Serdica J. Computing **1** (2007), 469–504

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

INTRODUCTION TO THE MAPLE POWER TOOL `intpakX`*

Walter Krämer

Abstract. The Maple Power Tool `intpakX` [24] defines Maple types for real intervals and complex disc intervals. On the level of basic operations, `intpakX` includes the four basic arithmetic operators, including extended interval division as an extra function. Furthermore, there are power, square, square root, logarithm and exponential functions, a set of standard functions, union, and intersection. Reimplementations of the Maple construction, conversion, and unapplication functions are available. Additionally, there is a range of operators for complex disc arithmetic.

As applications, verified computation of zeroes (Interval Newton Method) with the possibility to find all zeroes of a function on a specified interval, and range enclosure for real-valued functions of one or two variables are implemented, the latter using either interval evaluation or evaluation via the mean value form and adaptive subdivision of intervals. The user can choose between a non-graphical and a graphical version of the above algorithms displaying the resulting intervals of each iteration step.

The source code (about 2000 lines of Maple–code) of the extension `intpakX` is freely available [23].

ACM Computing Classification System (1998): G.1.0, G.1.5, G.4.

Key words: Computer Algebra, Validated Computations, Visualization of Interval Methods, Didactical Tool, Maple Power Tool, `intpakX`.

*The paper has been presented at the 12th International Conference on Applications of Computer Algebra, Varna, Bulgaria, June, 2006.

1. Introduction and Acknowledgement. The Maple Power Tool `intpakX` [8, 24] includes the former packages `intpak` [5] and `intpakX` [6, 7]. It provides the new data type `interval` (long number intervals), the corresponding basic arithmetic operations, basic interval functions, extended interval division, Interval Newton Method, functions to enclose a range of functions, complex basic interval operations (disk arithmetic), the complex exponential function for disc arguments, and a couple of plotting functions to visualize the interval methods. It is particularly useful for illustrating interval algorithms in the field of validating computing. The tool has been used (and will be used) by the author with great success when giving lectures in this field. It is planned to extend the tool by supervising further Bachelor's and Master's theses.

Please note: `intpakX` assumes that the basic operations provided by Maple are accurate to at least one unit in the last place (ulp) with respect to the actual value of Maple's environment variable `Digits` (number of decimal digits Mapel uses to represent software floating-point numbers). If so, it is guaranteed that `intpakX` computes enclosures for ranges of expressions built from these basic operations. Up to now, no violation of the assumption is known to the authors of `intpakX`.

A similar statement (1 ulp accuracy) about the accuracy of Maple functions like exp, log, sin, Bessel, etc., is probably not correct. Therefore `intpakX` uses several guard digits in its computations. Nevertheless computation of expressions involving such functions cannot be guaranteed to enclose the true range! Despite this limitation, we are convinced that `intpakX` is a valuable didactical tool for illustrating interval algorithms/methods. As soon as error bounds for Maple funcions are available, `intpakX` will use the bounds to compute guaranteed enclosures.

The author would like to thank his students/assistants for investing so much time and effort in this package. Particularly, thanks to Ilse Geuling and Markus Grimmer.

How to install `intpakX` and how to use this Maple Power Tool in your own worksheets is described in the source files, which are publicly available on the WEB (see [23]). From this link you get the most recent version of `inpakX`. The Maplesoft link [24] may present an older version.

2. Real Interval Operations and Functions. A variable x is of type `interval` if either x is an empty list or if $x = [x_1, x_2]$ is a list with two elements. The interval endpoints x_i, $i = 1, 2$ must fulfill one of the following requirements:

- x_i is a real number of type `float`
- x_i is equal to 0
- $x_i \in \{-$`infinity`$,$ `infinity`$\}$.

Numbers of type `integer` or of type `fraction` or constants predefined in Maple (to these belong `Pi, gamma, Catalan, FAIL, false, true`) are not allowed as interval endpoints! However, the `intpakX` function `construct` may be used to create valid intervals in these cases.

Basic Arithmetic Operations: The basic operations implemented are &+, &−, &∗, &/ and `inv`. Their input parameters must be either of type `interval` or of type `num_or_FAIL`. The output is an interval.

A variable is of type `num_or_FAIL` if its value is a number (i. e. the variable is of type `numeric`), −`infinity` or a constant predefined in Maple (see above).

The priorities for the so-called 'inert' operators &+, &−, &∗, &/ are unfortunately set in Maple so that &+ and &− have a higher priority than &∗ and &/. Because of this, one has to use lots of brackets in terms!

Basic Interval Functions: The interval functions implemented are &`sqr`, &`sqrt`, &`ln`, &`exp`, &∗∗, &`intpower`, &`sin`, &`cos`, &`tan`, &`arcsin`, &`arccos`, &`arctan`, &`sinh`, &`cosh` und &`tanh`. Originally [5] it was assumed that the corresponding mathematical functions in Maple deviate maximally by 0.6 ulp from the exact results. This assumption is, however, confirmed nowhere in the Maple manual. `intpakX` [7] uses several guard digits (concerning reliability refer to the Introduction).

Most names are self-explaining. The operation &`intpower` corresponds to the function x^n, n a natural number. The operator &∗∗ corresponds to the function x^α, where α can be an interval, an integer or a number of type `float`.

Auxiliary Functions: The function `construct` generates from a number or a pair of numbers an element of type `interval`. As an optional parameter may be entered the string 'rounded'. In this case the interval endpoints are rounded by one ulp from above or from below.

Also contained are the functions `midpoint`, `width`, &`intersect`, &`union` and `is_in`. The function `width` calculates the diameter and `midpoint` an enclosure (confinement) of the centre of an interval.

For converting a term into an interval term resp. into an interval function, there are the commands `'convert/interval'` and `inapply`.

Example 1. The data type `interval`

Interval endpoints of type `integer` are *not* admitted:

```
> x:=[1,2];
```

$$x := [1,\ 2]$$

```
> type(x,interval);
```

$$false$$

```
> x:=construct(1,2); type(x,interval);
```

$$x := [1.,\ 2.]$$

$$true$$

Generate confining interval:

```
> construct(1,rounded);
```

$$[.999999999,\ 1.000000001]$$

```
> y:=construct(1,infinity,rounded);
```

$$y := [.999999999,\ \infty]$$

```
> type(y,interval);
```

$$true$$

Diameter and enclosure of interval centre:

```
> width(x); width(y);
```

$$1.$$

$$\infty$$

```
> midpoint(x);
```

$$[1.499999999,\ 1.500000001]$$

Example 2. Set-theoretic operations

```
> x:=[1.,3.]; y:=[2.,infinity]; z:=[4.,5.];
```

$$x := [1.,\ 3.]$$

$$y := [2.,\ \infty]$$

$$z := [4., 5.]$$

```
> x &union y;
```

$$[1., \infty]$$

```
> x &union z;
```

$$[1., 3.], [4., 5.]$$

The resulting set consists of two intervals. Thus, &`union` does **not** calculate the interval hull. For this the procedure &`Convex_Hull` is available in `intpakX`.

```
> x &intersect y;
```

$$[2., 3.]$$

```
> x[1];
```

$$1.$$

```
> is_in(x[1],y);
```

false

```
> is_in(z,y);
```

true

Example 3. Arithmetic operations and interval functions

```
> [1.,2.]  &+ 3 &* 0; # Wrong priorities of operators
```

$$[0, 0]$$

```
> [1.,2.]  &+ (3 &* 0);
```

$$[.999999999, 2.000000001]$$

```
> [-1.,2.]  &intpower 3;
```

$$[-1.000000001, 8.000000001]$$

```
> &sqr(&cosh(1)) &- &sqr(&sinh(1));
```

$$[.9999999919, 1.000000008]$$

Example 4. Transforming expressions into interval functions

```
> f:=inapply(0.5*t,t);
```

$$f := t \rightarrow .5\,\text{`\&}*\text{`}\,t$$

```
> f(2);
```

$$[.999999999,\ 1.000000001]$$

```
> f:=inapply(sqrt(t),t);
```

$$f := \&sqrt$$

```
> f([4.,9.]);
```

$$[1.999999999,\ 3.000000001]$$

```
> f:=inapply(sin(x)+x,x);
```

$$f := x \rightarrow \text{`\&sin`}(x)\,\text{`\&}+\text{`}\,x$$

```
> f(0);
```

$$[0,\ 0]$$

Example 5. Range enclosures

Enclosure of the value domain of $f(x) := x^3 - x^2 - x + 1$ on the interval $[0, 0.5]$ by evaluation with intervals, using the mean value form and taking into consideration the monotony properties of f.

```
> x:='x'; # release variable
```

$$x := x$$

```
> f:=x^3-x^2-x+1;
```

$$f := x^3 - x^2 - x + 1$$

```
> F:=inapply(f,x); # Transformation of f into an interval
    function
```

$$F := x \rightarrow (x\,\text{`\&intpower`}\,3)\,\text{`\&}+\text{`}\,(((-1)\,\text{`\&}*\text{`}\,(x\,\text{`\&intpower`}\,2))\,\text{`\&}+\text{`}\,(((-1)\,\text{`\&}*\text{`}\,x)\,\text{`\&}+\text{`}\,1))$$

