Mathematical tools for robotics

J-P Merlet
HEPHAISTOS project
INRIA Sophia-Antipolis
France

Some math problems in robotics

Some math problems in robotics

- solve a square system of equations $\mathbf{F}(\mathbf{X}) = \mathbf{0}$ (algebraic or not)

Some math problems in robotics

- solve a square system of equations $\mathbf{F}(\mathbf{X}) = \mathbf{0}$ (algebraic or not)
  - appears in kinematics
  - inverse kinematics (IK): find the articular coordinates $\mathbf{\Theta}$ for a given pose $\mathbf{X}$ of the end-effector
  - direct kinematics (DK): find the pose(s) $\mathbf{X}$ of the end-effector for given articular coordinates $\mathbf{\Theta}$
  - possibly multiple solutions, find them all or a single one, real-time or not

Some math problems in robotics

- solve a square system of equations $\mathbf{F}(\mathbf{X}) = \mathbf{0}$ (algebraic or not)
- find the extremum of $\mathbf{F}(\mathbf{X})$ over a bounded domain, possibly with constraints

Some math problems in robotics

- find the extremum of $\mathbf{F}(\mathbf{X})$ over a bounded domain,
  is that really necessary ? not always!

Example: show that a motor torque τ will be lower than the maximal motor torque τ_{max} for any pose of the robot in a given workspace $\mathcal{W}$
$\rightarrow$ binary question, you don't really care about the exact value of the maximum $\Rightarrow$ optimization is overkill

- $\exists \mathbf{X} \in \mathcal{W} / \tau(\mathbf{X}) > \tau_{max}$?
- $\forall \mathbf{X} \in \mathcal{W} \quad \tau(\mathbf{X}) < \tau_{max}$

Some math problems in robotics

- solve a square system of equations $\mathbf{F}(\mathbf{X}) = \mathbf{0}$ (algebraic or not)
- find the extremum of $\mathbf{F}(\mathbf{X})$ over a bounded domain, possibly with constraints
- managing uncertainties

Some math problems in robotics

- managing uncertainties

$$\mathbf{F}(\mathbf{X}) \rightarrow \mathbf{F}(\mathbf{X}, \mathbf{P})$$

where $\mathbf{P}$ are parameters with bounded uncertainties

$\mathbf{P} \in \mathcal{W}_P = \mathbf{P}_0 \pm \mathbf{\Delta P}$

can we still guarantee that

$$\forall \mathbf{X} \in \mathcal{W}, \forall \mathbf{P} \in \mathcal{W}_P, \quad \tau(\mathbf{X}, \mathbf{P}) < \tau_{max}?$$

Some math problems in robotics

- solve a square system of equations $\mathbf{F}(\mathbf{X}) = \mathbf{0}$ (algebraic or not)
- find the extremum of $\mathbf{F}(\mathbf{X})$ over a bounded domain, possibly with constraints
- managing uncertainties
- numerical round-off errors

Some math problems in robotics

- numerical round-off errors

Be aware of calculation errors due to round-off errors

- a computer knows only a finite set of numbers: e.g. 0.1 does not exist for our computers: 0.099999994039536,0.100000008940697
- a clever system of rounding is implemented
- but it may fail even for simple expressions

Round-off errors

Example

$$f(x,y) = \frac{33375}{100}y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \frac{55}{10}y^8 + \frac{x}{2y}$$

to be evaluated at $x = 77617, y = 33096$

Round-off errors

Example

$$f(x,y) = \frac{33375}{100}y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \frac{55}{10}y^8 + \frac{x}{2y}$$

to be evaluated at $x = 77617, y = 33096$

Matlab	Scilab	C (with double)	Maple (10 Digits)	Maple (20 digits)
$-1.1806\ 10^{21}$	$-1.1806\ 10^{21}$	1.17260394	$0.1\ 10^{28}$	$-1.0\ 10^{17}$

- interval evaluation: $[-0.56610^{23}, 0.55510^{23}]$
- exact value: $\approx$ -0.8273960599

Don't always believe your computer!

Round-off errors

- check your results with packages allowing extended arithmetic (Maple, MPFR (C)): increase your confidence in the result but is not completely foolproof
- there are numerous equivalent mathematical formulations for a given problem but they may not be computer-numerically equivalent

  $x^2 + 2x + 1 \neq (x + 1)^2$

Newton method

Find a solution of $\mathbf{F}(\mathbf{X}) = \mathbf{0}$ with an initial guess $\mathbf{X_0}$, jacobian $\mathbf{J} = \partial\mathbf{F}/\partial\mathbf{X}$

iterative scheme:

$$\mathbf{X_k} = \mathbf{X_{k-1}} - \mathbf{J^{-1}}(\mathbf{X_{k-1}})\mathbf{F}(\mathbf{X_{k-1}})$$

stops when $||\mathbf{F}(\mathbf{X_k})|| \leq \epsilon$

Newton method

Advantages:

- works for any $\mathbf{F}$ at least C_1
- simple
- fast quadratic convergence

Newton method

Advantages:

- works for any $\mathbf{F}$ at least C_1
- simple
- fast quadratic convergence

Drawbacks:

- *"we are close to a solution S so Newton will converge to S"*: WRONG!
- Newton exhibits a fractal behavior
- a safe Newton method may be obtained with the Kantorovitch theorem

Kantorovitch theorem

1. the Jacobian of $\mathbf{F}$ has an inverse Γ_0 at $\mathbf{X_0}$ with $||\Gamma_0(\mathbf{X_0})|| \leq A_0$
2. $||\Gamma_0\mathbf{F}(\mathbf{X_0})|| \leq 2B_0$
3. $U = \{\mathbf{X}/||\mathbf{X} - \mathbf{X_0}|| \leq 2B_0\}$
4. $\sum_{k=1}^{n} |\frac{\partial^2 F_i(\mathbf{X}}{\partial X_j \partial X_k}| \leq C$ for $i, j = 1, \ldots, n$ and $\mathbf{X} \in U$

if $2nA_0B_0C \leq 1$, then:

- there is an unique solution of $\mathbf{F}(\mathbf{X}) = \mathbf{0}$ in U
- Newton used with $\mathbf{X_0}$ as estimate of the solution will converge toward this solution

Kantorovitch theorem

- $||\Gamma_0 \mathbf{F}(\mathbf{X_0})|| \leq 2B_0$
- $U = \{\mathbf{X}/||\mathbf{X} - \mathbf{X_0}|| \leq 2B_0\}$

Hence

- B_0 provides an error bound on $||SS_n||$
- using extended arithmetic allows one to decrease $||SS_n||$ at will $\rightarrow$ exact solution

safe Newton: at t_k you have found a solution S_k and at t_{k+1} you are sure that the new solution lies in a ball B centered at S_k with known radius. Kantorovitch

- will allow to check if there is a unique solution in B
- otherwise better stop the robot!

