A parser for the interval evaluation of analytical functions and its application to engineering problems

J-P. Merlet
INRIA Sophia-Antipolis
France

Abstract: We have developed a *parser* which takes as input a file containing the analytical expression of one or more formulas and ranges for each unknown in the formula and return an interval evaluation of the formula. We describe the use of this parser for solving robotics problems and, in a more general context, for analyzing and solving systems of equations.

1 Introduction

Interval analysis is a well known method for computing bounds of a function, being given bounds on the unknowns appearing in this function [2, 6]. Such bounds on the unknowns are frequently available in engineering as these unknowns represent physical, geometrical quantities which are subject to constraints. There are however some drawbacks of interval analysis. One of the main drawbacks is that the bounds obtained for a function are heavily dependent on the way the function is written. For example the interval evaluation of the function $x^2 + 2x$ may be quite different if we use one of the following forms:

$$x^2 + 2x \tag{1}$$

$$x(x + 2) \tag{2}$$

$$(x + 1)^2 - 1 \tag{3}$$

For example if x lies in the range [-1,1] the first form leads to the range [-2,3], the second form to [-3,3] and the third to [-1,3] (which is sharp). For more complex functions there is no known method which enables one to determine the form of the function that will lead to the best bound, being given ranges for the unknowns. As the sharpness of the function evaluation will clearly have a large influence on the efficiency of any algorithm based on interval analysis it will be quite useful to be able to change at will the analytical form that will be used (for example by simply editing a file which contains the analytical form of the functions) without having to go through a compilation phase that may be time intensive: this is exactly what we provide with the parser.

A second drawback lies in the difficulty to test rapidly if an algorithm based on interval analysis will be efficient. Here three levels of software must be distinguished:

1. *a basic level* where the basic operations of interval arithmetic is performed
2. *an end-user level* where the bounds for the function to be evaluated will be computed, using the procedures of level 1
3. finally *an algorithm level* where the function evaluation of level 2 will be used to solve the problem

For level 1 there are numerous packages that implement the basic operations of interval arithmetic using various data types or even generic data type as in C++. To implement level 2 it will first be necessary for an end-user to have a minimal knowledge of the procedures of level 1 (the minimal requirement being to be able to write the functions in a way that is compatible with these procedures). Furthermore if the end-user wants to change the analytical form of his functions at level 2 (for example using form (1) or (2) or (3) for the function $x^2 + 2x$) with the aim of improving the computation time, it will be necessary to modify the program and compile it for each change, thereby inducing a large development time.

We will also see in the examples (see section 3) that the functions that have to be evaluated at level 2 may not be known beforehand (for example they are obtained as the result of a symbolic computation) while the algorithm at level 3 may be the same for all functions. The traditional approach will be first to compute the symbolic form of the functions, then, second, to transform this form into a source code in conformity with the procedures used at level 1 and finally to compile the resulting program. Using the parser the second and third steps of this approach are not necessary as the analytical form obtained at step 1 is sufficient to obtain the function evaluation at level 2. This allows a high flexibility while reducing the development time.

The purpose of our parser is to enable the user to focus on the algorithmic part of the problem, while offering a convenient way to use interval analysis and, furthermore, enabling one to change easily the analytical form of the function that will be evaluated.

2 The parser

2.1 Principle

Our parser takes as input:

- the name of a file, called the *formula file*, that contains the analytical form of the function(s) that have to be evaluated
- the ranges for each unknowns appearing in the function(s)

and it will return the interval evaluation of the function(s). An example of a formula file is:

```
eq=(y^2-1.2341234567)*z+(2.34567899*y*t-2)*x
eq=2+(-10*t+(-10+2*y*t)*y)*y+((4+4*y^2)*z+(4+4*y*t-x*z)*x)*x
eq=(2*y*t-2)*z+(t^2-1)*x
eq=2+(4*x+(4-x*z)*z)*z+((-10+4*z^2)*y+(-10+4*x*z+2*y*t)*t)*t
```

