# (The Future of) C-XSC

Walter Krämer, Werner Hofschuster

Scientific Computing/ Software Engineering

Department of Mathematics

University of Wuppertal

Germany

# Contents

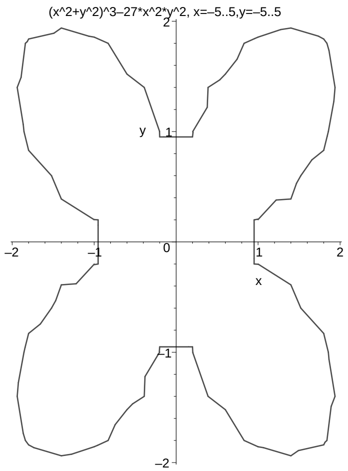

- Intervals may be a must (motivation)
- Research group in Wuppertal
- The class library C-XSC
  - History and some data
  - Main features
  - Simple applications and sample code
  - Toolbox with Verification Algorithms
  - Additional software based on C-XSC
- What can we learn from IntLab?
- What can we learn from Sun Forte C++ with interval support?
- The future of C-XSC

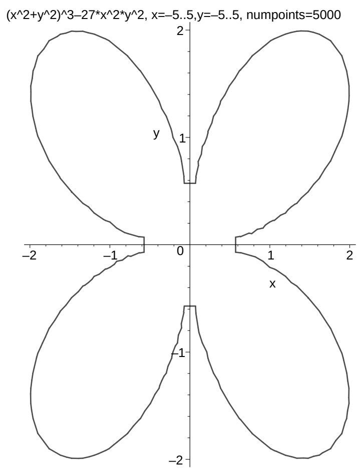
# Interval Comp. may be a Must

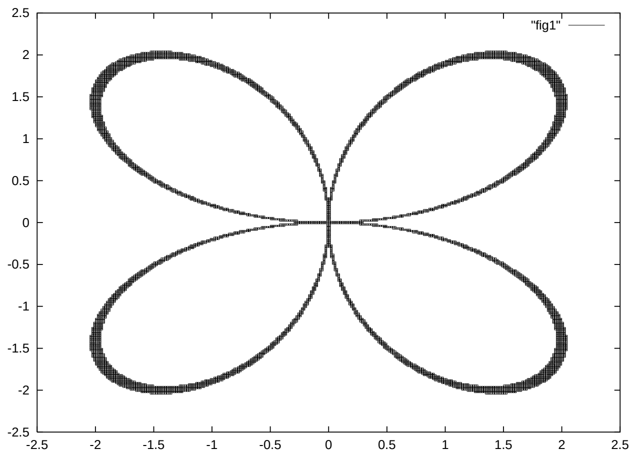
The problem in plotting the following implicitly defined simple curves is not the lack of precision in the computations. The erroneous results come from the interpolation process. At least near intersection points or near singularities this process is difficult to control automatically

Plotting the curve defined by $(x^2 + y^2)^3 - 27x^2y^2 = 0$ using Maples `implicitplot`-command

