Risa/Asir Book, 2003

Nobuki Takayama, Masayuki Noro

December 27, 2003.
Welcome comments to takayama@math.kobe-u.ac.jp and noro@math.kobe-u.ac.jp

This is a partial and experimental translation into English of the Risa/Asir book in Japanese. This English text is based on an intensive introductory lecture by N.Takayama at Hokkaido university. This lecture is

an introduction to a computer algebra system Risa/Asir with topics in solving algebraic equations and hypergeometric differential equations.

1. Tuesday 1: Newton's method (Chapter 3), homotopy method (Chapter **??**).
2. Tuesday 2: Exercise. Chapters 1, 2, 3.
3. Wednesday 1: GCD algorithm (Chapter 6), ring of differential operators in one variable (Chapter 7).
4. Wednesday 2: Exercise. Chapters 4 (Numerical integration), 6 (GCD),
5. Thursday 1: Gröbner basis (Chapter 8).
6. Thursday 2: Exercise. Chapter 8.
7. Friday 1: Linear differential equations for roots of a polynomial (Chapter 9).

Contents

Chapter 1

A Tour of Asir

1.1 Using Risa/Asir as a Calculator

In case of Windows, open the folder (directory) in which Risa/Asir is installed and double click the icon of `asirgui`

`Asirgui` looks like a standard application of Windows, but it is not. Its look and feel are `xterm`+`fep` on unix. In the sequel, `asirgui` will be simply called `asir` and examples are executed on unix.

Risa/Asir outputs the following opening message and waits for an input with the prompt;

[number]

```
bash-2.03$    asir    RETURN

This is Risa/Asir, Version 20010917 (Kobe Distribution).
Copyright (C) 1994-2000, all rights reserved, FUJITSU LABORATORIES LIMITED.
Copyright 2000,2001, Risa/Asir committers, http://www.openxm.org/.
GC 5.3, copyright 1999, H-J. Boehm, A. J. Demers, Xerox, SGI, HP.
PARI 2.0.17(beta), copyright (C) 1989-1999,
  C. Batut, K. Belabas, D. Bernardi, H. Cohen and M. Olivier.

[0]
```

Input `quit;` to terminate the Risa/Asir.

In this section, we will explain how to use Asir as a calulator.

Asir can do calculations not only for numbers, but also for polynomials. Let us see some examples.

Example 1.1 The operators `+,-,*,/,^` represent the sum, the subtraction, the multiplication, the division, and the power respectively. For example, inputting

```
2*(3+5^4);
```

we get the value of $2(3+5^4)$.

```
[0]      2*(3+5^4);      RETURN
1256
```

Example 1.2 `sin(x), cos(x)` are the trigonometric functions sine and cosine. The symbol `@pi` is the constant π.

In order to get approximate values of $\sin(x)$ $\cos(x)$, input as

```
deval(sin(3.14));
```

The function `deval` numerically evaluates the argument in 64 bit floating point arithmetic. As to details, see Chapter **??**.

Example 1.3 Let us evaluate

$$\left\{\left(2+\frac{2}{3}\right)4+\frac{1}{3}\right\}+5$$

by using Asir. Input the formula, `;` (semi-colon), and press RETURN . Then, the evaluation is started. Do not forget to input semi-colon and RETURN . Without these, evaluation will not start. The dollar sign `$` may be used instead of the semi-colon `;`. In this case, the evaluation is performed, but the result will not be printed. In mathematics, `(,)`, `[,]` ,`{,}` are used as brackets in expressions, but in Risa/Asir, only `(,)` can be used as brackets in expressions, and `[,]` and `{,}` are used for different purposes (list and grouping in programs).

```
[0]      ((2+2/3)*4+1/3)+5;      RETURN
16
[1]      ((2+2/3)*4+1/3)+5$      RETURN
[2]
```

Example 1.4 Asir can do calculations for polynomials.

1. Symbols starting small alphabetical character are variables of polynomials. For example, `x2` is the variable of the name x2. The expression `x*2` stands for x times 2.
2. The input *poly* `;` expands the polynomial *poly* .
3. The input `fctr(` *poly* `)` factors *poly* in the ring of polynomials with rational number coefficients.

```
[0]      fctr(x^10-1);      RETURN
[[1,1],[x-1,1],[x+1,1],[x^4+x^3+x^2+x+1,1],[x^4-x^3+x^2-x+1,1]]
[1]      (2*x-1)*(2*x+1);      RETURN
4*x^2-1
[2]      (x+y)^3;      RETURN
y^3+3*x*y^2+3*x^2*y+x^3
```

The output of `fctr` means that $x^{10}-1$ is factored as

$$(x-1)^1,(x+1)^1,(x^4+x^3+x^2+x+1)^1,(x^4-x^3+x^2-x+1)^1.$$

Example 1.5 The command `plot(f);` draws the graph of the function f in the variable x. When you need to specify the range of variables `x`, input, for example
`plot(f,[x,0,10])` Then, the variable `x` runs over $[0,10]$.

```
0
[1]     plot(sin(x));    RETURN
  0
[2]     plot(sin(2*x)+0.5*sin(3*x),[x,-10,10]);    RETURN
```

Example 1.6 If you want to interrupt a computation, type in CTRL+C (C is the initial of cancel). Then, the following message will be displayed.

```
interrupt ?(q/t/c/d/u/w/?)
```

For this message, input u RETURN . The next message will be

```
Abort this computation? (y or n)
```

and input y RETURN to abort the computation. Explanations for (q/t/c/d/u/w/) is displayed if ? is pressed.

```
[0]     fctr(x^1000-y^1000);    RETURN
CTRL+C
        interrupt ?(q/t/c/d/u/w/?)    u    RETURN
        Abort this computation? (y or n)    y    RETURN
return to toplevel
[1]
```

Symbols starting with capital alphabetical characters are *program variables*, which are used to store values. Names of functions defined in programs start with small alphabetical characters. Note that variable symbols starting with small alphabetical characters are variables in polynomials in Risa/Asir and they cannot be used to store values.

