Arkiveringsbeteckning: _____

HÖGSKOLAN I GÄVLE

Department of Mathematics, Natural- and Computer science, University-College of Gävle

Amoeba Program: Computing and visualizing amoebas for some complex-valued bivariate expressions

Magnus Leksell
Wojciech Komorowski
October 2007

Bachelor's Degree in Mathematics
Supervisor/examiner: Mikael Forsberg

Amoeba Program: Computing and visualizing amoebas for some complex-valued bivariate expressions

by

Magnus Leksell
Wojciech Komorowski

Department of Mathematics, Natural- and Computer science,
University-College of Gävle

S-80176 Gävle, Sweden

E-mail:
nfk03mll@student.hig.se
nmd04wki@student.hig.se

Abstract

In this paper, we present an implementation of a C++ program for visualizing the amoebas of arbitrary bivariate polynomials with complex coefficients, power series and simple bivariate functions of the form $f(z, w) = z + \cos w$.

Keywords: Amoeba, Polynomials, Complex numbers, Arbitrary precision, Parallel computing, C++, XML

Contents

Preface 2

1 Introduction **3**

2 Program requirements **3**

3 Implementation **4**
3.1 The polynomial 4
3.2 The intervals 4
3.3 Parallel computing 5
3.4 Root-finding 5
3.5 The rendering 8
3.5.1 The Rendering of Amoebas 8
3.5.2 The Rendering of The Newton-polytope 9
3.6 XML parsing 9

4 Tests **10**
4.1 Root-finder test 10
4.2 Program tests 12

5 Discussion **21**
5.1 The result 21
5.2 About the work 21

References **22**

A Description of command line options **23**

B The structure of the XML config file **24**

C Taylor tests **26**

D Source code **29**

Preface

This paper is part of the Bachelor's Diploma Work in Mathematics by Magnus Leksell and Wojciech Komorowski at the Department of Mathematics, Natural- and Computer science, University-College of Gävle.

The work is about making the program Amoeba Generator, which is developed using C++ and some open source libraries such as: stl – the Standard Template Library, CLN for arbitrary precision numbers, GD for image creation and libxml++ for XML parsing.

Except parts of the work and code that we developed together, there are some parts of the work which one can say was done mostly from one of us, and these are:

Magnus Leksell Overall system design, parallel processing/computing, XML parsing, power series, polynomial representation, LaTeX-producing, initial image creation code and administration of the Subversion repository.

Wojciech Komorowski Maybe the most important part of the program; implementation of the root-finding algorithm, amoeba and Newton polytope image rendering and command line parsing.

1 Introduction

The word Amoeba is a recent addition to the mathematical terminology, introduced by I. M. Gelfand, M. M. Kapranov and A. V. Zelevinsky in their book "Discriminants, resultants, and multidimensional determinants" from 1994 [GKZ]. It is defined as the image of the zero locus of a multivariate polynomial under the map

$$\mathrm{Log} : (\mathbb{C} \setminus 0)^n \to \mathbb{R}^n : (z_1, \ldots, z_n) \to (\log|z_1|, \ldots, \log|z_n|),$$

where 'log' denotes the natural logarithm. For the case of bivariate polynomials, the amoeba

$$\mathcal{A}_p = \{\mathrm{Log}(z) : z \in (\mathbb{C} \setminus 0)^2, p(z) = 0\}$$

is planar.

Planar amoebas of a not identically zero bivariate polynomial have a finite area. If the bivariate polynomial is not identical to any univariate polynomial, then the amoeba is 2-dimensional and has a number of infinitely long and narrowing "tentacles", and can even have holes (vacuoles), which makes it resemble a biological amoeba.

Every amoeba is associated with its Newton-polytope, which for a 2-dimensional amoeba is the convex hull of the exponents of the monomials of the amoeba's polynomial. As an example, the Newton-polytope of the polynomial $p(z, w) = 1+z+w$ is the polygon with vertices at $(0, 0)$, $(1, 0)$ and $(0, 1)$. Relations between Newton-polytopes and amoebas have been discovered by Mikael Forsberg, Mikael Passare and August Tsikh [FPT]. One such relation is that an amoeba's tentacles are always perpendicular to the sides of its Newton-polytope.

Amoebas are a relatively fresh field of research, and as far as we know, there does not exist any program with the general purpose of visualizing them.

Our objective has been to construct such a program, capable of visualizing amoebas of bivariate polynomials, power series and simple bivariate functions of the form $f(z, w) = z + \cos w$.

2 Program requirements

The requirements was to create a program that can:

- Calculate the amoeba for a polynomial in at least two variables.
- Calculate the amoeba for a power series in at least two variables.
- Calculate the amoeba for a function of type $z + \cos w$.
- Plot lists of real number pairs.
- Calculate and plot Newton-polytopes.
- Generate information about the polynomial (and the power series) in TEX.

3 Implementation

The program works by first parsing the command line for input. Then, either parse a XML-configuration file, setting up some initial parameters in the `Amoeba` class and, for each interval, compute the amoeba and produce images (the amoeba and the Newton polytope) and a LaTeX-file. Or plot an image from values given in a file. That are the two main functionalities of the program. In the XML-file one sets up what to be computed, a bivariate polynomial, power series or, a bivariate expression of type $z + \cos w$. Note that $\cos w$ will be approximated with a Taylor polynomial in desired terms (default 10 terms).

The computing of the zeros of the given polynimial is divided into intervals (or regions), and as the zeros are found for each region, they are saved in a file for later use. In this way, computing the amoeba is very memory efficient.

All error management is implemented using C++ exceptions and specifically the stl's exceptions `runtime_error` and `out_of_range`.

3.1 The polynomial

The bivariate polynomial is represented in C++ by the stl container `map`, as

```
1 typedef std::map<Key, cln::cl_N> CPolynomial;
```

where `Key` is defined as `typedef pair<int, int> Key` and `cl_N` is a complex number class. For example, the polynomial $2i - z^2 + zw^3$ would be represented as:

```
1 CPolynomial poly;
2 poly[0,0] = cln::complex(0,2);  // (0+2i)z^0w^0
3 poly[2,0] = cln::complex(-1,0); // (-1+0i)z^2w^0
4 poly[1,3] = cln::complex(1,0);  // (1+0i)z^1w^3
```

This is a very simple but still a very powerful structure to represent a bivariate polynomial, and is used throughout the whole program. We also use the CLN structure `cl_UP_N`, which is a univariate polynomial with complex coefficients, when constructing the p_z and p_w polynomials respectively (which is more in depth described in section 3.5.1).

3.2 The intervals

The calculation of the amoeba is divided into intervals, which will make computations a lot faster, and one can also do some experiment more precisely. These intervals are defined in the XML-configuration file and are implemented as two classes, `Square` and `Circle`. These two classes inherits an abstract base class, `AbstractInterval`, which is one of the input parameter types to the method `compute` in the class `Amoeba`.

The abstract base class represents a region in the complex plane from which substitute values will be taken to replace one of the variables in the bivariate polynomial $p(z, w)$, so that univariate polynomials can be created as a step in plotting the amoeba.

To determine what kind of interval which will be used in the `compute` method, we use the C++ function `typeid`, as the following sample code shows.

```
1 void Amoeba::compute(const CPolynomial& p, const AbstractInterval* interval,
2                       sem_t* mutex) {
3   // Determine the type of interval
4   if (typeid(*interval) == typeid(Square)) {
```

```
5       // compute values from a given square type of interval
6     } else {
7       // compute values from a given circle type of interval
8     }
9     // ...
10  }
```

3.3 Parallel computing

Computing the amoeba is done using parallel processes, which is beneficial for architectures with more than one core/CPU. Tests on a dual-core machine shows that time spent on computing, using two parallel processes, is cut down by 50% compared to using only one process. And one can expect that more cores or CPU's will cut down computing times even further. That is, the number of parallel processes should equal the number of CPU cores.

Due to the fact that the [CLN] library is not thread safe, we were forced to implement parallelism using `fork()` instead of threads, which might have been a more modern choice. But after some thinking we came up with a clean solution. The computed data for each interval is saved in a file, and to prevent data corruption, each process locks the file before saving the data. The file locking mechanism is implemented using a semaphore variable (`mutex`), which is allocated in a shared memory area, which enables it to be shared between the processes. The main computing algorithm in pseudo code is listed in algorithm 1.

Algorithm 1 Pseudo code for the main computing process

1: $interval \leftarrow$ first interval
2: $processCount \leftarrow 0$
3: **while** $interval \neq end(interval)$ **do**
4: create process and compute amoeba for $interval$
5: $processCount \leftarrow processCount + 1$
// If maximum number of processes running, wait for any process to finish
6: **if** $processCount == MAX\_PROCESSES$ **then**
7: wait for a process to finish
8: $processCount \leftarrow processCount - 1$
9: **end if**
10: $interval \leftarrow next(interval)$
11: **end while**
// Wait for remaining processes to finish
12: **while** $processCount > 0$ **do**
13: wait for a process to finish
14: $processCount \leftarrow processCount - 1$
15: **end while**

3.4 Root-finding

We have evaluated several algorithms for finding roots of univariate polynomials with complex coefficients.

Among them was the Newton-Rhapson method, which has quadratic convergence for simple roots and linear convergence for multiple roots. It did however not satisfy our needs since, in a straight forward implementation, polynomial deflation has to be used in order to find all the roots of a polynomial. Polynomial deflation works by finding one root of the polynomial and factoring it out by polynomial division, leaving a new polynomial of a smaller degree, for which the algorithm is applied again, until all the roots have been found. There are several problems with this approach.

Despite working with arbitrary precision arithmetic, there are inevitably small errors introduced with every arithmetic operation. These errors would grow in the coefficients of the polynomial obtained after each polynomial division.

Furthermore, the problem of finding roots of arbitrary polynomials has been shown to be ill-conditioned [Wilkinson], in the sense that small perturbations in the coefficients of a polynomial can lead to large perturbations in the roots of the polynomial.

Consider Wilkinson's polynomial: $w(x) = (x-1)(x-2)\dots(x-20)$, which clearly has 20 integer roots. Expanding the polynomial gives:

$$\begin{aligned} w(x) = {} & x^{20} - 210x^{19} + 20615x^{18} \\ & - 1256850x^{17} + 53327946x^{16} \\ & - 1672280820x^{15} + 40171771630x^{14} \\ & - 756111184500x^{13} + 11310276995381x^{12} \\ & - 135585182899530x^{11} + 1307535010540395x^{10} \\ & - 10142299865511450x^{9} + 63030812099294896x^{8} \\ & - 311333643161390640x^{7} + 1206647803780373360x^{6} \\ & - 3599979517947607200x^{5} + 8037811822645051776x^{4} \\ & - 12870931245150988800x^{3} + 13803759753640704000x^{2} \\ & - 8752948036761600000x + 2432902008176640000. \end{aligned}$$

If the coefficient of x^{19} is decreased from -210 to -210.0000001192, then the root at $x = 20$ grows to $x \approx 20.847$ [ratfun].

This does not bode well for a root finding algorithm based on polynomial deflation.

Another problem with the Newton-Rhapson method is that the set of initial values in the complex plane for which the algorithm converges to a root is a fractal depending on the polynomial. Some convergence fractals have regions where initial values never converge. To solve this, one could take a heuristic approach, letting an initial value jump around until the algorithm converges to a root, however that does not solve the problems mentioned earlier.

In a paper by John Hubbard, Dierk Schleicher and Scott Sutherland [HSS], a method of finding all roots of complex polynomials by Newton's method with no intermediate deflation is presented. We did not chose this path because we felt an implementation would be complex and hard to debug.

Another popular algorithm we considered was the Jenkins-Traub algorithm [JT], [RR], which has been described as "practically a standard in black-box polynomial root-finders" [Press]. It is an iterative three-stage method with faster convergence than the Newton-Rhapson method. The algorithm is pretty complicated, making an implementation hard to debug. Unfortunately it is also based on polynomial deflation, and

since we prioritize accuracy over speed when it comes to root finding, we chose not to implement it.

We also considered a method from linear algebra: Given the polynomial

$$p(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0,$$

with $a_n = 1$, one can form its $n \times n$ companion matrix

$$C = \begin{pmatrix} 0 & & & & -a_0 \\ 1 & 0 & & & -a_1 \\ 0 & 1 & 0 & & -a_2 \\ & & \ddots & & \vdots \\ & & & 1 & -a_{n-1} \end{pmatrix}$$

with characteristic polynomial $P_C(z) = \det(zI - C) = p(z)$. The problem of finding the roots of $p(z)$ is thus equivalent to computing the eigenvalues of C with any suitable algorithm. We rejected this method because:

- The companion matrix requires $\mathcal{O}(n^2)$ storage, which becomes a problem for polynomials of higher degrees
- The need of implementing an algorithm for finding the eigenvalues
- The accumulated errors in the eigenvalues after executing such an algorithm.

After having reviewed and rejected many more algorithms for similar reasons, we finally settled with the Weierstrass or Durand-Kerner method [petkovic-weierstrass]. It is an incredibly simple iterative algorithm with quadratic convergence to simple roots and linear convergence to a multiple root. The algorithm finds all roots of a polynomial simultaneously and without the need of polynomial deflation.

Given a monic polynomial P of degree $n \geq 3$, the method works by setting up a vector $Z = (z_1, \ldots, z_n)$ of distinct trial roots to P. Each z_i in Z is then refined by letting z_i-new $= z_i$-old $- W(z_i$-old$)$, where W is Weierstrass' correction defined as

$$W(z_i) = P(z_i) \Big/ \prod_{j=1, j \neq i}^{n} (z_i - z_j).$$

By successive iterations of these refinements, Z will eventually converge to the vector $R = (r_1, \ldots, r_n)$ of roots to P.

In pseudo code, the algorithm for a general n:th degree polynomial P is listed in algorithm 2.

There is some loss of precision in making a given polynomial monic, however it is not nearly as bad as repeatedly deflating it.

The speed at which the algorithm converges to the roots is dependent on the initial trial roots, and a natural question to ask is therefore which trial roots to use. We have found by experiments that distributing them around the unit circle is a good approach, and it is what our implementation uses.

The convergence check in our implementation is done by comparing $|P(z)|$ for each z in Z to a user defined error value. We have chosen to loop part 3. of the pseudo

Algorithm 2 General n:th degree polynomial

1: Make P monic (i.e. make the leading coefficient a 1 without changing the zero set of the polynomial)
2: Setup the starting vector $Z = (z_1, \ldots, z_n)$
3: **for** $i = 1$ to n **do**
4: $\quad Z[i] = Z[i] - P(Z[i]) / \prod_{j=1, j \neq i}^{n} (Z[i] - Z[j])$
5: **end for**
6: if no convergence, goto 3. (or give up)

code n times for an n:th degree polynomial before checking for convergence, and then in steps of n times until convergence occurs. Empirical tests showed this to be a good approach. We have included a threshold of 65535 iterations, should the algorithm fail to converge. In such a case, it will print an error message on stderr and continue.

Failure of convergence for this method is extremely rare. We have only come across it when specifying "unrealistic" error bounds, for which a given working precision simply is not enough. In each case, however, the error bounds could be satisfied by working with a higher precision.

It should be noted, unfortunately, that it is always possible to construct a polynomial which defeats any particular root finding algorithm working with a predefined precision. Polynomials of very high degrees, or which have roots with a huge difference in magnitude, or roots which are very tightly clustered fall into this category.

Test runs on various polynomials and amoebas are provided in section 4.

3.5 The rendering

When constructing the image, the program first reads all the points from the data-file that was created when calculating the zeros of the polynomial (or power series), and then calculates and adjust the points to fit an image structure. All images are created using the GD library and we chose the image type to be png.

3.5.1 The Rendering of Amoebas

Our approach to rendering the amoeba of a bivariate polynomial $p = p(z, w)$ requires constructing univariate polynomials $p_z(w)$ and $p_w(z)$ from p. This is done by replacing one of the variables of p with constants. These constants come from intervals, or regions in the complex plane, specified in the user-provided XML-file.

The regions have the shape of a rectangle, or a disk centered at the origin in the complex plane with a hole in the middle (much like a CD or DVD disk). Each region has a finite amount of points contained within them. For more information about these regions, see section 3.2.

An outline of the amoeba rendering algorithm is given below:

1. Setup a list that will contain points that make up the amoeba. We can call it the A-list.

2. For each point of each region:

   2.1 Construct $p_z(w)$ and $p_w(z)$ by replacing z and w with the point, respectively.

2.2 Set up two lists that will hold the roots of $p_z(w)$ and $p_w(z)$. We can call them $p_z(w)$-roots and $p_w(z)$-roots.

2.3 Find all the roots of $p_z(w)$ and $p_w(z)$ and place them in their respective roots-list.

2.4 For each root in $p_z(w)$-roots:

2.4.1 If $\log|z|$ and $\log|root|$ are within the limits of minLnZ/maxLnZ and minLnW/maxLnW respectively, then construct the point $(\log|z|, \log|root|)$ and place it in A-list.

2.5 For each root in $p_w(z)$-roots:

2.5.1 If $\log|root|$ and $\log|w|$ are within the limits of minLnW/maxLnW and minLnZ/maxLnZ respectively, then construct the point $(\log|root|, \log|w|)$ and place it in A-list.

3. Create an image with the dimensions specified in the XML-file, and map all points in A-list to it.

The mapping of points in the amoeba to the image in step 3. is such that the whole width and height of the image is used. The mapping is therefore appropriately scaled to make full use of the image dimensions.

If one chooses to use axes in the image, then the horizontal axis correspond to $\log|z|$ with positive direction to the right, and the vertical axis corresponds to $\log|w|$ with positive direction upwards.

For more information about how the roots of $p_z(w)$ and $p_w(z)$ are found, see section 3.4.

3.5.2 The Rendering of The Newton-polytope

The Newton-polytope to $p(z, w)$ is constructed from the exponents of the monomials of p with non-zero coefficients. Every such monomial is represented as a small disk, or dot, in the image. The location of each dot is dependent on the exponents of the monomial. Increasing exponents in the z variable of the monomial pushes the dot further to the right, and increasing exponents in the w variable of the monomial pushes the dot further upward.

The convex hull is constructed using the Graham's Scan algorithm [O'Rourke]. The dimensions of the image are determined automatically.

3.6 XML parsing

Implementation of the parsing of the configuration-file was done using the [libxml++] library, which is a C++-wrapper for the libxml library. It was fairly complicated to implement XML-parsing functionality, but it was absolutely worth it, as it allows us to expand and/or extend the functionality of the program more easily.

One small limitation of the structure in the XML-configuration file is that the `<settings>...</settings>` entity should be declared in the head of the file. That is because the `<max precision="`n`"/>` is used when constructing the other numbers. If one does not declare the `<max precision="`n`"/>` at all, the default precision (50) is used.

4 Tests

What follows are some tests. We intentionally did not include any tests of the `-p`, `--plot=LISTFILE` command, because we know it is working and it is not much to test. Also, most of the tests produces a LaTeX-file which is not shown in these tests. See also Taylor tests in appendix C.

4.1 Root-finder test

Since drawing amoebas by our method depends explicitly on the ability of the polynomial root finder to perform its task, we have extensively tested it with satisfactory results. We have constructed a small program called "testroots", specifically for this purpose. The test program reads a file that defines which polynomial the root finder shall try to solve, and presents the result. The root finder used in the tests differs only slightly from the one used in the main program, the only difference being that the one in the main program does not report the number of iterations used to solve a polynomial.