using interval computations

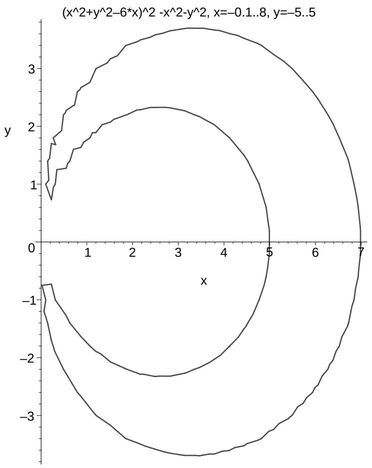


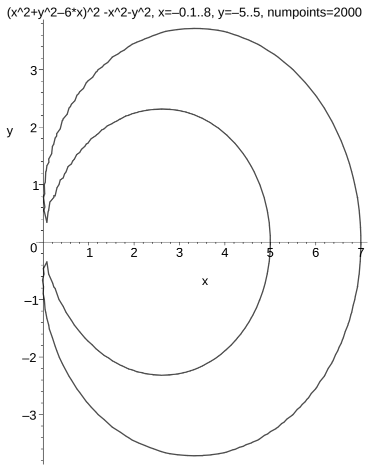



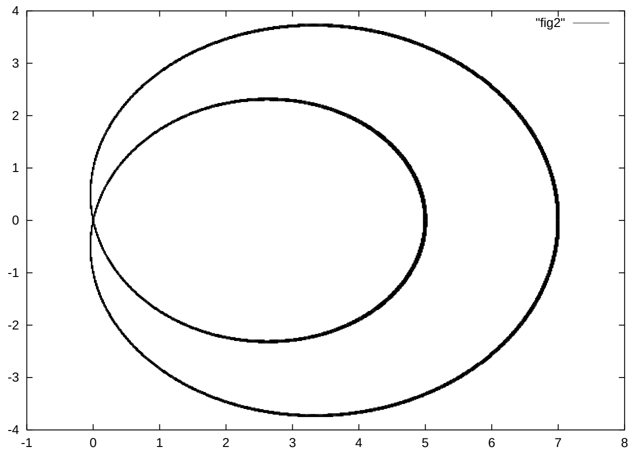



Plotting $(x^2 + y^2 - 6x)^2 - x^2 - y^2 = 0$ using Maples

`implicitplot`-command | using interval tools

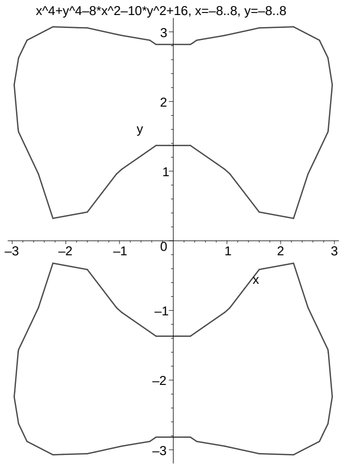


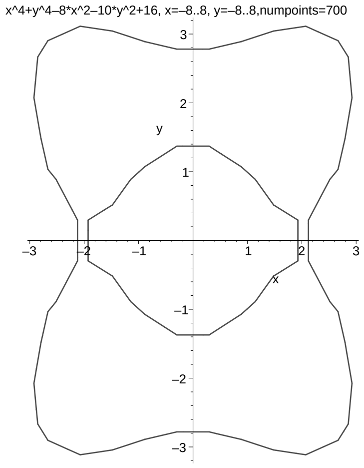



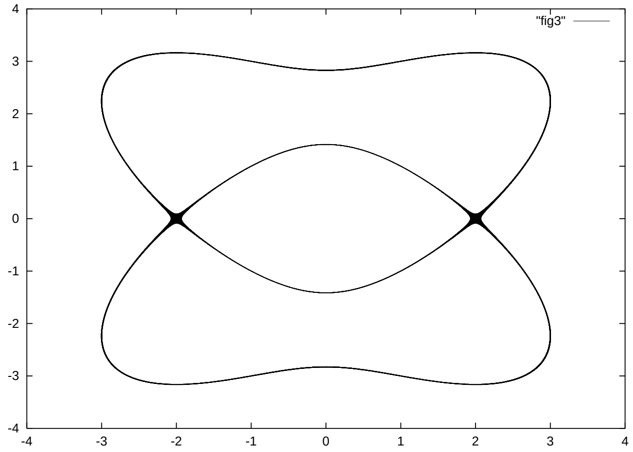
Plotting $x^4 + y^4 - 8x^2 - 10y^2 + 16 = 0$ using Maples

`implicitplot`-command




using interval tools



# C-XSC Research Group

Research group actually developing C-XSC 2.0 in Wuppertal:

- Markus Grimmer
- Werner Hofschuster
- Walter Krämer

# C-XSC

Cooperation: IAM Uni Karlsruhe, WRSWT Uni Wuppertal

Duration: about 10 years

Freely available: Betarelease C-XSC 2.0

Source code of C-XSC (without Toolbox):

- 45 classes
- 155 files (70 hpp-files, 43 inl-files, 42 cpp-files)
- Additionally RTS: 44 h-files, 715 c-files
- Additionally FILIB: 1 h-file, 67 c-files

Ca. 274 000 lines of code in ca. 1000 files

# C-XSC Highlights

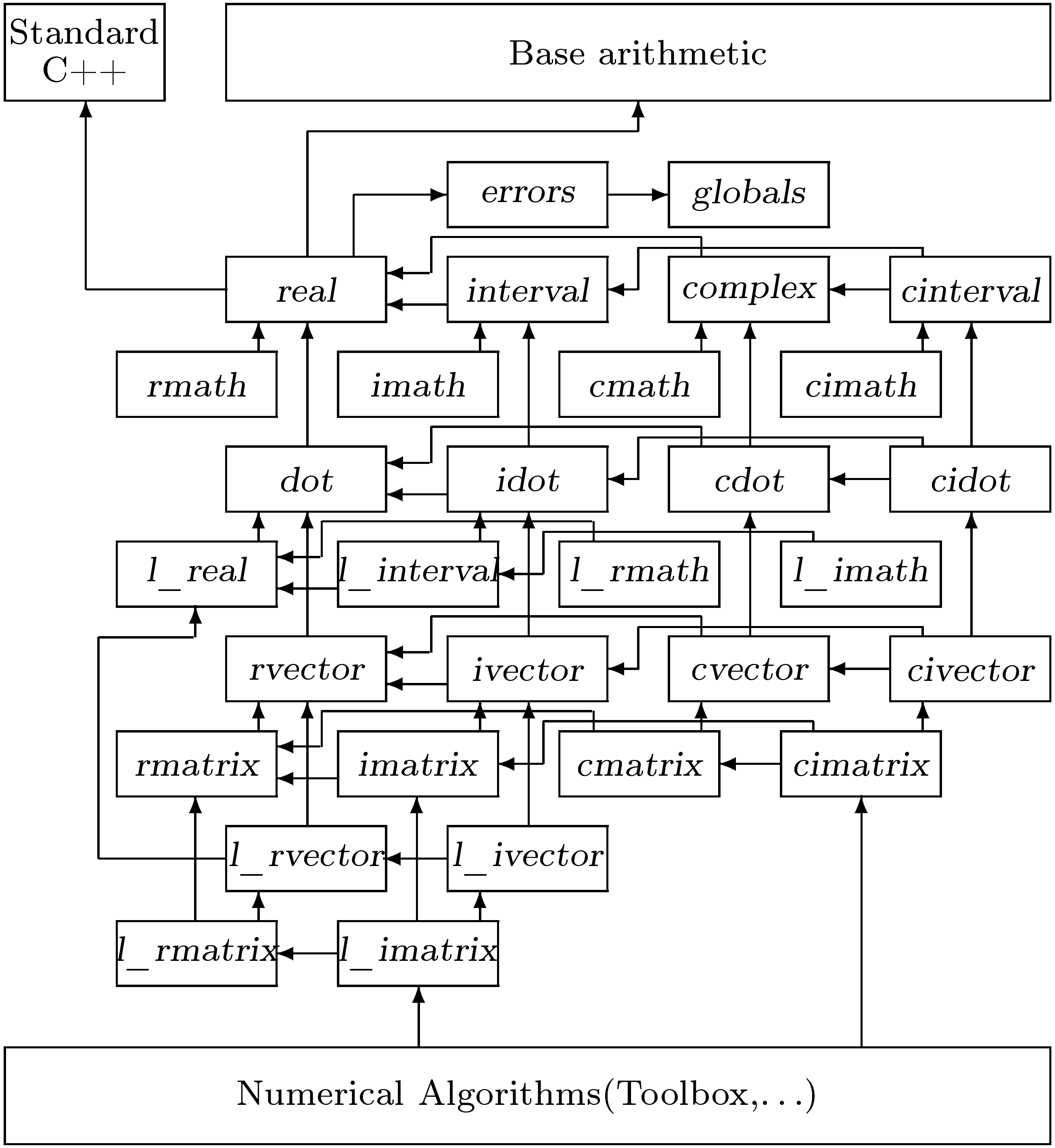
Tool for the development of numerical algorithms delivering highly accurate and automatically verified results