Example 1.7

```
[0]     Kazu = 5;    RETURN
5
[1]     Hehehe = x+ 8;    RETURN
x+8
[1]     Kazu+Hehehe;    RETURN
x+13
```

Example 1.8 By using Risa/Asir, evaluate the following values.

$$2+\sqrt{3}, \quad \frac{1}{2-\sqrt{3}}.$$

```
[0]     A=2+deval(3^(1/2));    RETURN
3.73205
[1]     B=1/(2-deval(3^(1/2)));    RETURN
3.73205
[2]     A-B;    RETURN
1.33227e-15
```

1.33227e-15 stands for 1.33227×10^{-15}. Note that we have $2 + \sqrt{3} = \frac{1}{2-\sqrt{3}}$, but the values are different. This is because these non-rational numbers are approximated by floating numbers. As to details, see Chapter **??**.

In this example, the difference is not 0, but the displayed values are the same; 3.73205. The reason for this is that Asir outputs only five digits below 1??? as the default. In order to change the default, input

`ctrl("real_digit",`*digits*`)`

See the example below.

```
[0]     ctrl("real_digit",16);        RETURN
16
[1]     A=2+deval(3^(1/2));           RETURN
3.732050807568877
[2]     B=1/(2-deval(3^(1/2)));       RETURN
3.732050807568876
```

A programming language is installed in Asir. Let us try the most basic programming; repeating and printing.

1. The sentence `for (K=`*initial value*`; K<=`*limit*`; K++) {`*commands*`};` is used to repeat commands. It is called the "for" loop. "`K<=N`" means that "$K \leq N$ holds". A similar expression "`K>=N`" implies that "$K \geq N$ holds" The expression "`K<N`" means that "$K < N$".

2. The expressions `++K` and `K++` mean increasing `K` by 1. In this example, it has the same meaning with `K = K+1` . Similarly `--K` and `K--` mean decreasing `K` by 1.

3. The command `print(` *expr* `)` outputs *expr* .

4. The command `load("`*file name*`");` reads the file *file name* to Risa/Asir. Write the line `end$` at the end of the file. In `asirgui`, load files with the "open" of the file menu.

Example 1.9 [02] By using `for` loop, generate a table of $\sqrt{x}$.

Example-Input 1.1

```
[0]     for (I=0; I<2; I = I+0.2) {    RETURN
            print(I,0); print(" : ",0);    RETURN
            print(deval(I^(1/2)));     RETURN
        }   RETURN
```

Output.

```
0 : 0
0.2 : 0.447214
0.4 : 0.632456
0.6 : 0.774597
0.8 : 0.894427
1 : 1
1.2 : 1.09545
1.4 : 1.18322
1.6 : 1.26491
1.8 : 1.34164
2 : 1.41421
```

The command `print(A)` displays the value of the variable `A` on the screen. The command `print(`*string*`)` outpus the *string* on the screen. The command `print(A,0)` displays the value of the variable `A`, but it does not make the newline. Note that the blank is a character. For example, if you input `A=10; print(A,0); print(A+1); 1011` will be displayed. So, input as `A=10; print(A,0); print(" ",0);print(A+1);` Then, `10 11` will be displayed.

In these examples, if you make a typing mistake, you need to start over again. Therefore, we usually write the inputs in a file by using an editor and load the file to execute the inputs. This method will be explained in the next section.

In this example, the termination condition is `I` < 2, but the case of `I` $= 2$ is executed. In order to understand the reason, we need to study the format of floating point numbers. (See **??** for details). For now, please keep the following in our mind.

> Arithmetics for integers and rational numbers are exact in Risa/Asir, but arithmetics for dicimal numbers are not.

1.2 Asirgui (for Windows)

Asir is launched by double-clicking the icon of `asirgui`

The letters explained below are used to edit input lines.

1. $\boxed{\rightarrow}$ or $\boxed{\texttt{CTRL}}+\boxed{\texttt{f}}$ moves the cursor forward (to the right).
2. $\boxed{\leftarrow}$ or $\boxed{\texttt{CTRL}}+\boxed{\texttt{b}}$ moves the cursor backward (to the left).
3. $\boxed{\uparrow}$ or $\boxed{\texttt{CTRL}}+\boxed{\texttt{p}}$ displays the previous line.
4. $\boxed{\downarrow}$ or $\boxed{\texttt{CTRL}}+\boxed{\texttt{n}}$ displays the next line.
5. $\boxed{\texttt{CTRL}}+\boxed{\texttt{a}}$ moves the cursor to the beginning of the line.
6. $\boxed{\texttt{CTRL}}+\boxed{\texttt{e}}$ moves the cursor to the end of the line.
7. $\boxed{\texttt{CTRL}}+\boxed{\texttt{d}}$ deletes the character on the cursor.
8. $\boxed{\texttt{CTRL}}+\boxed{\texttt{k}}$ deletes the all characters from the cursor position to the end of the line.
9. $\boxed{!}+\boxed{\text{string}}$ displays the latest input line starting with $\boxed{\text{string}}$.
10. $\boxed{!!}$ displays the last input line.

1.3 Writing program with an editor (for Windows)

In order to write a program for Asir in Windows, we need a text editor. For example, we may use the notepad editor in Start → program → accesory. Any text editor may be used to write programs for Asir.

Problem 1.1 Use `http://www.google.com` to find a distribution cite for emacs for Windows. Install the emacs for your machine. It may be a good idea to install cygwin, which is a collection of unix tools and X windows for Windows.

The note pad is launched by

Start→Program→Accesory

(It can also be launched by inputting `shell("notepad");` for the command prompt of `asirgui`). Input your program by the notepad. After finishing the input, save it by

File → save as

The notepad will ask you the folder and the file name to save the program.

To load the file from `asirgui`, use the pulldown menu

File→Open

of the `asirgui`.

1.3.1 Debugger

If an error occurs, a debugger is automatically started with the prompt `(debug)`. The usage of the debugger is discussed later. To exit the debugger, type in `quit`.