Algebraic geometry

all equations of $\mathbf{F}$ are polynomials in $\{x_1, x_2, \ldots, x_n\}$

Let's start simple:

- intersection of 2 circles in a plane

$$(x - u_1)^2 + (y - v_1)^2 = r_1^2$$
$$(x - u_2)^2 + (y - v_2)^2 = r_2^2$$

- how many intersection points ?
- how to calculate their intersection points ?

Algebraic geometry

How many intersection point ?

Bezout theorem: 2 curves of order n and m have exactly nm real and complex intersection points

Strange for the intersection of 2 circles ?

Algebraic geometry

How many intersection point ?

Bezout theorem: 2 curves of order n and m have exactly nm real and complex intersection points

Strange for the intersection of 2 circles ?

No: all circles includes two circular points at infinity

Algebraic geometry

How to calculate the intersection point ?

Algebraic geometry

How to calculate the intersection point ?

- subtract the 2 equations, the x^2, y^2 terms disappear, remains a linear equation $H = ax + by + c = 0$
- solve H in x
- report the result in one equation: you get a 2nd order polynomial in y

Algebraic geometry

what about

$$(x-1)^2+(y-1)^2=1$$
$$(3x-2)^2+(2y-2)^2=2^2$$

Algebraic geometry: elimination

$\mathbf{F} = \{F_1(x_1, \ldots, x_n) = 0, \ldots F_n(x_1, \ldots, x_n) = 0\}$

- monomials: all terms in x_i^k, including $x_i^0 = 1$
- assume that one x_j has a value
- list $\mathcal{L}$ of all monomials in $\mathbf{F}$
- $F_i = \mathbf{A_i}^T(x_j)\mathcal{L}$
- $\mathbf{F} = \mathbf{A}(x_j)\mathcal{L} = \mathbf{0}$
- if $\mathbf{A}$ is not square: add equations in $\mathbf{F}$ e.g. $x_k F_i$

$$\mathbf{F}' = \mathbf{A}'(x_j)\mathcal{L} = \mathbf{0}$$

$\mathcal{L} \neq \mathbf{0} \Rightarrow |\mathbf{A}'(x_j)| = 0$ provides an univariate polynomial in x_j

Algebraic geometry: elimination

- finding $\mathbf{A}'(x_j)$ is not always easy
- but for 2 polynomials it is easy: resultant

$$(1)(x-1)^2+(y-1)^2=1$$
$$(2)(3x-2)^2+(2y-2)^2=2^2$$

- `resultant`((1),(2),x)=$25y^4-140y^3+252y^2-128y+4$
- `resultant`((1),(2),y)=$25x^4-40x^3+42x^2-40x+1$

Algebraic geometry: Groebner basis

$$\mathbf{F} = \{F_1(x_1, \ldots, x_n) = 0, \ldots F_n(x_1, \ldots, x_n) = 0\}$$

An algorithm allows to transform $\mathbf{F}$ in an equivalent triangular system

$$G_1(x_1, \ldots, x_n) = 0$$
$$G_2(x_1, \ldots, x_{n-1}) = 0$$
$$\ldots$$
$$G_n(x_1) = 0$$

Algebraic geometry: Groebner basis

Drawbacks

- must have integer or rational coefficients
- the coefficients of the G_i may be very large
- **complexity**: double exponential in the number of unknowns and in the degree of the equations

incorporated in `Maple`

Continuation

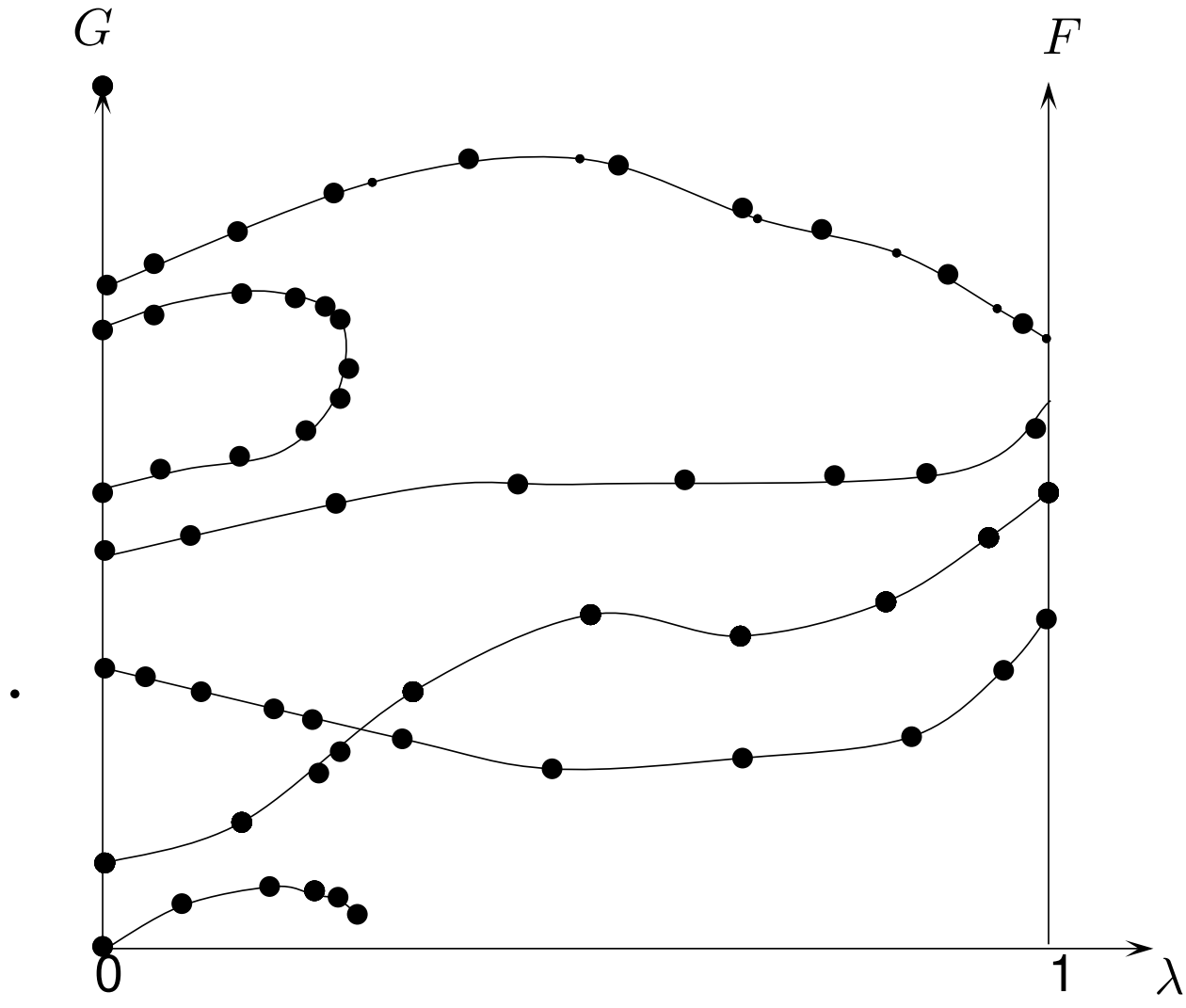
- valid for any type of $\mathbf{F}$
- require an *equivalent* system $\mathbf{G}$ for which all solutions are known