- Real, complex, interval, and complex interval arithmetic
- Dynamic vectors and matrices
- Subarrays of vectors and matrices
- Dotprecision data types
- Predefined arithmetic operators with highest accuracy
- Standard functions of high accuracy
- Dynamic multiple-precision arithmetic and standard functions
- Rounding control for I/O data
- Library of problem solving routines

# Numerical Data Types

C-XSC offers the classes:

- `real`, `interval`, `complex`, `cinterval` (complex interval)
- `rvector`, `ivector`, `cvector`, `civector`,
- `rmatrix`, `imatrix`, `cmatrix`, `cimatrix`,
- `dotprecision`, `idotprecision`, `cdotprecision`, `cidotprecision`,
- `l_real` (staggered correction), `l_interval`, `l_rvector`, `l_ivector`, `l_rmatrix`, `l_imatrix`,
- `intvector`, `intmatrix`


Standard C++
Base arithmetic
errors
globals
real
interval
complex
cinterval
rmath
imath
cmath
cimath
dot
idot
cdot
cidot
l_real
l_interval
l_rmath
l_imath
rvector
ivector
cvector
civector
rmatrix
imatrix
cmatrix
cimatrix
l_rvector
l_ivector
l_rmatrix
l_imatrix
Numerical Algorithms(Toolbox,...)

# Subarrays of vectors and matrices

- subarray: rectangular part of an array
- Access with ()- or [ ] operator
- () operator results in an object of the same type as the original object
- [ ] generates a subarry of "lower" type
- $A$ `rmatrix`, then $A[i]$ is the $i-$th row (`rvector`) and $A[i][j]$ the $(i,j)-$th element (`real`)
- A$[k](i,j)$ is a subvector from index $i$ to $j$ of the $k-$th row vector

# Dot Product Expressions

$$x_i, y_i \in G = G(b, l, \underline{e}, \overline{e}), i = 1(1)n, \qquad \Box \left( \sum_{i=1}^{n} x_i \cdot y_i \right)$$

Such expressions are computed with **maximum accuracy**!

Data types:

`dotprecision`, `cdotprecision`,

`idotprecision`,`cidotprecision`

(Intermediate) results of dot product expressions may be stored in dotprecision variables **without loss of information**

## System of Linear Equations $Ax = b$

$$
\begin{array}{rcl}
R := \tilde{A}^{-1} & \approx & A^{-1} \\
& \tilde{x} & \text{approximation} \\
& y & \text{correction of } \tilde{x}
\end{array}
$$

$$
\begin{aligned}
Ax &= b \\
A(\tilde{x} + y) &= b \\
0 &= b - A\tilde{x} - Ay \\
\tilde{A}y &= (b - A\tilde{x}) + (\tilde{A} - A)y \\
y &= \underbrace{R(b - A\tilde{x})}_{=:Z} + \underbrace{(I - RA)}_{=:C} y =: f(y)
\end{aligned}
$$

Interval vector $X$, $\quad Y := f(X) \subseteq \overset{\circ}{X} \quad \overset{\text{Brouwer}}{\Longrightarrow} \quad A^{-1}b \in \tilde{x} + Y$

## $I - RA$ using C-XSC

Every component may be computed as a dotproduct with **maximum accuracy**

```
imatrix C(n,n);     // Interval matrix
dotprecision Accu; // Long accumulator

for (i=1; i<=n; i++) {
   for (j=1; j<=n; j++) {
      Accu = (i==j) ? 1.0 : 0.0;
      accumulate(Accu, -R[i], A[Col(j)]);
      rnd(Accu, C[i][j]);
   }
}
```