1.4 Using cfep/asir for MacOS X by command lines

Chapter 2

Introduction to Risa/Asir

2.1 Short Programs written in Risa/Asir

Risa/Asir has a syntax similar to the programming language C. The authors believe that the superfacial understanding is not enough to become an expert of programming. We must always to try to understand what is going on in the memory. We will present small and easy programs, but we should understand them from the point of view of what is going on in the memory. Let us try to write short programs in Risa/Asir.

For statement is a loop.

Example 2.1 [02] Execute the following program. This program executes `print(K)` five times for K=1, 2, 3, 4, 5.

```
for (K=1; K<=5; K=K+1) { print(K); };
```

Example-Input-Output 2.1

```
[347]  for (K=1; K<=5; K=K+1) { print(K); };
1
2
3
4
5
[348] 0
[349]
```

Example 2.2 [02] Two programs to get the sum of $1+2+\cdots+100$.

```
S = 0;
for (K=1; K<=100; K++) {
   S = S+K;
}
print(S);
```

or

```
def main() {
  S = 0;
  for (K=1; K<=100; K++) {
      S = S+K;
  }
  print(S);
}
main();
```

Execute the two programs above. It is not recommended to input the program directly to asir; Open a file, for example, `a.rr` by an editor and input the program. Load the file `a.rr` by the command `load("./a.rr");` in case of unix. In case of Windows, use the pull-down menu "File → Open".

The `if` sentence is used for conditional branching. The example below numerically evaluates the approximate values of the roots of the quadratic equation $x^2 + bx + c = 0$ for given numbers `b` and `c`. The symbol `@i` stands for $\sqrt{-1}$.

```
B = 1.0; C=3.0;
D = B^2-4*C;
if (D >= 0) {
   DQ = deval(D^(1/2));
   print([ -B/2+DQ/2, -B/2-DQ/2]);
}else {
   DQ = deval((-D)^(1/2));
   print([ -B/2+@i*DQ/2, -B/2-@i*DQ/2]);
}
```

The below is a list of predicates

Symbol	meaning	example
`==`	equal?	`X == 1` ($X = 1$?)
`!=`	not equal?	`X != 1` ($X \neq 1$?)
`!`	not	`!(X==1)` ($X \neq 1$?)
`&&`	and	`(X < 1) && (X > -2)` ($X < 1 and X > -2$?)
`||`	or	`(X > 1) || (X < -2)` ($X > 1 and X < -2$?)

In Asir, the false is expressed by 0 and the true is expressed by a non-zero number. For example,

```
if (1) {
   print("hello");
}else{
   print("bye");
}
```

outputs `hello`.

2.2 Definition of Functions

A collection of procedures or commands can be bundled into one function. The function is defined by

```
def  FunctionName() {
   commands or procedures
}
```

The definition itself does not executes the commands. In order to execute the commands listed in the definition of the function, input as

```
FunctionName();
```

For example, the function `quad` below evaluates approximate values of the roots of the quadratic equation $x^2 + bx + c = 0$ for given numbers b(`B`) and c(`C`).

```
def quad() {
  B = 1.0; C=3.0;
  D = B^2-4*C;
  if (D >= 0) {
     DQ = deval(D^(1/2));
     print([ -B/2+DQ/2, -B/2-DQ/2]);
  }else {
     DQ = deval((-D)^(1/2));
     print([ -B/2+@i*DQ/2, -B/2-@i*DQ/2]);
  }
}
quad();
```

It will be more convinient to deal with the variables `B` and `C` as arguments of the functions `quad` as below. The new function name is `quad2`.

```
def quad2(B,C) {
  D = B^2-4*C;
  if (D >= 0) {
     DQ = deval(D^(1/2));
     print([ -B/2+DQ/2, -B/2-DQ/2]);
  }else {
     DQ = deval((-D)^(1/2));
     print([ -B/2+@i*DQ/2, -B/2-@i*DQ/2]);
  }
}
quad2(1.0,3.0);
quad2(2.0,5.0);
quad2(3.0,1.2);
```

There are sayings in the programming. One of them is

> Divide and Conquer

Use functions to divide procedures into groups and then the program will become more clean. Details on functions will be discussed in Chapter **??**.

Chapter 3

Control Structures and Some Fundamental Algorithms

3.1 Bisection method and Newton's Method

In computer, a complicated function consists of only arithmetics (addition, subtraction, multiplication, division) for numbers. For example, the calculation of $\sqrt{a}$ is performed by arithmetics. There are several methods to obtain approximate values of this number. One method is to find the intersection of $y = x^2 - a$ and $y = 0$. Let $f(x)$ be a differentiable function and consider the problem of finding the coordinate of the intersection of

$$y = f(x)$$

and $y = 0$ approximately.

The tangent of $y = f(x)$ at the point $x = x_n$ is

$$y - f(x_n) = f'(x_n)(x - x_n)$$

Therefore the x-coordinate of the intersection of the tangent and $y = 0$ is

$$x_n - f(x_n)/f'(x_n).$$

Consider a sequence of numbers x_k defined by the reccurrence

$$x_{k+1} = x_k - f(x_k)/f'(x_k).$$

Take x_0 sufficiently close to a root of $f(x) = 0$. Determine the sequence x_k by the reccurrence relation above, then x_k will converge to r when f' is not equal to 0. See the Figure 3.1. This method by which to find a root of $f(x) = 0$ is called Newton's method. In Newtoh's method, an initial value must be properly chosen. If it is properly chosen, the sequence rapidly converges to a root. If an initial values is not properly chosen, the sequence does not converges.

If an initial value of the sequence is properly chosen, Newton's method finds the root of $f(x) = 0$ near the initial value. The following is a program to find an approximate value of $\sqrt{x}$ by Newton's method.

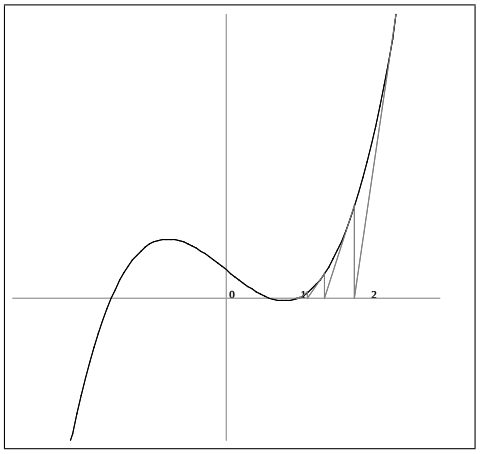

Figure 3.1: Example that Newton's method converges.