In this file we have defined 4 functions in the unknowns `x,y,z,t`, which are all prefixed by `eq=`. The parser is written using the lexicographic analyzer `lex` and the semantic analyzer `yacc` and is able to evaluates formula which use the basic arithmetic operations `+,-,*,/` and the power operator `**` together with most classical mathematical functions (trigonometric, log, exponential, hyperbolic functions) that are defined using their equivalent MAPLE name. The parser is also able to recognize ranges which are written using MAPLE syntax (e.g. `INTERVAL(0.1..0.2)` for the range [0.1,0.2]). Basic interval operations are done using the interval analysis package `BIAS/Profil`[1]. The unknowns are defined through an *unknown name*

[1] http://www.ti3.tu-harburg.de/Software/PROFILEnglisch.html

array: as soon as an unknown symbol is found (i.e. a string which does not correspond to an operator) the parser looks at the unknown name array and if an unknown name matches the symbol it is replaced by the range of the unknown.

We use also another name array: the *parameter name* array. To this array is associated an array of `double` which contains the fixed values of parameters that may be used in a formula. For example the formula:

```
eq=(y^2-1.2341234567)*z+(2.34567899*y*t-2)*x
```

may also be written as:

```
eq=(y^2-_IA0)*z+(_IA1*y*t-2)*x
```

where `_IA0, _IA1` will be present in the parameter name array and associated with the values 1.2341234567 and 2.34567899. This enables us first to speed up the evaluation of the formula. Indeed, in the first form the parser will consider the string "2.34567899", recognize that this string represents a number and will convert it to a double using the function `atof`. In the second form the parser read less characters and does not need to perform a conversion from a string to a double. Another role of the parameter name array is to enable one to deal with parametric systems: the formulas are expressed using symbolic parameter names like `_IA0, _IA1` which values are defined in a *parameters file*. Hence we may deal with any instance of a class of problem just by changing the values in this file, without any modification of the formula file. A similar array, the *interval name* array, may be used in the case where the parameters in the formula file are intervals.

The parser is also able to deal with intermediate variables, which may be used, for example, to decrease the evaluation time. For example, in the formula:

```
eq=2+(-10*t+(-10+2*y*t)*y)*y+((4+4*y^2)*z+(4+4*y*t-x*z)*x)*x
```

the term `y*t` appears twice. To avoid unnecessary computation it is interesting to write this formula as:

```
_XA0= y*t ;
eq=2+(-10*t+(-10+2*_XA0)*y)*y+((4+4*y^2)*z+(4+4*_XA0-x*z)*x)*x
```

Creating a formula file may be done by using a text editor or by using a specific MAPLE package that produces the formula file, but also performs some heuristic simplifications on the function in order to improve both the evaluation and its computation time. Hence the main effort that has to be done for using the parser is to write the necessary equations and then to call a specific C++ procedure:

```
int Evaluate_With_Parser(char *file_name, int nb_eq, INTERVAL_VECTOR
                     &Unknowns, INTERVAL_VECTOR &Value)
```

which returns in `Value` the interval evaluation of the `nb_eq` equations written in the file `file_name`. The procedure returns 1 if the formulas are syntaxicaly correct.

2.2 Improving the evaluation

Interval evaluation of an equation may be improved by using the monotonicity of the equation. Indeed if the derivative of the equation with respect to one variable has a constant sign, then the upper and lower bound of the evaluation are obtained by substituting for this variable the lower or upper bound of its range.

To check the monotonicity of a function we may compute an interval evaluation of the gradient of the equation. It must be noted that monotonicity checking must be implemented as a recursive process. Indeed consider the example where we have an equation F in n unknowns $x_1, x_2, \ldots, x_n$. At the first step the gradient is evaluated with the unknowns being all intervals ($x_1 = [\underline{x_1}, \overline{x_1}]$ and so on). If the equation is monotonic , say increasing, with respect to the variable x_1, then the lower bound of the function will be obtained by computing the interval evaluation of $F(\underline{x_1}, x_2, ..x_n)$ and the upper bound by $F(\overline{x_1}, ...x_n)$. In each case the variable x_1 has now a constant value and another component of the gradient may now have a constant sign. This process is implemented in the procedure:

```
Evaluate_With_Parser(char *file_name, char *gradient_name, int nb_eq,
                     INTERVAL_VECTOR &Unknowns,INTERVAL_VECTOR &Value)
```

where `gradient_name` is the name of file containing the analytical form of the gradient equation. We will now present some examples of use of the parser.