Transformation of the derivative f' of f into an interval function

```
> dF:=inapply(diff(f,x),x);
```

$$dF := x \rightarrow (3\,\text{`\&}*\text{`}\,(x\,\text{`\&intpower`}\,2))\,\text{`\&}+\text{`}\,(((-2)\,\text{`\&}*\text{`}\,x)\,\text{`\&}+\text{`}\,(-1))$$

Enclosure (confinements) of the value domain of f on the interval [0, 0.5]

```
> X:=[0,0.5];
```

$$X := [0,\ .5]$$

```
> mid_X:=midpoint(X);
```

$$mid\_X := [.2499999999,\ .2500000002]$$

```
> r_i:=F(X); # evaluation with intervals
```

$$r\_i := [.2499999994,\ 1.125000003]$$

```
> r_m:=F(mid_X) &- ( dF(X) &* (X &- mid_X) );#mean value form
```

$$r\_m := [.2031249977,\ 1.203125003]$$

The evaluation with intervals of f' on the interval $X = [0, 0.5]$ shows that f is monotone decreasing in X.

```
> dF([0,0.5]);
```

$$[-2.000000003,\ -.2499999985]$$

The exact value domain of f on X is the interval $[0.375, 1]$. A very sharp enclosure of the value domain can be calculated in the following way:

```
> r_e:=construct(F(X[2])[1],F(X[1])[2]);
```

$$r\_e := [.3749999993,\ 1.000000003]$$

`F(X[2])` calculates the evaluation with intervals of f at the point `X[2]`$=$ 0.5. `F(X[2])[1]` gives the lower interval end of `F(X[2])`, thus a **safe** lower bound for the minimum of f on the interval $[0, 0.5]$.

In order to verify if the 'exact' enclosure of the value domain `r_e` is contained in the intersection of the enclosures `r_i` and `r_m` calculated above, e. g. the procedure `is_in` may be used.

```
> is_in(r_e, r_i &intersect r_m);
```

true

The Procedure `Interval_power`**.** In order to enable the evaluation of e. g. the second derivative of $f(x) := \sqrt{x}$

$$f''(x) = -\frac{1}{4}x^{-\frac{3}{2}}$$

with intervals, the procedure `Interval_power` (alias &∗∗) allows rational numbers as second parameter.

Calculating $f''(4)$ with floating point arithmetic

```
> evalf(-1/4*4^(-3/2));
```

$$-.03125000000$$

Evaluation of $f''(4)$ with intervals

```
> df2:=inapply(diff(sqrt(x),x$2),x);
```

$$df2 := x \rightarrow \left(\frac{-1}{4}\right) \text{`\&}*\text{`} \left(x \text{`\&}**\text{`} \left(\frac{-3}{2}\right)\right)$$

```
> df2(4);
```

$$[-.03125000004,\ -.03124999997]$$

The Procedure `is_in`**.** The `intpak`-procedure `is_in` has two input parameters, and verifies if the first parameter is contained in the second (in a set-theoretical sense). As input parameters are allowed variables of type `interval`, numbers of type `numeric` and the values `FAIL`, `infinity` and `-infinity`. The procedure `is_in` gives out a correct result even if one of the numbers is rational or if the length of the entered numbers exceeds the value of the variable `Digits`. Example:

```
> Digits;
```

$$10$$

```
> is_in(1.9999999999999999,[2.,2.]);
```

$$\textit{false}$$

```
> is_in(1/3,[0.3333333332,0.33333333333333333]);
```

$$\textit{false}$$

The Procedure `construct`**.** Example

```
> construct(1/3);
```

$$[.3333333333,\ .3333333334]$$

```
> (1/3) &- 0.3333333333;
```

$$[0,\ .1000000001\,10^{-9}]$$

```
> construct(1.0000000001);
```

$$[1.000000000,\ 1.000000001]$$

```
> 1.0000000001 &- 1;
```

$$[0,\ .1000000001\,10^{-8}]$$

3. Application I: Ranges With Graphical Output.

3.1. Functions of One Variable. Two simple possibilities to confine the range of a function $f : D \subseteq \mathbb{R} \to \mathbb{R}$ over an interval $[x] \subseteq D$ were already presented in Example 5, Section 2: namely, the interval-evaluation of f (if it exists) and the mean value form (if the interval-evaluation of f' over $[x]$ exists.

An improved range enclosure is obtained if the interval $[x]$ is partitioned and a range enclosure is calculated over each subinterval. The interval hull of these subrange enclosures is then an enclosure of the range of f over $[x]$. If the partition is continued successively, then the initial range enclosure can be improved successively.

This is realized by the procedure `compute_range`. The procedure demands three input parameters

- a function `f`,
- the start interval `xstart` (may be entered either as interval or as `range`)
- the number of `iterationsteps` to be performed. It is used as an interrupt criterion.

The order of the three parameters mentioned above is compulsory. In addition, the entering of four optional parameters (in any order) is possible:

- Remitting a parameter `Nx = n`, `n` an integer greater than or equal to 1, effects the partitioning of the start interval into `n` intervals before all.

- The optional parameter `linear` (`quadratic`) effects the linear (quadratic) convergence of the procedure. On entering `linear`, the 'naive' interval evaluation is used for determining the subrange enclosures. If `quadratic` is remitted as parameter, the procedure `compute_combined_range` is used for determining the subrange enclosures, combining interval-evaluation, mean-value form and monotony-test. If none of these two parameters is entered, then in the first three iteration steps interval-evaluation is used for determining the range enclosures, and from step 4 on, the procedure `compute_combined_range` is used.

- The optional parameter `adaptive` effects the adaptive partitioning of the current interval list, and therefore generally leads to a calculation-time reduction.

- The optional parameter `colorlist = [color1,color2,...]` determines the colours used for the graphical illustration of each iteration step. `color1, color2`, etc. must be colours predefined in Maple, e. g. `blue, red, green, magenta, coral, brown` etc. This does not, however, influence the illustration of the last iteration step which is always illustrated in yellow.

For reasons of clarity, only the graphical illustration of the last three iteration steps and the function f are given out. The graphical illustration of all iteration steps is, however, stored in the global variable `q`. The variable `q` is a table. If 3 iteration steps were performed, then the entries `q[1]`, `q[2]` and `q[3]` would contain the illustrations of each iteration step, however, not the graph of the function. This is stored in the table entry `q[4]`.

Also the calculated range enclosures are stored — in the global variable `r`. It is also a table and `r[i]` contains the range enclosure calculated in the `i`-th step.

The current partition of the start interval is stored in the global variable `list_of_intervals`. The corresponding subrange enclosures are stored in the variable `list_of_ranges`.

Examples. Enclosure of the range of the function

```
> f:=x->exp(-x^2)*sin(Pi*x^3);
```

$$f := x \to e^{(-x^2)} \sin(\pi\, x^3)$$

over the interval $X := [0.5, 2.]$ using the procedure `compute_range`

```
> X:=[0.5,2.];
```

$$X := [.5,\ 2.]$$

```
> compute_range(f,X,4);

initial range confinement =     [-.7788007834, .7788007834]

range confinement after iteration step 4 =
[-.3233867682,.6103317518]
```

The initial range confinement is $\approx [-0.78, 0.78]$. The range confinement after 4 iteration steps, that is after partitioning into $2^4 = 16$ subintervals, is $\approx [-0.32, 0.61]$. The graphical output can be found in Figure 1.