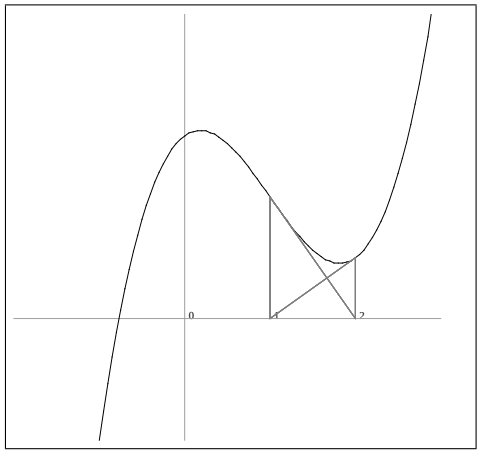

Figure 3.2: Example that Newton's method does not converge.

```
def sqrtByNewton(A) {
  Epsilon = 0.0001;
  P = 0.0;
  Q = deval(A);
  while (!( (Q-P > -Epsilon) &&
            (Q-P < Epsilon))) {
     P = Q;
     print(Q);
     Q = P-(P*P-A)/(2.0*P);
  }
  return(Q);
}
```

For example, $\sqrt{2}$ is evaluated as follows.

```
[422] sqrtByNewton(2);
2
1.5
1.41667
1.41422
1.41421
```

In this program `P` stands for x_k and `Q` stands for x_{k+1}. The expression
`Q = P-(P*P-A)/(2.0*P);`
is the reccurrence relation. Let us study the numerical error of Newton's method

Let α be the root and x_n sequence of Newton's method. The numerical error is denoted by

$$\varepsilon_n = x_n - \alpha.$$

We will prove that ε_{n+1} is approximated by $\frac{\varepsilon_n^2}{2}\frac{f''(\alpha)}{f'(\alpha)}$ by using Taylor expansions. @@@to be continued.

3.2 Maximal value and Arrays

An array is an indexed variable. In Asir, the data type vector may be used as an array.

1. How to create an array.

```
[0] A = newvect(5);
[ 0 0 0 0 0 ]
[1] B = newvect(3,[1,2,3]);
[ 1 2 3 ]
```

Create an array (vector) of the length 5. Put it `A`.
Create an array (vector) of the length 3 with the initial values [1,2,3]. It is equivalent to `B=newvect(3);` and `B[0] = 1`, `B[1] = 2`, `B[2] = 3`

2. Refering to an element of an array (vector).

```
[3] B[2];
3
[4] C = (B[0]+B[1]+B[2])/3;
2
[5] C;
2
```

The third element of `B` is `B[3]`(`B[0]` is the first element, `B[1]` is the second element).

2 is in the variable `C`.

3. How to obtain the length of a vector?

```
[8] size(A);
[5]
[9] size(A)[0];
5
```

The builtin function to get the size of a vector. It returns a list.
The first element of the list is the length of the vector.

As an application of the data structure array (vector), let us write a program to obtain the maximum of data in the vector.

```
/* cond8.rr */
def main() {
    Total=5;
    A = newvect(Total);
    for (K=0; K<Total; K++) {
      A[K] = random() % 100;
    }
    print(A);
    /* search the maximum */
    Max = A[0];
    for (K=0; K<Total; K++) {
      if (A[K] > Max) {
         Max = A[K];
      }
    }
    print("Maximum is ",0);
    print(Max);
}
```

Random data are generated by the function `random`. They are stored in the vector `A`. The result is as follows.

```
[450] load("cond8.rr");
1
[453] main();
[ 58 10 55 62 2 ]
Maximum is 62
0
```

The function `newvect(N)` creates the vector of the length `N`. The function `random()` returns random numbers. `A % N` is the remainder of `A` by `N`.

Chapter 4

Numerical Solution of Ordinary Differential Equations

4.1 The Method of Finite Difference

When h is small, the differential $f'(t)$ is approximated by

$$\frac{f(t+h)-f(t)}{h}.$$

The second differential $f''(t)$ is approximated by

$$\frac{f(t+h)-2f(t)+f(t-h)}{h^2} \tag{4.1}$$

which can be proved by the Taylor expansion formula of $f(t)$. By approximating differentials by formulas above, we can tranlate problems of solving differential equations into those of solving difference equations.

As a running example, let us solve numerically the oscillation equation

$$\frac{d^2}{dt^2}y + y = 0.$$

Let A and B be constants. The general solution of the oscillation equation is written as $A\cos(t) + B\sin(t)$. We will solve the oscillation equation numerically. The advantage of numerical methods is that even when solutions cannot be expressed in terms of special functions like cos and sin, (approximate) solutions can be expressed explicitely. It follows from (4.1) that

$$\frac{y(t+h)-2y(t)+y(t-h)}{h^2} + y(t)$$

is approximately equal to 0 when h is sufficiently small. Then, we may assume that

$$y(t+h) = 2y(t) - y(t-h) - h^2 y(t)$$

holds approximately. Define a sequence Y_k by $Y_k = y(kh)$, $k = 0, 1, 2, \ldots$. Then, we have an approximate equality

$$Y_{k+2} = 2Y_{k+1} - Y_k - h^2 Y_k$$

Therefore, by solving the difference equation

$$y_{k+2} = 2y_{k+1} - y_k - h^2 h_{k+1} \tag{4.2}$$

we will be able to obtain approximate solution of the oscillation equation. This method is called the method of finite difference. The difference equation (4.2) is called the discretization or the difference scheme.

