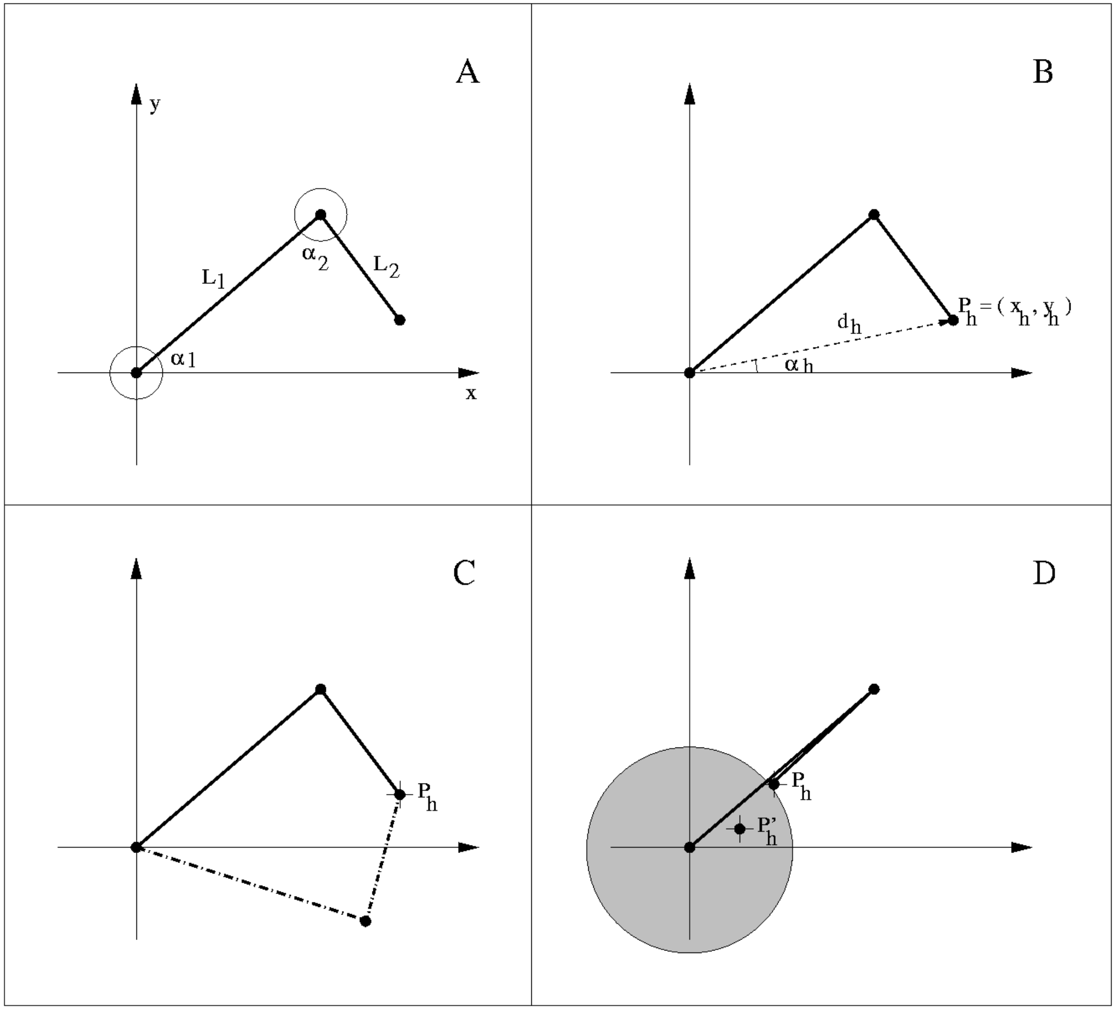
Inverse Kinematics

Inverse Kinematics (IK)

- usually more difficult than forward kinematics
- often under-/overdetermined problem
- hence, sometimes no or multiple solutions

e.g., given P_h find α_1, α_2



Two Approaches for Inverse Kinematics

- analytical
  - use of geometrical / algebraic relations
  - closed form
  - for specific systems
  - typically with a few DoFs
- iterative (numerical)
  - more general
  - suited for complex kinematic chains

From IK to Trajectories

note:

- IK, especially analytical, provides “final” solution
- i.e., DoF values that correspond to desired pose
- intermediate values needed to form a ***trajectory***
- i.e., a sequence of intermediate poses over time

From IK to Trajectories

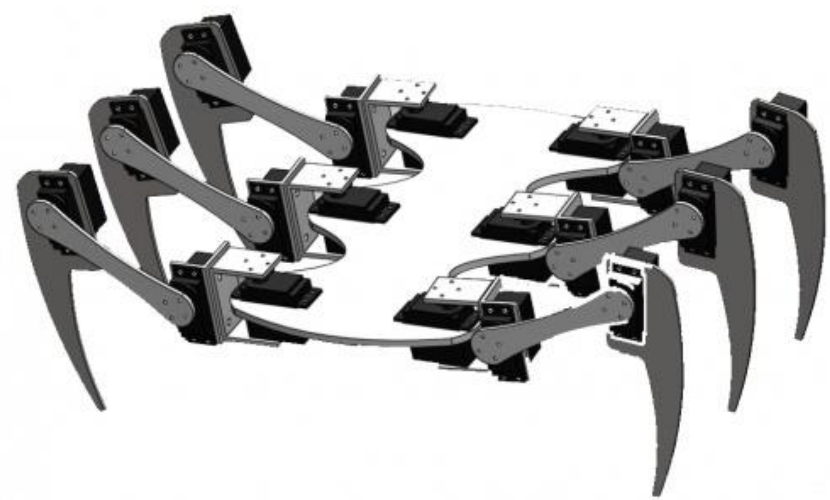
intermediate values needed to form a trajectory

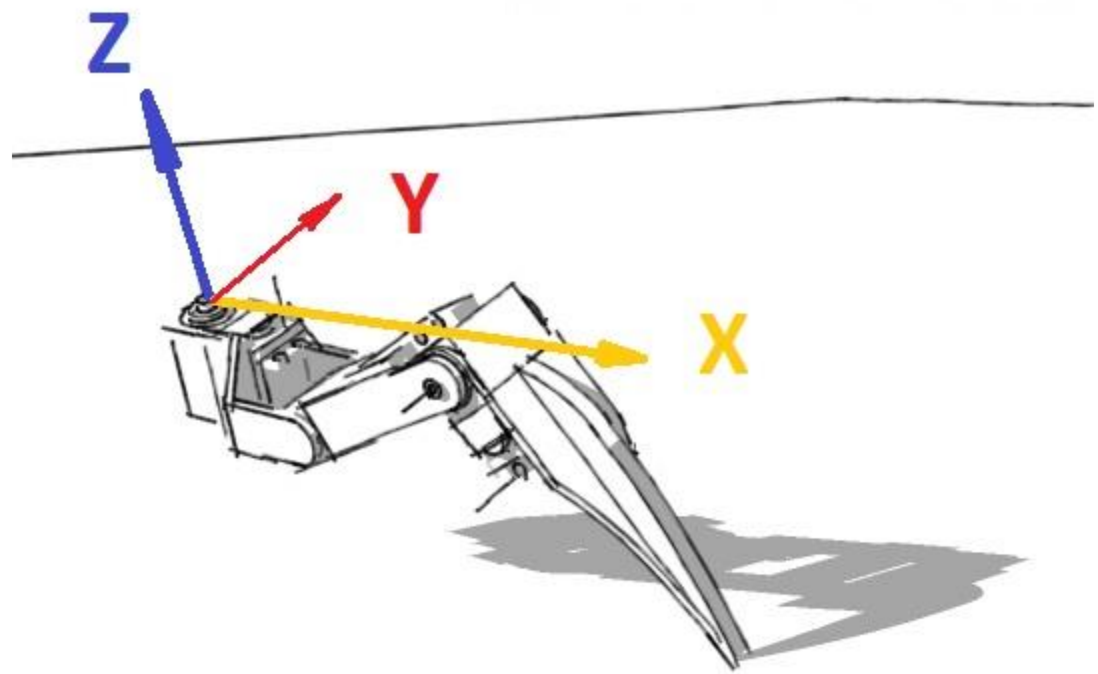
- simple approach with especially analytical IK
  - interpolation
  - especially: quaternions for orientation / rotation (SLERP)
- numerical IK
  - e.g., take intermediate values from the iterations
  - (plus interpolation)
- challenges: a.o., collision avoidance
  - more about this in the context of path-planning
  - in the AI lecture