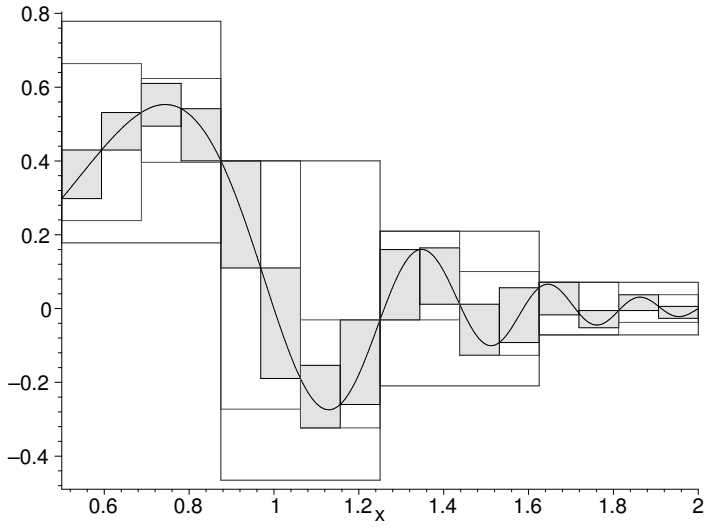


Fig. 1. Refining a range confinement by partitioning the argument domain into subintervals

The same range confinement is obtained already after 1 iteration step if the start interval is partitioned before all into 2^3 intervals and the optional parameter `quadratic` is given:

```
> compute_range(f,X,1,Nx = 2^3,quadratic);
```

```
initial range confinement =     [-.7788007834, .7788007834]

range confinement after partitioning into 8 subintervals =

[-.3233867682,.6639743998]

range confinement after iteration step 1 =
[-.3233867682,.6103317518]
```

Using the parameter `adaptive` can generally reduce the number of subintervals considerably:

```
> compute_range(f,[0.5,2.],6,adaptive);

initial range confinement =     [-.7788007834, .7788007834]

range confinement after iteration step 6 =
[-.2834388814,.5563221618]
```

The current partitioning of the start interval is stored in the variable `list_of_intervals`. Therefore, the total number of subintervals can be determined simply at any time. In the example above it is determined in the following way (after 6 iteration steps):

```
> nops(list_of_intervals);
```

24

Without the parameter `adaptive` the number of subintervals after 6 iteration steps would be $2^6 = 64$.

In order to illustrate only the last iteration step and the function f, the `plots`-command `display` can be used. The result of the following call-up is found in Figure 2.

```
> display([q[7],q[6]],title='Adaptive partitioning after
                    6 iterationsteps',titlefont=[TIMES,BOLD,12]);
```

3.2. Functions of Two Variables. The Procedure `compute_range3d` calculates range confinements for real-valued functions of two real variables over a two-dimensional interval $X \times Y$.

Its input parameters are (analogous to `compute_range`, however, **without** the optional parameters `linear/quadratic` and `adaptive`)

– the function `f`,

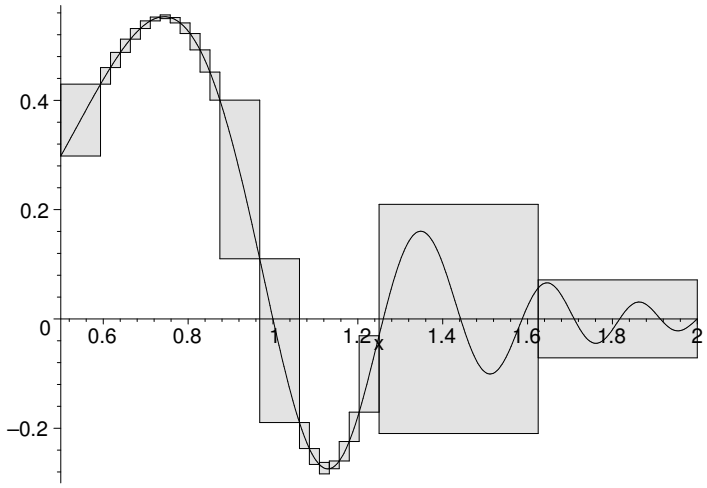


Fig. 2. Adaptive partitioning

– a real interval, or a domain `X`,

– a real interval, or a domain `Y`,

– the number of iteration steps to be performed.

The order of these parameters is compulsory. With this procedure, too, a series of optional parameters can be set:

– the parameters `Nx = n` and `Ny = m` effect a corresponding partitioning of the start interval (axe-parallel right-angle) in the `x`-`y`-direction,

– the parameter `colorlist` has the same meaning as in `compute_range`,

– the parameter `cutout = r` determines the strength of the lines in the illustration of the calculated confinements. Here, `r` should be 0, 1 or a fraction with $0 < r < 1$.

In addition, any options of the `plot3d`–command can be used.

For determining the subrange confinements in `compute_range3d` the 'naive' interval-evaluation is used. In each iteration step the subintervals are partitioned only in one direction. If e. g. two iteration steps are performed, then

in the first step a partitioning in x-direction and in the second step a partitioning in y-direction are carried out.

Example. Determining a range confinement for the function

```
> f:=(x,y)->exp(-x*y)*sin(Pi*x^2*y^2);
```

$$f := (x,\, y) \to e^{(-x\,y)} \sin(\pi\, x^2\, y^2)$$

over the interval $X \times Y = [\pi/8, \pi/2] \times [\pi/8, \pi/2]$.

```
> X:=[evalf(Pi)/8,evalf(Pi)/2]; Y:=X;
```

$$X := [.3926990818,\ 1.570796327]$$

$$Y := [.3926990818,\ 1.570796327]$$

```
> compute_range3d(f,X,Y,4);

initial range confinement =    [-.8570898115, .8570898115]

range confinement after iteration step 1 =
[-.8570898115, .8570898115]

range confinement after iteration step 2 =
[-.6800891261, .8570898115]

range confinement after iteration step 3 =
[-.6800891261, .8486122905]

range confinement after iteration step 4 =
[-.5093193828, .7559256232]
```

After each iteration step the calculated range confinement is given out. Only the illustration of the last iteration step and the graphical illustration of the function f are given out. Also in this case, the graphical illustrations of the other iteration steps are stored in the global variable q.

The graphical output of the procedure can, as in any other 3d-graphics in Maple, be edited thereafter with the commands from the graphics-menu. The desired graphic-options can, however, also be called up directly as parameters. E. g. the command

```
> compute_range3d(f,X,Y,3,cutout=9/10,color=yellow,
> lightmodel=light2,axes=framed,titlefont=[TIMES,BOLD,12],
  title=‘range confinement by partitioning into
  subintervals‘,);
```

generates the graphics in Figure 3.

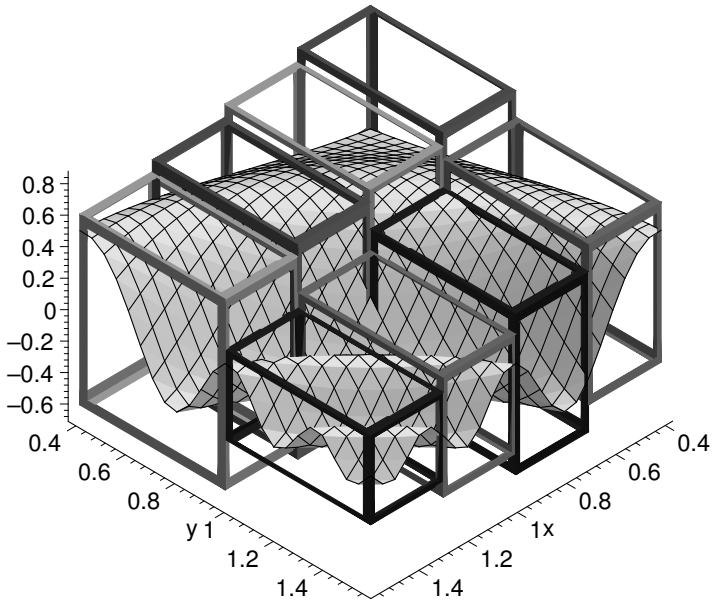


Fig. 3. Range confinement of a function in two variables

4. Application II: Verified Calculation of Zeroes With Graphical Illustration. The extension `intpakX` contains a realization of the extended interval-Newton-iteration

$$\begin{cases} [x]^0 & , \quad \text{real start interval} \\ [x]^{k+1} := N([x]^k) \cap [x]^k, & k = 0, 1, 2, \ldots , \end{cases}$$

where

$$N([x]) := m([x]) - \frac{f(m([x]))}{f'([x])}$$

denotes the interval-Newton-operator and $m([x])$ usually the centre of the interval $[x]$.

