Tools for non polynomial kinematics: interval analysis, continuation and neural networks

J-P. Merlet
HEPHAISTOS project-team
INRIA Sophia-Antipolis
France

Up to now we have seen algebraic/polynomial kinematics

Up to now we have seen algebraic/polynomial kinematics

- but recent mechanisms impose to move away from polynomial analysis because kinematics is mixed with statics

cable robots, robot with flexible links, continuum robots, soft robots, . . .

Anyway

- at some point we have to find the solutions of a parametric system of equations

Anyway

- at some point we have to find the solutions of a parametric system of equations
- . . . all the solutions

Anyway

- at some point we have to find the solutions of a parametric system of equations
- . . . all the solutions
- exactly

Anyway

- at some point we have to find the solutions of a parametric system of equations
- . . . all the solutions
- exactly
- with a computer

Anyway

- at some point we have to find the solutions of a parametric system of equations
- . . . all the solutions
- exactly
- with a computer
- possibly not in a single case but for a large number of occurrences

Anyway

- at some point we have to find the solutions of a parametric system of equations
- . . . all the solutions
- exactly
- with a computer
- possibly not in a single case but for a large number of occurrences
- although our models are uncertain

Disclaimer: I will take some liberty with mathematical rigor for the sake of simplicity but the topics are safely mathematically grounded

Interval analysis

Interval arithmetic

mathematical interval $X = [a, b], a \leq b$: set of reals y such that $a \leq y \leq b$

Interval arithmetic

mathematical interval $X = [a, b], a \leq b$: set of reals y such that $a \leq y \leq b$

computer interval

$X = [a, b]$:

set of floating point numbers y such that $a_r \leq y \leq b_r$ with

- a_r: the floating point number closest to a and lower or equal to a
- b_r: the floating point number closest to b and greater or equal to b

Interval arithmetic

You may think that $a - a_r$, $b_r - b$ is small and significant only for large numbers

Interval arithmetic

You may think that $a - a_r$, $b_r - b$ is small and significant only for large numbers

wrong!

- mathematical interval: [0.1,0.1]
- computer interval: [0.099999994039536,0.100000008940697]

Interval arithmetic

You may think that $a - a_r$, $b_r - b$ is small and significant only for large numbers

wrong!

- mathematical interval: [0.1,0.1]
- computer interval: [0.099999994039536,0.100000008940697]

numerical round-off errors may play an important role in computation

Don't always believe you computer . . . and Matlab !

Interval evaluation

Interval evaluation

- let a function $f(x_1, x_2, \ldots, x_n)$ from $\mathcal{R}^n$ to $\mathcal{R}$

Interval evaluation

- let a function $f(x_1, x_2, \ldots, x_n)$ from $\mathcal{R}^n$ to $\mathcal{R}$
- all x_i have an interval value X_i

interval evaluation F of $f = [A, B]$

$$\forall x_1 \in X_1, \ldots x_n \in X_n \quad A \leq f(x_1, \ldots, x_n) \leq B$$

Interval evaluation

How do we obtain an interval evaluation ?

the simplest way: the natural interval evaluation

$\rightarrow$ substitute any operator by its interval equivalent

and we have such equivalent for all major operators:

$+, -, *, \sin, \cos, exp, log, \sinh, \cosh, \ldots$

Interval evaluation

Example:

$f(x) = x + \sin(x)$ pour $x \in [1.1, 2]$

$$F([1.1, 2]) = [1.1, 2] + \sin([1.1, 2]) = [1.1, 2] + [0.8912, 1] = [1.9912, 3]$$

numerical round-off errors can be taken into account: certified result

Interesting properties for problem solving

Interesting properties for problem solving

- **for equation solving**: if the interval evaluation of a function over a set of intervals does not include 0, then the function has no root in the set

Interesting properties for problem solving

- **for equation solving**: if the interval evaluation of a function over a set of intervals does not include 0, then the function has no root in the set
- **for managing inequalities**: if $F(X) = [a, b]$ is such that $b < 0$, then $F(X) < 0 \ \ \forall x \ \in \ X$

Interesting properties for problem solving

- **for equation solving**: if the interval evaluation of a function over a set of intervals does not include 0, then the function has no root in the set
- **for managing inequalities**: if $F(X) = [a, b]$ is such that $b < 0$, then $F(X) < 0 \ \ \forall x \ \in \ X$
- **for optimization**: the global extremum of a function over a given range is included in its interval evaluation