3 A first example: trajectory verification of robot

Consider the *parallel robot*, called a Gough platform [1], described in figure 1. In this robot a base and a

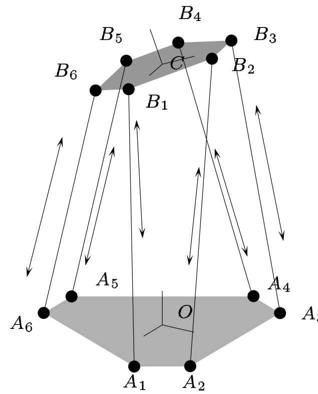

Figure 1: The classical Gough-type parallel robot

platform are connected through 6 legs which have a ball-and-socket joint at each extremity A_i, B_i. Linear actuators enable one to change the leg lengths, which in turn enables one to control the position and orientation of the platform.

We define a reference frame $O, (\mathbf{x}, \mathbf{y}, \mathbf{z})$ which is attached to the base and a mobile frame $C, (\mathbf{x_r}, \mathbf{y_r}, \mathbf{z_r})$ which is attached to the platform. We may represent a pose[2] of the platform by the coordinates x_C, y_C, z_C of the origin C of the mobile frame in the reference frame and its orientation by using the classical Euler angles ψ, θ, ϕ, which enable one to calculate the rotation matrix R for transforming the coordinates of a vector expressed in the mobile frame into its coordinates in the reference frame. The coordinates of the attachment points A_i are assumed to be known in the reference frame, while the coordinates of the B_i points are known in the mobile frame.

A *trajectory* for this robot is a set of time-dependent analytical functions

$$x_C = D_x(T) \quad y_C = D_y(T) \quad z_C = D_z(T) \quad \psi = D_\psi(T) \quad \theta = D_\theta(T) \quad \phi = D_\phi(T) \tag{4}$$

which describe the pose of the platform as a function of the time T, being supposed to lie in the range [0,1]. For physical reasons the leg lengths ρ of the robot are constrained to lie in a given range:

$$\rho_{min} \leq \rho_i \leq \rho_{max} \quad \forall i \in [1, 6] \tag{5}$$

For a Gough platform the length of a leg is simply the norm of the vector $\mathbf{AB}$ which may be written as

$$\mathbf{AB} = \mathbf{AO} + \mathbf{OC} + \mathbf{CB} \tag{6}$$

and the square of the leg length ρ is given by:

$$\rho^2 = ||\mathbf{AB}||^2 \tag{7}$$

For a given robot the first element of the right hand term of equation (6) is known. The last element, $\mathbf{CB}$ is equal to $R\mathbf{CB_r}$ where R is the rotation matrix, a function of ψ, θ, ϕ, and $\mathbf{CB_r}$ is the coordinate vector of B in the mobile frame, which is known. Using equations (4) we may transform equation (7) into a time function $\rho^2(T)$. A trajectory will be *valid* if for all T in [0,1] we have:

$$\rho_{min} \leq \rho_i(T) \leq \rho_{max} \quad \forall i \in [1, 6] \tag{8}$$

Verifying if a trajectory is valid is clearly a crucial point in the applications (which range from machine-tool to fine positioning of telescope antenna and surgical operations) as the robot will be able to perform its task only in that case. The usual approach is to consider a specific type of trajectory (typically line segments), to sample the trajectory at a regular step, then to compute the leg lengths at the corresponding poses and reject the trajectory for which a pose is not valid. This approach has some drawbacks:

- it is not flexible as only one type of trajectory can be tested
- if is not safe as, due to the non-linearity of the leg lengths, two successive poses may be valid while a pose between them is not
- the above problem may be partially solved by decreasing the sampling step but then the computation time become prohibitive

Our purpose is to propose a fast and safe method that enables one to test almost any type of trajectory.