Principle

$$\mathbf{H}(\mathbf{X}) = \mathbf{G}(\mathbf{X}) + \lambda(\mathbf{F}(\mathbf{X}) - \mathbf{G}(\mathbf{X})$$

- $\lambda =$0 $\rightarrow \mathbf{H} = \mathbf{G}$ with solutions S_0^i
- starting from $\lambda = 0$ increment λ by ϵ and use Newton with guess S_λ^i to obtain the solutions of $\mathbf{H}_{\lambda+\epsilon}$
- continue until $\lambda = 1 \Rightarrow$ you get the solutions of $\mathbf{F}$

Continuation



Continuation

Drawbacks

- finding $\mathbf{G}$? possible for algebraic system but possibly large number of branches to follow
- possible branch jump: Newton jumps to a solution of another branch
- singularities, complex to real solutions

good method to compute workspace border, singularity curves

Reference: Allgower, E.L., Numerical continuation methods, Springer 1990

Interval arithmetic

calculation with interval instead of number

- F a function of $\mathbf{X}$
- $\mathbf{X}$ has interval values: $\hat{\mathbf{X}} = [\underline{X}, \overline{X}]$

an interval evaluation of $F(\hat{\mathbf{X}})$ is an interval $[U, V]$ such that

$$\forall \mathbf{X} \in \hat{\mathbf{X}} \quad U \leq F(\mathbf{X}) \leq V$$

- U, V are guaranteed
- over $\hat{\mathbf{X}}$ we have $Min(F) \geq U$, $Max(F) \leq V$
- tight interval evaluation: $U = Max(F)$ and $V = Min(F)$

Interval arithmetic

simplest interval evaluation method: natural evaluation$\rightarrow$
replace mathematical operators by their interval equivalent

Example: $f = x - sin(x) + 1$ when $x \in [0, 1]$

$$[0,1] - [0, sin(1)] + [1,1] = [0,1] + [-sin(1), 0] + [1,1] = [0.158, 2]$$

$$\hat{f}([0,1]) = [0.158, 2]$$

- there is no root of $f(x)$ in [0,1]
- $f(x) \geq 1 - sin(1)\ \forall\ x \in [0,1]$

Interval arithmetic

Advantages:

- **simplicity**
- can be implemented to take into account **round-off errors**

Drawback:

- **sensitivity** to formulation

$$x \in [-2, 1] \quad x^2 + 2x + 1 \to [-3, 7] \qquad (x+1)^2 \to [0, 4]$$

- **overestimation** $\hat{f}([0, 1]) = [0.158, 2]$, real min 1, max $2 - sin(1)$
  - decreases with the width of the intervals
  - no overestimation if unknowns appear only once
  - methods to reduce the width of the interval evaluation

Interval analysis

Solving $\mathbf{F}(\mathbf{X}) = \mathbf{0}$

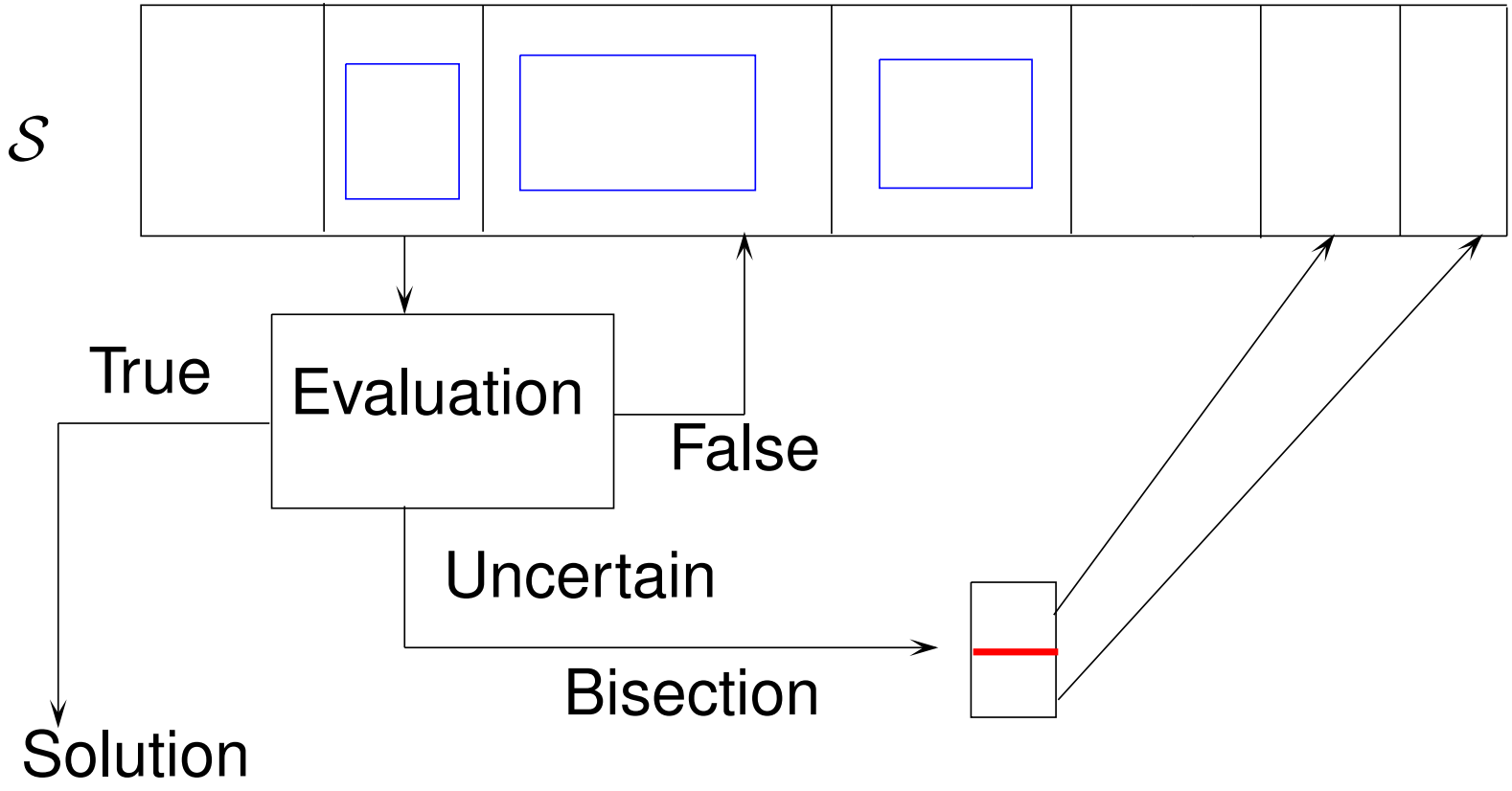
- more precisely finding the roots of $\mathbf{F}$ in $\hat{\mathbf{X}}$
- advantage of interval analysis: all previous methods find all the roots then you have to sort out the roots of interest, if any