The testroots program, along with various test polynomials, can be found in the `amoeba/source/polynomial/rootfinder` directory. The README file therein present more information about the format of the files used to define polynomials to test. The output generated by the testroots program can be pretty huge. We have therefore chosen not to include all tests directly in this report. Instead, we choose to include only the output of testroots on Wilkinson's polynomial, with a working precision of 50 decimal digits and an error threshold of 1e-30.

$$\begin{aligned} w(x) = {} & x^{20} - 210x^{19} + 20615x^{18} \\ & - 1256850x^{17} + 53327946x^{16} \\ & - 1672280820x^{15} + 40171771630x^{14} \\ & - 756111184500x^{13} + 11310276995381x^{12} \\ & - 135585182899530x^{11} + 1307535010540395x^{10} \\ & - 10142299865511450x^{9} + 63030812099294896x^{8} \\ & - 311333643161390640x^{7} + 1206647803780373360x^{6} \\ & - 3599979517947607200x^{5} + 8037811822645051776x^{4} \\ & - 12870931245150988800x^{3} + 13803759753640704000x^{2} \\ & - 8752948036761600000x + 2432902008176640000. \end{aligned}$$

below:

```
$ ./testroots testpolynomials/wilkinson20
p(x) = 1.0L0+0.0L0i*x^20 + -210.0L0+0.0L0i*x^19 + 20615.0L0+0.0L0i*x^18 +
-1256850.0L0+0.0L0i*x^17 + 5.3327946L7+0.0L0i*x^16 + -1.67228082L9+0.0L0i*x^15 +
4.017177163L10+0.0L0i*x^14 + -7.561111845L11+0.0L0i*x^13 +
1.1310276995381L13+0.0L0i*x^12 + -1.3558518289953L14+0.0L0i*x^11 +
1.307535010540395L15+0.0L0i*x^10 + -1.014229986551145L16+0.0L0i*x^9 +
6.3030812099294896L16+0.0L0i*x^8 + -3.1133364316139064L17+0.0L0i*x^7 +
1.20664780378037336L18+0.0L0i*x^6 + -3.5999795179476072L18+0.0L0i*x^5 +
8.037811822645051776L18+0.0L0i*x^4 + -1.28709312451509888L19+0.0L0i*x^3 +
1.3803759753640704L19+0.0L0i*x^2 + -8.7529480367616L18+0.0L0i*x +
2.43290200817664L18+0.0L0i
```

```
Total iterations: 1580
roots are:
r1:
18.999999999999999999999999999999999999999999999774780908074L0
3.341623260306345003183202426794910180487015391003997778391-2824i
r2:
10.000000000000000000000000000000000000000000000212899034792L0
3.470537591235824682751716401424411749539254085888924835862L-2824i
r3:
13.999999999999999999999999999999999999999999995216463252234L0
3.406988727118285830634969586701750606157721449964138234156L-2822i
r4:
16.000000000000000000000000000000000000000000008594758907363L0
-3.113922822499330697603550499242645412179993256983885072912 7L-2829i
r5:
7.000000000000000000000000000000000000000000000003023239204L0
-2.064083498633128815519526661654592169280371866098392366231 7L-2843i
r6:
17.000000000000000000000000000000000000000000000640420384 28L0
6.233334493940588076362312352147463732349551763276349358683L-2841i
r7:
18.000000000000000000000000000000000000000000001254153624 77L0
8.429947260995633116861425555783319578933469779815735327317L-2844i
r8:
19.999999999999999999999999999999999999999999999994558286856L0
-1.508954796724865315049951613054138389326582987037636903934L-2845i


r9:
12.000000000000000000000000000000000000000000000531255729008L0
-3.345328725896142194391171074936361703022971660271196270119L-2843i
r10:
5.000000000000000000000000000000000000000000000002571422L0
-9.537664928214108585109116583134946145137417492113570605 22L-2854i
r11:
15.000000000000000000000000000000000000000000000011433970346 54L0
9.441791450321380497396762990999969255730501669503179968742L-2848i
r12:
12.999999999999999999999999999999999999999999995844635935557L0
-4.910680794183162198029036150532519087130729193087554107469L-2862i
r13:
10.999999999999999999999999999999999999999999963682469921 38L0
6.423641755237283616668063991449957878971234578190355839 91L-2867i
r14:
9.000000000000000000000000000000000000000000000082021283866L0
3.263183124022433703994416423308504656696376587944248131632L-2869i
r15:
5.999999999999999999999999999999999999999999999706162552L0
4.922274257721006804327508158579962070795161104520353410442L-2871i
r16:
3.999999999999999999999999999999999999999999999999992194L0
-1.450199512569617547104205552561340210356879494215908051543L-2874i
r17:
3.000000000000000000000000000000000000000000000000000256074L0
5.171816019169585168573794985202463823472450831020226540 95L-2875i
r18:
8.000000000000000000000000000000000000000000000008103615448L0
-5.166494805732630214145289266233934822028131384685602715104L-2870i
r19:
2.000000000000000000000000000000000000000000000000001548L0
-4.386363714224298017213234554609193164819736000254129299 04L-2877i
r20:
1.000000000000000000000000000000000000000000000000000108L0
4.214712759409167325532351730736563879181933855782536744554L-2947i
--------------------------------------------------------------------------------
Verifying roots...
```

```
|p(r1)| = 4.193098383001176199674526026816237692613659354468809786439L-31
|p(r2)| = 1.197896334413083252151850733780358624823694775620271579 45L-33
|p(r3)| = 2.206009105885908186142148953626185684963565997099689628 5522L-32
|p(r4)| = 4.718319052285884546417758276183412794297342451991998348 5036L-32
|p(r5)| = 4.863607876527214564220647422757041491974728741144512120242L-35
|p(r6)| = 1.357895662302588007794306889610954386865078163883217745 7103L-31
|p(r7)| = 4.386344048637674782201032687623213003462971874240767118083L-31
|p(r8)| = 7.149575224875688027827962405944794689076914297148118075491L-31
|p(r9)| = 4.769325882751574466784629947868459749578185396831660875 47L-33
|p(r10)| = 1.300390625597155555690414794302109281086898772179121820669L-36
|p(r11)| = 8.994949775670140881364726200807245594672064504356749723574L-32
|p(r12)| = 3.161987894747398478831371979539836808998275900803695315265 5L-32
|p(r13)| = 1.386862194095550816565672495361184964857885720533195526415 3L-33
|p(r14)| = 3.376313649149315294763203519036595148114662145438628383653 5L-34
|p(r15)| = 2.379641376445868273944211027539283594034161086490494676354 3L-36
|p(r16)| = 1.763241526233431261953104805833368516727998335158131282263L-37
|p(r17)| = 3.085672670908504708417933410208394904273997086526729743960 5L-38
|p(r18)| = 1.777692759908852759004195460637844499529681554846696813323 07L-34
|p(r19)| = 3.673419846319648462402301678819517743183329864912773504715L-39
|p(r20)| = 5.126991568127135470858841755526648095199651459337606426141L-2930
```

The 'L' characters in the numbers above are due to CLN's way of representing "Long-Float" objects. It has the same function as the 'e' character often encountered in numbers written in scientific notation.

4.2 Program tests

What follows are some tests of generating some basic amoebas from their XML files. For reference, these tests were done using a computer with a Dual-Core AMD Opteron 165 1.8 GHz processor, running 64-bit GNU/Linux (gentoo). The amoeba and Newton polytope images they generated are presented in figures 1 to 5.

Test 1

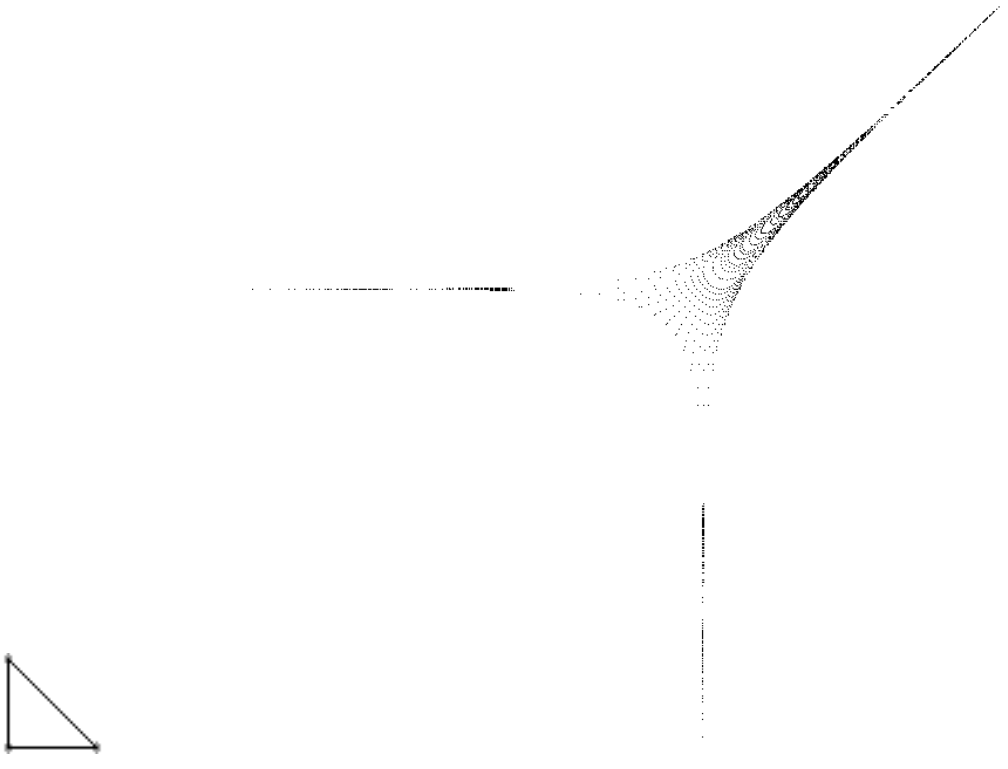
This is a test of the bivariate polynomial

$$p(z, w) = w - 2z - 1.$$

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <amoeba name="Amoeba 1">
3    <image filename="amoeba1.png" width="640" height="640"/>
4    <latex filename="amoeba1.tex"/>
5    <data filename="amoeba1.dat"/>
6    <npolytope filename="amoeba1npt.png"/>
7    <settings>
8     <max lnZ="10"/>
9     <min lnZ="-10"/>
10    <max lnW="10"/>
11    <min lnW="-10"/>
12   </settings>
13   <polynomial>
14     <!-- p(z,w) = w - 2z - 1 -->
15     <term wexp="1"/>
16     <term coeff="-2" zexp="1"/>
17     <term coeff="-1"/>
18   </polynomial>
19   <interval type="square" from="-0.001-0.001i" to="0.001+0.001i" step="0.0001"/>
20   <interval type="square" from="-0.01-0.01i" to="0.01+0.01i" step="0.001"/>
21   <interval type="square" from="-0.1-0.1i" to="0.1+0.1i" step="0.1"/>
22   <interval type="square" from="-1.0-1.0i" to="1.0+1.0i" step="0.1"/>
```

```
23    <interval type="square" from="-3.0-3.0i" to="3.0+3.0i" step="0.3"/>
24    <interval type="square" from="-6.0-6.0i" to="6.0+6.0i" step="0.6"/>
25    <interval type="square" from="-10.0-10.0i" to="10.0+10.0i" step="1.0"/>
26    <interval type="square" from="-100.0-100.0i" to="100.0+100.0i" step="10.0"/>
27  </amoeba>
```

```
$ time amoeba -a amoeba1.xml
Saving data in amoeba1.dat
Using 2 parallel processes.
Computing #1(8): -0.001L0-0.001L0i ... 0.001L0+0.001L0i, step=1.0L-4
Computing #2(8): -0.01L0-0.01L0i ... 0.01L0+0.01L0i, step=0.001L0
Computing #3(8): -0.1L0-0.1L0i ... 0.1L0+0.1L0i, step=0.1L0
Computing #4(8): -1.0L0-1.0L0i ... 1.0L0+1.0L0i, step=0.1L0
Computing #5(8): -3.0L0-3.0L0i ... 3.0L0+3.0L0i, step=0.3L0
Computing #6(8): -6.0L0-6.0L0i ... 6.0L0+6.0L0i, step=0.6L0
Computing #7(8): -10.0L0-10.0L0i ... 10.0L0+10.0L0i, step=1.0L0
Computing #8(8): -100.0L0-100.0L0i ... 100.0L0+100.0L0i, step=10.0L0
Saved image in amoeba1.png
Saved Newton Polytope in amoeba1npt.png
Saved LaTeX in amoeba1.tex

real    0m0.191s
user    0m0.280s
sys     0m0.000s
```



(a) $\mathcal{N}_p$ (b) $\mathcal{A}_p$

Figure 1: Newton polytope and amoeba for $p(z, w)$ in test 1

Test 2

This is a test of the bivariate polynomial

$$p(z, w) = 3z^2 + 5zw + w^3 - 1.$$

```
<?xml version="1.0" encoding="UTF-8"?>
<amoeba name="Amoeba 2">
  <description>A simple amoeba.</description>
  <image filename="amoeba2.png" width="640" height="480"/>
  <latex filename="amoeba2.tex"/>
  <npolytope filename="amoeba2npt.png"/>
  <settings>
    <max precision="20"/>
    <max error="1e-10"/>
  </settings>
  <polynomial>
    <!-- p(z,w) = 3z^2 + 5zw + w^3 + 1 -->
    <term coeff="3" zexp="2"/>
    <term coeff="5" zexp="1" wexp="1"/>
    <term wexp="3"/>
    <term coeff="-1"/>
  </polynomial>
  <interval type="square" from="-0.001-0.001i" to="0.001+0.001i" step="0.0001"/>
  <interval type="square" from="-0.1-0.1i" to="0.1+0.1i" step="0.01"/>
  <interval type="square" from="-1.0-1.0i" to="1.0+1.0i" step="0.1"/>
  <interval type="square" from="-3.0-3.0i" to="3.0+3.0i" step="0.3"/>
  <interval type="square" from="-6.0-6.0i" to="6.0+6.0i" step="0.6"/>
  <interval type="square" from="-10.0-10.0i" to="10.0+10.0i" step="1.0"/>
  <interval type="square" from="-100.0-100.0i" to="100.0+100.0i" step="10.0"/>
</amoeba>
```

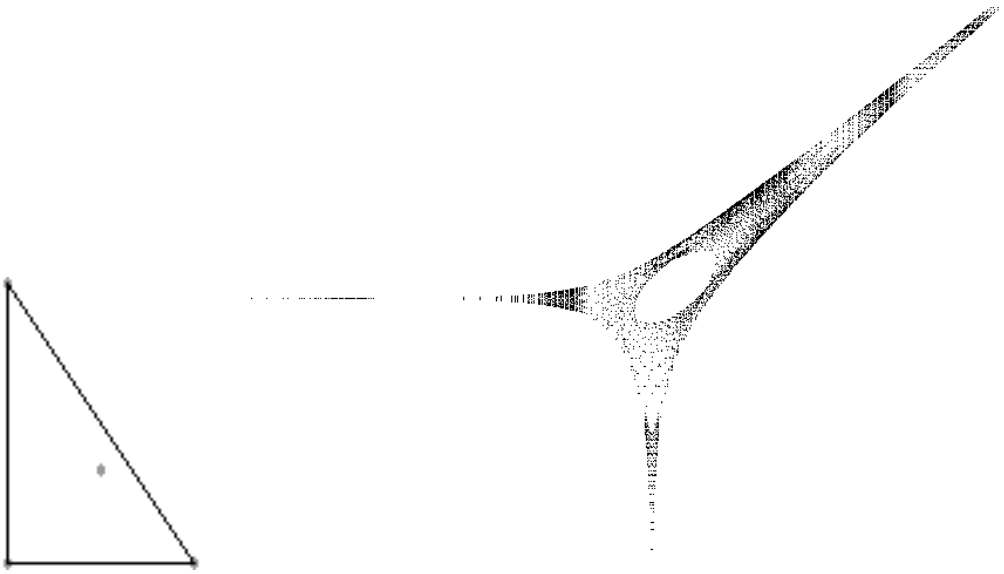
```
$ time amoeba -a amoeba2.xml
Saving data in amoeba.dat
Using 2 parallel processes.
Computing #1(7): -0.001L0-0.001L0i ... 0.001L0+0.001L0i, step=1.0L-4
Computing #2(7): -0.1L0-0.1L0i ... 0.1L0+0.1L0i, step=0.01L0
Computing #3(7): -1.0L0-1.0L0i ... 1.0L0+1.0L0i, step=0.1L0
Computing #4(7): -3.0L0-3.0L0i ... 3.0L0+3.0L0i, step=0.3L0
Computing #5(7): -6.0L0-6.0L0i ... 6.0L0+6.0L0i, step=0.6L0
Computing #6(7): -10.0L0-10.0L0i ... 10.0L0+10.0L0i, step=1.0L0
Computing #7(7): -100.0L0-100.0L0i ... 100.0L0+100.0L0i, step=10.0L0
Saved image in amoeba2.png
Saved Newton Polytope in amoeba2npt.png
Saved LaTeX in amoeba2.tex

real    0m0.937s
user    0m1.590s
sys     0m0.030s
```

Test 3

This is a test of the bivariate polynomial

$$p(z, w) = 1 + z + z^2 + z^3 + z^2w^3 + 10zw + 12z^2w + 10z^2w^2.$$

```
<?xml version="1.0" encoding="UTF-8"?>
<amoeba name="Amoeba 3">
  <description>A simple amoeba.</description>
  <image filename="amoeba3.png" width="640" height="480"/>
  <latex filename="amoeba3.tex"/>
```



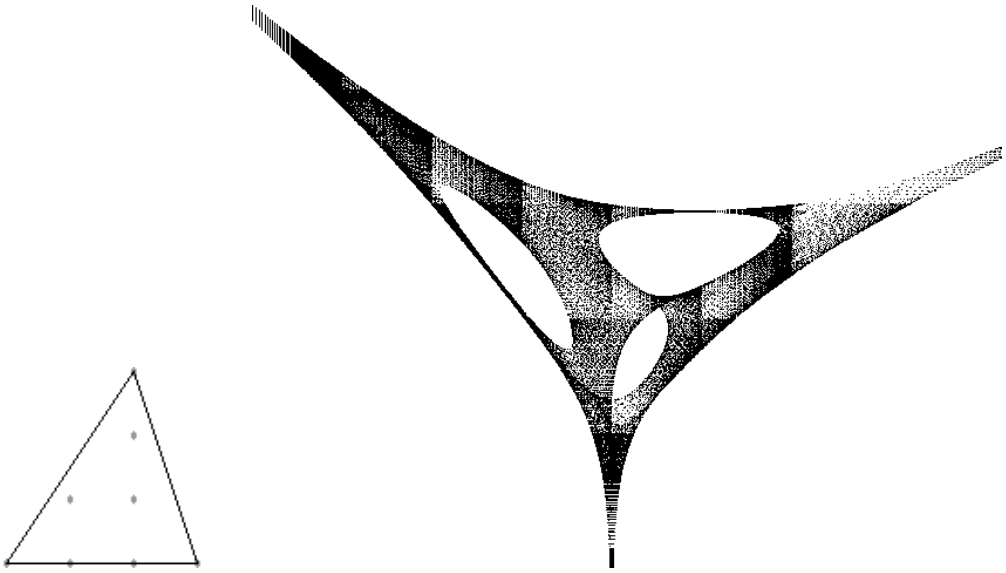
(a) $\mathcal{N}_p$ (b) $\mathcal{A}_p$

Figure 2: Newton polytope and amoeba for $p(z, w)$ in test 2

```
6     <npolytope filename="amoeba3npt.png"/>
7     <settings>
8       <max precision="50"/>
9     </settings>
10    <polynomial>
11      <!-- p(z,w) = 1 + z + z^2 + z^3 + z^2w^3 + 10zw + 12z^2w + 10z^2w^2 -->
12      <term coeff="1"/>
13      <term zexp="1"/>
14      <term zexp="2"/>
15      <term zexp="3"/>
16      <term zexp="2" wexp="3"/>
17      <term coeff="10" zexp="1" wexp="1"/>
18      <term coeff="12" zexp="2" wexp="1"/>
19      <term coeff="10" zexp="2" wexp="2"/>
20    </polynomial>
21    <!-- circle intervals -->
22    <interval type="circle" minradius="0.0001" maxradius="0.001" step="0.00001" points="20"/>
23    <interval type="circle" minradius="0.001" maxradius="0.01" step="0.0001" points="20"/>
24    <interval type="circle" minradius="0.01" maxradius="0.1" step="0.001" points="20"/>
25    <interval type="circle" minradius="0.1" maxradius="1.0" step="0.01" points="20"/>
26    <interval type="circle" minradius="1.0" maxradius="10.0" step="0.1" points="20"/>
27    <interval type="circle" minradius="10.0" maxradius="100.0" step="1.0" points="20"/>
28    <!-- square intervals -->
29    <interval type="square" from="-0.001-0.001i" to="0.001+0.001i" step="0.0001"/>
30    <interval type="square" from="-0.1-0.1i" to="0.1+0.1i" step="0.01"/>
31    <interval type="square" from="-1.0-1.0i" to="1.0+1.0i" step="0.1"/>
32    <interval type="square" from="-3.0-3.0i" to="3.0+3.0i" step="0.3"/>
33    <interval type="square" from="-6.0-6.0i" to="6.0+6.0i" step="0.6"/>
34    <interval type="square" from="-10.0-10.0i" to="10.0+10.0i" step="1.0"/>
35    <interval type="square" from="-100.0-100.0i" to="100.0+100.0i" step="10.0"/>
36  </amoeba>
```

```
$ time amoeba -a amoeba3.xml
Saving data in amoeba.dat
Using 2 parallel processes.
Computing #1(13): 1.0L-4 ... 0.001L0, step=1.0L-5, points=20
Computing #2(13): 0.001L0 ... 0.01L0, step=1.0L-4, points=20
```

```
Computing #3(13): 0.01L0 ... 0.1L0, step=0.001L0, points=20
Computing #4(13): 0.1L0 ... 1.0L0, step=0.01L0, points=20
Computing #5(13): 1.0L0 ... 10.0L0, step=0.1L0, points=20
Computing #6(13): 10.0L0 ... 100.0L0, step=1.0L0, points=20
Computing #7(13): -0.001L0-0.001L0i ... 0.001L0+0.001L0i, step=1.0L-4
Computing #8(13): -0.1L0-0.1L0i ... 0.1L0+0.1L0i, step=0.01L0
Computing #9(13): -1.0L0-1.0L0i ... 1.0L0+1.0L0i, step=0.1L0
Computing #10(13): -3.0L0-3.0L0i ... 3.0L0+3.0L0i, step=0.3L0
Computing #11(13): -6.0L0-6.0L0i ... 6.0L0+6.0L0i, step=0.6L0
Computing #12(13): -10.0L0-10.0L0i ... 10.0L0+10.0L0i, step=1.0L0
Computing #13(13): -100.0L0-100.0L0i ... 100.0L0+100.0L0i, step=10.0L0
Saved image in amoeba3.png
Saved Newton Polytope in amoeba3npt.png
Saved LaTeX in amoeba3.tex

real    0m10.372s
user    0m20.420s
sys     0m0.010s
```



(a) $\mathcal{N}_p$ (b) $\mathcal{A}_p$

Figure 3: Newton polytope and amoeba for $p(z, w)$ in test 3

Test 4

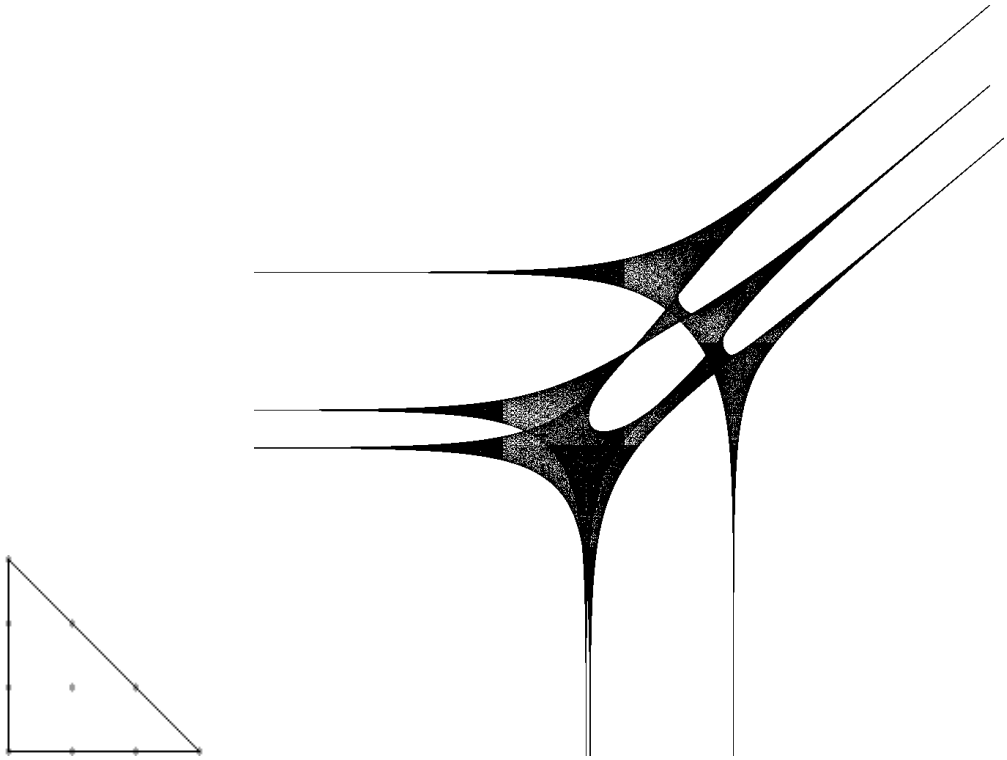
This is a test of the bivariate polynomial

$$p(z,w) = 50z^3 + 83z^2w + 24zw^2 + w^3 + 392z^2 + 414zw + 50w^2 - 28z + 59w - 100.$$