For a given trajectory we may obtain by using MAPLE an analytical expression of $\rho_i(T)$. Consider for example the planar trajectory shown in figure 2. Using the convention x_C =`x`, y_C =`y`, z_C =`z`, ψ =`p`, θ =`t`, ϕ =`h` the trajectory may be defined as:

[2] In robotics a *pose* denotes both the position and orientation of a rigid body

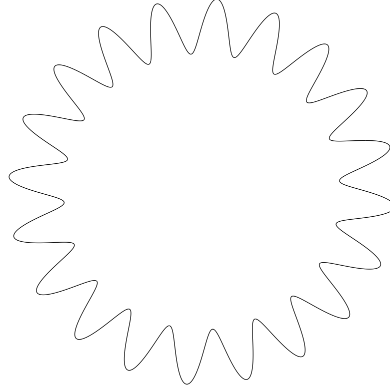

Figure 2: An example of complex trajectory: the gear trajectory

```
p:=0:t:=0:h:=0:
x:=(3+0.5*sin(40*Pi*T))*sin(2*Pi*T):
y:=(3+0.5*sin(40*Pi*T))*cos(2*Pi*T):
z:=56:
```

For a given robot the square of length of the first leg is:

$$\begin{aligned}\rho_1^2 &= \frac{12741}{4} + 3\sin(40\,\pi\,T) + 36\sin(2\,\pi\,T) - 1/4\,(\cos(40\,\pi\,T))^2 \\ &\quad +6\sin(2\,\pi\,T)\sin(40\,\pi\,T) - 12\cos(2\,\pi\,T) - 2\cos(2\,\pi\,T)\sin(40\,\pi\,T)\end{aligned}$$

To determine if such trajectory is valid we may determine the minimum and maximum value of each leg length over the whole trajectory and if one of these values does not lie in the allowed range, then the trajectory is not valid. Finding the maximum and minimum of such function is not an easy task. As we wish to be able to test any trajectory we will use our parser. The principle is to compute the analytical form of the leg lengths as functions of T (using MAPLE for example) as soon as a trajectory has been defined and then to use the algorithm that we will present now.

The interval evaluation of a quantity Q will be denoted $\mathcal{B}(Q)$, the lower bound of this interval evaluation $\underline{\mathcal{B}(Q)}$ and its upper bound $\overline{\mathcal{B}(Q)}$. We will also use a *bisection process* for the range $\hat{T} = [T_1, T_2]$: the result of the bisection process applied on this range is the 2 new ranges $[T_1, (T_1+T_2)/2], [(T_1+T_2)/2, T_2]$.

We will use a list $\mathcal{S}$ of ranges for T with n ranges. This list will be initialized with the range $S_1 = [0,1]$ and hence $n = 1$. During the algorithm we will consider the i-th element $\mathcal{S}_i$ of the list $\mathcal{S}$. We start with $i = 1$ and the algorithm proceeds along the following steps:

1. if $i > n$ return `VALID TRAJECTORY`
2. compute $\mathcal{B}(\rho_i(\mathcal{S}_i))$:
    (a) if there exists an i such that $\underline{\mathcal{B}(\rho_i)} > \rho_{max}$ or $\overline{\mathcal{B}(\rho_i)} < \rho_{min}$, then return `INVALID TRAJECTORY`

(b) if there exists an i such that $\underline{\mathcal{B}(\rho_i)} < \rho_{min}$ or $\overline{\mathcal{B}(\rho_i)} > \rho_{max}$, then bisect $\mathcal{S}_i$ and add the resulting ranges at the end of $\mathcal{S}$. Then $i = i + 1$, $n = n + 2$ and go to step 1