Example: Analytical IK

leg of a hexapod

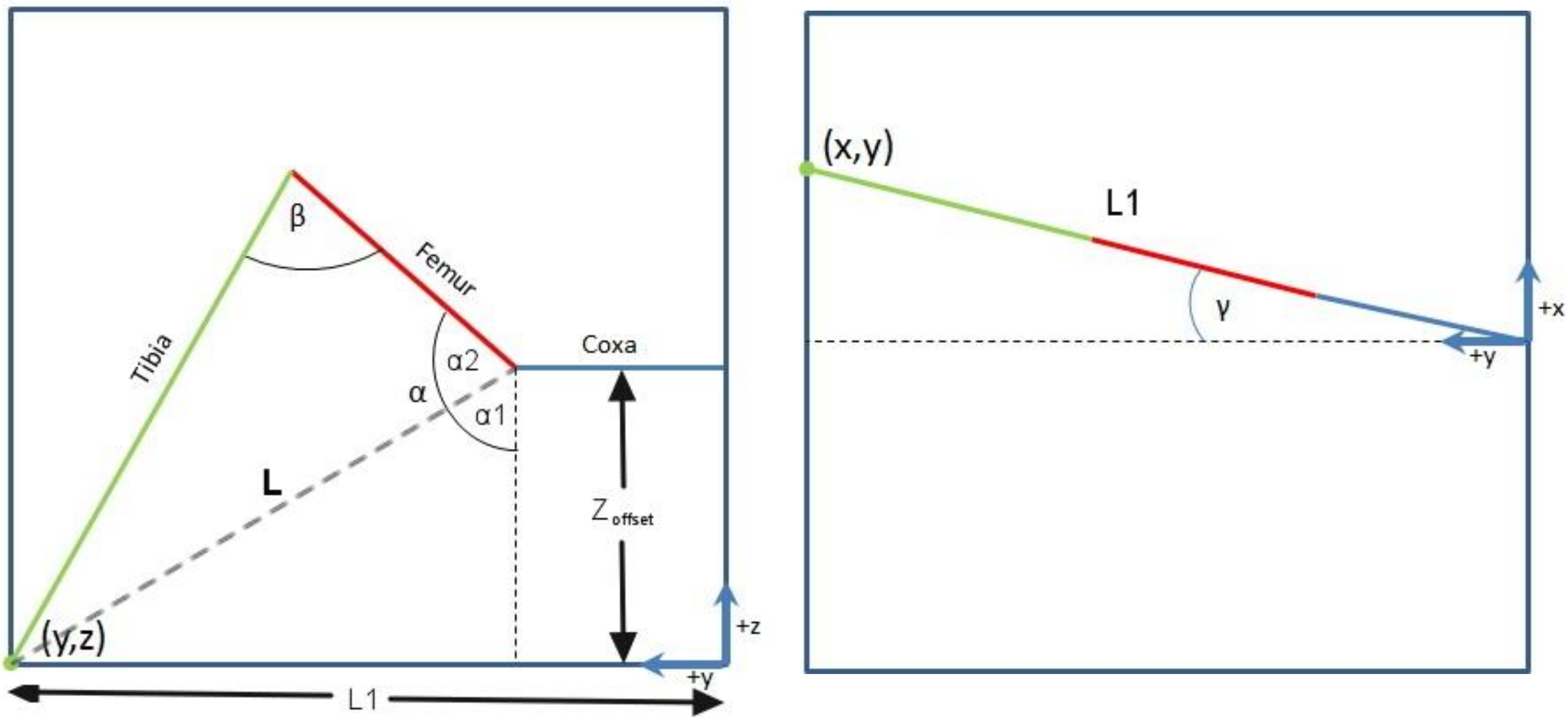
- 3 servos (3 active DoF)
- target position (x,y,z) of foot on the ground
- goal: find servo angles





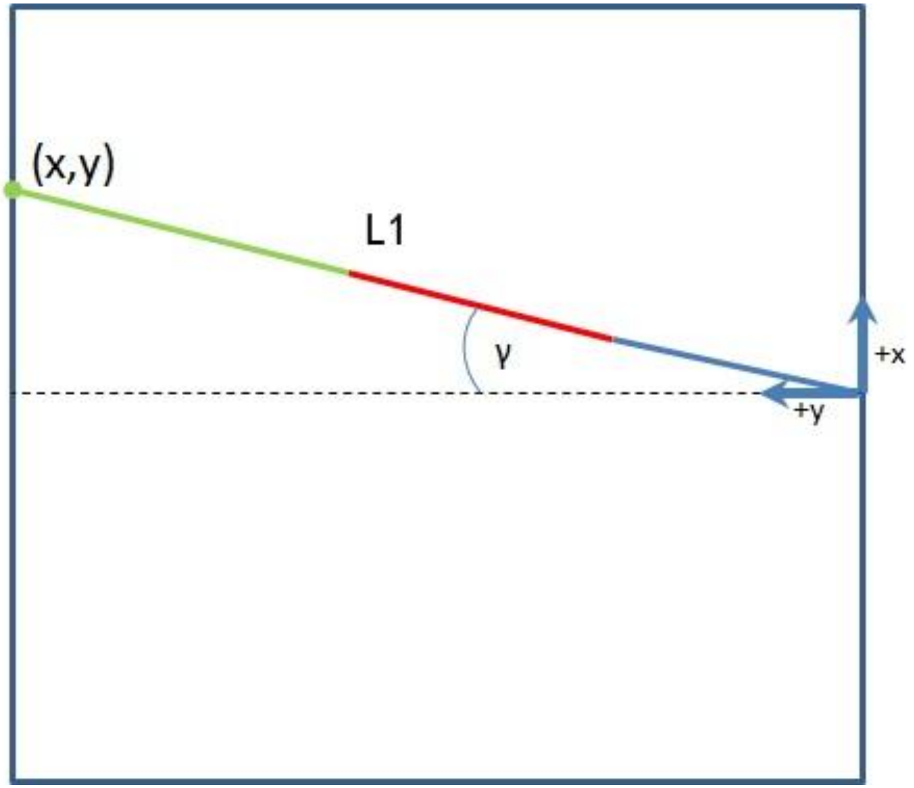
Example Analytical

3 servos = 3 angles



Example Analytical

gamma (leg for- & backward)

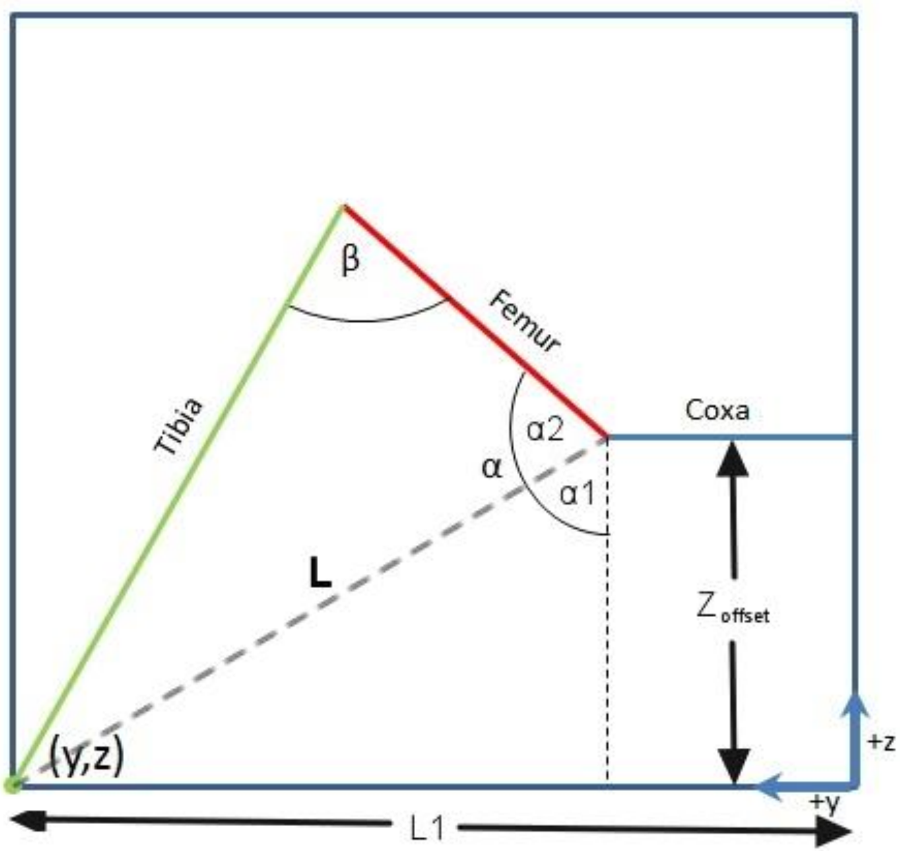


$$\frac{x}{y} = \tan(\gamma)$$

$$\rightarrow \gamma = \tan^{-1}\left(\frac{x}{y}\right)$$

Example Analytical

alpha, beta: up-down plus placement from body



split alpha in two parts:

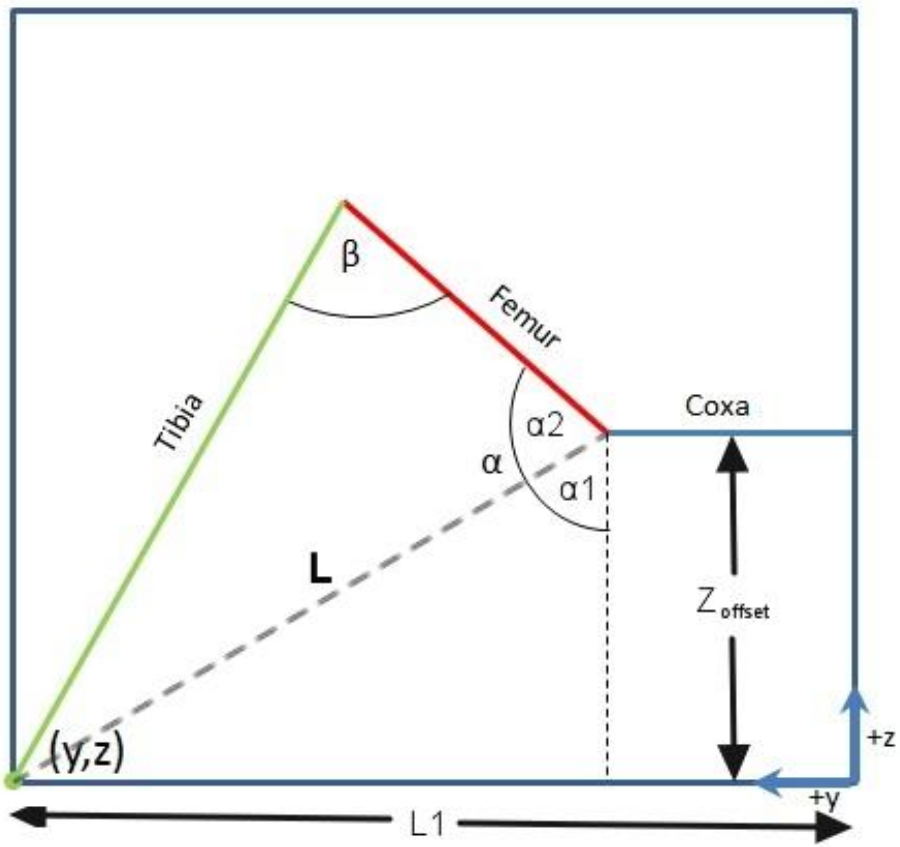
$$\alpha_1 = \cos^{-1}\left(\frac{Z_{offset}}{L}\right)$$

with

$$L = \sqrt[2]{Z_{offset}^2 + (L1 - coxa)^2}$$

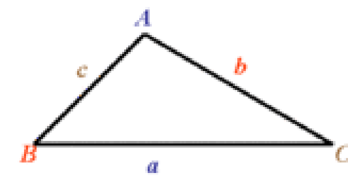
Example Analytical

alpha, beta: up-down plus placement from body



alpha$_2$, beta: cosine rules

Cosine Rule



$$a^2 = b^2 + c^2 - 2bc\ \cos A$$
$$b^2 = a^2 + c^2 - 2ac\ \cos B$$
$$c^2 = b^2 + a^2 - 2ab \cos C$$

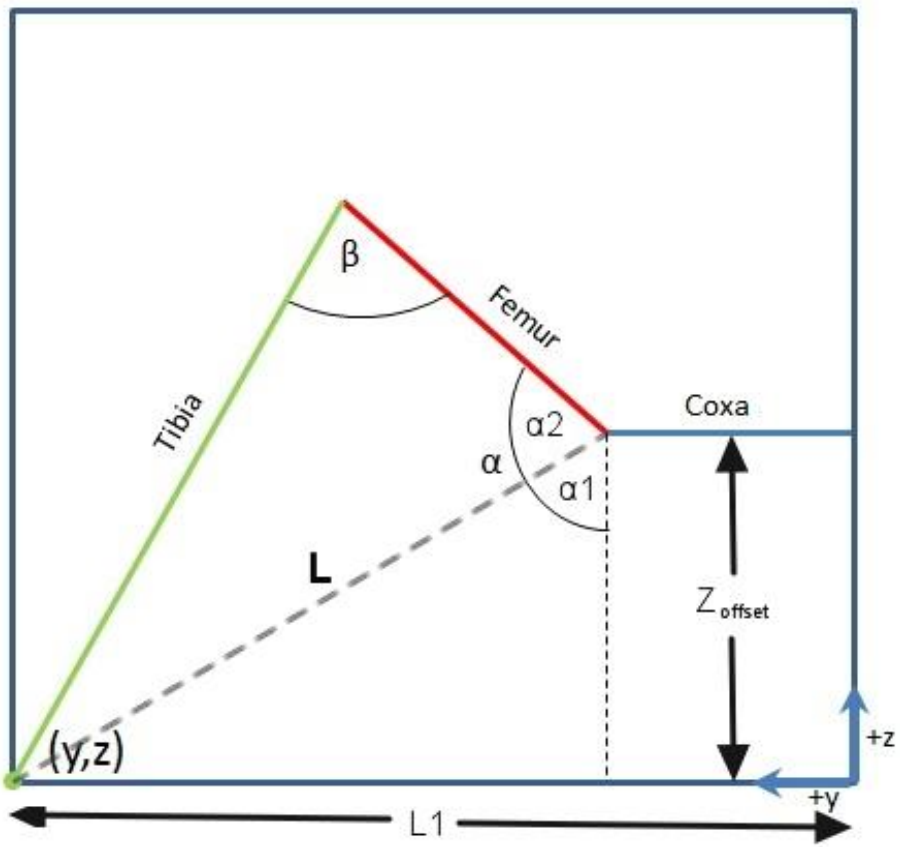
Which one to use depends whether the unknown is a length or an angle

The formula can be rearranged to:

$$\cos A = \frac{b^2 + c^2 - a^2}{2bc}$$
$$\cos B = \frac{a^2 + c^2 - b^2}{2ac}$$
$$\cos C = \frac{a^2 + b^2 - c^2}{2ab}$$

Example Analytical

alpha, beta: up-down plus placement from body



hence alpha_2 and beta:

$$Tibia^2 = Femur^2 + L^2 - 2(Femur)(L)\cos(\alpha_2)$$

$$\rightarrow \alpha_2 = \cos^{-1}\frac{Tibia^2 - Femur^2 - L^2}{-2(Femur)(L)}$$

$$L^2 = Tibia^2 + Femur^2 - 2(Tibia)(Femur)\cos(\beta)$$

$$\rightarrow \beta = \cos^{-1}\frac{L^2 - Tibia^2 - Femur^2}{-2(Tibia)(Femur)}$$