```
<?xml version="1.0" encoding="UTF-8"?>
<amoeba name="Amoeba 4">
  <description>A simple amoeba.</description>
  <image filename="amoeba4.png" width="1024" height="1024"/>
  <latex filename="amoeba4.tex"/>
  <data filename="amoeba4.dat"/>
  <npolytope filename="amoeba4npt.png"/>
  <settings>
```

```
    <max precision="50"/>
    <max error="1e-30"/>
    <max processes="2"/> <!-- use max 2 parallel processes -->
    <min lnZ="-7"/>
    <max lnZ="10"/>
    <min lnW="-7"/>
    <max lnW="10"/>
  </settings>
  <polynomial>
    <!-- p(z,w) = 50z^3 + 83z^2w + 24zw^2 + w^3 + 392z^2 + 414zw + 50w^2 - 28z + 59w - 100 -->
    <term coeff="50" zexp="3" wexp="0"/>
    <term coeff="83" zexp="2" wexp="1"/>
    <term coeff="24" zexp="1" wexp="2"/>
    <term coeff="1" zexp="0" wexp="3"/>
    <term coeff="392" zexp="2" wexp="0"/>
    <term coeff="414" zexp="1" wexp="1"/>
    <term coeff="50" zexp="0" wexp="2"/>
    <term coeff="-28" zexp="1" wexp="0"/>
    <term coeff="59" zexp="0" wexp="1"/>
    <term coeff="-100" zexp="0" wexp="0"/>
  </polynomial>
  <interval type="square" from="-1.0-1.0i" to="1.0+1.0i" step="0.5"/>
  <interval type="circle" minradius="0.0001" maxradius="0.001" step="0.000001" points="20"/>
  <interval type="circle" minradius="0.001" maxradius="0.01" step="0.00001" points="20"/>
  <interval type="circle" minradius="0.01" maxradius="0.1" step="0.0001" points="20"/>
  <interval type="circle" minradius="0.1" maxradius="1.0" step="0.001" points="20"/>
  <interval type="circle" minradius="1.0" maxradius="10.0" step="0.01" points="20"/>
  <interval type="circle" minradius="10.0" maxradius="100.0" step="0.1" points="20"/>
  <interval type="circle" minradius="100.0" maxradius="1000.0" step="1.0" points="20"/>

  <interval type="square" from="-0.001-0.001i" to="0.001+0.001i" step="0.0001"/>
  <interval type="square" from="-0.1-0.1i" to="0.1+0.1i" step="0.01"/>
  <interval type="square" from="-1.0-1.0i" to="1.0+1.0i" step="0.1"/>
  <interval type="square" from="-3.0-3.0i" to="3.0+3.0i" step="0.3"/>
  <interval type="square" from="-6.0-6.0i" to="6.0+6.0i" step="0.6"/>
  <interval type="square" from="-10.0-10.0i" to="10.0+10.0i" step="1.0"/>
  <interval type="square" from="-100.0-100.0i" to="100.0+100.0i" step="10.0"/>
</amoeba>
```

```
$ time amoeba -a amoeba4.xml
Saving data in amoeba4.dat
Using 2 parallel processes.
Computing #1(15): -1.0L0-1.0L0i ... 1.0L0+1.0L0i, step=0.5L0
Computing #2(15): 1.0L-4 ... 0.001L0, step=1.0L-6, points=20
Computing #3(15): 0.001L0 ... 0.01L0, step=1.0L-5, points=20
Computing #4(15): 0.01L0 ... 0.1L0, step=1.0L-4, points=20
Computing #5(15): 0.1L0 ... 1.0L0, step=0.001L0, points=20
Computing #6(15): 1.0L0 ... 10.0L0, step=0.01L0, points=20
Computing #7(15): 10.0L0 ... 100.0L0, step=0.1L0, points=20
Computing #8(15): 100.0L0 ... 1000.0L0, step=1.0L0, points=20
Computing #9(15): -0.001L0-0.001L0i ... 0.001L0+0.001L0i, step=1.0L-4
Computing #10(15): -0.1L0-0.1L0i ... 0.1L0+0.1L0i, step=0.01L0
Computing #11(15): -1.0L0-1.0L0i ... 1.0L0+1.0L0i, step=0.1L0
Computing #12(15): -3.0L0-3.0L0i ... 3.0L0+3.0L0i, step=0.3L0
Computing #13(15): -6.0L0-6.0L0i ... 6.0L0+6.0L0i, step=0.6L0
Computing #14(15): -10.0L0-10.0L0i ... 10.0L0+10.0L0i, step=1.0L0
Computing #15(15): -100.0L0-100.0L0i ... 100.0L0+100.0L0i, step=10.0L0
Saved image in amoeba4.png
Saved Newton Polytope in amoeba4npt.png
Saved LaTeX in amoeba4.tex

real    1m37.054s
user    2m52.130s
sys     0m0.200s
```



(a) $\mathcal{N}_p$ (b) $\mathcal{A}_p$

Figure 4: Newton polytope and amoeba for $p(z, w)$ in test 4

Test 5

This is a test of the bivariate polynomial

$$p(z, w) = z + \cos(w).$$

```
<?xml version="1.0" encoding="UTF-8"?>
<amoeba name="Amoeba 5">
  <description>A longer description of the amoeba of $p(z,w) = z + \cos(w)$.</description>
  <image filename="amoeba5.png" width="1024" height="1024" axes="yes"/>
  <latex filename="amoeba5.tex"/>
  <data filename="amoeba5.dat"/>
  <npolytope filename="amoeba5npt.png"/>
  <settings>
    <max precision="50"/>
    <max error="1e-8"/>
    <min lnZ="-10"/>
    <max lnZ="10"/>
    <min lnW="-10"/>
    <max lnW="15"/>
  </settings>
  <polynomial>
    <!-- p(z,w) = z + cos(w) -->
    <term zexp="1"/>
    <term function="cos" var="w" terms="10"/>
  </polynomial>
  <interval type="circle" minradius="0.0001" maxradius="0.001" step="0.00001" points="3"/>
  <interval type="circle" minradius="0.001" maxradius="0.01" step="0.0001" points="3"/>
  <interval type="circle" minradius="0.01" maxradius="0.1" step="0.001" points="3"/>
```

```
  <interval type="circle" minradius="0.1" maxradius="1.0" step="0.01" points="3"/>
  <interval type="circle" minradius="1.0" maxradius="10.0" step="0.1" points="3"/>
  <interval type="circle" minradius="10.0" maxradius="100.0" step="1.0" points="3"/>
  <interval type="square" from="-0.001-0.001i" to="0.001+0.001i" step="0.0001"/>
  <interval type="square" from="-0.1-0.1i" to="0.1+0.1i" step="0.01"/>
  <interval type="square" from="-1.0-1.0i" to="1.0+1.0i" step="0.1"/>
  <interval type="square" from="-3.0-3.0i" to="3.0+3.0i" step="0.3"/>
  <interval type="square" from="-6.0-6.0i" to="6.0+6.0i" step="0.6"/>
  <interval type="square" from="-10.0-10.0i" to="10.0+10.0i" step="1.0"/>
  <interval type="square" from="-100.0-100.0i" to="100.0+100.0i" step="10.0"/>
</amoeba>
```

```
$ time amoeba -a amoeba5.xml
Saving data in amoeba5.dat
Using 2 parallel processes.
Computing #1(13): 1.0L-4 ... 0.001L0, step=1.0L-5, points=3
Computing #2(13): 0.001L0 ... 0.01L0, step=1.0L-4, points=3
Computing #3(13): 0.01L0 ... 0.1L0, step=0.001L0, points=3
Computing #4(13): 0.1L0 ... 1.0L0, step=0.01L0, points=3
Computing #5(13): 1.0L0 ... 10.0L0, step=0.1L0, points=3
Computing #6(13): 10.0L0 ... 100.0L0, step=1.0L0, points=3
Computing #7(13): -0.001L0-0.001L0i ... 0.001L0+0.001L0i, step=1.0L-4
Computing #8(13): -0.1L0-0.1L0i ... 0.1L0+0.1L0i, step=0.01L0
Computing #9(13): -1.0L0-1.0L0i ... 1.0L0+1.0L0i, step=0.1L0
Computing #10(13): -3.0L0-3.0L0i ... 3.0L0+3.0L0i, step=0.3L0
Computing #11(13): -6.0L0-6.0L0i ... 6.0L0+6.0L0i, step=0.6L0
Computing #12(13): -10.0L0-10.0L0i ... 10.0L0+10.0L0i, step=1.0L0
Computing #13(13): -100.0L0-100.0L0i ... 100.0L0+100.0L0i, step=10.0L0
Saved image in amoeba5.png
Saved Newton Polytope in amoeba5npt.png
Saved LaTeX in amoeba5.tex

real    1m24.249s
user    2m37.750s
sys     0m0.090s
```

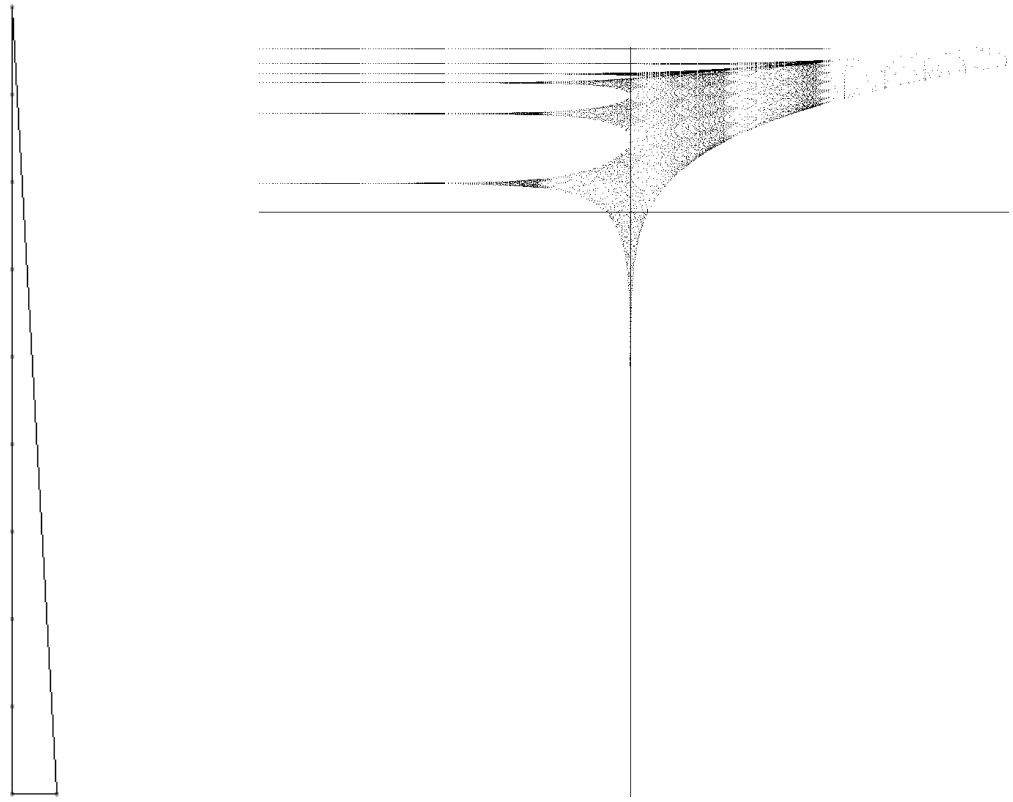
Test 6: Power series

This is a test of the power polynomial

$$p(z,w) = \sum_{j=0}^{4} z(-1)^j \frac{w^{2j+1}}{(2j+1)!},$$

which is actually $z\sin(w)$ over 5 terms.

```
<?xml version="1.0" encoding="UTF-8"?>
<amoeba name="Amoeba 6">
  <description>This sum is actually $z\sin(w)$ over $5$ terms.</description>
  <image filename="amoeba6.png" width="640" height="480"/>
  <latex filename="amoeba6.tex"/>
  <data filename="amoeba6.dat"/>
  <npolytope filename="amoeba6npt.png"/>
  <settings>
    <max precision="20"/>
    <max error="1e-8"/>
  </settings>
  <powerseries from="0" to="4">
    <!-- z*sin(w) := sum((-1)^jzw^{2j+1}/(2j+1)!, n=0..4) -->
    <!-- first term: (-1)^jz -->
    <term coeff="-1" cexp="j" var="z" vexp="1"/>
    <!-- second term: [(2j+1)!]^{-1}w^{2j+1} -->
```



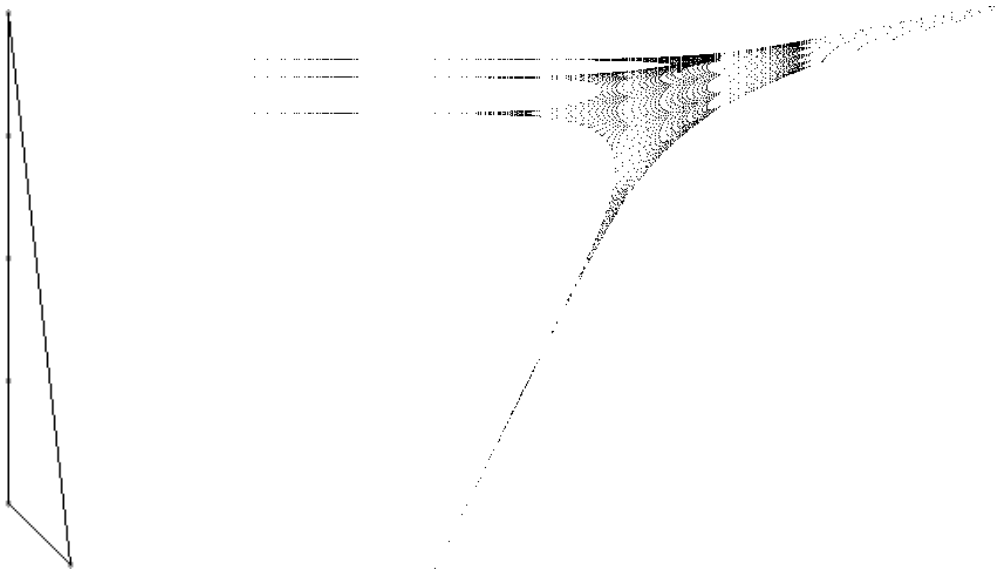
(a) $\mathcal{N}_p$ (b) $\mathcal{A}_p$

Figure 5: Newton polytope and amoeba for $p(z, w)$ in test 5

```
17      <term jcoeff="2j+1" function="fac" cexp="-1" var="w" vexp="2j+1"/>
18    </powerseries>
19    <interval type="square" from="-0.001-0.001i" to="0.001+0.001i" step="0.0001"/>
20    <interval type="square" from="-0.1-0.1i" to="0.1+0.1i" step="0.01"/>
21    <interval type="square" from="-1.0-1.0i" to="1.0+1.0i" step="0.1"/>
22    <interval type="square" from="-3.0-3.0i" to="3.0+3.0i" step="0.3"/>
23    <interval type="square" from="-6.0-6.0i" to="6.0+6.0i" step="0.6"/>
24    <interval type="square" from="-10.0-10.0i" to="10.0+10.0i" step="1.0"/>
25    <interval type="square" from="-100.0-100.0i" to="100.0+100.0i" step="10.0"/>
26  </amoeba>
```

```
$ time amoeba -a amoeba6.xml
Saving data in amoeba6.dat
Using 2 parallel processes.
Computing #1(7): -0.001L0-0.001L0i ... 0.001L0+0.001L0i, step=1.0L-4
Computing #2(7): -0.1L0-0.1L0i ... 0.1L0+0.1L0i, step=0.01L0
Computing #3(7): -1.0L0-1.0L0i ... 1.0L0+1.0L0i, step=0.1L0
Computing #4(7): -3.0L0-3.0L0i ... 3.0L0+3.0L0i, step=0.3L0
Computing #5(7): -6.0L0-6.0L0i ... 6.0L0+6.0L0i, step=0.6L0
Computing #6(7): -10.0L0-10.0L0i ... 10.0L0+10.0L0i, step=1.0L0
Computing #7(7): -100.0L0-100.0L0i ... 100.0L0+100.0L0i, step=10.0L0
Saved image in amoeba6.png
Saved Newton Polytope in amoeba6npt.png
Saved LaTeX in amoeba6.tex

