Scientific Computing with Maple Programming

Zhonggang Zeng

January 4, 2001

Contents

Chapter 1

The fundamentals

1.1 Maple as a calculator

Maple can perform many of mathematical calculations with one-line commands. For example:

```
>  3+5*8;
```

$$43$$

```
>  3^5;
```

$$243$$

```
>  sqrt(5);
```

$$\sqrt{5}$$

```
>  sqrt(5.0);
```

$$2.236067978$$

Notice the difference above between sqrt(5) and sqrt(5.0). When input numbers are integers, Maple performs *simbolic computation* without rounding numbers. If a decimal number is involved, Maple carries out the *numerical computation* with 10 digits precision unless otherwise specified.

The special function "evalf" forces a floating point evaluation:

```
>  evalf(  sqrt(5)  );
```

$$2.236067978$$

```
>  (1/2)^4; 0.5^4; evalf( (1/2)^4 );
```

$$\frac{1}{16}$$
$$.0625$$
$$.06250000000$$

Readers can see the differences above.

Maple does not execute the command until getting the semicolon ";" or colon ":". If a colon is used, the result of the command will not appear in the worksheet.

Maple accepts definitions of variables with :=

```
>  x:=2; y:=5;
```

$$x := 2$$
$$y := 5$$

```
>  (x+3*y)/(2*x-7*y);
```

$$\frac{-17}{31}$$

```
>  evalf(%);
```

$$-.5483870968$$

The sign % represents the most resent result. In the example above, it represents $\frac{-17}{31}$. As mentioned before, the function evalf calculates the decimal number result.

Maple accepts Greek letters

```
>  alpha,  beta,  gamma,  Alpha,  Beta,  Gamma;
```

$$\alpha,\quad \beta,\quad \gamma,\quad A,\quad B,\quad \Gamma$$

However, *the simbol "Pi" carries the value of the special number*

$$\pi = 3.1415926535897932384626\cdots$$

while "pi" does not.

```
>  evalf(Pi);
```

$$3.141592654$$

```
>  alpha:=2*Pi/3;
```

$$\alpha := \frac{2}{3}\pi$$

```
>  sin(alpha), cos(alpha), tan(alpha), cot(alpha);
```

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

```
> evalf(%);
```

$$.8660254040,\ -.5000000000,\ -1.732050808,\ -.5773502693$$

```
> ln(alpha), exp(alpha);
```

$$\ln(\frac{2}{3}\pi),\quad e^{(2/3\,\pi)}$$

```
> evalf(ln(alpha)), evalf(exp(alpha));
```

$$.7392647780, 8.120527402$$

Functions are defined as "operators". Once defined, they can be calculated or manipulated.

```
> f:=x->sin(x)/x;
```

$$f := x \to \frac{\sin(x)}{x}$$

```
> f(alpha), f(1.57);
```

$$\frac{\sin(\alpha)}{\alpha},\ .6369424732$$

```
> f(a+b);
```

$$\frac{\sin(a+b)}{a+b}$$

Taking derivative:

```
> D(f);
```

$$x \to \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2}$$

Table 1.1 is a list of frequently used mathematical operations and their corresponding maple symbols and syntax.

If an arithmetic expression consists of several operations and function evaluations, Maple follows common precedence of operations:

1. Exponentiation or function evaluation (highest)

2. Multiplication or division

3. arithemetic negation

4. Addition and/or subtraction (lowest)

For complicated mathematical expressions, we can use parentheses ( ) to override normal precedences.

Operation	symbol	Example	
		syntax	result
Addition	$+$	$3+5$	8
Subtraction	$-$	$3-5$	-2
Multiplication	*	3*5	15
Division	/	3/5	$\frac{3}{5}$
power	ˆ	3ˆ5	3^5
exponential function	exp	exp(3.2)	$e^{3.2}$
square root	$\sqrt{\ }$	sqrt(5)	$\sqrt{5}$
sine	sin	sin(3*Pi/4)	$\sin(\frac{3\,\pi}{4})$
cosine	cos	cos(Pi/6)	$\cos(\frac{\pi}{6})$
tangent	tan	tan(2*Pi/3)	$\tan(\frac{2\,\pi}{3})$
cotangent	cot	cot(x)	$\cot(x)$
natural logorithm	ln	ln(2.3)	.8129091229
factorial	!	5!	120

Table 1.1: Common operations

Warning: Although brackets [ ] and braces { } are legitimate substitutes for parentheses in mathematical writing, they are not allowed in Maple to override precedences. Brackets are used for data type *list* and braces are used for data type *set*.

Examples:

Notice the difference between **a+b/c+d** and **(a+b)/(c+d)**:

```
>  a+b/c+d;
```

$$a + \frac{b}{c} + d$$

```
>  (a+b)/(c+d);
```

$$\frac{a+b}{c+d}$$

Let's see some complicated expressions. Pay attention to the use of parentheses.

```
>  (a/(b+c) - d^2)/(3*e);
```

$$\frac{1}{3}\,\frac{\dfrac{a}{b+c} - d^2}{e}$$

The expression $\left(1 + \dfrac{3\,P}{h^2}\right)\left[1 - \left(\dfrac{a}{l}\right)^{.6}\right]$ should be entered as follows

```
>  (1+(3*P)/(h^2))*(1-(a/l)^0.6);
```

$$\left(1 + 3\,\frac{P}{h^2}\right)\left(1 - \left(\frac{a}{l}\right)^{.6}\right)$$

More examples:

```
>  c*a1*a2*sqrt(g*(h1-h2)/s)/(3+sqrt(a1^2-a2^2));
```

$$\frac{c\,a1\;a2\,\sqrt{\dfrac{g\,(h1 - h2)}{s}}}{3 + \sqrt{a1^2 - a2^2}}$$

```
>  2*Pi*epsilon/arccos(a^2+b^2-d^2/(2*a*b));
```

$$2\,\frac{\pi\,\varepsilon}{\arccos\left(a^2 + b^2 - \dfrac{1}{2}\,\dfrac{d^2}{a\,b}\right)}$$

1.2 Simple programming

A computational task, in general, involve three components:

1. Data,
2. computational method,
3. results.

As a simple example, we have a formula for the volume of a cylinder:

$$\text{volume of a cylinder} = \text{base area} \times \text{height}.$$

If the base of the cylinder is a circular disk with radius r, then

$$\text{base area} = \pi\, r^2.$$

To compute the volume of the cylinder from the radius r and height h, we have

Data:	r and h
method:	step 1: base area $s = \pi\, r^2$
	step 2: volume $v = s\, h$
	(or combination of step 1 and 2: $v = \pi\, r^2\, h$)
result:	volume v

A program is an implementation of the computational method for the task. It accepts data as *input*, performs the method, and transmits results out as *output*, as shown in the diagram:

$$\begin{tikzcd}
\text{Data} \arrow[r, "\text{input}", "\text{r, h}"'] & \boxed{\text{Program}} \arrow[r, "\text{output}", "\text{v}"'] & \text{results}
\end{tikzcd}$$

A program works as a "black box" for end uses, who enter data as input and activate the program, then see the results.

Using the example of volumn computation for cylinders,, we can write a simple program into Maple worksheet as follows. To break a line before the end of the program, use Shift-Enter instead of Enter key. If no apparent errors, Maple echoes with the program after hitting the semi-colon.

```
> cylinder_volume:=proc(radius,height)  # program definition
     local base_area, volume;           # local variables

     base_area:=Pi*radius^2;            # implement method
     volume:=evalf(base_area*height);

     print(`The volume`, volume);       # output result

  end;
```

```
cylinder_volume := proc(radius, height)
local base_area, volume;
    base_area := π × radius^2 ;
    volume := evalf(base_area × height) ;
    print(`The volume`, volume)
end
```

The program is named *cylinder_volume*, which accepts arguments *radius* and *height* as input. Local variables (i.e. variables only used inside the program) *base_area* and *volume* are defined. Then the computation is carried out by computing the values of base_area and volume. Finally, the result will be shown using the command "print".

Suppose the radius is 5.5m and the height is 8.2m. We execute the program under Maple:

```
>  r:=5.5; h:=8.2;
                              r := 5.5
                              h := 8.2
>  cylinder_volume(r,h);
                       The volume, 779.2720578
```

A program is designed to be used again and again. Every time we use it, all we need is to supply the input data according to the rule of the program. In this example, the rule is that radius first and height second.

For radius 3 ft and height 2 ft:

```
>  cylinder_volume(3,2);
                       The volume, 56.54866777
```

When order is reversed, the answer is different.

```
>  cylinder_volume(2,3);
                       The volume, 37.69911185
```

In summary, a maple program generally consists of the following components:

1. Program definition statement in the form of

   program_name:=proc(*argument_list*)

   Users will use *program_name* to activate the program. The list of variables in *argument_list* is the entrance for input data. *Do not end this line with semicolon ";" if there is local variable declaration line to be described below.*

2. Local variable declaration in the form of

   local *variable_list*;

   This declaration is part of program definition statement.

3. Maple statements that carry out the computational method, using the arguments representing input data.

4. Output mechanism. The simplest way is print statement.

5. closing statement, i.e. the line "end;"

6. Comments. Starting with the character #, the remainder of the line is considered comments by the programmer with no effect on the program execution.

Although you can directly type programs into Maple worksheet, It is prefered to separate the process of programming into 4 steps: (1) ***editing***, (2) ***loading***, (3) ***test run and revision***, (4) ***execution***.

(1) ***Editing***:

Use your favourate text editor, such as Notepad, to edit your program as a *text file* with filename, say "file.txt". Make sure to save it on the floppy disk (usually "a:" drive).

Remark: A DOS filename consists of {*filename*}.{*extention*}. There must be no more than 8 characters in the *filename* part and no more than 3 characters in the *extention* part. The extention of the filename is used to indicate the type of the file. The following are some common extentions:

.txt —- text file

.mws —- maple worksheet

.htm —- HTML file (for World Wide Web use)

.exe —- DOS executable file

Although Windows 95/98 accepts longer filenames in more flexible style, it is recommended that the original DOS restriction be followed.

(2) ***Loading:***

At Maple prompt “>”, type

```
>read(‘a:file.txt‘);
```

Maple then starts reading the text file. If Maple finds a mistake while reading, it will stop and give you an error message. Otherwise, your program will be loaded into the Maple worksheet.

Remark: The two single quotes ‘ used in the command above is usually on the upper-left corner of the keyboard below ESC key. A common mistake is using the quote ’ next to the Enter key.

(3) ***Test run and revision***

If Maple give you an error message, or you suspect a wrong result after a test run, you should go back to Notepad to revise your program

(4) ***Execution***

See example below

Example 1: The future value of an annuity: Suppose you make a *periotic deposit* \$R to an account that pays *interest rate i per period*, for *n periods*, then the *future value* \$S of the account after that n periods will be

$$S = \frac{R\left((1+i)^n - 1\right)}{i}.$$

Write a program to calculate the worth of a retirement plan at the end of the term if the plan pays an *annual rate A*, requires *monthly* deposit \$R, for y *years.* Then use the program to calculate the future value if the monthly deposit is \$300, the annual interest rate is 12%, for 30 year.

Solution: First, we see that the computational task involve input items:

R — deposit

A — annual rate (instead of i)

n — the number of years

We type the following program using Notepad and save as *a:future.txt.* Figure 1.1 shows the program text file.

Remarks:

```
# Program that compute the future value of an annuity
# Input: R: monthly deposit (\$)
#        A: annual interest rate (0.12 for 12\%)
#        y:  number of years
#
future:=proc(R,A,y)              # defines the program "future"
                                 #    and arguments R, I, y
   local  i, n, S;               #  define local variables

   i:=A/12;                      # monthly interest rate is
                                 #    1/12 of the annual rate
   n:=y*12;                      # totoal number of months
   S:=R*((1+i)\symbol{94}n-1)/i;    # calculation of future value

   print('The future value', S); # print the result
end;                             # the last line of the program.
```

Figure 1.1: Future value program

(1) Don't put the semi-colon at the end of proc(...). it is finished at the end of local ....

(2) At end of every definition or action inside the program, remember that you need ";".

(3) Especially, type ";" after *end.*

(4) You can, and should, put comments in your program using "#", as above.

Now read into maple.

```
> read('a:future.txt');
```

$$\begin{aligned}&\mathit{future} := \mathbf{proc}(R,\ A,\ y)\\&\mathbf{local}\, i,\ n,\ S;\\&\quad i := 1/12 \times A\,;\ n := 12 \times y\,;\ S := R \times \left((1+i)^n - 1\right)/i\,;\\&\quad \mathrm{print}(\text{`}\mathit{The\ future\ value}\text{`},\ S)\\&\mathbf{end}\end{aligned}$$

Define the input:

```
> Deposit:=300; AnnualRate:=0.12; Years:=30;
```

$$\mathit{Deposit} := 300$$
$$\mathit{AnnualRate} := .12$$
$$\mathit{Years} := 30$$

Now execute the program with the defined input above. Notice that the variables R, A, y in the program are "dummy" variables. When you execute the program, you just substitute them with variables carrying the data.

```
> future(Deposit,AnnualRate,Years);
```

$$\textit{The future value},\ .1048489240\,10^7$$

Or, you may directly input numbers

```
> future(300, .12, 30);
```

$$\textit{The future value},\ .1048489240\,10^7$$

Example 2: A program that compute the two solutions of the general quadratic equation

$$ax^2 + bx + c = 0$$

with the quadratic formula. Use the program to solve

$$3\,x^2 - 5\,x + 2 = 0 \quad \text{and} \quad x^2 - 3\,x + 2 = 0$$

Solution: Type the following program, shown below, using Notepad and save as file *a:quadsolv.txt*

```
quad:=proc(a,b,c)                      # define the program "quad"
                                       #   with arguments a, b, c
   local sol1, sol2;                   # define local variables
                                       # calculate sol1, sol2
   sol1:=(-b+sqrt(b\symbol{94}2-4*a*c))/(2*a);
   sol2:=(-b-sqrt(b\symbol{94}2-4*a*c))/(2*a);

   print( 'The solutions',sol1, sol2); # print results

end;                                   # end of program
```

Now load program

```
> read('a:quadsolv.txt');
```

$$\begin{array}{l}
\textit{quad} := \textbf{proc}(a,\ b,\ c)\\
\textbf{local}\ \textit{sol1},\ \textit{sol2};\\
\quad \textit{sol1} := 1/2 \times (-b + \mathrm{sqrt}(b^2 - 4 \times a \times c))\,\big/ a\,;\\
\quad \textit{sol2} := 1/2 \times (-b - \mathrm{sqrt}(b^2 - 4 \times a \times c))\,\big/ a\,;\\
\quad \mathrm{print}(\text{`}\textit{The solutions}\text{`},\ \textit{sol1},\ \textit{sol2})\\
\textbf{end}
\end{array}$$

To solve $3\,x^2 - 5\,x + 2 = 0$, notice that $a = 3$, $b = -5$, and $c = 2$:

```
> quad(3,-5,2);
```

$$\textit{The solutions}, 1, \frac{2}{3}$$

To solve $x^2 - 3\,x + 2 = 0$

```
>  a:=1; b:=-3; c:=2;
```

$$a := 1$$
$$b := -3$$
$$c := 2$$

```
>  quad(a,b,c);
```

$$\textit{The solutions}, 2, 1$$

Example 3: The principal stresses (whatever they mean) are given by

$$S_{max} = A + B, \qquad S_{min} = A - B$$

where

$$A = E\,1 - \frac{\mathrm{K}(Q_1 + Q_3)}{2\,(1 - \mu)}$$

$$B = \frac{E\,(1 + K)\,\sqrt{(Q_1 - Q_2)^2 + (Q_2 - Q_3)^2}}{2\,(1 + \mu)}$$

Write a program that will accept input for E, μ, K, Q_1, Q_2, Q_3 and print the values of the principal stresses.

Sample Data

E	μ	K	Q_1	Q_2	Q_3
9.3×10^6	0.32	0.05	138	-56	786

Solution: We write the program

```
principal_stress:=proc(E, mu, K, Q1, Q2, Q3)
   local Smax, Smin, A, B;

   A:=E*(1-K)*(Q1+Q3)/(2*(1-mu));

   B:=E*(1+K)*sqrt((Q1-Q2)\symbol{94}2+(Q2-Q3)\symbol{94}2)/(2*(1-mu));
```

```
     print(`S\_max,  S\_min`);
     print(A+B,A-B);

  end;
```

```
> read(`a:prnstres.txt`);
```

$$
\begin{array}{l}
\mathit{principal\_stress} := \mathbf{proc}(E,\ \mu,\ K,\ \mathit{Q1},\ \mathit{Q2},\ \mathit{Q3})\\
\mathbf{local}\ \mathit{Smax},\ \mathit{Smin},\ A,\ B;\\
\quad A := E \times (1 - K) \times (\mathit{Q1} + \mathit{Q3}) \big/ (2 - 2 \times \mu)\,;\\
\quad B := E \times (1 + K) \times \mathrm{sqrt}((\mathit{Q1} - \mathit{Q2})^2 + (\mathit{Q2} - \mathit{Q3})^2) \big/ (2 - 2 \times \mu)\,;\\
\quad \mathrm{print}(`\mathit{S\_max},\ \ \mathit{S\_min}`)\,;\\
\quad \mathrm{print}(A + B,\ A - B)\\
\mathbf{end}
\end{array}
$$

```
> principal_stress(9.3*10^6, 0.32, 0.05, 138.0, -56.0, 786.0);
```

$$\mathit{S\_max},\ \ \mathit{S\_min}$$

$$.1220668212\,10^{11},\ -.201476241\,10^{9}$$

1.3 Conditional statements

Example 1. A piecewise function

$$
\mathrm{f}(x) = \begin{cases} -(x-1)^2 + 2 & x < 0 \\ 2\,x + 1 & 0 \le x \le 2 \\ 5\,e^{(-(x-2)^2)} & x > 2 \end{cases}
$$

can be calculated with a maple program (edited using Notepad with filename a:func.txt)

```
  func:=proc(x)             # definition
                            # no local variables needed
     if x< 0 then
        -(x-1)^2+2;         # case for x < 0
     elif x<2 then
        2*x+1;              # case for 0 <= x <= 2
     else
        5*exp(-(x-2)^2);    # case for x > 2
     fi;

  end;
```

```
> read(`a:func.txt`);
```

```
func := proc(x)
    if x < 0 then − (x − 1)^2 + 2 elif x ≤ 2 then 2 × x + 1 else 5 × exp(−(x − 2)^2) fi
end
> func(-3); func(1); func(3);
                                        −14
                                         3
                                       5 e^(−1)
```

Program note: When executing a Maple program on Maple worksheet, the result of the very last executed statement is automatically shown without the need to use “print”. For example, if input x is negative, then $-(x-1)^2+2$ is the last statement executed and its result is “printed” automatically.

An if-block has the following structures:

1. Statements executed only if the condition is met. Otherwise, the statements will be skipped:

```
if   condition   then
   statements
fi;
```

2. There are two cases separated by one condition: the condition is met or else:

```
if   condition   then
   statement block 1
else
   statement block 2
fi;
```

   Entering this if-block, the computer verifies the *condition*, if the answer is “true”, the *statement block 1* will be executed and the *statement block 2* will be skipped. On the other hand, if the answer to the *condition* is “false”, the computer will skip the *statement block 1* and execute *statement block 2*

3. There are many cases with many conditions:

```
if   condition 1   then
   statement block 1
elif   condition 2   then
   statement block 2
   ...
   ...
elif   condition n   then
   statement block n
```

```
else
   final statement block
fi;
```

Entering this if-block, the computer verifies the *condition 1* first. If the answer is "false", the machine verifies the *condition 2*, and continue until *condition k* returns "true". In that case the *statement block k* will be executed and all other statements will be skipped. If the answers to all the *conditions* are "false", the computer execute *the final statement block* and skip all others. ("else" and *final statement block* are optional).

Remarks:

- Do not put semicolon at end of "then" or "else".
- It is a good practice to align those "if", "elif", "else" and "fi" of the same if block, and have a 3-space indentation for each statement block.
- All the conditions must be verifiable by Maple. For example, Maple cannot verify if $\sqrt{5} > 1$ and will return a "boolean error", which is common in Maple programming

Example 2. Suppose in a certain course, after five 100-point exams, the course grade will be determined by the scale: A for 90% or above, B for 80-89.9%, C for 70-79.9%, D for 60-69.9% and F for 0-59.9%. Write a program that, for input of total points, print (1) the course grade, (2) a borderline warning if the points is within 1% of next higher grade.

Solution: the program (filename: a:grade.txt)

```
grade:=proc(points)

   if points >= 450 then
      print(‘ A ‘);
   elif points >=400 then
      print(‘ B ‘);
      if points 445 then
         print(‘borderline A‘);
      fi;
   elif points >= 350 then
      print(‘ C ‘);\\
      if points > 395 then\\
         print(‘borderline B‘);\\
      fi;
   elif points >= 300 then\\
      print(‘ D ‘);\\
      if points > 345 then\\
         print(‘borderline C‘);\\
      fi;
  else
```

```
          print(' F ');\\
          if points > 295 then\\
             print('borderline D');\\
          fi;
      fi;

   end;
```

```
> read('a:grade.txt');
```

$grade := \textbf{proc}(points)$
  $\textbf{if}\, 450 \le points\ \ \textbf{then}\, \mathrm{print}(`\ A\ `)$
  $\textbf{elif}\, 400 \le points\ \ \textbf{then}\, \mathrm{print}(`\ B\ `)\,;\, \textbf{if}\, 445 < points\, \textbf{then}\, \mathrm{print}(`borderline\ \ A`)\, \textbf{fi}$
  $\textbf{elif}\, 350 \le points\ \ \textbf{then}\, \mathrm{print}(`\ C\ `)\,;\, \textbf{if}\, 395 < points\, \textbf{then}\, \mathrm{print}(`borderline\ \ B`)\, \textbf{fi}$
  $\textbf{elif}\, 300 \le points\ \ \textbf{then}\, \mathrm{print}(`\ D\ `)\,;\, \textbf{if}\, 345 < points\, \textbf{then}\, \mathrm{print}(`borderline\ \ C`)\, \textbf{fi}$
  $\textbf{else}\, \mathrm{print}(`\ F\ `)\,;\, \textbf{if}\, 295 < points\, \textbf{then}\, \mathrm{print}(`borderline\ \ D`)\, \textbf{fi}$
  fi
end

```
> grade(367);
```

$$C$$

```
> grade(397);
```

$$C$$

$$borderline\ \ B$$

```
> grade(310);
```

$$D$$

Example 3. (Techniques in this example will be used in probability simulations). Suppose we number a standard deck from 1 to 52, such that

hearts: 1-13; spades: 14-26, diamonds: 27-39, clubs: 40-52

in each suit, cards are numbered in the following order:

ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen, king

Write a program that for input of card name (e.g. 3, spade, or jack, diamond), print the number associated with the card.

Remark on data type: Maple classifies data in many types. (try ?type). We need to specify types *numeric* and *string* in this problem. Card value 2, 3, ..., 10 are numeric type. A sequence of characters within a pair of double quote " " is called a string. A string carries no value other than itself. For example, "ace", "jack", "queen", and "king" are string type. In comparison, ace, jack, queen and king without quotes are variable names. The function type(expression, type name) will return either "true" or "false" depending on whether the expression matches the type name. For example:

```
> type(3.5,string);
```

```
                                false
>  type(3.5,numeric);
                                true
>  type(queen, string);
                                false
>  type("queen",string);
                                true
>  Value:=6:      type(Value,numeric);
                                true
>  a:="king":  type(a,string);
                                true
```

Now the program:

```
# program that numbers a card in a deck from 1-52
#   input:
#      value  --- one of the following:
#                  ace, 2, 3, ..., 10, jack, queen, king
#      suit   --- one of the following
#                   "heart", "spade", "diamond", "club"
#
#   output:   the number of the card (1-52)
#
cardnum:=proc(value,suit)

  local order, number, error;

  error:=false;
  if type(value,numeric) then
     order:=value;
  elif type(value,string) then
     if value="ace" then
        order:=1;
     elif value="jack" then
        order:=11;
     elif value="queen" then
        order:=12;
     elif value="king" then
        order:=13;
     else
        print(`Error: wrong value for the first argument`);
        error:=true
     fi
  else
     print(`Error: wrong data type for first argument`);
     error:=true;
  fi;

  if suit="heart" then
     number:=order;
  elif suit="spade" then
```

```
        number:=13+order;
     elif suit="diamond" then
        number:=26+order;
     elif suit="club" then
        number:=39+order;
     else
        print(`Error: wrong value for the second argument`);
        error:=true;
     fi;

     if error=false then
        print(`The card number`, number);
     fi;

   end;
```

```
> read(`a:cardnum.txt`);
```

```
cardnum := proc(value, suit)
local order, number, error;
   error := false ;
   if type(value, numeric) then order := value
   elif type(value, string) then
      if value = "ace" then order := 1
      elif value = "jack" then order := 11
      elif value = "queen" then order := 12
      elif value = "king" then order := 13
      else print(`Error : wrong value for the first argument`) ; error := true
      fi
   else print(`Error : wrong data type for first argument`) ; error := true
   fi;
   if suit = "heart" then number := order
   elif suit = "spade" then number := 13 + order
   elif suit = "diamond" then number := 26 + order
   elif suit = "club" then number := 39 + order
   else print(`Error : wrong value for the second argument`) ; error := true
   fi;
   if error = false then print(`The card number`, number) fi
end
```

```
> cardnum(3,"diamond");
                        The card number, 29

> cardnum("king","club");
                        The card number, 52

> cardnum(queen,club);
                 Error : wrong data type for first argument
                Error : wrong value for the second argument

> cardnum("queen","spade");
                        The card number, 25
```

1.4 Loops with "do" statements

1.4.1 Structure with for–do

Repetitive tasks are carried out by *loops.* For example, if we want to square integer i, $i = 1, 2, 3, \cdots, 10$, the following statements, called a loop, does the job.

```
> for i from 1 to 10 do

> print(i^2)

> od;
```

$$\begin{matrix}1\\4\\9\\16\\25\\36\\49\\64\\81\\100\end{matrix}$$

The simplest loop structure is

for *loop_index* **from** *start_value* **to** *end_value* **do**
block of statements to be repeated
od;

We present, in the list below, the general rules and concepts of for–do loops.

1. The *loop_index* is a variable name such as i, j.

2. *start_value* and *end_value* can be numbers, variable name carrying numbers, or executable expressions resulting in numbers, such as 1, 10, $k1$, $k2$, $2 * n$, etc. The numbers they represent are usually integers, while decimal numbers or even fractions are allowed.

3. The loop works in the following way:

   (a) Assign the *start_value* to *loop_index* .

   (b) Comparing the *loop_index* value with *end_value* , if *loop_index* > *end_value* , jump out of the loop to the line following "od;". Otherwise go to the *block of statements to be repeated.*

   (c) After confirming that *loop_index* $<$ *end_value* , execute *block of statements to be repeated* .

   (d) Add 1 to *loop_index* and go back to (b).

4. Do not alter the value of *loop_index* . Let the loop handle it instead.

5. If the *start_value*=1, then "**from** *start_value* " part may be omitted. That is, the loop in the above example can be equivalently written as

```
for i to 10 do
  i^2;
od;
```

6. Do not put semicolon ";" at the end of "do". Semicolon or colon should be at the end of "od".

7. The *loop_index* increase by the default stepsize 1 Every time the execution of *block of statements to be repeated* is finished. Different stepsize can be used by adding "by *stepsize* " feature. For example,

```
for i from 20 to 2 by -2 do
  i^2;
od;
```

   produces 20^2, 18^2, 16^2, $\cdots$, 4^2, 2^2.

As an example, the following do loop calculates sine and cosine function from 0 to π by $\frac{\pi}{5}$.

```
> for t from 0 by evalf(Pi/5) to evalf(Pi) do
      s1:=evalf(sin(t));
      s2:=evalf(cos(t));
  od;
```

$$s1 := 0$$
$$s2 := 1.$$
$$s1 := .5877852524$$
$$s2 := .8090169943$$
$$s1 := .9510565165$$
$$s2 := .3090169938$$
$$s1 := .9510565160$$
$$s2 := -.3090169952$$
$$s1 := .5877852514$$
$$s2 := -.8090169950$$

1.4.2 An introduction of simple arrays

An array is used to store a sequence of data. For example, when we divide an interval [0,1] into 100 subintervals of equal length 0.01 with points (called nods)

$$x_0 = 0,\ x_1 = .01,\ x_2 = .02,\ \cdots,\ x_{99} = .99,\ x_{100} = 1.00$$

we can use an array to carry these nods.

First, open an (empty) array

```
>  nod:=array(0..100);
```

$$nod := \mathrm{array}(0..100, [])$$

Then we generate the nods with a loop.

```
> for i from 0 to 100 do
     nod[i]:=i*0.01
  od:
```

The values of x_i for each i is stored in nod[i] for i from 0 to 100. We can show the values using the sequence command seq:

```
>  seq(nod[i],i=0..100);
```

```
0, .01, .02, .03, .04, .05, .06, .07, .08, .09, .10, .11, .12, .13, .14, .15, .16, .17, .18, .19, .20,
.21, .22, .23, .24, .25, .26, .27, .28, .29, .30, .31, .32, .33, .34, .35, .36, .37, .38, .39,
.40, .41, .42, .43, .44, .45, .46, .47, .48, .49, .50, .51, .52, .53, .54, .55, .56, .57, .58,
.59, .60, .61, .62, .63, .64, .65, .66, .67, .68, .69, .70, .71, .72, .73, .74, .75, .76, .77,
.78, .79, .80, .81, .82, .83, .84, .85, .86, .87, .88, .89, .90, .91, .92, .93, .94, .95, .96,
.97, .98, .99, 1.00
```

Each member of the array is referenced by array_name[index]. In this example,

```
>  nod[19];
```

$$.19$$

```
>  nod[87];
```

$$.87$$

1.4.3 Iteration

Example 1: Write a program to generate the first n terms of the the sequence

$$x_k = \frac{x_{k-1}}{2} + \frac{1}{x_{k-1}},\quad x_0 = 1,\quad k = 1, 2, \cdots, n$$

The sequence approaches $\sqrt{2}$.

