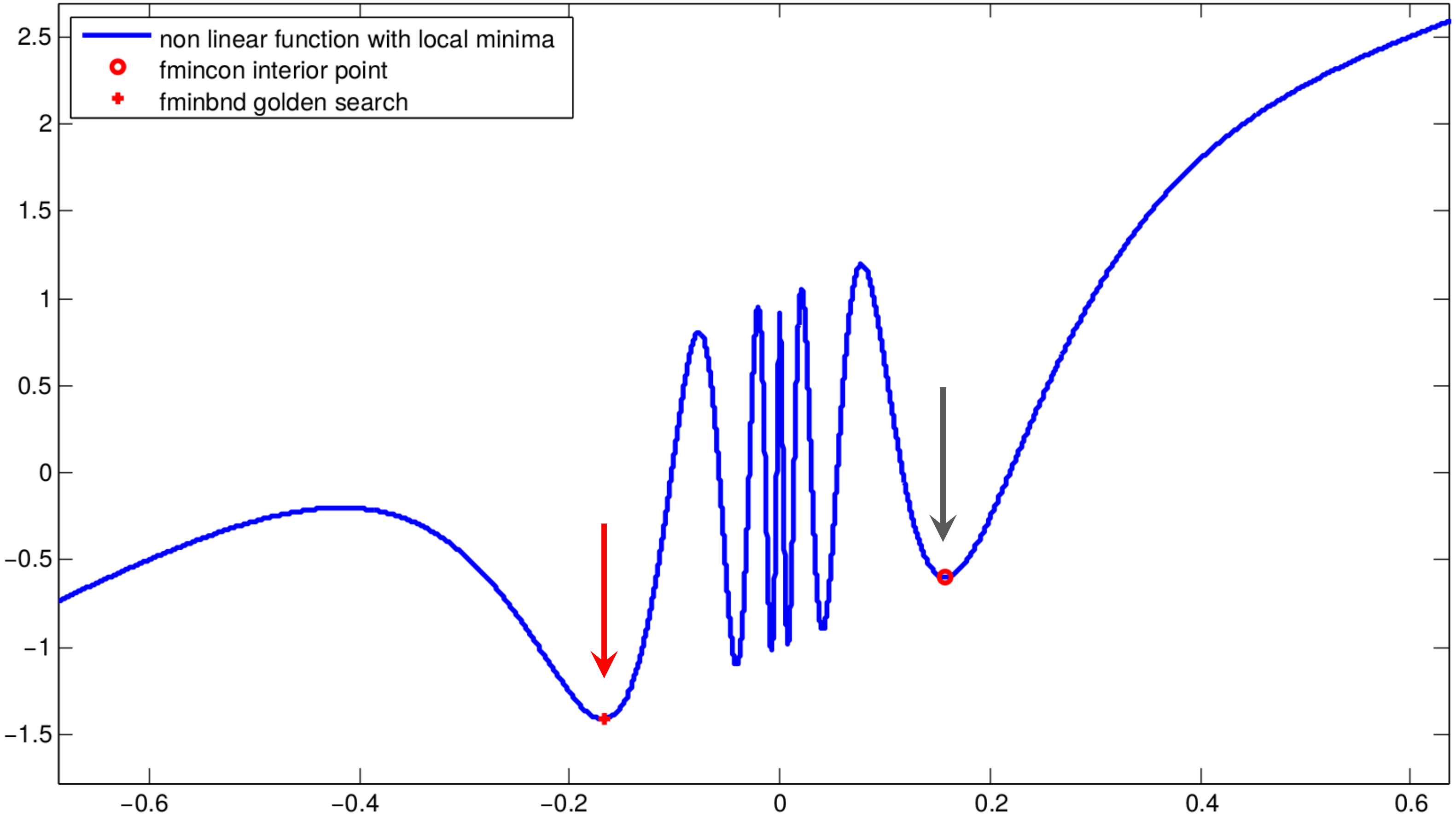


Example of gradient descent



Local minima problem

Search by gradient or continuous search can fail in complicated scenarios.



Constrained optimization

Constrained optimization

Practically, apart from maximizing or minimizing a function f(x) over all possible values of **x**, the range of feasible **x** are usually limited to some set **S**.

Thus, we have *constrained optimization* problems. Formally, if we define **S**,

$$\boldsymbol{S} = \{\boldsymbol{x} | \forall i, g_i(x) = 0 \text{ and } \forall j, h_j(x) \leq 0\}$$

where $g_i(\boldsymbol{x}) = c_i \ (i = 1, \ldots, n)$ are the **equality constraints** to be satisfied;

and $h_j(\boldsymbol{x}) \leq d_j \ (j = 1, \ldots, m)$ are the **inequality constraints** to be satisfied.

Constrained optimization

A general form of constrained optimization problem

$$\underset{\boldsymbol{x} \in \boldsymbol{S}}{\operatorname{argmin}} \; \boldsymbol{f}(\boldsymbol{x})$$

Or more specifically with the constraints as,

$$\begin{gathered} \operatorname{argmin} \; \boldsymbol{f}(\boldsymbol{x}) \\ \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{c} \\ \boldsymbol{h}(\boldsymbol{x}) \leq \boldsymbol{d} \end{gathered}$$

by using vectors (**c**, **d**) and matrices (**g**, **h**).

The objective function ***f*** is actually the sum of different cost functions.

Linear constrained optimization

- Linear least-squares solver with linear constraints.
- Example: Matlab ***lsqlin*** solves constrained linear least-squares problems

- Constrained linear least-squares of the form

$$\min_{x} \frac{1}{2} \| C \cdot x - d \|_2^2 \text{ such that } \begin{cases} A \cdot x \leq b, \\ Aeq \cdot x = beq, \\ lb \leq x \leq ub. \end{cases}$$

Nonlinear optimization

Nonlinear optimization

- Often, there are problems that are nonlinear, thus require nonlinear optimizers.
- Cost function ***J*** or ***f*** can be more complex, and are not necessarily the norm of vectors, ie, the cost can come from a function (e.g., forward kinematics).
- Matlab: optimization toolbox has solvers for linear, quadratic, integer, and nonlinear optimization problems. For example, fmincon is a very powerful tool, if the problem is well formulated, usually fmincon can find the solution.
- C++: NLopt

Matlab functions for unconstrained optimization

Functions

1. fminsearch: Find minimum of unconstrained multivariable function using derivative-free method
2. fminunc: Find minimum of unconstrained multivariable function

Note: these are actually nonlinear programming solvers.