real    0m9.732s
```

```
user    0m17.380s
sys     0m0.030s
```



(a) $\mathcal{N}_p$ (b) $\mathcal{A}_p$

Figure 6: Newton polytope and amoeba for $p(z, w)$ in test 6

5 Discussion

5.1 The result

The program does what it was intended to do and it meets the requirements. We have focused on the core of the amoeba creation, so we have not implement any graphical user interface (GUI). Maybe the natural extension of the program would be to also implement a GUI. But as one gain more confidence with editing the XML-configuration file by hand, one comes to insight that a GUI is not really necessary.

Further extensions and/or improvements might be to extend the list of functions available for polynomials and power series in the XML-configuration file. One other improvement could be more image manipulating features, but with some basic knowledge in using an image-manipulating program, one can always manipulate the image afterwards.

And by modifying the intervals into 3D-structures (i.e box and a ball), maybe not too much work would have to be done to extend the limitation of the two-variate polynomial into a three-variate polynomial. But of course, the time complexity of the computations would increase enourmosly.

5.2 About the work

We started our work by implementing the amoeba-algorithm [F] in Maple. And then worked on the Maple-code until it evolved into a state which was good enough that

we could move it over to C++. During this time we also searched for open source libraries that could and would fit our needs, such as image creation, arbitrary precision numbers, and an appropriate root-finding algorithm.

Why we chose C++ over other programming languages was mainly because of its efficiency, amount of available open source libraries, and fast execution. And we both know the language fairly well, and of course, speed was one major demand from our own side. The platform of development has been GNU/Linux. And the collaboration was done using an excellent version control system, Subversion [SVN]. For documentation we chose LaTeX, and divided the document in different files so we could more easily work on the text independently.

References

[CLN] Class Library for Numbers, `http://www.ginac.de/CLN/`

[F] M. Forsberg, *Amoebas and Laurent Series*, Royal Institute of Technology Department of Mathematics, Doctoral Thesis, 1998, page 63.

[FPT] Mikael Forsberg, Mikael Passare, August Tsikh, *Laurent determinants and arrangements of hyperplane amoebas*, Advances in Math. 151 (2000), 45–70.

[GD] The GD library, `http://www.libgd.org/`

[GKZ] I.M. Gelfand, M.M. Kapranov, A.V. Zelevinsky, *Discriminants, resultants, and multidimensional determinants*, Birkhäuser, 1994.

[HSS] John Hubbard, Dierk Schleicher, Scott Sutherland, *How to Find All Roots of Complex Polynomials by Newton's Method*, `http://citeseer.ist.psu.edu/hubbard00how.html`, 2000 Association of America, 1984, page 3.

[JT] M. A. Jenkins and J. F. Traub, *A three-stage variable-shift iteration for polynomial zeros and its relation to generalized rayleigh iteration*, Numerische Mathematik, Vol. 14. No. 3, 1970, p. 252–263.

[libxml++] The C++ XML parser,
`http://libxmlplusplus.sourceforge.net/`

[O'Rourke] Joseph O'Rourke, *Computational Geometry in C, 2nd ed (2001)*, Cambridge University Press, p.72–86.

[petkovic-weierstrass] Miodrag S. Petkovic, Carsten Carstensen, Miroslav Trajkovíc, *Weierstrass Formula and Zero-Finding Methods (1995)*, `http://citeseer.ist.psu.edu/petkovic95weierstrass.html`

[Press] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, *Numerical Recipes in FORTRAN: The Art of Scientific Computing*, 2nd ed. 1992, Cambridge, England, Cambridge University Press, p. 369.

[ratfun] Polynomials And Rational Functions,
`http://calcrpnpy.sourceforge.net/ratfunManual.html`

[RR] A. Ralston, P. Rabinowitz, *A First Course in Numerical Analysis, 2nd ed. (1978)*, McGraw-Hill, New York. p. 383.

[SVN] The Subversion version control system, `http://subversion.tigris.org/`

[Wilkinson] James H. Wilkinson, *The perfidious polynomial*, in ed. by Gene H. Golub, *Studies in Numerical Analysis*, Mathematical.

A Description of command line options

Our program uses Argp and therefore understands UNIX-style argument vectors. What follows is a description of each command line option.

`-a, --amoeba=AMOEBA.xml`
Plots an amoeba as defined in an XML file.
This option must be included if one wants to generate an amoeba from the polynomial specified in an XML file.

`-h, --height=HEIGHT`
Height of the image to generate from the list.
This is optional and should be used together with the `--plot` or `-p` option. It defines the height (in pixels) of the image that is to be constructed from a file containing x- and y-values.

`-o, --output=IMAGEFILE`
Name of the png file to plot – used in conjunction with `--plot` or `-p`.

`-p, --plot=LISTFILE`
Plots an image from a predefined list of x- and y-values.
This is optional but useful if one has a file containing x- and y-values of floating point numbers.

`-s, --skip`
Skip computing the amoeba, use data file from previous run – used in conjunction with the `-a` option.
This is optional but useful if an amoeba has already been computed but one wants to change the dimensions of the image or add/remove the coordinate axes. All one has to do then is to edit the image dimensions or axes option in the XML file and reconstruct the amoeba using the `-a` or `--amoeba` option together with this option. What will happen is that the program will use the data file from the previous run to construct the amoeba instead of going through the tedious process of polynomial root extraction all over again.

`-v, --verbose`
Verbose mode. Dump values read from XML file etc.

`-w, --width=WIDTH`
Width of the image to generate from the list.

This is optional and should be used together with the `--plot` or `-p` option. It defines the width (in pixels) of the image that is to be constructed from a file containing x- and y-values.

`-?, --help`
Give this help list.

`--usage`
Give a short usage message.

`-V, --version`
Print program version.

Mandatory or optional arguments to long options are also mandatory or optional for any corresponding short options.

B The structure of the XML config file

The configuration of the amoeba is defined as follows, with its default values given. Note: XML tags are case sensitive.

`<amoeba name="Amoeba">...</amoeba>`
Mandatory
Has one optional argument, `name`. This will be the title of the LaTeX document.

`<description>...</description>`
Optional
The description of the amoeba. Default is "An amoeba". This text is also inserted in the LaTeX document.

`<image filename="amoeba.png" width="640" height="480" axes="no">`
Optional
The file name of the generated image. Set `axes="yes"` to also produce coordinate axes.

`<latex filename="amoeba.tex">`
Optional
The name of the generated LaTeX dokument.

`<data filename="amoeba.dat">`
Optional
The name of the data file used when computing the amoeba.

`<npolytope filename="npolytope.png">`
Optional
The file name of the Newton polytope image.

`<settings>...</settings>`
Optional
Must be defined *before* `polynomial`. It should contain the XML tags `<max ...>` and `<min ...>`

`<max precision="50"/>`

Optional

The precision used when computing the amoeba. Valid range is 15 to 1024.

`<max error="1e-30"/>`

Optional

The maximum error used when finding roots. Valid range is 1e−30 to 1e−2.

`<max processes="2"/>`

Optional

The maximum number of parallel processes used when computing. Valid range is 1 to 1024.

`<max lnZ="20.0"/>`

Optional

The maximum lnZ value used when creating the amoeba image.

`<min lnZ="-5.0"/>`

Optional

The minimum lnZ value used when creating the amoeba image.

`<max lnW="10.0"/>`

Optional

The maximum lnW value used when creating the amoeba image.

`<min lnW="-10.0"/>`

Optional

The minimum lnW value used when creating the amoeba image.

`<polynomial>...</polynomial>`

Mandatory

Within this tag you define your polynomial using `<term .../>`.

`<powerseries from="0" to="0">...</powerseries>`

Mandatory

Within this tag you define your power series using `<term .../>`. Valid range of the from– and to–values are 1 to 1024 and that the to–value must be larger than the from–value.

`<term coeff="a+bi" zexp="0" wexp="0"/>`

Part of a polynomial.

A term in the polynomial on the form $(a + bi)z^{zexp}w^{wexp}$.

`<term coeff="a+bi" function="exp|sin|cos" var="z|w" terms="10"/>`

Part of a polynomial.

A term in the polynomial on the form $(a + bi)\exp\,|\,\sin\,|\,\cos(z|w)$, and `terms` are the number of terms used when doing the Taylor expansion of the desired function.

`<term coeff="a+bi" cexp="mj+n" var="z|w" vexp="mj+n"/>`

Part of a power series.

Defines a term in a power series with `coeff` as a complex number, `var` is the variable and `cexp` and `vexp` are valid j-expressions[1].

`<term jcoeff="mj+n" cexp="mj+n" var="z|w" vexp="mj+n"/>`

Part of a power series.

Defines a term in a power series where `var` is the variable and `jcoeff`, `cexp` and `vexp` are valid j-expressions.

`<term jcoeff="mj+n" function="fac|abs" cexp="mj+n" var="z|w" vexp="mj+n"/>`

Part of a power series.

As above, but here the `function` fac (factorial) or abs (absolute value) is applied to `jcoeff`.

`<interval type="square" from="a+bi" to="c+di" step="0.0"/>`

Mandatory

Defines an interval of type square, where `from` is the lower left corner and `to` is the upper right corner, and `step` defines how small or large each step should be.

`<interval type="circle" minradius="0.0" maxradius="0.0" step="0.0" points="0"/>`

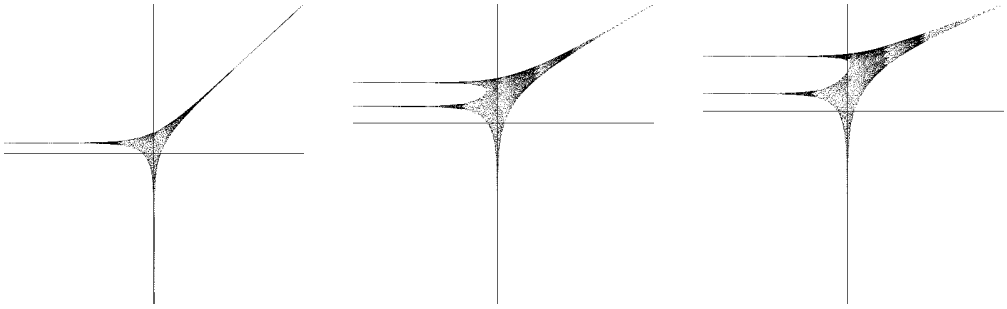
Mandatory

Defines an interval of type circle (actually a disc approximated by a set of circles), with minimum radius `minradius` and maximum radius `maxradius`, `step` defines how small or large each step should be between the radii of each circle, and `points` is the amount of points on the circumference of each circle.

C Taylor tests

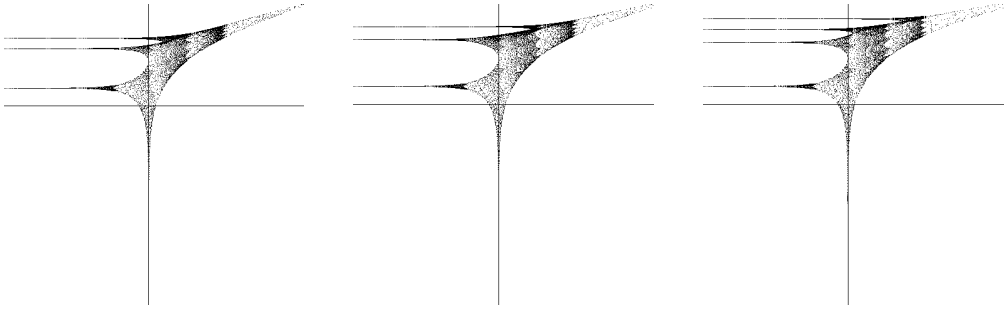
What follows is a series of approximations for the amoeba of $f(z, w) = z + \cos(w)$. The pictures (figures 7 to 12) are the amoebas of $p(z, w) = z + T_k(\cos(w))$ for $k = 2 \ldots 19$, where $T_k(\cos(w))$ is the Taylor approximation for $\cos(w)$ at the origin, having k terms.

[1] An expression on the form $mj + n$ where m and n are integers, e.g. $2j + 1$, 3, $j - 1$, $-j$, $5j$ or -4.



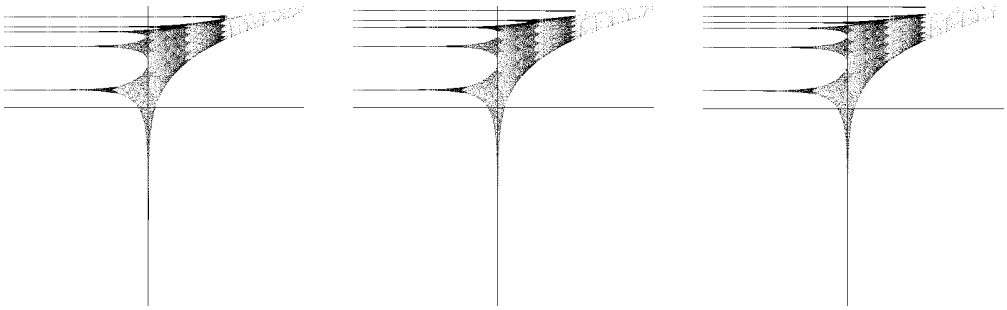
(a) $k = 2$ (b) $k = 3$ (c) $k = 4$

Figure 7: $z + T_k(\cos(w)), \quad k = 2 \ldots 4$



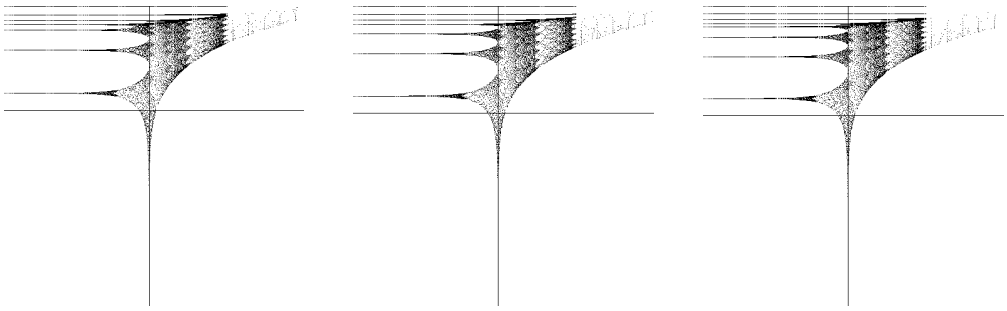
(a) $k = 5$ (b) $k = 6$ (c) $k = 7$

Figure 8: $z + T_k(\cos(w)), \quad k = 5 \ldots 7$



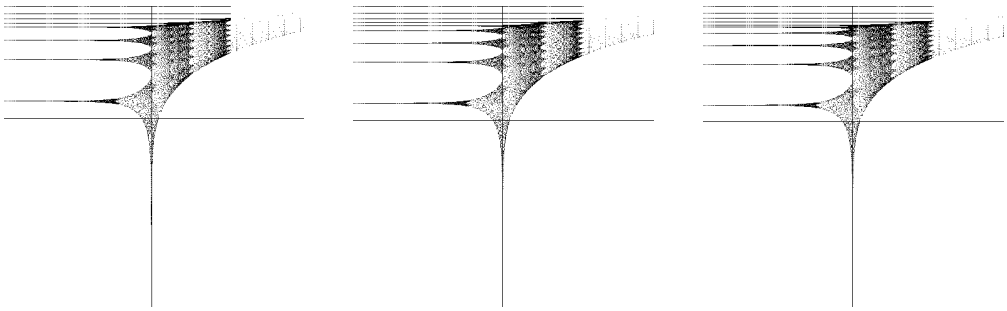
(a) $k = 8$ (b) $k = 9$ (c) $k = 10$

Figure 9: $z + T_k(\cos(w)), \quad k = 8 \ldots 10$



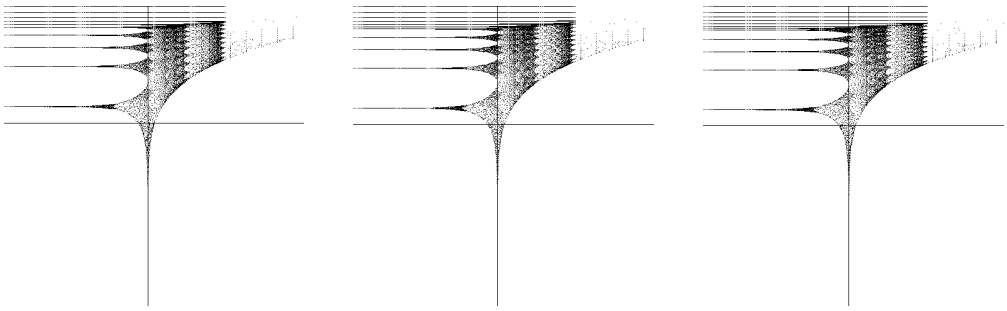
(a) $k = 11$ (b) $k = 12$ (c) $k = 13$

Figure 10: $z + T_k(\cos(w)), \quad k = 11 \ldots 13$



(a) $k = 14$ (b) $k = 15$ (c) $k = 16$

Figure 11: $z + T_k(\cos(w)), \quad k = 14 \ldots 16$



(a) $k = 17$ (b) $k = 18$ (c) $k = 19$

Figure 12: $z + T_k(\cos(w)), \quad k = 17 \ldots 19$

D Source code

amoeba.h

```
1  //
2  // $Id: amoeba.h 1883 2007-08-12 00:19:14Z magnus $
3  //
4  // Project: Amoeba Program
5  // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6  //          Wojciech Komorowski <nmd04wki@student.hig.se>
7  //
8
9  #ifndef AMOEBA_H_
10 #define AMOEBA_H_
11
12 #include <cln/univpoly_complex.h>
13 #include <list>
14 #include <stdexcept>
15 #include <semaphore.h>
16 #include "rootfinders/CPSolver.h"
17 #include "interval.h"
18 #include "point.h"
19 #include "cpolynomial.h"
20
21 using namespace cln;
22
23 // The type of the values in the list of (ln|z|, ln|w|) points that make up
24 // the amoeba
25 typedef pair<double, double> AmoebaListDataType;
26
27 // The list itself
28 class AmoebaList : public list<AmoebaListDataType> {
29 public:
30   void push_back(double x, double y);
31 };
32
33 class Amoeba {
34   int imageWidth_;     // width of the image in pixels
35   int imageHeight_;    // height of the image in pixels
36   int precision_;      // number of bits in the mantissa part of each arbitrary
37                        // precision number that is going to be used
38   cl_R maxError_;
39
40   // the limits on the values of (ln|z|, ln|w|) that make up the amoeba
41   double minLnW_, maxLnW_, minLnZ_, maxLnZ_;
42
43   // this stores the list of (ln|z|, ln|w|) points so that it can be reused
44   // in consecutive runs of the program
45   string dataFileName_;
46
47   // the polynomial root finder
48   CPSolver solver_;
49 public:
50   Amoeba();
51   ~Amoeba();
52
53   // sets the image dimensions and limits on the values of ln|z| and ln|w|
54   void setDimensions(int width, int height, double minLnW, double maxLnW,
55                      double minLnZ, double maxLnZ);
56
57   // sets the amount of mantissa bits for arbitrary precision floating point
58   // variables
59   void setPrecision(int precision);
60
61   void setMaxError(const cl_R& maxError);
```

```
62
63    // sets a name for the data file
64    void setDataFileName(const string& dataFileName);
65
66    /*
67      This method computes the amoeba of the bivariate polynomial p(z, w)
68      by replacing one of the variables by a constant to get an univariate
69      polynomial, say p_z(w). The roots of p_z(w) = 0 are then located, and
70      the pair (ln|z|, ln|root|) is inserted into a list, for all the roots
71      of p_z(w). The same procedure is applied by replacing the other variable
72      by a constant to get p_w(z) = 0 and adding (ln|root, ln|w|) into the list.
73      The set of constant values which replace one of the variables in p(z, w)
74      is taken from an AbstractInterval representing a Square or a Circle.
75      If the AbstractInterval represents a Square, the interval is a rectangular
76      region in the complex plane where the set of constants are evenly
77      distributed. Otherwise it is a disk centered at the origin,
78      modelled as a set of circles centered at the origin, with increasing radius,
79      and the set of constants is evenly distributed along the circumference of
80      each circle.
81    */
82    void compute(const CPolynomial& p, const AbstractInterval* interval,
83                 sem_t* mutex);
84
85    // maps the list of (ln|z|, ln|w|) points to pixels
86    Points mapToImage(bool wantsAxes = false) throw(std::runtime_error);
87    //void mapToImage(const char* fileName) throw(std::runtime_error);
88
89  private:
90
91    // these methods substitute one of the variables in the bivariate polynomial
92    // p(z, w) to create an univariate polynomial p_z(w) or p_w(z)
93    cl_UP_N createP_Z(const CPolynomial& p, const cl_N& w);
94    cl_UP_N createP_W(const CPolynomial& p, const cl_N& z);
95  };
96
97  #endif /* AMOEBA_H_ */
```

amoeba.cpp

```
1   //
2   // $Id: amoeba.cpp 1883 2007-08-12 00:19:14Z magnus $
3   //
4   // Project: Amoeba Program
5   // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6   //          Wojciech Komorowski <nmd04wki@student.hig.se>
7   //
8
9   #include <iostream>
10  #include <sstream>
11  #include <fstream>
12  #include <typeinfo>
13  #include <cln/number.h>
14  #include <cln/complex.h>
15  #include <cln/float.h>
16  #include <cln/real.h>
17  #include <cln/lfloat.h>
18  #include <cln/rational.h>
19  #include <cln/integer.h>
20  #include <cln/univpoly_complex.h>
21  #include <cln/float_io.h>
22  #include <cln/real_io.h>
23  #include <cln/complex_io.h>
24
25  #include "amoeba.h"
26  #include "number.h"
27  #include "complex.h"
```

```
28 #include "square.h"
29 #include "circle.h"
30
31 using namespace std;
32 using namespace cln;
33
34 void AmoebaList::push_back(double x, double y) {
35   list<AmoebaListDataType>::push_back(AmoebaListDataType(x, y));
36 }
37
38 Amoeba::Amoeba() {
39 }
40
41 Amoeba::~Amoeba() {
42 }
43
44 void Amoeba::setDimensions(int width, int height, double minLnW, double maxLnW,
45                            double minLnZ, double maxLnZ) {
46   imageWidth_ = width;
47   imageHeight_ = height;
48   minLnW_ = minLnW;
49   maxLnW_ = maxLnW;
50   minLnZ_ = minLnZ;
51   maxLnZ_ = maxLnZ;
52 }
53
54 void Amoeba::setPrecision(int precision) {
55   precision_ = precision;
56 }
57
58 void Amoeba::setMaxError(const cl_R& maxError) {
59   maxError_ = maxError;
60 }
61
62 void Amoeba::setDataFileName(const string& dataFileName) {
63   dataFileName_ = dataFileName;
64 }
65
66 void Amoeba::compute(const CPolynomial& p, const AbstractInterval* interval,
67                      sem_t* mutex) {
68   cl_inhibit_floating_point_underflow = cl_false;
69
70   // The list of (ln|z|, ln|w|) points that make up the amoeba
71   AmoebaList amoebaList;
72
73   // Determine the type of interval
74   if (typeid(*interval) == typeid(Square))
75
76     // Iterate through the rectangular region in the complex plane
77     for (Number y=imagpart(interval->from()); y <= imagpart(interval->to());
78          y = y + interval->step()) {
79       for (Number x=realpart(interval->from()); x <= realpart(interval->to());
80            x = x + interval->step()) {
81
82         // This variable holds the constant which will substitute one of the
83         // variables in p(z, w) = 0.
84         cl_N subs = complex(x, y);
85
86         // subs cannot be 0 since ln|0| is undefined
87         if (zerop(subs))
88           continue;
89
90         // Now substitute each of the variables in p(z, w) with the
91         // subs constant and thus create two univariate polynomials
92         cl_UP_N p_z = createP_Z(p, subs);
93         cl_UP_N p_w = createP_W(p, subs);
```

```
94
95          // Take the natural logarithm of the absolute value of the
96          // substituted variable
97          double lnAbsSubs = double_approx(ln(abs(subs)));
98
99          // Now find the roots of p_z(w) = 0 and p_w(z) = 0
100         list<cl_N> pz_roots =
101           solver_.getRoots(p_z, float_format(precision_), maxError_);
102         list<cl_N> pw_roots =
103           solver_.getRoots(p_w, float_format(precision_), maxError_);
104
105         list<cl_N>::const_iterator pzIter = pz_roots.begin();
106         list<cl_N>::const_iterator pwIter = pw_roots.begin();
107
108         // For each root of p_w(z) = 0...
109         for (; pzIter != pz_roots.end(); ++pzIter) {
110
111           // If it is 0, ignore it, since ln|0| is undefined
112           if (zerop(*pzIter))
113             continue;
114
115           // Get the natural logarithm of the absolute value of the root
116           double lnAbsRoot = double_approx(ln(abs(*pzIter)));
117
118           // If p(z, w) = 0 but ln|z| or ln|w| falls outside the
119           // user defined limits, then those values should be ignored...
120           if (lnAbsSubs < minLnW_ || lnAbsSubs > maxLnW_)
121             continue;
122           if (lnAbsRoot < minLnZ_ || lnAbsRoot > maxLnZ_)
123             continue;
124
125           // Otherwise, we have a new (ln|z|, ln|w|) point of the amoeba!
126           amoebaList.push_back(lnAbsRoot, lnAbsSubs);
127         }
128         // Now do the same for each root of p_z(w) = 0
129         for (; pwIter != pw_roots.end(); ++pwIter) {
130           if (zerop(*pwIter))
131             continue;
132           double lnAbsRoot = double_approx(ln(abs(*pwIter)));
133           if (lnAbsSubs < minLnZ_ || lnAbsSubs > maxLnZ_)
134             continue;
135           if (lnAbsRoot < minLnW_ || lnAbsRoot > maxLnW_)
136             continue;
137           amoebaList.push_back(lnAbsSubs, lnAbsRoot);
138         }
139       }
140     }
141
142     // The interval is of type Circle
143     else {
144       // Iterate through all the circles
145       for (Number radius = interval->minRadius()
146           ; radius <= interval->maxRadius()
147           ; radius = radius + interval->step()) {
148
149         // Get the amount of substitute variables on each circle
150         int points = interval->points();
151
152         cl_F Pi = pi(float_format(precision_));
153         cl_F random = random_F(Pi);  // 0 <= random < Pi
154
155         // For each substitute variable
156         for (int i = 0; i < points; i++) {
157
158           // Create the substitute!
159           // the +random is used so that the points get more evenly
```