Example Analytical

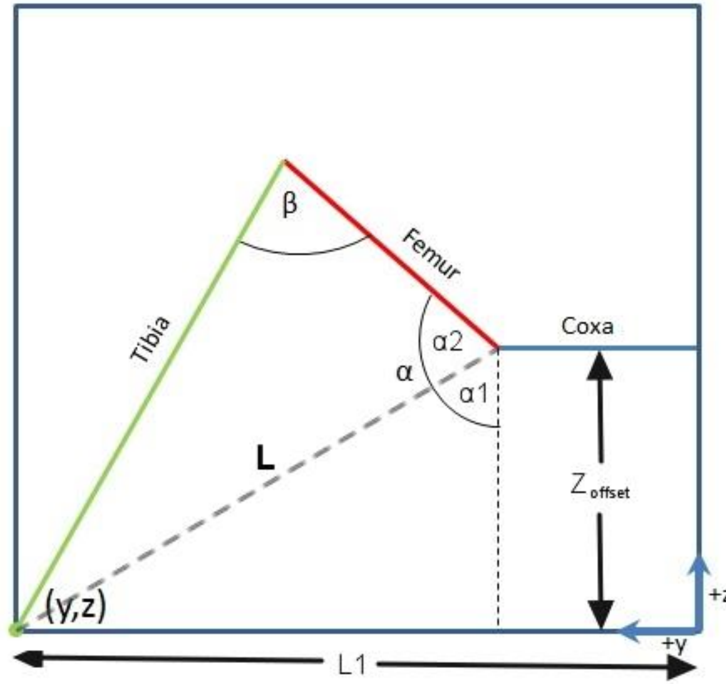
everything together:

$$\alpha = \cos^{-1}\left(\frac{Z_{offset}}{L}\right) + \cos^{-1}\frac{Tibia^2 - Femur^2 - L^2}{-2(Femur)(L)}$$

$$\beta = \cos^{-1}\frac{L^2 - Tibia^2 - Femur^2}{-2(Tibia)(Femur)}$$

$$\gamma = \tan^{-1}\left(\frac{x}{y}\right)$$

with $L = \sqrt[2]{Z_{offset}^2 + (L1 - coxa)^2}$



Arm & Hand Decoupling

- 6-DOF manipulator with a spherical wrist
- inverse kinematics may be separated
  - inverse position kinematics
  - inverse orientation kinematics
- first find position of wrist axes
- second find the orientation of the wrist

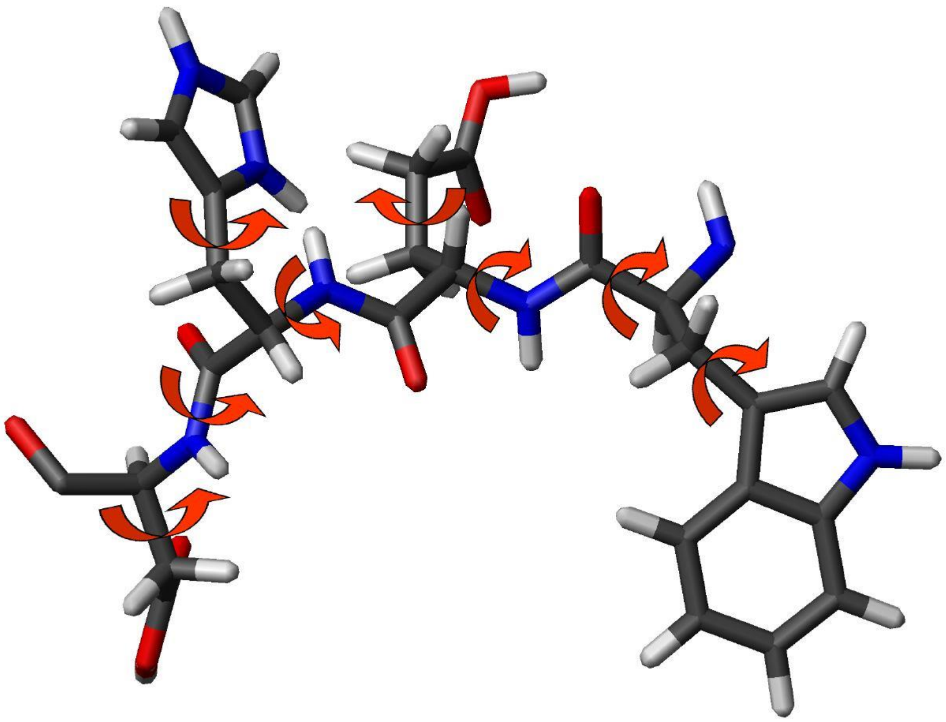
Arm & Hand Decoupling

hence closed form IK

for many (all commercial) robot arms

Numerical IK

analytical approaches can have their limits



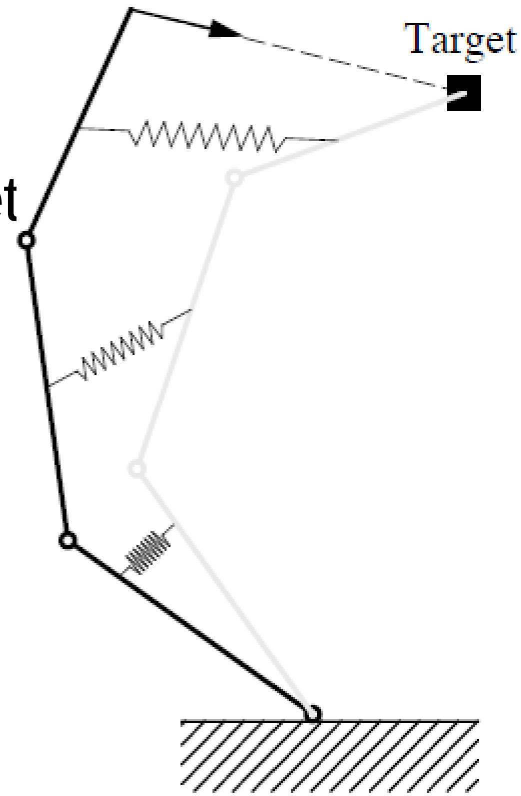
=> numerical approach

Numerical IK

common technique: **Newton's method**
(in general quite useful)

basic idea:

- use derivative to iterate to target
- i.e., minimize distance error
- over the DoF parameters
- in iterative process



(Basic) Newton’s Method

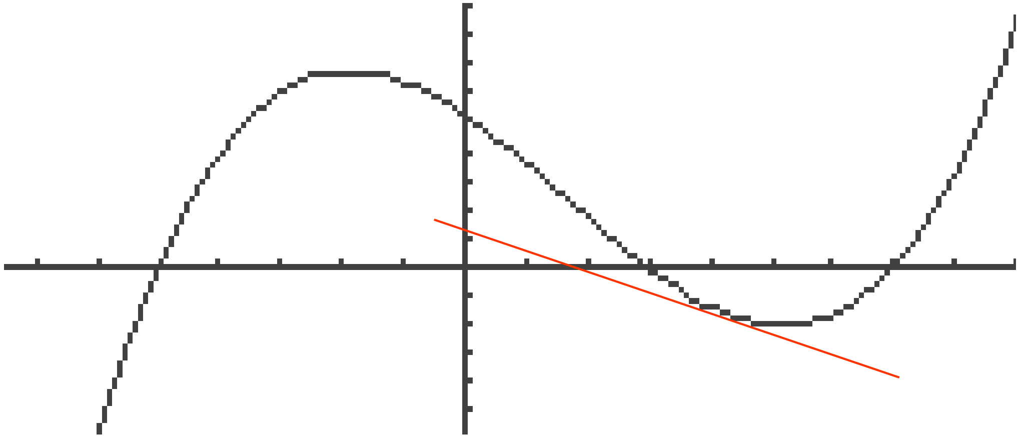
- aka Newton Raphson method
- to find roots of a function

i.e., given function f(x), find

- x’ where f() crosses the x-axis (the root)
- i.e., f(x’) = 0

Find Root of a Function

x’ where f() crosses the x-axis
i.e., f(x’) = 0



Newton:

- tangent line close to the root
- crosses the x-axis close to the root

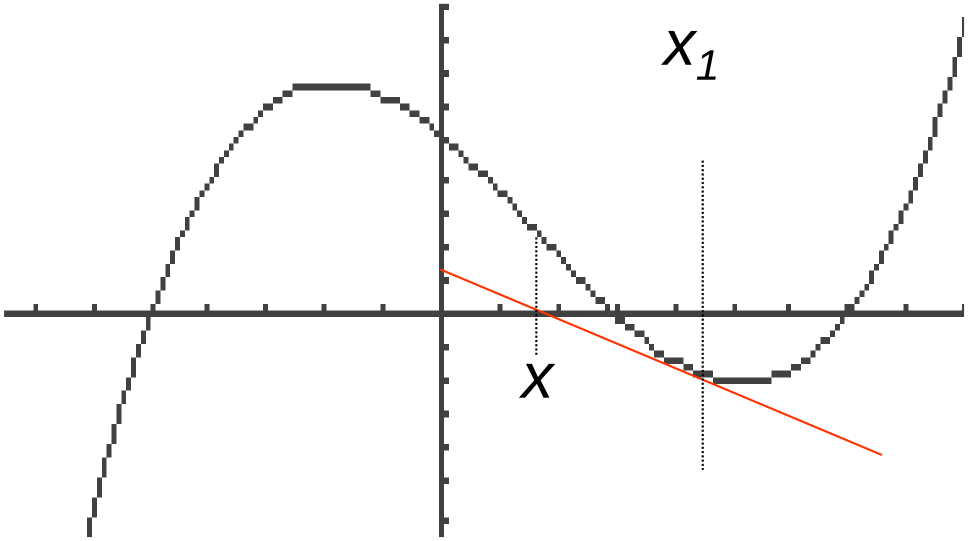
Newton's Method

tangent line

$$y = f(x_1) + f'(x_1)(x - x_1)$$

let y = 0, solve for x

$$x = x_1 - \frac{f(x_1)}{f'(x_1)}$$

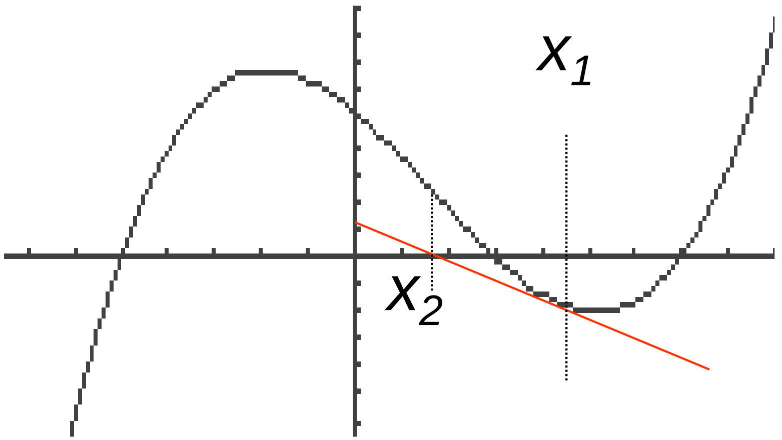


new point *x* as second (and usually better) estimate

Newton's Method

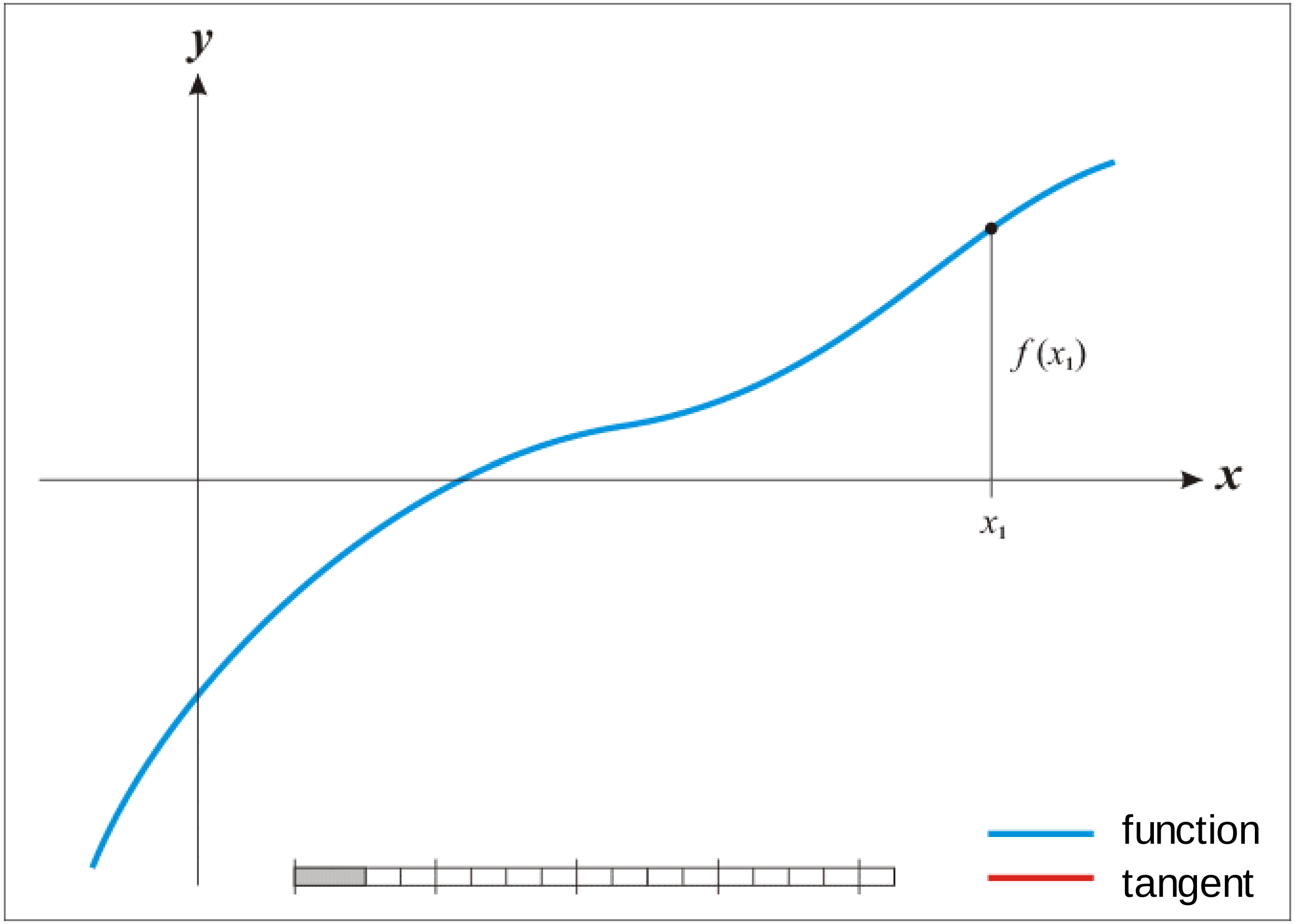
keep on iterating, i.e.,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