In order to determine the solutions of the difference equation, we need two initial values y_0 and y_1. They can be obtained by $y(0) = a, y'(0) = b$ and the relations

$$y(0) = a = y_0,\ y'(0) = b \simeq \frac{y_1 - y_0}{h}.$$

Example 4.1 Solve numerically the oscillation equation with an outer force

$$\frac{d^2}{dt^2}y + y = \sin(at).$$

Here, $\sin(at)$ is the outer force.

```
load("glib")$

def osci() {
  glib_window(0,-5,50,5);
  glib_clear();
  glib_line(0,0,50,0);
  glib_line(0,-10,0,10);

  X1 = 0.5; X2 = 0.501; A=0.5;

  Dt = 0.07; T = 0;
  while (T<50) {
     X3 = 2*X2-X1+Dt*Dt*(eval(sin(A*T))-X2);
     glib_putpixel(T,X1);
    /*     print([T,X1]); */
     T=T+Dt;
     X1=X2; X2=X3;
  }
}

print("Type in osci()")$
end$
```

4.2 Unstability of Numerical Solutions

The method of finite difference is not all-mighty. Solutions of the discretized equation are sometimes different with solutions of the original equations. Let us see an example.

Consider the equation

$$\frac{d}{dt}y = -2y, \quad y(0) = 1$$

The solutions is e^{-2t}. Since

$$\lim_{h\to 0} \frac{f(t+h) - f(t-h)}{2h} = f'(t)$$

holds, we may discretize the differential equation as follows.

$$y_{k+2} = y_k - 4hy_{k+1}, \quad y_0 = 1,\ y_1 = y_0 + (-2)h$$

4.3 Convergence Theorem of Numerical Solutions

```
load("glib")$

def un() {
  glib_window(0,-5,10,5);
  glib_clear();
  glib_line(0,0,10,0);
  glib_line(0,-10,0,10);

  X1 = 1.0;
  Dt = 0.01; T = 0.0;
  X2 = 1.0-2*Dt;
  while (T<50) {
     X3 = X1 - 4*Dt*X2;
     glib_line(T,X1,T+Dt,X2);
     print("[Time, (approx sol)-(true sol)] : ",0);
     print([T,X1-deval(exp(-2*deval(T)))]);
     T=T+Dt;
     X1=X2; X2=X3;
     if (X3 > 100 || X3 < -100) {
        print("The solution is blown up.");
        break;
     }
  }
}

print("Type in un()")$
end$
```

Figure 4.1: Program `unstable.rr`

Chapter 5

Solving algebraic equations in one variable by a homotopy method

We will consider the problem of solving numerically the algebraic equation

$$x^n + a_{n-1}x^{n-1} + \cdots a_1 x + a_0 = 0$$

Newton's method needs an initial value for the iteration, which must be close to a root. Therefore, Newton's method is not enough to find all the solutions of a given algebraic equation. In this chapter, we will present the homotopy continuation method, which can be used not only for algebraic equations in one variable, but also for systems of algebraic equations in several variables.

We put

$$F(t, x) = x^n + ta_{n-1}x^{n-1} + \cdots ta_1 x + a_0 = 0$$

Then, we have $F(0, x) = x^n + a_0$ and $F(1, x) = 0$ is the equation we want to solve. The root x is a function in t. Let us differentiate $F(t, x(t))$ by t. Then we have

$$x' n x^{n-1} + (a_{n-1}x^{n-1} + ta_{n-1}x'(n-1)x^{n-2}) + \cdots (a_1 x + ta_1 x') = 0$$

This is an ordinary differential equation in t. Note that it is non-linear differential equation as

$$x'(t) = -\frac{a_{n-1}x^{n-1} + \cdots + a_1 x}{nx^{n-1} + ta_{n-1}(n-1)x^{n-2} + \cdots + ta_1} \tag{5.1}$$

The initial condition at $t = 0$ is a root of $x^n + a_n = 0$.

```
extern Cc  $
Cc = 8 $

def homot() {
  Ans = [];
  /* The first solution */
  Xs = 2.0;
  Ans = append(Ans,[homot_aux(Xs)]);
  print("--------------------------------");

  /* The second solution */
  Xs = (-1.0000000000000000000+1.732050807568877293 46*@i);
  Ans = append(Ans,[homot_aux(Xs)]);
  print("--------------------------------");

  /* The third solution */
  Xs = (-1.0000000000000000000-1.732050807568877293 46*@i);
  Ans = append(Ans,[homot_aux(Xs)]);
  print("--------------------------------");
  return Ans;
}

def homot_aux(Xs) {
   X0 = Xs;
   H = 0.1; T = 0.0;
   while (1) {
     X1 = X0+H*((X0^2-X0)/(3*X0^2-2*X0*T+T));
     T += H;
     print("T=",0); print(T,0);
     print(", old=",0); print(X1);
     X1=newton_aux(X1,T);
     print("new=",0); print(X1);
     E=T-1.0; if (E < 0.0) E = -E;
     if (E < 0.0001) break;
     X0 = X1;
   }
   return X1;
}

def newton_aux(A,T) {
  Epsilon = 0.0001;
  Q = A;
  P = Q+1.0;
  while (!( (Q-P > -Epsilon) &&
            (Q-P < Epsilon))) {
     P = Q;
     print(Q);
     Q = P-(P^3-P^2*T+P*T-Cc)/(3*P^2-2*P*T+P);
  }
  return(Q);
}
end$
```

In this program, $F(x,t) = x^3 - x^2 * t + x * t - 8$ The differential equation for $x(t)$ is

$$\frac{dx}{dt} = (x^2 - x)/(3x^2 - 2xt + t).$$

The function `homot()` extends solutions of $F(x,0) = 0$ to those of $F(x,1) = 0$. A root of $F(x,0) = x^3 - 8 = 0$ is set to the variable `Xs`. The function `homot_aux(Xs)` extends the solution `Xs` by utilizing Algorithm **??**.

`X1 = X0+H*((X0^2-X0)/(3*X0^2-2*X0*T+T));` is the reccurrence relation obtained by discretizing the differential equation. The function `newton_aux(X1,T)` evaluates the root of $F(x,t) = 0$ by using Newton's method with the initial value `X1`.