IA solver

IA solver

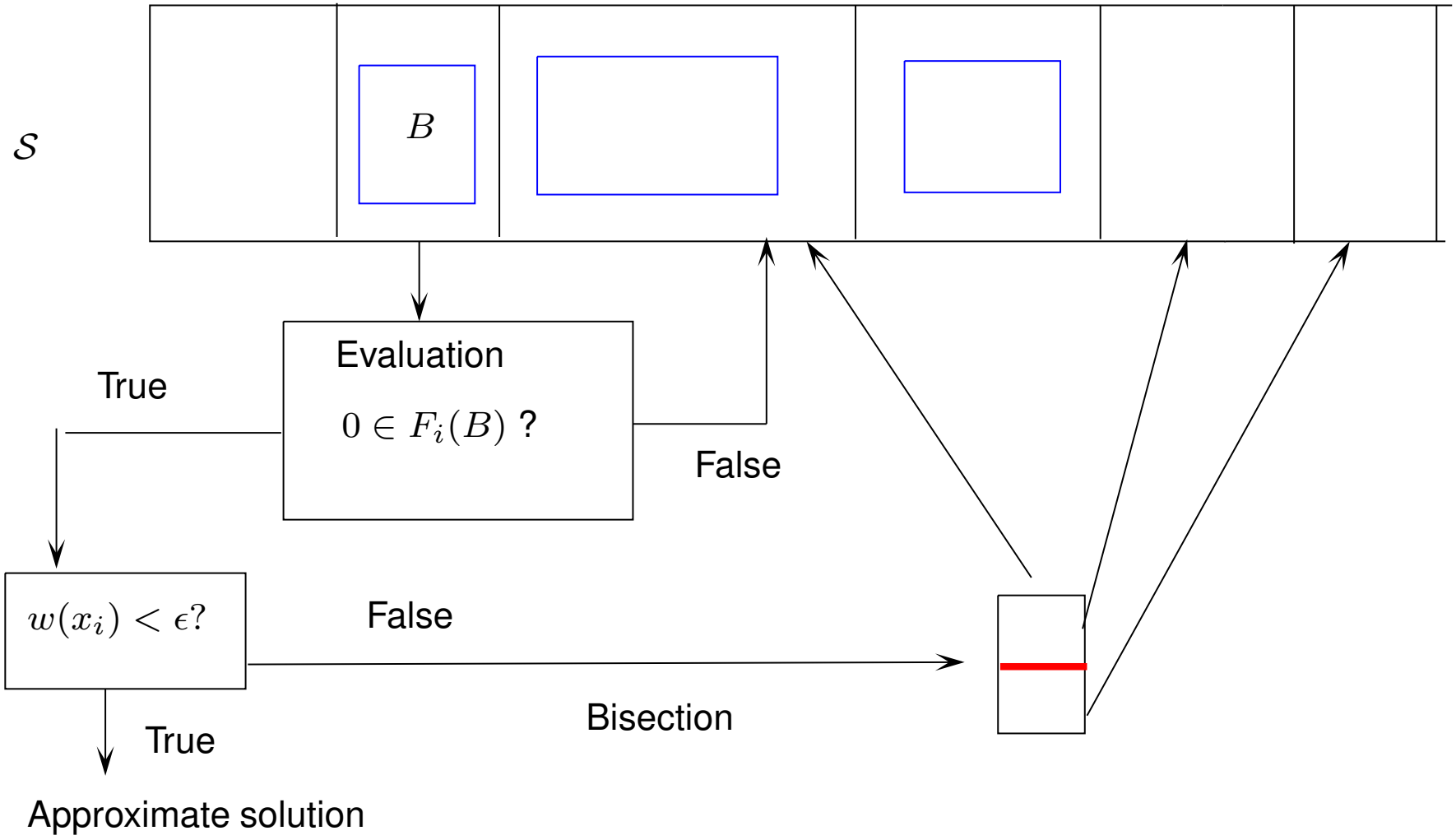
Square equations system solver, looking for solutions within a bounded domain $\mathcal{D}$ defined by intervals for each unknowns

$$\mathbf{F}(x_1, x_2, \ldots x_n) = \mathbf{0}$$

Tools:

- width of an interval $[a, b] = b - a$
- box: domain defined by intervals for each of the x_i
- bisection: splitting a box in two by splitting the interval of one x_i in two
- $\mathcal{S}$: a list of boxes, initialized with $\mathcal{D}$

IA solver


$\mathcal{S}$
B
Evaluation
$0 \in F_i(B)$?
True
False
$w(x_i) < \epsilon$?
False
Bisection
True
Approximate solution

IA solver

Numerous tricks to improve this basic algorithm

- Kantorovitch theorem to determine if a box contains a single solution and allow to compute this solution with an arbitrary accuracy
- filters: heuristics which allow to decrease the size of B without bisection
- $\mathcal{S}$ management: the box resulting from a bisection are put on top of the list

IA solver

success story: quantum mechanics, Kochen–Specker (KS) theorem,

- solving systems of 200/400 quadratic equations (out of reach of AG)
- 144 millions of such a system solved in about one month

IA is also a precious tool to manage uncertainties e.g in the physical parameters of the mechanism (usually having a bounded error)

$\rightarrow$ worst case analysis which is good for safety critical systems

Continuation

Continuation

A numerical method that is very useful in kinematics

Example : solving square system of equations

- you want to solve a square system $\mathbf{F}(\mathbf{X}) = \mathbf{0}$ that may have multiple solutions