Solution: The program requires an input item n. We edit the program as file `a:sqrt2.txt`

```
#
# Program that generates a sequence
# converging to square root of 2
#
# input   n --- ending index of the sequence
#
sqrt2:=proc(n)
   local x, k;

   x:=array(0..n); # define the array
   x[0]:=1;        # initialize the 0-th entry of the array

                   # loop genreating the remaining entries
   for k from 1 to n do
      x[k]:=evalf( x[k-1]/2 + 1/x[k-1] );
   od;

   print( seq( x[k], k=0..n ) )  # output

end;
```

```
>  read(`a:sqrt2.txt`);
```

$$
\begin{array}{l}
\mathrm{sqrt2} := \mathbf{proc}(n) \\
\mathbf{local}\, x,\, k; \\
\quad x := \mathrm{array}(0..n)\,; \\
\quad x_0 := 1\,; \\
\quad \mathbf{for}\; k\; \mathbf{to}\; n\; \mathbf{do}\; x_k := \mathrm{evalf}(1/2 \times x_{k-1} + 1\big/x_{k-1})\; \mathbf{od}\,; \\
\quad \mathrm{print}(\mathrm{seq}(x_k,\, k = 0..n)) \\
\mathbf{end}
\end{array}
$$

```
>  sqrt2(10);
```

```
1, 1.500000000, 1.416666667, 1.414215686, 1.414213562, 1.414213562, 1.414213562,
1.414213562, 1.414213562, 1.414213562, 1.414213562
```

We can see that the sequence converges to $\sqrt{2}$ very fast.

Example 3. Write a program that generate first n terms of the Fibonacci sequence

$$F_0 = 0, \quad F_1 = 1, \quad F_k = F_{k-1} + F_{k-2}, \qquad \text{for} \quad k = 2, 3, \cdots, n$$

Solution:

```
# program that generate first n terms of the Fibonacci sequence.
# input: n (n>=2 ).
#
Fibonacc:=proc(n)
   local k, F;

   F:=array(0..n);           # define the array
   F[0]:=0;  F[1]:=1;        # the first two terms

   for k from 2 to n do      # iterate
      F[k]:=F[k-1]+F[k-2]
   od;

   seq( F[k], k=0..n )       # output
end;
```

The test run

```
> read(`a:fibon.txt`);
```

$$\begin{aligned}&\mathit{Fibonacc} := \mathbf{proc}(n)\\&\mathbf{local}\, k,\, F;\\&\quad F := \mathrm{array}(0..n)\,;\\&\quad F_0 := 0\,;\\&\quad F_1 := 1\,;\\&\quad \mathbf{for}\, k\, \mathbf{from}\, 2\, \mathbf{to}\, n\, \mathbf{do}\, F_k := F_{k-1} + F_{k-2}\ \mathbf{od}\,;\\&\quad \mathrm{seq}(F_k,\, k = 0..n)\\&\mathbf{end}\end{aligned}$$

```
> Fibonacc(30);
```

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765,
10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040
```

1.4.4 Summation

Example 1: Write a program that adds the squares of the first n odd, positive integers. That is, the sum

$$1^2,\, 2^2,\, 3^2, \cdots, n^2$$

Solution 1: Analysis: The first odd number is 1. If k is an odd number, $k+2$ is the next one

```
# program that compute the sum of the 1st n
#     odd positive integers
# input: n
# output the sum.
#
oddsum1:=proc(n)
   local i, oddnum, s;
   s:=0;                   # initialize the sum as zero
```

```
    oddnum:=1;              # initialize the first odd number
    for i from 1 to n do
       s:=s+oddnum^2;       # accumulating the sum by
                            #   adding the new term
       oddnum:=oddnum+2;    # prepare the next odd number
    od;
    print(`The sum is`, s);
 end;
```

Solution 2: Analysis: odd numbers are

$$2k-1, \quad k=1,2,\cdots$$

```
 # program that compute the sum of the first n
 #     odd positive integers
 # input: n
 # output the sum.
 #
 oddsum2:=proc(n)
    local k, s;
    s:=0;                          # initialize the sum as zero
    for k from 1 to n do
       s:=s+(2*k-1)^2;             # accumulate the sum by
                                   # adding the new term
    od;
    print(`The sum is`, s);
 end;
```

The programming structure of adding n terms is as follows.

```
s:=0;
for$i$ from 1 to $n$ do
   {*prepare the new term*}
   s:=s+{*the new term*}
od;
```

The key is s:=s+{*the new term*}. The variable s (not necessarily s literally) here is used to accumulate the sum. Every time the loop come back to this statement, the value of s adds a new term and the results is assigned to s, replacing its old value. The loop goes from $i = 1$ to $i = n$ and s accumulates terms one by one, eventually accumulates all terms.

Example 2. The sine function can be calculated by

$$\sin(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \cdots$$

Write a program, for input values of x and n, to approximate the sine function at x by calculating the sum of the first n terms.

Analysis: Every term is in the form of $\frac{x^k}{k!}$ for odd k , and with alternating +/- sign.

```
# program to approximate sine function
#    by adding the first n terms of Taylor expansion
# input: x,  n,
# output: the approximate value of sin(x)
#
sine:=proc(x,n)
   local i, s, k, sgn;
   s:=0;                             # empty accumulator
   k:=1;                             # first odd number
   sgn:=1;                           # first sign is +
   for i from 1 to n do
      s:=s + sgn*x^k/k!;             # accumulating
      sgn:=-sgn;                     # alternate sign
      k:=k+2;                        # next odd number
   od;
   print(`The sine function`, s);    # output
end;
```

Now test run:

```
> read(`a:sine.txt`);
```

$sine := \textbf{proc}(x,\, n)$

$\textbf{local}\, i,\, s,\, k,\, sgn;$

$s := 0\,;$

$k := 1\,;$

$sgn := 1\,;$

$\textbf{for}\, i\, \textbf{to}\, n\, \textbf{do}\, s := s + sgn \times x^k \big/ k!\,;\ sgn := -sgn\,;\ k := k + 2\, \textbf{od}\,;$

$\text{print}(`\textit{The sine function}`,\, s)$

$\textbf{end}$

```
> x:=evalf(Pi/6);
```

$$x := .5235987758$$

```
> sine(x,3);
```

The sine function, .5000021328

```
> sine(x,4);
```

The sine function, .4999999921

```
> sine(x,5);
```

The sine function, .5000000003

We can compare with the maple built-in function

```
> sin(x);
```

```
.5000000002
```

The program is as accurate as the build in function sin for $n = 5$.

1.5 Exercises

1. **A mortgage problem**

   Let \$ A be the amount of a mortgage, n the total number of payments, i the interest rate per period of payment. Then the payment \$ R per period is given by the formula:

   $$R = \frac{A\,i}{1-(1+i)^{(-n)}}$$

   - Write a program that, for input: p, r, y, d as price of the purchase, annual interest rate, number of years, and down payment rate respectively, calculates and prints the monthly payment amount.
   - Use your program to calculate the monthly payment of a \$180,000 house, 20% down, 7.75% annual interest rate for 30 years.
   - Use your program to calculate the monthly payment of a \$15,000 car, 10% down, 9.25% annual interest, rate for 5 years.

   Sample results:

   ```
   > read(`a:mortgage.txt`):
   > price:=180000:  down:=20:  rate:=7.75:  year:=30:
   > mortgage(price,rate,year,down);
                         The monthly payment is
                              1031.633646
   ```

2. **Geometry of a circle**

   Write a program that displays radius, circumference, and area of a circle with a given diameter

   Sample results

   ```
   > read(`a:circle.txt`):
   > diameter:=10:
   > circle(diameter);
                           The radius :, 5.0
                    The circumference :, 31.41592654
                         The area :, 78.53981635
   ```

3. **Parallel resistors**

When three electrical resistors with resistance R_1, R_2, and R_3 respectively are arranged in parallel, their combined resistance R is given by the formula

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}$$

Write a program that calculate R

Sample results

```
> read('a:resistor.txt'):
> R1:=20:  R2:=50:  R3:=100:
> resistor(R1,R2,R3);
                         The combined resistance is
                                12.50000000
```

4. **Maximum stress**

The maximum stress in an eccentrically loaded column shown below is given by the formula

$$\sigma_{max} = \frac{P\left(1 + \frac{ec}{r^2 \cos\left(\frac{L\sqrt{\frac{P}{A\,E}}}{2\,r}\right)}\right)}{A}$$

Write a program that calculates and prints the maximum stress. Use the following sample data to test your program

Sample Data

P	A	e	c	r	L	E
4000	29.55	16.6	5.56	4.68	120	3.0×10^7

5. **Card identification**

(See Example 3) Write a program that for input of a number, identify the corresponding card name.

Sample results

```
> read('a:numcard.txt'):
> numcard(1);
                                ace, heart
> numcard(25);
                               queen, spade
> numcard(30);
                                4, diamond
> numcard(0);
                     Error : the input must be 1 − 52
> numcard(55);
                     Error : the input must be 1 − 52
> numcard(45);
                                  6, club
```

6. **Cost calculation**

   A small company offers a car rental plan: \$20.00 a day plus \$0.10/mile for up to 200 miles per day. No additional cost per day for more than 200 miles. (e.g., 3 days with 500 miles cost \$110.00 while 3 days with 800 miles cost \$120.00.) Write a program, for input (1) number of days, and (2) total mileage, calculates the total rental cost.

7. **Cost calculation**

   A travel agency offers a Las Vegas vacation plan. A single traveler costs \$400. If a traveler brings companions, each companion receives a 10% discount. A group of 10 or more receives 15% discount per person. Write a program that, for input number of travelers, print out the total cost.

8. **Real solutions to a quadratic equation**

   For the general quadratic equation

   $$ax^2 + bx + c = 0, \quad a \neq 0$$

   the discriminant Δ

   $$\Delta = b^2 - 4\,ac$$

   - When $\Delta > 0$, there are two real solutions;
   - when $\Delta = 0$, there is only one real solution $-\frac{b}{2\,a}$;
   - there are no real solutions otherwise (i.e. $\Delta < 0$).

   Write a program that for input of a, b, c, print out the number of real solutions and those real solutions themself, if any.

   Sample results

```
> read('a:quadreal.txt'):
> a:=1: b:=0: c:=1:
> quadreal(a,b,c);
                 There are no real solutions
> a:=4: b:=4: c:=1:
> quadreal(a,b,c);
                 There is only one real solution
                              -.5
> a:=3: b:=5: c:=2:
> quadreal(a,b,c);
                 There are two real solutions
                 -.6666666665, -1.000000000
```

9. **More on quadratic equations**

   The quadratic equation problem can be extended to more general case: $a = 0$, the equation is reduced to a linear equation and there is only one real solution $-\frac{c}{b}$ while b is nonzero. If $b = 0$ too, the equation is not valid.

Write a program that calculates real solutions of a quadratic equations for all the possible cases.

Sample results

```
> read(`a:quadadv.txt`):
> a:=0:  b:=0:  c:=0:
> quadadv(a,b,c);
```

Error : invalid equation

```
> a:=0:  b:=5:  c:=3:
> quadadv(a,b,c);
```

This is a linear equation with solution

$$\frac{-3}{5}$$

```
> a:=5:  b:=3:  c:=4:
> quadadv(a,b,c);
```

There are no real solutions

```
> a:=1:  b:=2:  c:=1:
> quadadv(a,b,c);
```

There is only one real solution

$$-1.0$$

```
> a:=2:  b:=-9:  c:=6:
> quadadv(a,b,c);
```

There are two real solutions

3.686140663, .8138593385

10. **Tax schedule**

The following is 1996 tax schedule. Write a program to calculate taxes, for input *Status* and *TaxableIncome*

```
   Status 1 (single)

  TaxableIncome          Tax
      0 -  23,350                0.15*TaxableIncome
 23,350 -  56,550      3,502.50+ 0.28*(TaxableIncome- 23,350)
 56,550 - 117,950     12,798.50+ 0.31*(TaxableIncome- 56,550)
117,950 - 256,500     31,832.50+ 0.36*(TaxableIncome-117,950)
256,500 - up          81,710.50+0.396*(TaxableIncome-256,500)

   Status 2 (Married filing jointly or Qualifying Widow(er)

      TaxableIncome          Tax
       0 -  39,000               0.15*TaxableIncome
  39,000 -  94,250     5,850.00+ 0.28*(TaxableIncome- 39,000)
  94,250 - 143,600    21,320.00+ 0.31*(TaxableIncome- 94,250)
 143,600 - 256,500    36,618.50+ 0.36*(TaxableIncome-143,600)
 256,500 - up         77,262.50+0.396*(TaxableIncome-256,500)

   Status 3 (Married filing separately)

      TaxableIncome          Tax
      0 -  19,500              0.15*TaxableIncome
 19,500 -  47,125     2,925.00+ 0.28*(TaxableIncome- 19,500)
 47,125 -  71,800    10,660.00+ 0.31*(TaxableIncome- 47,125)
 71,800 - 128,250    18,309.25+ 0.36*(TaxableIncome- 71,800)
128,250 - up         38,631.25+ 0.396*(TaxableIncome-128,250)

  Status 4 (Head of household)

     TaxableIncome          Tax
      0 -  31,250          0.15*TaxableIncome
 31,250 -  80,750    4,687.50 + 0.28*(TaxableIncome- 31,250)
 80,750 - 130,800    18,547.50+ 0.31*(TaxableIncome- 80,750)
130,800 - 256,500    34,063.00+ 0.36*(TaxableIncome-130,800)
256,500 - up         79,315.00+ 0.396*(TaxableIncome-256,500)
```

Sample results

```
> read(`a:taxschdl.txt`)
> status:=2:  TaxableIncome:=56890:
> taxschdl(status,TaxableIncome);
                              10859.20
> status:=4:  TaxableIncome:=35280:
> taxschdl(status,TaxableIncome);
                              5975.90
> status:=1:  TaxableIncome:=2590000:
> taxschdl(status,TaxableIncome);
                        .1005776500 10^7
```

11. **Cubic root of 21**

The sequence

$$x_k = \frac{20\,x_{k-1} + 21\,(\frac{1}{x_{k-1}})^2}{21}, \quad x_1 = 1, \quad k = 2, 3, \cdots$$

converges to the cubic root of 21 slowly. Write a program to print out the first n terms of the sequence and observe how many steps does it take to reach its limit within 5 digits.

12. $\frac{\pi}{4}$

Write a program to calculate

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$$

and verify that it equals to $\frac{\pi}{4}$.

13. **Combination of Lucas and Fibonacci sequences**

The Lucas sequence is defined by

$$L_0 = 2,\; L_1 = 1,\; L_k = L_{k-1} + L_{k-2}, \quad k = 2, 3, \cdots, n$$

Write a program that print out the first n terms of

$$S_k = {L_k}^2 - 5\,{F_k}^2,\ k = 1, 2, \cdots, n$$

where F_k is the k-th term of the Fibonacci sequence. Make a conjecture about its values. Can you prove your conjecture?

14. **The cosine function**

    The cosine function can be calculated by

    $$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \cdots$$

    Write a program to approximate the sine function by calculating the sum of the first n terms. How many terms is it necessary to be as accurate as the build in function for $x = \frac{\pi}{3}$

15. **The sum of the Fibonacci sequence**

    Write a program to compute the sum of the first n terms of Fibonacci sequence.

1.6 Further reading material

1.6.1 Documentation of programs

Programs should be well documented for reading, revision and applications. In a Maple program, a "#" sign starts a comment in any line. A programmer should, at least, include comments in the program to tell users and other programmers the following information

1. the purpose of the program
2. the required input
3. the output

It is also a good idea to comment all important steps inside the program.

The use of indentation and empty lines can also improve the readability of programs. Compare the following two programs:

```
oddsum:=proc(n)
local i, oddnum, s;
s:=0;
oddnum:=1;
for i from 1 to n do
s:=s+oddnum^2;
oddnum:=oddnum+2;
od;
print(`The sum is`, s);
end;
```

The one below is functionally the same program, but it is much easier to read with proper indentation and line separation.

```
#
# Program that calculate the squares
# of 1st n odd integers
#    input: n --- integer
#
oddsum:=proc(n)
   local i, oddnum, s;
```

```
      s:=0;                           # prepare initial values
      oddnum:=1;                      # for the summation

      for i from 1 to n do            # the loop of summation
         s:=s+oddnum^2;
         oddnum:=oddnum+2;
      od;

      print(`The sum is`, s);         # output the sum

   end;
```

The blank lines separate programs into three blocks based on their functionalities. By indenting the statements inside the loop, the repeating statements are obvious.

A programmer, when working alone, can use his/her own programming style. In reality, programers are most likely working in groups. Therefore it is important to have a common source code writing style. We suggest the following guidelines in Maple programming:

1. A blank line between logical blocks of statements.
2. Align the opening comments, the program definition line with the "end" statement, and have a three-space indentation for every line between these two lines.
3. If an if–block is used, align "if", "elif", "else", "fi" statements, and have an additional three-space indentation for every line between them.
4. If a loop is used, align "for" and "od" statements, and have an additional three-space indentation for every line between them.

Programming work will be a lot easier if those guidelines are followed. It is also important that you follow these guidelines WHILE writing your program, instead of after the program is done.

1.6.2 Formated printing

Maple's formated printing is quite similar to that in C programming language. The command syntax is that

printf(*'format'*, *expression_sequence*);

where *format* specifies how Maple is to write the *expression_sequence*. For example "%d" is to tell Maple to write the expression as a signed decimal integer:

```
> k:=-23456; j:=65432;
```

$$k := -23456$$
$$j := 65432$$

```
> printf(`The integer %d is assigned to variable k`,k);

The integer -23456 is assigned to variable k

> printf(`The integer %d plus integer %d is %d`,k,j,k+j);

The integer -23456 plus integer 65432 is 41976
```

The *format* part of printf must be inside the quotes ` `. The "%" begins the format specification. The simplist way of specifying the printing format is

$$\%code$$

where code can be one of the following (copied from Maple help file)

```
d       ---- The object is formatted as a signed decimal integer.
o       ---- The object is formatted as an unsigned octal (base 8)
             integer.


x or X  ---- The object is formatted as an unsigned hexadecimal (base 16)
             integer. The digits corresponding to the decimal numbers 10
             through 15 are represented by the letters A through F if X
             is used, or a through f if x is used.

e or E  ---- The object is formatted as a floating point number in
             scientific notation. One digit will appear before the
             decimal point, and the number of digits specified by the
             precision will appear after the decimal point (6 digits
             if no precision is specified).  This is followed by the
             letter e or E, and a signed integer specifying a power of
             10. The power of 10 will have a sign and at least 3 digits,
             with leading zeroes added if necessary.

f       ---- The object is formatted as a fixed point number. The number
             of digits specified by the precision will appear after the
             decimal point.

g or G  ---- The object is formatted using e (or E if G was specified),
             or f format, depending on its value. If the formatted value
             would contain no decimal point, d format is used. If the
             value is less than 10^(-4) or greater than 10^precision,
             e (or E) format is used.  Otherwise, f format is used.

c       ---- The object, which must be a Maple string containing exactly
             one character, is output as a single character.

s       ---- The object, which must be a Maple string, is output as a
             string of at least width characters (if specified) and at
             most precision characters (if specified).

a       ---- The object, which can be any Maple object, is output in
             correct Maple syntax. At least width characters are output
             (if specified), and at most precision characters are output
             (if specified).  NOTE:  truncating a Maple expression by
             specifying a precision can result in an incomplete or
             incorrect Maple expression in the output.

m       ---- The object, which can be any Maple object, is output in
             Maple's ".m" file format. At least width characters are
             output (if specified), and at most precision characters are
             output (if specified).  NOTE:  truncating a Maple ".m"
             format expression by specifying a precision can result in
             an incomplete or incorrect Maple expression in the output.

M or N  ---- The object, which can be any Maple object, is output in
             OpenMath syntax. At least width characters are output
             (if specified), and at most precision characters are output
             (if specified).  NOTE: truncating an OpenMath expression by
             specifying a precision can result in an incomplete or
             incorrect Maple expression in the output.  ALSO NOTE: The
             outut uses OpenMath syntax, but assumes Maple semantics;
             only Maple will be able to process the result. To produce
             an OpenMath compliant result, additional preprocessing is
             required.  If the "\%N" format was specified, no
             back-references will appear in the output.

\%      ---- A percent symbol is output verbatim.
```

Examples:

```
> s:=4.0/3;
```

$$s := 1.333333333$$

```
> printf(`Real number %e is printed in scientific notation`,s);

Real number 1.333333e+00 is printed in scientific notation
```

```
>  printf(`To be more specific, %15.4e is printed \nusing a total of

>  15 characters (including spaces) \nwith 4 digits after decimal

>  point`,s);

To be more specific,      1.3333e+00 is printed

using a total of 15 characters (including spaces)

with 4 digits after decimal point
>  printf(`%8.3f can be printed as a floating point number too.\nIt

>  occupies 8 characters with 3 digits after decimal point.`,s);

   1.333 can be printed as a floating point number too.

It occupies 8 characters with 3 digits after decimal point.
```

In examples above, "\n" is the specification for changing line.

Chapter 2

Iterations

2.1 Methods of golden section and bisection

The method of golden section can be applied to solve the optimization problem of unimodel functions. It is similar to the bisection method for finding zeros of a function. We'll use the method of golden section as an example to show the programming techniques and leave bisection method as a project for students.

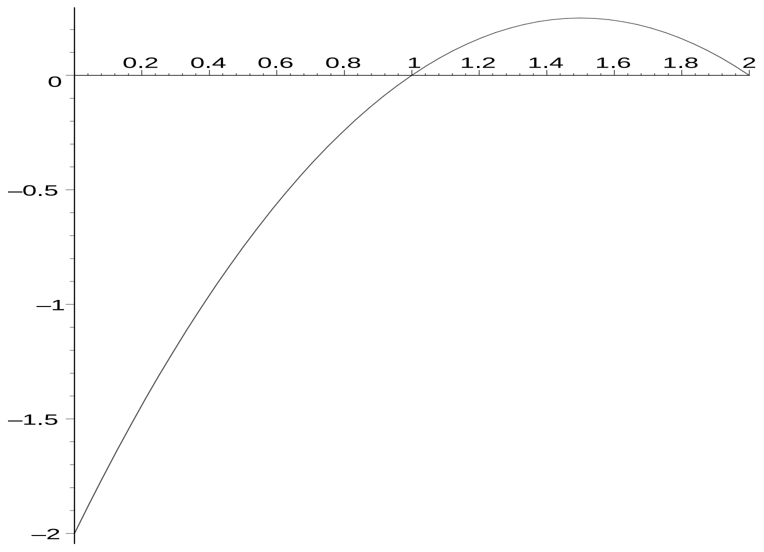
2.1.1 Unimodel functions

A function is called unimodel if it is strictly increasing, reaches its maximum, and then strictly decreasing. A typical unimodel function, with graph generated with Maple commands:

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

$$f := x \to -x^2 + 3\,x - 2$$

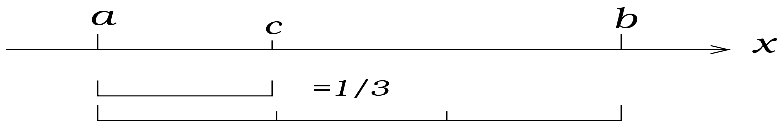
```
> plot(f,0..2);
```



The maximum value of the above function occurs at $x = 1.5$.

2.1.2 Golden section

Let $[a, b]$ be an interval with a point c inside. The location of c in $[a, b]$ can be described as "how much is c to the right of a". For example, we may say c is one third to the right of a, meaning the length of $[a, c]$ is $\frac{1}{3}$ of that of $[a, b]$. This "one third" is the *section ratio* of c in $[a, b]$.



In this case, the value of c equals that of a plus the length of $[a, c]$, which is $\frac{1}{3}$ of the length of $[a, b]$. That is,

$$\begin{aligned} c &= a + (c - a) \\ &= a + \frac{1}{3}(b - a) \\ &= (1 - \frac{1}{3})a + \frac{1}{3}b \end{aligned}$$

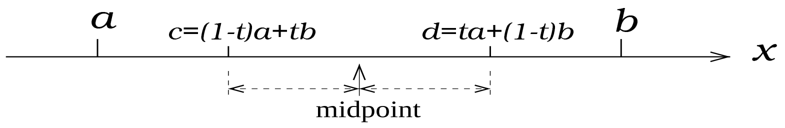
For the same reason, if we know that a point $d \in [a, b]$ with a section ratio equals to t where $0 \leq t \leq 1$, then

$$d = (1-t)\,a + t\,b$$

and the section ratio

$$t = \frac{c-a}{b-a} = \frac{\text{length of } [a,c]}{\text{length of } [a,b]}$$

Let c be a point in the interval $[a,b]$ with section ratio t. I.e. $c = (1-t)a + tb$. If we reverse the role of t and $1-t$, we have another point $d = ta + (1-t)b$, which is symmetric to c about the midpoint of $[a,b]$. These pair of points are said to be conjuagate to each other in $[a,b]$

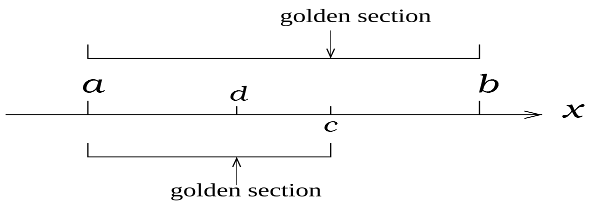


There is a special section ratio τ, called *golden section ratio*. It is special because if c cuts the interval $[a,b]$ with this section ratio τ,

$$c = (1-\tau)a + \tau b$$

then its conjuagate point d cuts the interval $[a,c]$ with the same golden section ratio

$$d = \tau a + (1-\tau)b = (1-\tau)a + \tau c$$



It can be verified that the golden section ratio is

$$\tau = \frac{-1+\sqrt{5}}{2}$$

which is approximately 0.618.

2.1.3 The method of golden section

Let f(x) be a unimodel function on $[a,b]$. We know there is a maximum point x_* of f(x) inside the interval $[a,b]$. Our objective is to shrink the interval so that it still contains x_*, thereby we can zeroin on x_*.

As shown in Figure 2.1, let m_l and m_r be a pair of conjuage section point corresponding to the golden section ratio, we may call them the mid-left and mid-right golden section points of $[a, b]$. We shall shrink $[a, b]$ to either left subinterval $[a, m_r]$ or the right subinterval $[m_l, b]$. We are looking for the unique maximum point of the unimodel function f(x). So we compare the values of $f(m_l)$ and $f(m_r)$. If $f(m_r) > f(m_l)$ as illustrated in Figure 2.1, then the interval $[m_l, b]$ containing m_r is the interval of choice. Similarly, if $f(m_l) > f(m_r)$, we would shrink $[a, b]$ to $[a, m_r]$.

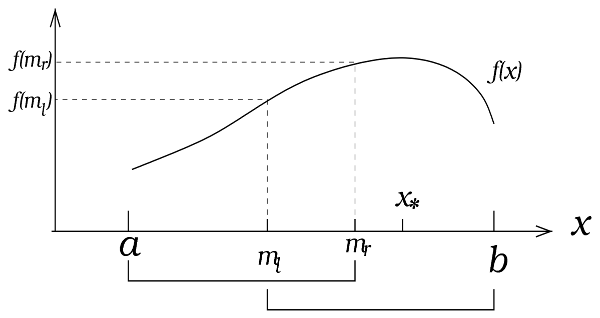


Figure 2.1: The method of golden section: interval $[m_l, b]$ is chosen over $[a, m_r]$ because $f(m_r) > f(m_l)$

We therefore have a simple rule of choosing subintervals:

- *If the "left value" $f(m_l)$ is larger, choose the "left subinterval" $[a, m_r]$.*
- *If the "right value" $f(m_r)$ is bigger, choose the "right subinterval" $[m_l, b]$*

The beauty of the method is that each subinterval is cut by either m_l or m_r with golden section ratio. Therefore, to shrink the subinterval, we need only the conjuagate section point and only one additional function evaluation.

So the method of golden section can be described in the following pseudo-code:

```
Input: $a$, $b$, $f$ and error tolerance $\delta$
Make sure $delta$ is positive
Start with working interval $[\alpha, \beta] = [a, b]$
Set $\tau$ the golden section ratio
Find the pair of golden section points $m_l$ and $m_r$
Calculate $v_l = f(m_l)$ and $v_r = f(m_r)$
```

```
While $|\beta - \alpha| > \delta$, repeat the following as a loop
    If $v_l > v_r$ then
        Replace $[\alpha, \beta]$ with $[\alpha, m_r]$
        Replace $m_r$ with $m_l$
        Replace $v_r$ with $v_l$
        Replace $m_l$ with $(1-\tau)\alpha + \tau\beta$
        Calculate $v_l = f(m_l)$
    else
        Replace $[\alpha, \beta]$ with $[m_l, \beta]$
        Replace $m_l$ with $m_r$
        Replace $v_l$ with $v_r$
        Replace $m_r$ with $\tau\alpha + (1-\tau)\beta$
        Calculate $v_r = f(m_r)$
    end if
```

The actual program based on the pseudo-code:

```
# The program that calculate the maximum point of a unimodel function
# input: f    --- the unimodel function, define in Maple as operator,
#                 e.g. f:=x->-x^2+3*x-2;
#        a, b --- end points of the interval [a,b]
#        tol  --- the error tolerance
#
goldsec:=proc(f,a,b,delta)
   local goldratio, gratio, alpha, beta, ml, mr, fml, fmr, stepcount;

   if delta <= 0.0 then                    # avoid negative delta
      print(`error tolerance must be positive`);
      RETURN();
   fi;

   tau := evalf( 0.5*(-1.0+sqrt(5.0)) );   # golden section ratio
   alpha:=a; beta:=b;                      # working interval
   mr:=(1-tau)*alpha+tau*beta;             # the mid-left point
   ml:=tau*alpha+(1-tau)*beta;             # the mid-right point
   vl:=f(ml);  vr:=f(mr);                  # function values

   while abs(beta-alpha) >= delta do       # the main loop

      if vl > vr then
         beta:=mr;                         # undate working interval
         mr:=ml;  vr:=vl;                  # update mr and vr
         ml:=tau*alpha+(1-tau)*beta;       # the new mid-left
         vl:=f(ml);                        # the new vl
      else
         alpha:=ml;                        # update working interval
         ml:=mr;  vl:=vr;                  # update mr and vr
```

```
            mr:=(1-tau)*alpha+tau*beta;       # the new mid-right
            vr:=f(mr)                         # the new vr
          fi;

        od;

        print(`The solution is`);
        print( 0.5*(alpha+beta) );            # output

      end;
```

In the program, we used command

RETRUN()

which force the machine exit the program.

Set the number of digits to be 20:

```
> Digits:=20;
```

$$Digits := 20$$

```
> read(`a:goldsec.txt`):
> a:=0; b:=2; tol:=0.001;
```

$$a := 0$$
$$b := 2$$
$$tol := .001$$

```
> goldsec(f,a,b,tol);
```

The solution is
1.5001934984462164631

If we reduce the tolerance to 0.0002, we have

```
> goldsec(f,a,b,0.00002);
```

The solution is
1.4999951775621607752

```
> goldsec(f,a,b,0.000000001);;
```

The solution is
1.4999999999534881866

2.1.4 The while–do loop

The implementation of golden section method involve another type of loops:

```
while  condition  do
    block of statements
od;
```

In this loop, the *block of statements* will be repeated as long as the *condition* remains "true" Using the method of golden section as an example, we started with a working interval $[\alpha, \beta]$ and enter the while–do loop. If the interval is small enough in the first place (i.e. $\beta - \alpha \leq \delta$), all the statements inside the loop(i.e. between "do" and "od") will be ignored and "print" statements will follow. Otherwise, the machine will enter the loop and shrink the working interval, reach "od" and come back to "while" to recheck the condition $\beta - \alpha \geq \delta$. Eventually the *condition* will become "false" the machine jumps to the line following "od".

There is a danger of using while–do loops. It may run forever if the condition can never be met. For example, the program goldsec, with accidental input of negative error tolerance, would be a dead loop if there were no statements

```
if delta <= 0.0 then                    # avoid negative delta
   print('error tolerance must be positive');
   RETURN();
fi;
```

2.1.5 The bisection method

From the Intermediate Value Theorem in Calculus, let $f(x)$ be a continuous function on the interval $[a, b]$ such that $f(a)$ and $f(b)$ have different signs, then there exists a number $x_* \in [a, b]$ such that $f(x_*) = 0$.

To find x_*, we can cut the interval with the mid-point $mp = \frac{a+b}{2}$ and calculate $f(mp)$. According to the sign of $f(mp)$,

$$\text{the solution } x_* \text{ is in or equal to } \begin{cases} [a,\ mp] & f(a)\,f(mp) \leq 0 \\ mp & f(mp) = 0 \\ [mp,\ b] & \textit{otherwise} \end{cases}$$

We can thereby replace the interval $[a,\ b]$ with the new subinterval, either $[a,\ mp]$ or $[mp,\ b]$, of half length. This process, which is called ***the bisection method***, can be repeated until the length of the final interval is of length less than the error tolerance. The midpoint of the final interval can be then output as the approximate solution.

The implementation of the bisection method is similar to the method of golden section, and left to readers.

2.2 Newton's and other iterative method

2.2.1 Newton's iteration

For a given function $\mathrm{f}(x)$ which is differentiable, the iteration

$$x_k = x_{k-1} - \frac{\mathrm{f}(x_{k-1})}{\mathrm{f}'(x_{k-1})}, \quad k = 0,\, 1,\, 2,\, \cdots$$

is called Newton's iteration. If the starting point x_0 is near a zero x_* of the function f, the iteration generates a sequence of points x_1, x_2, $\cdots$ that converges to the solution x_*. Usually Newton's iteration converges very fast, if it converges at all.

For example, let $\mathrm{f}(x) = x - \cos(\pi\, x)$, there is a solution 0.3769670099. If we start at $x_0 = .5$

```
> x:=array(0..10);
```

$$x := \mathrm{array}(0..10,\, [])$$

```
> f:=x->x-cos(Pi*x);
```

$$f := x \to x - \cos(\pi\, x)$$

```
> g:=D(f);  #find the derivative of f
```

$$g := x \to 1 + \sin(\pi\, x)\, \pi$$

```
> x[0]:=0.5;
```

$$x_0 := .5$$

```
> x[1]:=evalf(x[0]-f(x[0])/g(x[0]));
```

$$x_1 := .3792734965$$

Two digits are correct after one step.