3. $i = i + 1$ and go to step 1

Consider what will happen with the range $\mathcal{S}_1$. At step 2 we will compute the interval evaluation of the 6 leg lengths. At step 2(a) we have found that one of the leg lengths is always lower than ρ_{min} or greater than ρ_{max} i.e. the trajectory is not valid. At step 2(b) we have found that the interval evaluation of one of the leg lengths includes the range $[\rho_{min}, \rho_{max}]$. But this does not mean that the leg lengths are outside their allowed ranges as interval evaluation may lead to an over-estimation of the bounds. Thus we will bisect the range $\mathcal{S}_1 = [0, 1]$ and start again with the range $\mathcal{S}_2 = [0, 0.5]$ and $\mathcal{S}_3 = [0.5, 1]$. If the current range $\mathcal{S}_i$ reaches step 3, then we start again with the next range in the list. The algorithm will stop if a range has led to a non valid leg length (step 2a) or if all all the ranges have been processed, in which case the trajectory is valid (step 1).

There are clearly numerous tricks that can improve the efficiency of this algorithm. For example we may calculate with MAPLE the derivative of the leg lengths and use the improved evaluator described in section 2.2 to determine sharper bound for the leg lengths. Using the derivative may be a good option to solve a problem that we have not yet mentioned: at step 2a it may happen that even for a $\mathcal{S}_i = [T1, T2]$ having a very small width we have $\underline{\mathcal{B}(\rho_i)} < \rho_{min}$ and $\overline{\mathcal{B}(\rho_i)} \geq \rho_{max}$, due to interval overestimation. Assume now that we evaluate the derivative of ρ and that the lower and upper bounds of this derivative have same sign, say positive. In that case we get $\underline{\mathcal{B}(\rho_i)} = \rho(T1)$ and $\overline{\mathcal{B}(\rho_i)} = \rho(T2)$. We will evaluate $\rho(T1), \rho(T2)$ using the same data type that will be used in the controller of the robot (typically `float` or `double`). If $\rho(T1) < \rho_{min}$, then we will reject the trajectory: indeed although it may happen that the trajectory is in fact valid, it will anyway be rejected by the robot controller. If we assume now that the bounds on the derivative have not the same sign our algorithm is not able to determine if the trajectory is valid: an error signal will be returned if the width of $\mathcal{S}_i$ is lower than a fixed threshold.

In this trajectory checking algorithm most of the computation time is devoted to the computation of the analytical form of the leg lengths. If we want to check quickly trajectories that differ only by some numerical values we may use the parametric facilities of the parser. For example the gear trajectory consists of a main trajectory, which is a circle, and a perturbation of the main trajectory. Thus the gear trajectory may be written as:

```
p:=0:t:=0:h:=0:
x:=(x1+x2*sin(40*Pi*T))*sin(2*Pi*T):
y:=(x1+x2*sin(40*Pi*T))*cos(2*Pi*T):
z:=56:
```

The analytical form of the leg lengths will contain the parameters `x1, x2`. But when using the algorithm we will have to compute this analytical form only once, and they can be used for any similar trajectory just by setting the appropriate values of `x1, x2`.

A motion verifier prototype has been implemented. The argument of this program is a trajectory file name: the verifier will run first a MAPLE session which starts by reading the trajectory file and then run a specific MAPLE program which compute the analytical form of the leg lengths for the trajectory and write it in a file. Then the motion verifier algorithm is applied, using the file produced by the MAPLE session and the program returns a message indicating if the trajectory is valid or the time interval in which the trajectory is not valid.

We have considered the gear trajectory (figure 2): on a SUN ULTRA 1 workstation the verifier is able to determine that the trajectory is invalid (figure 3) in a computation time of 4.4 seconds (including the MAPLE computation). If we consider a parametric formulation of the problem, as explained in the previous paragraph, we process with MAPLE the analytical form of the leg lengths only once and we are then able to deal with any trajectory in a given class without having to run any MAPLE computation: in that case the computation time is reduced to 110 ms.