Ingredients

- a search box $\hat{\mathbf{X}} = \{\hat{x_1}, \hat{x_2}, \ldots, \hat{x_n}\}$
- a list of box $\mathcal{S} = \{\mathbf{B_1}, \ldots \mathbf{B_k}\}$. Initially $\mathcal{S} = \{\hat{\mathbf{X}}\}$
- $F_i(\hat{\mathbf{B_l}}) = [U, V]$ with $U > 0$ or $V < 0 \Rightarrow$ no root in $\mathbf{B_l}$

Interval analysis

a simple algorithm



Improvements: filtering

$\mathbf{F} = x + sin(2x) + 1 = 0$ for $x \in [-10, 10]$?

rewriting

- interval evaluation of $\mathbf{F}$: [-10,12] $\Rightarrow$ bisection
- $x = -sin(2x) - 1$
- right side evaluation : [-2,0]
- hence search domain [-10,10] $\rightarrow$ [-2,0]

2B

- evaluate for $[\underline{x}, \underline{x} + \epsilon], [\underline{x} + \epsilon, \underline{x} + 3\epsilon], \ldots$
- evaluate for $[\overline{x} - \epsilon, \overline{x}], [\overline{x} - 3\epsilon, \overline{x} - \epsilon], \ldots$
- search domain $\rightarrow$ [-0.4,-0.3]

Improvements: exact solutions

- use the Kantorovitch theorem to determine if a single solution exists in a box. If yes an exact solution will be found by using Newton using as guess the center of the box
- for $\mathbf{F} = x + sin(2x) + 1 = 0$ the Kantorovitch condition is satisfied for [-0.4,-0.3]
- other unicity test are possible

Generalization

Analysis: checking if a property is satisfied over a given workspace

Explicit example:checking if $|\tau| < \tau_{max}$ for all poses $\mathbf{X}$ of a given workspace $\mathcal{W}$ using $\tau = \mathbf{J^T}(\mathbf{X})\mathcal{F}$, external forces $\mathcal{F} = \hat{\mathcal{F}}$, $\mathcal{W} = \hat{\mathcal{W}}$

- check the mid-point of a box $\hat{\tau_{mid}} = \mathbf{J^T}(\mathbf{X_{mid}})\hat{\mathcal{F}}$ leads to $\hat{\tau_{mid}^{j}} = [U_j, V_j]$. If $V_j < -\tau_{max}$ or $U_j > \tau_{max} \rightarrow$ NO.
- $\hat{\tau} = \mathbf{J^T}(\mathbf{B_i})\hat{\mathcal{F}} = [U_j, V_j]$, if $V_j > -\tau_{max}$ and $U_j < \tau_{max} \rightarrow$ box $\mathbf{B_i}$ is ok, otherwise bisect $\mathbf{B_i}$
- if all boxes in $\mathcal{S}$ have been processed: $\mathcal{W}$ is ok

Generalization

Analysis: checking if a property is satisfied over a given workspace, taking into account uncertainties

Explicit example:checking if $|\tau| < \tau_{max}$ for all poses $\mathbf{X}$ of a given workspace $\mathcal{W}$ using $\tau = \mathbf{J^T}(\mathbf{X}, \hat{\mathbf{P}})\mathcal{F}$, $\mathbf{P}$ are model parameters that are uncertain $\rightarrow \hat{\mathbf{P}}$

Same principle except that we may have to consider boxes $\{\mathbf{X}, \mathbf{P}\}$

Result: $|\tau| < \tau_{max}$ is guaranteed for the real robot

Trajectory singularity detection

Purpose: determining if a singularity may be encountered on a trajectory without bothering about where is the singularity

parametric trajectory: $\mathbf{X} = \mathbf{F}(t)$ with $t \in [0, 1]$. We compute $|\mathbf{J}(\mathbf{X}, t = 0)|$, say positive

Objective: if there is a $t = t_1$ such that $|\mathbf{J}(\mathbf{X}, t_1)| < 0$, then there is at least one t_2 in $[0, t_1]$ such that $|\mathbf{J}(\mathbf{X}, t_2)| = 0$ (Rolle theorem)

Trajectory singularity detection

box $B_i(a,b)$: interval vector for $\mathbf{X}$ for $t \in [a,b]$, a list $\mathcal{S} = \{B_1 = B(0,1)\}$ with n boxes, $i = 1$

1. if $i > n$, exit, no singularity
2. compute $|\mathbf{J}(B_i)| = [u,v]$
3. if $v < 0$: $|\mathbf{J}(B_i)| < 0 \; \forall \mathbf{X} \in B_i \Rightarrow$ singularity
4. if $u > 0 : |\mathbf{J}(B_i)| > 0 \; \forall \mathbf{X} \in B_i \Rightarrow i = i + 1$, goto 1
5. $B_i = B(a,b)$, bisect $\rightarrow$ add $B(a,(a+b)/2)$, $B((a+b)/2,b)$ to $\mathcal{S}$, $n = n + 2, i = i + 1$, goto 1