# A Typical Numerical Example

$$Ax = b \text{ mit } a_{ij} = \binom{n+i-1}{i-1}\binom{n-1}{n-j}\frac{n}{i+j-1},\ n = 10$$

```
Naive floating-point approximation:     exact:
 3.769740075654227E-010                   0
 9.999999949022040E-001                   1
          ...
 8.999978443156579E+000                  n-1

Verified solution found in:
[ 0.000000000000000, 0.000000000000000]
[ 1.000000000000000, 1.000000000000000]
            ...
[ 9.000000000000000, 9.000000000000000]
```

## Interval Newton Method for Continuously Differentiable $f$

Interval $X \subseteq D_f, \ \mathrm{midX} \in X, \ 0 \not\ni f'(X)$

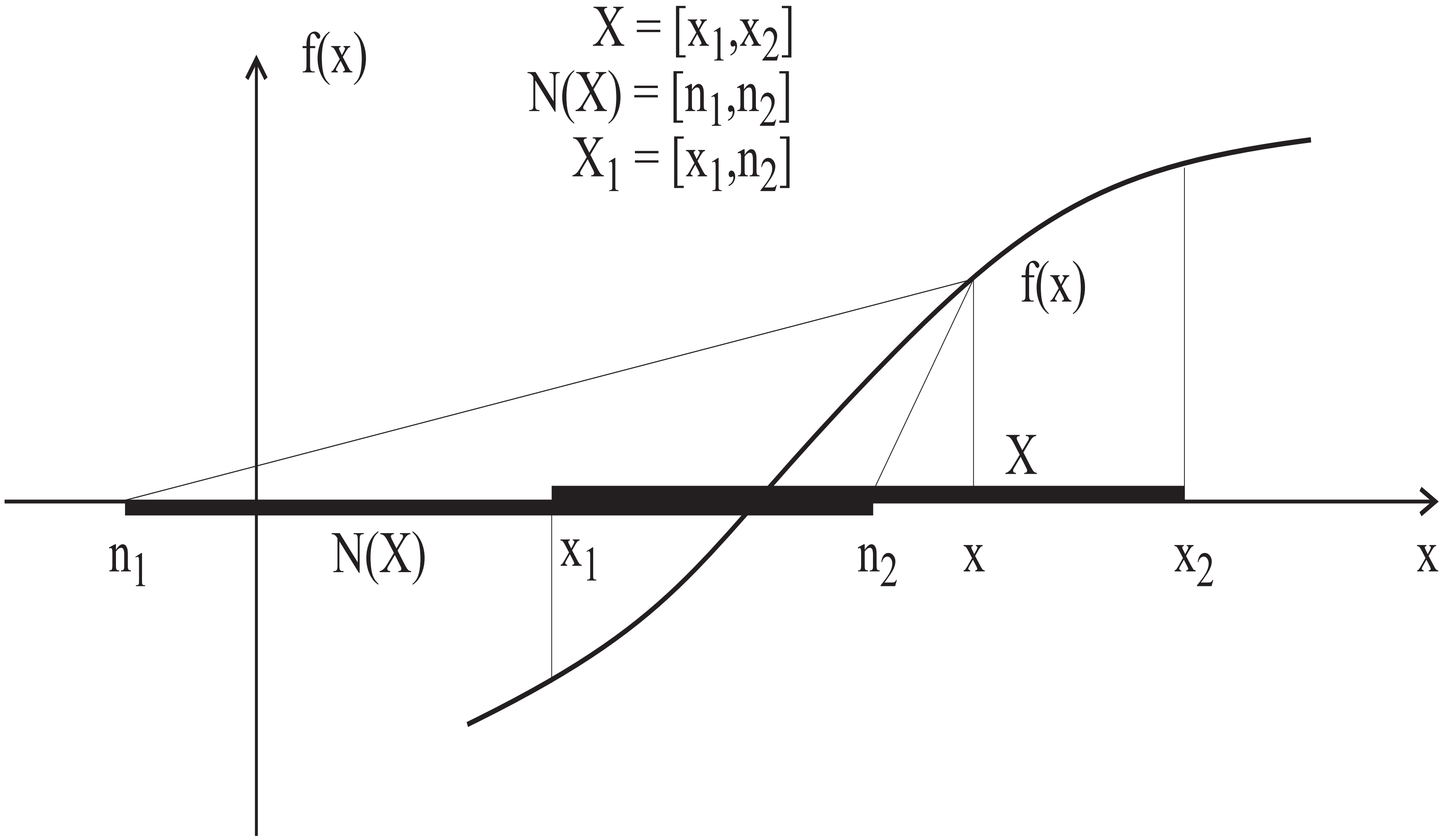
Interval Newton operator

$$N(X) := \mathrm{midX} - f(\mathrm{midX})/f'(X)$$

a) If $N(X) \subseteq X$, then $f(x)$ has exactly one zero $x^*$ in $X$

b) If $N(X) \cap X = \emptyset$, then $f(x)$ has no zero in $X$

c) If zero $x^* \in X_n$, then $x^* \in X_{n+1} := N(X_n) \cap X_n$



X = [x1,x2]
N(X) = [n1,n2]
X1 = [x1,n2]
f(x)
f(x)
X
n1
N(X)
x1
n2
x
x2
x

# Interval Newton Method in C-XSC

```
#include "interval.hpp"              // interval arithmetic
#include "imath.hpp"                 // interval functions
using namespace cxsc;

interval f(const real x)
{
  interval y(x);                     // y is a thin interval
  return sqrt(y) + (y+1)*cos(y);     //   initialized by x
}

interval deriv(const interval& x)
{ return 1/(2*sqrt(x)) + cos(x) - (x+1)*sin(x); }
```