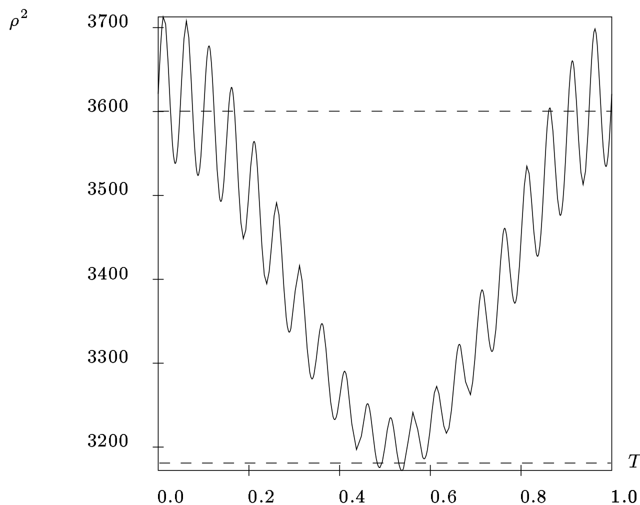

Figure 3: The square of one the leg lengths for the gear trajectory. The dashed lines represent the minimal and maximal value of this leg length. It may be seen that the trajectory is invalid and the motion verifier indicates that the trajectory is invalid at least in the time range [0,0.015625]

Note also that with minor modification of the kernel of the program we may define any other validity criteria for a trajectory. For example we may want to check that the angle between the direction of the leg and a fixed direction $\mathbf{u}$ does not exceed a given value θ_{max} (for example to check that the passive joints at A does not exceed their mechanical limits). In that case we will have the constraint equation:

$$\frac{\mathbf{AB.u}}{||\mathbf{AB}||} - \cos\theta_{max} > 0$$

The right hand term of this inequality may be transformed into a time dependent inequality that may be checked by the program.

4 A second example: generic system analyzer and solver

In this section we consider a system of n equations $F_1, \dots F_n$ in m $(n \geq m)$ unknowns $x_1, x_2, \dots, x_m$. We assume that at least one of the F_i is algebraic in at least one of the x_j. If F_i is algebraic of degree k in

x_j we will denote by C_{ir}^j with r in $[0, k]$ the coefficients of x_j^r in the equation F_i (these coefficients may be functions of the other unknowns). The set of these coefficients will be denoted $\mathcal{C}_i^j$. We address now two problems:

- analyze the system i.e. determine intervals for each unknown that may contain real roots of the system
- solve effectively the system, i.e. find all its real roots.

It must be noted that there are very few results on the analysis of such general systems, while there are general methods to solve them but none guaranteeing to get all the roots [4]. Our purpose is to propose very flexible tools which enables one to perform a fast analysis of this type of system and eventually to solve them.

We will assume that the analytical form of the equations is available in a formula file and that similarly all the sets $\mathcal{C}_i^j$ are also available in formula files. Note that a specific MAPLE package has been developed to perform automatically the computation of all the necessary formula files, being provided with a set of equations.

Consider for example the system:

$$F_1(x_1, x_2) = x_1^2 + x_1 \cos(x_2) - 1 = 0 \quad (9)$$

$$F_2(x_1, x_2) = x_2^2 - x_2 + \sin(x_1) - 1/10 = 0 \quad (10)$$

Using the MAPLE package we will get a formula file for F_1, F_2, a formula file for $\mathcal{C}_1^1 = \{1, \cos(x_2), -1\}$ and for $\mathcal{C}_2^2 = \{1, -1, \sin(x_1) - 1/10\}$.

4.1 The analyzer

The analyzer takes as input the formula files and a set of initial ranges for each of the $m \leq n$ unknowns of the system. Its output will be a set of sets of p intervals, all of them being strictly included in the initial set, which may include real roots of the system. The basic tools that we are using for producing this set are all the theorems that give bounds on the real roots of an univariate polynomial e.g. Newton method, Laguerre method, Cauchy theorems [9] etc... All these theorems have been implemented in the interval-based library `ALIAS`[3] and may be used with a polynomial whose coefficients are intervals. A global procedure applies in turn all these theorems and returns a lower bound and an upper bound for the roots of the polynomial: we will denote by $\mathcal{B} = [\underline{a}, \overline{a}]$ the range returned by the procedure. Note that many such theorems are able to find bounds either on the positive or negative real roots of a polynomial.