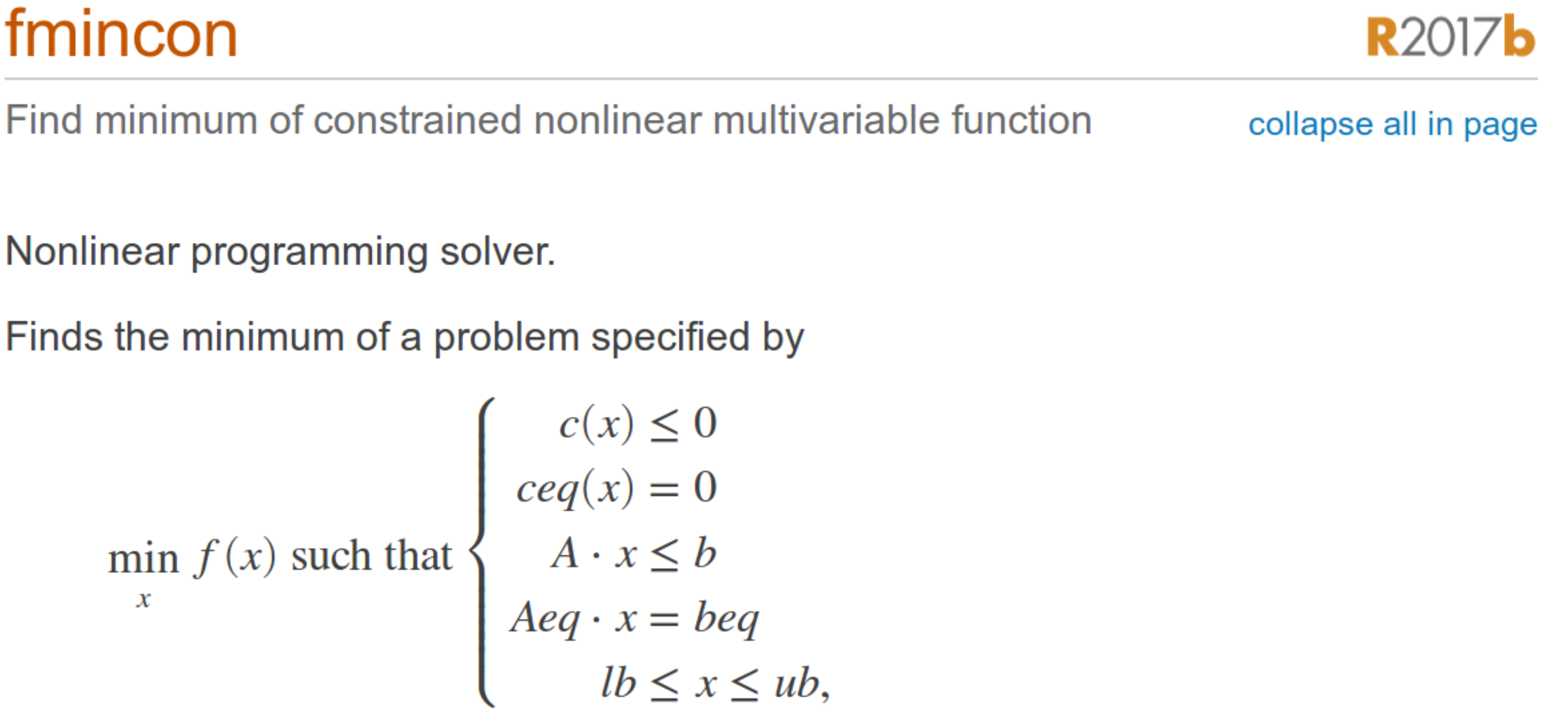
Matlab functions for constrained optimization

Functions

1. fminbnd: find minimum of single-variable function on fixed interval
2. fmincon: find minimum of constrained nonlinear multivariable function
3. fseminf: find minimum of semi-infinitely constrained multivariable nonlinear function

Formulating constraints: fmincon case

However, fmincon has different interface or formalization of the constraints, as shown below:



fmincon R2017b

Find minimum of constrained nonlinear multivariable function

Nonlinear programming solver.

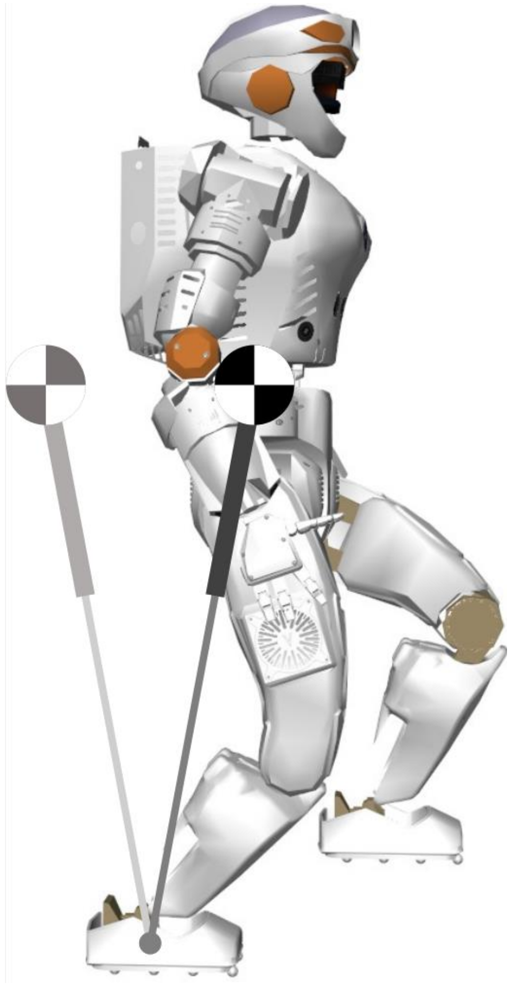
Finds the minimum of a problem specified by

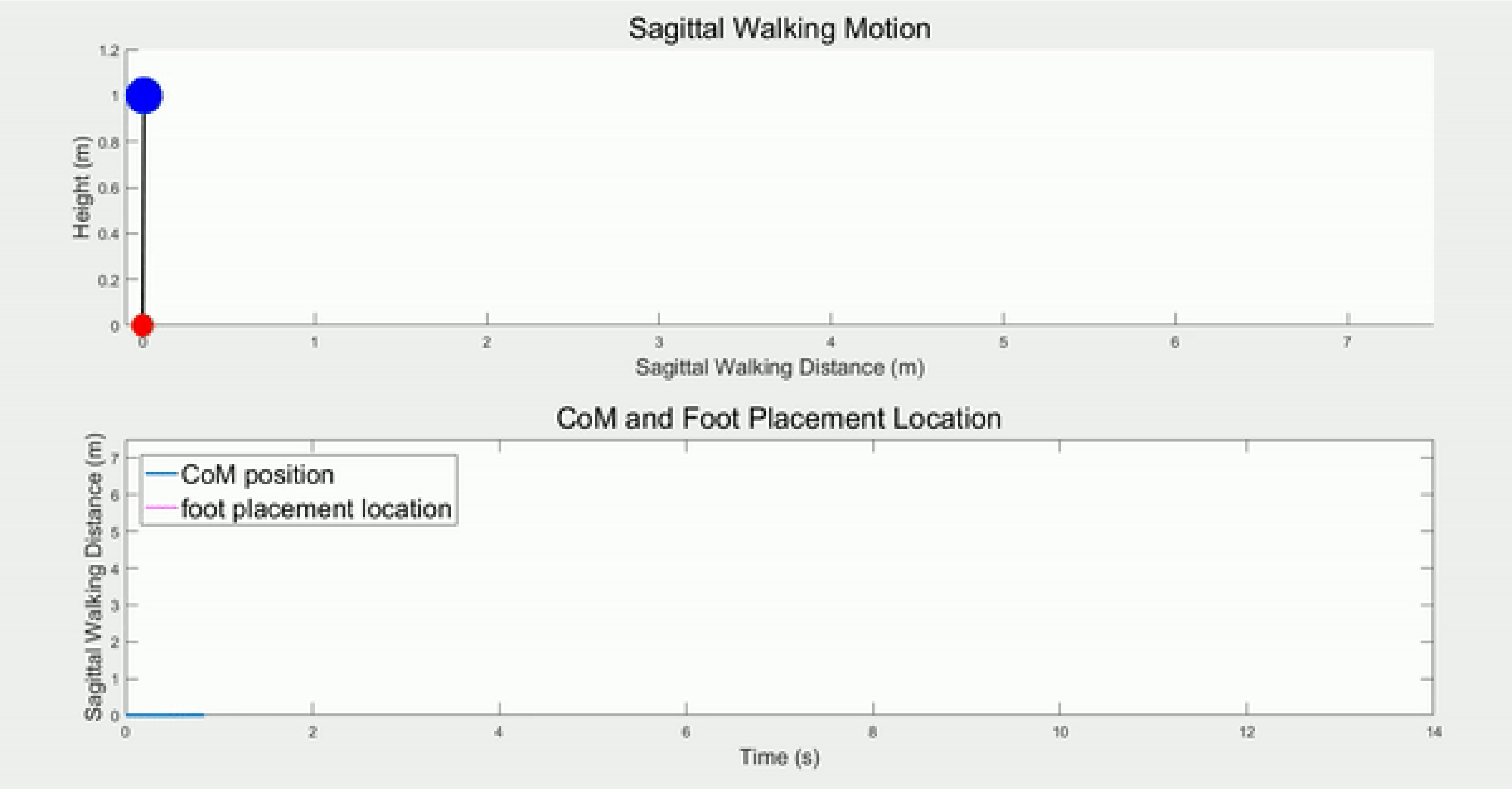
$$\min_{x} f(x) \text{ such that } \begin{cases} c(x) \leq 0 \\ ceq(x) = 0 \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub, \end{cases}$$

Quiz: how to formulate inequality constraints $\boldsymbol{lbA} \leq \boldsymbol{A} \cdot \boldsymbol{x} \leq \boldsymbol{ubA}$ in the form of

$$\boldsymbol{A} \cdot \boldsymbol{x} \leq \boldsymbol{b}$$

Application of nonlinear optimization (fmincon)





Task space control using optimization

Question: how to formulate inverse kinematics problem considering the constraints, eg joint angle limit?