```
> x[2]:=evalf(x[1]-f(x[1])/g(x[1]));
```

$$x_2 := .3769695051$$

Five digits are correct after two steps.

```
> x[3]:=evalf(x[2]-f(x[2])/g(x[2]));
```

$$x_3 := .3769670094$$

In only three steps we got the solution up to 9 digits.

The above process can be implemented using a loop.

```
> f:=x->x-cos(Pi*x);
```

```
    g:=D(f);
    for k from 1 to 10 do
       delta:=evalf(f(x[k-1])/g(x[k-1])):
       x[k]:= x[k-1] - delta:
       printf( " x[\%2d]= \%15.10f   delta=\%15.10f \\n",
               k, x[k], delta):
    od:
```

```
x[ 1]=     .3792734965   delta=    .1207265035

x[ 2]=     .3769695051   delta=    .0023039914

x[ 3]=     .3769670094   delta=    .0000024957

x[ 4]=     .3769670092   delta=    .0000000002

x[ 5]=     .3769670093   delta=   -.0000000001

x[ 6]=     .3769670094   delta=   -.0000000001

x[ 7]=     .3769670092   delta=    .0000000002

x[ 8]=     .3769670093   delta=   -.0000000001

x[ 9]=     .3769670094   delta=   -.0000000001

x[10]=     .3769670092   delta=    .0000000002
```

There are several important issues about Newton's iteration:

- Newton's iteration converges *locally.* Namely, it could fail, if the starting iterate x_0 is not close enough to the solution x_s.
- Even if Newton's iteration does converge, we generally don't know in advance how many steps are needed to reach the desired accuracy.

Therefore, we must set certain *stop criteria* to terminate Newton's iteration.

First, there must be a limit to the number of steps for Newton's iteration. An input integer n is needed so that Newton's iteration stops at the step x_n. Secondly, the iteration should stop when the accuracy is good enough. The example above shows that the magnitude of

$$\delta_k = \frac{\mathrm{f}(x_{k-1})}{\mathrm{f}'(x_{k-1})}$$

indicates the accuracy of x_k. Actually, for a given tolerance *tol*,

$$|\delta_k| \leq tol$$

a widely used criterion for stoping Newton's iteration.

From the above discussion, we can set up a loop of iteration that stops either at step n or when $|\delta_k| \le tol$, which ever achieves first. This is can be done with the following Maple loop command that combines for–do and while–do:

```
for  loop_index   from ··· to ··· while  condition do
   statements to be repeated
od;
```

Using the previous example

```
> delta:=1.0:
  for k from 1 to 10 while abs(delta)>10.0^(-8) do
     delta:=evalf(f(x[k-1])/g(x[k-1])):
     x[k]:= x[k-1] - delta:
     printf( " x[\%2d]= \%15.10f   delta=\%15.10f \\n",
             k, x[k], delta):
  od:

x[ 1]=     .3792734965   delta=    .1207265035

x[ 2]=     .3769695051   delta=    .0023039914

x[ 3]=     .3769670094   delta=    .0000024957

x[ 4]=     .3769670092   delta=    .0000000002
```

This loop stops at the fourth step because $\delta > 10^{(-8)}$ is no longer true. In worst case, the loop will stop at 10-th step.

A Newton's iteration program can be implemented with the following pseudocode:

```
Input: function $f$, initial iterate $x0$, step limit $n$, tolerance $tol$

calculate derivative of $f$ and assign it to $g$

generate an (empty) array $x$ with index from 0 to $n$

assign the value $x0$

set an initial value delta larger than tol

use "for...from...to...while...do" loop to carry out the iteration
```

check if abs(delta)<tol.

if so, print the last iterate

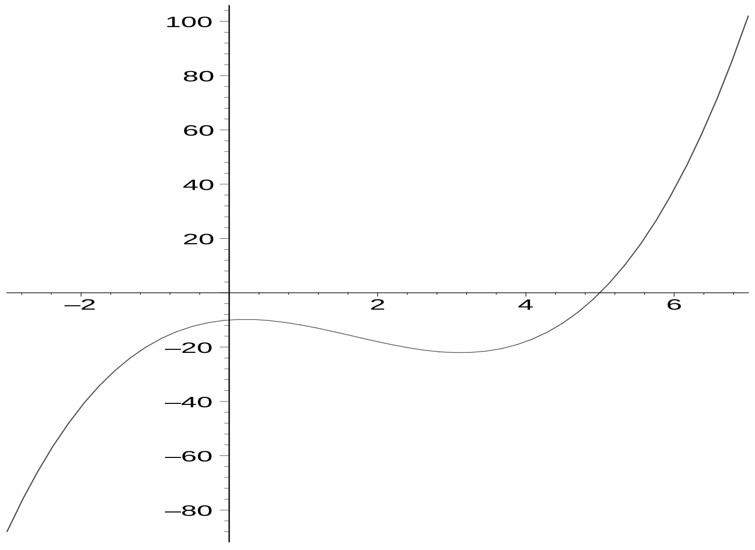
otherwise, print out a failure message.

The actual implementation will be left as an exercise. Here are some examples of Newton's iteration.

```
>  f:=x->x^3-5*x^2+2*x-10;
```

$$f := x \to x^3 - 5\,x^2 + 2\,x - 10$$

```
>  plot(f,-3..7);
```



The graph of f(x) shows that there is a zero around $x = 5$. Now lets apply Newton's iteration

```
>  x0:=4.0:  tol:=10.0^(-8):  n:=10:
>  read(`a:newton.txt`):
>  newton(f,x0,n,tol);

 x[ 1]=    5.8000000000   delta=  -1.8000000000

 x[ 2]=    5.1652715940   delta=    .6347284061

 x[ 3]=    5.0092859510   delta=    .1559856433

 x[ 4]=    5.0000317760   delta=    .0092541752
 x[ 5]=    5.0000000010   delta=    .0000317752

 x[ 6]=    5.0000000010   delta=   0.0000000000

Newton's iteration successful in  6 steps

The solution is         5.0000000010
```

Starting from certain points you'll see that Newton's iteration fails

```
> y0:=-3.0:
> newton(f,y0,n,tol);
```

```
 x[ 1]=   -1.5084745760   delta=  -1.4915254240

 x[ 2]=    -.3447138130   delta=  -1.1637607630

 x[ 3]=    1.6065726800   delta=  -1.9512864930

 x[ 4]=    -.8521934710   delta=   2.4587661510

 x[ 5]=     .4039993150   delta=  -1.2561927860

 x[ 6]=   -6.0088494770   delta=   6.4128487920

 x[ 7]=   -3.5470641770   delta=  -2.4617853000

 x[ 8]=   -1.8900892540   delta=  -1.6569749230

 x[ 9]=    -.6757701190   delta=  -1.2143191350

 x[10]=     .7009957250   delta=  -1.3767658440

 Newton's iteration unsuccessful in 10 steps
```

2.3 Exercises

1. **The ratio of golden section**

   Show that the ratio of golden section is $\frac{-1+\sqrt{5}}{2}$

2. **Programming the bisection method**

   Using the program for the method of golden section as an example, write a program for solving the equation

   $$\mathrm{f}(x) = 0$$

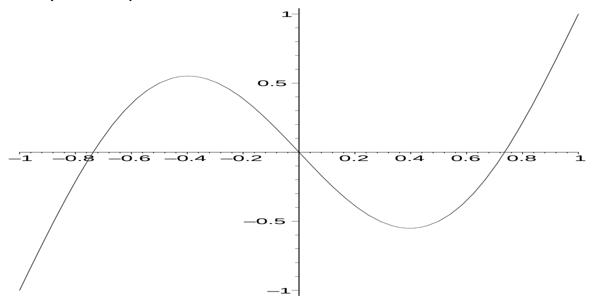
   using the bisection method.

   Sample results We are interested in the zeros of the function $\mathrm{f}(x) = x - \sin(\pi\, x)$ in the interval $[-1, 1]$.

```
> f:=x->x-sin(Pi*x);
```

$$f := x \to x - \sin(\pi\,x)$$

```
> plot(f,-1..1);
```



There are three zeros in intervals $[-1,-0.5]$, $[-0.5,0.5]$, and $[0.5,1]$. Most importantly, the function f(x) have opposite signs at the end points of each interval. Therefore we can apply the bisection method to locate them.

```
> read(`a:bisect.txt`):
> bisect(f,-1,-0.5,0.00000001);
```

The approximate solution is

$$-.7364844497$$

```
> bisect(f,-0.4,0.5,0.00000001);
```

The approximate solution is

$$-.372532403\;10^{-9}$$

```
> bisect(f,0.5,1,0.00000001);
```

The approximate solution is

$$.7364844497$$

Another example: suppose we are interested in the equation

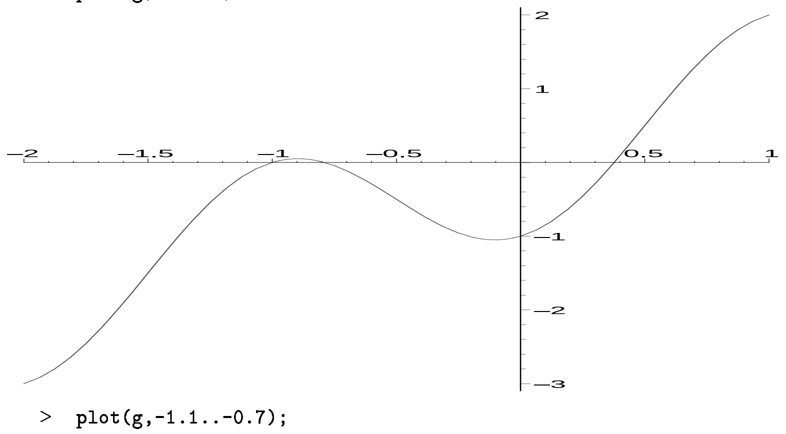
$$x = \cos(\pi\,x)$$

on the interval $[-2,1]$. We investigate the function by plotting the graph:

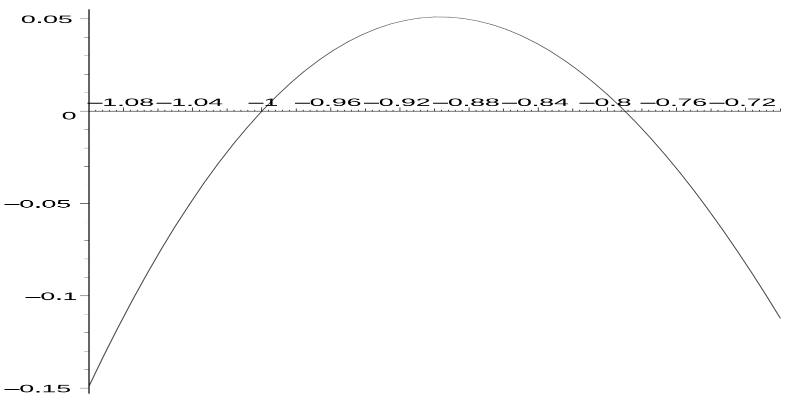
```
> g:=x->x-cos(Pi*x);
```

$$g := x \to x - \cos(\pi\,x)$$

```
> plot(g,-2..1);
```



```
> plot(g,-1.1..-0.7);
```



Therefore there are three solutions.

```
> bisect(g,-1.1,-0.9,0.00000001);
```

The approximate solution is

−1.000000003

```
> bisect(g,-0.9,-0.7,0.00000001);
```

The approximate solution is

−.7898326312

```
> bisect(g,0,1,0.000000001);
```

The approximate solution is

.3769670099

Try h(x) = $\sin(x) - e^{(-x)}$ on the interval $[0, 7]$.

3. **Programming Newton's iteration**

   Write a Newton's iteration program, according to the pseudo-code in Section 2.2.1 and following the examples there.

4. **The Euclidean Algorithm**

   A common divisor of two integers m and n is an integer d that divides both m and n. For example, integer 60 and 45 have common divisors 3, 5, and 15. In fact, 60=(2)(2)(3)(5) and 45=(3)(3)(5). The largest number among the common divisors m and n is called the greatest common divisor of m

   $$gcd(60, 45) = 15$$

   The classical method of finding the greatest common divisor is the Euclidean algorithm:

   Suppose $n < m$.

   (1) Let m be divided by n and let r be the remainder.

   (2) If $r = 0$, then n is the gcd, exit the process.

   (3) Otherwise, replace the pair (m, n) with the pair (n, r), and go back to (1).

For example, we want to find $gcd(13578, 9198)$:

```
> m:=13578; n:=9198;
```

$$m := 13578$$
$$n := 9198$$

```
> r:=m mod n;
```

$$r := 4380$$

```
> m:=n; n:=r; r:=m mod n;
```

$$m := 9198$$
$$n := 4380$$
$$r := 438$$

```
> m:=n; n:=r; r:=m mod n;
```

$$m := 4380$$
$$n := 438$$
$$r := 0$$

Now the gcd is 438 because the remainder $r = 0$.

Write a program that, for any positive integer input m and n, prints out gcd(m,n) using the Euclidean algorithm.

Sample results

```
> read('a:gcdiv.txt'):
> gcdiv(13578,9198);
                    The greatest common divisor is
                                 438
> gcdiv(9198,13578);
                    The greatest common divisor is
                                 438
> gcdiv(60,45);
                    The greatest common divisor is
                                 15
```

5. **Mortgage interest**

   Let the *amount of a mortgage* be $\$A$, with *interest rate per payment period* be i, and the total *number of payments* be n, then the *payment* amount $\$PMT$ *per period* can be calculated with the following formula:

$$PMT = A \frac{i}{1 - (1+i)^{(-n)}}$$

   Suppose you want to buy a \$120,000 house with 20% down. You can afford \$713/month payment for 30 years. What is the interest rate you need to shop for?

6. **Mortage points**

   Banks often charges points for lending a mortgage. One point means 1% of the mortgage amount. Usually you can borrow that additional amount into that mortgage. Let p be the number of points the bank charges. Your actual amount of mortgage increases from $\$A$ to $\$A(1+0.01p)$. Thus the periodic payments increases to

$$PMT = A\,(1+.01\,p)\,\frac{i}{1-(1+i)^{(-n)}}$$

Banks offer different rate and point combinations and make consumers confused. However, every rate/point combination is equivalent to an *actual interest rate* with 0 point. It makes sense (and cents) to compare different rate/point combination via comparing their actual rates. Find the actual annual rate for a 30 year mortgage at annual rate of 7.75% with 3 points.

(Hint: 1. You may assume an amount of mortgate, say $100,000. The final answer should be independent of that amount. 2. The crucial idea: there is a payment calculated by the formula above for the rate/point combination. With a monthly interest rate x with no point, there is also a would-be payment. If that x is equivalent to the rate/point combination, those two payments must be the same.)

7. **More on mortgage points**

   Banks often charges points for lending a mortgage. One point means 1% of the mortgage amount. Usually you can borrow that additional amount into that mortgage.

   Let p be the number of points the bank charges. Your actual amount of mortgage increases from $\$A$ to $\$A(1+0.01p)$. Thus the periodic payments increases to

$$Pmt = \frac{A\,(1+.01\,p)\,i}{1-(1+i)^{(-n)}}$$

   Banks offer different rate and point combinations and make consumers confused. However, every rate/point combination is equivalent to an *actual interest rate* with no point. It makes sense (and cents) to compare different rate/point combination via comparing their actual rates. Write a program that, for input

```
Amount      --- amount of mortgage

annual_rate --- the percentage of annual rate
                (i.e.  annual_rate=7.75 for 7.75%)

points        --- the number of points

years         --- the number of years
```

print out the actual annual rate with zero point. For \$150000 mortgage of 30 years, use your program to calculate the actual annual rate for (i) 7.75%, 3 points, (ii) 8%, 1 point.

Hint:

(a) Let x be any monthly rate with zero point, the corrsponding payment is $\frac{A\,x}{1-(1+x)^{(-n)}}$. To make it equivalent to the payment calculated with points, you have to solve the equation

$$\frac{A\,x}{1-(1+x)^{(-n)}} = \frac{A\,(1+.01\,p)\,i}{1-(1+i)^{(-n)}}$$

(b) Define a function $f(x)$ such that $f(x) = 0$ equivalent to the equation above

(c) Make careful analysis to find the interval $[a, b]$ such that $f(a)$ and $f(b)$ have different signs. It is good idea to know the meaning of $f(x) > 0$ and $f(x) < 0$.

(d) Fact 1: Adding points makes payment higher. So monthly rate i with 0 point has **lower** payment than the rate i with any point.

(e) Fact 2: In practical cases, points don't double the actual rate. That is, monthly rate $2i$ with 0 point has **higher** monthly payment than monthly rate i with points.

8. **The best rational approximation**

Let r be a real number, such as $\sqrt{2}$, π, e. one can use a rational number $\frac{p}{q}$ to approximate it. For example $\sqrt{2}$ can be approximated by $\frac{3}{2}$, $\frac{7}{5}$, $\cdots$. One can get better approximation if larger denomenator is allowed. The objective is: for a given bound N on the denomenator, what is the best rational approximation to r.

There is a simple method to find the best rational approximation $\frac{p}{q}$. Start from $p = 0$, and $q = 1$. The initial error is $\left|\frac{p}{q} - r\right| = |r|$. Repeat the following until either $N < q$ or *error* $= 0$:

- If $\frac{p}{q} < r$, increase p by 1
- Otherwise, increase q by 1, to obtain a new set of p, q
- With this new set of p and q, calculate the current error $\left|\frac{p}{q} - r\right|$.
- If this error is less than the previous error, a better approximation is obtained. Update the the error and print out this set of p and q.

Write a program that carres out the above method for input r and N. For a given N, the above method output a sequence $p - q$ pairs, with each pair produces a fraction p/q that is a better approximation to r than the previous pair, within the denomenator limit N.

Use your program to print out sequences of every better approximation to $\sqrt{2}$, π, and e, with N to tens of thousands.

Chapter 3

Arrays

3.1 Arrays

3.1.1 The use of arrays

An array is used to store a sequence of data. For example, if we want to store the first 11 terms of Fibonacci sequence, F_0, F_1, $\cdots$ F_{10} we first open an (empty) array with index range from 0 to 10.

```
>  fib:=array(0..10);  # define an (empty) array
```

$$fib := \mathrm{array}(0..10, [])$$

Then assign the values to each entry of the array:

```
>  fib[0]:=0; fib[1]:=1;  # initialize the Fibonacci sequence
```

$$fib_0 := 0$$
$$fib_1 := 1$$

```
>  for i from 2 to 10 do           # generate remaining

>  fib[i]:=fib[i-1]+fib[i-2];   # entries

>  od;
```

$$fib_2 := 1$$
$$fib_3 := 2$$
$$fib_4 := 3$$
$$fib_5 := 5$$
$$fib_6 := 8$$

$$fib_7 := 13$$
$$fib_8 := 21$$
$$fib_9 := 34$$
$$fib_{10} := 55$$

Generally, an array is defined by

>{*array name*}:=array({*leading index*}..{*ending index*});

Each entry of the array is referenced as {*arrayname*}[{*index*}]. You may use ?array in Maple to open documentation about arrays.

```
> fib[6];
                                    8
> fib[10];
                                    55
```

3.1.2 The command "seq"

The command "seq" is very useful in generating a sequence with a known pattern:

for example, the partial Fibonacci sequence f_0, f_1, $\cdots$, f_{20} is a sequence with pattern

$$f_i, i = 0,\ 1,\ \cdots,\ 10$$

In earlier calculation, this sequence was generated and stored as

fib[1], fib[2], ..., fib[10].

Namely, fib[i] for i from 0 to 10:

```
> seq( fib[i], i=0..10 );
                    0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55
```

In general, seq(f,i=m..n) generate a sequence in ther form of f for i from m to n. For example, to generate a sequence of odd numbers:

```
> seq( 2*i-1, i=1..20 );
      1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39
```

To generate a sequence of perfect squares:

```
> seq( i^2,  i=1..15 );
        1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225
```

3.1.3 Initialize an array using seq

An array can be initialize directly

```
> t:=array(1..5,[2,6,8,10,s]);
                         t := [2, 6, 8, 10, s]
```

We may initialize an array with command seq, if the entries of the array have a clear pattern. For example: To generate an array of even numbers:

```
> even_number:=array(1..20, [seq(2*i, i=1..20)]);
even_number := [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40]
```

After the array is so generated, we can conduct computation with the array. For example, the sum of the 8-th and 12-th even numbers:

```
> even_number[8]+even_number[12];
                                  40
```

Example: The array of perfect squares:

```
> squ:=array(1..50,[seq(j^2, j=1..50)]);
  squ := [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361,
  400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156,
  1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209,
  2304, 2401, 2500]
```

3.1.4 Entries of an array

You can define any entry of an existing array as any type of data allowed in Maple, even as another array.

```
> s:=array(1..3);
                         s := array(1..3, [])
> s[1]:=5; s[2]:=`John Doe`;  s[3]:=array(1..3,[Mark,Bob,Jack]);
```

$$s_1 := 5$$

$$s_2 := \mathit{John\ Doe}$$

$$s_3 := [\mathit{Mark},\ \mathit{Bob},\ \mathit{Jack}]$$

Since s[3] is an array, to reference its 2nd entry:

```
> s[3][2];
```

Bob

More examples:

Define an array of four Names, each name consists of first and last names. At beginning, open an array of 4 entries, each entry is an (empty) array of 2 entries:

```
> Name:=array(1..4,[seq( array(1..2), i=1..4)]);
```

$$Name := [\%1, \%1, \%1, \%1]$$
$$\%1 := [?_1, ?_2]$$

```
> Name[1][1]:=Bill:  Name[1][2]:=Clinton:
> Name[2][1]:=Al:  Name[2][2]:=Gore:
> Name[3][1]:=Bob:  Name[3][2]:=Dole:
> Name[4][1]:=Jack:  Name[4][2]:=Kemp:
> eval(Name);
```

$$[[Bill,\ Clinton],\ [Al,\ Gore],\ [Bob,\ Dole],\ [Jack,\ Kemp]]$$

The third name is Bob Dole:

```
> eval(Name[3]);
```

$$[Bob,\ Dole]$$

The first name of the fourth person is Jack:

```
> Name[4][1];
```

Jack

3.1.5 Example: An array of prime numbers

A prime number is a number that can only be divided by 1 and itself. Maple has a function "isprime" to identify prime numbers. For example, we know that 3 is prime but 4 is not:

```
> isprime(3);
```

true

```
> isprime(4);
```

false

We can write a simple program to put n prime numbers in an array:

```
# The program of generating an array of prime numbers
# Arguments:
#   Input: n --- the number of primes to be generated
#   Output: p --- the array that contains the arrays
#
```

```
primearray:=proc(n,p)
   local i, k, count;

   p:=array(1..n);          # open the array to store output data
   count:=0;                # initialize the count
   k:=2;                    # the first integer to be examined

   while count < n do       # repeat until n primes are counted
      if isprime(k) then    # check if k is a prime
         count:=count+1;    # if so, count it
         p[count]:=k        # and store it in array p
      fi;
      k:=k+1;               # prepare the next integer
   od;

   print(`I got them`)      # print out a success message
end;
```

```
> read(`a:priarray.txt`);
```

primearray := **proc**(n, p)
local i, k, *count*;
$p :=$ array($1..n$) ;
count := 0 ;
$k := 2$;
while *count* $< n$ **do**
if isprime(k) **then** *count* := *count* + 1 ; $p_{count} := k$ **fi** ; $k := k + 1$
od;
print(`*I got them*`)
end

```
> primearray(50,p);
```

I got them

```
> seq(p[i],i=1..50);
```

```
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,
101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179,
181, 191, 193, 197, 199, 211, 223, 227, 229
```

```
> p[25]*p[12];
```

```
3589
```

Remark: In this example, we don't print out solutions inside the program. Instead, we use an argument p as output, which is an array of the data we want. After running the program, the data is on surface of Maple and can be referenced for other purpose.

3.1.6 Example: Identifying the maximum entry of an array

Find the maximum number in an number array: 2, -5, 9, π

```
#
# program to find the index of the maximum entry of a number array
# Input: m --- leading index of the array
#        n --- (>m) ending index of the array
#        s --- the array of numbers
# Output: index_max --- the index of the maximum entry
#         value_max --- the value of the maximum entry
#
find_max:=proc(m, n, s)
   local i, value_max, index_max;

   value_max:=evalf(s[m]);              # assume the first one is max
   index_max:=m;

   for i from m+1 to n do               # check the remaining entries 1 by 1
      if evalf(s[i]) > value_max then   # if a bigger value is found ...
         value_max:=evalf(s[i]);        #   update the maximum value
         index_max:=i;                  #   update the index of the max
      fi;
   od;

   print(`The maximum value:`);
   print( value_max );
   print(`The index of the maximum value`);
   print( index_max );

end;
```

```
> data:=array(1..7,
> [2, -5, 9, Pi, sqrt(5), -exp(1), sin(4*Pi/5)]);
```

$$data := \left[2,\ -5,\ 9,\ \pi,\ \sqrt{5},\ -e,\ \frac{1}{4}\sqrt{2}\sqrt{5-\sqrt{5}}\right]$$

```
> read(`a:findmax.txt`);
```

find_max := **proc**(m, n, s)
local i, *value_max*, *index_max*;
 value_max := evalf(s_m) ;
 index_max := m ;
 for i **from** m **to** n **do**
  if *value_max* < evalf(s_i) **then** *value_max* := evalf(s_i) ; *index_max* := i **fi**
 od;
 print(‘*The maximum value* : ‘) ;
 print(*value_max*) ;
 print(‘*The index of the maximum value*‘) ;
 print(*index_max*)
end

```
> find_max(1,7,data);
```

The maximum value :

9.

The index of the maximum value

3

3.2 Some statistical measurements

3.2.1 Example 1. Range

Let x_1, x_2, $\cdots$, x_n be a sequence of data. The range R is defined as $\max_i\{x_i\} - min_i\{x_i\}$. Write a program to calculate the range for input n and the data sequence.