Continuation

A numerical method that is very useful in kinematics

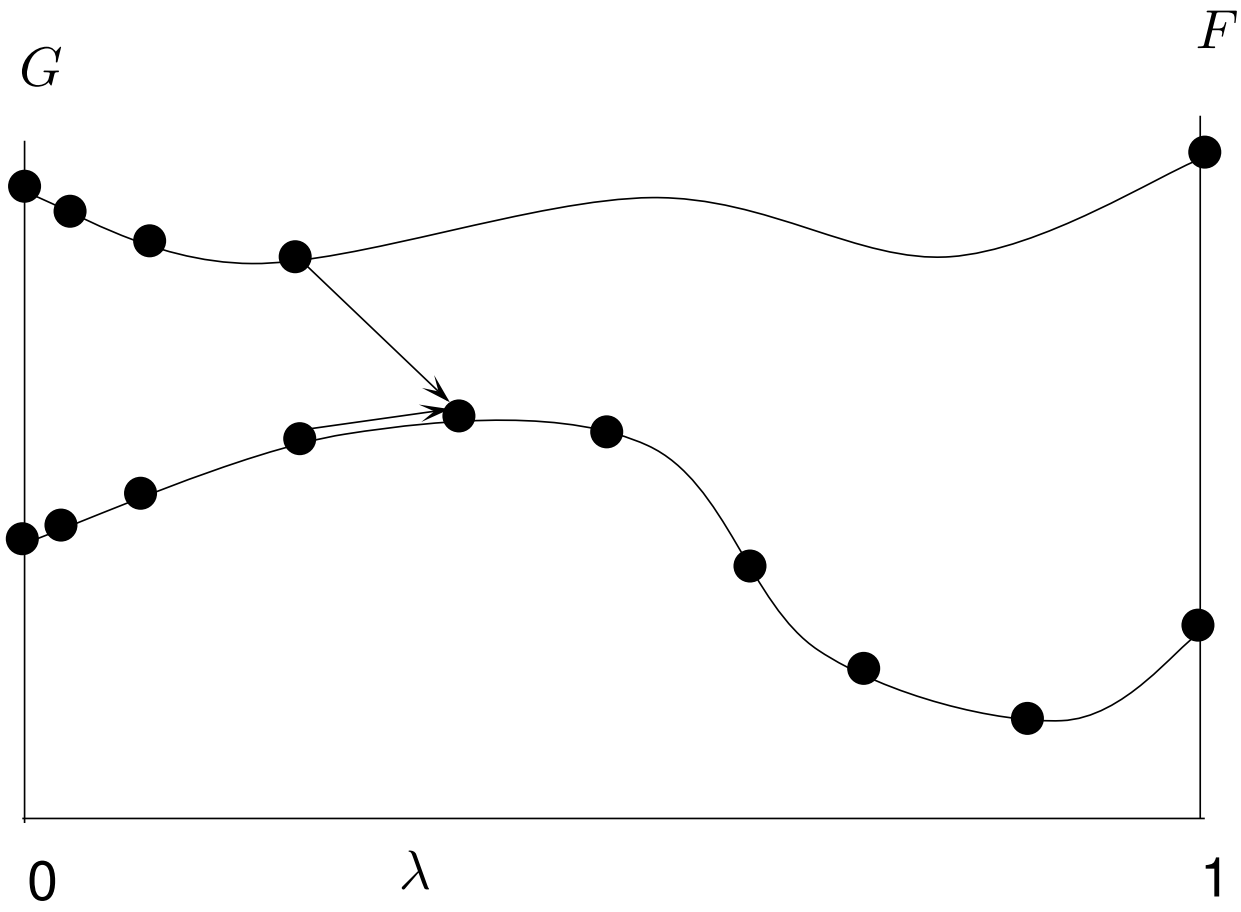
Example: solving a square system of equations

- you want to solve a square system $\mathbf{F}(\mathbf{X}) = \mathbf{0}$ that may have multiple solutions
- you know a parametric system $\mathbf{G}(\lambda, \mathbf{X})$ such that
  - you have all solutions $\{S_{0_1}, S_{0_2}, \ldots, S_{0_m}\}$ of $\mathbf{G}(0, \mathbf{X}) = \mathbf{0}$
  - you have $\mathbf{G}(1, \mathbf{X}) = \mathbf{F}(\mathbf{X})$