Formulation of cost function?

How to design it?

Design of cost function *f*(*x*)

<u>Problem Formulation</u>: how to build up the cost function.

$$\boldsymbol{f}(\boldsymbol{q}) = ||\boldsymbol{F}(\boldsymbol{q}) - \boldsymbol{y}^d||^2_{\boldsymbol{QP}} + ||\boldsymbol{O}(\boldsymbol{q}) - \boldsymbol{vec}^d||^2_{\boldsymbol{Qr}} + ||\boldsymbol{q} - \boldsymbol{q}_0||^2_{\boldsymbol{Qj}}$$

$$\boldsymbol{q}_{min} \leq \boldsymbol{q} \leq \boldsymbol{q}_{max}$$

q: joint configuration → decision variables

q$_0$, **q**$_{\mathbf{min}}$, **q**$_{\mathbf{max}}$: initial, minimum, maximum joint configurations

F(q), O(q): forward kinematics, returns end effector position and orientation

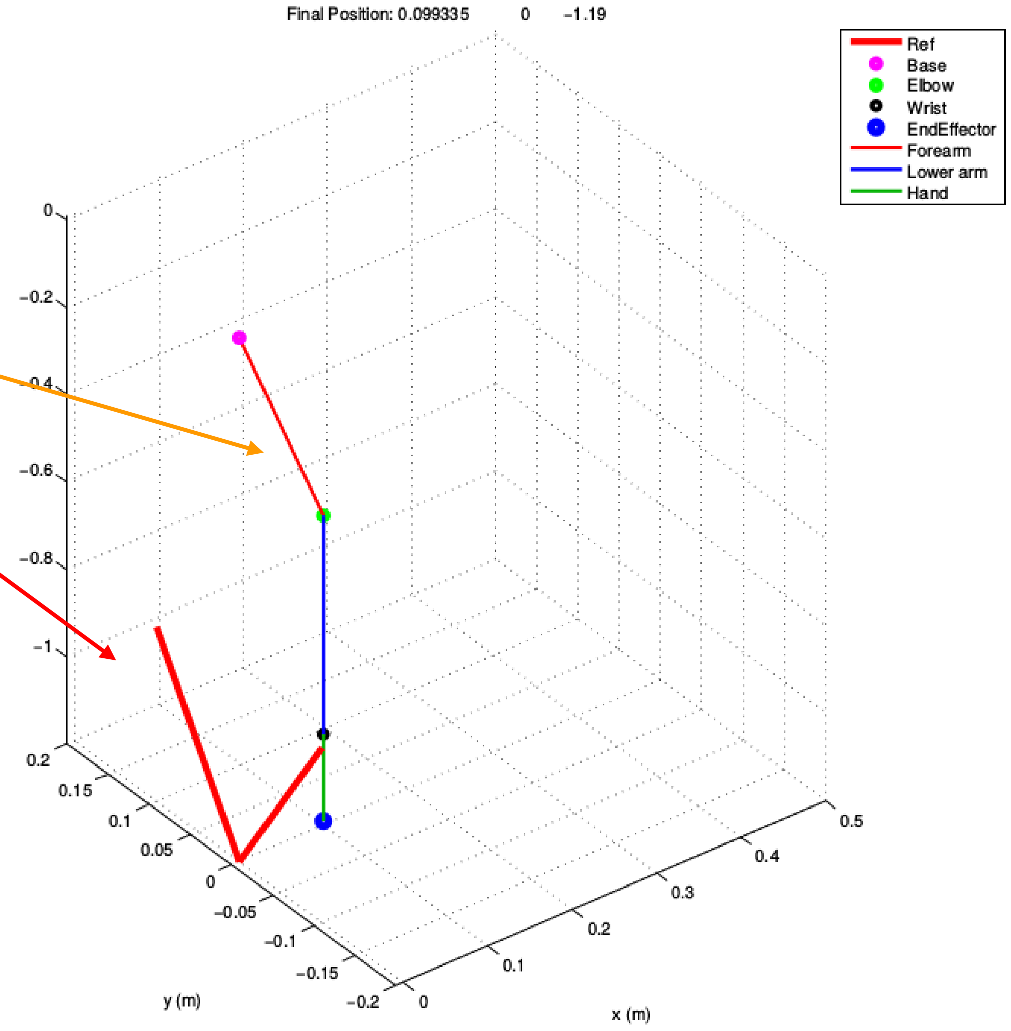
y$^{\mathbf{d}}$, **vec**$^{\mathbf{d}}$: desired task space or workspace position, orientation

Q$_{\mathbf{p}}$, **Q**$_{\mathbf{r}}$, **Q**$_{\mathbf{j}}$ are the weighting matrices for workspace position, rotation, and joint/configuration space respectively.

Application of nonlinear optimization (fmincon)

Initial configuration of robot arm at $\mathbf{q}_0$

V shape is the workspace trajectory



Application of nonlinear optimization (fmincon)

Optimization only over end-effector position and minimum change of joint angles.

$$\boldsymbol{f}(\boldsymbol{q}) = ||\boldsymbol{F}(\boldsymbol{q}) - \boldsymbol{y}^d||^2_{\boldsymbol{QP}} + ||\boldsymbol{q} - \boldsymbol{q}_0||^2_{\boldsymbol{Qj}}$$

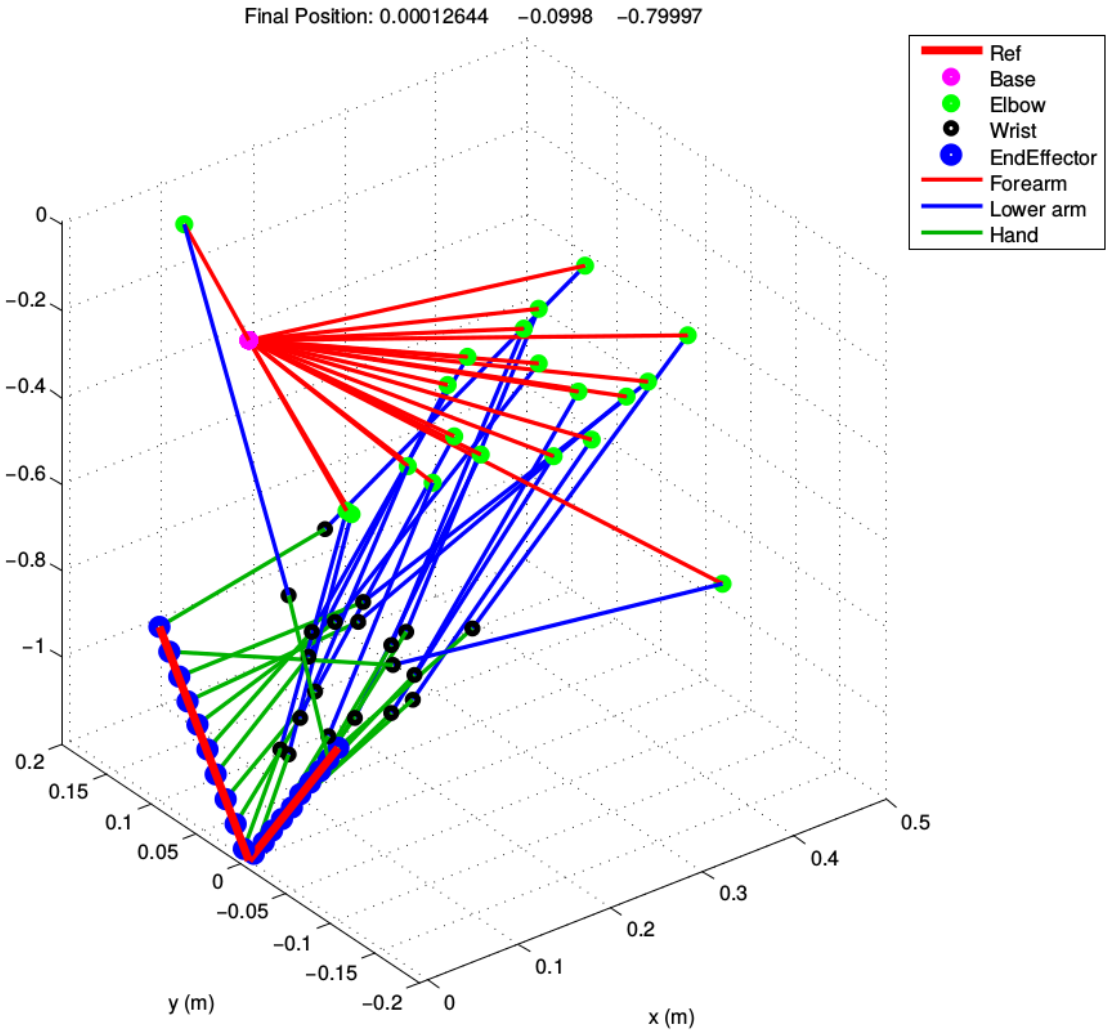
$\mathbf{Q_p}$ = 1;