If $f : D \subset \mathbb{R} \to \mathbb{R}$ is a continuously differentiable function on D and $[x]^0 \subset D$ a real interval for which the interval-evaluation $f'([x]^0)$ exists, then the interval-Newton-iteration calculates confinements of all zeroes of f contained in $[x]^0$. In addition, using this procedure the existence and uniqueness of simple

zeroes of f in the given start interval can be proved. A more detailed description of this procedure can be found in [10].

The case $0 \in f'([x]^0)$ is permitted! Therefore, for performing this procedure one needs the extended interval division and the subtraction of an extended interval of a real number (extended interval subtraction, see page 485).

4.1.Extended Interval Division and Extended Interval Subtraction. Let $\mathbb{IR}$ be the set of real intervals and

$$\mathbb{IR}^* := \mathbb{IR} \cup \{[-\infty, r] \,|\, r \in \mathbb{R}\} \cup \{[l, +\infty] \,|\, l \in \mathbb{R}\} \cup \{[-\infty, +\infty]\}$$

the set of extended intervals.

The definitions of extended interval division and extended interval subtraction used in `intpak` resp. `intpakX` correspond to the definitions used in [19]. The interval operations defined in this way are inclusion-isotonic.

For two real intervals $[x] = [\underline{x}, \overline{x}]$ and $[y] = [\underline{y}, \overline{y}]$, the extended interval division is defined as follows

$$[x]/[y] := \begin{cases} [x] \cdot [1/\overline{y}, 1/\underline{y}], & \text{if } 0 \notin [y] \\ [-\infty, +\infty], & \text{if } 0 \in [x] \text{ and } 0 \in [y] \\ [\overline{x}/\underline{y}, +\infty], & \text{if } \overline{x} < 0 \text{ and } \underline{y} < \overline{y} = 0 \\ [-\infty, \overline{x}/\overline{y}] \cup [\overline{x}/\underline{y}, +\infty], & \text{if } \overline{x} < 0 \text{ and } \underline{y} < 0 < \overline{y} \\ [-\infty, \overline{x}/\overline{y}], & \text{if } \overline{x} < 0 \text{ and } 0 = \underline{y} < \overline{y} \\ [-\infty, \underline{x}/\underline{y}], & \text{if } 0 < \underline{x} \text{ and } \underline{y} < \overline{y} = 0 \\ [-\infty, \underline{x}/\underline{y}] \cup [\underline{x}/\overline{y}, +\infty], & \text{if } 0 < \underline{x} \text{ and } \underline{y} < 0 < \overline{y} \\ [\underline{x}/\overline{y}, +\infty], & \text{if } 0 < \underline{x} \text{ and } 0 = \underline{y} < \overline{y} \\ [\,], & \text{if } 0 \notin [x] \text{ and } 0 = [y]. \end{cases}$$

As the data type `interval` admits the points $-$`infinity` and `infinity` as interval endpoints, the extended interval division can be included without difficulty in the interval package. The corresponding command in `intpakX` is called `ext_int_div`. Examples:

```
> ext_int_div([1.,2.],[-1.,1.]);
```

$$[-\infty,\ -.999999999],\ [.999999999,\ \infty]$$

```
> ext_int_div([-2.,-1.],[0,2.]);
```

$$[-\infty,\ -.4999999999]$$

For $r \in \mathbb{R}$ and an interval $[y] \in \mathbb{IR}^* \cup \{[\,]\}$ the extended interval subtraction is defined by

$$r - [y] := \begin{cases} [r - \overline{y}, r - \underline{y}], & \text{if} \quad [y] = [\underline{y}, \overline{y}] \in \mathbb{IR} \\ [-\infty, +\infty], & \text{if} \quad [y] = [-\infty, +\infty] \\ [r - \overline{y}, +\infty], & \text{if} \quad [y] = [-\infty, \overline{y}] \\ [-\infty, r - \underline{y}], & \text{if} \quad [y] = [\underline{y}, +\infty] \\ [\,], & \text{if} \quad [y] = [\,]. \end{cases}$$

This is already realized by the subtraction operator &− contained in `intpak`. Examples:

```
> 1 &- [1.,infinity];
```

$$[-\infty,\ 0]$$

```
> 1 &- [-infinity,1.];
```

$$[0,\ \infty]$$

```
> 1 &- [];
```

$$[]$$

```
> 1 &- [-infinity,infinity];
```

$$[-\infty,\ \infty]$$

4.2. Extended Interval-Newton-Iteration. The procedure `compute_all_zeros` computes confinements of all zeroes of a continuously differentiable function in a given entered start interval using the interval-Newton-iteration.

The input parameters of the procedure `compute_all_zeros` are

- the function `f` whose zeroes are supposed to be calculated,
- the start interval `xstart` of the iteration and
- the desired relative diameter `eps` of the zero-confinements to be calculated.

The relative diameter of a real interval is defined as follows

$$d_{rel}([x]) := \begin{cases} \dfrac{d([x])}{\langle [x] \rangle}, & \text{if } 0 \notin [x] \\ d([x]), & \text{otherwise.} \end{cases}$$

Hereby, $d([x])$ denotes the diameter and $\langle [x] \rangle$ the minimum absolute value of the interval $[x]$.

The order of the parameters mentioned above is compulsory. As an optional fourth parameter the desired accuracy, i. e. the value of the system variable `Digits` within the procedure can be entered. The fourth parameter should therefore be a positive integer greater than or equal to 10. If this fourth parameter is missing, then the accuracy is adapted to the relative accuracy needed and the length of the input-parameters, but in any case is greater than or equal to the current value of the variable `Digits`.

Given out is the accuracy used, the computed confinements of the zeroes, and for each confinement the information if existence and uniqueness of a zero in the given interval was proved.

If the calculated interval was only a *potential* confinement of a zero, then it may contain one, several or no zeroes of f at all.

The computed zero-confinements are stored in a global variable `zeros` and can therefore be used further in any way. `zeros` is a table and the access to an entry of the table is made in the usual way, e. g. using `zeros[2]`.

Further global variables initialized in the procedure are the table `infos` containing the additional informations, the number of calculated zero-confinements `N` and the step counter `iter_counter`.

Example 1. Calculating all zeroes of

```
> f:=x->2*exp(tan(cos(x))) - sin(x) + cos(2*x);
```

$$f := x \to 2\, e^{\tan(\cos(x))} - \sin(x) + \cos(2\,x)$$

in the interval $[0, 8]$.

The function has three simple zeroes in the interval $[0, 8]$ (see Figure 4). Two of them can be given exactly, namely $\frac{\pi}{2}$ und $\frac{5\pi}{2}$. An approximation of the third zero can be determined with the command `fsolve`.

Testing if $\pi/2$ and $5\pi/2$ are zeroes of f:

```
> f(Pi/2);
```

$$0$$

```
> f(5*Pi/2);
```

$$0$$

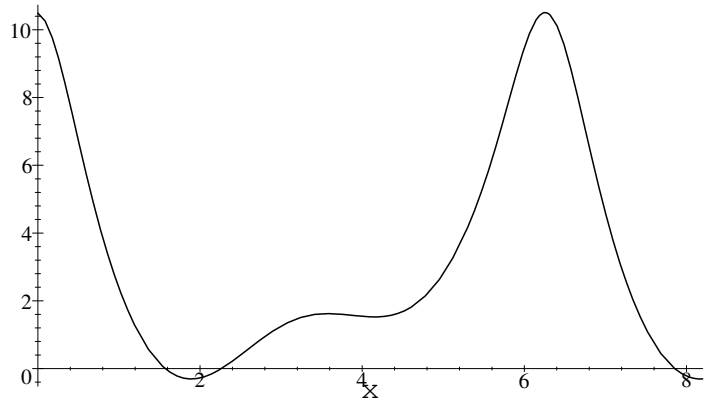


Fig. 4. illustration of the function $f(x) := 2\mathrm{e}^{\tan(\cos(x))} - \sin(x) + \cos(2x)$

Calculating the third zero with `fsolve`, `Digits=30`:

```
> Digits:=30:  zero3:=fsolve(f(x),x=2..2.5); Digits:=10:
```

$$zero3 := 2.264800742000049965058142861 26$$

Calculating the zero-confinements, `Digits=20` (fourth input parameter):

```
> compute_all_zeros(f,[0,8.],10^(-10),20);
Digits =    20
```

[7.8539816339705772082, 7.8539816339775304044]

contains exactly one zero

[2.2648007419999768034, 2.2648007420001249007]

contains exactly one zero

[1.5707963267948966180, 1.5707963267948966204]

contains exactly one zero

```
number of zero confinements:    3

number of iteration steps:    21
```