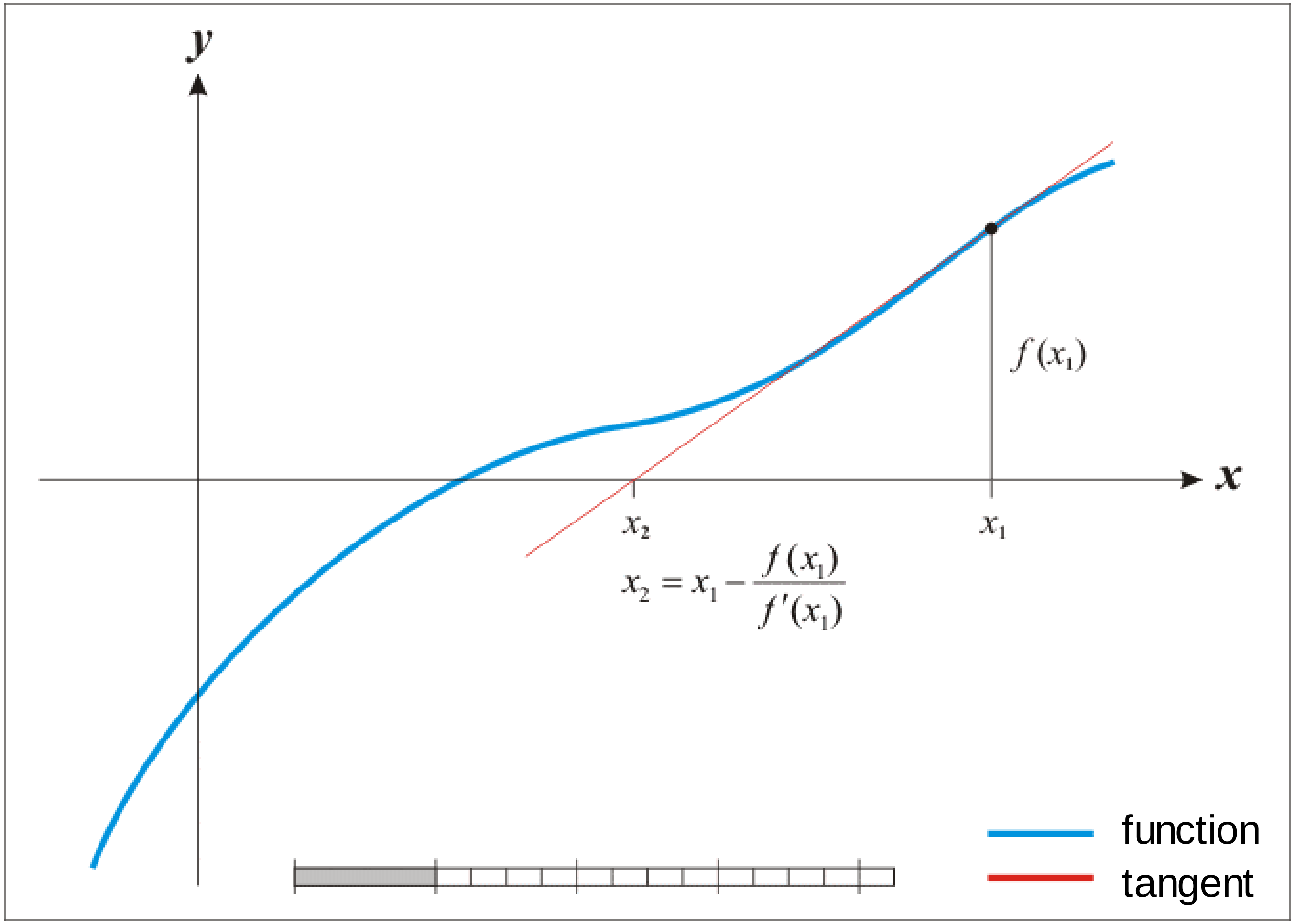


until $f(x_{n+1})$ close enough to Zero

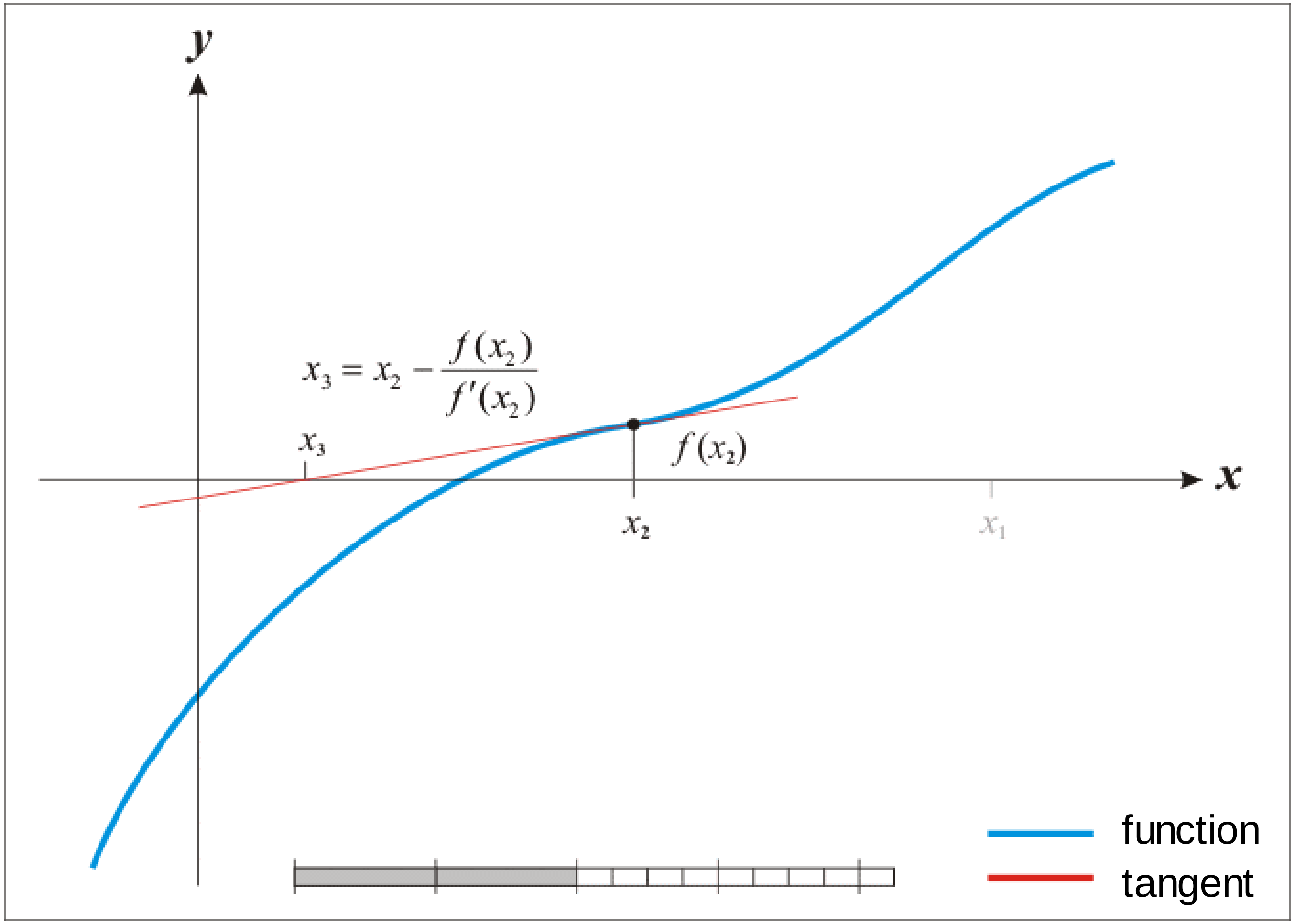
Newton’s Method



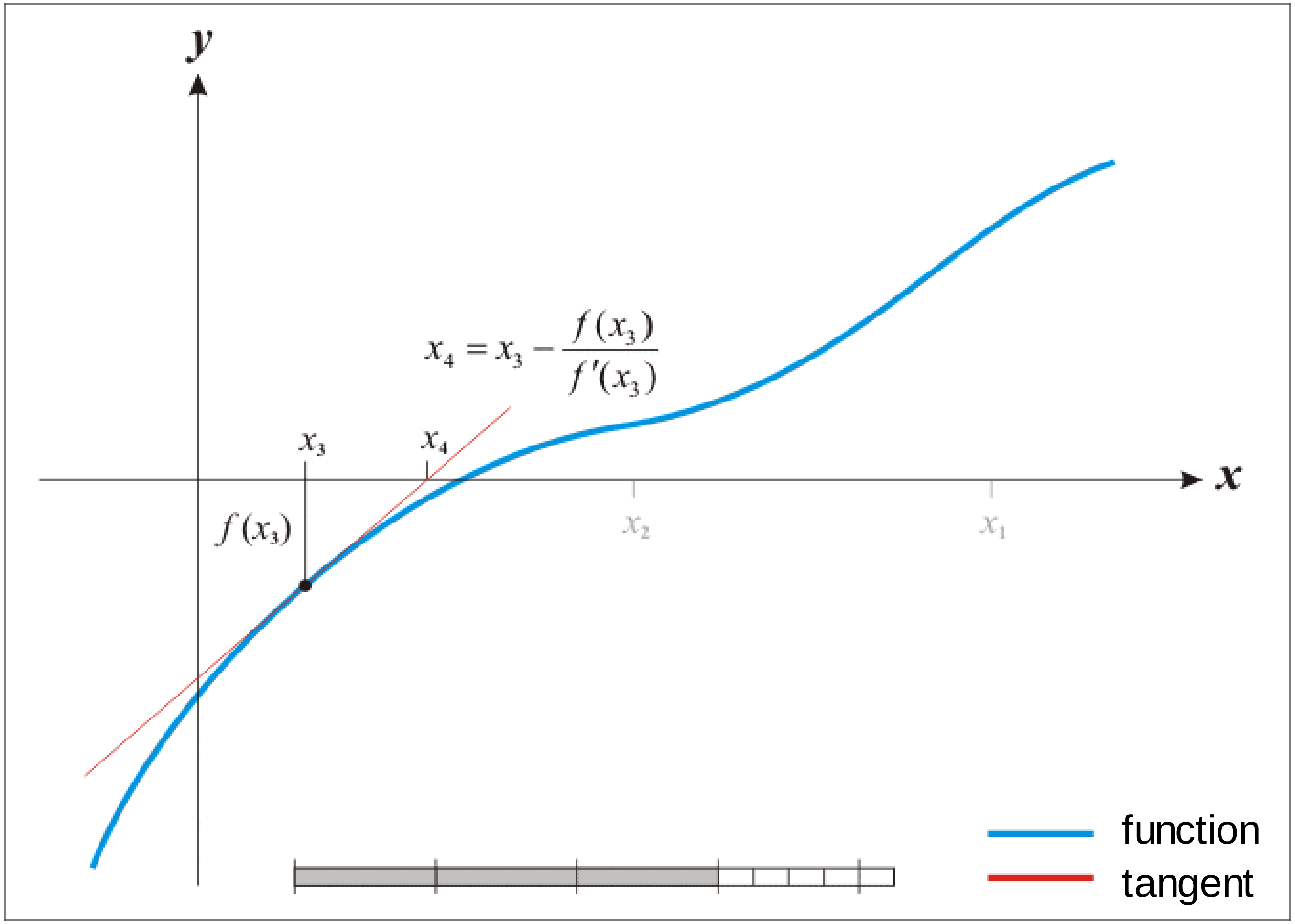
Newton's Method



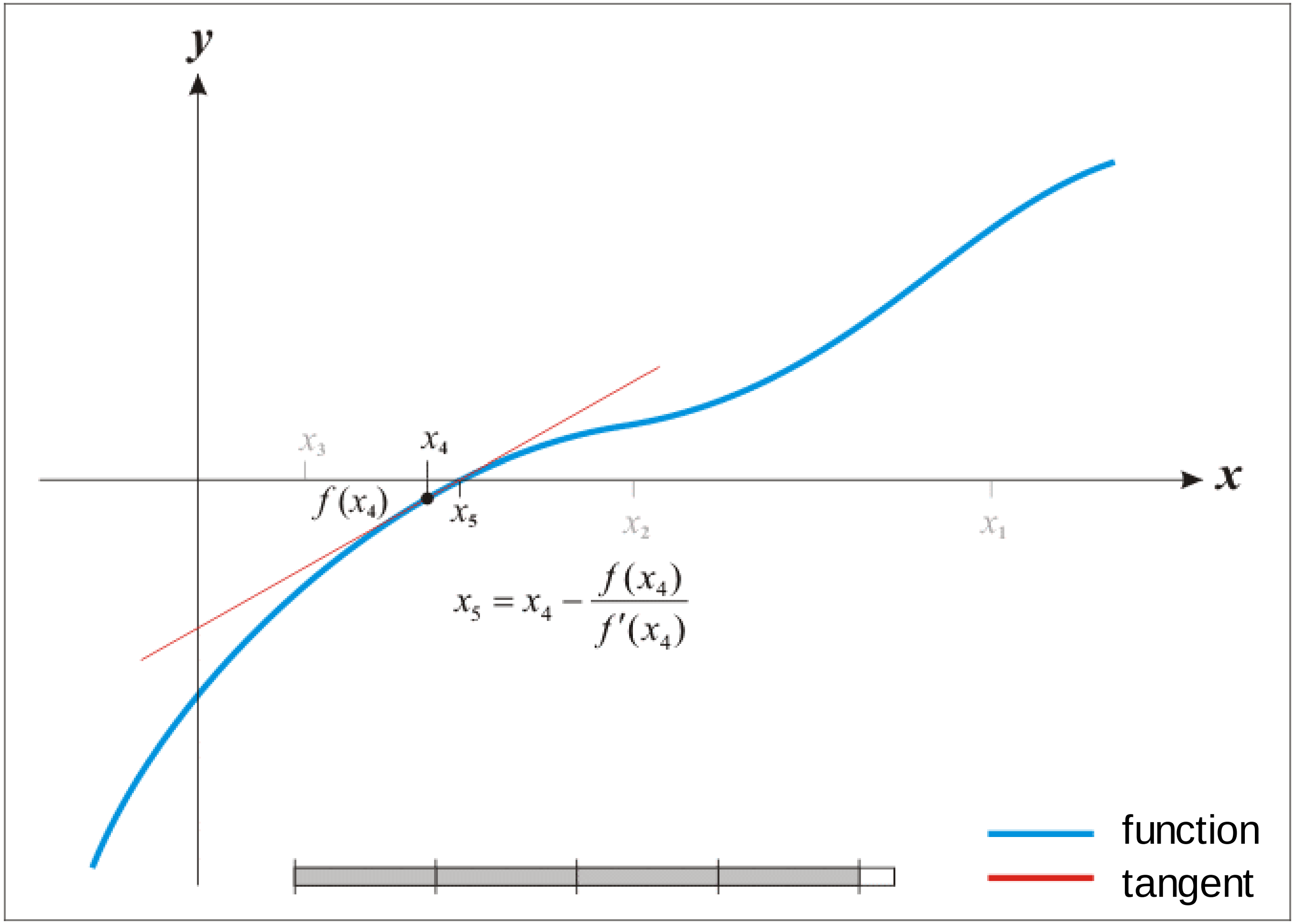
Newton's Method



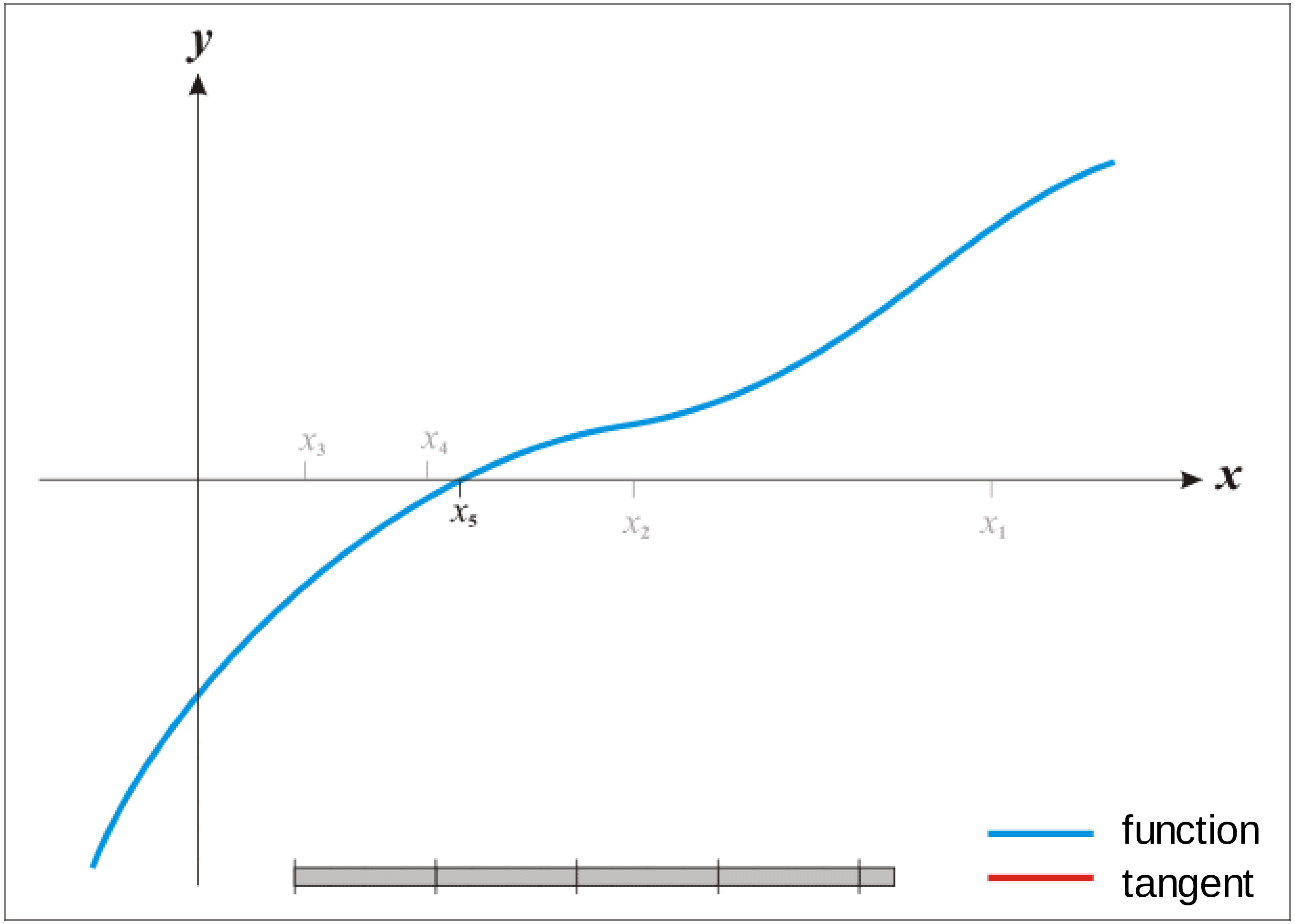
Newton's Method



Newton's Method



Newton's Method



Notes on Newton

- f() not differentiable, or
- f'() too hard/tedious to determine

local approximation of the derivative:

$$f'(x) \cong \left(\frac{f(x+\delta) - f(x)}{\delta} \right)$$

Notes on Newton

- quadratic convergence
  (under [medium optimistic] prerequisites)
- need to start “close enough” to root