Note the mathematical notation of interval expressions

```
bool criter(const interval& x)
{
   return Sup( f(Inf(x))*f(Sup(x)) ) < 0.0
     &&  !(0.0 <= deriv(x)); // '<=' means 'element of'
}

int main(void)
{
   interval x, xOld;
   x= interval(2,3);
   cout << "Starting interval is [2,3]" << endl;
   if (criter(x))
   {
      do {
         xOld = x;
         cout << "Actual enclosure is " << x << endl;
         x = (mid(x)-f(mid(x))/deriv(x)) & x;
      } while (x != xOld);
   } ...
```

```
f(x) = sqrt(x) + (x+1)*cos(x)

Starting interval is [2,3]
Actual enclosure is [   2.0                 ,   3.0               ]
Actual enclosure is [   2.0                 ,   2.2               ]
Actual enclosure is [   2.05                ,   2.06              ]
Actual enclosure is [   2.05903             ,   2.05905           ]
Actual enclosure is [   2.059045253413      ,   2.059045253416    ]
Actual enclosure is [   2.059045253415142,   2.059045253415145]
```

# Verified Graphical Results

## Level curves

**Given:** $f : \mathbb{R}^2 \longrightarrow \mathbb{R}, \;\; c \in \mathbb{R}, \;\; \text{Box } X \subset \mathbb{R}^2$

**To do:** plot all points $x \in X$ with $f(x) = c$
or with $f(x)$ in some given interval

Example (Pascalsche Schnecke):

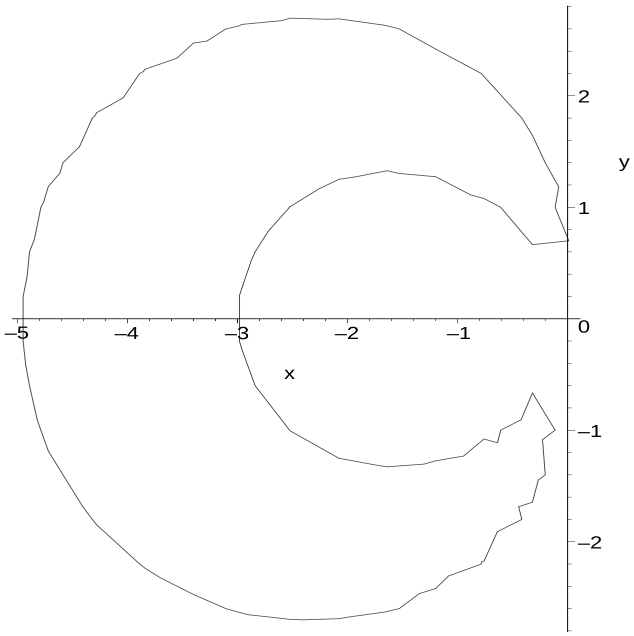
$$f(x, y) = (x^2 + y^2 + 4x)^2 - x^2 - y^2 \stackrel{!}{=} 0$$

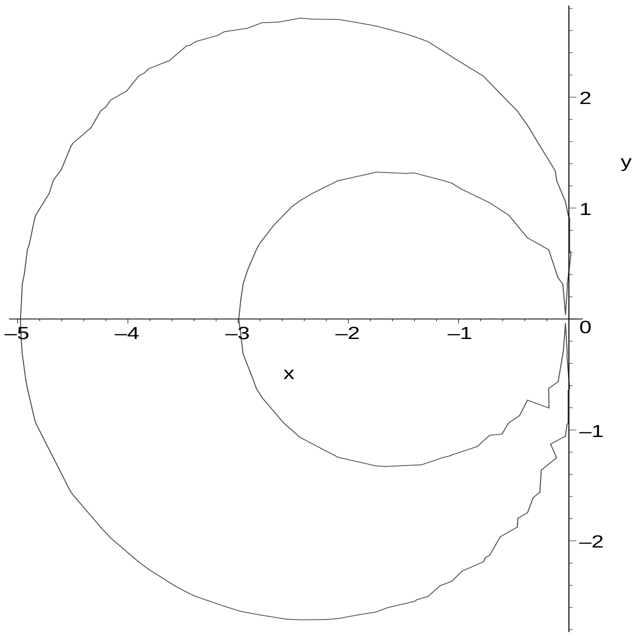
using Maple and using C-XSC, the STL and gnuplot

# Maple Plot

What is the correct shape of $f(x,y) = (x^2+y^2+4x)^2 - x^2 - y^2 \overset{!}{=} 0$ ?