Problem 5.1 (1) Add codes to our program to plot roots in the complex plane of $F(x,t) = 0$ for $t \in [0,1]$. Use `glib`.
(2) Solve an algebraic equation in the applied mathematics by using the homotopy method.

Finally, let us discuss about a weak point of the homotopy method. When the denominator of the differential equation (5.1) $nx^{n-1} + ta_{n-1}(n-1)x^{n-2} + \cdots + ta_1$ is close to 0, the numerical error will become big. This case stands for the case that $F(x,t) = 0$ has a multiplicity. In order to avoid this kind of numerical errors, we may deform the path of numerical integration so that the path avoid the point of the double root or we may use series solution around the point. However, such analysis of non-linear equations is not easy. If there are a linear differential equation satisfied by roots, then it is fine. In fact, such linear equation exits and is called $\mathcal{A}$-hypergeometric differential equation. The discussion about the equation will be done in Chapter 9

Chapter 6

GCD of polynomials in one variable and its applications

6.1 Euclidean Algorithm

6.2 Principal Ideal and Solving Systems of Algebraic Equations in One Variable

The function `g_c_d(F,G)` returns the GCD of `F` and `G`. The function `division(F,G)` returns the The function `variety(F,G)` returns the common zero of `F G`.

```
def in(F) {
  D = deg(F,x);
  C = coef(F,D,x);
  return(C*x^D);
}
```

```
def division(F,G) {
   Q = 0; R = F;
   while ((R != 0) && (deg(R,x) >= deg(G,x))) {
      D = red(in(R)/in(G));
      Q = Q+D;
      R = R-D*G;
   }
   return([Q,R]);
}
```

```
def g_c_d(F,G) {
   if (deg(F,x) > deg(G,x)) {
     S = F; T = G;
   }else {
     S = G; T = F;
   }
   while (T != 0) {
     R = division(S,T)[1];
     S = T;
     T = R;
   }
   return(S);
}
```

```
def variety1(F,G) {
   R = g_c_d(F,G);
   if (deg(R,x) == 0) {
     print("No solution.(variety is empty.)");
     return([]);
   }else{
     Ans = pari(roots,R);
     print("The number of solutions is ",0); print(size(Ans)[0]);
     print("The variety consists of  : ",0); print(Ans);
     return(Ans);
   }
}
end$
```

Let us explain about builtin functions used in the program.

1. `deg(F,x)` : Degree of `F` with respect to `x`. For example, `deg(x^2+x*y+1,x)` returns 2.
2. `coef(F,D,x)` : The coefficient of `F` with respect to the variable `x` at the degree `D`. For example, `coef(x^2+x*y+2*x+1,1,x)` returns `y+2` .

As to details, refer to the online help messages and the manual.
Example. The common roots $V(x^4-1, x^6-1)$ of $x^4-1=0$ and $x^6-1=0$.

```
[346] load("gcd.rr");
1
[352] variety1(x+1,x-1);
No solution.(variety is empty.)
[]
[353] variety1(x^4-1,x^6-1);
The number of solutions is 2
The variety consists of  : [ -1.0000000000000000000 1.0000000000000000000 ]
[ -1.0000000000000000000 1.0000000000000000000 ]
[354]
```

As we see in Chapter **??**, the Euclidean algorithm is fundamental not only in mathematics, but also in enginering of cryptography.

Problem 6.1 Write a program to find a solution of the linear indefinite equation

$$p(x)f(x) + q(x)g(x) = d(x)$$

by using the Euclidean algorithm. Here, f, g and d are polynomial in x and p and q are unknown polynomials.

6.3 Efficient GCD Computation

Let us get the GCD of $f = (2x^3+4x^2+3)(3x^3+4x^2+5)^{10}$ and $g = (2x^3+4x^2+3)(4x^3+5x^2+6)^{10}$ by using the function `g_c_d`.

```
[151] F=(2*x^3+4*x^2+3)*(3*x^3+4*x^2+5)^10$
[152] G=(2*x^3+4*x^2+3)*(4*x^3+5*x^2+6)^30$
[153] H=g_c_d(F,G)$
6.511sec + gc : 0.06728sec(6.647sec)
```

You will obtain a polynomial with a big coefficients. (to be continued) @@@

Chapter 7

The Ring of Differential Operators in One Variable and the GCD Algorithm

7.1 Multiplication in the Ring of Differential Operators

The symbol `d` stands for ∂ and `x` for x. The function `d_mult(F,G)` returns `F` times `G` in the ring of the differential operators.

```
ord([d,x]) ;
def d_mult(F,G){
  N=deg(nm(F),d);
  A=0;
  for(K=0;K<=N;K++){
    A=A+(1/fac(K))*diff2(F,d,K)*diff2(G,x,K);
  }
  return A;
}
def diff2(F,G,K){
  for(I=0;I<K;I++){
    F=diff(F,G);
  }
  return F;
}
```

Here, `diff2(F,G,K)` returns `K`th differentiation of `F` with respect to the variable `G`.

7.2 Application of the GCD Algorithm

```
def in(F) {
  Dn = dn(F); F=nm(F);
  D = deg(F,d);
  C = coef(F,D,d);
  return( C*d^D/Dn);
}

def division(F,G) {
   Q = 0; R = F;
   while ((R != 0) && (deg(nm(R),d) >= deg(nm(G),d))) {
      D = red(in(R)/in(G));
      Q = red(Q+D);
      R = red(R-d_mult(D,G));
   }
   return([Q,R]);
}

def g_c_d(F,G) {
   if (deg(nm(F),d) > deg(nm(G),d)) {
     S = F; T = G;
   }else {
     S = G; T = F;
   }
   while (T != 0) {
     R = division(S,T)[1];
     S = T;
     T = R;
   }
   return(S);
}
```

Chapter 8

Division Algorithm and Gröbner Basis

8.1 Initial Term

There are several nice introductory text books on the Buchberger algorithm and Gröbner basis (see, e.g., [?], [?]). However, there are few elementary introduction for implementation of it. In this Chapter, we will discuss about elementary topics about implementations of the Buchberger algorithm, in particular the division algorithms and the ideal membership problem. At the end of this Chapter, we shortly discuss about applications of the Buchberger algorithm to solving systems of algebraic equations.