We will now describe the analysis using the same notation as in the previous section and a range for the variable x_k will be denoted $x_k^\star$.

1. perform an interval evaluation of $F_i(x_1^\star, \ldots, x_m^\star)$ $\forall$ i in $[1, m]$
2. if there exists an i such that $0 \notin \mathcal{B}(F_i(x_1^\star, \ldots, x_m^\star))$ return `NO SOLUTION`
3. for j from 1 to n do

[3] freely available on the Web, see http://www.inria.fr/saga/logiciels/ALIAS/ALIAS.html

- for k from 1 to m do
- if F_j is algebraic in x_k then
  - compute the interval evaluation of the coefficients of F_j considered as a univariate polynomial in x_k
  - compute $\mathcal{B}(x_k)$
  - if $\underline{\mathcal{B}(x_k)} > \underline{x_k}$ or $\overline{\mathcal{B}(x_k)} < \overline{x_k}$, then
    * compute $v_k = \underline{\mathcal{B}(x_k)} - \underline{x_k}$ or $v_k = \overline{x_k} - \overline{\mathcal{B}(x_k)}$
    * update $\underline{x_k}$ or $\overline{x_k}$
    * if $v_k > \epsilon$ go to step 1
- end-do

4. end-do
5. return $(x_1^\star, \ldots, x_m^\star)$

The principle of the algorithm is to consider in sequence the equation $F_1, \ldots, F_m$. If the current equation F_i is algebraic in terms of x_k we compute the interval coefficients of F_i considered as an univariate polynomial in x_k. Using these coefficients we use the classical theorems to determine bounds on the real roots of this polynomial. If the lower or upper bound is better than the current bound on x_k we update $x_k^\star$. This process is repeated as soon as an improvement has been obtained for one of the variable x_k as an improvement may then be obtained for another variable. For example consider the system (9,10): assume that an improvement is obtained when we consider F_2 as an univariate polynomial in x_2, then the interval coefficients of F_1, considered as an univariate polynomial in x_1, will be modified and may lead to an improvement on $x_1^\star$. This algorithm will always terminate either because no variable has been improved or there is no solution in the ranges or the changes on the ranges are lower than the fixed threshold ϵ.

Consider the system (9,10) and assume that we are trying to improve the range [-5,0] for x_1 and x_2. For the first equation, considered algebraic in x_1, the algorithm finds out that the bound on the real roots is [-1.6180,-0.5]; hence x_1 is updated to this range. The second equation is algebraic in x_2 and the bounds on the real roots are [-0.66189,-0.4107], which is an improvement of x_2. Since both x_1 and x_2 have been improved the analyzer iterates and produces after the third iteration the result:

$$x_1 \in [-\underline{1.4}703139968387926029, -\underline{1.4}693992456867683849]$$
$$x_2 \in [-\underline{0.659}72233611863173586, -\underline{0.659}68259403205231628]$$

As we may see on this example we may get very quickly tight bounds on the roots. Thus it may be interesting to use theorems that enable to determine if a unique solution exists in a given ball, especially as we may get rid, at least partially, of the threshold ϵ. This is what is done in our implementation in which we use:

- Kantorovitch theorem [8], which enables one to determine if the Newton scheme initialized at a given point will converge, together with the radius of the ball centered at this point for which the solution of the system is unique

- Moore test [5, 7] which enables one to determine if the system has a unique solution for given ranges of the unknowns and provides a numerical procedure to compute the solution

To apply Kantorovitch theorem we need the Jacobian and Hessian matrix of the system (which are computed automatically if the MAPLE package is used), while for Moore test only the Jacobian is necessary. In the previous example Kantorovitch theorem enables one to determine that there is a unique solution in the ranges determined by the analyzer at the third iteration, the solution being:

$$x_1 = -1.4702973934583798421 \quad x_2 = -0.65972161798209860706$$

Note that the analysis may not be very sensitive to the input ranges. For example if the range for x_1, x_2 in the previous system have been defined as [-200,0] we get exactly the same result for the analyzer.