```
160             // distributed around the circumference of each circle
161             cl_N subs = radius*cis(((2*i*Pi)/points)+random);
162
163             // Create the univariate polynomials by substituting one of
164             // the variables in p(z, w)
165             cl_UP_N p_z = createP_Z(p, subs);
166             cl_UP_N p_w = createP_W(p, subs);
167
168             // Find the roots of p_z(w) = 0 and p_w(z) = 0
169             list<cl_N> pz_roots =
170               solver_.getRoots(p_z, float_format(precision_), maxError_);
171             list<cl_N> pw_roots =
172               solver_.getRoots(p_w, float_format(precision_), maxError_);
173             list<cl_N>::const_iterator pzIter = pz_roots.begin();
174             list<cl_N>::const_iterator pwIter = pw_roots.begin();
175
176             // Take the natural logarithm of the absolute value of the
177             // substituted variable
178             double lnAbsSubs = double_approx(ln(abs(subs)));
179
180             // For each root of p_w(z) = 0...
181             for (; pzIter != pz_roots.end(); ++pzIter) {
182
183               // If it is 0, ignore it, since ln|0| is undefined
184               if (zerop(*pzIter))
185                 continue;
186
187               // Get the natural logarithm of the absolute value of the root
188               double lnAbsRoot = double_approx(ln(abs(*pzIter)));
189
190               // If p(z, w) = 0 but ln|z| or ln|w| falls outside the
191               // user defined limits, then those values should be ignored...
192               if (lnAbsSubs < minLnW_ || lnAbsSubs > maxLnW_)
193                 continue;
194               if (lnAbsRoot < minLnZ_ || lnAbsRoot > maxLnZ_)
195                 continue;
196
197               //Otherwise we have a new (ln|z|, ln|w|) point of the amoeba!
198               amoebaList.push_back(lnAbsRoot, lnAbsSubs);
199             }
200             // Now do the same for each root of p_z(w) = 0
201             for (; pwIter != pw_roots.end(); ++pwIter) {
202               if (zerop(*pwIter)) continue;
203               double lnAbsRoot = double_approx(ln(abs(*pwIter)));
204               if (lnAbsSubs < minLnZ_ || lnAbsSubs > maxLnZ_)
205                 continue;
206               if (lnAbsRoot < minLnW_ || lnAbsRoot > maxLnW_)
207                 continue;
208               amoebaList.push_back(lnAbsSubs, lnAbsRoot);
209             }
210           }
211         }
212       }
213
214     // mutex lock
215     //cout << "Waiting..." << endl;
216     if (sem_wait(mutex) != 0) {
217       perror("Mutex wait error");
218       exit(1);
219     }
220     //cout << "  writing... ";
221     // Save to file
222     ofstream os(dataFileName_.c_str(),
223                 ios_base::out | ios_base::binary | ios_base::app);
224     AmoebaList::const_iterator iter = amoebaList.begin();
225     for (; iter != amoebaList.end(); ++iter) {
```

```
226       double x = (*iter).first;
227       double y = (*iter).second;
228       os.write((char*)&x, sizeof(double));
229       os.write((char*)&y, sizeof(double));
230     }
231     os.close();
232     //cout << "done." << endl;
233     // mutex unlock
234     if (sem_post(mutex) != 0) {
235       perror("Mutex post error");
236       exit(1);
237     }
238   }
239
240   //
241   // This method maps the list of (ln|z|, ln|w|) points to an image with
242   // integer pixel coordinates
243   //
244   Points Amoeba::mapToImage(bool wantsAxes) throw(std::runtime_error) {
245     Points points;
246     AmoebaList amoebaList;
247
248     // these are used to determine the minimum and maximum values of
249     // all the ln|z|'s and ln|w|'s in the list, so that the image could
250     // be scaled apropriately
251     double minX, maxX, minY, maxY;
252
253     minX = maxX = minY = maxY = 0;
254
255     // Read points from data file
256     ifstream is(dataFileName_.c_str(), ios_base::binary);
257
258     if (!is)
259       throw std::runtime_error(string("Could not open " + dataFileName_).c_str());
260
261     while (is.good()) {
262       double x;
263       double y;
264       is.read((char*)&x, sizeof(double));
265       is.read((char*)&y, sizeof(double));
266       if (x < minX)
267         minX = x;
268       if (x > maxX)
269         maxX = x;
270       if (y < minY)
271         minY = y;
272       if (y > maxY)
273         maxY = y;
274       amoebaList.push_back(x, y);
275     }
276     is.close();
277
278     double dx = (maxX - minX)/(double)imageWidth_;
279     double dy = (minY - maxY)/(double)imageHeight_;
280
281     /* minX + (pixel_x)*dx = ln|z|
282      * maxY + (pizel_z)*dy = ln|w|
283      * pixel_x = (ln|z| - minX)/dx
284      * pixel_y = (ln|w| - maxY)/dy
285      */
286
287     /**** DEBUG: make coord axes        ****/
288     /**** shall we make this an option?  ***/
289     if (wantsAxes) {
290       double origo_x = -minX/dx;
291       double origo_y = -maxY/dy;
```

```
292       int foox = (int)origo_x;
293       int barx = (int)origo_y;
294
295       for (int i = 0; i < imageHeight_; i++) {
296         Key key(foox, i);
297         points[key] = DEFAULT_COLOR;
298       }
299
300       for (int i = 0; i < imageWidth_; i++) {
301         Key key(i, barx);
302         points[key] = DEFAULT_COLOR;
303       }
304     }
305     /**********************************/
306
307     // MOVE and adjust the points to fit an image structure.
308     while (!amoebaList.empty()) {
309       AmoebaListDataType xy = amoebaList.front();
310       int Xpix = (int)((xy.first - minX)/dx);
311       int Ypix = (int)((xy.second - maxY)/dy);
312       Key key(Xpix, Ypix);
313       points[key] = DEFAULT_COLOR;
314       amoebaList.pop_front();
315     }
316
317     return points;
318   }
319
320   //
321   // Private method createP_Z
322   // Creates a polynomial p(z) from the given polynomial p(z, w)
323   // by substituting w in p(z, w) with a constant
324   //
325   cl_UP_N Amoeba::createP_Z(const CPolynomial& p, const cl_N& w) {
326     cl_univpoly_complex_ring R = find_univpoly_ring(cl_C_ring);
327     cl_UP_N p_z = R->zero();
328     p_z.finalize();
329
330     CPolynomial::const_iterator it;
331     for (it = p.begin(); it != p.end(); ++it) {
332       int z_exp = (*it).first.first;
333       int w_exp = (*it).first.second;
334       cl_UP_N tmp = R->create(z_exp);
335       tmp.set_coeff(z_exp, expt(w, cl_I(w_exp))*(*it).second);
336       tmp.finalize();
337       p_z = p_z + tmp;
338     }
339     return p_z;
340   }
341
342   //
343   // Private method createP_W
344   // Creates a polynomial p(w) from the given polynomial p(z, w)
345   // by substituting z in p(z, w) with a constant
346   //
347   cl_UP_N Amoeba::createP_W(const CPolynomial& p, const cl_N& z) {
348     cl_univpoly_complex_ring R = find_univpoly_ring(cl_C_ring);
349     cl_UP_N p_w = R->zero();
350     p_w.finalize();
351
352     CPolynomial::const_iterator it;
353     for (it = p.begin(); it != p.end(); ++it) {
354       int z_exp = (*it).first.first;
355       int w_exp = (*it).first.second;
356       cl_UP_N tmp = R->create(w_exp);
357       tmp.set_coeff(w_exp, expt(z, cl_I(z_exp))*(*it).second);
```

```
358       tmp.finalize();
359       p_w = p_w + tmp;
360     }
361     return p_w;
362   }
```

circle.h

```
1   //
2   // $Id: circle.h 1684 2007-07-23 14:15:04Z magnus $
3   //
4   // Project: Amoeba Program
5   // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6   //          Wojciech Komorowski <nmd04wki@student.hig.se>
7   //
8
9   #ifndef CIRCLE_H_
10  #define CIRCLE_H_
11
12  #include <queue>
13  #include <cln/complex_io.h>
14  #include "interval.h"
15
16  using namespace std;
17
18  class Circle : public AbstractInterval {
19  public:
20    Circle(const Number& minRadius, const Number& maxRadius, const Number& step, int points) {
21      minRadius_ = minRadius;
22      maxRadius_ = maxRadius;
23      step_ = step;
24      points_ = points;
25    }
26
27    ostream& print(ostream& os) const {
28      os << minRadius_ << " ... " << maxRadius_ << ", step=" << step_ << ", points=" << points_;
29      return os;
30    }
31  };
32
33  #endif /* CIRCLE_H_ */
```

color.h

```
1   //
2   // $Id: color.h 1757 2007-07-27 16:12:28Z cortex $
3   //
4   // Project: Amoeba Program
5   // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6   //          Wojciech Komorowski <nmd04wki@student.hig.se>
7   //
8
9   #ifndef COLOR_H_
10  #define COLOR_H_
11
12  typedef int Color;
13  const Color DEFAULT_COLOR = 0;    // Amoebas will get plotted with this color
14
15  #endif /* COLOR_H_ */
```

complex.h

```
1   //
2   // $Id: complex.h 1437 2007-06-19 22:38:57Z magnus $
3   //
```

```
4  // Project: Amoeba Program
5  // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6  //          Wojciech Komorowski <nmd04wki@student.hig.se>
7  //
8
9  #ifndef COMPLEX_H_
10 #define COMPLEX_H_
11
12 #include <cln/complex.h>
13 //#include "number.h"
14
15 //using namespace cln;
16
17 typedef cln::cl_N Complex;
18
19 /*
20 struct ComplexConverter {
21   static const Number& getRe(const Complex& c) {
22     return the<Number>(realpart(c));
23   }
24   static const Number& getIm(const Complex& c) {
25     return the<Number>(imagpart(c));
26   }
27 };
28 */
29
30 #endif /* COMPLEX_H_ */
```

conffile.h

```
1  //
2  // $Id: conffile.h 1910 2007-08-18 14:30:21Z magnus $
3  //
4  // Project: Amoeba Program
5  // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6  //          Wojciech Komorowski <nmd04wki@student.hig.se>
7  //
8
9  #ifndef CONFFILE_H_
10 #define CONFFILE_H_
11
12 #include <string>
13 #include <list>
14 #include <cln/lfloat.h>
15 #include <cln/complex.h>
16 #include <libxml++/libxml++.h>
17 #include "environment.h"
18 #include "number.h"
19 #include "cpolynomial.h"
20 #include "interval.h"
21
22 //
23 // Class ConfFile
24 //
25 class ConfFile : public Environment, protected xmlpp::SaxParser {
26   bool onDescription_;
27   bool onPowerSeries_;
28   bool firstTime_;
29   bool verbose_;
30   //std::map<int, cln::cl_N> zPoly_;
31   //std::map<int, cln::cl_N> wPoly_;
32
33   // Private methods
34   cln::cl_LF createLF(const string& x);
35   cln::cl_N createComplexLF(const string& x);
36   void createTaylor(const cln::cl_N& coeff, const string& fun,
```

```
37                    const string& var, unsigned int terms);
38    void taylorexpSin(const cln::cl_N& coeff, const string& var,
39                      unsigned int terms);
40    void taylorexpCos(const cln::cl_N& coeff, const string& var,
41                      unsigned int terms);
42    void taylorexpExp(const cln::cl_N& coeff, const string& var,
43                      unsigned int terms);
44    bool validate(const string& exp);
45    int eval(const string& exp, unsigned int j);
46    void createPowerSeries(const cln::cl_N& coeff, const string& var,
47                           const string& cexp, const string& vexp,
48                           const string& jcoeff, const string& fun);
49    void addSeriesToPolyString(const cln::cl_N& coeff, const string& var,
50                               const string& cexp, const string& vexp,
51                               const string& jcoeff, const string& fun);
52    void addToPolyString(const cln::cl_N& coeff, unsigned int zexp,
53                         unsigned int wexp);
54    void addToPolyString(const cln::cl_N& coeff, const string& fun,
55                         const string& var);
56    void setAmoeba(const AttributeList& properties);
57    void setSettings(const AttributeList& properties);
58    void setImage(const AttributeList& properties);
59    void setLatex(const AttributeList& properties);
60    void setData(const AttributeList& properties);
61    void setNPolytope(const AttributeList& properties);
62    void setPolynomial(const AttributeList& properties);
63    void setPowerSeries(const AttributeList& properties);
64    void setTerm(const AttributeList& properties);
65    void setInterval(const AttributeList& properties);
66    void setMax(const AttributeList& properties);
67    void setMin(const AttributeList& properties);
68
69  protected:
70    // Overrides xmlpp:
71    virtual void on_start_document();
72    virtual void on_end_document();
73    virtual void on_start_element(const Glib::ustring& name,
74                                  const AttributeList& properties);
75    virtual void on_end_element(const Glib::ustring& name);
76    virtual void on_characters(const Glib::ustring& characters);
77    virtual void on_comment(const Glib::ustring& text);
78    virtual void on_warning(const Glib::ustring& text);
79    virtual void on_error(const Glib::ustring& text);
80    virtual void on_fatal_error(const Glib::ustring& text);
81
82  public:
83    ConfFile(bool verbose = false);
84    ~ConfFile();
85    // Overrides Environment:
86    //const CPolynomial& getPolynomial();
87    void parseFile(const string& filename);
88  };
89
90  #endif /* CONFFILE_H_ */
```

conffile.cpp

```
1  //
2  // $Id: conffile.cpp 1970 2007-09-04 19:58:53Z magnus $
3  //
4  // Project: Amoeba Program
5  // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6  //          Wojciech Komorowski <nmd04wki@student.hig.se>
7  //
8
9  #include <iostream>
```

```
10 #include <sstream>
11 #include <stdexcept>
12 #include <cln/cln.h>
13 #include <sys/types.h>
14 #include <regex.h>
15 #include "conffile.h"
16 #include "square.h"
17 #include "circle.h"
18
19 using namespace cln;
20
21 //
22 // Constructor
23 //
24 ConfFile::ConfFile(bool verbose) {
25   onDescription_ = false;
26   onPowerSeries_ = false;
27   firstTime_ = true;
28   verbose_ = verbose;
29 }
30
31 //
32 // Destructor
33 //
34 ConfFile::~ConfFile() {
35   // Deallocate intervals
36   while (!intervals_.empty()) {
37     AbstractInterval* i = intervals_.front();
38     delete i;
39     intervals_.pop_front();
40   }
41 }
42
43 //
44 // Private method createLF
45 //
46 // Creates a cl_LF with correct precision.
47 //
48 cl_LF ConfFile::createLF(const string& x) {
49   ostringstream os;
50   //os << x + "L+0_" << precision_;
51   os << x << "_" << precision_;
52
53   return cl_LF(os.str().c_str());
54   /*
55   cl_LF lf(os.str().c_str());
56   cout << "DEBUG: createLF(" << x << ") -> "
57        << os.str() << " -> " << lf << endl;
58   return lf;
59   */
60 }
61
62 //
63 // Private method createComplexLF
64 //
65 // Creates a cl_N with realpart and imagpart as cl_LF.
66 //
67 cl_N ConfFile::createComplexLF(const string& x) {
68   // Read a complex
69   istringstream is(x);
70   cl_N dummy;
71   is >> dummy;
72
73   //float_format_t prec = float_format(precision_);
74
75   // Create cl_LF of the realpart
```

```
76    ostringstream os;
77    os << realpart(dummy);
78    cl_LF a = createLF(os.str());
79    //cl_F a = cl_float(realpart(dummy), prec);
80
81    // Create cl_LF of the imagpart
82    os.str("");
83    os << imagpart(dummy);
84    cl_LF b = createLF(os.str());
85    //cl_F b = cl_float(imagpart(dummy), prec);
86
87    return complex(a, b);
88  }
89
90  //
91  // Private method createTaylor
92  // This method creates the Taylor expansion for some functions so that
93  // functions of the type f(z, w) = z + cos(w) can be approximated by
94  // bivariate polynomials p(z, w) = z + 1 - w^2/2! + ...
95  //
96  void ConfFile::createTaylor(const cl_N& coeff, const string& fun,
97                              const string& var, unsigned int terms) {
98    if (var != "z" && var != "w")
99      throw runtime_error("Unknown variable. Check spelling in XML file");
100
101   if (fun == "sin")
102     taylorexpSin(coeff, var, terms);
103   else if (fun == "cos")
104     taylorexpCos(coeff, var, terms);
105   else if (fun == "exp")
106     taylorexpExp(coeff, var, terms);
107   else
108     throw runtime_error("Unknown function. Check spelling in XML file");
109 }
110
111 //
112 // Creates a Taylor expansion for the sine function
113 // sin z = z - z^3/3! + z^5/5! - ...
114 //
115 void ConfFile::taylorexpSin(const cl_N& coeff, const string& var,
116                             unsigned int terms) {
117   bool negative = false;
118   bool z_var = (var == "z");
119
120   for (unsigned int i = 0; i < terms; i++) {
121     unsigned int exponent = 2*i+1;
122     cl_I fact = factorial(exponent);
123     if (z_var) {
124       cl_N tmp = poly_[Key(exponent, 0)];
125       if (negative)
126         tmp = tmp - coeff/fact;
127       else
128         tmp = tmp + coeff/fact;
129       poly_[Key(exponent, 0)] = tmp;
130     }
131     else {
132       cl_N tmp = poly_[Key(0, exponent)];
133       if (negative)
134         tmp = tmp - coeff/fact;
135       else
136         tmp = tmp + coeff/fact;
137       poly_[Key(0, exponent)] = tmp;
138     }
139     negative = !negative;
140   }
141 }
```

```
142
143 //
144 // Creates a Taylor expansion for the cosine function
145 // cos z = 1 - z^2/2! + z^4/4! - ...
146 //
147 void ConfFile::taylorexpCos(const cl_N& coeff, const string& var,
148                             unsigned int terms) {
149   bool negative = false;
150   bool z_var = (var == "z");
151
152   for (unsigned int i = 0; i < terms; i++) {
153     unsigned int exponent = 2*i;
154     cl_I fact = factorial(exponent);
155     if (z_var) {
156       cl_N tmp = poly_[Key(exponent, 0)];
157       if (negative)
158         tmp = tmp - coeff/fact;
159       else
160         tmp = tmp + coeff/fact;
161       poly_[Key(exponent, 0)] = tmp;
162     }
163     else {
164       cl_N tmp = poly_[Key(0, exponent)];
165       if (negative)
166         tmp = tmp - coeff/fact;
167       else
168         tmp = tmp + coeff/fact;
169       poly_[Key(0, exponent)] = tmp;
170     }
171     negative = !negative;
172   }
173 }
174
175 //
176 // Creates a Taylor expansion for the exponential function
177 // exp z = 1 + x + x^2/2! + x^3/3! + ...
178 //
179 void ConfFile::taylorexpExp(const cl_N& coeff, const string& var,
180                             unsigned int terms) {
181   bool z_var = (var == "z");
182
183   for (unsigned int i = 0; i < terms; i++) {
184     cl_I fact = factorial(i);
185     if (z_var) {
186       cl_N tmp = poly_[Key(i, 0)];
187       tmp = tmp + coeff/fact;
188       poly_[Key(i, 0)] = tmp;
189     }
190     else {
191       cl_N tmp = poly_[Key(0, i)];
192       tmp = tmp + coeff/fact;
193       poly_[Key(0, i)] = tmp;
194     }
195   }
196 }
197
198 //
199 // Private method validate
200 //
201 bool ConfFile::validate(const string& exp) {
202   if (exp == "j" || exp == "-j")
203     return true;
204
205   // This accepts patterns like (n, m integer):
206   // 0, j, -j, j+n, j-n, mj, -mj, mj+n, mj-n, -mj+n, -mj-n, m, -m
207   // Or shortly, [-][m][j][+-][n]
```

```
208   string expr = "^\\([0]\\)\\|\\([-]\\{0,1\\}\\([1-9][0-9]*\\)\\{0,1\\}";
209          expr += "\\)\\{0,1\\}[j]\\{0,1\\}";
210          expr += "\\([+-][1-9][0-9]*\\)\\{0,1\\}$";
211   regex_t preg;
212
213   // Compile the expression
214   if (regcomp(&preg, expr.c_str(), REG_NOSUB) != 0) {
215     // This should never happend!
216     cerr << "regcomp error!" << endl;
217     return false;
218   }
219
220   // Validate the exponent
221   int status = regexec(&preg, exp.c_str(), (size_t)0, NULL, 0);
222   regfree(&preg);
223
224   return status == 0;
225 }
226
227 //
228 // Private method eval
229 //
230 // Evaluates the exponent string to a integer.
231 //
232 // Precondition: That the string exp is a valid
233 // expression according to the validate method.
234 //
235 int ConfFile::eval(const string& exp, unsigned int j) {
236   int m = 1;
237   int n = 0;
238   int xj = 1;
239   string str = exp;
240   int jindex = exp.find_first_of('j');
241
242   if (str == "j")
243     xj = j;
244   else if (str == "-j")
245     xj = -j;
246   else if (jindex == 1 && exp[0] == '-') { // "-j..."
247     str.replace(0, 2, 1, ' ');             // Remove the "-j" substring
248     xj = -j;
249     istringstream is(str);
250     is >> n;                               // Read one integer
251   }
252   else if (jindex == 0) {                  // "j..."
253     str.replace(jindex, 1, 1, ' ');        // Remove the 'j' character
254     xj = j;
255     istringstream is(str);
256     is >> n;                               // Read one integer
257   }
258   else if (jindex > 0) {                   // "...j..."
259     str.replace(jindex, 1, 1, ' ');        // Remove the 'j' character
260     xj = j;
261     istringstream is(str);
262     is >> m;                               // Get the first integer
263     if (is.good())                         // If there is a second integer:
264       is >> n;                             //   get it
265   }
266   else {                                   // No 'j' in exp
267     istringstream is(str);
268     is >> m;                               // Get the first integer
269     if (is.good())                         // If there is a second integer:
270       is >> n;                             //   get it
271   }
272
273   int value = m * xj + n;
```