Uncertainties (robot geometry, control error, ...) may be taken into account

Neural networks

Solving numerous occurrences of $\mathbf{F}(\mathbf{X}, \mathbf{\Theta}) = \mathbf{0}$ where $\mathbf{\Theta}$ has fixed value

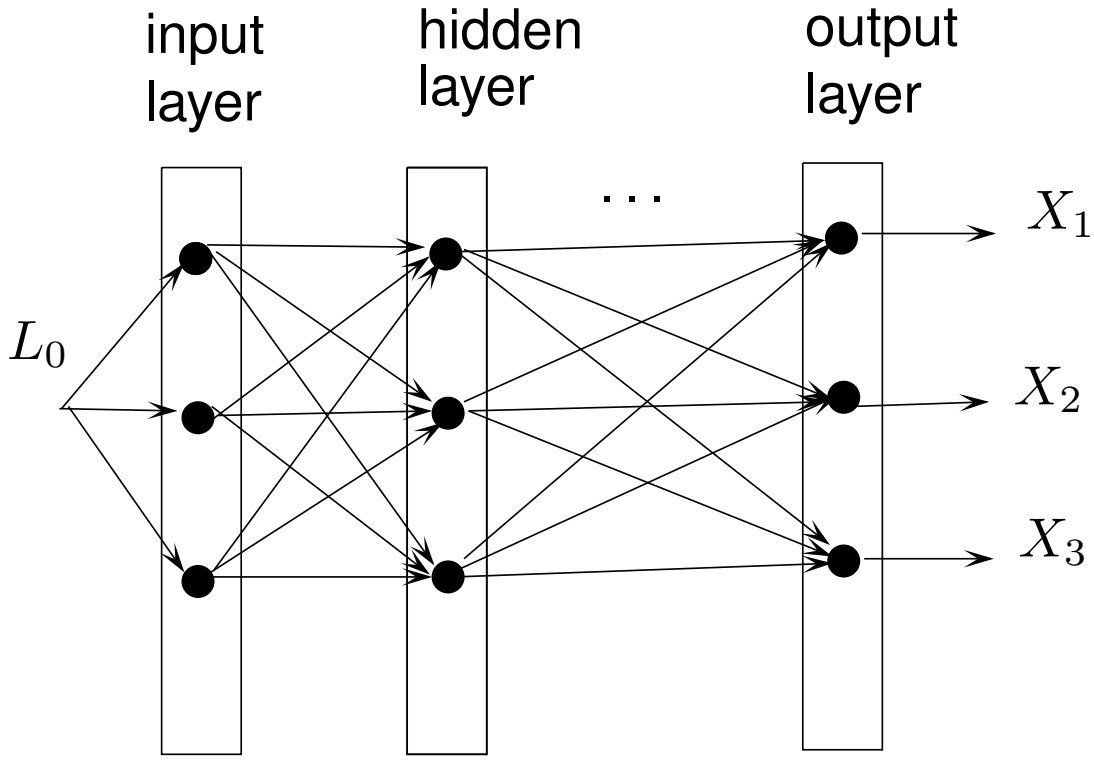
- for a given $\mathbf{\Theta}$ we may have multiple solutions
- Issue for NN: one input $\rightarrow$ one solution prediction

Advantage

- assume that for a set of $\mathbf{\Theta} = \{\mathbf{\Theta_1}, \ldots \mathbf{\Theta_k}\}$ you have obtained set of solutions $\{\{S_1^1, S_1^2, \ldots\}, \ldots, \{S_k^1, S_k^2, \ldots\}\}$
- by using continuation on $\mathbf{\Theta_u}$ along arbitrary direction(s) starting from solution S_u^i you may obtain an arbitrary large learning set

Neural networks

In spite of having a large training set the results of NN such as PINs, Multi-layer Perceptron are very poor..



number of hidden layers, of neurons, activation functions ?

Neural networks

In spite of having a large training set the results of NN such as PINs, Multi-layer Perceptron are very poor..and that's normal!

Neural networks

In spite of having a large training set the results of NN such as PINs, Multi-layer Perceptron are very poor..and that's normal!

In the learning set, because of the multiples solutions, you have samples with very similar $\mathbf{\Theta}$ but radically different solutions S_i

The purpose of training a NN is to decrease the loss

loss: some average value of the prediction error $P_i S_i$ such as:

$$MSE = \frac{\sum_{j=1}^{j=k} ||\hat{\mathbf{X}}_\mathbf{j}(\mathbf{\Theta_j}) - \mathbf{X_j}(\mathbf{\Theta_j})||^2}{k}$$

- the loss poorly reflects the extreme cases
- there will a lower bound for the loss