Using Maple's `implicitplot`-command we get




Symmetry with respect to $y$?

```
//Verified computation of level curves
//using C-XSC and routines from the STL

#include <fstream>
#include <list>

#include <interval.hpp>  //C-XCS intervals
#include <imath.hpp>     //intreval functions

using std::ofstream;
using std::list;
using std::pair;
using std::make_pair;
using namespace cxsc;

//Abbreviate instantiation of intervals
typedef interval I;

//Data type for two dimensional boxes
typedef pair<I,I> rectangle;
```

```
void levelCurve( ofstream& out,      //output file
                 I(*f)(rectangle),   //function
                 rectangle x, I level, double epsilon )
{
   list<rectangle> toDo;
   toDo.push_back(x);
   while( !toDo.empty() )
   {
      rectangle box= toDo.front();
      toDo.pop_front();
      I fRange= f(box);
      if (!disjoint(level, fRange) //box may contain points of level curve(s)
      {
         if ( diam(box) < epsilon )
            plotBox(out, box);
         else
            toDo.bisect_and_push_back(box));
      }
   }
}
```

```
I f(rectangle x) //two dimensional (interval) function
{
   I h= sqr(x.first) + sqr(x.second);
   return sqr(h+4*x.first) - h;
}

int main()
{  //...
   rectangle x(make_pair(I(-6, 1), I(-6, 5)));
   levelCurve(out, f, x, level, epsilon);    //fine
}
```

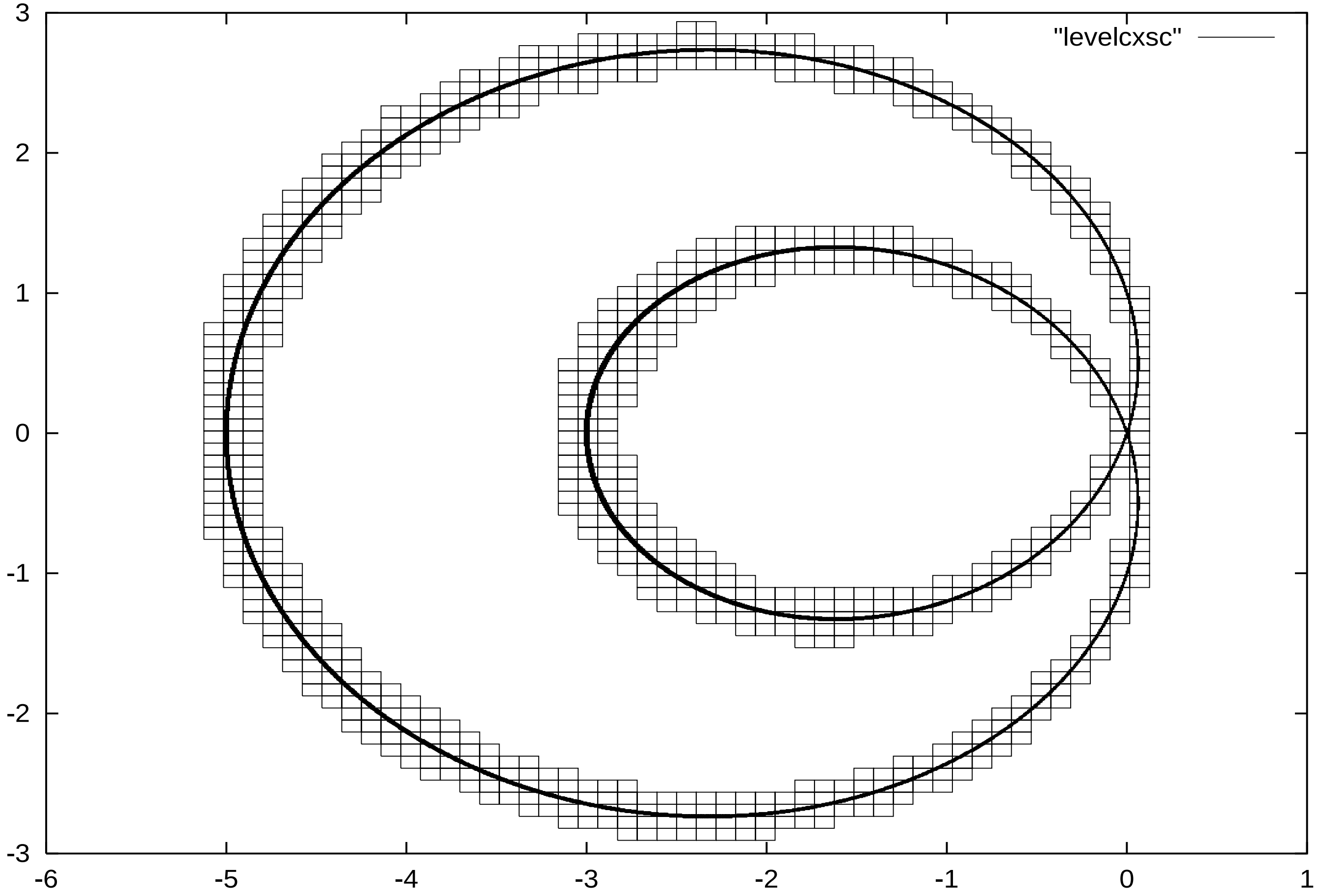
# Coarse and fine coverings of the level curve (verified)

xRange: [ -6.000000,  1.000000]

yRange: [ -6.000000,  5.000000]