```
#
# program to find the range of a number array, defined as
#      the maximum minus the minimum
#
# Input: m --- leading index of the array
#        n --- (>m) ending index of the array
#        x --- the array of numbers
# Output: The range
#
Range:=proc(m, n, x)
   local i, value_max, value_min;

   value_max:=evalf(x[m]);             # At beginning, assume the first
   value_min:=evalf(x[m]);             #   one is max and min

   # find the maximum

   for i from m to n do                # check the remaining entries
      if evalf(x[i]) > value_max then  # if a bigger value is found ...
         value_max:=evalf(x[i]);       #   update the maximum value
      fi;
   od;

   # find the minimum

   for i from m to n do                # check the remaining entries
      if evalf(x[i]) < value_min then  # if a smaller value is found ...
         value_min:=evalf(x[i]);       #   update the maximum value
      fi;
   od;

   # calculate and output the range

   value_max - value_min               # output of the range

end;
```

```
> read(`a:range.txt`);
```

```
Range := proc(m, n, x)
local i, value_max, value_min;
    value_max := evalf(x_m);
    value_min := evalf(x_m);
    for i from m to n do if value_max < evalf(x_i) then value_max := evalf(x_i) fi
    od;
    for i from m to n do if evalf(x_i) < value_min then value_min := evalf(x_i) fi od
    ;
    value_max − value_min
end
```

```
> a:=array(1..10,[2,45,-23,25,43,-26,89,-19,56,9]);
```

$$a := [2, 45, -23, 25, 43, -26, 89, -19, 56, 9]$$

```
> rang:=Range(1,10,a);
```

$$rang := 115.$$

3.2.2 Sorting

Suppose we have a sequence x_m, x_{m+1}, $\cdots$, x_n, and we want to rearrange the sequence to ascending order. We use m, instead of number 1, here as the leading index of the array for flexibility. In earlier examples we have seen that the array index may start from 0 rather than 1. The following process rearranges the array in ascending order..

step m: find the smallest entry among $x_m, x_{m+1}, \cdots, x_n$ and swap it to x_m.

step m+1: find the smallest entry among $x_{m+1}, x_{m+2}, \cdots, x_n$ and swap it to x_{m+1}.

step m+2: find the smallest entry among $x_{m+2}, x_{m+3}, \cdots, x_n$ and swap it to x_{m+2}.

. . . . . .

step n-1: find the smaller entry between x_{n-1} and x_n and swap it to x_{n-1}

(Question: Is there step n?)

Generally, we have a loop for $k = m, m+1, m+2, \cdots, n-1$,

step k: find the smallest entry among $x_k, x_{k+1}, \cdots, x_n$ and swap it to x_k.

Within step k, it is also necessary to have a loop to find the smallest entry in a subsequence. Therefore, we have a *nested loop*.

The program:

```
#
# Program that sort a sequence of data
#    x[m], x[m+1], ... , x[n]
# into ascending order
#
# input: m   ---  leading index of the sequence
#          n   ---  (>n) ending index of the sequence
#          x   ---  the data array with index range equal
#                   beyond [m,n]
#
sort_ascend:=proc(m,n,x)
   local k, j, tmp, value_min, index_min;

   if n < m then
      print(`starting index should not be larger than ending index`);
      RETURN()
   fi

   for k from m to n-1 do

      # find the value_min and index_min of   x[k],x[k+1],...,x[n]

      value_min:=evalf(x[k]);
      index_min:=k;

      for j from k+1 to n do
         if value_min > evalf(x[j]) then
            value_min:=evalf(x[j]);
            index_min:=j;
         fi;
      od;

      # swap to k-th entry if necessary

      if index_min > k then
         tmp:=x[index_min];
         x[index_min]:=x[k];
         x[k]:=tmp;
      fi;

   od;

   print(`ascending sorting finished`);

end;
```

```
> read(`a:sortascd.txt`):
> a:=array(2..10,[3,-4,9,8,-Pi,exp(1),sqrt(12),1/3,0]):
> sort_ascend(2,10,a);
```

ascending sorting finished

```
> seq(a[i],i=2..9);
```

$$-4,\ -\pi,\ 0,\ \frac{1}{3},\ e,\ 3,\ 2\sqrt{3},\ 8$$

3.3 Sieving

3.3.1 Sieving

Sieving is the process of striking out entries in a sequence

$$x_m, x_{m+1}, \cdots, x_n$$

that have (or do not have) a certain property.

One way to sieve is to have a shadow sequence, say

$$s_m, s_{m+1}, \cdots, s_n$$

all initialized with string "alive". If x_k is to be removed, assign $s_k :=$ "killed". At the end of the sieving, check those s_m, s_{m+1}, $\cdots$, s_n, if anyone, say $s_j =$ "alive", then the corresponding x_j survives the sieving.

3.3.2 Example: prime numbers by sieving

The problem is to find all prime numbers between 1 and n by striking out multiples of prime numbers.

The first prime number is 2, so we strike all multiples of 2. i.e. we mark numbers 2*3, 2*4, 2*5, $\cdots$ (up to n) as killed.

The next survived number is prime number 3, so we strike all multiples of 3. i.e. we mark 3*3, 3*4, 3*5, $\cdots$ (up to n) as killed. (Note we don't have to strike out 3*2, which is already killed)

The next survivor is 5 so we strike all multiples of 5, i.e., we mark 5*5, 5*6, 5*7, $\cdots$ (up to n) as killed. (Note, we don't have to kill 5*2, 5*3, 5*4, which were killed already).

Generally, the current prime number is k, we strike all multiple of $k$as killled. (Note, we don't have to kill k*2, k*3, ..., k*(k-1), which are killed already). Then we search the next survivor as next k, and strike out its multiples if $k\,k \leq n$.

```
#
# Program that find all prime numbers within n by sieving
#
# input:  n --- the upper bound for searching
# output: p --- the array that contains all prime
#               numbers up to n
```

```
#
prime_sieve:=proc(n,p)
   local s, i, k, count, flag;

   # initialize the shadow array
   s:=array(1..n,[ seq("alive",i=1..n) ] );

   # 1 is not prime
   s[1]:="killed";

   # quick exit if n < 2
   if n < 2 then
      print(`There is no prime numbers in the given range`);
      RETURN()
   fi

   # sieve until k*k > n
   for k from 2 to n while k^2 <= n do

      # mark every i*k as killed if k is prime
      if s[k]="alive" then
         for i from k to n while i*k <= n do
            s[i*k]:="killed";
         od;
      fi;

   od;

   # count the number of primes found
   count:=0;
   for i from 1 to n do
      if s[i]="alive" then
         count:=count+1;
      fi;
   od;

   # open an array to store prime numbers
   p:=array(1..count);

   # initialize k as counter
   k:=0;
   for i from 1 to n do
      if s[i]="alive" then
         # when s[i] is `alive`, i is the k-th prime
         k:=k+1;
         p[k]:=i;
      fi;
   od;

   print(`The number of primes:`);
   count;

end;
```

Now let's find out all prime numbers from 1 to 1000.

```
> read(`a:prmsieve.txt`):
```

```
> prime_sieve(1000,p);
```

The number of primes :

168

```
> eval(p);
```

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,
101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179,
181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269,
271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367,
373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461,
463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571,
577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661,
673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773,
787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883,
887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]

3.3.3 Example: Ulam's lucky numbers

From the list of positive integers 1, 2, 3, 4, $\cdots$, remove every second number, leaving 1, 3, 5, 7, 9, ... Since 3 is the first surviving number above 2 that has not been used as the "killer", we remove every 3rd number from the *remaining* numbers, yielding 1, 3, 7, 9, 13, 15, 19, 21, ... Now every 7th survivor is removed, leaving 1, 3, 7, 9, 13, 15, 21, ... Numbers that are never removed are considered to be "lucky". Write a program which prints the lucky numbers up to n.

```
#
#  Ulam's lucky numbers: For a sequence
#        1, 2, 3, ..., n
#  remove every 2nd number, it remains:
#        1, 3, 5, 7, 9, 11, 13, ...
#  then remove every 3rd number. Generally, after
#  removing every k-th number, let m be the first
#  surviving number that is larger than k. Update
#  k to be m and remove every k-th number in the
#  surviving sequence. This process is continued
#  until k is larger than the size of the surviving
#  sequence.
#
#  input:  n        --- the upper bound of searching
#  output: lucky    --- array that contains all lucky numbers up to n
#

ulumluck:=proc(n,lucky)

   local i,count,u,k,flag,survive;
   #
   #  initialize the array u.
   #     u[i]='alive'  : i survives
   #     u[i]='killed' : i removed.
   #
   u:=array(1..n);
   for i from 1 to n do
      u[i]:="alive"
   od;

   survive:=n;   # there are n survivors at beginning

   for k from 2 to n while survive > k  do

      #
      #  remove every k-th number if k is alive
      #
      if u[k]="alive" then
         count:=0;                    # initialize count
         for i from 1 to n do
            if u[i]="alive" then      # found a survivor
               count:=count+1;        # count the survivor
               if count=k then        # the k-th one?
                  u[i]:="killed";     # remove it
                  count:=0;           # reset count
                  survive:=survive-1; # one less survivor
               fi;
            fi;
         od;
      fi;

   od;

   #
   # record survivors in array lucky
   #
   lucky:=array(1..survive);   # open array lucky
   count:=0;                   # set count
   for i to n do
      if u[i]="alive" then     # found a survivor
         count:=count+1;       # count it
         lucky[count]:=i;      # record in lucky

      fi;
   od;

   print('The Ulam's lucky numbers');
   seq(lucky[i],i=1..count);   # print

end;
```

```
> read('a:ulam.txt'):
```

```
> ulamluck(100,lucky);
```

The Ulam's lucky numbers

1, 3, 7, 9, 13, 15, 21, 25, 31, 33, 37, 43, 49, 51, 63, 67, 69, 73, 75, 79, 87, 93, 99

3.4 Projects with arrays

3.4.1 Pascal triangle

The coeficients of binomials $(a+b)^n$ can be calculated using the so called the Pascal triangle:

n	$(a+b)^n$	coeficients $c_0, c_1, \cdots c_n$
0	1	1
1	$a+b$	1 1
2	$a^2+2\,a\,b+b^2$	1 2 1
3	$a^3+3\,a^2\,b+3\,a\,b^2+b^3$	1 3 3 1
4	$a^4+4\,a^3\,b+6\,a^2\,b^2+4\,a\,b^3+b^4$	1 4 6 4 1
5	$a^5+5\,a^4\,b+10\,a^3\,b^2+10\,a^2\,b^3+5\,a\,b^4+b^5$	1 5 10 10 5 1
...	...	...

There are some obvious facts about the coeficients $c_0, c_1, c_2, \cdots, c_n$

- For each n, there are $(n+1)$ coeficients $c_0, c_1, c_2, \cdots, c_n$
- $c_0 = c_n = 1$.
- Let $d_0, d_1, d_2, \cdots, d_{k-1}$ be the coeficients of $(a+b)^{(k-1)}$, then

$$c_0 = 1, \;\; c_1 = d_0 + d_1, \;\; c_2 = d_1 + d_2, \;\; \cdots, \;\; c_{k-1} = d_{k-2} + d_{k-1}, \;\; c_k = 1$$

Write a program, for input n output (1) the coeficient array c, **as an argument**, and (2) print out the Pascal triangle. You program should work like the following:

```
>  read(‘a:pascal.txt‘):
>  Pascal_triangle(10,c);
```

```
                 1
                1, 1
              1, 2, 1
             1, 3, 3, 1
            1, 4, 6, 4, 1
          1, 5, 10, 10, 5, 1
        1, 6, 15, 20, 15, 6, 1
      1, 7, 21, 35, 35, 21, 7, 1
    1, 8, 28, 56, 70, 56, 28, 8, 1
```

```
                    1, 9, 36, 84, 126, 126, 84, 36, 9, 1
                 1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1
>  seq( c[j], j=0..10 );
                 1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1
```

3.4.2 Means

For a sequence of real numbers

$$x_m,\ x_{m+1},\ x_{m+2},\ \cdots, x_n$$

various means are defined as follows:

(i) Arithmetic mean A:

$$A = \sum_{i=m}^{n} \frac{x_i}{n-m+1}$$

(ii) Geometric mean G:

$$G = (\prod_{i=m}^{n} x_i)^{(\frac{1}{n-m+1})}$$

(iii) Harmonic mean H:

$$\frac{1}{H} = \frac{\sum_{i=m}^{n} \frac{1}{x_i}}{n-m+1}$$

Write a program that, for input m, n, x; print out all three means. Your program should work like

```
>  x:=array(0..10,[2,5,9,12,21,33,18,8,11,15,17]):
>  read(‘a:means.txt‘):
>  Mean(0,10,x);
                          The arithmetic mean :
                              13.72727273
                          The geometric mean :
                              11.05802293
                          The harmonic mean :
                              8.033176344
>  y:=array(1..9,[3,9,12,43,31,18,24,38,5]):
>  Mean(1,9,y);
                          The arithmetic mean :
                              20.33333333
```

The geometric mean :
14.86548789
The harmonic mean :
9.924686319

We can also calculate the means of the binomial coeficients:

```
>  Pascal_triangle(12,c):
```

```
                          1
                         1, 1
                       1, 2, 1
                      1, 3, 3, 1
                    1, 4, 6, 4, 1
                  1, 5, 10, 10, 5, 1
               1, 6, 15, 20, 15, 6, 1
             1, 7, 21, 35, 35, 21, 7, 1
          1, 8, 28, 56, 70, 56, 28, 8, 1
       1, 9, 36, 84, 126, 126, 84, 36, 9, 1
  1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1
 1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1
1, 12, 66, 220, 495, 792, 924, 792, 495, 220, 66, 12, 1
```

```
>  Mean(0,12,c);
```

The arithmetic mean :
315.0769231
The geometric mean :
78.51839612
The harmonic mean :
5.872498536

3.5 Exercises

1. **Fibonacci array**

   Write a program that, for input n, output an array of Fibonacci sequence

   $$F_0,\, F_1 \cdots F_n$$

   Do not print out the solution inside the program. Instead, use an argument as output and use seq to show results after running the program.

   Sample results

   ```
   >  read(`a:fibarray.txt`):
   >  n:=40:
   ```

```
> fibarray(40,f);
```

Fibonacci sequence is generated

```
> seq(f[k],k=0..n);
```

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817, 39088169, 63245986, 102334155

2. **Inner product (or dot product)**

   A n-vector is defined as an array of n numbers

$$(v_1, v_2, \cdots, v_n)$$

   such as

$$a = (1, 3, 5, -3, 2), \quad b = (9, -5, 2, \sqrt{7}, \pi)$$

   The inner product (or dot product) of two n-vectors

$$a = (a_1, a_2, \cdots, a_n), \quad \text{and} \quad b = (b_1, b_2, \cdots, b_n)$$

$$a \cdot b = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

   Write a program, for input n, a, and b, print out the value of the their inner product.

   Sample results

```
> read(`a:dotprod.txt`):
> n:=5:
> a:=array(1..n, [1,3,5,-3,2]);
```

$$a := [1,\ 3,\ 5,\ -3,\ 2]$$

```
> b:=array(1..n, [9, -5, 2, sqrt(7), Pi]);
```

$$b := \left[9,\ -5,\ 2,\ \sqrt{7},\ \pi\right]$$

```
> dotprod(a,b,n);
```

The dot product is

2.345931375

3. **Deck list**

   Write a program that defines an array of 52 entries, each entry contains the the rank and suit of a card in a standard deck. That is, each entry is an array of two entries, the first entry is the rank, and the second entry the suit. The deck must be in the order of Example 3, Lecture Note 02. Write a program using do loops instead of 52 line program.

   (Hint: Define *deck* as a 52-entry array. deck[1] is defined as a two entry array, to be referenced as deck[1][1] and deck[1][2], such that deck[1][1]=ace, deck[1][2]=heart.)

   Sample results

```
> read(`a:decklist.txt`):
```

```
> decklist(deck):
> eval(deck);
```

[[*ace*, *heart*], [2, *heart*], [3, *heart*], [4, *heart*], [5, *heart*], [6, *heart*], [7, *heart*], [8, *heart*], [9, *heart*], [10, *heart*], [*jack*, *heart*], [*queen*, *heart*], [*king*, *heart*], [*ace*, *spade*], [2, *spade*], [3, *spade*], [4, *spade*], [5, *spade*], [6, *spade*], [7, *spade*], [8, *spade*], [9, *spade*], [10, *spade*], [*jack*, *spade*], [*queen*, *spade*], [*king*, *spade*], [*ace*, *diamond*], [2, *diamond*], [3, *diamond*], [4, *diamond*], [5, *diamond*], [6, *diamond*], [7, *diamond*], [8, *diamond*], [9, *diamond*], [10, *diamond*], [*jack*, *diamond*], [*queen*, *diamond*], [*king*, *diamond*], [*ace*, *club*], [2, *club*], [3, *club*], [4, *club*], [5, *club*], [6, *club*], [7, *club*], [8, *club*], [9, *club*], [10, *club*], [*jack*, *club*], [*queen*, *club*], [*king*, *club*]]

```
> eval(deck[18]);
```

[5, *spade*]

```
> eval(deck[37]);
```

[*jack*, *diamond*]

4. **Minimum**

   Write a program that identifies the mininum entry of an array by printing out the value of the manimum entry and its index. Use the data in Example 2 to test your program.

5. **Standard deviation**

   Write a program to calcaulate the standard deviation

$$s = \sqrt{\sum_{i=1}^{n} \frac{(x_i - \sigma)^2}{n}}$$

   where σ is the average of x_1, x_2, $\cdots$, x_n, i.e.:

$$\sigma = \sum_{i=1}^{n} \frac{x_i}{n}$$

6. **Median and quartiles**

   Let

$$x_m, x_{m+1}, \cdots, x_n$$

   be a sequence of $n - m + 1$ real numbers in *ascending* order. The leading index m can be 0, 1 or any other integer. The **median** this sequence is defined as follows:

   case 1. if $n-m+1$ is odd, then the median is the middle term at index $\frac{n+m}{2}$.

   case 2. if $n - m + 1$ is even, then the median is the average of middle-left term and the middle-right term with indices $\frac{n+m-1}{2}$ and $\frac{n+m+1}{2}$ respectively.

The median is also called the **second quartile**.

The **first quartile** and the **third quartile** are defined as follows:

case 1: *if* $n+m-1$ *is even*, then the sequence can be devided into the first half sub-sequence

$$x_m, \cdots, x_{\frac{n+m-1}{2}}$$

and the second half sub-sequence

$$x_{\frac{n+m+1}{2}}, \cdots, x_n.$$

The **first quartile** is the median of the first half sub-sequence and the **third quartile** is the median of the second half sub-sequence

case 2: *if n-m+1 is odd*, then there is a middle term $x_{\frac{n+m}{2}}$ The **first quartile** is defined as the median of the sub-sequence

$$x_m, \cdots, x_{\frac{n+m}{2}-1}$$

and the **third quartile** is defined as the median of the sub-sequence

$$x_{\frac{n+m}{2}+1}, \cdots, x_n.$$

Write a program, for any given sequence

$$x_m, x_{m+1}, \cdots, x_n$$

print out the three quartiles. Your program should accept input m, n, and array x, rearrange it into ascending order, and find quartiles.

Hint: You may consider the following steps:

Input: m, n, x

(1) re-arrange the array x in ascending order

(2) according to the odd/even number $n-m+1$

- find the median (2nd quartile)
- find the index range $m1$ and $n1$ of the first sub-sequence
- find the index range $m2$ and $n2$ of the second sub-sequence

(3) according to odd/even type of $n1-m1+1$ and $n2-m2+1$, find the first quartile and the third quartile respectively.

Sample results

```
> read(`c:/zeng/teach/340/quartile.txt`):
> x:=array(0..10,[23,12,34,87,25,10,5,19,65,29,71]):
> quartile(x,0,10);
```

The sequence in ascending order

```
            5, 10, 12, 19, 23, 25, 29, 34, 65, 71, 87

                      The 1st subsequence
                       5, 10, 12, 19, 23
                      The 2nd subsequence
                       29, 34, 65, 71, 87

              The 1st, 2nd, and 3rd quartiles :
                             12
                             25
                             65
>  y:=array(1..14,[22,11,33,44,51,62,12,81,37,19,9,20,18,5]);
     y := [22, 11, 33, 44, 51, 62, 12, 81, 37, 19, 9, 20, 18, 5]
>  quartile(y,1,14);
                 The sequence in ascending order
        5, 9, 11, 12, 18, 19, 20, 22, 33, 37, 44, 51, 62, 81

                      The 1st subsequence
                    5, 9, 11, 12, 18, 19, 20
                      The 2nd subsequence
                   22, 33, 37, 44, 51, 62, 81

              The 1st, 2nd, and 3rd quartiles :
                             12
                            21.0
                             44
```

7. **Partial amnesty**

   In the central prison of Sikinia there are n cells numbered from 1 to n, each occupied by a single prisoner. The state of each cell can be changed from closed to open and vice versa by a half-turn of the key. To celebrate the Centennial Anniversary of the Republic it was decided to grant a partial amnesty. The president sent an officer to the prison with the instruction:

   ```
   for i from 1 to n do
       turn the keys of cells i, 2*i, 3*i, ... (up to n)
   od;
   ```

   A prisoner was freed if *at the end* his/her door was open. Which prisoners are were set free? Write a program that carry out the amnesty for n=100, 200, 500.

   (Hint: The door can be either ‘open‘ or ‘closed‘. Each time the key is turned, the door becomes ‘closed‘ if it is open, or becomes ‘open‘ if it is ‘closed‘. A door may be set open or closed many times)

   Sample results

```
> read('a:amnesty.txt'):
> amnesty(100,free);
                    The number of prisoners freed
                                10
```

The following prisoners were set free:

```
> eval(free);
              [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
> amnesty(200,free);
                    The number of prisoners freed
                                14
> eval(free);
        [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196]
> amnesty(500,free);
                    The number of prisoners freed
                                22
> eval(free);

          [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169,
           196, 225, 256, 289, 324, 361, 400, 441, 484]
```

8. **The Josephus problem**

During the Jewish rebellion against Rome (A.D. 70) 40 Jews were caught in a cave. In order to avoid slavery they agreed upon a program of mutual destruction. They would stand in a circle and number themselves from 1 to 40. Then every seventh person was to be killed until only one was left who would commit suicide. The only survivor, Flavius Josephus, who did not carry out the last step, became a historian.

Josephus problem: n persons are arranged in a circle and numbered from 1 to n, Then every k-th person is removed, with the circle closing up after every removal. Print the list of persons in the order of being removed. What is the number f(n) of the last survivor?

Write a program to

(1) Print out the sequence of persons in the order of being killed.

(2) Find out where Josephus is.

Sample results

```
> read('a:josephus.txt'):
> Josephus(40,7);
                People were killed in the following order

7, 14, 21, 28, 35, 2, 10, 18, 26, 34, 3, 12, 22, 31, 40, 11, 23, 33, 5, 17, 30,
 4, 19, 36, 9, 27,6, 25, 8, 32, 16, 1, 38, 37, 39, 15, 29, 13, 20
                            Josephus is at
                                24
```

Suppose we have 14 students registered for this class, so we have 15 people (including the instructor) in the classroom, if we play this game:

```
>  Josephus(15,7);
```

People were killed in the following order

7, 14, 6, 15, 9, 3, 13, 11, 10, 12, 2, 8, 1, 4

Josephus is at

5

The fifth person is the final survivor. Or, if every 5th person is removed,

```
>  Josephus(15,5);
```

People were killed in the following order

5, 10, 15, 6, 12, 3, 11, 4, 14, 9, 8, 13, 2, 7

Josephus is at

1

The first person will survive.

9. **Weighted means:**

Let

$$\lambda = (\lambda_m, \lambda_{m+1}, \cdots, \lambda_n)$$

be a vector of positive numbers such that

$$\sum_{i=m}^{n} \lambda_i = 1$$

then the weighted arithmetic and geometric means associated with λ are defined as

(i) weighted arithmetic mean:

$$A_\lambda = \sum_{i=m}^{n} \lambda_i\, x_i = \lambda_m\, x_m + \lambda_{m+1}\, x_{m+1} + \cdots + \lambda_n\, x_n$$

(ii) weighted geometric mean:

$$G_\lambda = \prod_{i=m}^{n} {x_i}^{\lambda_i} = \left(({x_m}^{\lambda_m}\right)\left({x_{m+1}}^{\lambda_{m+1}}\right)\cdots\left({x_n}^{\lambda_n}\right)$$

Let the weights

$$\lambda_m,\ \lambda_{m+1},\ \cdots,\ \lambda_n$$

be defined as follows:

$$\lambda_m = \frac{1}{2}, \lambda_{m+1} = \frac{1}{4}, \cdots, \lambda_i = \frac{\lambda_{i-1}}{2}, \cdots, \lambda_{n-1} = \frac{\lambda_{n-2}}{2}, \lambda_n = \lambda_{n-1}$$

Write a program that, for input m, n, x, calculate both weighted means. Use your program to calculate the weighted means of the two sequences in Project 2 above.

Sample results

```
> read('a:wmeans.txt'):
> Weighted_means(0,10,x);
                    The weighted arithmetic mean
                           5.521484376
                    The weighted geometric mean
                           3.933248559
> Weighted_means(1,9,y);
                    The weighted arithmetic mean
                           9.54296875
                    The weighted geometric mean
                           6.309771865
> Weighted_means(0,12,c);
                    The weighted arithmetic mean
                           64.87329102
                    The weighted geometric mean
                           6.551390952
```

10. **The power of binomials**

Write a program that, for input a, b, n, calculate $(a+b)^n$ using the Pascal triangle and the resulting coeficients of binomials. Your program should have the structure like:

step 1, calculate

$$c_0,\, c_1,\, \cdots,\, c_n$$

using Pascal triangle

step 2, calculate and print out

$$c_0\, a^n + c_1\, a^{(n-1)}\, b + c_2\, a^{(n-2)}\, b^2 + c_3\, a^{(n-3)}\, b^3 + \cdots + c_{n-1}\, a\, b^{(n-1)} + c_n\, b^n$$

Chapter 4

Probability simulations

4.1 Declaring data types for program arguments

Maple classifies data into the following types (ask Maple: ?type)

```
!          .          And        NONNEGATIVE Not
Or         PLOT       PLOT3D     Point       Range
RootOf     TEXT       ‘**‘       ‘*‘         ‘+‘
‘^‘        algebraic  algext     algfun      algnum
algnumext  anyfunc    anything   arctrig     array
atomic     boolean    complex    complexcons constant
cubic      dependent  disjcyc    equation    even
evenfunc   expanded   exprseq    facint      float
fraction   freeof     function   hfarray     identical
indexed    indexedfun infinity   integer     intersect
laurent    linear     list       listlist    literal
logical    mathfunc   matrix     minus       monomial
name       negative   negint     nonneg      nonnegint
nonposint  nothing    numeric    odd         oddfunc
operator   point      polynom    posint      positive
prime      procedure  protected  quadratic   quartic
radalgfun  radalgnum  radext     radfun      radfunext
radical    radnum     radnumext  range       rational
ratpoly    realcons   relation   rgf_seq     scalar
```

Among them, the following types are frequently used in this book:

```
array, numeric, integer, odd, even, positive, negative
```

When writing a program, usually we want each argument to be a certain type data, and we would like the machine to check if the input item fits the data type. For example, in the program of finding quartiles in the array members

$$x_m, x_{m+1}, \cdots, x_n$$

we can use three arguments m, n, x as integer, integer, and array respectively. Then we can start the program definition like

```
quartiles:=proc(m::integer, n::integer, x::array)
```

When executing the program, Maple will automatically check each input item to see if it fits the corresponding type and, if not, output an error message.

Example: The following program calculates the square root of the sum of three real numbers.

```
sqrt3:=proc(
            a::numeric,   # input: real number
            b::numeric,   # same as above
            c::numeric,   # same as above
          ans::evaln      # output: must be declared "evaln"
            );

   ans:=sqrt(a+b+c);

end;
```

```
>  read(`a:/txt/3sqrt.txt`):

>  sqrt3(3,5,8,t);
                                4

>  sqrt3(s,5,7,ans);

Error, sqrt3 expects its 1st argument, a, to be of type numeric, but
received s
```

As shown in this example, if an argument is designated as an output item, it must be declared "evaln".

4.2 Probability experiment

4.2.1 Introduction

In probability theory, an **experiment** is an activity or occurrence with an observable result, such as drawing a card from a deck. Each repetition of an experiment is called a **trial**. For example, there are three white balls and five red balls in a box. Drawing a ball from the box, one can observe its color. So drawing a ball from a box is an experiment. If one replace the ball back to the box after it is drawn, this experiment is called *drawing with replacement*. If that's the case, there can be infinitely many trials for the experiment.

If we ask a question: what is the probability of drawing, with replacement, three consecutive red balls? We may answer the question theoretically: $p = (\frac{5}{8})^3$, or we may set a program and ask Maple to simulate the experiment and calculate the probability approximately.

4.2.2 Random number generators

4.2.3 Random real number generator

The function rand() generates a 12 digit random positive integer. The following program generates a random real number in the interval $[a, b]$:

```
#
# function that output a random real number in [a,b]
#
real_ran:=proc(
               a::numeric,
               b::numeric
              )
   local x;
   x:=rand();
   evalf( a+ (b-a)*x/999999999999 );

end;
```

```
> read(`a:/txt/realran.txt`):

> s:=real_ran(-3,2);
```

$$s := 1.603124737$$

4.2.4 Random integer generator

The following function generates an integer between m and n, inclusive

```
#
# program generates a random integer between m and n, inclusive
#
int_ran:=proc(
              m::integer,
              n::integer
             )

   round(evalf(m-0.5+(n-m+1)*rand()/999999999999))

end;
```

```
> read(`a:/txt/intran.txt`):
> k:=int_ran(1,52);
```

$$k := 50$$

```
> m:=int_ran(1,52);
```

$$m := 8$$

4.2.5 Example: Drawing a ball with replacement

Suppose in a box, there are 3 white balls and 5 red balls. We can consider balls numbered from 1 to 3 are white and those numbered from 4 to 8 are red. Randomly drawing a ball, mathematically, is equivalent to drawing a number from 1 to 8, and we'll know the color according to the number. Therefore, the experiment of "drawing a ball" can be simulated by the program

```
one_ball:=proc()
   local k;
   k:=int_ran(1,8);
   if k<=3 then
      "white";
   else
      "red";
   fi;
end;

int_ran:=proc(
              m::integer,
              n::integer
             )

   round(evalf(m-0.5+(n-m+1)*rand()/999999999999))

end;
```

Program note

- The program one_ball does not require any input.
- The program int_ran is used by one_ball as a "subprogram". Therefore it should be included in the same text file. When the file is read into Maple, both main program and subprogram will be loaded.

Test run:

```
> read('a:oneball.txt'):
> one_ball();
                                    "white"
> one_ball();
                                    "red"
```

4.2.6 Example: Drawing, say 3, balls without replacement

Suppose there are 8 balls in a box, 3 white and 5 red. Generally, if we draw k balls from the box *without replacement*, the experiment can be simulated in the following way:

(1) number balls from 1 to 8. The first three numbers correponds to while, remaining for red.

(2) Draw k balls one by one:

- The first draw is from ball 1 to 8. Then swap the drawn ball to number 8 (=8-1+1)
- The 2nd draw is from ball 1 to 7. Then swap the drawn ball to number 7 (=8-2+1)
- The 3rd draw is from ball 1 to 6. Then swap the drawn ball to number 6 (=8-3+1)
- Generally, the j-th drawn is from ball 1 to $n = 8 - j + 1$, then swap the drawn ball to number n so it won't be redrawn.

(4) Translate numbers to colors.

```
#
#  Program simulating drawing k balls in a box
#  containing 3 white balls and 5 red balls
#
draw_k_ball:=proc(
                 k::integer, # number of balls to draw
              draw::evaln    # output: colors drawn
                )
   local ball, i, j, n, m, temp;

   ball:=array(1..8);      # making a box of balls
   for i to 3 do
      ball[i]:="white";
   od;
   for i from 4 to 8 do
      ball[i]:="red";
   od;

   draw:=array(1..k);      # open space for balls to be drawn

   n:=8;                   # n is the number of balls remaining
                           #    there are 8 balls initially
   for j to k do           # loop to draw balls one by one

      m:=int_ran(1,n);     # draw a number from 1 to n

      draw[j]:=ball[m];    # get the color

      if m <> n and ball[m]<>ball[n] then
         temp:=ball[m];    # move the drawn ball to end of pool
         ball[m]:=ball[n]; # so it won't be redrawn
         ball[n]:=temp;
      fi;

      n:=n-1;              # update n

   od;

end;
```

```
> read(‘a:drawball.txt‘):
> draw_k_ball(3,ball):
> eval(ball);
```

["red", "white", "white"]

4.2.7 Example: Approximate a probability

If we want to calculate the probability of drawing 3 balls without replacement and getting all red, we can simulate the experiment n times, count the successful drawings and use

$$\frac{number\_of\_successful\_drawings}{n}$$

as approximate value of the probability.