Verifying if the zeroes of f are contained in the calculated zero-confinements:

```
> is_in(evalf(Pi/2,30),zeros[3]);
```

true

```
> is_in(evalf(5*Pi/2,30),zeros[1]);
```

true

```
> is_in(zero3,zeros[2]);
```

true

Example 2. Confining the zero of

```
> f:=x->(x-1)^3;
```

$$f := x \to (x-1)^3$$

with start interval $[-3., 4]$. The relative diameter of the confinement should be $\leq 10^{-50}$. (The accuracy used is adapted automatically.)

```
> compute_all_zeros(f,[-3.,4.],10^(-50));
Digits =    55
```

[.999999999999999999999999999999999999999999999997208,
1.000000000000000000000000000000000000000000000005551]

potential zero confinement

```
number of zero confinements:    1

number of iteration steps:    163
```

Diameter and relative diameter of the calculated zero-confinement:

```
> Digits:=55:
> diam:=width(zeros[1]);
```

$$diam := .58302\,10^{-50}$$

```
> `relative_diam`:=rel_diam(zeros[1]);
```

$$relative\_diam := .583020000000000000000000000000000000000000000000001628\,10^{-50}$$

```
> Digits:=10:
```

4.3. Graphical Illustration. For graphical illustration of the interval-Newton-iteration, the procedure `compute_all_zeros_with_plot` is at the user's disposal. It calculates analogously to the procedure `compute_all_zeros` zero-confinements with the interval-Newton-iteration. Additionally, however, each iteration step is illustrated graphically.

Here, too, the value of the variable `Digits` can be entered as an optional fourth parameter. Further, the input of a fifth optional parameter is possible, stating how many iteration steps may be performed at most. If this fifth parameter is missing, then the maximum number of iteration steps must be entered interactively. The input must end with a colon or a semicolon.

Example. Calculating a zero-confinement of the function

```
> f:=x->exp(sin(x-1))-1;
```

$$f := x \to e^{\sin(x-1)} - 1$$

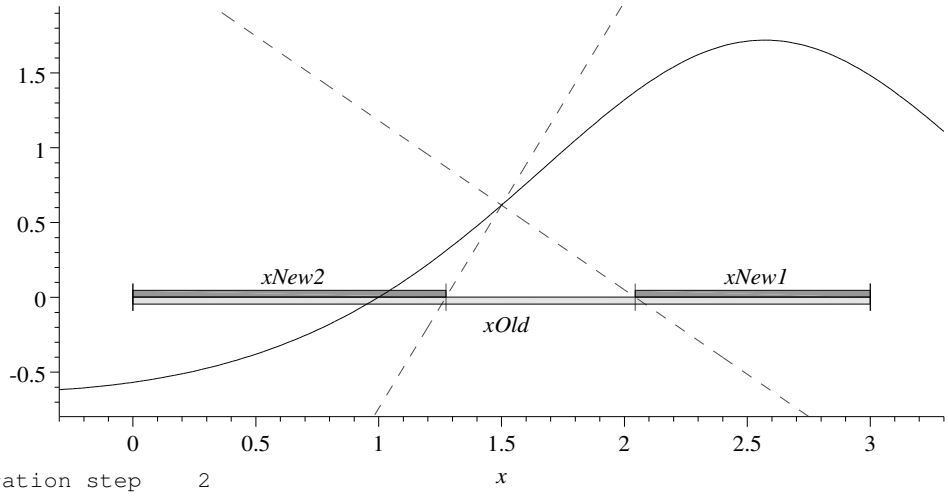
in the interval $[0, 3.]$:

```
> compute_all_zeros_with_plot(f,[0.,3.],10^(-3));
> 10;
```

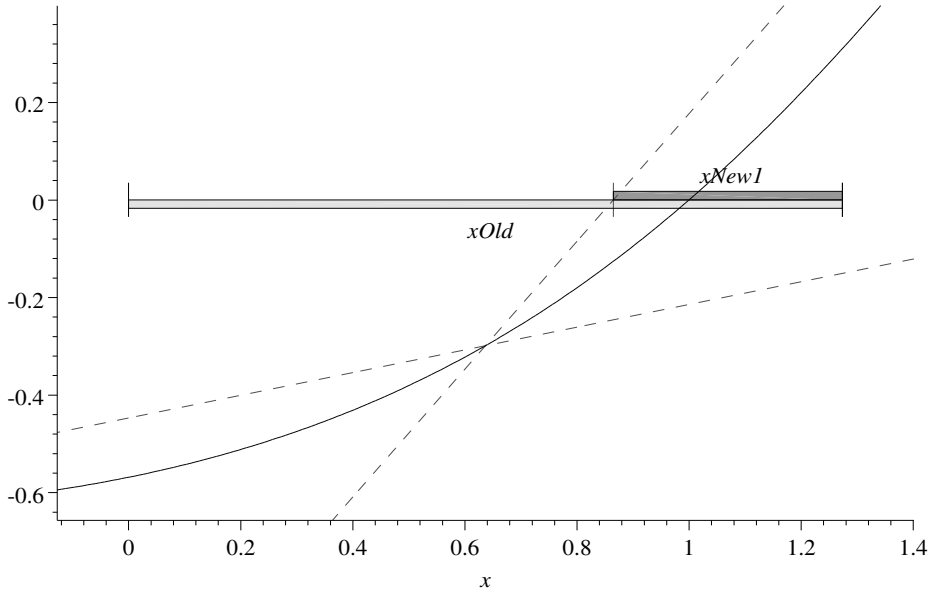
The output of the procedure call-up is found on the pages 490 and 491. The slopes of the dotted lines are given by the smallest resp. greatest slope of all tangents to the graph of the function in the current argument domain `xalt`. The lines intersect in the point (expansion point, f(expansion point)). The intersection points of these lines with the `x`-axis are important auxiliary quantities for determining the next iteration of the extended interval-Newton-iteration (see page 483).

```
Digits =    10

Iteration step    1
xOld=   [0, 3.]
xNew1=   [2.043797652, 3.]
xNew2=   [0, 1.273700327]
```

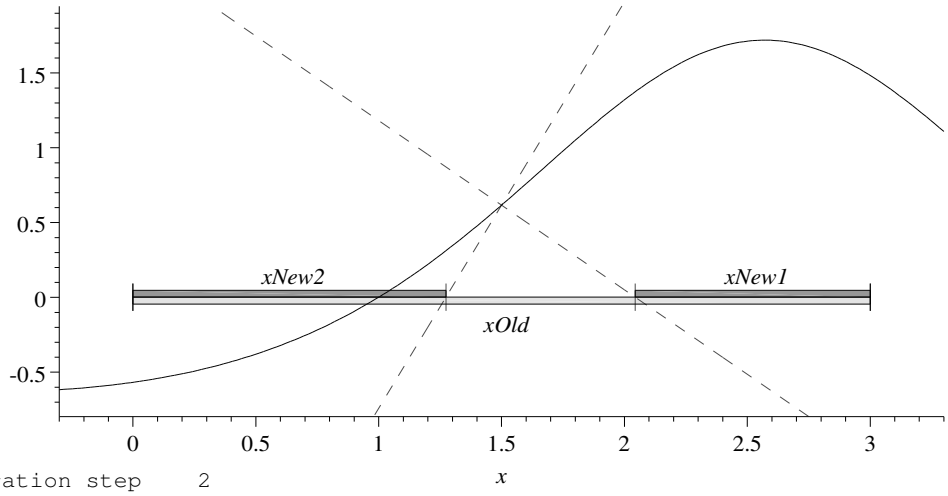



```
Iteration step    2
xOld=   [0, 1.273700327]
xNew1=   [.8650186701, 1.273700327]
```