$\mathbf{Q_r}$ = 0; (orientation is NOT controlled)

$\mathbf{Q_j}$ = 1e-6;

$\mathbf{q_{min}}$: -pi/2

$\mathbf{q_{max}}$: pi/2



Application of nonlinear optimization (fmincon)

Optimization only over end-effector position and minimum change of joint angles.

$$\boldsymbol{f}(\boldsymbol{q}) = ||\boldsymbol{F}(\boldsymbol{q}) - \boldsymbol{y}^d||^2_{\boldsymbol{QP}} + ||\boldsymbol{q} - \boldsymbol{q}_0||^2_{\boldsymbol{Qj}}$$

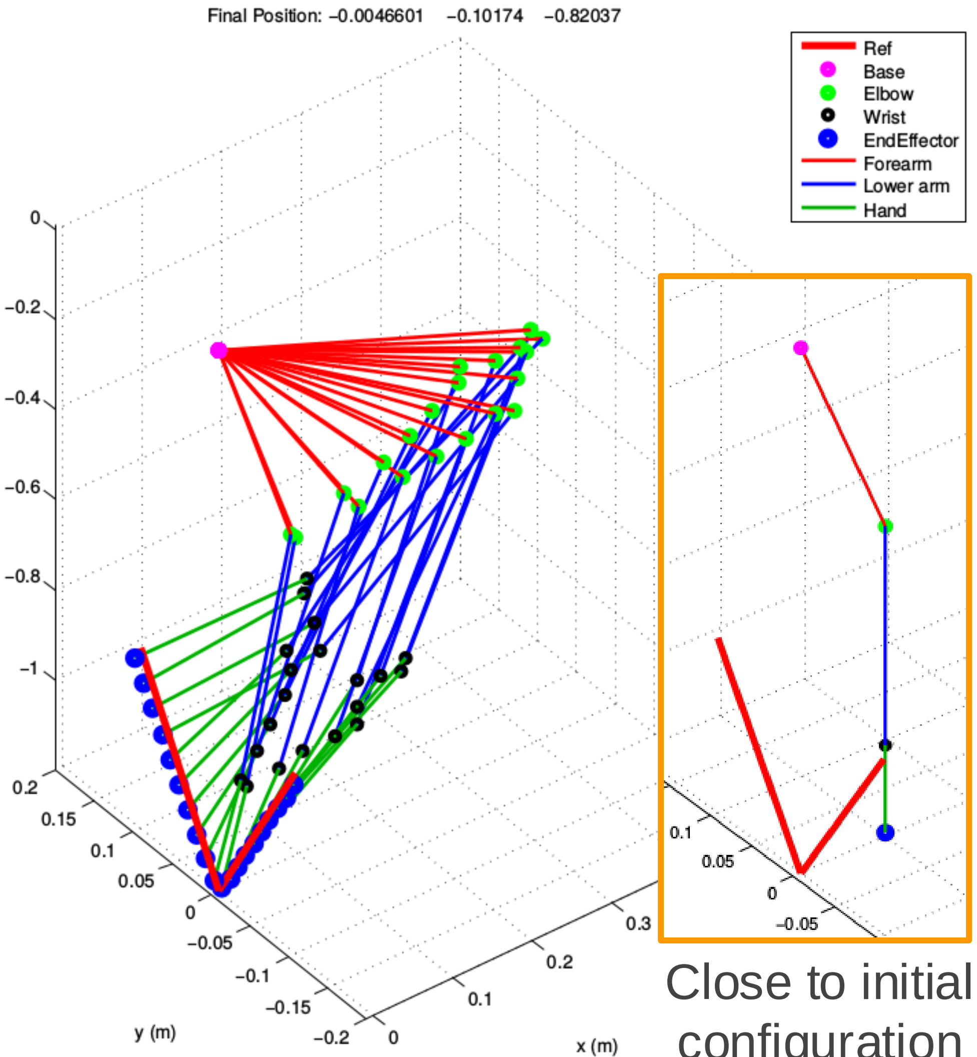
$\mathbf{Q_p}$ = 1;

$\mathbf{Q_r}$ = 0; (orientation is NOT controlled)

$\mathbf{Q_j}$ = 5e-3; ⇐ **Increase weight** for $\mathbf{q}_0$

$\mathbf{q_{min}}$: -pi/2

$\mathbf{q_{max}}$: pi/2



Close to initial configuration

Application of nonlinear optimization (fmincon)

Optimization of all terms:

$$\boldsymbol{f}(\boldsymbol{q}) = ||\boldsymbol{F}(\boldsymbol{q}) - \boldsymbol{y}^d||^2_{\boldsymbol{QP}} + ||\boldsymbol{O}(\boldsymbol{q}) - \boldsymbol{vec}^d||^2_{\boldsymbol{Qr}} + ||\boldsymbol{q} - \boldsymbol{q}_0||^2_{\boldsymbol{Qj}}$$

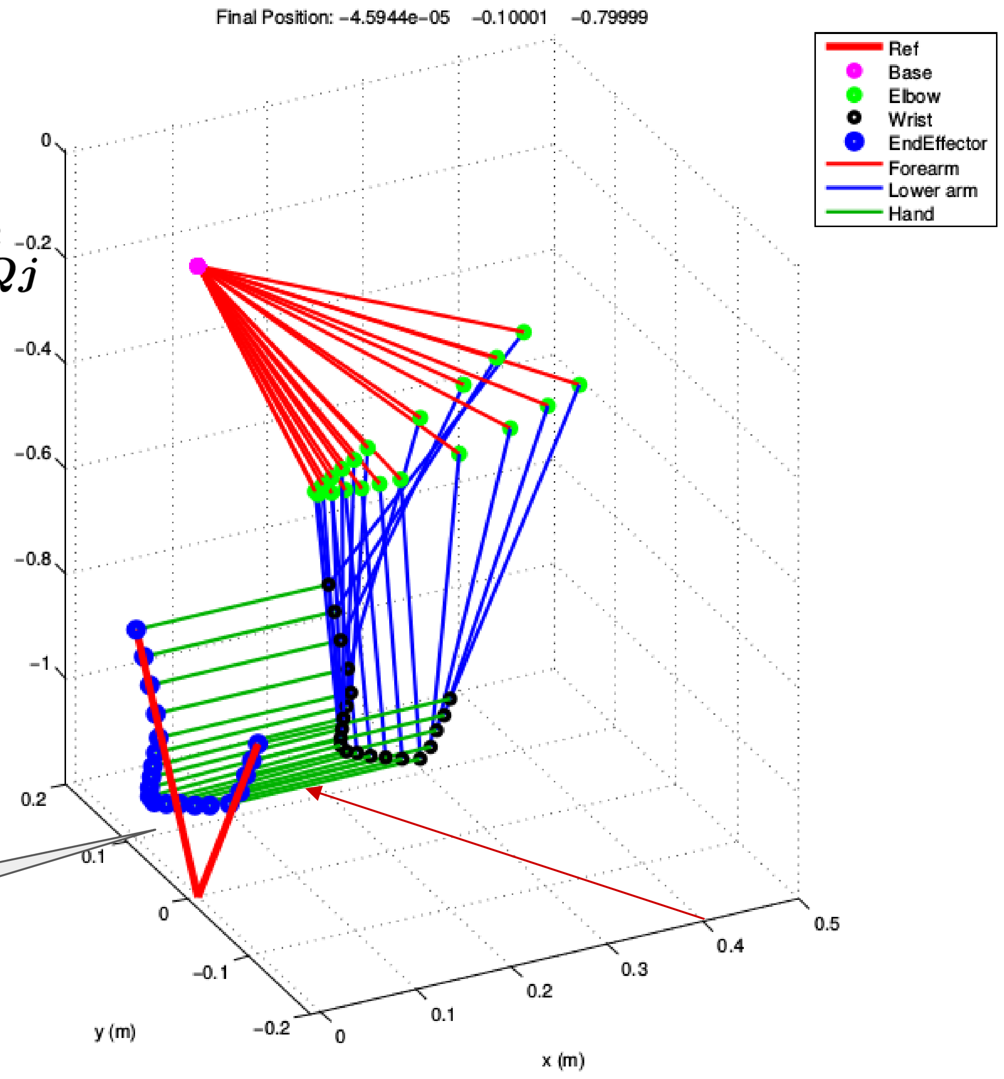
Q_p = 1;

Q_r = 1; (note: orientation is controlled)

Q_j = 1e-6;

q_{min} : -pi/2

q_{max} : pi/2



What is the problem here?