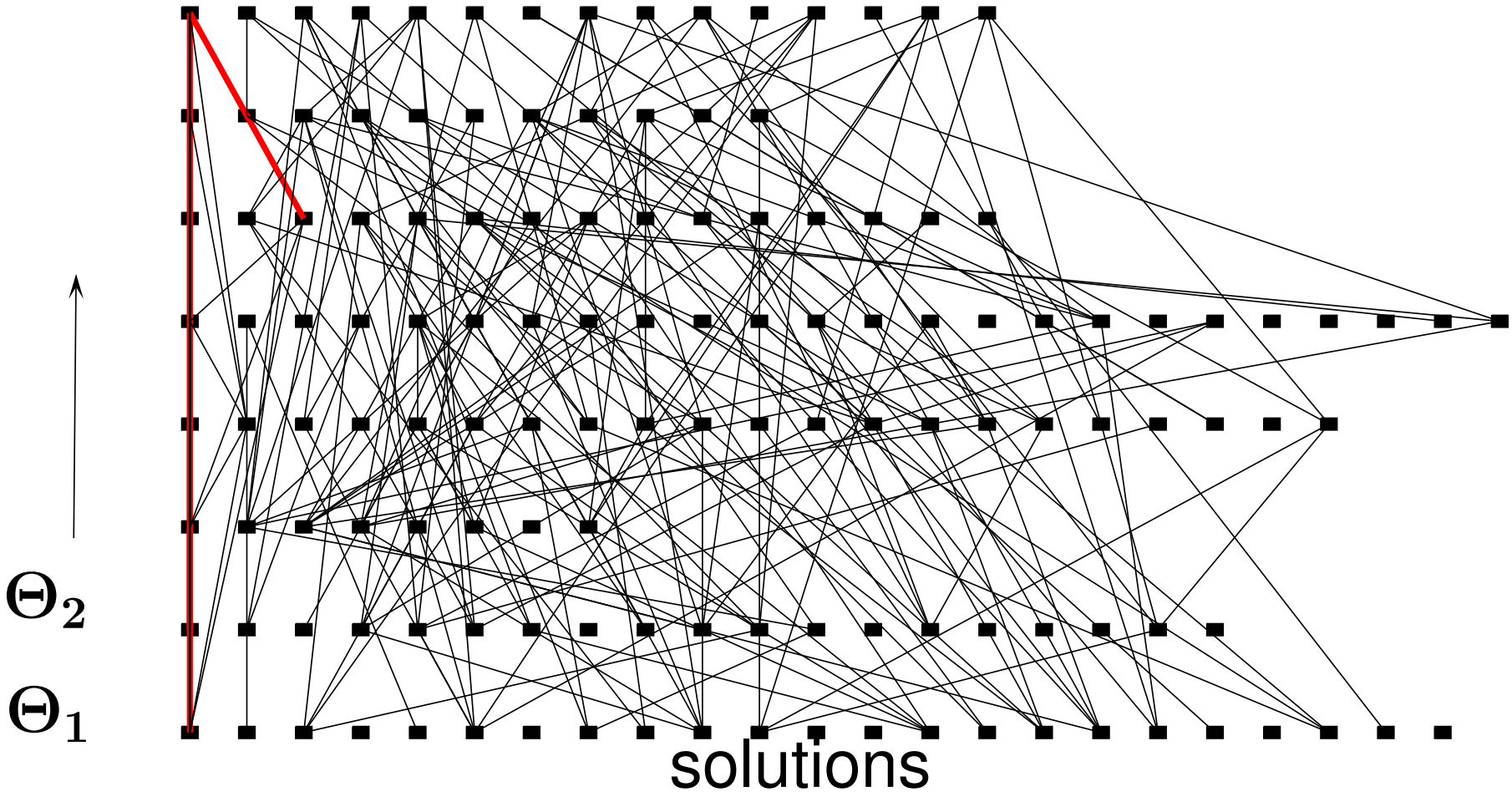
Neural networks

let us assume again that for a set of $\mathbf{\Theta} = \{\mathbf{\Theta_1}, \ldots \mathbf{\Theta_k}\}$ you have obtained set of solutions $\{\{S_1^1, S_1^2, \ldots\}, \ldots, \{S_k^1, S_k^2, \ldots\}\}$ (initial solution set)

This may be represented graphically in a plane:

- an horizontal line represents a specific $\mathbf{\Theta_i}$
- a node on a line represents a solution S_i^j for its $\mathbf{\Theta_i}$

Neural networks



Neural networks

Consider a pair $(\mathbf{\Theta_i}, S_i^j)$ and an objective $\mathbf{\Theta_k}$

- using continuation from $\mathbf{\Theta_i}$ toward $\mathbf{\Theta_k}$ we may update the solution S_i^j
- continuation may:
  - fail
  - reach a solution S_k^m already in the initial solution set
  - reach a new solution S_k^m: update the initial solution set
- when two solutions S_i^j,S_k^m are connected we draw an edge between them
- we repeat the process until all triplets $(\mathbf{\Theta_i}, S_i^j, \mathbf{\Theta_k})$ have been examined. We get the solution graph

Neural networks

In the solution graph we have:

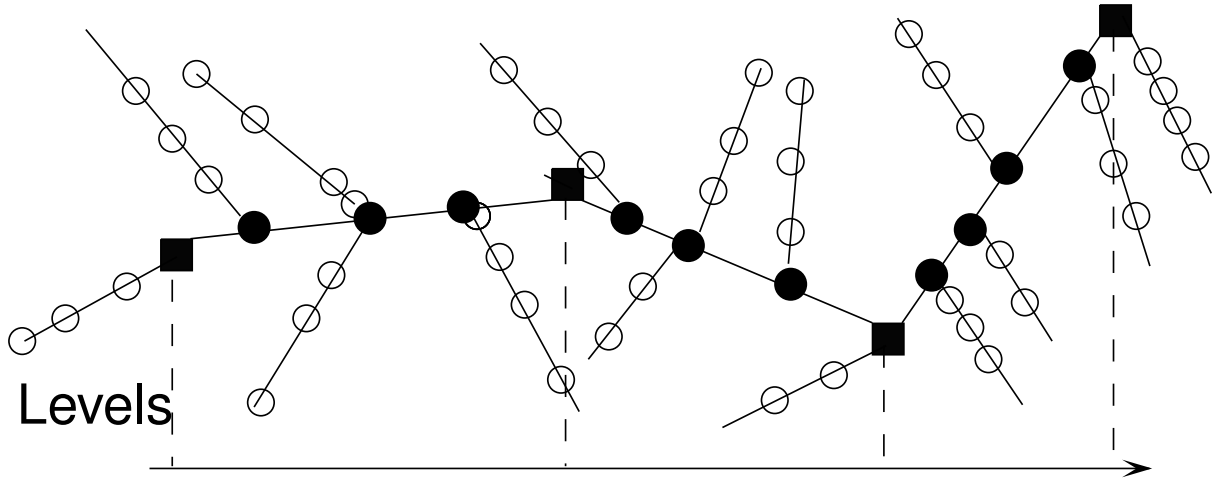
- isolated nodes: solutions not connected
- paths: a set of successively connected nodes

Branches:

- a path that do not include different solutions for the same $\mathbf{\Theta_k}$
- $\Rightarrow$ a branch includes at most n nodes of the initial solution set where n is the number of different $\mathbf{\Theta_k}$

Neural networks

- for each branch and each isolated node we create a specific learning set using continuation



- we train a specific multi-layer perceptron (MLP) for each learning set
- we have as many MLPs as branches and isolated nodes

Neural networks

- as we have several MLPs we will get different predictions for each Θ_k
- the prediction on the learning samples are still poor: we get some reasonable prediction but mostly prediction with large error (200-300%)

We need to change the training strategy:

- for improving the predictions
- to get exact solutions

Neural networks: hybridization

To get exact solutions we will use the MLP prediction as guess for the Newton method