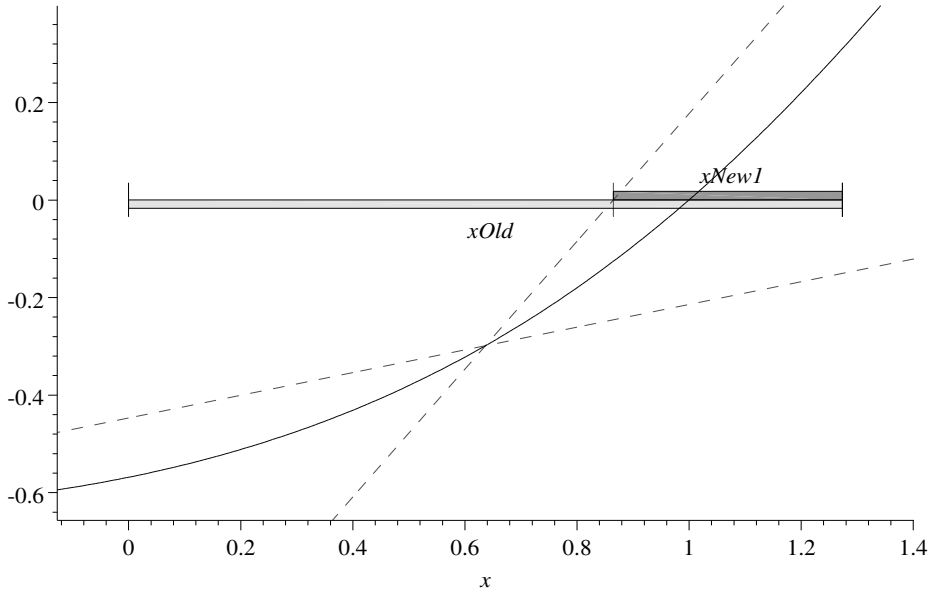


```
Digits =    10

Iteration step    1
xOld=   [0, 3.]
xNew1=   [2.043797652, 3.]
xNew2=   [0, 1.273700327]
```




```
Iteration step    2
xOld=   [0, 1.273700327]
xNew1=   [.8650186701, 1.273700327]
```



5. Complex Intervals and Operations (Disc Arithmetics). Based on the real interval operations, a complex interval arithmetic for disc intervals is defined and implemented in the extension `intpakX`.

A disc interval with centre $z_0 \in \mathbb{C}$ and radius $r > 0$

$$Z = \langle z_0,\, r\rangle := \{z \in \mathbb{C} \,|\, |z - z_0| \le r\}$$

is stored in `intpakX` as a list with three entries: real part of the centre z_0 of Z, imaginary part of z_0 and radius r.

The name of this new data type is `complex_disc`. As components of this new type numbers of type `numeric` are permitted, in particular numbers of type `integer` and of type `fraction`.

The procedure `complex_disc_plot` can be used for graphical illustration of a disc interval. It has as input parameter a variable of type `complex_disc`. As further optional parameters the usual illustration options of the Maple-command `plot` can be entered.

5.1. Arithmetic (Disc-)Operations. The basic arithmetic operations for disc intervals are usually defined (see e. g. [1]) as follows. Let $A = \langle a,\, r_a\rangle$ and $B = \langle b,\, r_b\rangle$ be two disc intervals. Then

$$\begin{aligned}
A + B &:= \langle a + b,\, r_a + r_b\rangle,\\
A - B &:= \langle a - b,\, r_a + r_b\rangle,\\
A \cdot B &:= \langle a \cdot b,\, |a| r_b + |b| r_a + r_a r_b\rangle,\\
1 \,/\, B &:= \left\langle \frac{\overline{b}}{b\overline{b} - r_b^2},\, \frac{r_b}{b\overline{b} - r_b^2} \right\rangle, \quad 0 \notin B,\\
A \,/\, B &:= A \cdot (1 \,/\, B), \quad 0 \notin B,
\end{aligned}$$

where $|a| = \sqrt{a_1^2 + a_2^2}$ denotes the absolute value of the complex number $a = a_1 + i\, a_2$ and $\overline{b} = b_1 - i\, b_2$ is the complex conjugate of $b = b_1 + i\, b_2$.

For the operations defined in this way, we have with the notations as

above

$$\begin{aligned} A \pm B &= \{a \pm b \,|\, a \in A,\, b \in B\}, \\ A \cdot B &\supseteq \{a \cdot b \,|\, a \in A,\, b \in B\}, \\ 1 \,/\, B &= \{1/b \,|\, b \in B\}, \\ A \,/\, B &\supseteq \{a/b \,|\, a \in A,\, b \in B\}, \end{aligned}$$

where the inclusions are distinct from the equalities in general.

The operations for disc intervals defined above are realized in the extension `intpakX` by the operators &`cadd`, &`csub`, &`cmult` and &`cdiv`. For the inversion, there is no extra operator.

Area-Optimal Multiplication and Division. The (so-called **centred**) multiplication of two disc intervals $A = \langle a,\, r_a \rangle$ and $B = \langle b,\, r_b \rangle$ defined above delivers, for a given centre $a \cdot b$, an optimal confinement of the resulting point complex $\{\alpha \cdot \beta \,|\, \alpha \in A,\, \beta \in B\}$. This confinement, however, is not area-optimal.

Determining an area-optimal confinement of the resulting set under multiplication of two disc intervals is more tedious and leads to solving an equation of third degree (see [14]).

For $A = \langle a,\, r_a \rangle$ and $B = \langle b,\, r_b \rangle$, the **area-optimal** multiplication is defined as

$$\begin{aligned} A \cdot_{opt} B := \quad \langle ab\,(1 + x_0), \quad & (3\,|ab|^2 {x_0}^2 \\ & +2\,(|ab|^2 + |r_a b|^2 + |r_b a|^2)\, x_0 \\ & +|r_a b|^2 + |r_b a|^2 + (r_a r_b)^2\,)^{\frac{1}{2}} \rangle \end{aligned}$$

where x_0 is the non-negative zero of the polynomial

$$P(x) \;=\; 2\,|ab|^2 x^3 + (|ab|^2 + |r_a b|^2 + |r_b a|^2) x^2 - r_a^2 r_b^2$$

if $\texttt{grad}(P) \geq 2$ (otherwise, we set $x_0 = 0$).

The area-optimal division of two disc intervals is then defined as

$$A /_{opt} B := A \cdot_{opt} (1 / B).$$

The package `intpakX` contains a realization of the area-optimal multiplication (&`cmult_opt`) and the area-optimal division (&`cdiv_opt`) of two disc intervals.

General Procedure when Implementing the Basic Operations. Let A and B be two disc intervals, which can be displayed on the calculator exactly, and let $* \in \{+, -, \cdot, /\}$. In order to obtain a safe confinement C on the machine of the exact result complex $A * B$, during the implementation it was proceeded as follows:

1. Calculate a real machine interval cx confining the real part of the centre of the result interval, and a real machine interval cy confining the imaginary part.
2. Calculate the radius r of the resulting circle as:

$$\begin{aligned} r1 &:= \text{sup(formula for the radius evaluated with intervals)} \\ r2 &:= \triangle(r1 + d(cx)) \\ r &:= \triangle(r2 + d(cy)) \end{aligned}$$

   where $\triangle$ denotes the rounding from above and $d(cx)$, $d(cy)$ the diameter of cx resp. cy.
3. Let $C = \langle m(cx) + i \cdot m(cy),\, r\rangle$. $m(cx)$ and $m(cy)$ denote the centre of cx resp. cy.

In order to determine the centre of an interval, the procedure `mid` is used. In contrast to the `intpak`-procedure `midpoint`, it does not calculate a confinement of the centre of an interval, but a number (approximation of the centre of the interval) lying certainly within the entered interval.

A Numerical Example. For $A = \langle 1, 1\rangle,\ B = \langle -1 + i, 1\rangle$ we have

$$\begin{aligned} A + B &= \langle i, 2\rangle \\ A - B &= \langle 2 - i, 2\rangle \\ 1 \,/\, B &= \langle -1 - i, 1\rangle \end{aligned}$$

Calculation with Maple

```
> A:=[1,0,1]:  B:=[-1,1,1]:
> A &cadd B;
```

$$[0,\ 1.000000000,\ 2.000000005]$$

```
> A &csub B;
```

$$[2.000000000,\ -1.000000000,\ 2.000000007]$$

```
> 1 &cdiv B;
```

$$[-1.000000001,\ -1.000000001,\ 1.000000051]$$

Using centred multiplication (see p. 492) one obtains

$$A \cdot B = \langle -1 + i, 2 + \sqrt{2} \rangle \approx \langle -1 + i, 3.414213562 \rangle$$
$$A \,/\, B = \langle -1 - i, 2 + \sqrt{2} \rangle$$

```
> A &cmult B;
```

$$[-1.000000000,\ 1.000000000,\ 3.414213579]$$

```
> A &cdiv B;
```

$$[-1.000000001,\ -1.000000001,\ 3.414213685]$$

and using area-optimal multiplication leads to

```
> A &cmult_opt B;
```

$$[-1.390388204,\ 1.390388204,\ 2.969562256]$$

```
> A &cdiv_opt B;
```

$$[-1.390388219,\ -1.390388219,\ 2.969562345]$$

In Figure 5 are displayed simultaneously the resulting point set of the product of A and B, the centred confinement and the area-optimal confinement of the set.