  λ is the continuation parameter

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

Continuation



Continuation

Issue

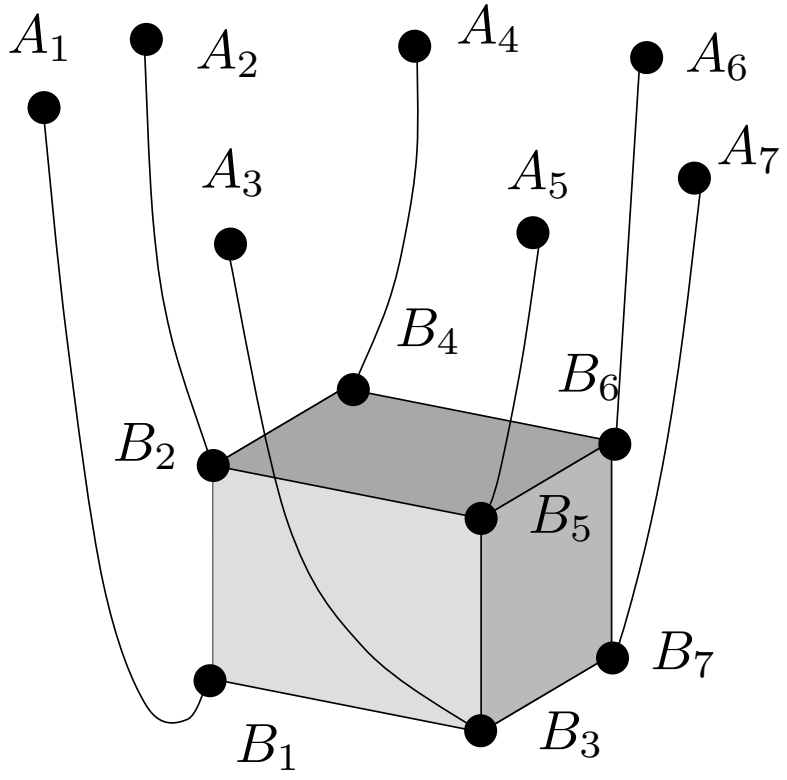
- finding the system G

Continuation

Issue

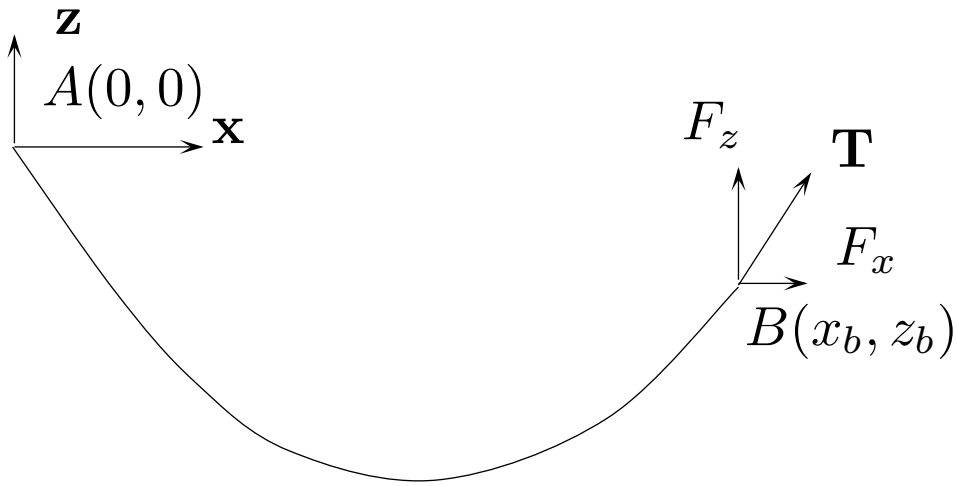
- finding the system $\mathbf{G}$

non-polynomial: direct kinematics of CDPR with sagging cables



Continuation

non-polynomial: direct kinematics of CDPR with sagging cables



Irvine cable model

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

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

Continuation

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

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

If $E \to \infty$ and $\mu \to 0$, then cable =rigid leg if taut: ideal cable

DK for a CDPR with n cables

- equations: $2n$ (Irvine) +6 (equilibrium) $\to$ 2n+6
- unknowns: pose $\mathbf{X}$ (6) + $2n F_x, F_z \to$ 2n+6

$n = 8 \to$ 22 equations

Continuation

DK solving, first step: ideal cables

- find the DK solutions for the same robot with rigid legs,
- but you have to consider that leg tensions may be 0 and the 0-tension legs may be ignored
- $\Rightarrow$ for a robot with n cables you have to consider the DK for all combinations of CDPR with 1 to n taut legs
- for $n = 8$ cables there will be at most 33363 solutions

Continuation

DK solving, second step, continuation on E, μ

- consider one of the solution obtained for the rigid legs

Continuation

DK solving, second step, continuation on E, μ

- consider one of the solution obtained for the rigid legs
- assume E very large and μ very small

Continuation

DK solving, second step, continuation on E, μ

- consider one of the solution obtained for the rigid legs
- assume E very large and μ very small
- use Newton to obtain a solution for the CDPR

Continuation

DK solving, second step, continuation on E, μ

- consider one of the solution obtained for the rigid legs
- assume E very large and μ very small
- use Newton to obtain a solution for the CDPR
- then use continuation on the E, μ toward the nominal E, μ: if they are reached you get a DK solution for your CDPR
- Drawbacks:
  - solving time: several hours
  - all solutions ? not sure

Continuation

Issue

- singularities: isolated, non isolated, . . .

Continuation

singularities: isolated, non isolated, . . .

We have considered continuation in the real domain but continuation may also be used in the complex domain

Baskar, A., "A numerical continuation approach using monodromy to solve the forward kinematics of cable-driven parallel robots with sagging cables", Mechanism and Machine Theory, May 2024, Vol 195