Still even on the learning samples we get only 2% of exact solutions

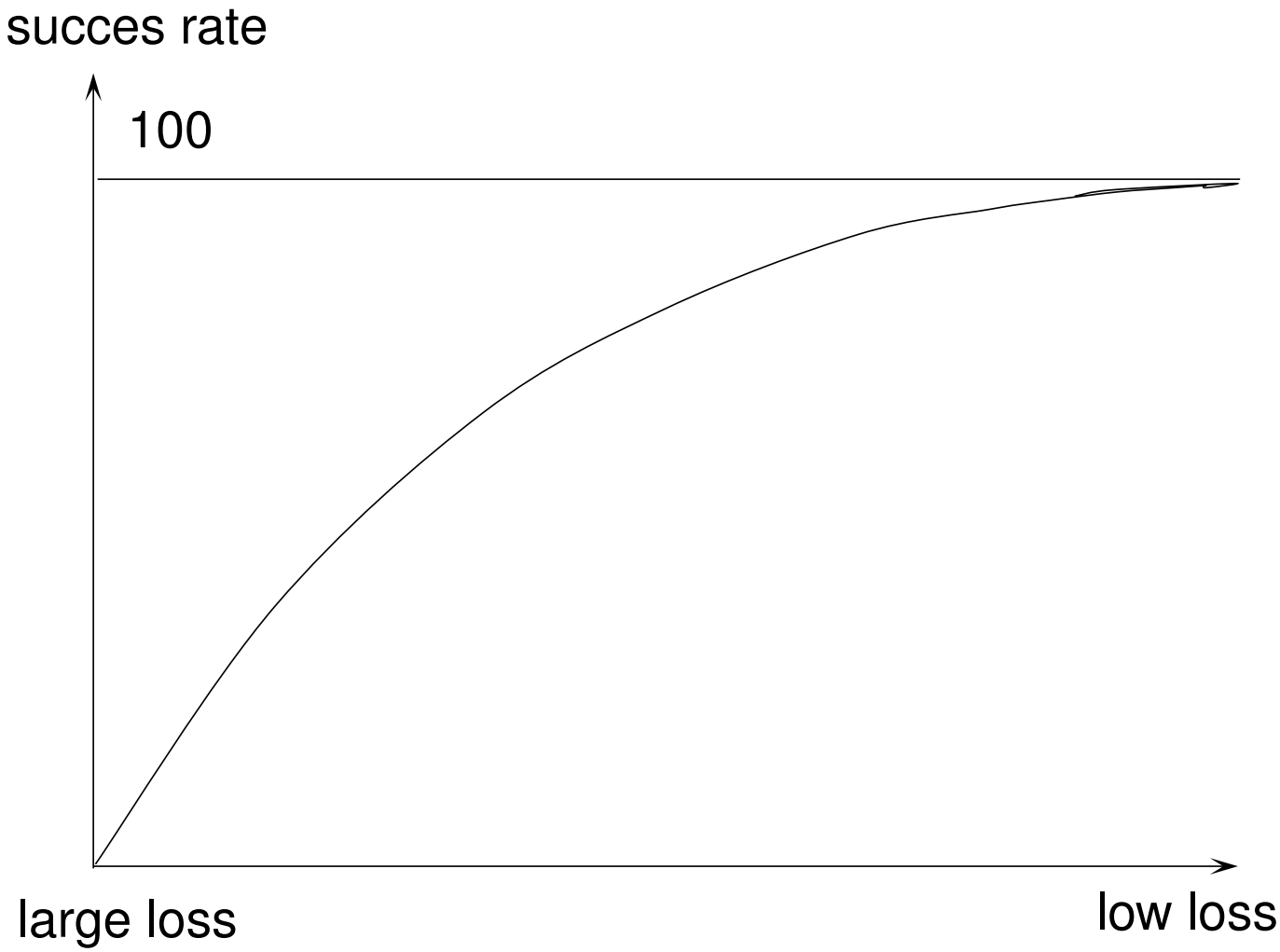
The problem is with the learning strategy that tries to minimize all parameters without taking into account their effect on Newton convergence

Neural networks: learning strategy

- success rate (SR) of a MLP: percentage of solutions found by the hybrid MLP over the samples of the learning set
- if SR is 100 the hybrid MLP provides all the exact solutions over the learning set