```
#
# program that calculate the approximate probability of
# drawing k balls, without replacement, out a box containing 3 red and 5
# white and  getting all red
#
# input:  k --- the number of balls to be drawn
#         n --- the number of trials to repeat
#
draw_test:=proc(  k::integer,
                  n::integer
               )
   local i, j, count, ball, flag;

   ball:=array(1..k);     # open the space for balls
   count:=0;              # initialize counting
   #
   # repete the experiment n times
   #
   for i to n do
      draw_k_ball(k,ball);   # draw k balls w/o replacement
      flag:=0;               # initialize flag=0, assuming successful
      for j to k while flag=0 do   # check balls if they are red
         if ball[j]="white" then
            flag:=1;              # raise the flag: a white ball is found
         fi;
      od;
      #
      # count a successful drawing if all red (flag=0)
      #
      if flag=0 then
         count:=count+1;
      fi;
   od;

   evalf(count/n);    # approximate probability

end;
#
#  Program simulating drawing k balls in a box
#  containing 3 white balls and 5 red balls
#
draw_k_ball:=proc(
                  k::integer, # number of balls to draw
               draw::evaln    # output: colors drawn
                 )
   local ball, i, j, n, m, temp;

   ball:=array(1..8);      # making a box of balls
   for i to 3 do
      ball[i]:="white";
   od;
   for i from 4 to 8 do
      ball[i]:="red";
   od;

   draw:=array(1..k);      # open space for balls to be drawn

   n:=8;                   # n is the number of balls remaining
                           #    there are 8 balls initially
   for i to k do          # loop to draw balls one by one

      m:=int_ran(1,n);    # draw a number from 1 to n

      draw[i]:=ball[m];   # get the color

      if m <> n and ball[m]<>ball[n] then
         temp:=ball[m];   # move the drawn ball to end of pool
         ball[m]:=ball[n]; # so it won't be redrawn
         ball[n]:=temp;
      fi;

      n:=n-1;             # update n

   od;

end;
#
# function that generates a random integer from m to n, inclusive
#
int_ran:=proc(
             m::integer,
             n::integer
            )
   round(evalf(m-0.5+(n-m+1)*rand()/999999999999))
end;
```

```
> read(`a:drawtest.txt`):
> draw_test(3,300);
                              .1700000000
```

The exact probability is

```
> evalf((5!)^2/((2!)*(8!)));
                              .1785714286
```

Now we increase the number of trials to 10000

```
> draw_test(3,10000);
                              .1758000000
```

We got much better approximation. When the number of trials are bigger, it is *more likely*, though no guarantee, to get a more accurate approximation of the probability.

4.2.8 Example: Two points in a square having distance less than one

Given a square of 1 ft by 1 ft, what is the probability of throwing two stones in to the box having distance less than one? It is not easy to calculated the probability exactly. So let's write a program to approximate it.

A 1x1 square can be seen as set of points { (x, y) — $0 \le x \le 1$, and $0 \le y \le 1$,} on the xy-coordinate system. A randomly thrown stone can be represented as a point (x, y) in that set. The simulation of the experiment is to generate two sets of random numbers x and y in the interval [0,1].

```
#
# program that approximates the probability of throwing two points
# in a 1x1 square having distance less than 1
#
#  input:   num_of_trials --- number of trials
#  output:  the approximate probability
#
two_pts:=proc(num_of_trials::integer)
   local count, i, x1, y1, x2, y2, distance;

   if num_of_trials > 0 then
      count:=0;
      for i to num_of_trials do
         #
         # simulate the experiment
         #

         x1:=real_ran(0,1);  # generate the first point
         y1:=real_ran(0,1);

         x2:=real_ran(0,1);  # generate the second point
         y2:=real_ran(0,1);

         #
         # checking
         #
         distance:=evalf(sqrt( (x2-x1)^2+(y2-y1)^2 ));
         if distance < 1.0 then
            count:=count+1;
         fi;
      od;
      evalf(count/num_of_trials);
   else
      print(`invalid input`);
   fi;
end;
#
# function that output a random real number in [a,b]
#
real_ran:=proc(
               a::numeric,
               b::numeric
              )
   local x;
   x:=rand();
   evalf( a+ (b-a)*x/999999999999 );
end;
```

```
> read(`a:twopoint.txt`):
> two_pts(500);
```

```
                                   .9800000000
  >  two_pts(2000);
                                   .9670000000
```

Combined estimate:

```
  >  (0.98*500+0.967*2000)/2500;
                                   .9696000000
```

We can make an estimate of 0.97. Further testing

```
  >  two_pts(10000);
                                   .9769000000
```

To get even better approximation, the number of trials many should be in millions.

4.3 Card games

4.3.1 Program that draws a random k-card hand

The following is the program that draws a random k-card hand. It actually consists of three programs. One program is considered the main program that does the following process:

< 1 > Get k distinct random numbers from 1 to 52 by calling "int_ran" as subprogram

< 2 > For each number obtained, translate it into the corresponding card name by calling "card" as subprogram.

```
#
################## main program ########################################
#
# program that drawing k cards, without replacement
# out a standard deck
#
#  input:  k    --- the number of balls to be drawn
#  output: hand --- array that contains the names of cards drawn
#
draw_k_card:=proc(
                k::integer, # number of cards to draw
              hand::evaln    # output: the drawn hand
                )
   local deck, i, m, n, temp;
   if k < 1 or k > 52 then   # quick exit if k < 1 or k > 52
      print(`1st argument must be between 1 and 52`);
      RETURN();
   fi;

   hand:=array(1..k);        # open the space for output
   deck:=array(1..52,        # making a pool of numbers to draw
         [seq(i,i=1..52)]);
   n:=52;                    # the number of cards remaining

   for i from 1 to k do      # loop to draw card 1 by 1

      m:=int_ran(1,n);       # draw a number from 1 to n
      hand[i]:=deck[m];

      if m<>n then           # move the number to the end
         temp:=deck[m];      # of the pool to avoid redrawn
         deck[m]:=deck[n];
         deck[n]:=temp;
      fi;

      n:=n-1;                # update n

   od;

   #
   # get the names of the cards
   #
   for i from 1 to k do
      m:=hand[i];          # get the number
      hand[i]:=card(m);    # call subprogram that identifies the card
   od;

end;
################# end of the main program ############################

###################### subprogram ####################################
#
#
# function that generates a random integer between m and n, inclusive
#
int_ran:=proc(m::integer,n::integer)
   round(evalf(m-0.5+(n-m+1)*rand()/999999999999))
end;
############# end of subprogram int_ran ##############################

###################### subprogram ####################################
#
# function that identify the card name given its number
# input: m --- a number from 1 to 52
# output: an array of two entries that identifies the card
#
card:=proc(m::integer)
   local rank, suit;
   if m>0 and m<= 52 then  # quick exit if m is beyond 1-52
```

```
      rank:= m mod 13;      # get numerical rank (1-13)
      suit:=ceil(m/13);     # get numerical suit (1-4)
      #
      # identifies the actual suit as a string
      #
      if suit=1 then
         suit:="heart";
      elif suit=2 then
         suit:="spade";
      elif suit=3 then
         suit:="diamond";
      else
         suit:="club";
      fi;
      #
      # identifies the actual rank
      #
      if rank=1 then
         rank:="ace";
      elif rank=11 then
         rank:="jack"
      elif rank=12 then
         rank:="queen"
      elif rank=0 then
         rank:="king"
      fi;                 # remaining cases are rank=2,3,4,...,10,
                          # already defined
      #
      # output the card name as an array
      #
      array(1..2,[rank,suit])
   fi;
end;
############# end of subprogram card #################################
```

4.3.2 Simulation of card games

We can use Maple programs to simulate many probability experiment. Those programs share a similar structure:

```
count:=0; # initialize the counter

for i from 1 to n do

   block 1: simulation
            simulate the experiment

   block 2: checking
            count 1 if the experiment was successful
```

```
        od;
        probability:=evalf(count/n); # calculate the probability
```

We usually need "int_ran", "real_ran" and/or other subprograms.

4.3.3 Example: Drawing a k-card hand having at least one ace

The following is the program

```
#
# program that compute the approximate probability of
# drawing a k-card hand having at least one ace
#
# input:  k --- number of cards in a hand
#         n --- number of trials for the experiment
#
# This program require draw_k_card package as subprogram
#
k_card_get_ace:=proc(k::integer,n::integer)
   local count, i, j, hand, flag;

   count:=0;   # initialize counting
   for i from 1 to n do
      #
      # simulate the experiment
      #
      draw_k_card(k,hand);  # draw a k card hand
      #
      # checking
      #
      flag:=0;              # initialize the flag=0, meaning no ace
      for j from 1 to k while flag=0 do
         if hand[j][1] = "ace" then
            flag:=1;        # raise the flag, an ace is in the hand
         fi;
      od;
      count:=count+flag;    # count
   od;

   evalf(count/n);          # calculate the probability
end;

#
################## subprogram ###################################
#
# program that drawing k cards, without replacement
# out a standard deck
#
```

```
#  input:  k    --- the number of balls to be drawn
#  output: hand --- array that contains the names of cards drawn
#
draw_k_card:=proc(
                 k::integer, # number of cards to draw
               hand::evaln    # output: the drawn hand
                 )
   local deck, i, m, n, temp;

   if k < 1 or k > 52 then   # quick exit if k < 1 or k > 52
      print(`1st argument must be between 1 and 52`);
      RETURN();
   fi;

   hand:=array(1..k);       # open the space for output
   deck:=array(1..52,       # making a pool of numbers to draw
         [seq(i,i=1..52)]);
   n:=52;                   # the number of cards remaining

   for i from 1 to k do     # loop to draw card 1 by 1

      m:=int_ran(1,n);      # draw a number from 1 to n
      hand[i]:=deck[m];

      if m<>n then          # move the number to the end
         temp:=deck[m];     # of the pool to avoid redrawn
         deck[m]:=deck[n];
         deck[n]:=temp;
      fi;

      n:=n-1;               # update n

   od;

   #
   # get the names of the cards
   #
   for i from 1 to k do
      m:=hand[i];         # get the number
      hand[i]:=card(m);   # call subprogram that identifies the card
   od;

end;
################# end of the main program ############################

###################### subprogram ####################################
#
#
# function that generates a random integer between m and n, inclusive
#
int_ran:=proc(m::integer,n::integer)
   round(evalf(m-0.5+(n-m+1)*rand()/999999999999))
end;
############# end of subprogram int_ran ##############################


###################### subprogram ####################################
```

```
#
# function that identify the card name given its number
# input: m --- a number from 1 to 52
# output: an array of two entries that identifies the card
#
card:=proc(m::integer)
   local rank, suit;
   if m>0 and m<= 52 then  # quick exit if m is beyond 1-52

      rank:= m mod 13;     # get numerical rank (1-13)
      suit:=ceil(m/13);    # get numerical suit (1-4)
      #
      # identifies the actual suit as a string
      #
      if suit=1 then
         suit:="heart";
      elif suit=2 then
         suit:="spade";
      elif suit=3 then
         suit:="diamond";
      else
         suit:="club";
      fi;
      #
      # identifies the actual rank
      #
      if rank=1 then
         rank:="ace";
      elif rank=11 then
         rank:="jack"
      elif rank=12 then
         rank:="queen"
      elif rank=0 then
         rank:="king"
      fi;              # remaining cases are rank=2,3,4,...,10,
                       # already defined
      #
      # output the card name as an array
      #
      array(1..2,[rank,suit])
   fi;
end;
############# end of subprogram card #################################
```

```
> read(‘a:k-ace.txt‘):
```

Let’s see what’s the probability of drawing a 4-card hand having at least one ace by repeat the trial 400 times.

```
> k_card_get_ace(4,400);
```

$$.3125000000$$

The exact probability is $\frac{15225}{52129} = .2812632745$

Find out the probability of drawing a 5-card hand having at least one ace through 200 trials.

```
>  k_card_get_ace(5,200);
                              .3100000000
```

The exact probability is $1 - \frac{\frac{48!}{5!\,43!}}{\frac{52!}{5!\,47!}} = .3411580017$

```
>  k_card_get_ace(5,500);
                              .3480000000
```

A little better approximation. If we combine the 700 trials, the total number of successful trials is

0.31*200+0.348*500=236, so the probability is $\frac{236}{700} = .3371428572$, very good approximation.

```
>  k_card_get_ace(5,1000);
                              .3460000000
```

4.3.4 Example: Drawing a k-card hand having exactly one ace

The main program is shown below with subprograms omitted. The program file "kcard1ac.txt" should include the package "draw_k_card"

```
#
# program that compute the approximate probability of
# drawing a k-card hand having exactly one ace
#
# input:  k --- number of cards in a hand
#         n --- number of trials for the experiment
#
# This program require draw_k_card package as subprogram
#
k_card_get_1_ace:=proc(k::integer,n::integer)
   local count, i, j, hand, num_of_ace;

   count:=0;   # initialize counting
   for i from 1 to n do
      #
      # simulate the experiment
      #
      draw_k_card(k,hand);  # draw a k card hand
      #
      # checking
      #
      num_of_ace:=0;        # initialize the flag=0, meaning no ace
      for j from 1 to k do
         if hand[j][1] = "ace" then
            num_of_ace:=num_of_ace+1;   # count an ace
         fi;
      od;
      if num_of_ace=1 then
         count:=count+1;    # count a successful trial
      fi;
   od;

   evalf(count/n);          # calculate the probability
end;
```

(... draw_k_card package omitted in printing)

```
> read(`a:kcard1ac.txt`):
```

Let's investigate 4-card hands, try 300 and 1000 times.

```
> k_card_get_1_ace(4,300);
                              .2766666667
> k_card_get_1_ace(4,1000);
                              .2480000000
```

The exact probability is $\frac{69184}{270725} = .2555$. If we combine the 1300 trials:

```
> (0.2766666667*300+0.248*1000)/1300;
                              .2546153846
```

Much better.

4.4 More on probability simulations

4.4.1 How accurate is HIV test?

Suppose an HIV test is 99% accurate. The meaning of the accuracy is understood as

- if a person carries HIV virus, he/she has a 0.99 probability of being tested positive and 0.01 probability of being tested negative;
- if a person is does not carry HIV virus he/she has a 0.99 probability of being tested negative and 0.01 probability of being tested positive.

It is estimated that 0.5% of the population carry HIV virus. If there is someone is tested HIV positive, what is the probability that he/she actually carry HIV virus?

You can console this person with the fact that he/she has only about $\frac{1}{3}$ of the chance to be an actual HIV carrier. And you can use a Maple program to prove it to him/her.

The program is to simulate the situation that someone in a certain population is tested positive, and find out if this person is an HIV carrier.

To simulate the situation that someone is tested positive:

1. generate a population that 0.5% of them are HIV carrier and the rest are healthy
2. pick persons one by one to take the test:
   (a) if the person is a carrier, 99% chance positive, 1% chance negative
   (b) if the person is not a carrier, 99% chance negative, 1% chance negative until an HIV positive is found

```
#
#  In a population, 1 out of 200 people carries HIV virus.
#  Suppose that an HIV test is 99\% accurate, meaning for a HIV carrier,
#  he has the probability of 0.99 of testing positive, while a healthy
#  person has 0.99 chance of being tested negative. Estimate the
#  probability of the person who is tested  positive is actually a HIV
#  carrier.
#
#       Input: n --- the number of times to repeat the experiment
#       Output: --- the approximate probability
#
true_hiv:=proc(n)
   local count, i, j, k, u, s, result;

   count:=0;                # initialize the counting of success
   #
   # construct the population with 1/200 sick
   #
   u:=array(1..200,["sick",seq("healthy",i=2..200)]);

   for i to n do            # loop for repeating the experiment

      result:="negative";   # loop of finding a person tested positive
      while result="negative" do
         #
         # pick a person out out the population
         #
         k:=int_ran(1,200);
         #
         # perform the test, which is simulated as a lottery
         #
         if u[k]="sick" then
            s:=int_ran(1,100);      # sick person's lottery:
            if s = 1 then
```

```
               result:="negative"   # 1 out 100 chance of being negative
            else
               result:="positive"
            fi;
         else
            s:=int_ran(1,100);      # healthy person's lottery:
            if s = 1 then
               result:="positive"   # 1 out 100 chance of being positive
            else
               result:="negative"
            fi;
         fi;
      od;
      #
      # check if the positive testee is actually sick
      #
      if u[k]="sick" then
         count:=count+1
      fi;

   od;

   evalf(count/n);

end;
#
#
# function that generates a random integer between m and n, inclusive
#
int_ran:=proc(m::integer,n::integer)
   round(evalf(m-0.5+(n-m+1)*rand()/999999999999))
end;
```

```
> read(`a:true_hiv.txt`):
> true_hiv(500);
```

When someone is tested positive, the probability that he carries HIV is

.2740000000

The exact probability is $\frac{99}{298} = .33221476$

4.4.2 Are you lonesome?

In a group of k people, every one is to pick two others as friends. If a person was not picked by anyone, this person is lonesome. Find the probability that someone is lonesome.

```
# In a group of k people, every one wants to make friend with two
# other people in the group. What is the probability that someone
# is lonesome, meaning he/she is not picked by anyone
```

```
#
#  input:   k  --- the number of people in the group
#           n  --- the number of trials of the experiment
#
lonesome:=proc(  k::integer,
                 n::integer
               )
   local  u, i, j, r, s, flag, count;

   u:=array(1..k);      # open the space for status of each person

   count:=0;            # initialize counting

   for i to n do        # repeat the socializing experiment n times

      for j to k do  # at beginning, everyone is lonesome
         u[k]:="lonesome"
      od;

      for j to k do     # loop for each one to make friends
         #
         # pick the first friend: the person r
         #
         r:=j;
         while r=j do          # no one make friend with himself
            r:=int_ran(1,k);   # person j pick a friend
         od;
         u[r]:="happy";        # person r is picked so no longer lonesome
         #
         # pick the second friend: the person s}
         #
         s:=j;
         while s=j or s=r do   # keep picking until getting a new person
            s:=int_ran(1,k)
         od;
         u[s]:="happy";        # now the person s is no longer lonesome
      od;
      #
      # check if anyone is lonesome after
      #
      flag:=0;
      for j to k while flag=0 do
         if u[j]="lonesome" then
            flag:=1
         fi;
      od;

      if flag=1 then
         count:=count+1
      fi;

   od;

   evalf(count/n);

end;
#
```

```
#
# function that generates a random integer between m and n, inclusive
#
int_ran:=proc(m::integer,n::integer)
   round(evalf(m-0.5+(n-m+1)*rand()/999999999999))
end;
```

```
   >  read(‘a:/txt/lonesome.txt‘):
```

Let try a 5 person group via 200 trials

```
   >  lonesome(5,200);
                              .06000000000
```

Quite low probability

```
   >  lonesome(30,500);
                              .1180000000
```

In a 30-person group, it seems the probability that someone is lonesome is a little higher.

4.4.3 The Monty Hall dillema

———————————————————————————————————-

To Switch or Not to Switch
by Donald Granberg, University of Missouri

In the September 9, 1990, issue of *Parade*, Craig F. Whitaker of Columbia Maryland, poses this query to Marilyn vos Savant (in the "Ask Marilyn" column):

Suppose you are on a game show and you are given the choice of three doors. Behind one door is a car; behind others, goats. You pick a door, say number 1, and the host, who knows what's behind the doors, opens another door, say number 3, which has a goat. He then says to you, "Do you want to switch to door number 2?" Is it to your advantage to switch your choice?

Marilyn's answer was direct and unambiguous,

Yes, you should switch. The first door has a one-third chance of winning, but the second door has a two-thirds chance. Here is a good way to visualize what happend. Suppose there are a million doors, and you pick door number 1. The the host, who knows what's behind the doors and will always avoid the one with the prize, opens them all except door number 777,777. You'd switch to that door pretty fast, wouldn't you?

Despite her explanation, she received a large volume of mail, much of which was from irate and incredulous readers who took issue with her answer. . . . Letters came from great variety of people and places, from people with lofty titles and affiliations and from others of more humble circumstances. For example, Andrew Bremner of the Department of Pure Mathematics at Cambridge University in England, wrote with a touch of noblesse oblige.

Dear Marilyn,

... your answer that you should switch to door number 2 ... is incorrect. Each of doors number 1 and number 2 has 1/2 chance of winning. ... Your correspondents seem rather rude; I wager your womanhood is a factor!

Yours sincerely,

Andrew Bremne

> ... no other statistical puzzle comes so close to fooling all all the people all the time.... The phenomenon is paticularly interesting precisely because of its specificity, its reproducibility, and its immunity to higher education.... Think about it. Ask your brightest friends. Do not tell them though (or at least not yet), that even Nobel physicists systematically give the wrong answer, and that they insist on it, and are ready to berate in print those who propose the right answer.... By Massimo Piattelli-Palmarini in Bostonia (Jul/Aug, 91)

Project: Write a maple program to simulate the game and to approximate the probability of winning by switching. Your simulation should include the following components:

1. randomly assign a car and two goats to door[1], door[2], and door[3].
2. randomly pick a door number k among 1, 2, 3.
3. the host reveals a losing door number m from the two doors other than the door you chose.
4. switch to the remaining door and check if it is the winning door.

Play the game n times and calculate the approximate probability of winning by switching.

4.5 Exercises

1. **Estimating π** A circle of radius 1 is drawn inside a 2×2 square. If one throws an object into the square, the probability of landing inside the circle is $\frac{\pi}{4}$. If one repeats this experiment n times, he can estimate this probability and the estimated probability times 4 should be close to π. This is an easy (but inefficient) algorithm of estimating π. Write a program, using this algorithm, to estimate π for input n, which is the number of trials of the experiment.

2. **Quality control** When shipping diesel engines abroad, it is common to pack 12 engines in one container that is then loaded on a rail car and sent to a port. Suppose that a company has received complaints from its customers that many of the engines arrive in nonworking condition. The company thereby makes a spot check of containers after loading. The company will test three engines from a container randomly. If any of the three are nonworking, the container will not be shipped. Suppose a given container has 2 nonworking engines. Use a Maple program to estimate the probability that the container will not be shipped.

3. **A k-card hand flush**

   Write a program to estimate the probability of drawing a k-card hand having the same suit (called a *flush*. The exact probability is $4\,\frac{13!\,(52-k)!}{(13-k)!\,52!}$ The exact probabilities for $k = 2$ and $k = 3$ are $\frac{4}{17}$ and $\frac{22}{425}$ respectively

4. **Problem 4: Drawing balls**

   A box contains 4 white balls and 8 red balls. Write a program that estimates the probability of drawing $k$balls without replacement and getting all red.

5. **How lucky are you if you are tested HIV negative?**

   Use the HIV example above, find the probability that, if a person is tested HIV negative, the person is indeed healthy.

6. **Are you lonesome again?**

   There are a 15 boys and 10 girls. Every boy will date a girl. However, there might be several boys who want to date a same girl. Find the probability that at least one girl is lonesome.

7. **Friend match**

   There are k people in a group and you are one of them, say the person #1. Every one is to make friends with two other people. If you pick a friend, who also pick you as a friend, then you two matches. Find the probability that you can find your match.

8. **Two boys** In an issue of *Parade*, a question was debated in the column *Ask Marilyn*: **If a family has exactly two kids and at least one of them is a boy, what is the probability that both kids are boys?** Marilyn vos Savant, the columnist, gave the answer 1/3 but many readers didn't agree. One of the readers thought the answer should have been 1/2 and challenged Marilyn with $ 1000 bet. Write a program to simulate the experiment and settle the bet.

   The simulation part of the program should consist of the following components:

   (a) generate two "kids", in an array with entries either "boy" or "girl", each kid has a 50/50 chance of being a boy or a girl.

   (b) repeat step (a) until at least one of the two kids is a boy.

   To assign a gender to each kid, you can use the random integer generator from 1 to 2 and designate them as boy and girl respectively.

Chapter 5

Simple Systems of equations

5.1 Solving equations

5.1.1 Maple commands

The Maple function *solve* is used for solving equations. Ask Maple by ?solve for details.

For example, to solve a single equation, the syntax is

> solve(equation,variable);

For example, to solve $5\,x + \pi = 9$:

```
> solve(5*x+Pi=9,x);
```

$$-\frac{1}{5}\,\pi + \frac{9}{5}$$

Or, preferablly:

```
> eqn:=5*x+Pi=9:          variable:=x:
> solve(eqn,variable);
```

$$-\frac{1}{5}\,\pi + \frac{9}{5}$$

To solve a system of equations, the syntax is

> solve({equations}, {variables})

For example, to solve

$$\begin{cases} 3x + 6y = 5 \\ 4x - 5y = 3 \end{cases}$$

```
> equations:={ 3*x+6*y=5,4*x-5*y=3};
```

$$equations := \{3\,x + 6\,y = 5,\, 4\,x - 5\,y = 3\}$$

```
> variables:={x,y};
```

$$variables := \{x,\, y\}$$

```
> solve(equations,variables);
```

$$\{y = \frac{11}{39},\, x = \frac{43}{39}\}$$

However, before you can manipulate the solutions, you have to *assign* it:

```
> equations:={a*x+b*y+c*z=d, e*x+f*y+g*z=h, i*x+j*y+k*z=l};
```

$$equations := \{a\,x + b\,y + c\,z = d,\, e\,x + f\,y + g\,z = h,\, i\,x + j\,y + k\,z = l\}$$

```
> variables:={x,y,z};
```

$$variables := \{x,\, y,\, z\}$$

```
> solutions:=solve(equations,variables);
```

$$solutions := \{z = \frac{-a\,j\,h + a\,l\,f - b\,e\,l + b\,i\,h + d\,e\,j - d\,i\,f}{\%1},$$
$$y = \frac{i\,g\,d - i\,h\,c - e\,k\,d + e\,l\,c - g\,a\,l + h\,a\,k}{\%1},$$
$$x = -\frac{j\,g\,d - j\,h\,c + k\,b\,h - k\,d\,f - l\,b\,g + l\,c\,f}{\%1}\}$$
$$\%1 := -a\,j\,g + a\,k\,f - b\,e\,k + b\,i\,g + c\,e\,j - c\,i\,f$$

```
> assign(solutions);
> x; y; z;
```

$$-\frac{j\,g\,d - j\,h\,c + k\,b\,h - k\,d\,f - l\,b\,g + l\,c\,f}{-a\,j\,g + a\,k\,f - b\,e\,k + b\,i\,g + c\,e\,j - c\,i\,f}$$
$$\frac{i\,g\,d - i\,h\,c - e\,k\,d + e\,l\,c - g\,a\,l + h\,a\,k}{-a\,j\,g + a\,k\,f - b\,e\,k + b\,i\,g + c\,e\,j - c\,i\,f}$$
$$\frac{-a\,j\,h + a\,l\,f - b\,e\,l + b\,i\,h + d\,e\,j - d\,i\,f}{-a\,j\,g + a\,k\,f - b\,e\,k + b\,i\,g + c\,e\,j - c\,i\,f}$$

5.1.2 Example: Percent mixture problem

A chemist mixes an 11% acid solution with 4% acid solution. How many milliliters of each solution should the chemist use to make a 700-milliliter solution of 6%?

Generally, if one wants to mix two solutions with rates of concentration $a\%$ and $b\%$ respectively, to make a mixed solution of $c\%$ and of amound s. Let x and y be the amounts for the two solutions used. The equations to solve is:

$$\begin{cases} x + y = s \\ ax + by = cs \end{cases}$$

We can make a program to find the amount of each solution used

```
#
# program to find the amounts of two chemical solutions, with rates of
# concentration a% and b% respectively, required to make a new solution,
# with amount s and rate of concertration c
#
#  input:     a, b, c, s  --- described above
#  output:    x           --- the amount of a% solution required
#             y           --- the amount of b% solution required
#
mix_2_solutions:=proc(   a::numeric,
                         b::numeric,
                         c::numeric,
                         s::numeric,
                         x::evaln,
                         y::evaln
                       )
   local equations, variables, solutions;

   x:='x';  y:='y';    # clear posible value in x, y

   equations:={ x + y = s,  a*x+b*y=c*s  };    # define equations
   variables:={ x, y };                        # define variables
   solutions:=solve( equations, variables );   # solve equations
   assign( solutions );                        # assign solutions to x, y

end;
```

```
> read(`a:/txt/mix2slns.txt`):

> mix_2_solutions(11,4,6,700,x,y);

> x; y;
                              200
                              500
```

That is, 200 milliliters of 11% solution and 500 milliliters of 4% solution are required to make 700 milliliters of 6% solution.

5.1.3 Line fitting

For a given set of data, say

```
> x:=array(1..5,[1,2,3,4,5]);
```

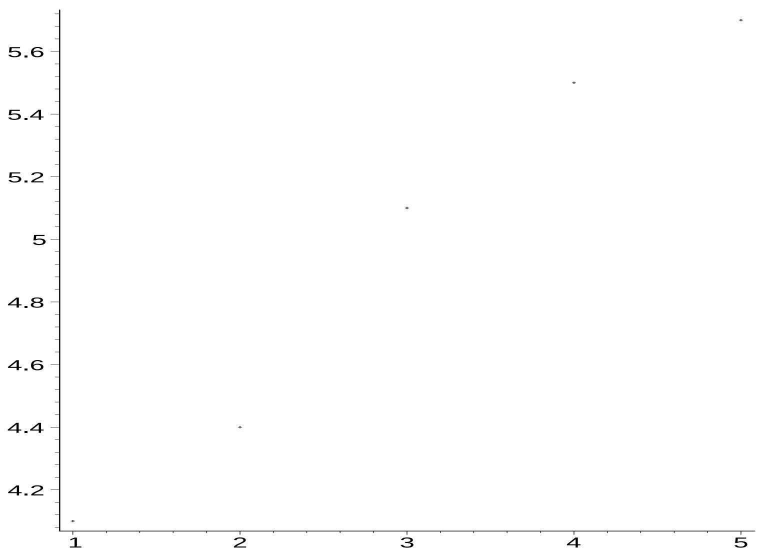
$$x := [1, 2, 3, 4, 5]$$

```
> y:=array(1..5,[4.1,4.4,5.1,5.5,5.7]);
```

$$y := [4.1, 4.4, 5.1, 5.5, 5.7]$$

The graph shows the data is close to a line:

```
> points:=[  seq( [x[i],y[i]], i=1..5  )  ]:
> plot(points,style=point);
```



The question is, which line fit the data the best?

In other words, the data represent an unknown function $y = \mathrm{g}(x)$. Because there is apparently no way we can get the formula of the function, we set a linear model

$$y = a\,x + b$$

and try to find a and b.

A line can be described as an equation $y = a\,x + b$, where the number a is the slope and the number b is the y-intercept. At every point x_i, $i = 1, \cdots, 5$, the difference between the data and the line is

$$y_i - (a\,x_i + b), \quad i = 1, 2, 3, 4, 5$$

We may define the "best fit" as

$$\sum_{i=1}^{5} [y_i - (a\,x_i + b)]^2$$

to be minimal. Generally, for data x_i, y_i, $i = m, \cdots, n$, We try to find a and b such that the function

$$\mathrm{f}(a,\, b) = \sum_{i=m}^{n} [y_i - (a\,x_i + b)]^2$$

is minimized.

Recall that the necessary condition for a function to be minimized (or maxmized) is that all the first derivatives are zero. The function $\mathrm{f}(a,\, b)$ is a two variable function. To reach the minimum, both $\frac{\partial}{\partial a} f$ and $\frac{\partial}{\partial b} f$ must be zero.

$$\frac{\partial}{\partial a} f = 0 \;\text{ leads to }\; -2\left(\sum_{i=m}^{n} [y_i - (a\,x_i + b)]\,x_i\right) = 0$$

$$\frac{\partial}{\partial b} f = 0 \;\text{ leads to }\; -2\left(\sum_{i=m}^{n} [y_i - (a\,x_i + b)]\right) = 0$$

After simplification

$$\left(\sum_{i=m}^{n} {x_i}^2\right) a + \left(\sum_{i=m}^{n} x_i\right) b = \sum_{i=m}^{n} x_i\, y_i \tag{5.1}$$

$$\left(\sum_{i=m}^{n} x_i\right) a + (n - m + 1)\, b = \sum_{i=m}^{n} y_i \tag{5.2}$$

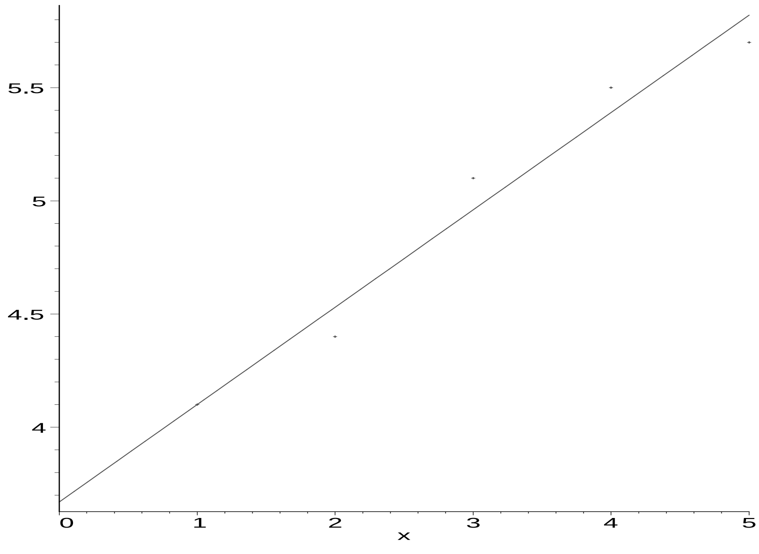
By solving this system of equations for a and b, we can find the line best fit the data.