```
274
275   if (verbose_) {
276     cout << "    j=" << j << ", eval(" << exp << "): " << m << "*" << xj
277          << (n<0?"":"+") << n << "=" << value << endl;
278   }
279
280   return value;
281 }
282
283 //
284 // Private method createPowerSeries
285 //
286 void ConfFile::createPowerSeries(const cl_N& coeff, const string& var,
287                                  const string& cexp, const string& vexp,
288                                  const string& jcoeff, const string& fun) {
289   if (var != "z" && var != "w")
290     throw runtime_error("Unknown variable. Check spelling in XML file");
291
292   bool z_var = (var == "z");
293
294   for (unsigned int j = powerSeriesRange_.first
295     ; j <= powerSeriesRange_.second; j++) {
296
297     int zexp = 0;
298     int wexp = 0;
299
300     if (z_var)
301       zexp = eval(vexp, j);
302     else
303       wexp = eval(vexp, j);
304
305     // Validate exponents
306     if (zexp < 0)
307       throw out_of_range("z-exponent below 0");
308
309     if (wexp < 0)
310       throw out_of_range("w-exponent below 0");
311
312     int cj = eval(cexp, j);
313     cl_I jcoeffValue = 1;
314
315     if (jcoeff != "") {
316       jcoeffValue = eval(jcoeff, j);
317       if (fun != "") {
318         if (fun == "fac")
319           jcoeffValue = factorial(cl_I_to_uint(jcoeffValue));
320         else if (fun == "abs")
321           jcoeffValue = abs(jcoeffValue);
322         else
323           throw runtime_error("Unknown function. Check spelling in XML file");
324       }
325     }
326
327     cl_N tmp;
328
329     /*if (z_var)
330       tmp = zPoly_[zexp];
331     else
332       tmp = wPoly_[wexp];*/
333
334     tmp = poly_[Key(zexp, wexp)];
335
336     if (tmp == complex(0, 0))
337       tmp = complex(1, 0);
338
339     // DEBUG
```

```
340       /*cout << "tmp=" << tmp << ", "
341            << "coeff=" << coeff << ", "
342            << "jcoeffValue=" << jcoeffValue << ", "
343            << "cj=" << cj << endl;*/
344
345       // multiply with current coeff
346       tmp = tmp * expt(jcoeffValue, cj) * expt(coeff, cj);
347
348       /*if (z_var)
349         zPoly_[zexp] = tmp;
350       else
351         wPoly_[wexp] = tmp;*/
352
353       poly_[Key(zexp, wexp)] = tmp;
354
355       if (verbose_) {
356         cout << "    added: (" << tmp
357              << ")" << "z^" << zexp << "w^" << wexp << endl;
358       }
359     }
360   }
361
362   //
363   // Private method addSeriesToPolyString
364   //
365   void ConfFile::addSeriesToPolyString(const cl_N& coeff, const string& var,
366                                        const string& cexp, const string& vexp,
367                                        const string& jcoeff, const string& fun) {
368     ostringstream os;
369
370     // TeXify the power series
371     if (firstTime_) {
372       os << "\\sum_{j=" << powerSeriesRange_.first << "}^{"
373          << powerSeriesRange_.second << "}";
374       firstTime_ = false;
375     }
376
377     if (realpart(coeff) != 1 || imagpart(coeff) != 0)
378       os << "(" << coeff << ")^{" << cexp << "}";
379
380     if (jcoeff != "") {
381       if (fun != "") {
382         if (fun == "fac")
383           os << "[(" << jcoeff << ")!]^{" << cexp << "}";
384         else
385           os << "[\\" << fun << "(" << jcoeff << ")]^{" << cexp << "}";
386       }
387       else
388         os << "(" << jcoeff << ")^{" << cexp << "}";
389     }
390
391     os << var << "^{" << vexp << "}";
392
393     polyString_ += os.str();
394   }
395
396   //
397   // Private method addToPolyString
398   //
399   void ConfFile::addToPolyString(const cl_N& coeff, unsigned int zexp,
400                                  unsigned int wexp) {
401     ostringstream os;
402
403     if (!firstTime_ && realpart(coeff) >= 0)
404       os << "+";
405
```

```
406   if (zexp == 0 && wexp == 0)
407     os << coeff;
408   else if (realpart(coeff) != 1 || imagpart(coeff) != 0)
409     if (imagpart(coeff) == 0)
410       os << coeff;
411     else
412       os << "(" << coeff << ")";
413
414   if (zexp == 1)
415     os << "z";
416   else if (zexp != 0)
417     os << "z^" << zexp;
418
419   if (wexp == 1)
420     os << "w";
421   else if (wexp != 0)
422     os << "w^" << wexp;
423
424   firstTime_ = false;
425
426   polyString_ += os.str();
427 }
428
429 //
430 // Private method addToPolyString
431 //
432 void ConfFile::addToPolyString(const cl_N& coeff, const string& fun,
433                                const string& var) {
434   ostringstream os;
435
436   if (!firstTime_ && realpart(coeff) >= 0)
437     os << "+";
438
439   if (realpart(coeff) != 1 || imagpart(coeff) != 0)
440     os << coeff;
441
442   // TeXify the function
443   os << "\\" << fun << "(" << var << ")";
444
445   firstTime_ = false;
446   polyString_ += os.str();
447 }
448
449 //
450 // Private method setAmoeba
451 //
452 void ConfFile::setAmoeba(const AttributeList& properties) {
453   AttributeList::const_iterator iter = properties.begin();
454   for (; iter != properties.end(); ++iter) {
455     if (iter->name == "name")
456       amoebaName_ = iter->value;
457     else
458       cout << "Warning: Unknown XML entity: " << iter->name << endl;
459   }
460 }
461
462 //
463 // Private method setSettings
464 //
465 void ConfFile::setSettings(const AttributeList& properties) {
466 /*
467   AttributeList::const_iterator iter = properties.begin();
468   for (; iter != properties.end(); ++iter) {
469     //cout << "  Attribute " << iter->name << " = " << iter->value << endl;
470     if (iter->name == "precision")
471       precision_ = atoi(iter->value.c_str());
```

```
472       else if (iter->name == "iterations")
473         iterations_ = atoi(iter->value.c_str());
474       else if (iter->name == "rooterror")
475         ; // TODO: rootError_ = ?
476     }
477   */
478   }
479
480   //
481   // Private method setImage
482   //
483   void ConfFile::setImage(const AttributeList& properties) {
484     AttributeList::const_iterator iter = properties.begin();
485     for (; iter != properties.end(); ++iter) {
486       //cout << "  Attribute " << iter->name << " = " << iter->value << endl;
487       if (iter->name == "filename")
488         imageFileName_ = iter->value;
489       else if (iter->name == "width")
490         imageWidth_ = atoi(iter->value.c_str());
491       else if (iter->name == "height")
492         imageHeight_ = atoi(iter->value.c_str());
493       else if (iter->name == "axes")
494         wantsAxes_ = iter->value == "yes";
495       else
496         cout << "Warning: Unknown XML entity: " << iter->name << endl;
497     }
498   }
499
500   //
501   // Private method setLatex
502   //
503   void ConfFile::setLatex(const AttributeList& properties) {
504     AttributeList::const_iterator iter = properties.begin();
505     for (; iter != properties.end(); ++iter) {
506       //cout << "  Attribute " << iter->name << " = " << iter->value << endl;
507       if (iter->name == "filename")
508         latexFileName_ = iter->value;
509       else
510         cout << "Warning: Unknown XML entity: " << iter->name << endl;
511     }
512   }
513
514   //
515   // Private method setData
516   //
517   void ConfFile::setData(const AttributeList& properties) {
518     AttributeList::const_iterator iter = properties.begin();
519     for (; iter != properties.end(); ++iter) {
520       //cout << "  Attribute " << iter->name << " = " << iter->value << endl;
521       if (iter->name == "filename")
522         dataFileName_ = iter->value;
523       else
524         cout << "Warning: Unknown XML entity: " << iter->name << endl;
525     }
526   }
527
528   //
529   // Private method setNPolytope
530   //
531   void ConfFile::setNPolytope(const AttributeList& properties) {
532     AttributeList::const_iterator iter = properties.begin();
533     for (; iter != properties.end(); ++iter) {
534       //cout << "  Attribute " << iter->name << " = " << iter->value << endl;
535       if (iter->name == "filename")
536         newtonPolytopeFileName_ = iter->value;
537       else
```

```
538         cout << "Warning: Unknown XML entity: " << iter->name << endl;
539     }
540   }
541
542   //
543   // Private method setPolynomial
544   //
545   void ConfFile::setPolynomial(const AttributeList& properties) {
546     if (verbose_)
547       cout << "Parsing polynomial..." << endl;
548
549     AttributeList::const_iterator iter = properties.begin();
550     for (; iter != properties.end(); ++iter) {
551       //cout << "  Attribute " << iter->name << " = " << iter->value << endl;
552     }
553   }
554
555   //
556   // Private method setPowerSeries
557   //
558   void ConfFile::setPowerSeries(const AttributeList& properties) {
559     unsigned int from = 0;
560     unsigned int to = 0;
561     cl_N coeff = complex(1, 0);
562     string var = "";
563
564     if (verbose_)
565       cout << "Parsing power series, ";
566
567     AttributeList::const_iterator iter = properties.begin();
568     for (; iter != properties.end(); ++iter) {
569       if (iter->name == "from")
570         from = atoi(iter->value.c_str());
571       else if (iter->name == "to")
572         to = atoi(iter->value.c_str());
573       else
574         cout << "Warning: Unknown XML entity: " << iter->name << endl;
575     }
576
577     if (verbose_)
578       cout << "j=" << from << ".." << to << endl;
579
580     // Verify some values
581     if (from >= to)
582       throw runtime_error("to-value not larger than from-value");
583
584     if (to > 1024)
585       throw out_of_range("to-value larger than 1024");
586
587     onPowerSeries_ = true;
588     powerSeriesRange_.first = from;
589     powerSeriesRange_.second = to;
590   }
591
592   //
593   // Private method setTerm
594   //
595   void ConfFile::setTerm(const AttributeList& properties) {
596     cl_N coeff = complex(1, 0);
597     unsigned int zexp = 0;
598     unsigned int wexp = 0;
599     bool gotFunction = false;
600     string fun = "";
601     string var = "";
602     string cexp = "j"; // coeff exponent
603     string vexp = "j"; // var exponent
```

```
604     unsigned int terms = 10; // default
605     string jcoeff = "";
606
607     AttributeList::const_iterator iter = properties.begin();
608     for (; iter != properties.end(); ++iter) {
609       if (iter->name == "coeff") {
610         istringstream is(iter->value.c_str());
611         is >> coeff;
612       }
613       else if (iter->name == "jcoeff") {
614         jcoeff = iter->value;
615       }
616       else if (iter->name == "zexp")
617         zexp = atoi(iter->value.c_str());
618       else if (iter->name == "wexp")
619         wexp = atoi(iter->value.c_str());
620       else if (iter->name == "function") {
621         fun = iter->value;
622         gotFunction = true;
623       }
624       else if (iter->name == "var")
625         var = iter->value;
626       else if (iter->name == "terms")
627         terms = atoi(iter->value.c_str());
628       else if (iter->name == "cexp")
629         cexp = iter->value;
630       else if (iter->name == "vexp")
631         vexp = iter->value;
632       else
633         cout << "Warning: Unknown XML entity: " << iter->name << endl;
634     }
635
636     // Print some info
637     if (verbose_)
638       if (onPowerSeries_) {
639         cout << "  adding term: (" << coeff << ")^{" << cexp << "}";
640         if (jcoeff != "") {
641           if (gotFunction)
642             cout << "[" << fun << "(" << jcoeff << ")]^{" << cexp << "}";
643           else
644             cout << "(" << jcoeff << ")^{" << cexp << "}";
645         }
646         if (var != "")
647           cout << var << "^{" << vexp << "}";
648         cout << endl;
649       }
650       else if (gotFunction)
651         cout << "  adding function: (" << coeff << ")"
652              << fun << "(" << var << "), expansion terms = " << terms << endl;
653       else
654         cout << "  adding term: (" << coeff << ")z^"
655              << zexp << "w^" << wexp << endl;
656
657     // Verify some values
658     if (zexp < 0 || zexp > 1024)
659       throw out_of_range("z exponent out of range 0..1024");
660
661     if (wexp < 0 || wexp > 1024)
662       throw out_of_range("w exponent out of range 0..1024");
663
664     if (terms < 0 || terms > 1024)
665       throw out_of_range("terms out of range 0..1024");
666
667     if (!validate(cexp))
668       throw runtime_error("coeff-exponent not a valid j-expression");
669
```

```
670     if (!validate(vexp))
671       throw runtime_error("var-exponent not a valid j-expression");
672
673     if (jcoeff != "" && !validate(jcoeff))
674       throw runtime_error("jcoeff not a valid j-expression");
675
676     // Add to the polynomial
677     if (onPowerSeries_) {
678       addSeriesToPolyString(coeff, var, cexp, vexp, jcoeff, fun);
679       createPowerSeries(coeff, var, cexp, vexp, jcoeff, fun);
680     }
681     else if (gotFunction) {
682       addToPolyString(coeff, fun, var);
683       createTaylor(coeff, fun, var, terms);
684     }
685     else {
686       addToPolyString(coeff, zexp, wexp);
687       poly_[Key(zexp, wexp)] = coeff;
688     }
689   }
690
691   //
692   // Private method setInterval
693   //
694   void ConfFile::setInterval(const AttributeList& properties) {
695     Glib::ustring type;
696     Complex from = complex(0, 0);
697     Complex to = complex(0, 0);
698     Number minRadius = 0.0;
699     Number maxRadius = 0.0;
700     Number step = 0.0;
701     unsigned int points = 0;
702
703     AttributeList::const_iterator iter = properties.begin();
704     for (; iter != properties.end(); ++iter) {
705       if (iter->name == "type")
706         type = iter->value;
707       else if (iter->name == "minradius")
708         minRadius = createLF(iter->value);
709       else if (iter->name == "maxradius")
710         maxRadius = createLF(iter->value);
711       else if (iter->name == "step")
712         step = createLF(iter->value);
713       else if (iter->name == "points")
714         points = atoi(iter->value.c_str());
715       else if (iter->name == "from") {
716         from = createComplexLF(iter->value);
717       }
718       else if (iter->name == "to") {
719         to = createComplexLF(iter->value);
720       }
721       else
722         cout << "Warning: Unknown XML entity: " << iter->name << endl;
723     }
724
725     if (type == "circle")
726       intervals_.push_back(new Circle(minRadius, maxRadius, step, points));
727     else if (type == "square")
728       intervals_.push_back(new Square(from, to, step));
729     else
730       throw runtime_error("Unknown interval type");
731   }
732
733   //
734   // Private method setMax
735   //
```

```
736 void ConfFile::setMax(const AttributeList& properties) {
737   AttributeList::const_iterator iter = properties.begin();
738   for (; iter != properties.end(); ++iter) {
739     //cout << "  Attribute " << iter->name << " = " << iter->value << endl;
740     if (iter->name == "precision")
741       precision_ = atoi(iter->value.c_str());
742     else if (iter->name == "error")
743       maxError_ = cl_R(iter->value.c_str());
744     else if (iter->name == "processes")
745       maxProcesses_ = atoi(iter->value.c_str());
746     else if (iter->name == "lnZ")
747       maxLnZ_ = atof(iter->value.c_str());
748     else if (iter->name == "lnW")
749       maxLnW_ = atof(iter->value.c_str());
750     else
751       cout << "Warning: Unknown XML entity: " << iter->name << endl;
752   }
753 
754   // Validate some data
755   if (precision_ < 15 || precision_ > 1024)
756     throw out_of_range("precision out of range 15..1024");
757 
758   if (maxError_ < cl_R("1e-30") || maxError_ > cl_R("1e-2"))
759     throw out_of_range("max error out of range 1e-30..1e-2");
760 
761   if (maxProcesses_ < 1 || maxProcesses_ > 1024)
762     throw out_of_range("max processes out of range 1..1024");
763 }
764 
765 //
766 // Private method setMin
767 //
768 void ConfFile::setMin(const AttributeList& properties) {
769   AttributeList::const_iterator iter = properties.begin();
770   for (; iter != properties.end(); ++iter) {
771     if (iter->name == "lnZ")
772       minLnZ_ = atof(iter->value.c_str());
773     else if (iter->name == "lnW")
774       minLnW_ = atof(iter->value.c_str());
775     else
776       cout << "Warning: Unknown XML entity: " << iter->name << endl;
777   }
778 }
779 
780 void ConfFile::on_start_document() {
781   //cout << "on_start_document()" << endl;
782 }
783 
784 void ConfFile::on_end_document() {
785   //cout << "on_end_document()" << endl;
786 }
787 
788 void ConfFile::on_start_element(const Glib::ustring& name,
789                                 const AttributeList& attributes) {
790   // Check node name (case sensitive) and call appropriate method.
791   if (name == "amoeba")
792     setAmoeba(attributes);
793   else if (name == "description")
794     onDescription_ = true;
795   else if (name == "settings")
796     setSettings(attributes);
797   else if (name == "image")
798     setImage(attributes);
799   else if (name == "latex")
800     setLatex(attributes);
801   else if (name == "data")
```

```
802       setData(attributes);
803     else if (name == "npolytope")
804       setNPolytope(attributes);
805     else if (name == "polynomial")
806       setPolynomial(attributes);
807     else if (name == "powerseries")
808       setPowerSeries(attributes);
809     else if (name == "term")
810       setTerm(attributes);
811     else if (name == "interval")
812       setInterval(attributes);
813     else if (name == "max")
814       setMax(attributes);
815     else if (name == "min")
816       setMin(attributes);
817     else
818       cerr << "Warning: Unknown XML entity: " << name << endl;
819   }
820
821   void ConfFile::on_end_element(const Glib::ustring& name) {
822     //cout << "on_end_element()" << endl;
823   }
824
825   void ConfFile::on_characters(const Glib::ustring& text) {
826     //cout << "on_characters(): " << text << endl;
827     if (onDescription_) {
828       description_ = text;
829       onDescription_ = false;
830     }
831   }
832
833   void ConfFile::on_comment(const Glib::ustring& text) {
834     //cout << "on_comment(): " << text << endl;
835   }
836
837   void ConfFile::on_warning(const Glib::ustring& text) {
838     //cout << "on_warning(): " << text << endl;
839   }
840
841   void ConfFile::on_error(const Glib::ustring& text) {
842     //cout << "on_error(): " << text << endl;
843   }
844
845   void ConfFile::on_fatal_error(const Glib::ustring& text) {
846     //cout << "on_fatal_error(): " << text << endl;
847   }
848
849   /*
850   const CPolynomial& ConfFile::getPolynomial() {
851     if (onPowerSeries_) {
852       // Construct polynomial from the two series
853       map<int, cl_N>::const_iterator wIter = wPoly_.begin();
854       for (; wIter != wPoly_.end(); ++wIter) {
855         map<int, cl_N>::const_iterator zIter = zPoly_.begin();
856         for (; zIter != zPoly_.end(); ++zIter) {
857           int zexp = zIter->first;
858           int wexp = wIter->first;
859           cl_N prod = zIter->second * wIter->second;
860
861           poly_[Key(zexp, wexp)] = prod;
862
863           if (verbose_)
864             cout << "Added: " << prod << "z^" << zexp << "w^" << wexp << endl;
865         }
866       }
867     }
```

```
868   return poly_;
869 }
870 */
871
872 void ConfFile::parseFile(const string& filename) {
873   set_substitute_entities(true); // what is this and why?
874   parse_file(filename);
875 }
```

cpolynomial.h

```
1  //
2  // $Id: cpolynomial.h 1646 2007-07-20 21:55:49Z magnus $
3  //
4  // Project: Amoeba Program
5  // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6  //          Wojciech Komorowski <nmd04wki@student.hig.se>
7  //
8
9  #ifndef CPOLYNOMIAL_H_
10 #define CPOLYNOMIAL_H_
11
12 #include <map>
13 #include <cln/complex.h>
14 #include <cln/float.h>
15 #include <cln/lfloat.h>
16 #include "point.h" // Key
17
18 /* Does not work?!
19 struct lessKey {
20   bool operator()(const Key& a, const Key& b) const {
21     return (a.first < b.first && a.second < b.second);
22   }
23 };
24 */
25
26 typedef std::map<Key, cln::cl_N> CPolynomial;
27
28 #endif /* CPOLYNOMIAL_H_ */
```

environment.h

```
1  //
2  // $Id: environment.h 1908 2007-08-18 13:53:37Z magnus $
3  //
4  // Project: Amoeba Program
5  // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6  //          Wojciech Komorowski <nmd04wki@student.hig.se>
7  //
8
9  #ifndef ENVIRONMENT_H_
10 #define ENVIRONMENT_H_
11
12 #include <string>
13 #include <list>
14 #include "number.h"
15 #include "cpolynomial.h"
16 #include "interval.h"
17
18 typedef std::pair<unsigned int, unsigned int> UIntPair;
19
20 //
21 // Class Environment
22 //
23 class Environment {
24 protected:
```