---

still

- solving time: several hours
- solution(s) may be missed

Neural networks

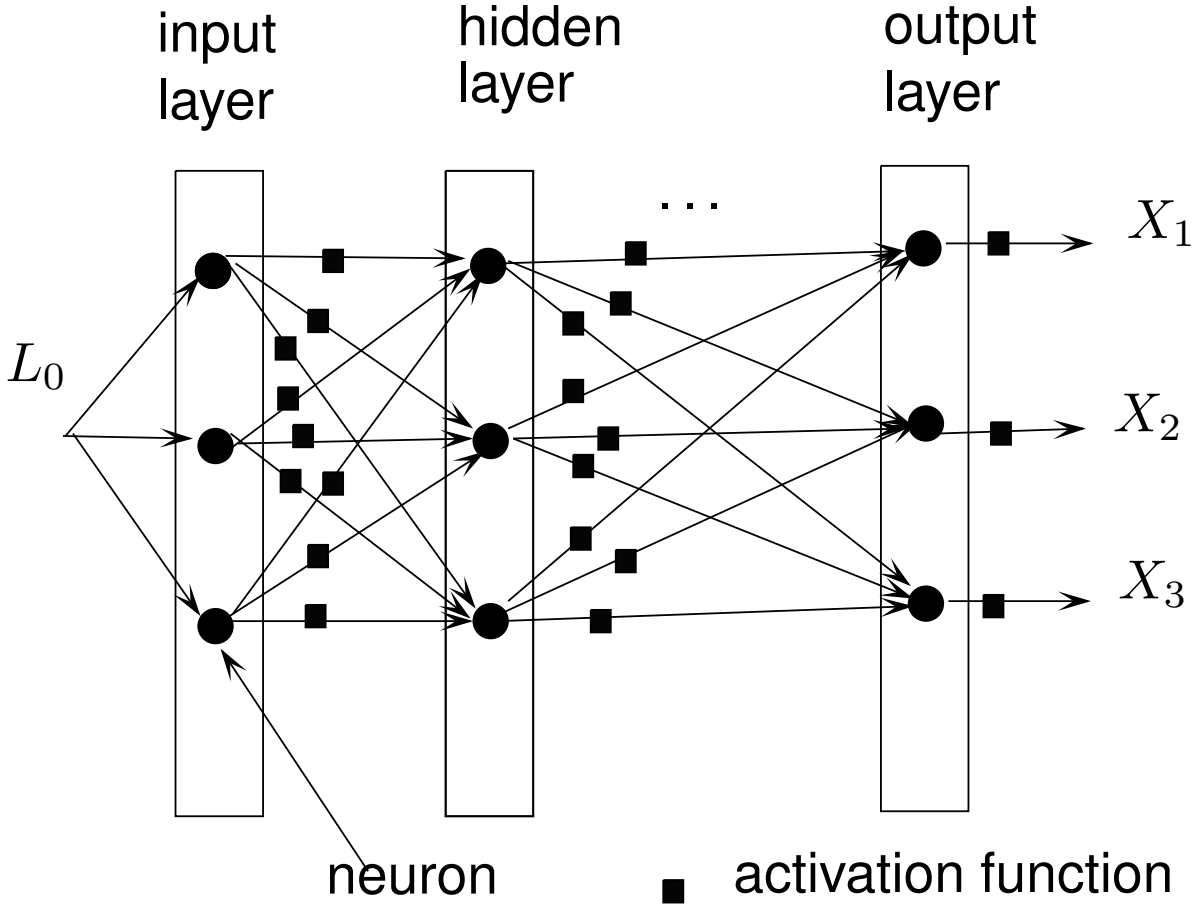
Neural network and kinematics

Neural network is a jungle:

- multiple types, most of which are dedicated to specific tasks and are not really easy to implement . . . and to use
- curiously very few works dedicated to a basic problem in science: finding all solutions of a square system of non-differential equations . . .

Neural network, the MLP

Multi Layer Perceptron, MLP



Neuron j output=Activation $(\sum a_{ij} Input_i + b_{ij})$

MLP

- An MLP is a supervised neural network. It requires a learning set i.e. a significant number m of samples (L_0, S)

MLP

- An MLP is a supervised neural network. It requires a learning set i.e. a significant number m of samples (L_0, S)
- a stochastic optimizer will try to determine the a_{ji}, b_{ij} that minimize a **cost function** called the loss function

MLP

- An MLP is a supervised neural network. It requires a learning set i.e. a significant number m of samples (L_0, S)
- a stochastic optimizer will try to determine the a_{ji}, b_{ij} that minimize a **cost function** called the loss function
- the loss function is usually a statistical function on the error between the exact S and the prediction P over the whole learning set (e.g. the MSE=$(\sum_{j=1}^{j=m} ||P_j - S_j||^2)/m$)