level : 0

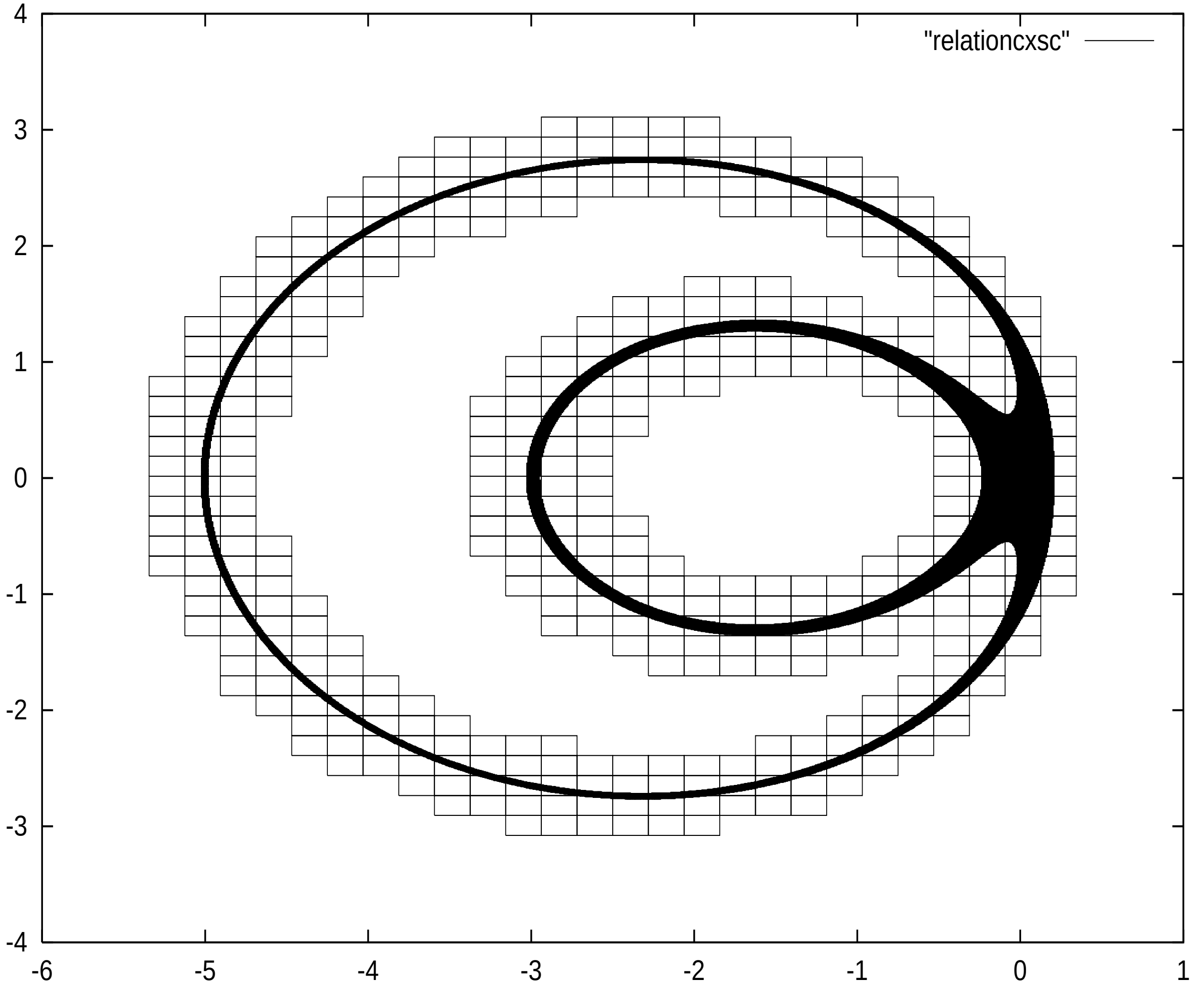
Epsilon (fine): 0.01



## Simultaneous plotting of (infinitely many) level curves

`level` -parameter may be an interval

`xRange: [ -6.000000,  1.000000]  yRange: [ -6.000000,  5.000000] level = [ -0.300000,  0.700000]      <== !!!`



## Toolbox with Verification Algorithms

One-dimensional:

- Polynomials
- Automatic differentiation
- Nonlinear equations in one variable
- Global optimization
- Arithmetical expressions
- Zeros of complex polynomials

## Toolbox with Verification Algorithms

Multi-dimensional:

- Systems of linear equations
- Linear optimization
- Automatic differentiation (gradient, Jacobi-, Hesse matrix)
- Nonlinear systems of equations
- Global optimization

## Further Software Packages Based on C-XSC

- Slope arithmetic (Breuer)
- Taylor arithmetic (Breuer)
- Mathematical functions for complex rectangular intervals (Westphal)
- Verified solutions of ordinary differential equations (Lohner)
- Quadrature and cubature of singular integrals (Wedner)
- Estimates for Tyalor coefficients of analytic functions (Neher)

## What can we learn from IntLab?

Powerful interactive tool to implement prototypes of verification algorithms, fast linear algebra, build on commercial MatLab (Rump)

- Switching the rounding mode is done in assembler
- Midpoint-radius as well as infimum-supremum representations of intervals
- Midpoint-radius arithmetic entirely based on BLAS
- Fast real and complex interval matrix/vector operations (BLAS), switching the rounding mode is minimized in a clever way!