```
25    CPolynomial poly_;
26    string polyString_;
27    int imageWidth_;
28    int imageHeight_;
29    string amoebaName_;
30    string description_;
31    string imageFileName_;
32    string latexFileName_;
33    string dataFileName_;
34    string newtonPolytopeFileName_;
35    list<AbstractInterval*> intervals_;
36    unsigned int precision_;
37    Number maxError_;
38    unsigned int maxProcesses_;
39    double minLnW_;
40    double maxLnW_;
41    double minLnZ_;
42    double maxLnZ_;
43    UIntPair powerSeriesRange_;
44    bool wantsAxes_;
45
46  public:
47    Environment();
48    virtual ~Environment();
49    virtual const CPolynomial& getPolynomial();
50    virtual const string& getPolynomialString() const;
51    virtual const list<AbstractInterval*>& getIntervals() const;
52    virtual int getImageWidth() const;
53    virtual int getImageHeight() const;
54    virtual const string& getAmoebaName() const;
55    virtual const string& getDescription() const;
56    virtual const string& getImageFileName() const;
57    virtual const string& getLatexFileName() const;
58    virtual const string& getDataFileName() const;
59    virtual const string& getNewtonPolytopeFileName() const;
60    virtual unsigned int getPrecision() const;
61    virtual const Number& getMaxError() const;
62    virtual unsigned int getMaxProcesses() const;
63    virtual double getMinLnZ() const;
64    virtual double getMaxLnZ() const;
65    virtual double getMinLnW() const;
66    virtual double getMaxLnW() const;
67    virtual const UIntPair& getPowerSeriesRange() const;
68    virtual bool wantsAxes() const;
69    virtual void parseFile(const string& filename) = 0;
70    friend ostream& operator << (ostream& os, const Environment& env);
71  };
72
73  #endif /* ENVIRONMENT_H_ */
```

environment.cpp

```
1   //
2   // $Id: environment.cpp 1908 2007-08-18 13:53:37Z magnus $
3   //
4   // Project: Amoeba Program
5   // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6   //          Wojciech Komorowski <nmd04wki@student.hig.se>
7   //
8
9   #include "environment.h"
10
11  //
12  // Constructor
13  //
14  Environment::Environment() {
```

```
15    // Set some default values
16    polyString_ = "p(z,w)=";
17    imageWidth_ = 640;
18    imageHeight_ = 480;
19    amoebaName_ = "Amoeba";
20    description_ = "An amoeba";
21    imageFileName_ = "amoeba.png";
22    latexFileName_ = "amoeba.tex";
23    dataFileName_ = "amoeba.dat";
24    newtonPolytopeFileName_ = "npolytope.png";
25    precision_ = 50;
26    maxError_ = Number("1.0e-30");
27    maxProcesses_ = 2;
28    minLnW_ = -5.0;
29    maxLnW_ = 20.0;
30    minLnZ_ = -10.0;
31    maxLnZ_ = 10.0;
32    powerSeriesRange_.first = 0;  // from
33    powerSeriesRange_.second = 0; // to
34    wantsAxes_ = false;
35  }
36
37  //
38  // Destructor
39  //
40  Environment::~Environment() {
41  }
42
43  const CPolynomial& Environment::getPolynomial() {
44    return poly_;
45  }
46
47  const string& Environment::getPolynomialString() const {
48    return polyString_;
49  }
50
51  const list<AbstractInterval*>& Environment::getIntervals() const {
52    return intervals_;
53  }
54
55  int Environment::getImageWidth() const {
56    return imageWidth_;
57  }
58
59  int Environment::getImageHeight() const {
60    return imageHeight_;
61  }
62
63  const string& Environment::getDescription() const {
64    return description_;
65  }
66
67  const string& Environment::getAmoebaName() const {
68    return amoebaName_;
69  }
70
71  const string& Environment::getImageFileName() const {
72    return imageFileName_;
73  }
74
75  const string& Environment::getLatexFileName() const {
76    return latexFileName_;
77  }
78
79  const string& Environment::getDataFileName() const {
80    return dataFileName_;
```

```
81  }
82
83  const string& Environment::getNewtonPolytopeFileName() const {
84    return newtonPolytopeFileName_;
85  }
86
87  unsigned int Environment::getPrecision() const {
88    return precision_;
89  }
90
91  const Number& Environment::getMaxError() const {
92    return maxError_;
93  }
94
95  unsigned int Environment::getMaxProcesses() const {
96    return maxProcesses_;
97  }
98
99  double Environment::getMinLnZ() const {
100   return minLnZ_;
101 }
102
103 double Environment::getMaxLnZ() const {
104   return maxLnZ_;
105 }
106
107 double Environment::getMinLnW() const {
108   return minLnW_;
109 }
110
111 double Environment::getMaxLnW() const {
112   return maxLnW_;
113 }
114
115 const UIntPair& Environment::getPowerSeriesRange() const {
116   return powerSeriesRange_;
117 }
118
119 bool Environment::wantsAxes() const {
120   return wantsAxes_;
121 }
122
123 ostream& operator << (ostream& os, const Environment& env) {
124   os << "Amoeba name: " << env.amoebaName_ << endl
125      << "Image file name: " << env.imageFileName_ << endl
126      << "Image width: " << env.imageWidth_ << endl
127      << "Image height: " << env.imageHeight_ << endl
128      << "Image coord axes: " << (env.wantsAxes_?"yes":"no") << endl
129      << "Description: " << env.description_ << endl
130      << "LaTeX file name: " << env.latexFileName_ << endl
131      << "Data file name: " << env.dataFileName_ << endl
132      << "Newton polytope file name: " << env.newtonPolytopeFileName_ << endl
133      << "Precision: " << env.precision_ << endl
134      << "Max error: " << env.maxError_ << endl
135      << "Processes: " << env.maxProcesses_ << endl
136      << "Min ln w: " << env.minLnW_ << endl
137      << "Max ln w: " << env.maxLnW_ << endl
138      << "Min ln z: " << env.minLnZ_ << endl
139      << "Max ln z: " << env.maxLnZ_ << endl
140      << "Polynomial: " << env.polyString_ << endl;
141 }
142
143 /*
144 void Environment::parseFile(const string& filename) {
145   // You must overide this method
146 }
```

```
147 */
```

image.h

```
1  //
2  // $Id: image.h 1845 2007-08-08 01:01:49Z magnus $
3  //
4  // Project: Amoeba Program
5  // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6  //          Wojciech Komorowski <nmd04wki@student.hig.se>
7  //
8
9  #ifndef IMAGE_H_
10 #define IMAGE_H_
11
12 #include "point.h"
13
14 class ImageProducer {
15 public:
16   static void write(const char* filename, const Points& points,
17          int width, int height) throw(std::runtime_error, std::out_of_range);
18 };
19
20 #endif /* IMAGE_H_ */
```

image.cpp

```
1  //
2  // $Id: image.cpp 1848 2007-08-08 02:19:05Z magnus $
3  //
4  // Project: Amoeba Program
5  // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6  //          Wojciech Komorowski <nmd04wki@student.hig.se>
7  //
8
9  #include <gd.h>
10 #include <stdexcept>
11 #include "image.h"
12
13 using namespace std;
14
15 void ImageProducer::write(const char* filename, const Points& points,
16   int width, int height) throw(runtime_error, out_of_range) {
17
18   // Validate image size
19   if (width < 1)
20     throw out_of_range("Image width < 1");
21
22   if (height < 1)
23     throw out_of_range("Image height < 1");
24
25   gdImagePtr im = gdImageCreate(width, height);
26   int white = gdImageColorAllocate(im, 255, 255, 255); // Background color white
27   int black = gdImageColorAllocate(im, 0, 0, 0);
28
29   Points::const_iterator iter = points.begin();
30   for (; iter != points.end(); ++iter) {
31     Key key = (*iter).first;
32     int x = key.first;
33     int y = key.second;
34     if (x >=0 && x < width && y >= 0 && y < height) {
35       //cout << "setPixel (" << x << "," << y << ")" << endl;
36       gdImageSetPixel(im, x, y, black);
37     }
38   }
39
```

```
40    FILE* pngout = fopen(filename, "wb");
41    if (pngout == NULL)
42      throw runtime_error("Could not save data to file!");
43
44    gdImagePng(im, pngout);
45    gdImageDestroy(im);
46  }
```

interval.h

```
1   //
2   // $Id: interval.h 1757 2007-07-27 16:12:28Z cortex $
3   //
4   // Project: Amoeba Program
5   // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6   //          Wojciech Komorowski <nmd04wki@student.hig.se>
7   //
8
9   #ifndef INTERVAL_H_
10  #define INTERVAL_H_
11
12  #include <queue>
13  #include <ostream>
14  #include "complex.h"
15  #include "number.h"
16
17  using namespace std;
18
19  // This class represents a region in the complex plane from which
20  // substitute values will be taken to replace one of the variables
21  // in the bivariate polynomial p(z, w), so that univariate
22  // polynomials can be created as a step in plotting the amoeba
23
24  class AbstractInterval {
25  protected:
26    // If the region is a rectangular
27    Complex from_;
28    Complex to_;
29
30    // If the region is a disk centered at the origin
31    Number minRadius_;
32    Number maxRadius_;
33    Number step_;    // step length between the radiuses
34    int points_;     // amount of points on the circumference of each circle
35  public:
36    // If the region is rectangular, this is the upper left corner
37    virtual const Complex& from() const { return from_; }
38
39    // If the region is rectangular, this is the lower right corner
40    virtual const Complex& to() const { return to_; }
41
42    // If the region is a disk centered at the origin, this is the
43    // minimum radius of the circle-set that models the disk
44    virtual const Number& minRadius() const { return minRadius_; }
45
46    // And this is the corresponding maximum radius
47    virtual const Number& maxRadius() const { return maxRadius_; }
48
49    // This is the difference between each circle radius
50    virtual const Number& step() const { return step_; }
51
52    // Amount of points on the circumference of each circle
53    virtual int points() const { return points_; }
54
55    virtual ostream& print(ostream& os) const = 0;
56    friend ostream& operator<<(ostream& os, const AbstractInterval& i) {
```

```
57      return i.print(os);
58    }
59  };
60
61  #endif /* INTERVAL_H_ */
```

latex.h

```
1   //
2   // $Id: latex.h 1750 2007-07-27 13:55:21Z magnus $
3   //
4   // Project: Amoeba Program
5   // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6   //          Wojciech Komorowski <nmd04wki@student.hig.se>
7   //
8
9   #ifndef LATEX_H_
10  #define LATEX_H_
11
12  #include "environment.h"
13
14  class LatexProducer {
15  public:
16    static void write(const Environment& env) throw(std::runtime_error);
17  };
18
19  #endif /* LATEX_H_ */
```

latex.cpp

```
1   //
2   // $Id: latex.cpp 1923 2007-08-22 21:38:27Z magnus $
3   //
4   // Project: Amoeba Program
5   // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6   //          Wojciech Komorowski <nmd04wki@student.hig.se>
7   //
8
9   #include <iostream>
10  #include <fstream>
11  #include <stdexcept>
12  #include "latex.h"
13
14  using namespace std;
15
16  void LatexProducer::write(const Environment& env) throw(runtime_error) {
17    string fileName = env.getLatexFileName();
18    extern const char* argp_program_version; // quick and dirty solution!
19
20    if (fileName == "") // Empty string -- no file name given:
21      return;           // silently skip this.
22
23    ofstream os(fileName.c_str());
24    if (!os)
25      throw runtime_error("Could not create LaTeX file " + fileName);
26
27    // Produce info about the amoeba
28    os << "%" << endl
29       << "% Created by " << argp_program_version << endl
30       << "% http://amoeba.sajberspejs.com" << endl
31       << "%" << endl
32       << "\\documentclass[a4paper]{article}" << endl
33       //<< "\\usepackage[english]{babel}" << endl
34       //<< "\\usepackage[utf8]{inputenc}" << endl
35       << "\\usepackage[T1]{fontenc}" << endl
36       << "\\usepackage{a4}" << endl
```

```
37        << "\\usepackage{amsmath}" << endl
38        << "\\usepackage{subfigure}" << endl
39        //<< "\\usepackage[pdftex]{graphicx}" << endl
40        << "\\usepackage{graphicx}" << endl
41        << "\\title{" << env.getAmoebaName() << "}" << endl
42        << "\\author{}" << endl
43        << "\\date{}" << endl
44        << "\\begin{document}" << endl
45        << "\\maketitle" << endl
46        << "\\thispagestyle{empty}" << endl
47        << "\\begin{figure}[!hbt]" << endl
48        << "\\centering" << endl
49        << "\\subfigure[$\\mathcal{N}_p$]{"
50        << "\\includegraphics[width=.20\\textwidth]{"
51        << env.getNewtonPolytopeFileName() << "}}" << endl
52        << "\\hfill" << endl
53        << "\\subfigure[$\\mathcal{A}_p$]{"
54        << "\\includegraphics[width=.75\\textwidth]{"
55        << env.getImageFileName() << "}}" << endl
56        << "\\caption{Newton polytope and amoeba for $p(z,w)$}"
57        //<< env.getPolynomialString() << "$}" << endl
58        << "\\end{figure}" << endl
59        << "\\[" << env.getPolynomialString() << "\\]" << endl
60        << "\\paragraph{Description}" << env.getDescription() << "\\\\" << endl;
61
62        // TODO:
63        // Precision/settings used.
64        // ...
65
66    /*const list<AbstractInterval*> intervals = env.getIntervals();
67    list<AbstractInterval*>::const_iterator iter = intervals.begin();
68    int count = 1;
69    for (; iter != intervals.end(); ++iter) {
70      os << "Interval " << count++ << "(" << intervals.size() << "): "
71         << **iter << "\\\\" << endl;
72    }*/
73
74    os << "\\vfill" << endl
75       << "\\noindent Created by " << argp_program_version << "\\\\" << endl
76       << "\\texttt{http://amoeba.sajberspejs.com}" << endl
77       << "\\end{document}" << endl;
78
79    os.close();
80
81    cout << "Saved LaTeX in " << fileName << endl;
82  }
```

listplotter.h

```
1   //
2   // $Id: listplotter.h 1943 2007-09-02 14:48:08Z magnus $
3   //
4   // Project: Amoeba Program
5   // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6   //          Wojciech Komorowski <nmd04wki@student.hig.se>
7   //
8
9   #ifndef LISTPLOTTER_H_
10  #define LISTPLOTTER_H_
11
12  #include <istream>
13  #include <stdexcept>
14
15  class ListPlotter {
16  public:
17    static void plotList(const char* infile, const char* outfile, int width,
```

```
18                          int height) throw(std::runtime_error);
19  private:
20    static void plotList(std::istream& is, const char* outfile, int width,
21                          int height) throw(std::runtime_error);
22  };
23
24  #endif /* LISTPLOTTER_H */
```

listplotter.cpp

```
1   //
2   // $Id: listplotter.cpp 1943 2007-09-02 14:48:08Z magnus $
3   //
4   // Project: Amoeba Program
5   // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6   //          Wojciech Komorowski <nmd04wki@student.hig.se>
7   //
8
9   #include <map>
10  #include <list>
11  #include <string>
12  #include <iostream>
13  #include <fstream>
14  #include <stdexcept>
15  #include "listplotter.h"
16  #include "point.h"
17  #include "color.h"
18  #include "image.h"
19
20  using namespace std;
21
22  //
23  // Method plotList
24  //
25  void ListPlotter::plotList(const char* infile, const char* outfile, int width,
26                             int height) throw(runtime_error) {
27    // Treat input from stdin as special case
28    string str = infile;
29    if (str == "-") {
30      plotList(cin, outfile, width, height);
31      return;
32    }
33
34    ifstream ifile(infile);
35    if (!ifile)
36      throw runtime_error("Could not open input file: " + str);
37
38    cout << "Reading list of x, y values from " << infile << endl;
39    try {
40      plotList(ifile, outfile, width, height);
41    }
42    catch (const exception& e) {
43      ifile.close();
44      throw runtime_error(e.what());
45    }
46
47    ifile.close();
48  }
49
50  //
51  // Private method plotList
52  //
53  void ListPlotter::plotList(istream& is, const char* outfile, int width,
54                             int height) throw(runtime_error) {
55    list<pair<double, double> > values;
56    double minX, maxX, minY, maxY;
```

```
57    minX = maxX = minY = maxY = 0;
58    double xpos, ypos;
59
60    while (!is.eof() ) {
61      is >> xpos;
62      is >> ypos;
63      if (xpos < minX) minX = xpos;
64      if (xpos > maxX) maxX = xpos;
65      if (ypos < minY) minY = ypos;
66      if (ypos > maxY) maxY = ypos;
67      values.push_back(pair<double, double>(xpos, ypos));
68    }
69
70    double dx = (maxX - minX)/(double)width;
71    double dy = (minY - maxY)/(double)height;
72    int intx, inty;
73    map<Key, Color> points;
74    list<pair<double, double> >::const_iterator iter;
75    for (iter = values.begin(); iter != values.end(); iter++) {
76      xpos = (*iter).first;
77      ypos = (*iter).second;
78      intx = (int)((xpos - minX)/dx);
79      inty = (int)((ypos - maxY)/dy);
80      Key key(intx, inty);
81      points[key] = DEFAULT_COLOR;
82    }
83
84    ImageProducer::write(outfile, points, width, height);
85  }
```

npolytope.h

```
1   //
2   // $Id: npolytope.h 1862 2007-08-09 02:01:39Z cortex $
3   //
4   // Project: Amoeba Program
5   // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6   //          Wojciech Komorowski <nmd04wki@student.hig.se>
7   //
8
9   #ifndef NPOLYTOPE_H_
10  #define NPOLYTOPE_H_
11
12  #include <gd.h>
13  #include "cpolynomial.h"
14
15  class NewtonPolytope {
16  private:
17    static double angle(gdPoint p1, gdPoint p2);
18    static bool leftTurn(gdPoint p1, gdPoint p2, gdPoint p3);
19  public:
20    static void write(const char* filename,
21                      const CPolynomial& p) throw(std::runtime_error);
22  };
23
24  #endif /* NPOLYTOPE_H_ */
```

npolytope.cpp

```
1   //
2   // $Id: npolytope.cpp 1920 2007-08-21 02:42:35Z magnus $
3   //
4   // Project: Amoeba Program
5   // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6   //          Wojciech Komorowski <nmd04wki@student.hig.se>
7   //
```

```
8
9  #include <iostream>
10 #include <cmath>
11 #include <stdexcept>
12 #include <deque>
13 #include <gd.h>
14 #include "npolytope.h"
15 #include "point.h"
16
17 using namespace std;
18
19 void NewtonPolytope::write(const char* filename,
20                            const CPolynomial& p) throw(runtime_error) {
21
22   if (filename == "") // Empty string -- no file name given:
23     return;           // silently skip this.
24
25   int width;
26   int height;
27   gdImagePtr im;
28
29   gdPoint points[p.size()];
30   int index = 0;
31   int maxX = 0, maxY = 0;
32   CPolynomial::const_iterator iter = p.begin();
33   for (; iter != p.end(); ++iter) {
34     Key key = (*iter).first;
35     unsigned int zexp = key.first;
36     unsigned int wexp = key.second;
37     points[index].x = zexp;
38     points[index].y = wexp;
39     if (zexp > maxX) maxX = zexp;
40     if (wexp > maxY) maxY = wexp;
41     ++index;
42   }
43
44   // if we can't draw a polytope because of too few
45   // points, then silently skip it
46   if (p.size() < 2)
47     return;
48
49   // find the pivot
50   int pIndex = 0;   // pivot index
51   for (int i = 0; i < p.size(); i++) {
52     if (points[i].y < points[pIndex].y)
53       pIndex = i;
54     else if (points[i].y == points[pIndex].y && points[i].x < points[pIndex].x)
55       pIndex = i;
56   }
57
58   // place the pivot at the beginning
59   if (pIndex != 0) {
60     gdPoint tmp = points[0];
61     points[0] = points[pIndex];
62     points[pIndex] = tmp;
63     pIndex = 0;
64   }
65
66   // sort the rest by increasing angle relative to the pivot
67   int tmpIndex;
68   double tmpAngle;
69   gdPoint tmpPoint;
70   gdPoint pivot = points[0];
71   int j;
72   for (int i = 1; i < p.size(); i++) {
73     tmpIndex = i;
```

```
74      tmpPoint = points[i];
75      tmpAngle = angle(pivot, points[i]);
76      for (j = i; j < p.size(); j++) {
77        if (angle(pivot, points[j]) <= tmpAngle) {
78          tmpPoint = points[j];
79          tmpAngle = angle(pivot, tmpPoint);
80          tmpIndex = j;
81        }
82      }
83      points[tmpIndex] = points[i];
84      points[i] = tmpPoint;
85    }
86
87    // if several points have the same angle relative to the pivot,
88    // delete them except the one with the furthest distance from
89    // the pivot
90    deque<gdPoint> ptsQue;
91    for (int i = 0; i < p.size(); i++) {
92      ptsQue.push_back(points[i]);
93    }
94    {
95      gdPoint p0 = ptsQue[0];
96      gdPoint p1, p2;
97      for (int i = 1; i < ptsQue.size()-1; i++) {
98        p1 = ptsQue[i];
99        p2 = ptsQue[i+1];
100       if (angle(p0, p1) == angle(p0, p2)) {
101         if ((p0.x-p1.x)*(p0.x-p1.x)+(p0.y-p1.y)*(p0.y-p1.y) <
102             (p0.x-p2.x)*(p0.x-p2.x)+(p0.y-p2.y)*(p0.y-p2.y)) {
103           i--;
104           ptsQue.erase(ptsQue.begin() + (i+1));
105         }
106         else {
107           i--;
108           ptsQue.erase(ptsQue.begin() + (i+2));
109         }
110       }
111     }
112   }
113
114   // construct the convex hull
115   deque<gdPoint> hullQue;
116   hullQue.push_back(ptsQue[0]);
117   hullQue.push_back(ptsQue[1]);
118   gdPoint p1, p2;
119   int i = 2;
120   while (i < ptsQue.size()) {
121     p1 = hullQue[hullQue.size()-2];
122     p2 = hullQue[hullQue.size()-1];
123     if (leftTurn(p1, p2, ptsQue[i])) {
124       hullQue.push_back(ptsQue[i]);
125       i++;
126     }
127     else {
128       hullQue.pop_back();
129     }
130   }
131
132   i = hullQue.size();
133   int hullpts = i;
134   gdPoint hull[i];
135   for (i--; i > -1; i--) {
136     hull[i] = hullQue.back();
137     hullQue.pop_back();
138   }
139
```