Application of nonlinear optimization (fmincon)

Optimization of all terms:

$$\boldsymbol{f}(\boldsymbol{q}) = ||\boldsymbol{F}(\boldsymbol{q}) - \boldsymbol{y}^d||^2_{\boldsymbol{QP}} + ||\boldsymbol{O}(\boldsymbol{q}) - \boldsymbol{vec}^d||^2_{\boldsymbol{Qr}} + ||\boldsymbol{q} - \boldsymbol{q}_0||^2_{\boldsymbol{Qj}}$$

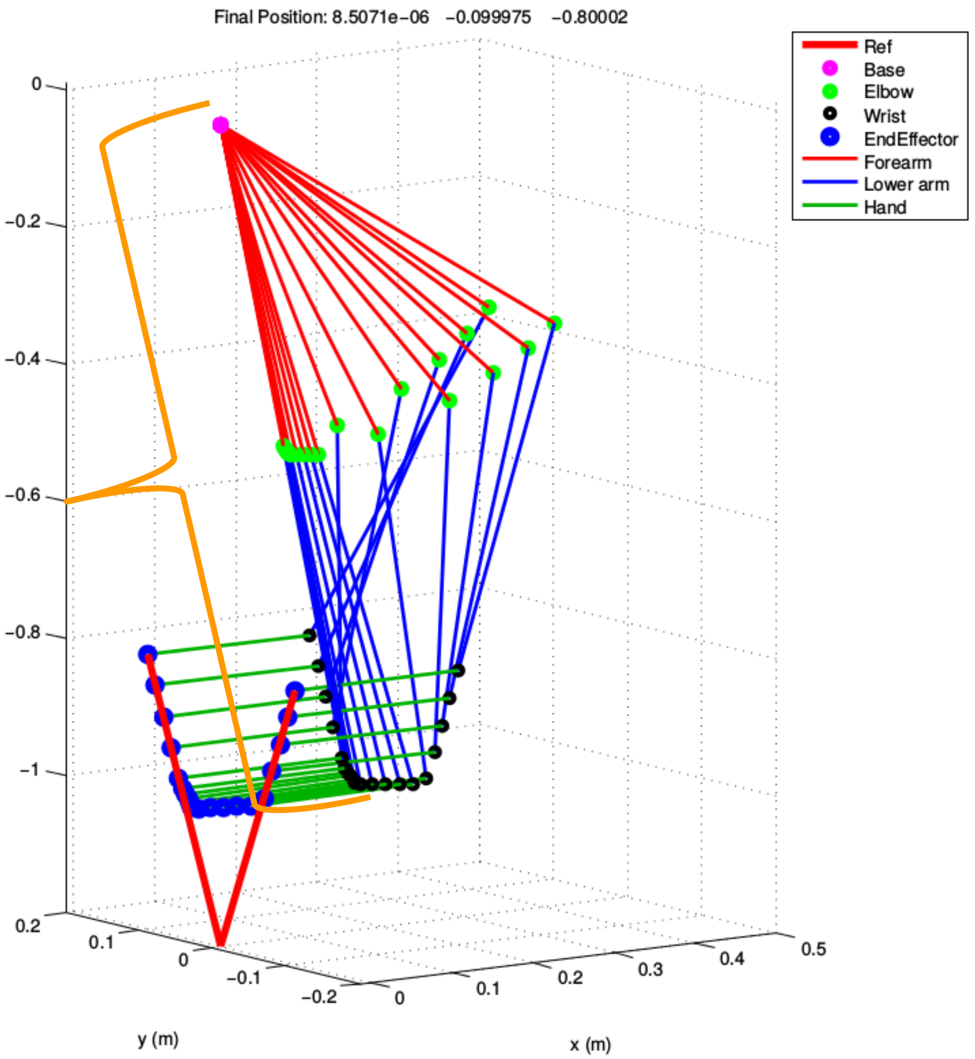
Q_p = 1;

Q_r = 1; (note: orientation is controlled)

Q_j = 1e-6;

q_{min} : [-150; -90; -90; 0; -90; -150]/180*pi

q_{max} : [150; 90; 90; 150; 90; 150]/180*pi



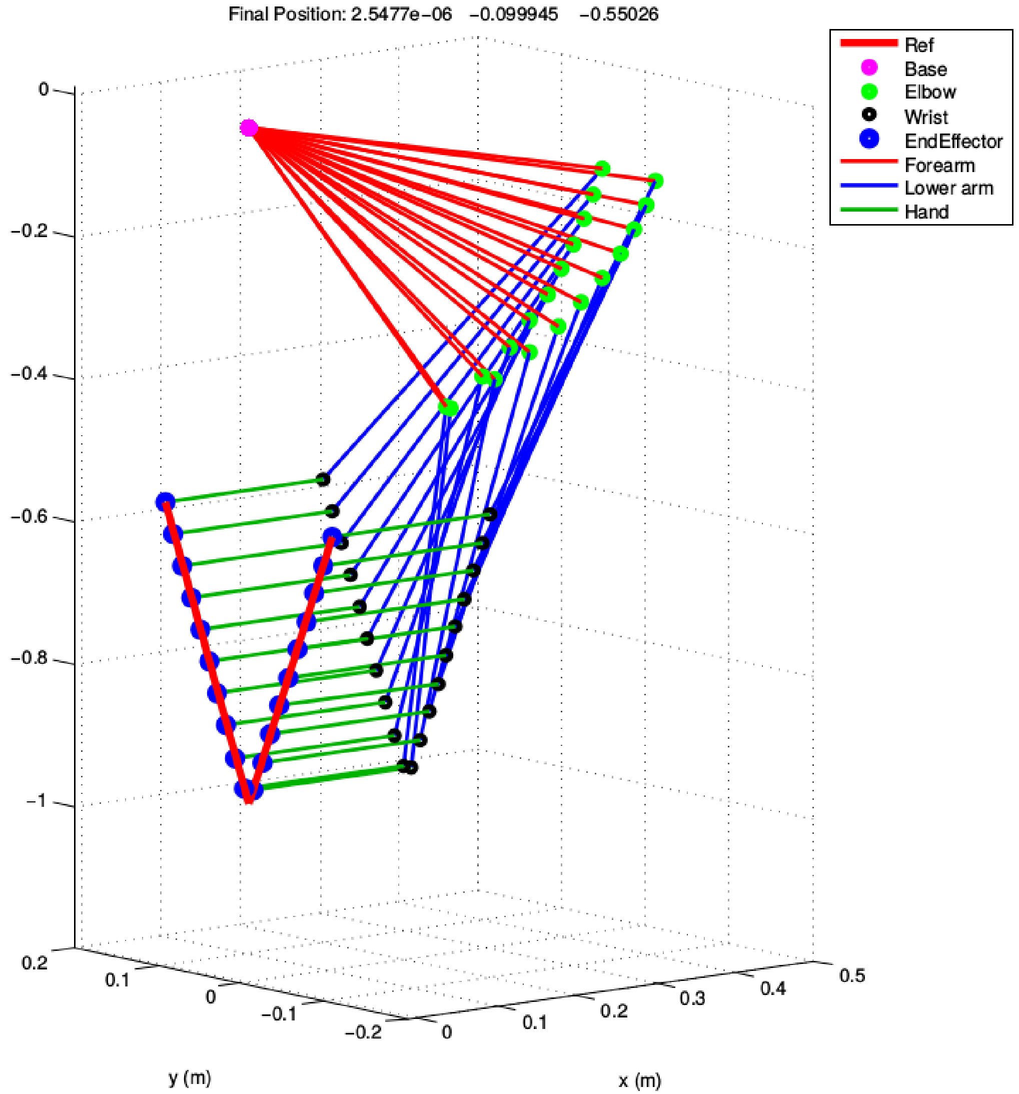
Singularity

Application of nonlinear optimization (fmincon)

Same setting for optimization, only change the task space **V** trajectory to be ***within*** workspace.