The functions `in(F)` and `in2(F)` defined in the codes below are functions of taking the initial term with respect to the lexicographic order with $z > y > x$ and $x > y > z$ respectively.

```
def in(F) {
   D = deg(F,z);
   F = coef(F,D,z);
   T = z^D;
   D = deg(F,y);
   F = coef(F,D,y);
   T = T*y^D;
   D = deg(F,x);
   F = coef(F,D,x);
   T = T*x^D;
   return [F*T,F,T];
}
```

```
def in2(F) {
   D = deg(F,x);
   F = coef(F,D,x);
   T = x^D;
   D = deg(F,y);
   F = coef(F,D,y);
   T = T*y^D;
   D = deg(F,z);
   F = coef(F,D,z);
   T = T*z^D;
   return [F*T,F,T];
}
```

The result is returned as a list. The first element is the initial term of `F`, the second element is the coefficient of the initial term, and the third element is the normalized initial term.

8.2 Internal Representation of Polynomials and Efficiency

You might think there are few differences in efficiency However, there are differences in execution time. Try the following codes.

```
def test1() {
  F = (x+y+z)^4;
  for (I=0; I<1000; I++) G=in(F);
  return(G);
}
def test2() {
  F = (x+y+z)^4;
  for (I=0; I<1000; I++) G=in2(F);
  return(G);
}
```

```
[764] cputime(1);
0
0sec(0.000193sec)
[765] test2();
[z^4,1,z^4]
0.06sec + gc : 0.08sec(0.1885sec)
[766] test1();
[x^4,1,x^4]
0.2sec + gc : 0.08sec(0.401sec)
```

Why is `test1` about 3 times slower than `test2`?

In order to understand the reason, we have to understand how polynomials are stored in the memory.

In Asir, polynomials are stored as a list of cells. For example, the polynomials $5x^2$ is stored as follows.

The name of the main variable	x
The degree with respect to the main variable	2
Coefficient	5
The address of the next monomial	None

For example, $5x^2 + x$ is stored as a list of cells as follows.

The name of the main variable	x
The degree with respect to the main variable	2
The coefficient	5
The address of the next monomial	AAAA

AAAA:

The name of the main variable	x
The degree with respect to the main variable	1
The coefficient	1
The address of the next monomial	None

Not yet translated.

In Asir, polynomials in this expression are called recursive polynomials. The type function returns 2 for these polynomials.

The recursive polynomial is not relevant for taking the initial terms. In Asir, there is a different internal expression of polynomials, which are called distributed polynomials. Distributed polynomials are lists of monomials sorted by an given order. For example, the polynomial $5x^3+5xz^2+xz$ is stored as distributed polynomials with the lexicographic order $x > y > z$ as follows.

The multidegree with respect to x, y, z	[3,0,0]
The coefficient	5
The address of the next monomial	DDDD

DDDD :

The multidegree with respect to z, y, x	[1,0,2]
The coefficient	5
The address of the next monomial	EEEE

EEEE :

The multidegree with respect to z, y, x	[1,0,1]
The coefficient	1
The address of the next monomial	None

By virtue of the expression, taking the initial term is taking the top cell of a list. It can be performed in O(1)-time.

8.3 The Division Algorithm

A program for the 3 variables x, y, z.

```
def multi_degree(F) {
  F = in(F);
  T = F[2];
  return([deg(T,x),deg(T,y),
          deg(T,z),F[1]]);
}
```

Take the initial term with respect to the lexicographic order $x > y > z$.
The multidegree and the leading coefficient are returned as a list.

```
def is_reducible(F,G) {
  DF = multi_degree(F);
  DG = multi_degree(G);
  if (DF[0] >= DG[0]
   && DF[1] >= DG[1]
   && DF[2] >= DG[2])
    return red(DF[3]/DG[3])
           *x^(DF[0]-DG[0])
           *y^(DF[1]-DG[1])
           *z^(DF[2]-DG[2]);
  else return 0;
}
```

When `F` is not reducible by `G`, it returns 0. When `F` is reducible `G`, it returns `M` such that `M G` is equal to the initial term of `F`. In other words, it returns the monomial `M` such that `in(F) = in(M) in(G)`.

```
def division(F,G) {
  Q = 0; R = F;
  D = is_reducible(R,G);
  while (type(D) != 0) {
    Q = Q+D;
    R = R-D*G;
    D = is_reducible(R,G);
  }
  return [Q,R];
}
```

While the initial term of `F` is reducible by the initial term of `G`, subtract from `F` a multiple of `G` such that the initial term of `F` is eliminated. The initial term of the output of this function is not reducible (divisible) by the initial term of `G`. The output is called the remainder.

The function `reduction` below computes the remainder of `F` by a set of polynomials `G`. Note that the result is not necessarily unique.

Program

```
def reduction(F,G)
{
  Rem = 0;
  while ( F ) {
    for ( U = 0, L = G; L != [];
          L = cdr(L) ) {
      Red = car(L);
      Mono = is_reducible(F,Red);
      if ( Mono != 0 ) {
        U = F-Mono*Red;
        if ( !U )
          return Rem;
        break;
      }
    }
    if ( U )
      F = U;
    else {
      H = in(F);
      Rem += H[0];
      F -= H[0];
    }
  }
  return Rem;
}
```

`G` is a list of polynomials. All the terms of the output of this function are not divisible by the initial terms of `G`.

Example.

```
[216] reduction(x^2+y^2+z^2,
      [x-y*z,y-z*x,z-x*y]);
2*x^2+y^2
[217] reduction(x^2+y^2+z^2,
      [z-x*y,y-z*x,x-y*z]);
(y^2+1)*x^2+y^2
```

Outputs are not unique depending on the order of elements in `G`.