6. Application III: Range of Complex Polynomials. A first possible application of the disc arithmetic defined in `intpakX` is the determining of safe confinements for the range of a polynomial with complex coefficients over a disc interval.

For this, there are three procedures

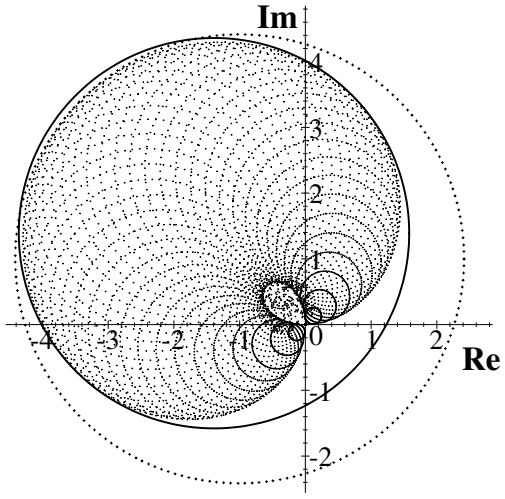


Fig. 5. Centred and area-optimal confinement of $\langle 1, 1\rangle \cdot \langle -1 + i, 1\rangle$

1. `horner_eval_cent` (Horner-scheme using centred multiplication &`cmult`),
2. `horner_eval_opt` (Horner-scheme using area-optimal multiplication &`cmult_opt`),
3. `centred_form_eval` (centred form for complex polynomials).

The procedures `horner_eval_opt` and `centred_form_eval` generally give certainly better confinements than the procedure `horner_eval_cent`. They have, however, a substantially higher demand for time and memory.

The first input parameter of each procedure is a (complex) polynomial in the variable z. The denomination of this variable is compulsory! As a second parameter a number or variable of type `complex_disc` must be entered.

Example 1. Confining the range of

$$\begin{aligned} p(z) \;:=\; & (0.15 - 0.1i) + (0.15 - 0.12i)z + (-0.2 - 0.2i)z^2 \\ & +(0.1 + 0.3i)z^3 + (0.1 - 0.2i)z^4 + (0.1 - 0.2i)z^5 \\ & +(0.2 - 0.2i)z^6 + (0.1 - 0.2i)z^7 + (0.2 - 0.1i)z^8 \\ & +(0.1 - 0.1i)z^9 \end{aligned}$$

over the interval $Z = \langle -0.1 + 0.2i,\ 0.9 \rangle$.

```
> p_H:=horner_eval_cent(p,Z);
```

$$p\_H := [.1590115281,\ -.04050517670,\ 3.058832329]$$

```
> p_Hopt:=horner_eval_opt(p,Z);
```

$$p\_Hopt := [.2219721872,\ .2917174855,\ 2.243412729]$$

```
> p_C:=centred_form_eval(p,Z);
```

$$p\_C := [.1590115281,\ -.04050517670,\ 1.717944237]$$

In this example the procedure `centred_form_eval` gives the best confinement. The graphical illustration of the range of p over Z and the calculated confinements can be found in Figure 6.

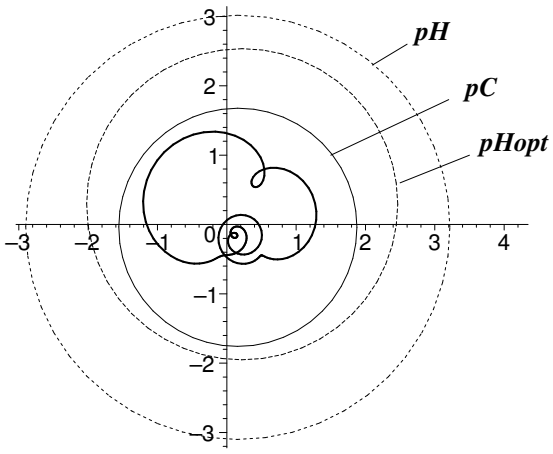


Fig. 6. Confinement of the range of a complex polynomial

Generating the graphics:

For illustrating the range of p over Z the command `complexplot` from the `plots`-package can be used. Each command should be ended by a colon (otherwise very much, generally unnecessary, information is given out).

```
> c1:=complexplot(subs(z=Z[1]+I*Z[2]+Z[3]*(cos(t)+I*sin(t)),p),
                  t=0..2*Pi,color=black,thickness=3,numpoints=200):
```

For graphical illustration of the computed confinements the command `complex_disc_plot` was used.

```
> c2:=complex_disc_plot(p_H,color=black,thickness=2,
      linestyle=4):
> c3:=complex_disc_plot(p_Hopt,color=black,thickness=2,
      linestyle=3):
> c4:=complex_disc_plot(p_C,color=black,thickness=2):
```

The labelling of the graphic was generated with a series of `plot`- and `textplot`-commands. `textplot` is a command from the `plots`-Package.

```
> s1:=plot([[2.1,2.3],[2.6,2.6]],color=black,thickness=2):
> t1:=textplot([3,2.8,pH],font=[TIMES,BOLDITALIC,12]):
> s2:=plot([[1.5,1],[3.2,1.8]],color=black,thickness=2):
> t2:=textplot([3.7,2,pC],font=[TIMES,BOLDITALIC,12]):
> s3:=plot([[2.5,0.6],[3.5,1]],color=black,thickness=2):
> t3:=textplot([4.2,1.2,pHopt],font=[TIMES,BOLDITALIC,12]):
```

For simultaneous display of the generated graphics the command `display` from the `plots`-package can be used.

```
> display([c1,c2,c3,c4,s1,s2,s3,t1,t2,t3],scaling=constrained);
```

Example 2. Confinement of the value of

$$p(z) := (z-i)^4(z-1-i)^5$$

at the place $z = 1$. The exact value is $p(1) = 4i$.

```
> subs(z=1,p);
```

$$4\,I$$

Calculation of confinements using the disc arithmetic:

```
> Digits:=60:
> horner_eval_cent(p,1);
```

$$[0,\ 4.000000000000000000000000000000000000000000000000000000000000,$$
$$.28131000000000000000000000000000000000000000000000000000000045\,10^{-55}]$$

```
> horner_eval_opt(p,1);
```

$$[0,\ 4.000000000000000000000000000000000000000000000000000000000000000000000000000000,$$
$$.28131000000000000000000000000000000000000000000000000000000000000000000000000045\,10^{-55}]$$

```
> centred_form_eval(p,1);
```

$$[0,\ 4.000000000000000000000000000000000000000000000000000000000000000000000000000000,$$
$$.11254200000000000000000000000000000000000000000000000000000000000000000000000010\,10^{-54}]$$

All three procedures give out as confinement of $p(1)$ a disc interval with centre $4i$ and the radii of the computed confinements are around 10^{-55}, if one calculates with 60 digits (`Digits = 60`).

7. The Exponential Function for Disc Intervals. The image of a disc interval $Z = \langle c,\, r\rangle$ under the exponential function is in general not a disc. If as centre of the resulting interval the point $\exp(c)$ is prescribed, then

$$\exp(Z) := \langle \mathsf{e}^c,\ |\mathsf{e}^c|(\mathsf{e}^r - 1)\rangle$$

defines an optimal confinement (under given centre $\exp(c)$ of the resulting point complex $\{\exp(z) \,|\, z \in Z\}$).

A detailed discussion of images of disc intervals under the exponential function is found in [4].

The realization of the exponential function for disc intervals `cexp` from the extension `intpakX` has as input parameter a variable of type `complex_disc` or a complex number and gives a safe confinement of the resulting point complex.

Example. Confinement of the image of $Z = \langle 0,\, \pi + 1\rangle$ under the exponential function

```
> Cexp:=cexp([0,0,evalf(Pi+1)]);
```

$$Cexp := [1.000000000,\ 0,\ 61.90292461]$$

Graphical illustration of the computed disc interval confinement:

```
> c1:=complex_disc_plot(Cexp,color=black,thickness=3,
      numpoints=400):
```

Image of the boundary of $\langle 0,\ \pi{+}1\rangle$ under the exponential function (Attention: `plots` must be loaded beforehand with `with`!):

```
> c2:=complexplot(exp(polar(Pi+1,phi)),phi=0..2*Pi,
                          color=black,thickness=3,numpoints=400):
```

For illustrating inner points of the range, the radius r is varied from 0 to $\pi + 1$ for fixed angle. Example:

```
> c2:=complexplot(exp(polar(r,0.5)),r=0..Pi+1,
                          color=black,thickness=3,numpoints=400):
```

The different graphic commands are bundled afterwards with `display`. In display, an additional option `scaling=constrained` should be entered. In order to display a cut-out piece, the `plot`-option `view` was used.