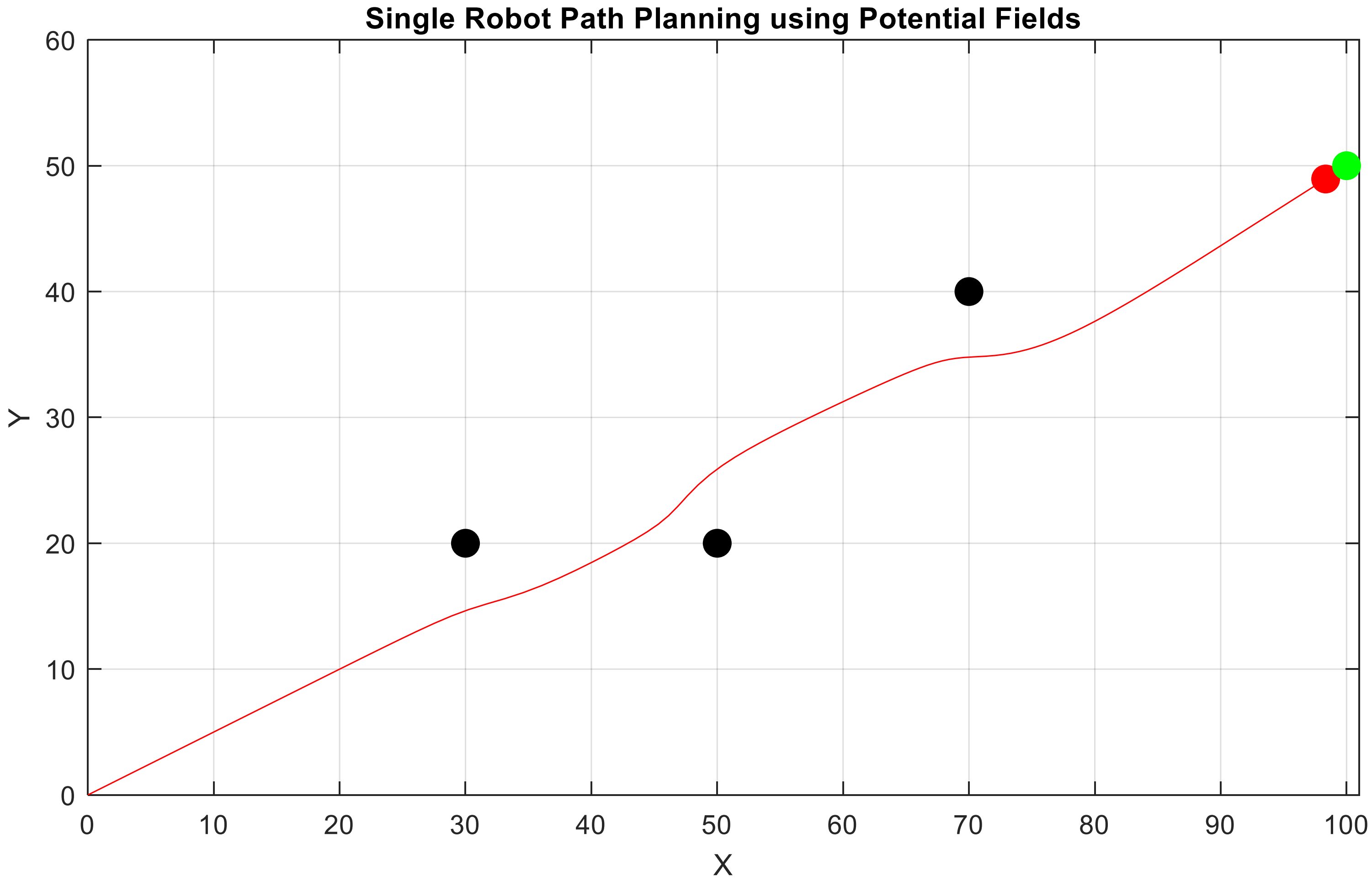
The optimization scheme works nicely, considering:

- the minimum deviation from its home position
- minimum error of pose (position & orientation)
- the joint limits!



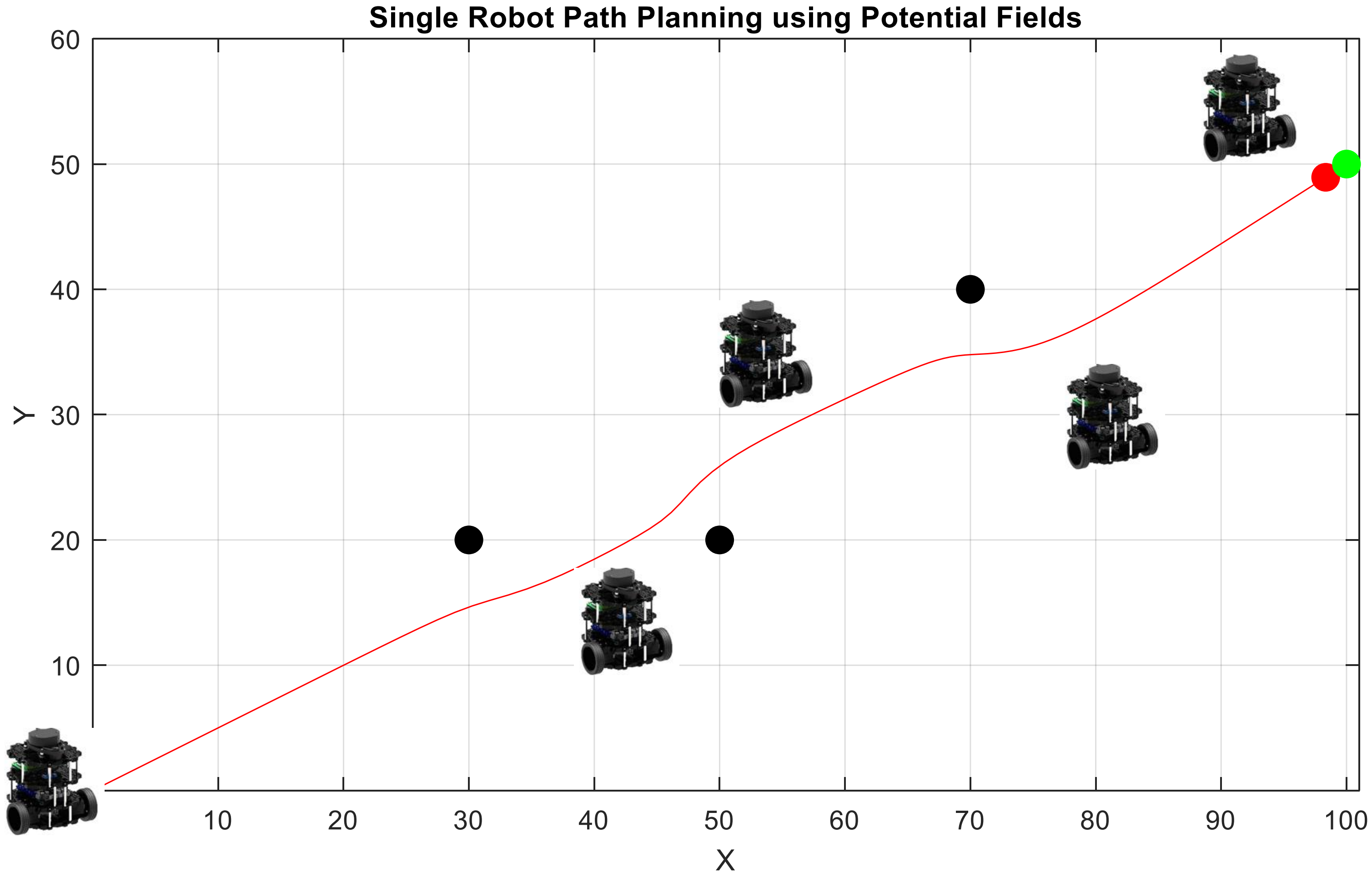
Optimization for motion planning

- Can we replace artificial potential field method? Without computing forward physics?