8.4 Gröbner Basis

```
def spolynomial(F,G)
{
  DF = multi_degree(F);
  DG = multi_degree(G);
  Mx = DF[0]>DG[0]?DF[0]:DG[0];
  My = DF[1]>DG[1]?DF[1]:DG[1];
  Mz = DF[2]>DG[2]?DF[2]:DG[2];
  return x^(Mx-DF[0])*y^(My-DF[1])*z^(Mz-DF[2])*F/DF[3]
        -x^(Mx-DG[0])*y^(My-DG[1])*z^(Mz-DG[2])*G/DG[3];
}
```

The expression `A?B:C` means that if `A` is 0, return `C` else return `B`.

In case of polynomials in 3 variables, the Buchberger algorithm is implemented as follows.

Program

```
def buchberger(F)
{
  N = length(F);
  Pairs = [];
  for ( I = N-1; I >= 0; I-- )
    for ( J = N-1; J > I; J-- )
      Pairs = cons([I,J],Pairs);
  G = F;
  while ( Pairs != [] ) {
    P = car(Pairs);
    Pairs = cdr(Pairs);
    Sp = spolynomial(G[P[0]],G[P[1]]);
    Rem = reduction(Sp,G);
    if ( Rem ) {
      G = append(G,[Rem]);
      for ( I = 0; I < N; I++ )
        Pairs = cons([I,N],Pairs);
      N++;
    }
  }
  return G;
}
```

The procedure reduces an S-polynomial `Sp`. If the result is 0, then try a next S-polynomial. If the result is not 0, then add the result to `G`. The program stops when all S-polynomials reduce to 0.

Example.

```
[218] F=[x-y*z,y-z*x, z-x*y];
[219] G=buchberger(F);
[x-z*y,-z*x+y,-y*x+z,-x^2+y^2,
 -y*x^3+y*x,-x^5+x^3,-x^4+x^2,
 x^3-x,-y*x^2+y]
[220] reduction(x^2+y^2+z^2,G);
3*x^2
[221] reduction(x^2+y^2+z^2,
                reverse(G));
3*x^2
```

Note that the outputs of the reductions are the same.

This is the simplest implementation of the Buchberger algorithm. However, this implementation can compute only small inputs. Several improved algorithms are studied. Points are as follows.

1. Strategies to choose a pair from the list `Pairs`.
2. Removing unnecessary pairs from the list `Pairs`.
3. Avoiding big numbers in coefficients.

As to these topics, see [**?**].

The builtin function `gr` returns the Gröbner basis of a given set of polynomials and a given order. Several improved algorithms are installed.

```
gr(a list of polynomials, a list of variables, order)

     Example:  gr([x^2+y^2-1, x*y-1],[y,x], 2);
```

The new function `nd_gr` is from a few times to more than ten times faster than `gr` for Gröbner basis computation over small finite fields.

```
[1039] F = [-3*x^3+4*y^2+(-2*z-3)*y+3*z^2,(-8*y-4)*x^2+(2*z+3)*y,
           -2*x^2-3*x-2*y^2+2*z*y-z^2]$
[1040] nd_gr(F,[x,y,z],13,2);
ndv_alloc=4896
[z^14+4*z^13+2*z^12+7*z^11+ ....  ]
```

Chapter 9

Roots of an Algebraic Equation and Hypergeometric Differential Equations

9.1 $\mathcal{A}$-Hypergeometric Differential Equations

We consider an algebraic equation

$$x_0 + x_1 z + \cdots + x_n z^n = g(z) = 0$$

with respect to z. $z = f(x_0, \ldots, x_n)$ is a function in $x_0, \ldots, x_n$. We address the question to characterize the function.

Theorem 9.1 *The function f satisfies the following systems of differential equations.*

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_k \partial x_\ell}, \quad i + j = k + \ell \tag{9.1}$$

$$\sum_{k=0}^{n} k x_i \frac{\partial f}{\partial x_i} + f = 0, \tag{9.2}$$

$$\sum_{k=0}^{n} x_i \frac{\partial f}{\partial x_i} = 0 \tag{9.3}$$

The system is $\mathcal{A}$-hypergeometric differential equations associated to the matrix

$$A = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 2 & \cdots & n \end{pmatrix}$$

Proof: We have $f(x_0, tx_1, t^2 x_2, \ldots, t^n x_n) = \frac{1}{t} f(x_0, x_1, \ldots, x_n)$ (since the both sides are solutions of $x_n (tz)^n + \cdots + x_0 = 0$. Differentiating the both sides by t and taking the limit $t \to 1$, we obtain (9.2). (9.3) is obtained in a similar way. (To be continued.)

There is an algorithm to construct series solutions of this differential equation around infinities of a toric variety.

9.2 Linear Ordinary Differential Equations for Roots

Example 9.1 Let us translate the $\mathcal{A}$-hypergeometric system for a root f of $x_1 + x_2 z + x_3 z^2 = 0$ into a Pfaffian system (total differential equation). Let us take the base $(f, x_3 \partial_3 f)$, then M_1, M_2,

M_3 are as follows ($d - \sum M_i dx_i$).

$$M_1 = \begin{pmatrix} -1/x_1 & 1/x_1 \\ -x3/(4x_1x_3 - x_2^2) & 0 \end{pmatrix}$$

$$M_2 = \begin{pmatrix} 1/x_2 & -2/x_2 \\ 2x_1x_3/(4x_1x_2x_3 - x_2^3) & -1/x_2 \end{pmatrix}$$

$$M_3 = \begin{pmatrix} 0 & 1/x_3 \\ -x_1/(4x_1x_3 - x_2^2) & 1/x_3 \end{pmatrix}$$

This differential equation is linear. The differential equation for roots in Chapter **??** (homotopy method) is non-linear. Nonlinear equations are usually more difficult than linear equations in general. For example, it is difficult to predict the blowing-up point for non-linear equation. However, for linear equations, the blowing-up points of solutions are in the zero set of the highest order coefficient.

```
dp_weyl_gr_main([x*dx+y*dy-1,x*dx-3],[x,y,dx,dy],1,1,2);
[y*dy+2,x*dx-3]

dp_weyl_set_weight(newvect(4,[0,0,1,0]));
/* (The last argument 0 implies the following.)
   Computing the Grobner basis with the weight vector [0,0,1,0]
   The tie-breaker is reverse lex order
*/
```

Index