MLP issues

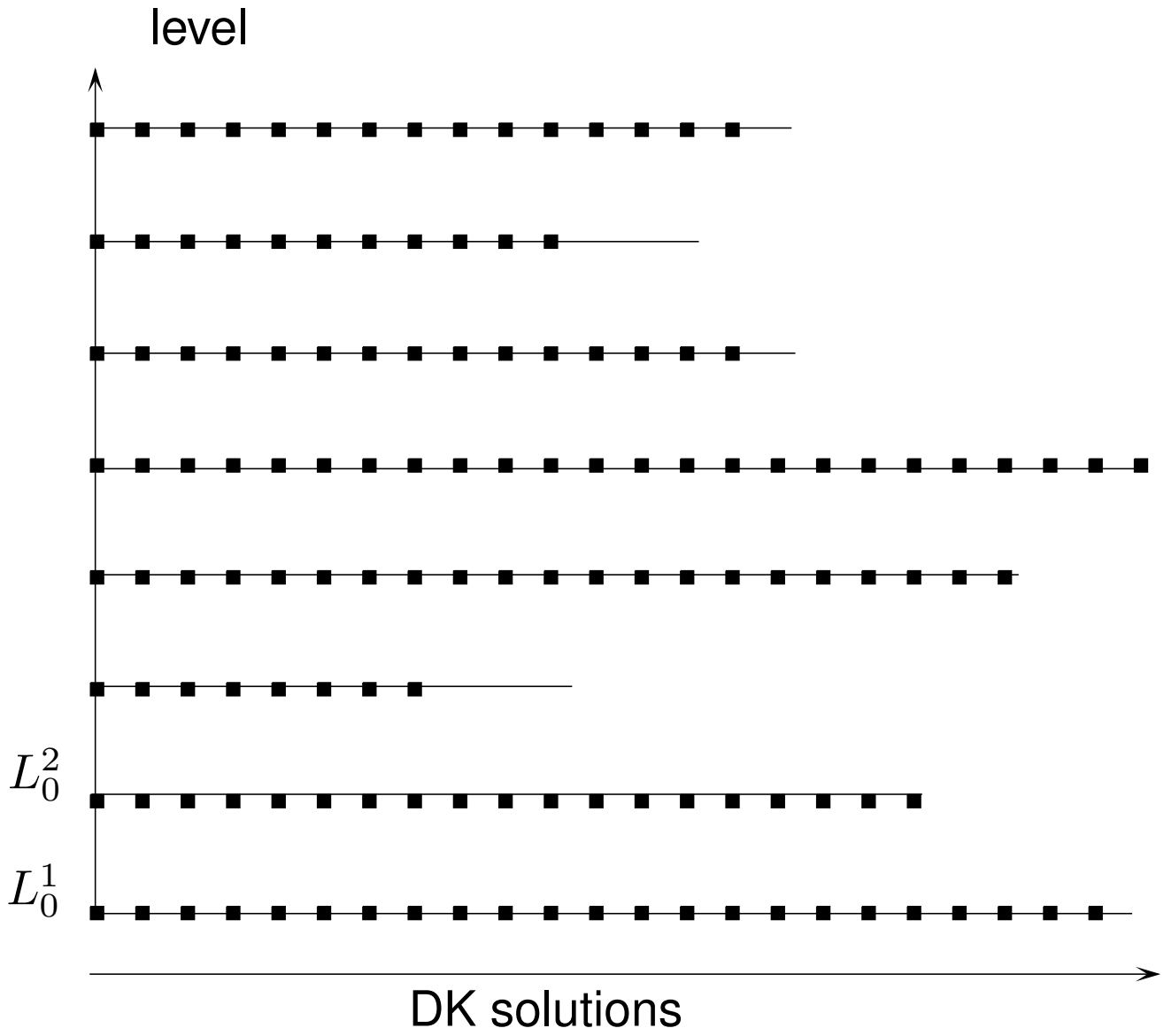
- how to build learning set ?
- no rules to choose the number of layers, neurons, activation function, learning rate
- learning time ?
- a MLP provides only a single solution, how to find multiples solutions ?

Building learning sets for the CDPR DK

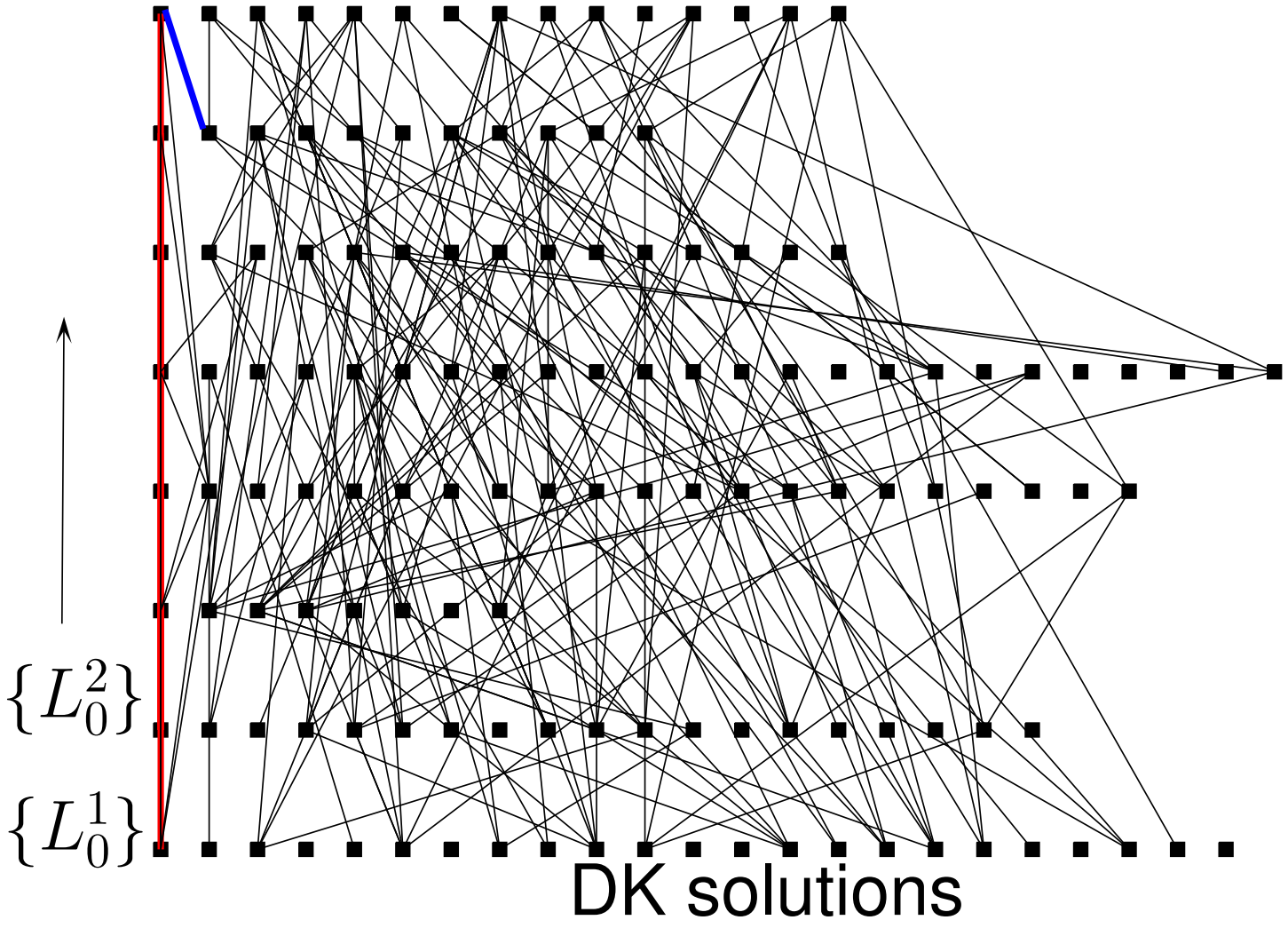
Starting point:

- we consider 8 different sets L_0^i of 8 $L_0 = \{\mathcal{L}_{0_1}^1, \ldots \mathcal{L}_{0_8}^8\}$
- we calculate with IA, continuation the DK solutions $\{\mathcal{S}_i\}$ for each set (possibly missing solution(s))

Building learning sets for the CDPR DK



Building learning sets for the CDPR DK



Building learning sets for the CDPR DK

- for a given $\mathcal{L}_i$ we may have isolated solutions that are not connected to any other solution among the remaining $\mathcal{L}_j$

Building learning sets for the CDPR DK

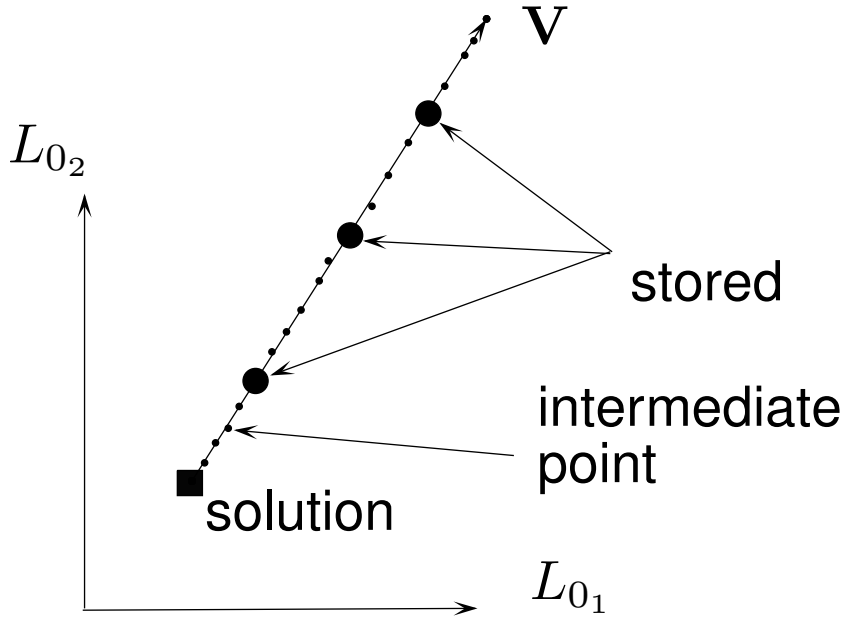
- for a given $\mathcal{L}_i$ we may have isolated solutions that are not connected to any other solution among the remaining $\mathcal{L}_j$
- we have paths connecting solutions in the graph

Kinematic branches: paths that do not include different DK solutions for a given $\mathcal{L}_i$

Building learning sets

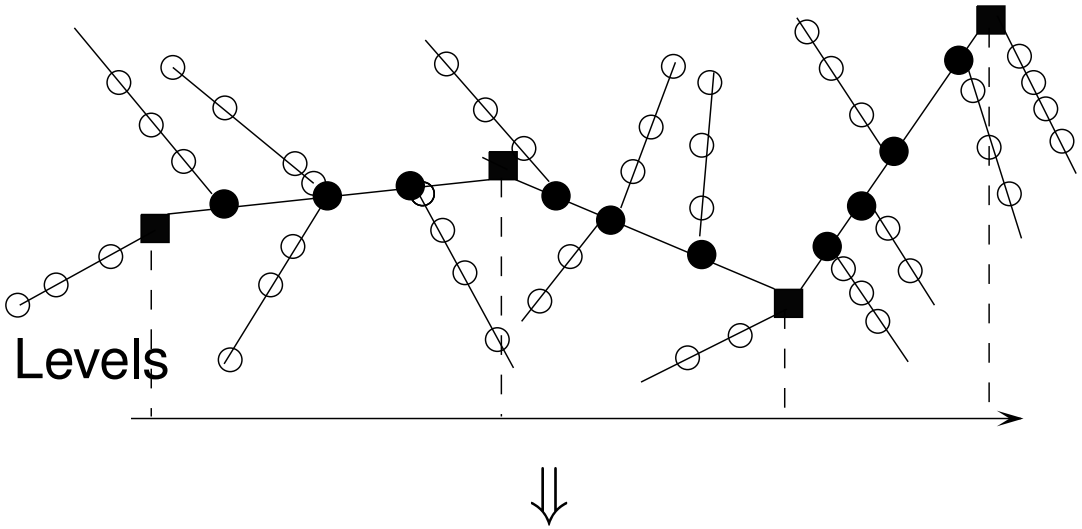
A learning set is a set of L_0 with one of its DK solution