```
> read(`a:line_fit.txt`):
> line_fit(1,5,x,y,a,b);
> a; b;
```

```
                                .4300000000
                                3.670000000
```

That is, the line best fit the data above is $y = .43\,x + 3.67$. We may observe it from the graph:

```
> plot1:=plot(points,style=point):
> plot2:=plot(a*x+b,x=0..5,style=line):
> plots[display]({plot1,plot2});
```



If we want to estimate the value of the unknown function at $x = 2.5$ the best we can do is

$$y = a(2.5) + b = 4.745$$

5.1.4 Project

Write a program, for given input m, n, x, and y, that solves equations (5.1) and (5.2) above and output a and b as argument of the program. Duplicate the results and graphs above and use your program to find the line best fit the following data (show similar graph)

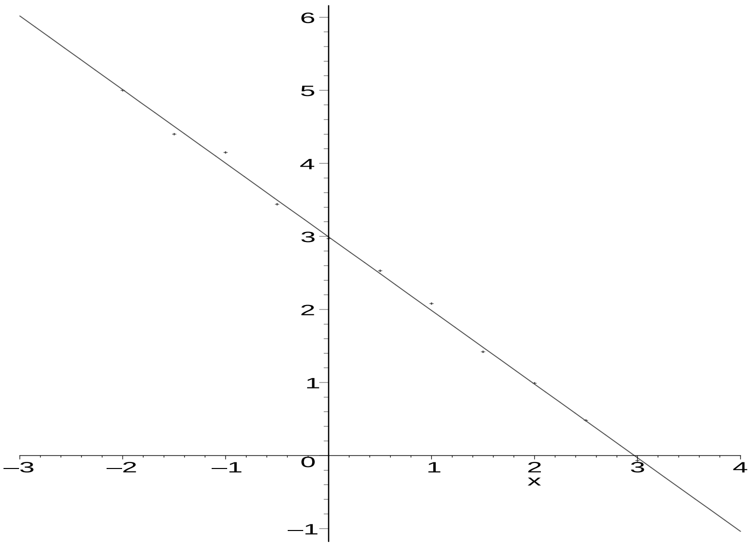
```
> x:=array(0..10,[-2,-1.5,-1.0,-0.5,0.0,0.5,1.0,1.5,2.0,2.5,3.0]):
```

```
> y:=array(0..10,[5.0,4.4,4.15,3.44,2.97,2.53,2.08,

> 1.42,0.99,0.48,-0.07]):
> line_fit(0,10,x,y,s,t);
> s; t;
```

$$-1.008000000$$
$$2.994000000$$

Your graph should look like the following.

```
> plots[display]({p1,p2});
```



5.2 Leontief's model of economy

5.2.1 A three sector example

Wassily Leontief explained his input-output system of economy in the April 1965 issue of *Scientific American*, using 1958 American economy as an example. He divided the economy into 81 sectors. To keep explanation simpler, lets put 1947 American economy into three sectors: agriculture, manufacturing, and the household (i.e., the sector produces labor). American economy in 1947 are described in the *input-output table*

	Agriculture	Manufacturing	Household
Agriculture	.245	.102	.051
Manufacturing	.099	.291	.279
Household	.433	.372	.011

The first column is the Agriculture's demand from all sectors. For example, every \$1,000 agricultural output require \$245 from agriculture, \$99 from manufacturing, \$433 from household. Similarly \$1,000,000 of household output require \$51,000, \$279,000 and \$11,000 from Agriculture, Manufacturing, and Household respectively.

Generally, to produce \$ x_1 of agriculture, \$ x_2 of manufacturing, and \$ x_3 of household output, the required input items are

$$\begin{array}{ll} .245\,x_1 + .102\,x_2 + .051\,x_3 & \text{from agriculture} \\ .099\,x_1 + .291\,x_2 + .279\,x_3 & \text{from manufacturing} \\ .433\,x_1 + .372\,x_2 + .011\,x_3 & \text{from household} \end{array}$$

The *bill of demand* from the society (outside the three sectors) in 1947 was

Agriculture:	\$	2.88	billion
Manufacturing:	\$	31.45	billion
Household:	\$	30.91	billion

What was the output from the three sectors are required to meet the demand?

So suppose x_1, x_2, x_3 are the output from those three sectors respectively (in billions of dollars). The agricultural output x_1 is the combination of outside demand 2.88 and internal demand $.245\,x_1 + .102\,x_2 + .051\,x_3$. That is,

$$x_1 = 2.88 + .245\,x_1 + .102\,x_2 + .051\,x_3$$

similarly, we have equations that determine the required output:

$$\left\{ \begin{array}{rcr} x_1 & = & 2.88 + .245\,x_1 + .102\,x_2 + .051\,x_3 \\ x_2 & = & 31.45 + .099\,x_1 + .291\,x_2 + .279\,x_3 \\ x_3 & = & 30.91 + .433\,x_1 + .372\,x_2 + .011\,x_3 \end{array} \right.$$

By solving this equation, we can obtain the required output from each sector.

This is a three sector economy model. The input-output table is a 3x3 matrix:

```
>  T:=matrix(3,3,[.245,.102,.051,.099,.291,.279,.433,.372,.011]);
```

$$T := \begin{bmatrix} .245 & .102 & .051 \\ .099 & .291 & .279 \\ .433 & .372 & .011 \end{bmatrix}$$

The entries of T are identified as T[i,j] (the entry at the i-th row and j-th column). The bill of demand is a vector

```
> d:=vector([2.88,31.45,30.91]);
```

$$d := [2.88, 31.45, 30.91]$$

The equations that determine the required output can be written as

$$x_1 = d_1 + T_{1,1}\,x_1 + T_{1,2}\,x_2 + T_{1,3}\,x_3 = d_1 + \left(\sum_{j=1}^{3} T_{1,j}\,x_j\right)$$

$$x_2 = d_2 + T_{2,1}\,x_1 + T_{2,2}\,x_2 + T_{2,3}\,x_3 = d_2 + \left(\sum_{j=1}^{3} T_{2,j}\,x_j\right)$$

$$x_3 = d_3 + T_{3,1}\,x_1 + T_{3,2}\,x_2 + T_{3,3}\,x_3 = d_3 + \left(\sum_{j=1}^{3} T_{3,j}\,x_j\right)$$

Or, even simpler:

$$x_i = d_i + \left(\sum_{j=1}^{3} T_{i,j}\,x_j\right), i = 1, 2, 3.$$

We thus solve the problem:

```
> x:=array(1..3);
```

$$x := \text{array}(1..3, [])$$

```
> eqn:={ seq(x[i]=d[i]+sum(T[i,j]*x[j],j=1..3), i=1..3 ) };
```

$$\begin{aligned} eqn := \{&x_1 = 2.88 + .245\,x_1 + .102\,x_2 + .051\,x_3, \\ &x_2 = 31.45 + .099\,x_1 + .291\,x_2 + .279\,x_3, \\ &x_3 = 30.91 + .433\,x_1 + .372\,x_2 + .011\,x_3\} \end{aligned}$$

```
> var:={  seq(  x[i], i=1..3    )  };
```

$$var := \{x_1,\, x_2,\, x_3\}$$

```
> solutions:=solve( eqn, var );
```

$$solutions := \{x_1 = 18.20792271,\ x_2 = 73.16603495,\ x_3 = 66.74600155\}$$

```
> assign(solutions);
> eval(x);
```

$$[18.20792271,\ 73.16603495,\ 66.74600155]$$

That is, in 1947, \$18.2 billions of agriculture, \$73.17 billions of manufacturing, and \$66.75 billions of household outputs were required to meet the bill of demand.

5.2.2 The general n-sector case

Suppose the economy is divided into n sectors. The input-output table is a matrix:

	Sector 1	Sector 2	Sector 3	$\cdots$	Sector n
Sector 1	$T_{1,1}$	$T_{1,2}$	$T_{1,3}$	$\cdots$	$T_{1,n}$
Sector 2	$T_{2,1}$	$T_{2,2}$	$T_{2,3}$	$\cdots$	$T_{2,n}$
Sector 3	$T_{3,1}$	$T_{3,2}$	$T_{3,3}$	$\cdots$	$T_{3,n}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
Sector n	$T_{n,1}$	$T_{n,2}$	$T_{n,3}$	$\cdots$	$T_{n,n}$

Let the bill of demand be

Sector 1	d_1
Sector 2	d_2
Sector 3	d_3
$\cdots$	$\cdots$
Sector n	d_n

Let the required output from sector i be x_i, $i = 1, 2, \cdots, n$. Then the required output x_i from Sector i is a combination of external demand and internal demand:

output	=	external demand	+	internal demand
x_i	$=$	d_i	$+$	$T_{i,1}\,x_1 + T_{i,2}\,x_2 + \cdots + T_{i,n}\,x_n$
	$=$	d_i	$+$	$\sum_{j=1}^{n} T_{i,j}\,x_j$

$$i = 1, 2, 3, \cdots, n$$

5.3 Using linear algebra package

5.3.1 Matrices and vectors

The system of linear equation

$$\begin{cases} 3\,x_1 + 6\,x_2 = 5 \\ 4\,x_1 - 3\,x_2 = 2 \end{cases}$$

can be written in matrix/vector form

$$\begin{bmatrix} 3 & 6 \\ 4 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

or

$$A\,x = b$$

with

$$A = \begin{bmatrix} 3 & 6 \\ 4 & -3 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad b = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

Here, A is called a 2×2 (read 2 by 2) matrix, x and b are called 2-vectors.

Maple accepts matrices and vectors, For example,

```
> B:=matrix(3,2,[1,2,3,4,5,6]);
```

$$B := \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
> d:=vector([4,5,2]);
```

$$d := [4,\ 5,\ 2]$$

B is a 3×2 matrix and d a 3-vector. The syntax of entering a matrix is

[> matrix(m,n,[· · ·]);

where

m	—	number of rows
n	—	number of columns
[· · ·]	—	entries of the matrix, row by row.

A matrix can also be entered entry by entry:

```
> A:=matrix(2,2);
```

$$A := \text{array}(1..2,\ 1..2,\ [])$$

```
> A[1,1]:=3; A[1,2]:=6; A[2,1]:=4; A[2,2]:=-3;
```

$$A_{1,1} := 3$$
$$A_{1,2} := 6$$
$$A_{2,1} := 4$$
$$A_{2,2} := -3$$

```
> b:=vector(2);
```

$$b := \text{array}(1..2,\ [])$$

```
> b[1]:=5; b[2]:=2;
```

$$b_1 := 5$$
$$b_2 := 2$$

```
> evalm(A), evalm(b);
```

$$\begin{bmatrix} 3 & 6 \\ 4 & -3 \end{bmatrix}, [5, 2]$$

The command "evalm" is used to show matrices and vectors.

5.3.2 Linear algebral package

Maple has a collection of commands that perform linear algebra operations. This collection can be loaded into maple in the following way:

```
> with(linalg);

Warning, new definition for norm

Warning, new definition for trace
```

[*BlockDiagonal*, *GramSchmidt*, *JordanBlock*, *LUdecomp*, *QRdecomp*, *Wronskian*, *addcol*, *addrow*, *adj*, *adjoint*, *angle*, *augment*, *backsub*, *band*, *basis*, *bezout*, *blockmatrix*, *charmat*, *charpoly*, *cholesky*, *col*, *coldim*, *colspace*, *colspan*, *companion*, *concat*, *cond*, *copyinto*, *crossprod*, *curl*, *definite*, *delcols*, *delrows*, *det*, *diag*, *diverge*, *dotprod*, *eigenvals*, *eigenvalues*, *eigenvectors*, *eigenvects*, *entermatrix*, *equal*, *exponential*, *extend*, *ffgausselim*, *fibonacci*, *forwardsub*, *frobenius*, *gausselim*, *gaussjord*, *geneqns*, *genmatrix*, *grad*, *hadamard*, *hermite*, *hessian*, *hilbert*, *htranspose*, *ihermite*, *indexfunc*, *innerprod*, *intbasis*, *inverse*, *ismith*, *issimilar*, *iszero*, *jacobian*, *jordan*, *kernel*, *laplacian*, *leastsqrs*, *linsolve*, *matadd*, *matrix*, *minor*, *minpoly*, *mulcol*, *mulrow*, *multiply*, *norm*, *normalize*, *nullspace*, *orthog*, *permanent*, *pivot*, *potential*, *randmatrix*, *randvector*, *rank*, *ratform*, *row*, *rowdim*, *rowspace*, *rowspan*, *rref*, *scalarmul*, *singularvals*, *smith*, *stackmatrix*, *submatrix*, *subvector*, *sumbasis*, *swapcol*, *swaprow*, *sylvester*, *toeplitz*, *trace*, *transpose*, *vandermonde*, *vecpotent*, *vectdim*, *vector*, *wronskian*]

The list above shows all the linear algebra commands. As ususal you can use the question mark "?" to get the explanation of each command. Once the package is loaded, all the commands above can be accessed as standard Maple commands.

For example, to solve the linear system

$$\begin{bmatrix} 3 & 6 \\ 4 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

we can use the command "linsolve"

```
>  A:=matrix(2,2,[3,6,4,-3]); b:=vector([5,2]);
```

$$A := \begin{bmatrix} 3 & 6 \\ 4 & -3 \end{bmatrix}$$

$$b := [5,\ 2]$$

```
>  x:=linsolve(A,b);
```

$$x := \left[\frac{9}{11},\ \frac{14}{33}\right]$$

5.3.3 The least squares problem.

If a linear system has more equations than variables, it has no conventional solution except in unusual cases. In terms of matrices/vectors, let A be an $m \times n$ matrix with $m > n$ then the equation

$$A\,x = b$$

has no (conventional) solution in general. In this case, we are looking for vector x such that the norm of $A\,x - b$ to be minimized. This vector x is called a *least squares solution* to the equation $A\,x - b = 0$

For example, the line fitting problem

```
>  x:=vector([1,2,3,4,5]);
```

$$x := [1,\ 2,\ 3,\ 4,\ 5]$$

```
>  y:=vector([4.1,4.4,5.1,5.5,5.7]);
```

$$y := [4.1,\ 4.4,\ 5.1,\ 5.5,\ 5.7]$$

we are looking for a and b

$$a\,x_i + b = y_i, \quad \text{for} \quad i = 1,\ 2,\ 3,\ 4,\ 5$$

Namely, we have 5 equations and 2 unknows:

$$\begin{cases} a + b = 4.1 \\ 2\,a + b = 4.4 \\ 3\,a + b = 5.1 \\ 4\,a + b = 5.5 \\ 5\,a + b = 5.7 \end{cases}$$

or

$$\begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 5 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 4.1 \\ 4.4 \\ 5.1 \\ 5.5 \\ 5.7 \end{bmatrix}$$

```
> A:=matrix(5,2,[1,1,2,1,3,1,4,1,5,1]);
```

$$A := \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 5 & 1 \end{bmatrix}$$

```
> d:=vector([4.1,4.4,5.1,5.5,5.7]);
```

$$d := [4.1,\ 4.4,\ 5.1,\ 5.5,\ 5.7]$$

```
> u:=leastsqrs(A,d);
```

$$u := [.430000002,\ 3.669999993]$$

That is, $a = .43$, $b = 3.67$, or

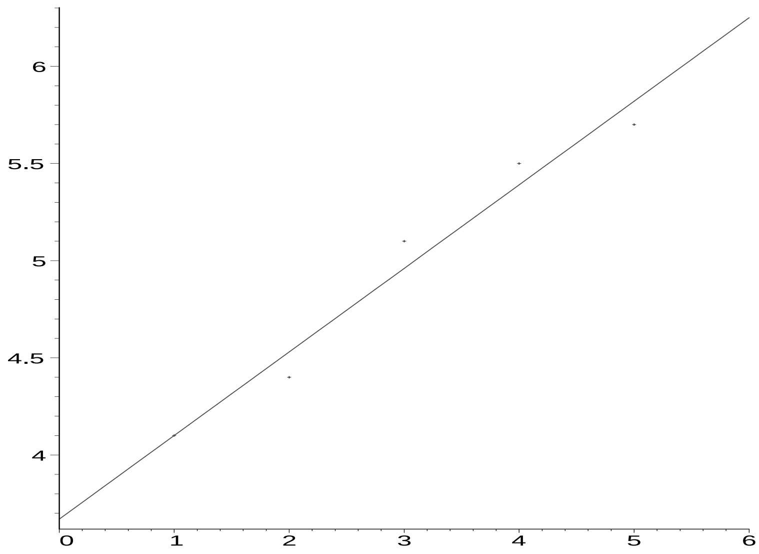
$$y = .43\,x + 3.67$$

We can plot the points $(x_i,\ y_i)$ and the line $y = .43\,x + 3.67$ and visualize the solution:

```
> points:=[seq([x[i],y[i]],i=1..5)];
```

$$points := [[1,\ 4.1],\ [2,\ 4.4],\ [3,\ 5.1],\ [4,\ 5.5],\ [5,\ 5.7]]$$

```
> plot1:=plot(points,style=point):
> plot2:=plot(.43*t+3.67,t=0..6):
> plots[display]({plot1,plot2});
```



Example: Market research: A company want to predict sales b; according to population x; and per capita income y;. The model is

$$b = c_1 + c_2\, x + c_3\, y$$

The investigation in 5 towns yields the following data:

b	x (1000)	y ($)
162	274	24500
120	180	32540
223	375	38020
131	205	28380
67	86	23470

According to the data, we have the following linear equations:

$$\left\{\begin{array}{rcl} 162 & = & c_1 + 274\, c_2 + 24500\, c_3 \\ 120 & = & c_1 + 180\, c_2 + 32500\, c_3 \\ 223 & = & c_1 + 375\, c_2 + 38020\, c_3 \\ 131 & = & c_1 + 205\, c_2 + 28380\, c_3 \\ 67 & = & c_1 + 86\, c_2 + 23470\, c_3 \end{array}\right.$$

Or, in matrix/vector form $A\,x = b$;

$$\left[\begin{array}{ccc} 1 & 274 & 24500 \\ 1 & 180 & 32500 \\ 1 & 375 & 38020 \\ 1 & 205 & 28380 \\ 1 & 86 & 23470 \end{array}\right] \left[\begin{array}{c} c_1 \\ c_2 \\ c_3 \end{array}\right] = \left[\begin{array}{r} 162 \\ 120 \\ 223 \\ 131 \\ 67 \end{array}\right]$$

We can solve the problem using Maple linear algebra package

```
> A:=matrix(5,3,[1,274,24500, 1,180,32540, 1,375,38020,

> 1,205,28380,1,86,23470]);
```

$$A := \left[\begin{array}{ccc} 1 & 274 & 24500 \\ 1 & 180 & 32540 \\ 1 & 375 & 38020 \\ 1 & 205 & 28380 \\ 1 & 86 & 23470 \end{array}\right]$$

```
> b:=vector(5,[162.0,120,223,131,67]);
```

$$b := [162.0, 120, 223, 131, 67]$$

```
> v:=leastsqrs(A,b);
```

$$v := [7.032503419, .5044475947, .000700130535]$$

```
> c[1]:=v[1]; c[2]:=v[2]; c[3]:=v[3];
```

$$c_1 := 7.032503419$$
$$c_2 := .5044475947$$
$$c_3 := .000700130535$$

If a town has a population of 500,000 and per capita income of $20000, then the expected sale will be

```
> c[1]+c[2]*500+c[3]*20000;
                                273.2589115
```

5.4 Exercises

1. **Price and demand** An owner of a restauant wants to estimate the daily demand of her steak at price $9.50/dish. She has a record of price-demand of the past:

Price:	7.50	7.75	8.00	8.50	9.00
demand (daily)	36	34	31	25	20

   Use the line fitting program to answer this question.

2. **Population estimate** Population growth is usually modelled as an exponential function:

$$p = b\, e^{(a\,t)}$$

   or

$$\ln(p) = \ln b + a\,t \qquad (5.3)$$

   For the following population statistics of Mexico, find the exponential function (i.e. find a, and b) that best fit the data

year (after 1980)	0	1	2	3	4	5	6
population (millions)	67.4	69.1	71	72.8	74.7	76.6	78.6

   (a) Plot the population statistics and the exponential function on the same graph to show that the exponential model fit the data.

   (b) Use your exponential function to estimate the population of Mexico in 1990.

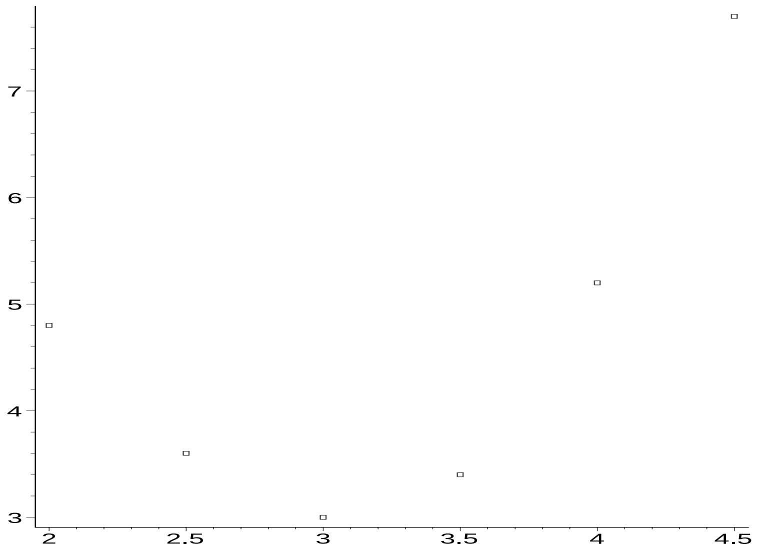
   Hint: Don't write a new program, your existing line fit program works for this problem because equation (5.3) is a linear equation.

3. **Quadratic fitting** The following data

x:	2.0	2.5	3.0	3.5	4.0	4.5
y:	4.8	3.6	3.0	3.4	5.2	7.7

should be close to a parabola (see the graph)

```
> x:=array(1..6,[2.0,2.5,3.0,3.5,4.0,4.5]):
> y:=array(1..6,[4.8,3.6,3.0,3.4,5.2,7.7]):
> points:=[ seq( [x[i],y[i]], i=1..6  ) ]:
> plot(points,style=point,symbol=box);
```



We can use a quadratic model to fit these data. That is, assuming

$$y = a\,x^2 + b\,x + c$$

and set the total variation

$$\mathrm{g}(a,\, b,\, c) = \sum_{i=m}^{n} [y_i - (a\,{x_i}^2 + b\,x_i + c)]^2$$

to be minimized.

Theoretical question: the coeficients a, b, and c satisfy a system of three equations similar to the equations (5.1) and (5.2). Find those equations which determine a, b, and c.

Computational question: Write a program that determines am, n, x, and y.

4. **Programming Leontief's model of economy**

   Write a program to calculate the required output from all n sectors for given number of sectors n, input-output matrix T, and bill of demand array d. Your program should work like

```
> read(`a:leontief.txt`):
> Leontief(3,T,d,x);
> eval(x);
```

$$[18.20792271, 73.16603495, 66.74600155]$$

5. **A 6-sector model** American economy in 1958 can be simplified into 6 sectors:

Final nonmetal	(FN):	Furniture, processed food, etc
Final metal	(FM):	Household appliances, cars, etc
Basic metal	(BM):	mining, steel, etc
Basic nonmetal	(BN):	Agriculture, printing, etc
Energy	(E):	Coal, electricity, etc
Services	(S):	Amusements, real estate, etc

The input-output table and the bill of demand for the 6-sector economy were

	FN	FM	BM	BN	E	S	Bill of demand
FN	.170	.004	0	.029	0	.008	99,640
FM	.003	.295	.018	.002	.004	.016	75,548
BM	.025	.173	.460	.007	.011	.007	14,444
BN	.348	.037	.021	.403	.011	.048	33,502
E	.007	.011	.039	.025	.358	.025	23,527
S	.120	.074	.014	.123	.173	.234	263,985
							\$million

Calculate the required output from each sector.

6. **Daylight hours** Let $\mathrm{S}(t)$ be the number of daylight hours on the t-th day of the year 1997, in Rome, Italy. We are given the following data for $\mathrm{S}(t)$:

Day	January 28	March 17	May 3	June 16
t	28	77	124	168
$\mathrm{S}(t)$	10	12	14	15

We wish to fit a trigonometri function of the form

$$\mathrm{f}(t) = a + b\sin\left(\frac{2\,\pi\,t}{365}\right) + c\cos\left(\frac{2\,\pi\,t}{365}\right)$$

to the data.

(a) Find the best approximation to a, b and c

(b) Plot both the points and $\mathrm{f}(t)$ in a graph.

7. **Stretching a metal strip** A strip of experimental metallic alloy is being tested. It is stretched to lengths $\lambda_1 = 63$, 68 and 72 inches by applied weights of $\omega = 1$, 2 and 3 tons. Assuming Hooke's Law $\lambda = h{+}c\,\omega$, find this metal strip's normal length h and the elasticity constant c. Plot the graph of the function and data points in the same $\lambda - \omega$

8. **Radioactive materials** Two radioactive materials in amounts x_1 and x_2 are being contained. We know the half-lives $\lambda_1 = 84$ years and $\lambda_2 = 115$ years, respectively for each, but not the actual amounts of each present. We take radiation readings ρ and anticipate that these readings will behave as the sum of two exponantials:

$$\rho = x_1\, e^{(-\mu_1\, t)} + x_2\, e^{(-\mu_2\, t)}$$

where $\mu_1 = \dfrac{\ln(2)}{\lambda_1}$ and $\mu_2 = \dfrac{\ln(2)}{\lambda_2}$ The actual readings for 7 time periods, given as (t, ρ), are as follows: (0,12.2), (1,12.11), (2,12.02), (3,11.93), (4,11.84), (5,11.75), (6,11.66). Find the amounts x_1 and x_2 by the least squares method. Plot the points and the function ρ = f(t) together.

9. **Polynomial fit** Find the cubic polynomial

$$y = a\,x^3 + b\,x^2 + c\,x + d$$

that best fits the data points in the form of (x, y): (1,-2.4), (2,-1.2), (3,0.4), (4,1.7), (5,2.4), (6,2). Plot the points and the polynomial in the same graph.

Chapter 6

Ordinary differential equations

6.1 The initial value problem of ordinary differential equations

6.1.1 Example: Population growth

Suppose *now* a city has an population of 25 thousands. It is estimated that for the next 5 years, every year the birth rate is 1.8% and death rate is 0.6%. Also there are 500 people moving in and 200 people moving out every year. How to estimate the future population based on those data, say 3 years and 9 mouths from now?

Solution: The unknown future population p is a function of time. So we set the function be $p(t)$, where the time variable t is the number of years from now. Clearly $p(0) = 25$ (thousands).

The growth of population is due to (1) natural growth, and (2) imigration.