The graphical illustration in Figure 7 shows that the centred confinement of the image of $\langle 0,\ \pi{+}1\rangle$ under the exponential function is the optimal confinement

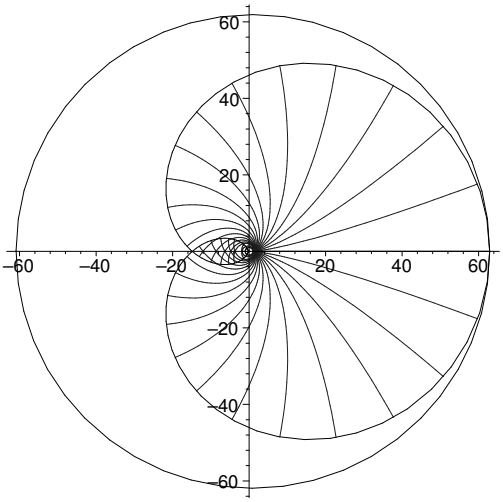


Fig. 7. Image of the disc interval $\langle 0,\ \pi + 1\rangle$ under the exponential function and centred disc interval confinement of $\exp(\langle 0,\ \pi + 1\rangle)$

for prescribed result-centre $\exp(0) = 1$. It is far from area-optimal. Figure 8 shows a cut out piece around the origin.

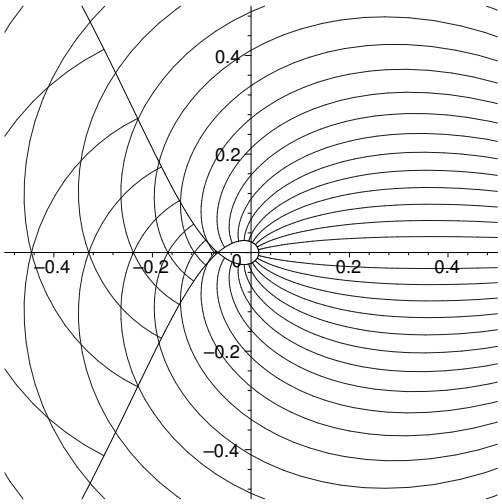


Fig. 8. Illustration of a cut-out piece around the origin

The exponential function is 2π`i`-periodic, and as the diameter of the interval $Z = \langle 0, \pi + 1 \rangle$ is greater than 2π, there is an area in the image Z of the exponential function in which every point has exactly two preimages. This area can be recognized by its double hatching. Besides, in the illustration of the cut out piece the drop-like image-free area around the origin can be observed.

8. Conclusion and Outlook. Using verification algorithms it is possible, if occasion arises, to prove with the computer automatically the existence and uniqueness of a solution for a given problem, and also to compute a (narrow) confinement of the exact solution. The results obtained in this way have the same mathematical quality as results obtained e. g. by using computer algebra systems, i. e. by automatic formula manipulations. Thereby it turns out to be a great advantage that verification algorithms can handle safely numerical input-data with errors within tolerance. In such cases infinitely many problems are solved simultaneously. For an entire family of problems it is e. g. proved that each one has a unique solution.

Whenever computer algebra packages make use of numerical routines (e. g. when computing determined integrals), a verification algorithm should be used if possible (e.g., [2]). The results obtained are then mathematically safe (a

property usually expected when working with a CA-system). Pretended solutions resp. far-off approximations are then excluded.

Also by using interval methods, the graphic abilities of computer algebra systems can be improved resp. made secure. That this is necessary, is impressively shown by [15, 13]. See also [3, 7, 16, 20].

Computer algebra and verification numerics complete each other ideally. Thus the computer becomes for the mathematician, but also for the engineer, a safe mathematical tool. Especially in view of processors which are becoming faster and faster and more powerful, the symbiosis of symbolic calculus and safe numerical routines [1, 17, 10] should be pushed forward massively.

`intpakX` is also intended to be a valuable didactical tool in teaching self-verifying (numerical) methods. This tool allows learning by experience. Because the source code is publicly available, it should be easy to extend the functionality of the package by students/users. The multiple precision interval arithmetic [9] realized in `intpakX` allows the implementation of algorithms as described in [22, 21].

REFERENCES

[1] Alefeld G., J. Herzberger. Introduction to Interval Computations. New York, Academic Press, 1983.

[2] Bauknecht T. Verifizierte numerische Quadratur in Maple. Diplomarbeit (Betreuer: W. Krämer), Univ. Karlsruhe, 1997.

[3] Bolz U. Verifizierte graphische Darstellung reeller Funktionen. Diplomarbeit (Betreuer R. Lohner), Univ. Karlsruhe, 1996.

[4] Börsken N. C. Komplexe Kreis-Standardfunktionen. Diplomarbeit, Univ. Freiburg, 1978.

[5] Connell A. E., R. M. Corless. An Experimental Interval Arithmetic Package in Maple. Tex-Document distributed with the Maple Share Library, 1993.

[6] Geulig I. Computeralgebra und Verifikationsalgorithmen. Diplomarbeit (Betreuer: W. Krämer), Univ. Karlsruhe, 1998.

[7] Geulig I., W. Krämer. Intervallrechnung in Maple – Die Erweiterung intpakX zum Paket intpak der Share-Library. Preprint 99/2 des Instituts für Wissenschaftliches Rechnen und Mathematische Modellbildung (IWRMM), Universität Karlsruhe, 1999.

[8] Grimmer M. Interval Arithmetic in Maple with intpakX. *Proceedings in Applied Mathematics and Mechanics* **2** (2003), No 1, 442–443.

[9] Grimmer M., K. Petras, N. Revol. Multiple Precision Interval Packages: Comparing Different Approaches. In: Numerical Software with Result Verification, Lecture Notes in Computer Science, Volume 2991/2004, Springer-Verlag, Heidelberg, 2004, 64–90.

[10] Hammer R., M. Hocks, U. Kulisch, D. Ratz. Numerical Toolbox for Verified Computing I. Berlin, Heidelberg, Springer-Verlag, 1993.

[11] Kofler M. Maple: An Introduction and Reference. Addison-Wesley, 1997.

[12] Krämer W. Computeralgebra und Verifikationsalgorithmen I und II. Vorlesungen im WS 96/97 bzw. SS 97, Univ. Karlsruhe.

[13] Krämer W. Pitfalls in Maple. Preprint BUW-WRSWT 2006/7, Universität Wuppertal.

[14] Krier N. Komplexe Kreisarithmetik. Dissertation, Univ. Karlsruhe, 1973.

[15] Lerch M., G. Tischler, J. Wolff Von Gudenberg, W. Hofschuster, W. Krämer. FILIB++, a fast interval library supporting containment computations. *ACM Trans. Math. Software* **32** (2006) No 2, 299–324.

[16] Lohner R. Private communication about questionable Maple results, 1998.

[17] Neumaier A. Interval Methods for Systems of Equations. Cambridge, Cambridge University Press, 1990.

[18] Ratschek H. J. Rokne. Computer Methods for the Range of Functions. Chichester, West Sussex, England, Ellis Horwood Limited, 1984.

[19] Ratz D. Inclusion Isotone Extended Interval Arithmetic. Bericht 5/96, Institut für Angewandte Mathematik, Univ. Karlsruhe, 1996.

[20] Popova E. Web-accessible tools for interval linear systems. *Proceedings in Applied Mathematics and Mechanics* **5** (2005), No 1, 713–714.

[21] Rogat A. Schnelle hoch genaue Einschließung von Werten arithmetischer Ausdrücke mit beliebiger vorgegebener Genauigkeit. Dissertation, Universität Wuppertal, 2002.

[22] Steins A. Verifizierte Formelauswertung in Computer-Algebra-Systemen. Dissertation, Universität Wuppertal, 1996.

[23] `intpakX` link at the University of Wuppertal:
`http://www.math.uni-wuppertal.de/~xsc/software/intpakX/`

[24] Maplesoft: `http://www.maplesoft.com/applications/app_center_browse.aspx?CID=13&SCID=155`

University of Wuppertal
42119 Wuppertal, Germany
e-mail: `kraemer@math.uni-wuppertal.de`

Received December 4, 2006
Final Accepted October 18, 2007