creating learning set from an isolated solutions



Building learning sets

creating learning set from a kinematic branch



- we get multiple learning sets
- $\Rightarrow$ multiples MLPs $\Rightarrow$ potentially different solutions for the DK

Training

the training time is low (a few minutes in the worst case)

$$\Downarrow$$

systematic testing is possible

- we build all MLPs with all combinations of [1,. . . ,20] layers, [2,. . . ,200] neurons, activation function chosen among a set of 5 usual one
- for each of them we test the prediction error on the learning set, the loss function reaches quickly a relatively small value

Training

- we rank the trained MLPs by increasing value of the final loss
- we consider the ten best MLPs for each learning set

Training

- we rank the trained MLPs by increasing value of the final loss
- we consider the ten best MLPs for each learning set

Quality of prediction is poor even if the loss is low (for some samples 200-300% error for some unknowns) !

Training

- we rank the trained MLPs by increasing value of the final loss
- we consider the ten best MLPs for each learning set

Quality of prediction is poor even if the loss is low (for some samples 200-300% error for some unknowns) !

Good point (in modern time...): you can find a sample for which the error is reasonably small, write a paper about it, knowing that the result is stochastic and nobody will be able to reproduce it !

Improving the MLP, step 1

Hybridization at the end: use Newton method with the final MLP prediction as guess

- if convergent we get an exact solution
- but possibly not the expected one

Improving the MLP, step 2

Hybridization during the training

- during the training when the loss decreases we save the current MLP

Improving the MLP, step 2

Hybridization during the training

- during the training when the loss decreases we save the current MLP
- for this MLP we compute over all n samples of the learning set how many time C the Newton scheme converges toward the expected solution

success rate: $S = \frac{100C}{n}$

Improving the MLP, step 2

Hybridization during the training

MLP created during the training, success rate: $S = \frac{100C}{n}$

- if $S = 100$ we have an optimal MLP which leads to obtaining the exact solution for each sample of the learning set: stop the training

Improving the MLP, step 2

Hybridization during the training

MLP created during the training, success rate: $S = \frac{100C}{n}$

- if $S = 100$ we have an optimal MLP which leads to obtaining the exact solution for each sample of the learning set: stop the training
- if $S = 100$ is not reached just save the MLP with largest S and create new learning sets from the missed solutions

Improving the MLP, step 2

- NO: S does not always increase when the loss decreases !

Improving the MLP, step 2

- NO: S does not always increase when the loss decreases !
- YES: S may be =100 although the loss is not that low !

Improving the MLP, step 2

- NO: S does not always increase when the loss decreases !
- YES: S may be =100 although the loss is not that low !
- YES: S may be $\neq 100$ for the MLP with the minimal loss although it may have been 100 for a previous MLP !

Improving the MLP, step 2

We stop the training when all the created MLPs allow to find **all** solutions for **all** samples of **all** learning sets: **full coverage**

Solver: run the hybrid version for each existing MLPs

Verification set

Verification set: set of L_0 together with their DK solutions

Using the same learning set creation principle but following all solutions we can create **verification sets**

Verification set

- we use a different seed for the random process to ensure that the L_0 of the samples are different from the one of the learning sets

Verification set

- if solutions are missed we generate new learning set for creating new MLps to obtain full coverage
- note that the MLPs may be redundant but we can trim them to reduce their number

DK results

- after testing 77 verification sets we end up with 106 MLPs
- solving time for a given set of L_0: 0.022 seconds !
- training time: about 70 hours

But

- current implementation mixing C and Python: not optimal
- both the solving time and the training time may be drastically reduced by a parallel implementation

J-P. Merlet, Mixing neural networks, continuation and symbolic computation to solve parametric systems of non linear equations, Neural Networks, 2024

Results and perspectives

Results

- IK problem of a 4-cables CDPR (optimization, constant or variable platform mass), DK of a 8-cables CDPR, IK of a 2 dof continuum robot with flexible rods
- failure: DK of the 8-cables CDPR with variable mass for the platform

AI is a complementary tool to existing one:

- but not a black box that will provide automatically all answers, even approximate
- AI is a stochastic process $\rightarrow$ possibly non reproducible on a single instance

Results and perspectives

Perspectives:

- systems mixing equations and ODE (continuum and soft robots)
- mixing IA and AI: AI provides a solution for an ideal robot and IA manage locally uncertainties. For example
  - managing cable tensions τ for a CDPR
  - for a given set of L_0 (uncertain) of a given CDPR (uncertain) will we always have $\tau \leq \tau_{max}$?