The natural growth (birth-death) percentage rate is 1.8%-0.6%=1.2%. So every year, the number of people increased due to natural growth is 1.2% *of the population*, i.e. $.012\,p$. the net imigration is 500-200=300 = 0.3 thousands. Thus, the annual growth rate is

$$.012\,p + .3 \quad \text{thousand/year}$$

On the other hand, the population growth per year is the derivative of population function with respect to time, i.e., $\frac{dp}{dt}$. Thus we have an ordinary differential equation (ODE).

$$\frac{dp}{dt} = .012\,p + .3 \quad \text{(in thousands of people)}$$

Considering $p(0) = 25$ (initial condition), we have an initial value problem of ODE:

$$\frac{dp}{dt} = .012\,p + .3, \qquad t \in [0,\ 5]$$

$$p(0) = 25$$

6.1.2 Using Maple ODE solver

```
> dsolve( diff(p(t),t) = 0.012*p(t)+0.3, p(t));
```

$$\mathrm{p}(t) = -25. + e^{(.01200000000\,t)}\ \_C1$$

Namely, the ODE $\frac{dp}{dt} = .012\,p + .3$ has a general solution as above. $\_C1$ represents a constant to be determined. Using the fact $p(0) = 25$, we have

$$-25 + \_C1 = 25$$

That is, the constant $\_C1 = 50$. We now have the complete solution

$$p(t) = -25 + 50\,e^{(.012\,t)}$$

If we want to estimate the population 3 years 9 months from now, we have

```
> p:=t->-25+50*exp(0.012*t);
```

$$p := t \to -25 + 50\,e^{(.012\,t)}$$

```
> p(3.75);
```

$$27.30139300$$

That is, the population is expected to be 27,301 after 3 years and 9 months.

Maple command "dsolve" has many more features. Readers may use "?dsolve" for more information.

6.1.3 The general initial value problem of ODE

Generally for a given function $f(t, x)$, an interval $[a, b]$, and number x_0, the following is called an initial value problem of ordinary diffential equation:

$$\begin{cases} \dfrac{dx}{dt} = f(t, x), & t \in [a, b] \\ x(a) = x_0 \end{cases}$$

Finding the unknown function $x = x(t)$ is the objective of solving the problem.

6.1.4 Numerical solution

Not every differential equation can be solved theoretically. For example

```
> dsolve(diff(x(t),t)=cos(t)+exp(sin(x(t))), x(t));
```

Maple gave no response for that call.

We thereby need to develop numerical methods. That is, instead of searching for the formula of the function $x(t)$ as the solution, we look for a table of the value of the function. For example, if we can't get the population function in the form of $p(t) = -25 + 50\,e^{(.012\,t)}$, we may want the values of the function in a table like:

t	0	0.25	0.5	0.75	1.00	1.25	1.50	1.75	...
$x(t)$	25	25.15	25.30	25.45	25.60	25.75	25.91	26.06	...

This table of values is called a numerical solution to the initial value problem.

6.1.5 Euler method

The simpliest method of obtaining numerical solution is Euler's method:

(1) Determine n, the number of subintervals we want to divide the interval $[a, b]$.

(2) By dividing $[a, b]$ into n pieces, we have the points on the t-axis: $a = t_0 < t_1 < t_2 < \cdots < t_n = b$, where

$$t_i = a + ih, \quad \text{with} \quad h = \frac{b-a}{n}, \quad i = 0, 1, 2, \cdots n$$

(3) Determine $x_i \approx x(t_i)$, $i = 1, 2, \cdots, n$ using the recursive process (note that x_0 is given)

$$x_{i+1} = x_i + hf(t_i,\, x_i), \quad i = 0, 1, 2, ..., n-1,$$

6.1.6 Euler program

```
#
# Euler method for solving initial value problem of
# ordinary differential equation
#           x'(t) = f(t,x)        for t in [a,b]
#           x(a) = x0
#
#   Input:      f  --- the right hand side function,
#                        e.g.  f:(t,x)->sin(t)+x^2
#               a  --- the left end of the interval
#               b  --- the right end of the interval
#               x0 --- the initial value
#               n  --- the number of subintervals
#                        dividing [a,b]
#
#   Output:     t  --- the array of points on t-axis
#               x  --- the array of points on x-axis
#
Euler:=proc(f::operator,
            a::numeric,
            b::numeric,
           x0::numeric,
            n::integer,
            t::evaln,
            x::evaln)
   local i, delta;

   # the length of subintervals
   delta:=evalf( (b-a)/n );

   # t values
   t:=array(0..n, [ seq( a+i*delta, i=0..n) ] );

   # opening space for x values
   x:=array(0..n);

   # pass the initial value x0 to x[0]
   x[0]:=x0;

   #
   # loop of Euler method
   #
```

```
    for i from 0 to n-1 do
       x[i+1]:=evalf( x[i] + delta*f(t[i],x[i])  )
    od;

    print(‘End‘);

end;
```

We can use the Euler program to solve the population problem.

```
> read(‘a:euler.txt‘):
```

```
> f:=(t,p)->0.012*p + 0.3;
```

$$f := (t, p) \to .012\,p + .3$$

```
> x0:=25;
```

$$x0 := 25$$

```
> a:=0; b:=5; n:=20;
```

$$a := 0$$
$$b := 5$$
$$n := 20$$

```
> Euler(f,a,b,x0,n,t,x);
```

End

```
> t[15];
```

3.750000000

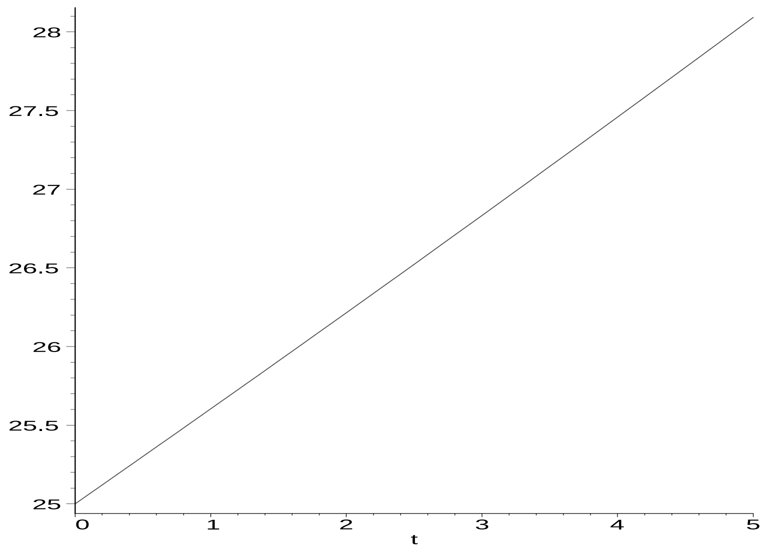
```
> x[15];
```

27.29786982

That is, 3.75 years from now, the population is expected to be 27,301.

6.1.7 To graph the exact (theoretical) solution

```
> plot(-25+50*exp(0.012*t), t=0..5);
```



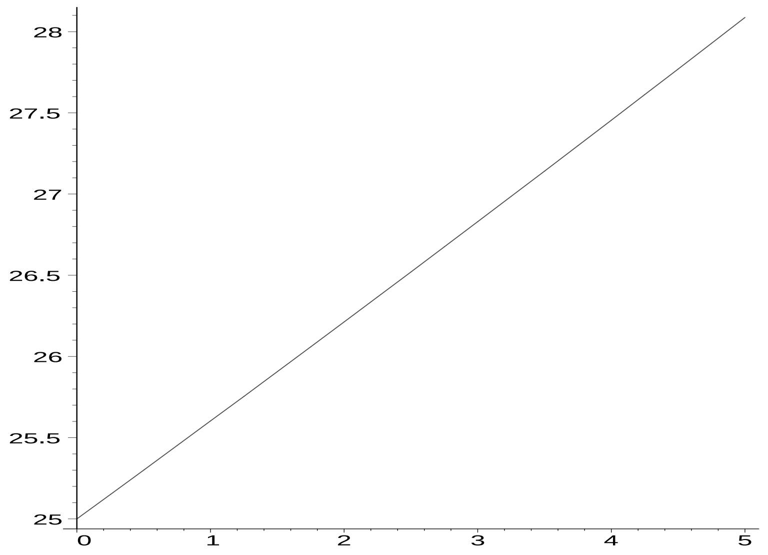
6.1.8 To graph the numerical solution

a) Define the sequence of points

```
> P:=[  seq( [t[i],x[i]], i=0..n ) ]:
```

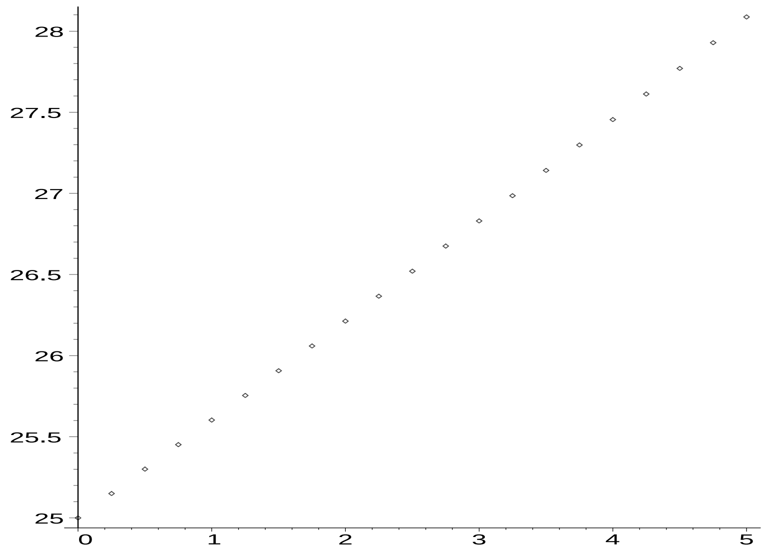
b) plot

```
> plot(P);
```



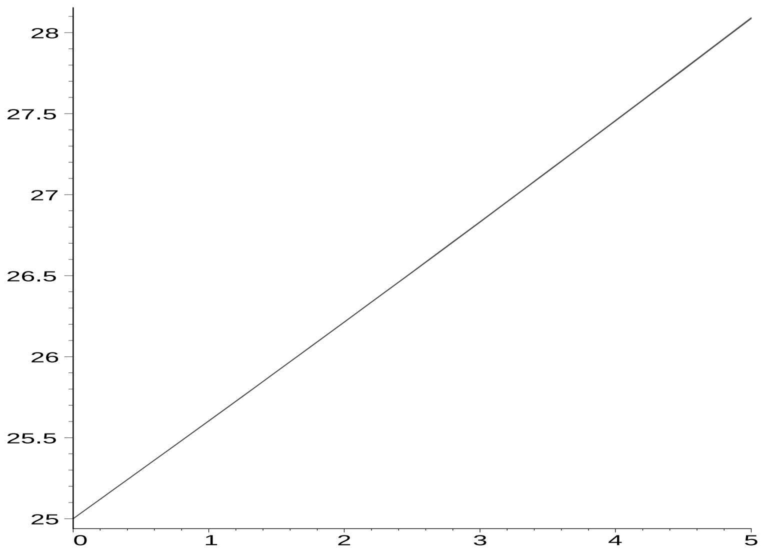
or

```
> plot(P,style=point, symbol=diamond);
```



6.1.9 To graph both exact and numerical solutions

```
> plot1:=plot(-25+50*exp(0.012*t), t=0..5):

> plot2:=plot(P):

> plots[display]({plot1,plot2});
```



Two curves are nearly identical

6.1.10 To plot the error graph

The exact solution and the numerical solution are nearly identical. One may want to see how good the Euler method and plot the error.

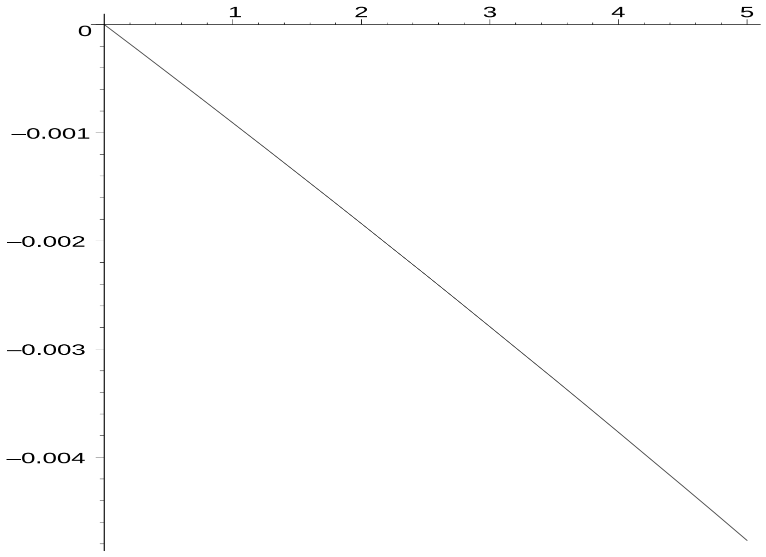
The exact solution as an operator function

```
> p:=t->-25+50*exp(0.012*t);
```

$$p := t \to -25 + 50\, e^{(.012\, t)}$$

Generate the error points using the exact solution

```
> P1:=[  seq( [ t[i], x[i]-p(t[i]) ], i=0..n ) ]:

> plot(P1);
```



The graph shows that the magnitude of error is increasing, with maximum error about 0.001

6.1.11 To increase the accuracy of the numerical solution

The accuracy of the Euler method is determined by n, the number of subintervals dividing $[a, b]$. If we increase n to 400, we'll see that the error reducing by a half.

```
> n:=40;
```

$$n := 40$$

```
> f:=(t,p)->0.012*p+.3; a:=0; b:=5; x0:=25;
```

$$f := (t, p) \to .012\,p + .3$$
$$a := 0$$
$$b := 5$$
$$x0 := 25$$

```
> Euler(f,a,b,x0,n,t,x):
```

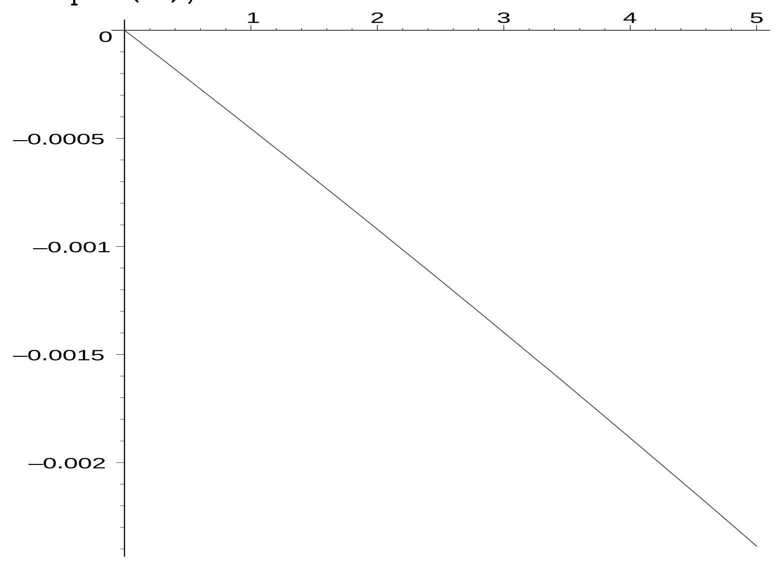
$$\mathit{End}$$

```
> p:=t->-25+50*exp(0.012*t);
```

$$p := t \to -25 + 50\,e^{(.012\,t)}$$

```
> P1:=[  seq( [ t[i], x[i]-p(t[i]) ], i=0..n ) ]:
```

```
> plot(P1);
```



Generally, the errorof Euler method is inversely propotional to n.

6.1.12 Additional example

The initial value problem of the ODE:

$$\begin{cases} \dfrac{dx}{dt} = -\dfrac{x}{t+1}, & t \in [0,9] \\ x(0) = 1 \end{cases}$$

The theoretical solution:

```
> x:='x';
```

$$x := x$$

```
> dsolve(diff(x(t),t)=-x(t)/(t+1),x(t));
```

$$\mathrm{x}(t) = \frac{\_C1}{t+1}$$

Since $\mathrm{x}(0) = 1$, we see that $\_C1 = 1$. Thus the solution is $\mathrm{x}(t) = \frac{1}{t+1}$. The numerical solution:

```
> n:=50; a:=0; b:=9; x0:=1; g:=(t,x)->-x/(t+1);
```

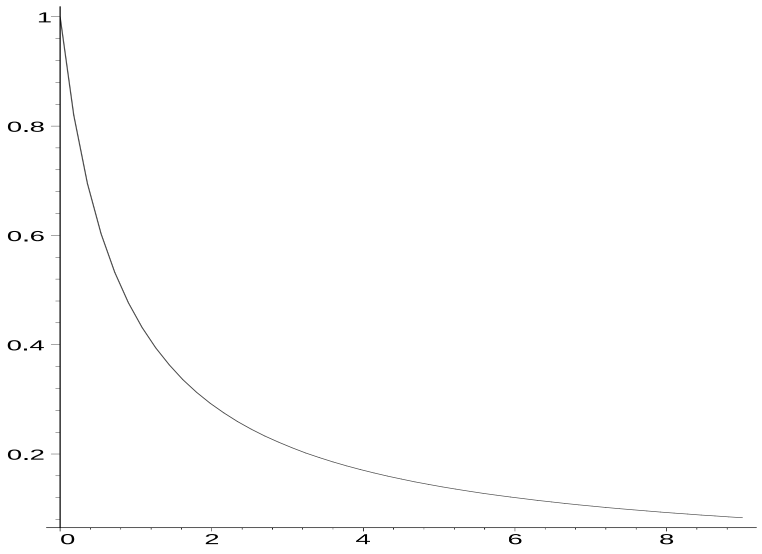
$$n := 50$$
$$a := 0$$
$$b := 9$$
$$x0 := 1$$
$$g := (t,\, x) \to -\frac{x}{t+1}$$

```
> Euler(g,a,b,x0,n,t,x):
```

End

```
> P:=[  seq( [t[i],x[i]], i=0..n ) ]:
```

```
> plot(P);
```

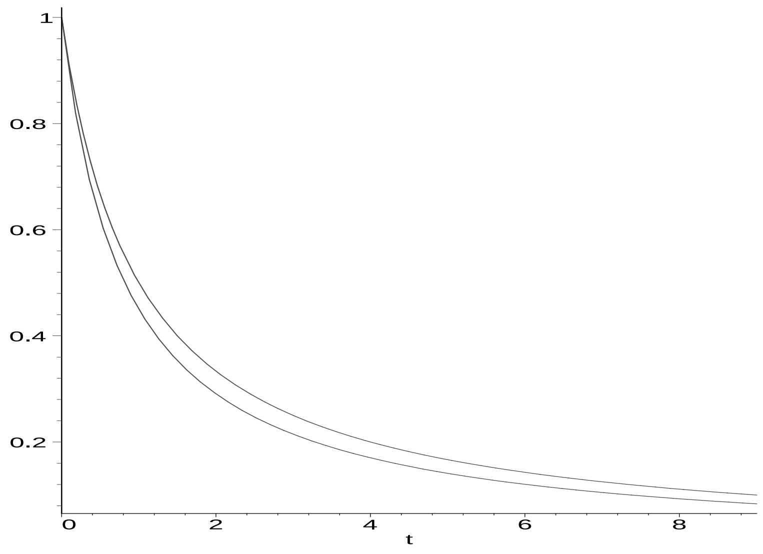


Graph both exact and numerical solutions:

```
> plot1:=plot(P):
```

```
> plot2:=plot(1/(t+1), t=0..9):
```

```
> plots[display]({plot1,plot2});
```

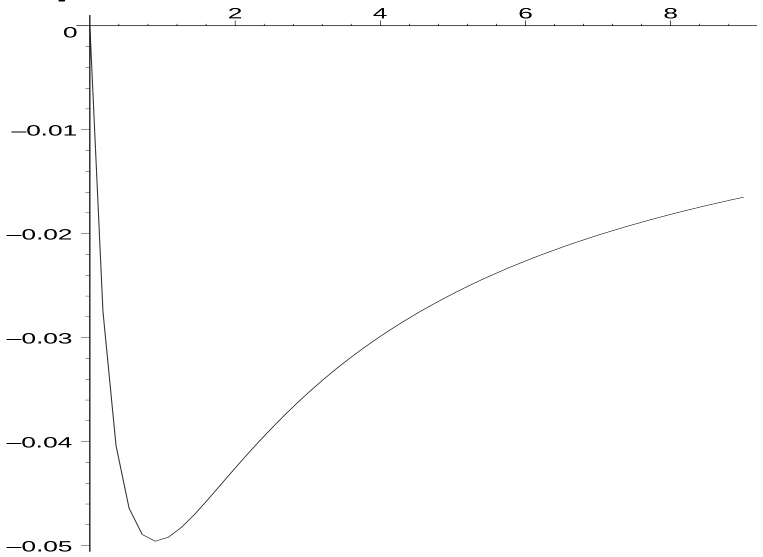


The error is evident. We can plot the error graph

```
>  s:=t->1/(t+1);
```

$$s := t \to \frac{1}{t+1}$$

```
>  P1:=[ seq( [t[i],x[i]-s(t[i])],i=0..n ) ]:
>  plot(P1);
```



The biggest error, about 0.05 occur at around $t = 1.5$.

To reduce the error:

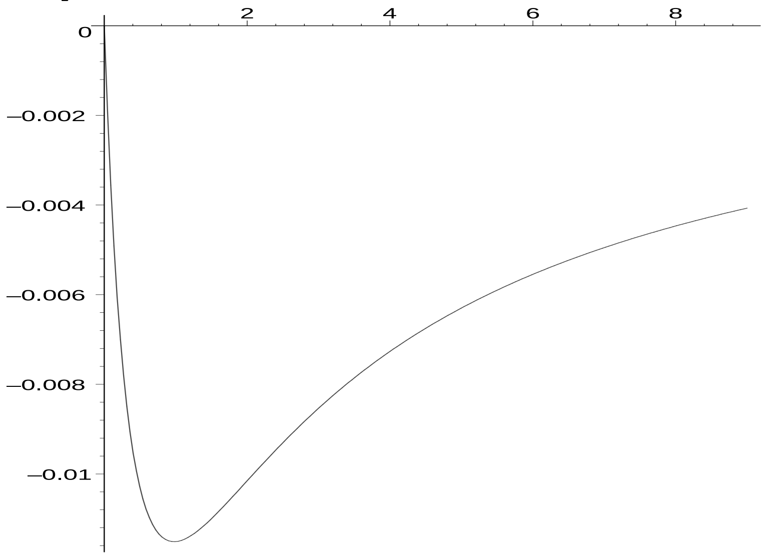
```
>  n:=200;
```

$$n := 200$$

```
>  Euler(g,a,b,x0,n,t,x);
```

$$End$$

```
>  P2:=[ seq( [t[i],x[i]-s(t[i])],i=0..n ) ]:
>  plot(P2);
```



The error is reduced to about a quarter because we increased n four-fold.

6.2 The initial value problem of of ODE systems

6.2.1 Predator-prey model

There are wolves and sheep in an enviorenment. Sheep give births to sheep and are food of wolves. If there were no sheep, wolves would die of hunger, while the more sheep are present, the more wolves would grow. The interaction of wolves and sheep is measured by their potential encounters. Every sheep can potentially meet with every wolf. If there are 20 sheep and 4 wolves the number of potential encounters is 20x4=80.

Every year, the birth rate of sheep is 100% and the death rate of wolves is 50%. Every year 10% of those potential encounters result in the death of sheep, while the number births of wolves coincides with 2% of potential encounters. Currently there are 20 sheep and 4 wolves. Predict their numbers 1, 2, 3.5 years later.

Solution: Both numbers of sheep and wolves are functions of time t. So let $x(t)$ and $y(t)$ be those functions. We now translate the facts into equations:

increasing rate of sheep	is	birth rate	minus	death rate
$\frac{dx}{dt}$	$=$	$1.00\,x$	$-$	$0.10\,x\,y$

increasing rate of wolves	is	birth rate	minus	death rate
$\frac{dy}{dt}$	$=$	$0.02\,x\,y$	$-$	$0.50\,y$

Adding initial conditions $x(0) = 20$, $y(0) = 4$, we have a system of inital value problems of ordinary differential equations:

$$\begin{cases} \dfrac{dx}{dt} = x - .1\,x\,y & \\ & t \in [0, 10] \\ \dfrac{dy}{dt} = -0.5\,y + .02\,x\,y & \\ x(0) \;=\; 20, \quad y(0) \;=\; 4 & \end{cases}$$

It can also be solved approximately by Euler method, which we'll discuss below.

Genreally, for positive constants

$$\begin{cases} \dfrac{dx}{dt} = c_1\,x - c_2\,x\,y & \\ & t \in [a, b] \\ \dfrac{dy}{dt} = c_3\,y + c_4\,x\,y & \\ x(0) \;=\; x_0, \quad y(0) \;=\; y_0 & \end{cases}$$

6.2.2 The Euler method for systems of IVP of ODE

An initial value problem of a 2×2 (i.e. two variables and two equations) system ODE can be generally written as

$$\begin{cases} \dfrac{dx}{dt} = f(t,\, x,\, y) & \\ & t \in [a,b] \\ \dfrac{dy}{dt} = g(t,\, x,\, y) & \\ x(0) = x_0, \quad y(0) = y_0 & \end{cases}$$

In applications, exact solutions are difficult or impossible to obtain. So we divide the t interval $[a,\, b]$ into n subintervals with nodes

$$a = t_0 < t_1 < \cdots < t_n = b$$

and use Euler method

$$\begin{aligned} h &= \frac{b-a}{n} \\ t_i &= a + i\,h, \quad i = 0, 1, \cdots, n \end{aligned}$$

$$\begin{array}{l} x_{i+1} = x_i + h\,f(t_i,\, x_i,\, y_i) \\ y_{i+1} = y_i + h\,g(t_i,\, x_i,\, y_i) \end{array} \quad i = 0, 1, 2, ..., n-1,$$

6.2.3 Project: implement the Euler method for 2x2 system of IVP of ODE

Write a program that, for input f, g, a, b, x_0, y_0, n, output arrays t, x, y as arguments. Use your program to solve the predator-prey problem above and plot both x and y in the same coordinate system. Make some observations from the graph.

6.2.4 Project 2: An epidemic model

An epidemic starts in a population of 100 people. At present, 5 people are sick (called infectives) and 95 people are healthy (called susceptibles). Every month, 0.5% of *potential encounters* between susceptibles and infectives results in susceptibles infected, while 10% of infectives are removed by isolation or death. To be simple, the birth rate is considered zero. Let $x(t)$ and $y(t)$ be the numbers of susceptibles and invectives respectively. Construct a 2x2 system of IVP of ODE and solve it by your program.

(Hint: translate the following sentences into equations: (i) the inceasing rate of susceptibles is the birth rate minus the infection rate; (ii) the increasing rate of infectives is the infection rate minus the removal rate.)

Remark: The resulting model is called *Kermack-McKendreck model.*

6.3 Direction fields

6.3.1 Direction field of an ODE

Consider the ordinary differential equation

$$\frac{dx}{dt} = -\frac{x}{t+1}$$

The left-hand side is the derivative of the function $x(t)$, which is the tangent (or direction) of the graph of $x(t)$ in tx-plane. The right-hand side is a two variable function $f(t, x)$. Therefore, we can consider the ODE as a family of directions: for every pair (t, x), there is a direction (or tangent).

For example, for $(t, x) = (3, 5)$, we have

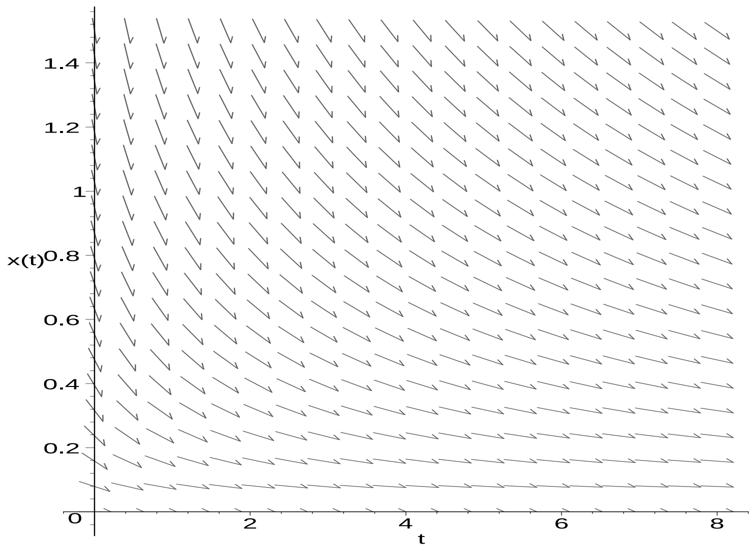
$$\frac{dx}{dt} = -\frac{5}{3+1} = -\frac{5}{4}$$

That is, if the curve $x(t)$ passes through (3,5), the curve should go along the direction (tangent) $-\frac{5}{4}$. Since every point (t, x) represents a direction, the tx-plane can be considered a direction field. Actually, this direction field can be ploted by Maple:

```
>  with(DEtools); x:='x':  t:='t':
```

[*DEnormal*, *DEplot*, *DEplot3d*, *DEplot_polygon*, *DFactor*, *Dchangevar*, *GCRD*, *LCLM*, *PDEchangecoords*, *RiemannPsols*, *abelsol*, *adjoint*, *autonomous*, *bernoullisol*, *buildsol*, *buildsym*, *canoni*, *chinisol*, *clairautsol*, *constcoeffsols*, *convertAlg*, *convertsys*, *dalembertsol*, *de2diffop*, *dfieldplot*, *diffop2de*, *eigenring*, *endomorphism_charpoly*, *equinv*, *eta_k*, *eulersols*, *exactsol*, *expsols*, *exterior_power*, *formal_sol*, *gen_exp*, *generate_ic*, *genhomosol*, *hamilton_eqs*, *indicialeq*, *infgen*, *integrate_sols*, *intfactor*, *kovacicsols*, *leftdivision*, *liesol*, *line_int*, *linearsol*, *matrixDE*, *matrix_riccati*, *moser_reduce*, *mult*, *newton_polygon*, *odeadvisor*, *odepde*, *parametricsol*, *phaseportrait*, *poincare*, *polysols*, *ratsols*, *reduceOrder*, *regular_parts*, *regularsp*, *riccati_system*, *riccatisol*, *rightdivision*, *separablesol*, *super_reduce*, *symgen*, *symmetric_power*, *symmetric_product*, *symtest*, *transinv*, *translate*, *untranslate*, *varparam*, *zoom*]

```
> dfieldplot( diff(x(t),t)=-x(t)/(t+1), x(t), t=0..8, x=0..1.5);
```



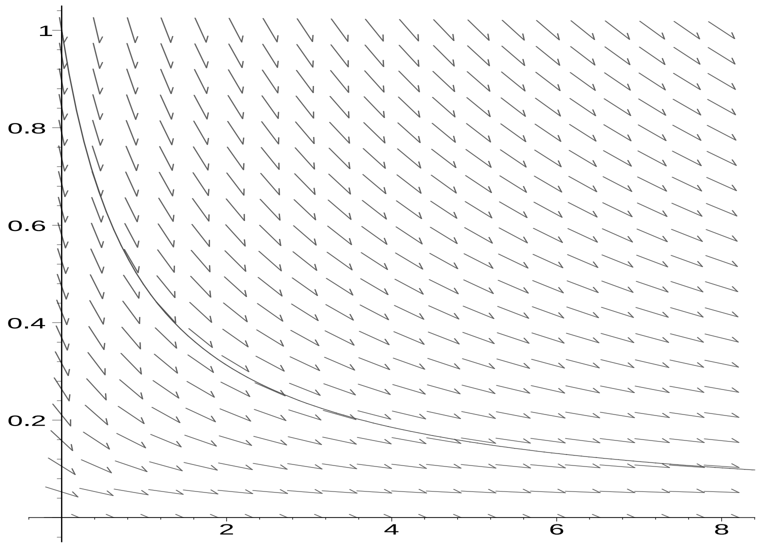
Every solution of this ODE follows this direction field. For example, the solution satisfying the initial condition $x(0) = 1$ can be seen inside the direction field

```
> f:=(t,x)->-x/(t+1):
> read(‘a:euler.txt‘):
> a:=0:  b:=9:  x0:=1:  n:=100:
> Euler(f,a,b,x0,n,t,x):
```

End

```
> P:=[ seq( [t[i],x[i]],i=0..n )]:
```

```
> plot2:=plot(P,thickness=2):
> t:='t': x:='x':
> plot1:=dfieldplot( diff(x(t),t)=-x(t)/(t+1), x(t), t=0..8,

> x=0..1.):
> plots[display]({plot1,plot2});
```



Genereally, an ODE $\frac{dx}{dt} = f(t, x)$ can be considered a direction (tangent) field in tx-plane. For every point (t, x), the right-hand side function $f(t, x)$ output a number for $\frac{dx}{dt}$, which is a slope of tangent (direction). If the solution curve passes through that point, it must follow that direction.