f(x), f’(x) defined on interval $\mathrm{I} = \left[x_* - \mu, x_* + \mu\right]$ where $f(x_*) = 0, \mu > 0$
and positive constants rho, delta exist such that

$$\left| f'\left(x\right)\right| \geq \rho \text{ for all } x \in \mathrm{I}$$

$$\left| f'(x) - f'(y)\right| \leq \delta \left| x - y\right| \text{for all } x, y \in \mathrm{I}$$

$$\mu \leq \frac{\rho}{\delta}$$

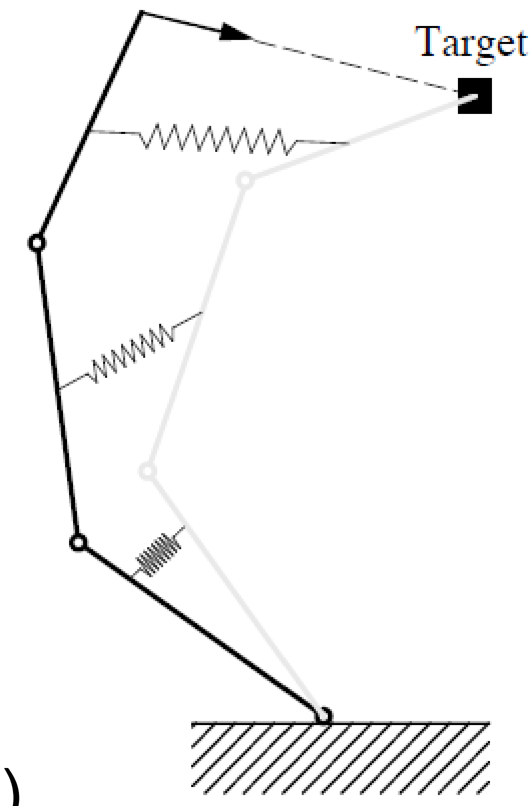
if x_c is in **I**, then $x_+ = x_c - \dfrac{f(x_c)}{f'(x_c)}$ is in **I** and

x_+ is at least half the distance to x_* than x_c was

$$\left| x_+ - x_* \right| \leq \frac{\delta}{2\rho} \left| x_c - x_* \right|^2 \leq \frac{1}{2} \left| x_c - x_* \right|$$

Newton & IK

- forward kinematics: $\mathbf{p} = K(\mathbf{q})$
  - joint parameters $\mathbf{q}$ (n-dimensional configuration space)
  - pose $\mathbf{p}$ (end-effector: 6 DoF in 3D space)
- target pose $\mathbf{t}$
- IK: find $\mathbf{q}$ with $(\mathbf{t} - \mathbf{p}) = (\mathbf{t} - K(\mathbf{q})) = 0$



$\Rightarrow$ need n-dim Newton

$\Rightarrow$ need n-dim equivalent of 1st derivate f'()

Jacobian

Jacobian matrix

- matrix of all first-order partial derivatives
- of a multi-variate, vector-valued function

F : $\mathbb{R}^n \rightarrow \mathbb{R}^m$

Jacobian **J** of F() : *m*×*n* matrix $Jij = \frac{\partial Fi}{\partial xj}$

Jacobian

F : $\mathbb{R}^n \rightarrow \mathbb{R}^m$

Jacobian **J** of F() : $m \times n$ matrix

$$J(x) = \left(\frac{\partial F(x)}{\partial x_1} \cdots \frac{\partial F(x)}{\partial x_n} \right) = \begin{pmatrix} \frac{\partial F_1(x)}{\partial x_1} & \cdots & \frac{\partial F_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_m(x)}{\partial x_1} & \cdots & \frac{\partial F_m(x)}{\partial x_n} \end{pmatrix}$$

Jacobian

common notations for the Jacobian

- **J**, F() assumed to be known from context: **J**
- F() as index: $\mathbf{J_F}$
- D like *Derivative*: **DF()**

  D(F)
- delta notation: $$\frac{\partial(f_1,\ldots,f_m)}{\partial(x_1,\ldots,x_n)}$$

Jacobian

simple example
$$F(x) = \begin{pmatrix} F_1(x_1, x_2) \\ F_2(x_1, x_2) \end{pmatrix} = \begin{pmatrix} {x_1}^2 + {x_2}^2 - 1 \\ 5{x_1}^2 + 2{x_2}^2 + 4 \end{pmatrix}$$

$$J(x) = \begin{pmatrix} \dfrac{\partial F_1(x)}{\partial x_1} & \dfrac{\partial F_1(x)}{\partial x_2} \\ \dfrac{\partial F_2(x)}{\partial x_1} & \dfrac{\partial F_2(x)}{\partial x_2} \end{pmatrix}$$

$$= \begin{pmatrix} \dfrac{\partial({x_1}^2 + {x_2}^2 - 1)}{\partial x_1} & \dfrac{\partial({x_1}^2 + {x_2}^2 - 1)}{\partial x_2} \\ \dfrac{\partial(5{x_1}^2 + 2{x_2}^2 + 4)}{\partial x_1} & \dfrac{\partial(5{x_1}^2 + 2{x_2}^2 + 4)}{\partial x_2} \end{pmatrix}$$

$$= \begin{pmatrix} 2x_1 & 2x_2 \\ 10x_1 & 4x_2 \end{pmatrix}$$

Jacobian

differentiation may be difficult or even impossible

- option: numerical approximation (like local approx. f'(x))
- i.e., use small delta on x to sample neighborhood of F(x)

$$J(x) = \left(\frac{\partial F(x)}{\partial x_1} \cdots \frac{\partial F(x)}{\partial x_n} \right)$$

$$\frac{\partial F(x)}{\partial x_i} \approx \frac{\Delta F(x)}{\Delta x_i} = \left(\frac{\Delta F(x)}{\Delta x_1} \cdots \frac{\Delta F(x)}{\Delta x_n} \right)$$

Jacobian: Approximation

simple example $$F(x) = \begin{pmatrix} F_1(x_1, x_2) \\ F_2(x_1, x_2) \end{pmatrix} = \begin{pmatrix} {x_1}^2 + {x_2}^2 - 1 \\ 5{x_1}^2 + 2{x_2}^2 + 4 \end{pmatrix}$$

$x = (1, 2)$

proper Jacobian $$J(1,2) = \begin{pmatrix} 2x_1 & 2x_2 \\ 10x_1 & 4x_2 \end{pmatrix} = \begin{pmatrix} 2 \cdot 1 & 2 \cdot 2 \\ 10 \cdot 1 & 4 \cdot 2 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 10 & 8 \end{pmatrix}$$

approx. Jacobian (delta = 0.1)

$$J(1,2) \approx \begin{pmatrix} \dfrac{F_1(x_1+\partial, x_2) - F_1(x_1, x_2)}{\partial} & \dfrac{F_1(x_1, x_2+\partial) - F_1(x_1, x_2)}{\partial} \\ \dfrac{F_2(x_1+\partial, x_2) - F_2(x_1, x_2)}{\partial} & \dfrac{F_2(x_1, x_2+\partial) - F_2(x_1, x_2)}{\partial} \end{pmatrix}$$

$$= \begin{pmatrix} \dfrac{1.1^2 - 1^2}{0.1} & \dfrac{2.1^2 - 2^2}{0.1} \\ \dfrac{5 \cdot 1.1^2 - 5 \cdot 1^2}{0.1} & \dfrac{2 \cdot 2.1^2 - 2 \cdot 2^2}{0.1} \end{pmatrix}$$

$$= \begin{pmatrix} 2.1 & 4.1 \\ 10.5 & 8.2 \end{pmatrix}$$

Jacobian

option: numerical approximation (like local approx. f'(x))

$$\frac{\partial F(x)}{\partial x_i} \approx \frac{\Delta F(x)}{\Delta x_i}$$

- but whenever possible: try to properly derive J
- by hand or use symbolic math software
  - Mathematica, (symbolic differentiation in) MATLAB
  - or open alternatives like Scilab, GNU Octave

Multidimensional Newton

find **x***: F(**x***) = 0

- with F : $\mathbb{R}^N \rightarrow \mathbb{R}^N$
- i.e., **x***$\in \mathbb{R}^N$

$$F(x^*) = F(x_k) + J(x_k)(x^* - x_k)$$

$$\Rightarrow x_{k+1} = x_k - J(x_k)^{-1} F(x_k)$$

iteration function