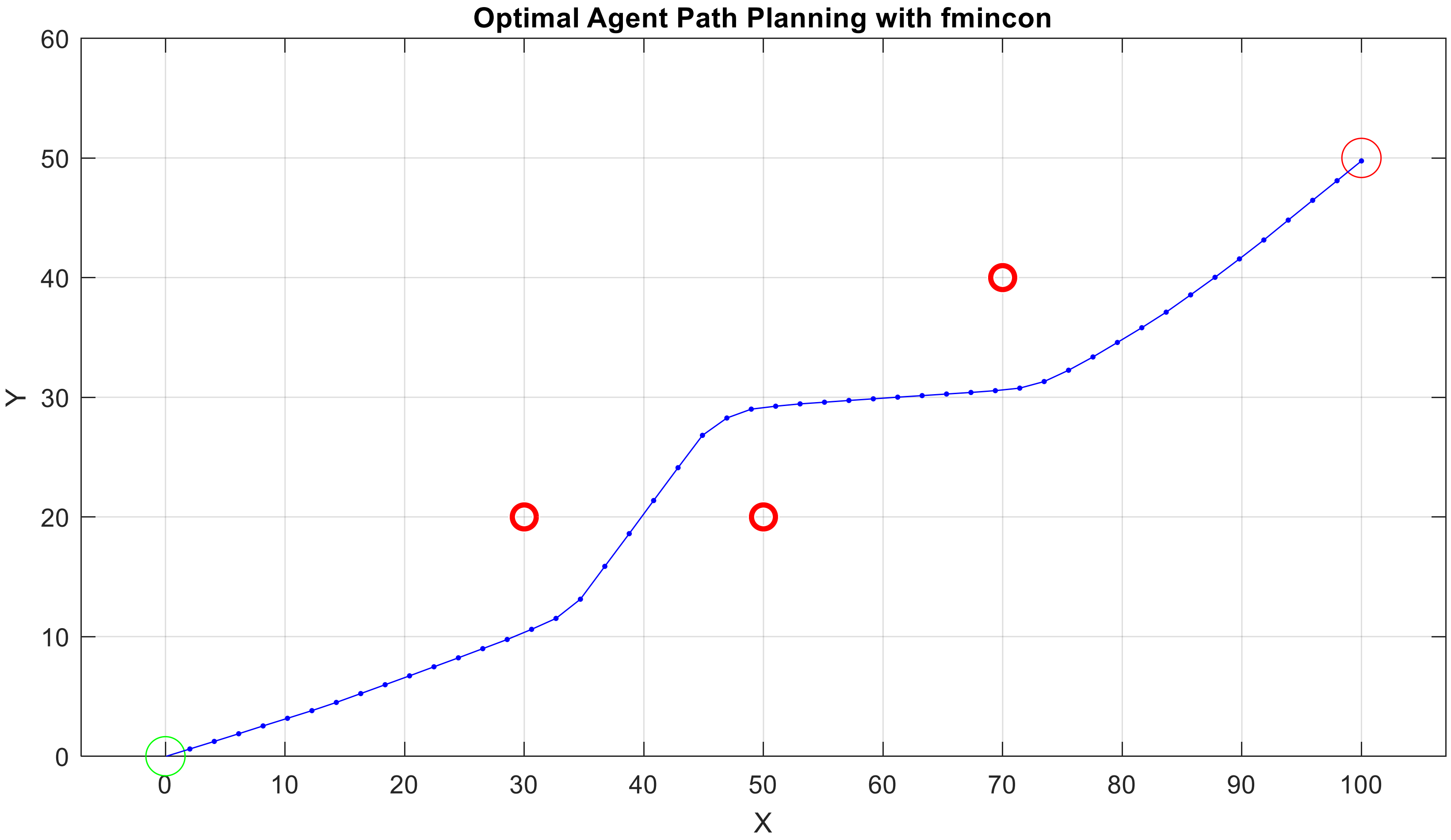
Optimization for motion planning

- What are the **decision variables**?
- What is the **cost function**?



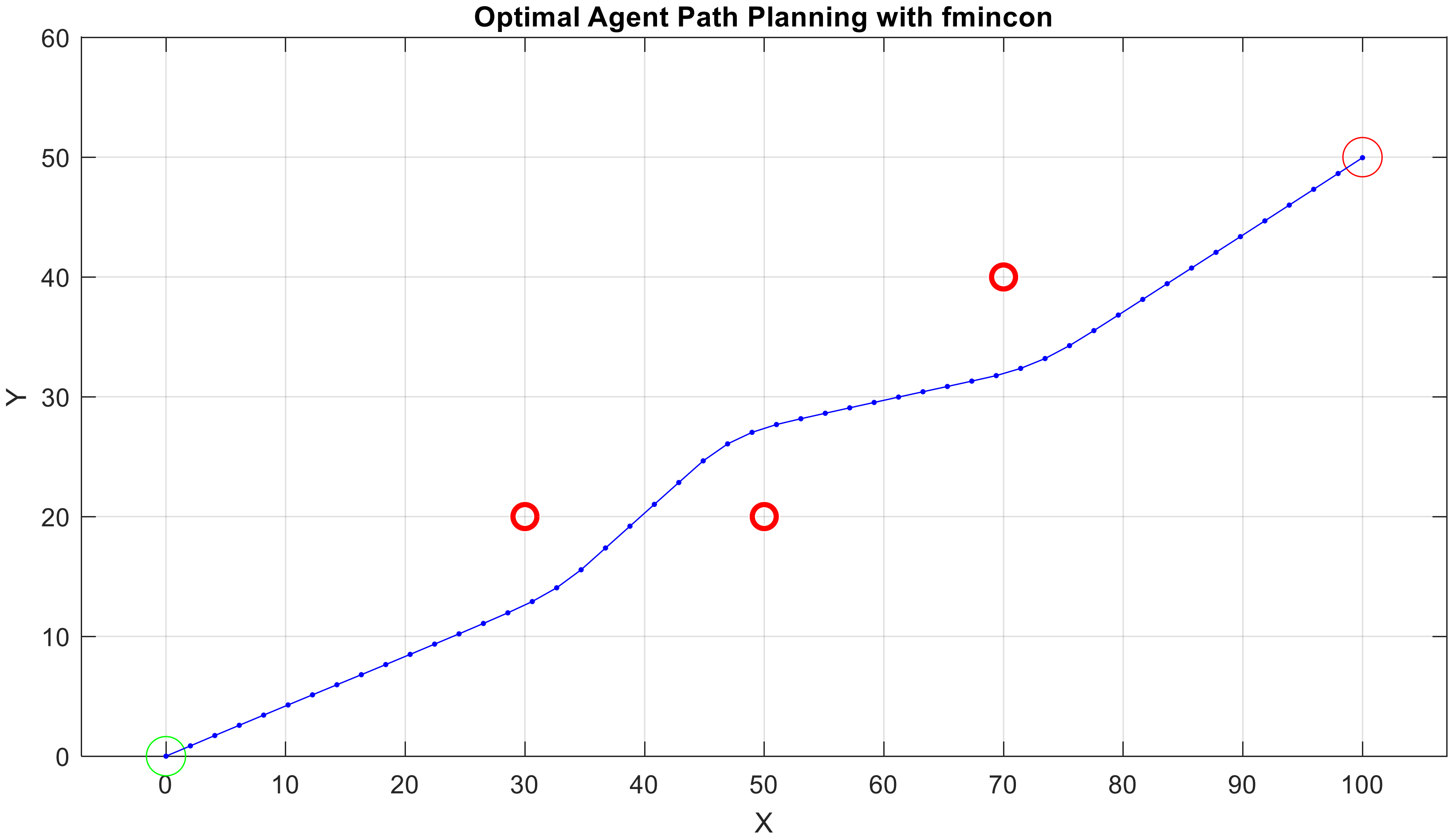
Motion planning using optimization (Code demo)