6.3.2 Direction field of predator-prey model

As an example, consider the Predator-Prey Model:

$$\begin{cases} \dfrac{dx}{dt} = x - .1\, x\, y \\[2ex] \dfrac{dy}{dt} = -0.5\, y + .02\, x\, y \end{cases}$$

The left-hand side $\begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix}$ is the *tangent vector* of the solution curve. The right-hand side can be considered a *vector* function $\begin{bmatrix} x - .1\,x\,y \\ -.5\,y + .02\,x\,y \end{bmatrix}$ of the variable $(x,\ y)$. That is, at every point $(x,\ y)$ on xy-plane, the right-hand side vector function produces a vector, that must be the tangent of vector of a solution curve passing through this point.

For example, at the initial point $(x,\ y)$=(20, 4), the right-hand side vector is

$$\begin{bmatrix} x - .1\,x\,y \\ -.5\,y + .02\,x\,y \end{bmatrix} = \begin{bmatrix} 12 \\ -.4 \end{bmatrix}$$

Therefore, at the start of the solution curve, the direction vector (12,-0.4) must be followed.

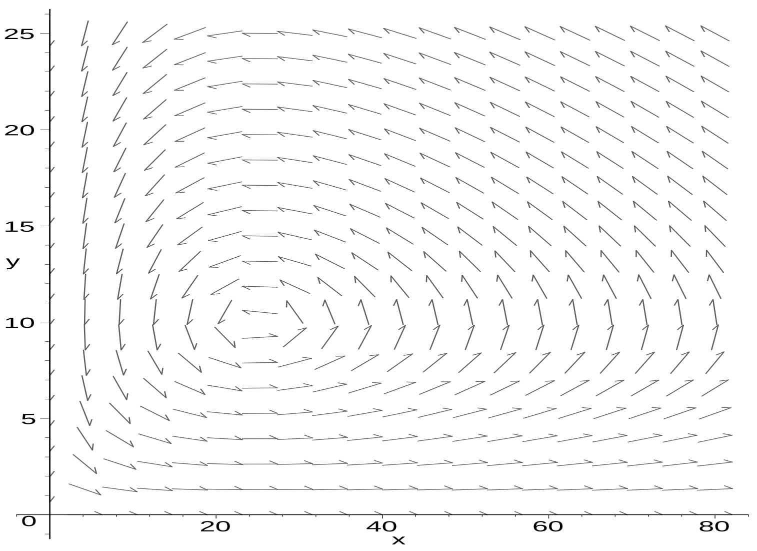
Since there is a direction vector at every point $(x,\ y)$ on xy-plane produced by the right-hand side functions, the hole xy-plane can be considered a vector field. We can actually plot this direction field:

```
> ode1:=diff(x(t),t)=x-0.1*x*y; ode2:=diff(y(t),t)=-0.5*y+0.02*x*y;
```

$$ode1 := \frac{\partial}{\partial t}\,\mathrm{x}(t) = x - .1\,x\,y$$

$$ode2 := \frac{\partial}{\partial t}\,\mathrm{y}(t) = -.5\,y + .02\,x\,y$$

```
> dfieldplot( [ode1,ode2],[x(t),y(t)],t=0..10,x=0..80,y=0..25);
```



```
> plot3:=%:
```

Note: the direction field plot is saved for later use!

We shall solve the initial value problem of ODE system by the Runge-Kutta method, which is much more accurate than Euler's method. The theory of Runge-Kutta methods is out of scope of this text. We shall use it without further discussion. Generally, we need much smaller n to achieve the same accuracy than Euler's method. (You may use the program in homework).

```
#
#  Program that, with Runge-Kutta method, solves an initial value problem
#  mxm system of ODE
#        x[i]' = f[i](t,x[1],x[2],...,x[m]),     t in [a,b]
#        x[i](0) = x0[i],
#                i=1,2,...,m
#
#        input: m        --- size of the system
#               f        --- an array of operators
#                            for example:
#                            >f:=array(1..2);
#                            >f[1]:=(t,x,y)->x-0.1*x*y;
#                            >f[2]:=(t,x,y)->-.5*y+.02*x*y;
#               a, b     --- end points of the t interval
#               x0       --- array(1..m) of initial conditions
#               n        --- number of subintervals dividing [a,b]
#
#       output: t        --- the array of nodes dividing [a,b]
#               x        --- the array(1..m), each entry x[i] is an
#                            array(0..n)
#                            x[i][j] is the approximate value of
#                            x[i](t[j])
#
RKmxm:=proc(m::integer,

            f::array,

            a::numeric,

            b::numeric,

           x0::array,
            n::integer,

            t::evaln,

            x::evaln)
   local i, j, k, l, delta, delta2;

   delta:=evalf( (b-a)/n );  delta2:=0.5*delta;
   t:=array(0..n, [ seq( evalf( a+i*delta ), i=0..n )]);
   x:=array(1..m, [ seq( array(0..n), i=1..m) ] );
   k:=array(1..4, [ seq( array(1..m), i=1..4) ] );
   for i to m do
      x[i][0]:=evalf(x0[i])
   od;

   for i from 0 to n-1 do
      for j to m do
         k[1][j]:= evalf( delta*f[j](t[i], seq( x[l][i], l=1..m ) ) )
      od;
      for j to m do
         k[2][j]:= evalf( delta*f[j](t[i]+delta2,

                          seq( x[l][i]+0.5*k[1][l], l=1..m ) ) )
      od;
      for j to m do
         k[3][j]:= evalf( delta*f[j](t[i]+delta2,

                          seq( x[l][i]+0.5*k[2][l], l=1..m ) ) )
      od;
      for j to m do
         k[4][j]:= evalf( delta*f[j](t[i]+delta,

                          seq( x[l][i]+k[3][l], l=1..m ) ) )
      od;
      for j to m do
         x[j][i+1]:= evalf( x[j][i] +
                 (k[1][j]+2.0*(k[2][j]+k[3][j])+k[4][j])/6.0 )
      od;
   od;

   print(` End of Runge-Kutta method for mxm IVP-ODE `)

end;
```

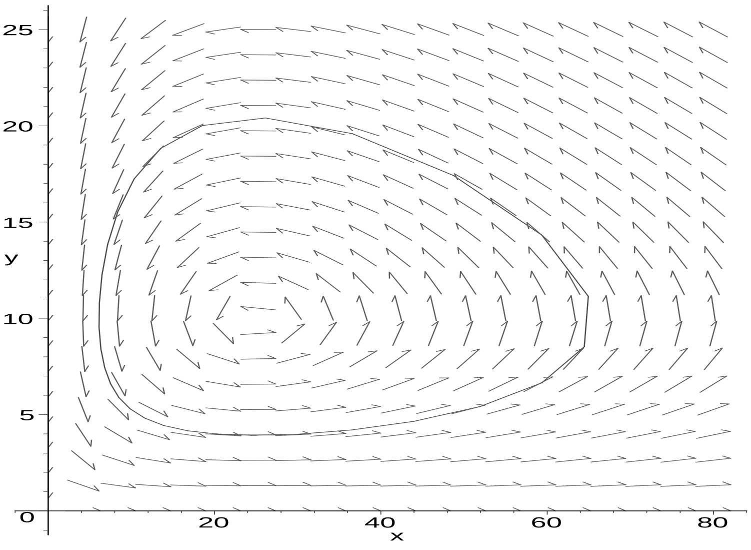
```
   > read(`a:R-K_mxm.txt`):
   > m:=2:
   > f:=array(1..m):
   > f[1]:=(t,x,y)->x-0.1*x*y:
   > f[2]:=(t,x,y)->-.5*y+.02*x*y:
   > a:=0:    b:=10:  n:=30:
   > x0:=array(1..m,[20,4]):
   > RKmxm(m,f,a,b,x0,n,t,x);
```

End of Runge – Kutta method for mxm IVP – ODE

```
   > P:=[ seq( [x[1][i],x[2][i]], i=0..n) ]:
   > plot4:=plot(P,thickness=2):
```

Now we plot the solution and direction field together. Recalled that plot3 was defined earlier.

```
> plots[display]({plot3,plot4});
```



From this graph, we can clearly observe the ecological cycle: Starting from 20 sheep and 4 wolves, sheep increase while wolves decrease and then increase too, until about 65 sheep and 10 wolves are present. Then wolves keep increasing but sheep decrease until there are about 23 sheep and 20 wolves. Then both sheep and wolves decreases, then sheep rebound, back to 20 sheep and 4 wolves.

From this direction field, we can also observe that, if the enviorenment starts with 25 sheep and 10 wolves, there should be little, if any, changes for the two populations. That's an ecological equilibrium.

We can also investigate the solutions of different initial values using DEplot (notice that ode1 and ode2 were defined earlier).

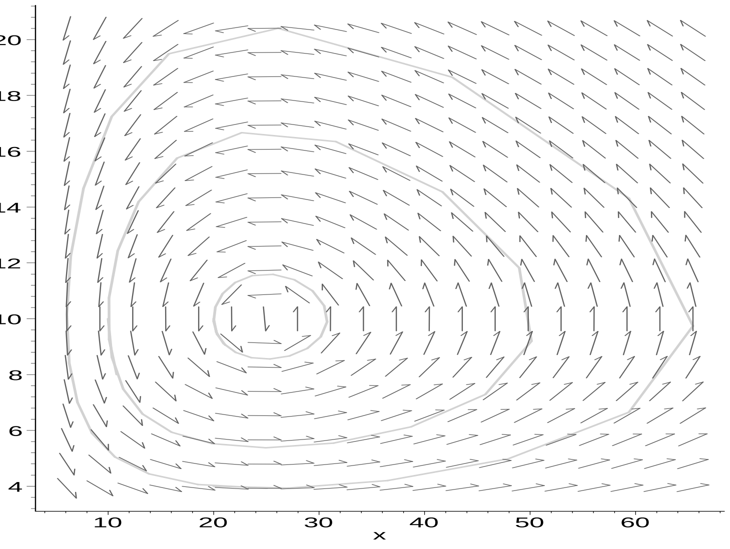
```
> ode1:=diff(x(t),t)=x-0.1*x*y; ode2:=diff(y(t),t)=-0.5*y+0.02*x*y;
```

$$\mathit{ode1} := \frac{\partial}{\partial t}\,\mathrm{x}(t) = x - .1\,x\,y$$

$$\mathit{ode2} := \frac{\partial}{\partial t}\,\mathrm{y}(t) = -.5\,y + .02\,x\,y$$

```
> ini_cond1:=[ x(0)=20, y(0)=4 ]:
> ini_cond2:=[ x(0)=10, y(0)=10]:
> ini_cond3:=[ x(0)=20, y(0)=10]:
> ini_cond4:=[ x(0)=25, y(0)=10]:  x:='x':  y:='y':  t:='t':
```

```
> DEplot( [ode1, ode2], [x(t),y(t)], 0..10,
> [ini_cond1,ini_cond2,ini_cond3,ini_cond4]);
```



Observe that the fourth initial condition produced only one point. That is, if we start with 25 sheep and 10 wolves, both populations stay the same.

6.3.3 Direction fields of general 2x2 systems of ODE's

Generally, for a 2x2 system of first order ODE's, *with right-hand side independent of t,*

$$\begin{aligned}\frac{dx}{dt} &= f(x,\,y)\\ \frac{dy}{dt} &= g(x,\,y)\end{aligned}$$

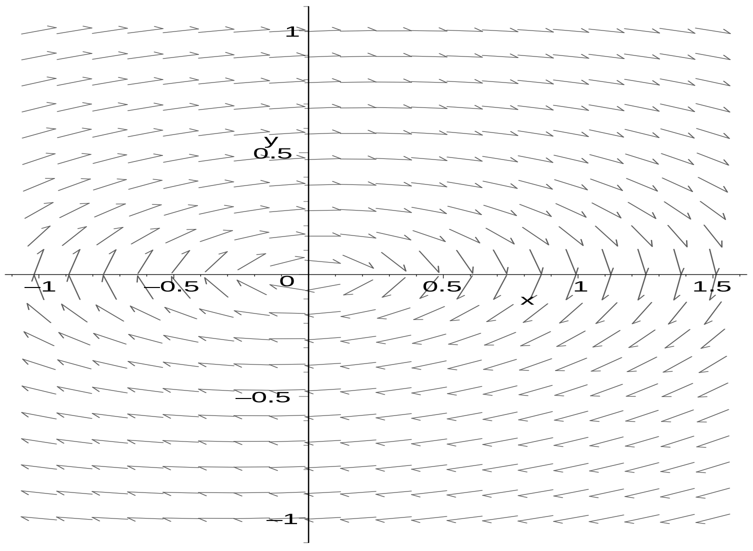
every point $(x,\,y)$ on the xy-plane corresponds a direction vector produced by the right-hand functions. Thus, the whole xy-plane is a direction field. Solutions of the ODE system are curves following the directions in the field.

Example 1: The 2x2 ODE system

$$\begin{aligned}\frac{dx}{dt} &= y\\ \frac{dy}{dt} &= -.15\,x + .05\,y\end{aligned}$$

is a direction field:

```
>  t:='t':  x:='x':  y:='y':

>  ode3:=diff(x(t),t)=y:

>  ode4:=diff(y(t),t)=-.15*x+0.05*y:

>  dfieldplot( [ode3,ode4],[x(t),y(t)],t=0..1,x=-1..1.5,y=-1..1);
```



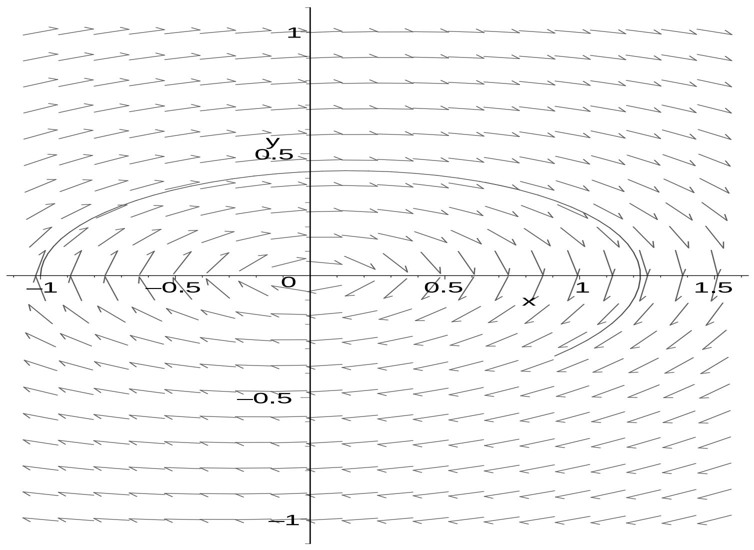
```
>  plot5:=%:

>  m:=2:

>  g:=array(1..m,[(t,x,y)->y,

>  (t,x,y)->-0.15*x+.05*y]);
```

$$g := [(t,\, x,\, y) \rightarrow y,\, (t,\, x,\, y) \rightarrow -.15\, x + .05\, y]$$

```
>  a:=0:    b:=10:  y0:=array(1..m,[-1,0]):  n:=50:

>  RKmxm(m,g,a,b,y0,n,t,y);
```

End of Runge – Kutta method for mxm IVP – ODE

```
>  Q:=[ seq( [y[1][i],y[2][i]], i=0..n )]:

>  plot6:=plot(Q,thickness=2):

>  plots[display](plot5,plot6);
```



6.3.4 Parametric curves in xy-plane

For every t value in the common domain $[a, b]$ of functions $x(t)$ and $y(t)$, $(x(t), y(t))$ is a point in xy-plane. When t goes continuously from a to b, the point set

$$\left\{ (x(t),\, y(t)) \;\middle|\; t \in [a, b] \right\}$$

is called a parametric curve. The solution of

$$\left\{ \begin{array}{ll} \dfrac{dx}{dt} = f(t,\, x,\, y) & \\ & t \in [a, b] \\ \dfrac{dy}{dt} = g(t,\, x,\, y) & \\ x(0) = x_0, \quad y(0) = y_0 & \end{array} \right.$$

can thereby be considered a parametric curve, with t considered a parameter. The above examples show the graphs of such parametric curves.

6.3.5 Graphs the solutions of initial value problems of ODE systems as parametric curves

Use Runge-Kutta program to solve the following initial value problems of ODE systems and graph the corresponding parametric curves.

6.3.6 Cardioid

$$\begin{cases} \dfrac{dx}{dt} = -y + \cos t\, \sin t \\ \dfrac{dy}{dt} = x + \sin^2 t \\ x(0) = 0, \quad y(0) = 0 \end{cases} \qquad t \in [0,\, 2\pi]$$

The exact solution is $x(t) = (1 - \cos t)\cos t, \quad y(t) = (1 - \cos t)\sin t$

6.3.7 Three-leaved rose

$$\begin{cases} \dfrac{dx}{dt} = -y + 3\cos 3t\, \cos t \\ \dfrac{dy}{dt} = x + 3\cos 3t\, \sin t \\ x(0) = 0, \quad y(0) = 0 \end{cases} \qquad t \in [0,\, 2\pi]$$

The exact solution is $x = \sin 3t\, \cos t, \quad y = \sin 3t\, \sin t.$

6.3.8 Folium of Descartes

$$\begin{cases} \dfrac{dx}{dt} = \dfrac{1}{1+t^3} - 3\,x\,y \\ \dfrac{dy}{dt} = 2\,x - 3\,y^2 \\ x(0) = 0, \quad y(0) = 0 \end{cases} \qquad t \in [0,\, 10]$$

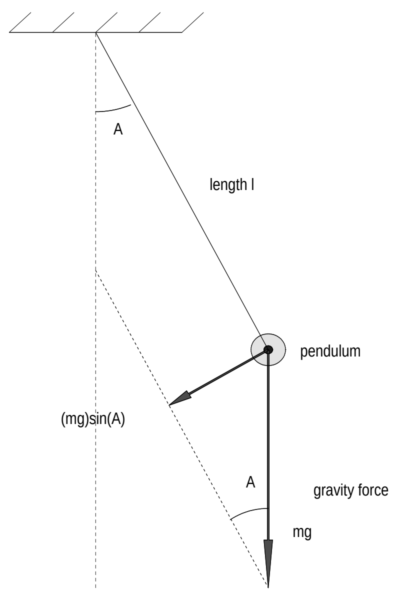
The exact solution is $x = \dfrac{t}{1+t^3}, \quad y = \dfrac{t^2}{1+t^3}.$

6.4 The initial value problem of second order ODE's

6.4.1 Example: Pendulum

Let's consider a pendulum in the following figure

```
> with(plottools): with(plots): # load packages
Warning, new definition for translate
> pendulum:=line([0,14],[4,6],thickness=2), disk( [4,6],0.4 ,
> color=yellow), disk( [4,6], 0.1, color=blue):
> ceiling:=line([-2,14],[2,14],thickness=2),line([-2,14],[-1.5,14.5],th
> ickness=1),line([-1,14],[-0.5,14.5],thickness=1),line([0,14],[0.5,14.5
> ],thickness=1),line([1,14],[1.5,14.5],thickness=1),line([2,14],[2.5,14
> .5],thickness=1):
> gravity:=arrow([4,6],[4,0],.04,.2,.2,color=red):
> actual_force:=arrow([4,6],[1.7,4.6],.04,.2,.2,color=red):
> ref_line:=line([0,0],[0,14],thickness=1,linestyle=2),line([0,8],[4,0]
> ,linestyle=4):
> angle:=arc([0,14],2,-Pi/2..-1.1*Pi/3),arc([4,0.5],1.5,Pi/2..2.1*Pi/3)
> :
> text:=textplot([[6.0,2.5,`gravity
> force`],[3.6,2.7,`A`],[5.5,6,`pendulum
> `],[0.53,11.6,`A`],[-0.0,4.3,`(mg)sin(A)`],[4.8,1.4,`mg`],[3.2,10.2,`l
> ength l `]]):
> display({ceiling,pendulum,gravity,actual_force,ref_line,angle,text
>},axes=NONE);
```



The pendulum, with mass m, is subject to gravity force $m\,g$ Newton (i.e. $\frac{kilogram \cdot meter}{second^2}$, the force that causes 1 kilogram having 1 meter/second^2 acceleration). However, because of the pendulum string attached to the ceiling, the actual force that creats the acceleration is the component of the gravity force perpendicular to the pendulum string. This force is $m\,g$ sin A, where A is the angle the pendulum string and the vertical line. The pendulum moves along the circular arc, the distance traveled $s = l\,A$, where l is the length of the pendulum string.

According to the Newton's Second Law $f = m\,a$, the acceleration of the pendulum multiplied by its mass equal to the force it subjects to:

$$m\,\frac{d^2\,s}{dt^2} = -m\,g\sin(A)$$

Since $s = l\,A$, $\frac{d^2\,s}{dt^2} = l\,\frac{d^2\,A}{dt^2}$. Canceling m on both sides, we have the ODE for the pendulum swing angle A:

$$\frac{d^2\,A}{dt^2} = -\frac{g\sin(A)}{l} \tag{6.1}$$

Example: A pendulum, attatched to a string of 2 meters, is pulled $\frac{\pi}{6}$ away from the vertical line and released. The angle A of the pendulum then is a function of time t. Use Runge-Kutta method to calculate the approximate values of the function and plot the function.

Solution: The pendulum angle is subject to the ODE in equation (6.1). At beginning, the angle $A = \frac{\pi}{6}$, and $\frac{dA}{dt} = 0$ (because the pendulum is simply released without initial velocity). We have an IVP of second order ODE:

$$\begin{aligned} \frac{d^2 A}{dt^2} &= -\frac{9.8\sin(A)}{2}, \quad t \in [0, \infty] \\ & A(0) = \frac{\pi}{6}, \quad \left.\frac{dA}{dt}\right|_{t=0} = 0 \end{aligned}$$

The numerical methods we discussed so far are disigned for first order ODE's. To solve second order ODE problems, we need to convert the problem to a first order system. This conversion can be done with a substitution

$$B = \frac{dA}{dt}, \quad \text{therefore} \quad \frac{dB}{dt} = \frac{d^2A}{dt^2}$$

the pendulem problem above becomes an IVP of 2x2 system of ODE:

$$\begin{aligned} \frac{dA}{dt} &= B \\ \frac{dB}{dt} &= -4.9 \sin A \\ & A(0) = \frac{\pi}{6}, \quad B(0) = 0 \end{aligned}$$

```
> m:=2:  f:=array(1..m,[(t,A,B)->B, (t,A,B)->-4.9*sin(A)]);
```
$$f := [(t, A, B) \to B, (t, A, B) \to -4.9\sin(A)]$$

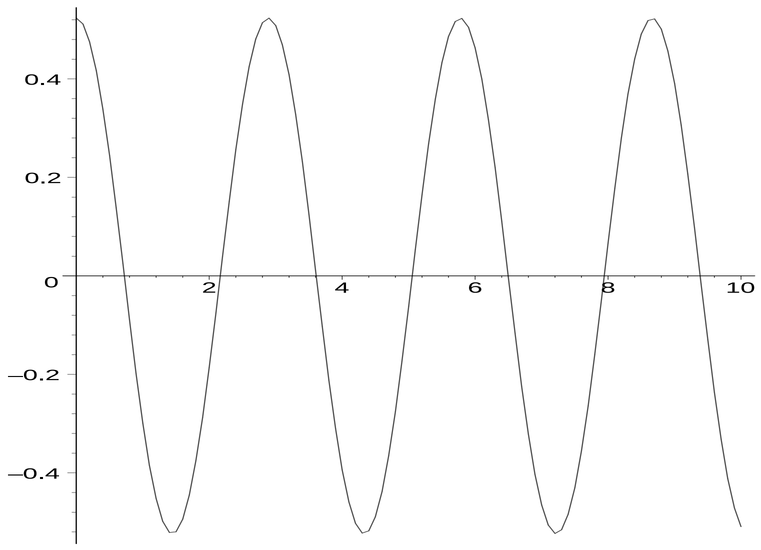
```
> a:=0:    b:=10:    x0:=[Pi/6,0]:    n:=100:
```

```
> RKmxm(m,f,a,b,x0,n,t,x);
```
End of Runge – Kutta method for mxm IVP – ODE

The objective of the pendulum problem is the angle function $A = A(t)$. The extra variable B we introduced is no longer useful. We plot the function $A(t)$ only.

```
> P:=[ seq( [t[i],x[1][i]],i=0..n) ]:
```

```
> plot(P);
```



The pendulum angle goes back and forth. The most important important feature is that the pendulum swing back and forth in equal time. That's the theoretical fundation of pendulum clock.

6.4.2 Transform a high order ODE to a system of ODE

6.4.3 The transformation

Generally, for a given ODE of order m

$$\frac{d^m x}{dt^m} = f\left(t, x, \frac{dx}{dt}, \frac{d^2 x}{dt^2}, \cdots, \frac{d^{(m-1)} x}{dt^{(m-1)}}\right) \tag{6.2}$$

we can use substitutions

$$\begin{cases} x_1 = x \\ x_2 = \dfrac{dx}{dt} \\ \vdots \\ x_m = \dfrac{d^{(m-1)} x}{dt^{(m-1)}} \end{cases}$$

to construct a system of m ODE's in m variables:

$$\begin{cases} \dfrac{dx_1}{dt} = x_2 \\ \dfrac{dx_2}{dt} = x_3 \\ \vdots \\ \dfrac{dx_{m-1}}{dt} = x_m \\ \dfrac{dx_m}{dt} = f(t, x_1, x_2, \cdots, x_m) \end{cases}$$

6.4.4 Projects: solve IVP of high order ODE with R-K method

Solve the following IVP of high order ODE by Runge-Kuta method and compare with exact solutions using error plot.

Project 1

$$t^2 \frac{d^2 x}{dt^2} - 2t \frac{dx}{dt} + 2x = t^3 \ln(t), \quad t \in [1, 4]$$

$$x(1) = 1, \quad x'(1) = 0$$

The actual solution:

$$x(t) = \frac{7t}{4} + \frac{t^3 \ln(t)}{2} - \frac{3t^3}{4.}$$

Project 2

$$t^3 \frac{d^3 x}{dt^3} - t^2 \frac{d^2 x}{dt^2} + 3t \frac{dx}{dt} - 4x = 5t^3 \ln(t) + 9t^3, \quad t \in [1, 5]$$

$$\mathrm{x}(1) = 0, \quad x'(1) = 1, \quad x''(1) = 3$$

The actual solution:

$$\mathrm{x}(t) = -t^2 + t\cos(\ln(t))) + t\sin(\ln(t)) + t^3 \ln(t)$$

6.5 Exercises

1. **Using the Euler program**

   a) Use Maple dsolve to solve the ODE and determine the constant *_C1*

   b) Use the Euler program to solve the IVP of ODE, and produce four graphs: graph of exact solution, graph of numerical solution as discrete points, graph of both exact and numerical solutions, graph of error.

   $$\frac{dx}{dt} = \frac{x}{t} - (\frac{x}{t})^2, \quad t \in [1,3]$$
   $$\mathrm{x}(1) = 1$$

2. **Using the Euler program (cont.)**

   Follow the instruction of Problem 1, and do the same for the IVP of ODE:

   $$\frac{dx}{dt} = 1 + (t-x)^2, \quad t \in [2,3]$$
   $$\mathrm{x}(2) = 1$$

3. **Chemical reaction**

   Suppose three chemical species are in a process of reaction, where every minute 20% of chemical A becomes chemical B, and 25% of chemical B become chemical C. There is no supply of chemical A. The reaction stops at chemical C. At present, there are 100 grams of chemical A, 20 grams of chemical C, and no chemical C. We are interested in the first 10 minutes of the reaction.

   (i) Formulate a 3x3 system of IVP of ODE that describes the interaction between three chemicals.

   (ii) Solve the system by Euler method

   (iii) Graph the amount of three chemicals together.

4. **Direction field**

   Investigate the direction field of the ODE:

   $$\frac{dx}{dt} = t^2 - x$$

   and use Euler method to solve the corresponding IVP with x(0) = 1

5. **Direction field**

   Investigate the direction field for

   $$\frac{dx}{dt} = x + y, \quad \frac{dy}{dt} = x - y$$

   and use a couple of initial values to construct solutions using RKmxm. Plot direction field and solutions together.

6. **Euler method for IVP of second order ODE**

   Write a program that, for input f, a, b, α_0, β_0, n, output the solution array t, x for the IVP of second order ODE:

$$\frac{d^2 x}{dt^2} = \mathrm{f}(t,\, x,\, \frac{dx}{dt}), \quad t \in [a,\, b]$$

$$\mathrm{x}(a) = \alpha_0, \;\; x'(a) = \beta_0$$

   Use your program to solve the problem in Project 1, Section 6.4.4

7. **Euler method for IVP of 3rd order ODE**

   Write a program to solve IVP of third order ODE directly without substitution. Use your program to solve problem in Project 2, Section 6.4.4.

8. **Euler method for IVP of m-th order ODE**

   Can you write a program to solve general IVP of m-th order ODE in equation (6.2)?