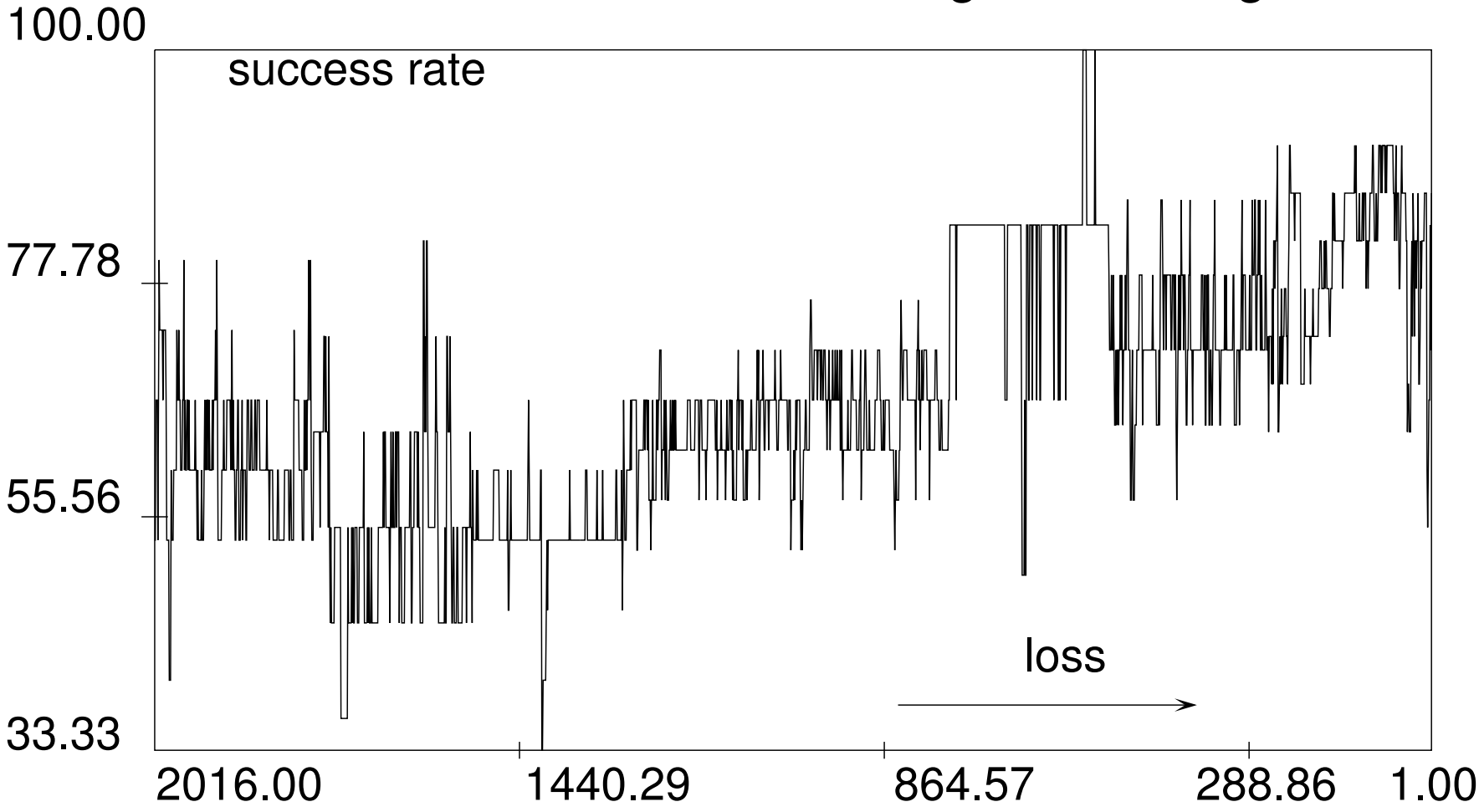
Neural networks: learning strategy

Expected success rate versus loss during the training



Neural networks: learning strategy

Measured success rate versus loss during the training



Neural networks: learning strategy

- select as MLP the one obtained during the training having SR=100 and lowest loss, if any
- if largest SR $<$ 100, select this MLP and create a new one using the learning set generated from the missed solutions
- iterate until the set of MLPs provide all solutions for all learning sets

Neural networks: solver

- solver consists in multiple hybrid MLPs that all receive the same Θ_k
- solutions from the MLPs that have Newton convergence are sorted out to obtain exact solutions

Neural networks: solver verification

We may create verification sets using continuation

- select a $\mathbf{\Theta_k}$, an arbitrary unit vector $\mathbf{v}$ in the $\mathbf{\Theta}$ space
- $\mathbf{\Theta} = \mathbf{\Theta_k} + \lambda \mathbf{v}$ and follow **all** the solutions coming from $\mathbf{\Theta_k}$, store regularly $(\mathbf{\Theta_v}, \{S_v^1, S_v^2, \ldots, S_v^m\})$, provided that $\mathbf{\Theta_v}$ is not present in the learning sets

Neural networks: solver verification

Use the verification set to test the solver. We may have

- missed solution(s)
- new solution(s)

Typically on a kinematic solver with 96 MLPs:

- we found 2 new solutions
- we miss 11 solutions among the 519 925 solutions in the verification set (coverage: 99.25%)

Neural networks: solver verification

- we use the missed solutions to create new MLPs
- note that a MLP obtained for one missed solution may provide a solution for other missed solutions
- we add the new MLP(s) to the solver

We repeat the process with another verification set.

With 106 MLPs obtained from 78 verification sets we miss 0 solution in a set of 39 millions solutions but evidently we cannot guarantee that we will always obtain all solutions

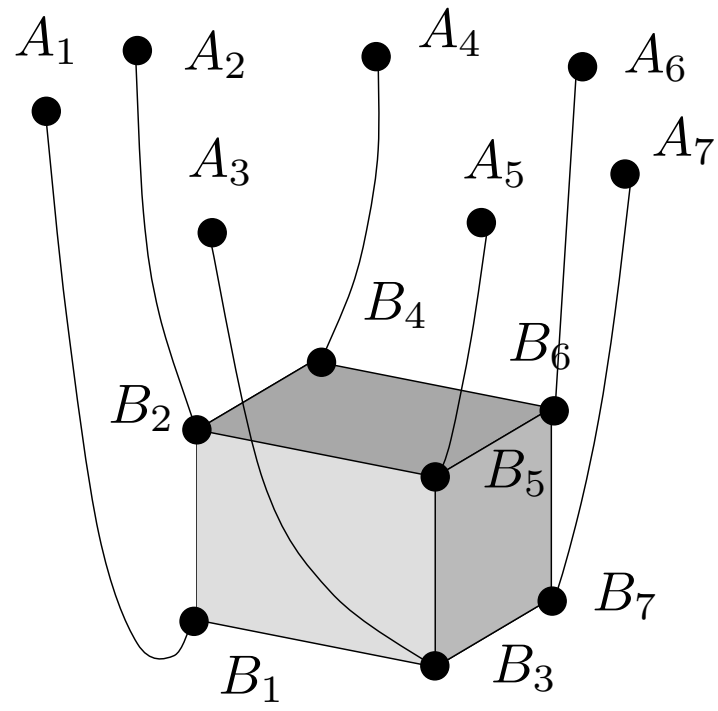
Neural networks: solver

Solver:

- may require a large training time
- but a very low solving time
- parallel implementation may drastically reduce both the training and solving time

Example

Direct kinematics of cable-driven parallel robots with sagging cables



Example

cable model

$$x_b = F_x\left(\frac{L_0}{EA_0} + \frac{sinh^{-1}\left(\frac{F_z}{F_x}\right) - sinh^{-1}\left(\frac{F_z-\mu g L_0}{F_x}\right)}{\mu g}\right)$$

$$z_b = \frac{F_z L_0}{EA_0} - \mu g L_0^2/2 + \frac{\sqrt{F_x^2+F_z^2} - \sqrt{F_x^2+(F_z-\mu g L_0)^2}}{\mu g}$$

+ 6 statics equations: 2n +6 equations, unknowns: 2n F_x, F_z+ 6 pose parameters $\rightarrow$ square system

- non-algebraic
- 8 cables, up to 31 solutions

Example

continuation

- if $E \to \infty$ and $\mu \to 0$ CDPR $\Leftrightarrow$ parallel robot
- the FK of parallel robot is a well known problem. Compute its solutions
- continuation starting with E large, μ very small going toward E_{nom}, μ_{nom}

Result

- large computation time: several hours
- but continuation very efficient for computing workspace border and singularity curves

Example

interval analysis

- the components of the pose and the F_z can easily be bounded
- only a lower bound for F_x (0) but no upper bound
- unfortunately several solution may exhibit very large F_x
- very large computation time (10 hours) or more

but

- efficient for obtaining solution with reasonable F_x

Example

neural networks

- 60 hours training time
- $<$ 0.3 s solving time

but

- parallel implementation may drastically reduce the training time for the 100+ MLPs