- We use exactly the same repulsive field as the penalty of the cost function
- Solution 1

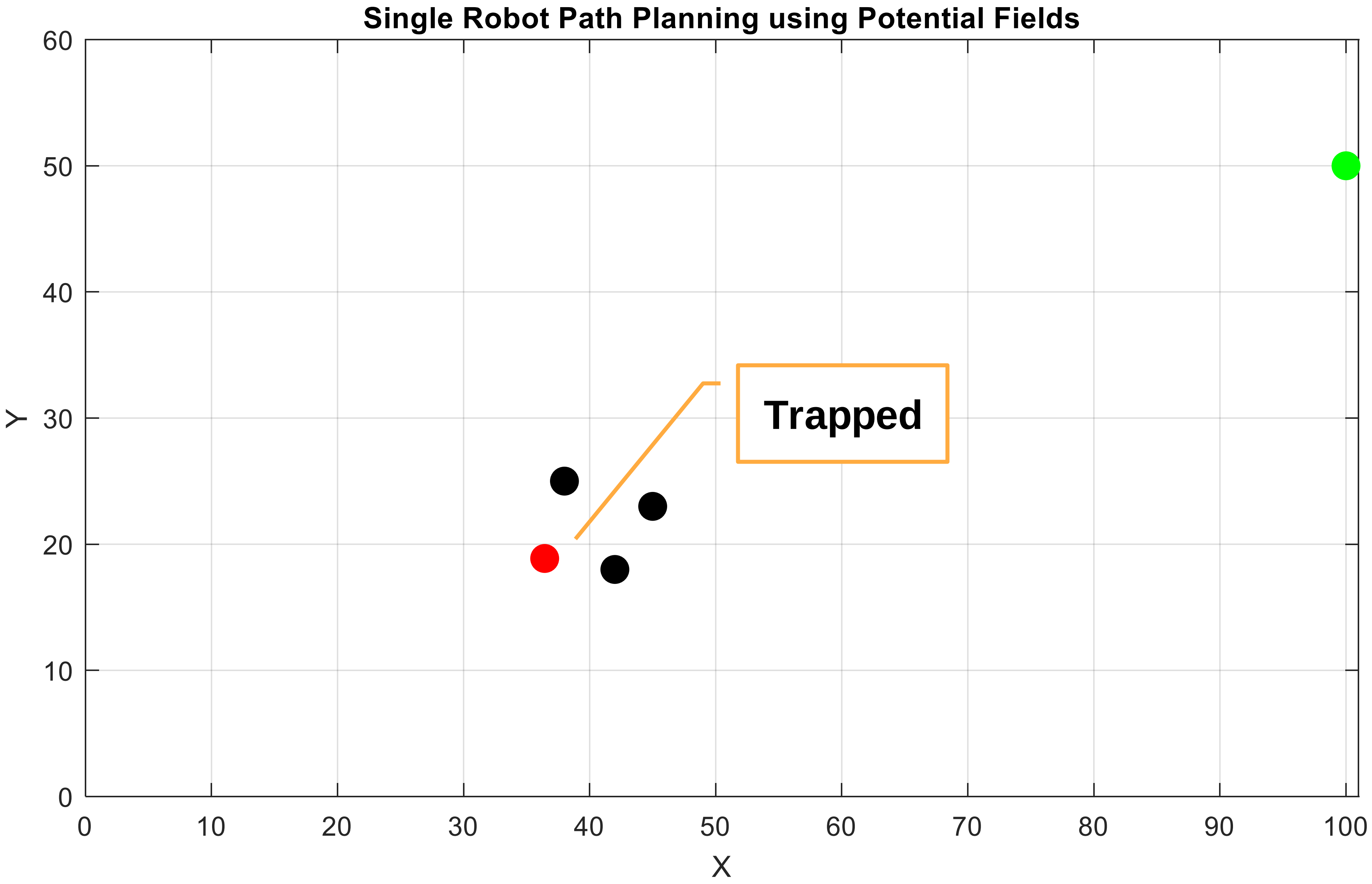


Motion planning using optimization (Code demo)

- We use exactly the same repulsive field as the penalty of the cost function
- Solution 2

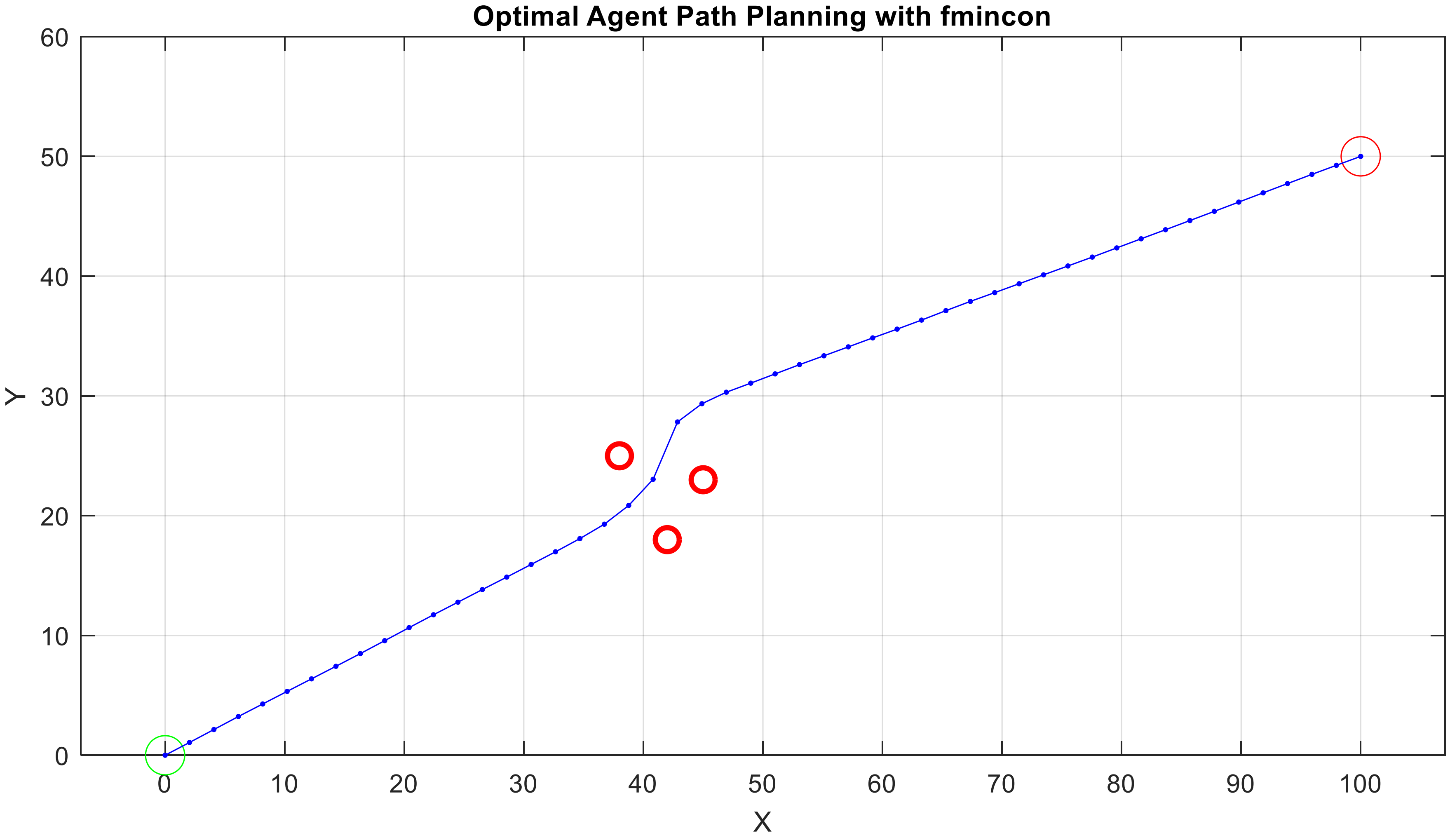


Solving the hard case – get around the trap



Solving the hard case – get around the trap

- What to tune or change in the optimization?


Optimal Agent Path Planning with fmincon
Y
X