## What can we learn from Sun Forte C++ with interval support?

- Input and output of intervals also in a special single-number form
- Extended `interval` arithmetic operations and mathematical functions (exception free containment computations)
- Three types of interval relational functions:
  - certainly relational functions
  - possibly relational functions
  - set relational functions

Three types of interval relational functions:

- certainly relational functions are true if the underlying relation (e. g. less than) is true for **every element** of the operand intervals
- possibly relational functions are true if **any element** of the operand intervals satisfy the underlying relation
- set relational functions are true if the interval operands satisfy the underlying relation in the **ordinary set theoretic** sense

# Containment Mode

Extended interval arithmetic (containment computations):

**Exception free** computations in a closed interval system with

- infinite intervals
- only partially defined functions
- functions with singularities

Containment set:

$$f^*(X) := \{ f(x) \mid x \in X \cap D_f \} \bigcup \{ \lim_{D_f \ni x_k \to x^*} f(x_k) \mid x^* \in X \} \subseteq I\!R^*$$

$\sqrt{[-1,4]} = [0,2],\ \log[-1,e] = [-\infty,1],\ 1/[0,0] = \{-\infty,\infty\}$

# Sun Forte C++ Code Example

```
//To compile and link:  CC -xia <progname>

//Some containment computations

#include <iostream>
#include <suninterval.h> //header file for intervals

//Simplify instantiation of intervals
typedef SUNW_interval::interval<float> interval;

using namespace SUNW_interval;

using std::cout;
using std::endl;
```

# Sun Forte C++ Code Example

```
int main() {
   interval x("[-1.5, 3]");
   cout << "x=            " << x << endl;
   cout << "cos(x)=       " << cos(x) << endl;
   cout << "log(x)=       " << log(x) << endl;
   cout << "atan(log(x))= " << atan(log(x)) << endl;
   cout << "log([-2,-1])= " << log(interval(-2,-1)) << endl;
   cout << "[1]/[0]=      " << interval(1)/interval(0) << endl;
}
```

```
x=            [-.15000000E+001,0.30000000E+001]
cos(x)=       [-.98999250E+000,0.10000000E+001]
log(x)=       [      -Infinity,0.10986124E+001]
atan(log(x))= [-.15707964E+001,0.83235294E+000]
log([-2,-1])= [EMPTY                          ]
[1]/[0]=      [      -Infinity,       Infinity]
```

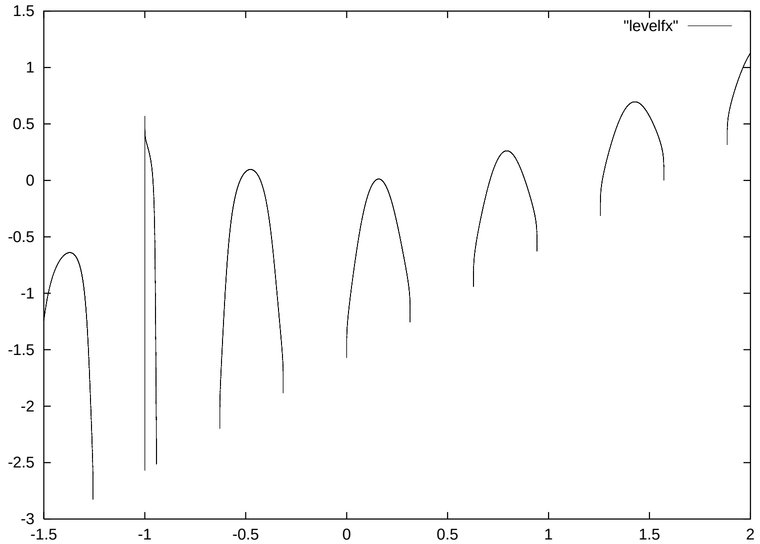
$$f(x) = x + \arctan(\log(\sin(10x)/(x+1)))$$

**All** zeros of $f$ on the **complete** real line using bisection

Note:

- natural domain of definition $D_f$ is not the complete real line
- $x = -1$ leads to division by zero

Due to containment computations **no** exceptions are thrown!



# Numerical Results

```
All zeros of x + atan( log( sin(10*x)/(x+1) ) )
Search range: [ ENTIRE ]
eps: 1e-7
There may be zeros in the interval(s):
  1  [-1.000000059604645,  -0.9999999999999999]
  2  [-0.9594483375549315, -0.9594482779502868]
  3  [-0.5274564027786254, -0.5274562835693358]
  4  [-0.4268767237663268, -0.4268766045570373]
  5  [0.1427576541900634,    0.142757773399353]
  6  [0.1746257543563842,   0.1746259331703186]
  7  [0.7093482613563536,   0.7093483209609984]
  8  [0.8833859562873839,   0.8833860158920287]
  9  [1.266079187393188,     1.266079246997833]
  10 [1.570796310901642,     1.570796370506286]  lim f(x) = 0
                                                x->pi/2
```

# Future of C-XSC

- Adapt C-XSC to Wuppertals parallel machine ALiCE
- Add features coming from IntLab
- Add features from Sun Forte
- Develop fast parallel self verifying problem solving routines

# Future of C-XSC (detailed)

- Special hardware adaptation for Alpha processors
- Midpoint-radius representation of intervals
- New fast mode for matrix/vector operations not using the long accumulator but using BLAS
- New containment mode (exception free computations)
- Global optimization, sparse system of linear equations, ...
- Real life applications

**To do:** Standardization of interfaces (IntLab, Sun Forte, XSC, ...)

**Note: C-XSC sources** as well as a lot of **documentation** are freely available!