```
140     // setup the image
141     int scale = 35;
142     int radius = 4;
143     width  = scale*maxX + radius/2 + 3;
144     height = scale*maxY + radius/2 + 3;
145     im = gdImageCreate(width, height);
146     int white = gdImageColorAllocate(im, 255, 255, 255); // Background color white
147     int black = gdImageColorAllocate(im, 0, 0, 0);
148     int grey  = gdImageColorAllocate(im, 153, 153, 153);
149     int blue  = gdImageColorAllocate(im, 0, 0, 255);
150
151     // map all points to the image
152     for (int i = 0; i < p.size(); i++) {
153       int zexp = points[i].x;
154       int wexp = points[i].y;
155       points[i].x = (zexp * scale) + radius/2;
156       points[i].y = height - (wexp * scale) - radius/2 - 1;
157     }
158     for (int i = 0; i < hullpts; i++) {
159       int zexp = hull[i].x;
160       int wexp = hull[i].y;
161       hull[i].x = (zexp * scale) + radius/2;
162       hull[i].y = height - (wexp * scale) - radius/2 - 1;
163     }
164
165     // draw points together with the hull
166     for (int i = 0; i < p.size(); i++) {
167       int cx = points[i].x;
168       int cy = points[i].y;
169       gdImageFilledArc(im, cx, cy, radius, radius, 0, 360, grey, gdArc);
170     }
171     for (int i = 0; i < hullpts-1; i++) {
172       gdImageLine(im, hull[i].x, hull[i].y, hull[i+1].x, hull[i+1].y, black);
173     }
174     gdImageLine(im, hull[hullpts-1].x, hull[hullpts-1].y, points[0].x, points[0].y, black);
175
176     //gdImagePolygon(im, points, p.size(), black);
177
178     FILE* pngout = fopen(filename, "wb");
179     if (pngout == NULL)
180       throw runtime_error("Could not save image to file!");
181
182     gdImagePng(im, pngout);
183     gdImageDestroy(im);
184
185     cout << "Saved Newton Polytope in " << filename << endl;
186   }
187
188   double NewtonPolytope::angle(gdPoint p1, gdPoint p2)
189   {
190     double pi = 3.14159265358979;
191     int Vx = p2.x - p1.x;
192     int Vy = p2.y - p1.y;
193     double norm = sqrt(Vx*Vx + Vy*Vy);
194     if (norm == 0.0) return norm;
195
196     double cosangle = (double)Vx / norm;
197     if (p1.y < p2.y) return acos(cosangle);
198     else if (p1.y > p2.y) return 2*pi*acos(cosangle);
199     else if (p1.x > p2.x) return pi;
200     else return 0.0;
201   }
202
203   bool NewtonPolytope::leftTurn(gdPoint p1, gdPoint p2, gdPoint p3)
204   {
205     return ((p2.x - p1.x)*(p3.y - p1.y) - (p2.y - p1.y)*(p3.x - p1.x)) >= 0;
```

```
206 }
```

number.h

```
1  //
2  // $Id: step.h 944 2007-06-14 01:23:45Z magnus $
3  //
4  // Project: Amoeba Program
5  // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6  //          Wojciech Komorowski <nmd04wki@student.hig.se>
7  //
8
9  #ifndef NUMBER_H_
10 #define NUMBER_H_
11
12 #include <cln/float.h>
13 #include <cln/float_io.h>
14 #include <cln/real.h>
15 #include <cln/real_io.h>
16
17 typedef cln::cl_R Number;
18 //typedef cln::cl_LF Number;
19 //typedef double Number;
20
21 #endif /* NUMBER_H_ */
```

point.h

```
1  //
2  // $Id: point.h 958 2007-06-14 20:18:33Z magnus $
3  //
4  // Project: Amoeba Program
5  // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6  //          Wojciech Komorowski <nmd04wki@student.hig.se>
7  //
8
9  #ifndef POINT_H_
10 #define POINT_H_
11
12 #include <map>
13 #include "color.h"
14
15 typedef std::pair<int, int> Key;
16 typedef std::map<Key, Color> Points;
17
18 #endif /* POINT_H_ */
```

square.h

```
1  //
2  // $Id: square.h 1684 2007-07-23 14:15:04Z magnus $
3  //
4  // Project: Amoeba Program
5  // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6  //          Wojciech Komorowski <nmd04wki@student.hig.se>
7  //
8
9  #ifndef SQUARE_H_
10 #define SQUARE_H_
11
12 #include <queue>
13 #include <cln/complex_io.h>
14 #include "interval.h"
15
16 using namespace std;
```

```
17
18 class Square : public AbstractInterval {
19 public:
20   Square(const Complex& from, const Complex& to, const Number& step) {
21     from_ = from;
22     to_ = to;
23     step_ = step;
24   }
25
26   ostream& print(ostream& os) const {
27     os << from_ << " ... " << to_ << ", step=" << step_;
28     return os;
29   }
30 };
31
32 #endif /* SQUARE_H_ */
```

CPSolver.h

```
1  //
2  // $Id: CPSolver.h 1971 2007-09-04 21:59:17Z cortex $
3  //
4  // CPSolver.cpp
5  //
6  // Project: Amoeba Program
7  // Authors: Magnus Leksell      <nfk03mll@student.hig.se>,
8  //          Wojciech Komorowski <nmd04wki@student.hig.se>
9  //
10 ////////////////////////////////////////////////////////////////////////////
11
12 #ifndef CPSOLVER_H_
13 #define CPSOLVER_H_
14
15 #include <list>
16 #include <cln/complex.h>
17 #include <cln/univpoly_complex.h>
18
19 using namespace std;
20 using namespace cln;
21
22 class CPSolver {
23   private:
24     inline cl_UP_N derive(const cl_UP_N &p, unsigned int n);
25   public:
26     list<cl_N> getRoots(const cl_UP_N &p, float_format_t precision, const cl_R& maxError);
27 };
28
29 #endif /*CPSOLVER_H_*/
```

CPSolver.cpp

```
1  //
2  // $Id: CPSolver.cpp 1984 2007-09-09 23:26:07Z cortex $
3  //
4  // CPSolver.cpp
5  //
6  // Project: Amoeba Program
7  // Authors: Magnus Leksell      <nfk03mll@student.hig.se>,
8  //          Wojciech Komorowski <nmd04wki@student.hig.se>
9  //
10 // Description:
11 // An implementation of the Durand-Kerner method for simultaneously finding
12 // all the roots of polynomials of arbitrary degree with complex coefficients.
13 ////////////////////////////////////////////////////////////////////////////
14
15 #include <list>
```

```
16 #include <iostream>
17
18 #include <cln/abort.h>
19 #include <cln/number.h>
20 #include <cln/complex.h>
21 #include <cln/real.h>
22 #include <cln/float.h>
23 #include <cln/sfloat.h>
24 #include <cln/ffloat.h>
25 #include <cln/dfloat.h>
26 #include <cln/lfloat.h>
27 #include <cln/rational.h>
28 #include <cln/integer.h>
29 #include <cln/univpoly_complex.h>
30
31 #include <cln/complex_io.h>
32 #include <cln/lfloat_io.h>
33
34 #include "CPSolver.h"
35
36 using namespace std;
37 using namespace cln;
38
39 // The Durand-Kerner iterative method for polynomial root finding
40 list<cl_N> CPSolver::getRoots(const cl_UP_N &p, float_format_t precision,
41                                                 const cl_R& maxError)
42 {
43   list<cl_N> roots;             // where the roots will be stored
44   sintL p_degree = degree(p);   // degree of the polynomial
45
46   // solve explicitly for simple polynomials
47   if (p_degree < 1) {
48     cerr << "ERROR: rootfinder called with polynomial degree < 1" << endl;
49     cl_abort();
50   }
51   if (p_degree == 1) {
52     cl_N root = -coeff(p, 0)/coeff(p, 1);
53     roots.push_back(cl_N(root));
54     return roots;
55   }
56   else if (p_degree == 2) {
57     cl_N a = coeff(p, 2);
58     cl_N b = coeff(p, 1);
59     cl_N c = coeff(p, 0);
60     cl_N discroot = sqrt(b*b-4*a*c);
61     roots.push_back(cl_N((-b+discroot)/(2*a)));
62     roots.push_back(cl_N((-b-discroot)/(2*a)));
63     return roots;
64   }
65
66   /* set starting values by distributing them around a circle which is known
67    * to contain all the roots of the polynomial - see (1), end of Section 6.
68    */
69
70   // consider all roots to be a cluster and take its center
71   // to be the mean of all the roots, i.e. -C_(n-1)/(nC_n)
72   //cl_N center = -(coeff(p, p_degree-1)/(p_degree*coeff(p, p_degree)));
73
74 /*
75   // calculate the radius
76   cl_RA one(1);
77   cl_R radius = 1.0;
78   for (int i = 1; i <= p_degree; i++) {
79     cl_UP_N P = derive(p, p_degree-i);
80     cl_R foo = abs(P(center)/coeff(p, p_degree));
81     cl_F n = cl_float((int)p_degree, precision);
```

```
82      cl_I fact = factorial(p_degree-i);
83      n = n/fact;
84      cl_N expr = n*foo;
85      cl_RA blah(i);
86      cl_N exponent = one/blah;
87      cl_R tmp = realpart(expt(expr, exponent));
88      if (tmp > radius)
89        radius = tmp;
90    }
91  */
92
93    // distribute n numbers around the circle...
94    // {z_(1,..,n)} = {center + radius * e^(2ikPi/n), k=1...n-1}
95    cl_N r_approx[p_degree];
96    cl_F Pi = pi(precision);
97    for (int i = 1; i < p_degree; i++)
98      //r_approx[i] = center + radius * cis((2*i*Pi)/p_degree);
99      r_approx[i] = cis((2*i*Pi)/p_degree);
100
101   // now add the final point, so that it doesn't end up on the real axis
102   //r_approx[0] = center + radius * cis(Pi/p_degree);
103   r_approx[0] = cis(Pi/p_degree);
104
105   cl_N r_prod[p_degree];
106
107   // the leading coefficient of the polynomial must be 1 for
108   // this method to converge...
109   cl_univpoly_complex_ring R = find_univpoly_ring(cl_C_ring);
110   cl_UP_N C_n = R->create(0);
111   C_n.set_coeff(0, 1/coeff(p, p_degree));
112   C_n.finalize();
113   cl_UP_N P = C_n*p;
114
115   for (unsigned int iter = 1; iter <= p_degree; iter++) {
116     for (int i = 0; i < p_degree; i++) {
117       cl_N prod = complex(1, 0);
118       for (int j = 0; j < p_degree; j++) {
119         if (j == i) continue;
120         prod = prod * (r_approx[i] - r_approx[j]);
121       }
122       cl_N tmp = r_approx[i];
123       r_approx[i] = tmp - (P(tmp)/prod);
124     }
125   }
126
127   unsigned int iterations = p_degree;
128
129   for (int i = 0; i < p_degree; i++) {
130     cl_R error = abs(p(r_approx[i]));
131     if (error > maxError) {
132       bool convergence;
133       do {
134         convergence = true;
135         for (int i = 0; i < p_degree; i++) {
136           cl_N prod = complex(1, 0);
137           for (int j = 0; j < p_degree; j++) {
138             if (j == i) continue;
139             prod = prod * (r_approx[i] - r_approx[j]);
140           }
141           cl_N tmp = r_approx[i];
142           r_approx[i] = tmp - (P(tmp)/prod);
143           iterations++;
144         }
145         if (iterations >= 65535) {
146           cerr << "Root finder failed after " << iterations << " iterations!" << endl;
147           for (int i = 0; i < p_degree; i++)
```

```
148             roots.push_back(cl_N(r_approx[i]));
149           return roots;
150         }
151         for (int i = 0; i < p_degree; i++) {
152           error = abs(p(r_approx[i]));
153           if (error > maxError) {
154             convergence = false;
155             break;
156           }
157         }
158       }
159       while (!convergence);
160       break;
161     }
162   }
163
164   for (int i = 0; i < p_degree; i++)
165     roots.push_back(cl_N(r_approx[i]));
166
167   return roots;
168 }
169
170 inline cl_UP_N CPSolver::derive(const cl_UP_N &p, unsigned int n)
171 {
172   cl_UP_N tmp = p;
173   for (unsigned int i = 0; i < n; i++)
174     tmp = deriv(tmp);
175   return tmp;
176 }
```

main.cpp

```
1  //
2  // $Id: main.cpp 2099 2007-09-26 21:02:11Z magnus $
3  //
4  // Project: Amoeba Program
5  // Authors: Magnus Leksell <nfk03mll@student.hig.se>,
6  //          Wojciech Komorowski <nmd04wki@student.hig.se>
7  //
8
9  #include <iostream>
10 #include <fstream>
11 #include <sstream>
12 #include <argp.h>
13 #include <stdexcept>
14 #include <semaphore.h>
15 #include <unistd.h>
16 #include <sys/types.h>
17 #include <sys/wait.h>
18 #include <sys/ipc.h>
19 #include <sys/sem.h>
20 #include <sys/shm.h>
21 #include <sys/stat.h>
22 #include "environment.h"
23 #include "conffile.h"
24 #include "amoeba.h"
25 #include "image.h"
26 #include "listplotter.h"
27 #include "latex.h"
28 #include "npolytope.h"
29
30 using namespace std;
31
32 const char *argp_program_version = "Amoeba Generator 1.0";
33 const char *argp_program_bug_address = "<amoeba@sajberspejs.com>";
34 static char doc[] = "Amoeba Generator -- a program for plotting amoebas.";
```

```
35 static char args_doc[] = "";
36
37 static struct argp_option options[] = {
38   {"plot", 'p', "LISTFILE", 0,
39    "Plots an image from a predefined list of x, y values" },
40   {"output", 'o', "IMAGEFILE", 0,
41    "Name of the png file to plot - used in conjunction with --plot or -p" },
42   {"width", 'w', "WIDTH", 0,
43    "Width of the image to generate from the list" },
44   {"height", 'h', "HEIGHT", 0,
45    "Height of the image to generate from the list" },
46   {"amoeba", 'a', "AMOEBA.xml", 0,
47    "Plots an amoeba as defined in an XML file" },
48   {"skip", 's', 0, 0,
49    "Skip computing the amoeba, use data file from previous run - \
50 used in conjunction with -a" },
51   {"verbose", 'v', 0, 0, "Verbose mode. Dump values read from XML file etc."},
52   { 0 }
53 };
54
55 struct arguments {
56   bool plot, amoeba, skip, verbose;
57   char* list_file;
58   char* image_file;
59   int width, height;
60   char* amoeba_file;
61 };
62
63 static error_t parse_opt (int key, char *arg, struct argp_state *state) {
64   struct arguments *arguments = (struct arguments*) state->input;
65   switch (key) {
66     case 'a':
67       arguments->amoeba = true;
68       arguments->amoeba_file = arg;
69       break;
70     case 's':
71       arguments->skip = true;
72       break;
73     case 'v':
74       arguments->verbose = true;
75       break;
76     case 'p':
77       arguments->plot = true;
78       arguments->list_file = arg;
79       break;
80     case 'o':
81       arguments->image_file = arg;
82       break;
83     case 'w':
84       arguments->width = atoi(arg);
85       break;
86     case 'h':
87       arguments->height = atoi(arg);
88       break;
89     case ARGP_KEY_ARG:
90       //if (state->arg_num >= 2)
91         //argp_usage (state);
92       break;
93     case ARGP_KEY_END:
94       if (state->argc < 2)
95         argp_usage (state);
96       //if (state->arg_num < 2)
97         //argp_usage (state);
98       break;
99     default:
100      return ARGP_ERR_UNKNOWN;
```

```
101     }
102     return 0;
103   }
104
105   /* Our argp parser. */
106   static struct argp argp = {options, parse_opt, args_doc, doc};
107
108   int main(int argc, char** argv) {
109     key_t shm_key;
110     int shm_id;
111     sem_t* mutex;
112     struct arguments arguments;
113
114     // Create shared memory
115     shm_id = shmget(IPC_PRIVATE, sizeof(sem_t), IPC_CREAT | 0600);
116     if (shm_id == -1) {
117       perror("Could not get shared memory");
118       return 1;
119     }
120
121     // Attach to shared memory
122     mutex = (sem_t*)shmat(shm_id, (void *)0, 0);
123     if ((long)mutex == -1) {
124       perror("Could not attach to shared memory");
125       return 1;
126     }
127
128     // Create mutex
129     if (sem_init(mutex, 1, 1) != 0) {
130       perror("Could not create mutex");
131       return 1;
132     }
133
134     arguments.plot = false;
135     arguments.amoeba = false;
136     arguments.skip = false;
137     arguments.verbose = false;
138     arguments.list_file = "-";
139     arguments.image_file = "from_list.png";
140     arguments.width = 640;
141     arguments.height = 480;
142     arguments.amoeba_file = "-";
143
144     argp_parse(&argp, argc, argv, 0, 0, &arguments);
145
146     if (arguments.plot)
147       try {
148         // Plot image and then quit
149         ListPlotter::plotList(arguments.list_file, arguments.image_file,
150                               arguments.width, arguments.height);
151         return 0;
152       }
153       catch (const exception& e) {
154         cerr << "Exception caught: " << e.what() << endl;
155         return 1;
156       }
157
158     if (!arguments.amoeba) {
159       cerr << "Could not understand arguments." << endl;
160       return 1;
161     }
162
163     string infile = arguments.amoeba_file;
164     Environment* env = new ConfFile(arguments.verbose);
165
166     // Try parse the config file.
```

```
167   try {
168     env->parseFile(infile);
169     if (arguments.verbose)
170       cout << *env;
171   }
172   catch (const exception& e) {
173     cerr << "Exception caught: " << e.what() << endl;
174     return 1;
175   }
176
177   if (arguments.skip) // Skip computing the amoeba
178     cout << "Skipping computing the amoeba." << endl
179          << "Using data from " << env->getDataFileName() << endl;
180
181   // Set some initial values.
182   int imgWidth = env->getImageWidth();
183   int imgHeight = env->getImageHeight();
184   double minLnW = env->getMinLnW();
185   double maxLnW = env->getMaxLnW();
186   double minLnZ = env->getMinLnZ();
187   double maxLnZ = env->getMaxLnZ();
188
189   Amoeba amoeba;
190   amoeba.setDataFileName(env->getDataFileName());
191   amoeba.setDimensions(imgWidth, imgHeight, minLnW, maxLnW, minLnZ, maxLnZ);
192   amoeba.setPrecision(env->getPrecision());
193   amoeba.setMaxError(env->getMaxError());
194
195   if (!arguments.skip) { // Calculate the amoeba
196     // Remove the data file before computing, if it exists
197     remove(env->getDataFileName().c_str());
198
199     cout << "Saving data in " << env->getDataFileName() << endl
200          << "Using " << env->getMaxProcesses()
201          << " parallel processes." << endl;
202
203     int intervalCount = 0; // Keep track of number of intervals processed
204     int processCount = 0;  // Number of parallel processes currently running
205     CPolynomial poly = env->getPolynomial();
206     const list<AbstractInterval*> intervals = env->getIntervals();
207     list<AbstractInterval*>::const_iterator iter = intervals.begin();
208
209     while (iter != intervals.end()) { // For each interval
210       pid_t child_pid;
211
212       ++intervalCount;
213
214       // Start a new process
215       child_pid = fork();
216
217       if (child_pid == 0) { // Child process
218         cout << "Computing #" << intervalCount
219              << "(" << intervals.size() << "): " << **iter << " " << endl;
220         amoeba.compute(poly, *iter, mutex);
221         if (arguments.verbose)
222           cout << "...done with #" << intervalCount << endl;
223         exit(0); // Finished, just exit child process
224       }
225       else if (child_pid == -1) {
226         perror("Could not start new process");
227         exit(1);
228       }
229
230       // Parent process
231       ++processCount; // Increase number of processes running
232
```

```
233         // If maximum number of processes running,
234         // wait for any process to finish
235         if (processCount == env->getMaxProcesses()) {
236           wait(NULL);     // Waiting for a process to finish
237           --processCount; // Decrease number of processes running
238         }
239
240         ++iter; // Next interval
241
242       } // while intervals...
243
244       // Wait for remaining childs to finish
245       while (processCount > 0) {
246         wait(NULL);
247         --processCount;
248       }
249     } // !arguments.skip
250
251     // Remove the data file when calculations is done
252     //if (!arguments.skip)
253       //remove(env->getDataFileName().c_str());
254
255     // Detach and remove shared memory
256     if (shmdt(mutex) == 0)
257       shmctl(shm_id, IPC_RMID, NULL);
258
259     try {
260       // Map the amoeba to an image structure
261       // and write the image to a file
262       const char* imgfile = env->getImageFileName().c_str();
263       Points points = amoeba.mapToImage(env->wantsAxes());
264       ImageProducer::write(imgfile, points, env->getImageWidth(),
265                            env->getImageHeight());
266       cout << "Saved image in " << imgfile << endl;
267
268       // Produce the Newton polytope image
269       const char* npfile = env->getNewtonPolytopeFileName().c_str();
270       NewtonPolytope::write(npfile, env->getPolynomial());
271
272       // Produce a LaTeX file.
273       LatexProducer::write(*env);
274     }
275     catch (const exception& e) {
276       cerr << "Exception caught: " << e.what() << endl;
277       return 1;
278     }
279
280     // Done!
281     return 0;
282   }
```