The previous algorithm may be seen as a *local analyzer* that tries to improve a given set of ranges. We may also design a *global analyzer* that uses first the local analyzer until no more improvement on the ranges is obtained and then uses a *bisection process* on the ranges to improve the result of the local analyzer. The bisection process will generate a list of 2^m sets of m ranges and will apply in sequence the local analyzer on each set. Alternatively we may bisect only one of the variables (chosen by using, for example, the *smear* function described in [3]), the bisection process leading now to only 2 sets . We may evidently proceed to more than one bisection process to improve the result. For example consider the system:

$$x_1 e^{x_2} + 0.1 + x_1 \sin(x_2) = 0 \tag{11}$$

$$x_2 x_1 - 0.5 + x_2 \cos(x_1) = 0 \tag{12}$$

with x_1, x_2 in $[-\pi, 0]$. The local analyzer produces as ranges $x_1 \in [-\pi, -0.1128]$, $x_2 \in [-\pi, -0.1207]$. If we bisect twice (a first bisection on the result of the first run of the local analyzer followed by a local analysis of the resulting ranges, a second bisection on the result followed by a local analysis) then we get $x_1 \in [-1.627, -0.924]$ and $x_2 \in [-0.964, -0.2969]$.

4.2 The generic solver

Interval-based methods for solving systems are implemented in the `ALIAS` library and have been adapted to be used with the parser. They use basically a bisection process together with the Jacobian and Hessian matrix of the system. These matrices are used first to improve the interval evaluation of the equations and, second, for applying Kantorovitch and Moore theorem [5, 7] to determine if a unique solution exists within a set of ranges. Hence a generic solver has been developed, which takes as input the formula files and the ranges for the unknowns (which may be produced by the analyzer for semi-algebraic equations). Therefore we may compute in a user-friendly way the real solutions of a system just by defining the equations of the system in a MAPLE file (note that the generic solver may also be called directly from MAPLE). As a an example of the use of the generic solver consider the system:

$$xe^y + 0.1 + \sin(xy) = 0$$

$$x + 0.5 + \cos(x + y) = 0$$

The generic solver, called from MAPLE, finds three real roots for x, y in $[-\pi, \pi]$:

$$x = 0.0479909 \quad y = -2.1987$$
$$x = -0.00980269 \quad y = 2.09292$$
$$x = -0.785962 \quad y = -.494825$$

while the `fsolve` procedure of MAPLE is able to find only the first root.

5 Conclusion

The proposed parser is a convenient method to compute the interval evaluation of arbitrary equations without having to write any specific code. This parser has two main advantages: it enables one to change easily the form of the function to be evaluated and to deal with problems in which a general algorithm has to be used on arbitrary functions which are not known beforehand. This later point has been used to solve an open robotics problem and for the development of an analyzer and a solver for system of non-linear equations.

References

[1] Gough V.E. and Whitehall S.G. Universal tire test machine. In *Proceedings 9th Int. Technical Congress F.I.S.I.T.A.*, volume 117, pages 117–135, May 1962.

[2] Hansen E. *Global optimization using interval analysis.* Marcel Dekker, 1992.

[3] Kearfott R.B. and Manuel N. III. INTBIS, a portable interval Newton/Bisection package. *ACM Trans. on Mathematical Software*, 16(2):152–157, June 1990.

[4] Kelley C.T. *Iterative methods for linear and nonlinear equations.* SIAM, 1995.

[5] Moore R.E. A test for the existence of solution to nonlinear systems. *SIAM J. of Numerical Analysis*, 14:611–615, 1977.

[6] Moore R.E. *Methods and Applications of Interval Analysis.* SIAM Studies in Applied Mathematics, 1979.

[7] Moore R.E. and Qi L. A succesive interval test for nonlinear systems. *SIAM J. of Numerical Analysis*, 19(4):845–850, 1982.

[8] Tapia R.A. The Kantorovitch theorem for Newton's method. *American Mathematic Monthly*, 78(1.ea):389–392, 1971.

[9] Zippel R. *Effective polynomial computation.* Kluwer, 1993.