Maple and Standing Waves ⋆

Inna Shingareva and Carlos Lizárraga Celaya

Departments of Mathematics and Physics, University of Sonora,
Blvd. Luis Encinas and Rosales, 83000, Hermosillo, Sonora, México
E-mail addresses: *inna@fisica.uson.mx, carlos@fisica.uson.mx*

Abstract

The present subject, Maple and standing waves, can be considered as a blend of physics, mathematics, and computer science that has led to powerful methods for solving many complex problems in science and engineering. Consequently, the subject can be considered from the point of view of a physicist and a mathematician, for creating a mathematical model of a physical problem and a method for solving the problem; of a computer scientist, for implementing it in Maple or modifying it for its optimal operation. The present article can be divided in two parts, the description of Maple; and standing waves with Maple. The material of the present paper can be used in several undergraduate and graduate courses, as well as for solving new research problems.

1 Introduction to Maple

Many problems in various branches of science and engineering often require cumbersome analytic computations, that are difficult and in many cases impossible to perform by hand. The need to perform extensive analytic computations for those problems has led to the idea of using a computer as a tool.

The first two articles describing analytic calculations performed with the aid of a computer were published in 1953, see Calmet and van Hulzen (1983). In the early 70's, systems of analytic computations (SAC), or computer algebra systems (CAS), began to appear. Computer algebra systems are computational interactive programs that facilitate symbolic mathematics. The first popular systems were Reduce, Derive, and Macsyma, which are still available. Macsyma is one of the oldest and most mature systems. It was developed at the Massachusetts Institute of Technology (MIT) but practically its evolution has stopped. But a free software version of Macsyma, Maxima, is actively being maintained. To the present day, more than

⋆ Many topics considered in the present paper are discussed in more details in Shingareva, Lizárraga-Celaya, and Ochoa-Ruiz (2006).

100 computer algebra systems have been developed. All these systems can be conventionally subdivided into two classes, specialized and general-purpose computer algebra systems, see Akritas (1989), Calmet and van Hulzen (1983), and Grosheva and Efimov (1988).

In the present work, we consider the general-purpose computer algebra system Maple. This system is one of the most popular powerful reliable systems being used worldwide by research mathematicians, scientists, engineers, and students. The computer algebra system Maple was developed at the University of Waterloo, in the 1980's, see Char et al. (1990) and Geddes et al. (1992). Maple includes a rich set of functions that makes it comparable to the Macsyma in symbolic power, see Grosheva and Efimov (1988). Maple incorporates many of the best features of other systems developed in the late 1960s. It is written in C language and has been ported to many operating systems. With the realization of symbolic, numeric, and graphic calculus, Maple becomes the most powerful tool for students, professors, scientists, and engineers.

In Sections 1–5, we present the brief description of computer algebra system Maple, considering different areas of mathematics (e.g., differential, integral, and vector calculus, 2D and 3D graphic visualization, solutions of equations). In Sections 6 and 7, we describe standing waves in strings and fluids applying Maple. Research in the theory of waves leads to the consideration of a linear hyperbolic partial differential equation, called the wave equation. The wave equation is a fundamental equation and one of many equations which admit periodic solutions and describe a number of physical phenomena that are observed in many situations and different media. In Section 6 we consider the mathematical description of standing waves that can be observed in infinite and fixed strings, and in Section 7 we consider nonlinear standing waves in fluids. Applying the perturbation theory, Lagrangian formulation of the problem, and Maple, we show the construction of high order asymptotic solutions to a nonlinear system of partial differential equations and their graphic illustrations.

1.1 Basic features of Maple

◦ Fast symbolic and numerical computation, and interactive visualization;
◦ easy to use, help can be found within the program or on the Internet;
◦ extensibility; accessible to large numbers of students and researchers;
◦ available for almost all operating systems;
◦ powerful programming language, intuitive syntax, easy debugging;
◦ extensive library of mathematical functions and specialized packages;
◦ two forms of interactive interfaces: a command-line and a graphic environment;
◦ free resources (Maple Application Center), collaborative character of development (Maple Community);
◦ understandable, open-source software development path.

1.2 Design of Maple

◦ Maple consists of *three parts*: the interface, the kernel (basic computational engine), and the library.

◦ *The interface* and *the kernel* form a smaller part of the system, which has been written in the programming language C; they are loaded when a Maple session is started. *The interface* handles input of mathematical expressions and display of expressions, plots functions, and supports other user communication with the system. The medium of the interface is *the Maple worksheet*. *The kernel* interprets the user input and carries out the basic algebraic operations, and deals with storage management.

◦ *The library* consists of two parts: the basic library and additional packages. The basic library includes many functions in which resides most of the mathematical knowledge of Maple and that has been coded in the Maple language.

◦ *Maple Language* is a high-level programming language, well-structured, comprehensible. It supports a large collection of *data structures* or Maple *objects* (functions, sequences, sets, lists, arrays, tables, matrices, vectors, etc.) and operations on these objects (type-testing, selection, composition, etc.). The Maple procedures in the library are available in readable form. The library can be complemented with locally developed programs and packages.

1.3 Basic rules

◦ *Basic arithmetical operators*: `+ - * / ^`.

◦ *Logical operators*: `and, or, xor, implies, not`. *Relational operators*: `<, <=, >, >=, =, <>`.

◦ *A variable*, is a combination of letters, digits, or the underline symbol (_), beginning with a letter, e.g., `var`, `a12_new`.

◦ *Abbreviations* for the longer Maple functions or any expressions: `alias`, e.g. `alias(H=Heaviside); diff(H(t),t);` to remove this alias, `alias(H=H);`

◦ Maple *is case sensitive*: there is a difference between lower-case and upper-case letters, e.g. `evalf(Pi)` and `evalf(pi)`.

◦ *Various reserved keywords, symbols, names, and functions*: these words cannot be used as variable names, e.g., operator keywords, additional language keywords, global names that start with (_) (see `?reserved`, `?ininames`, `?inifncs`, `?names`).

◦ *The assignment/unassignment operators*: a variable can be "free" (with no assigned value) or assigned any value (symbolic, numeric) by the assignment operators `a:=b` or `assign(a=b)`. To unassign (clear) an assigned variable: `x:='x'`, `evaln(x)`, or `unassign('x')`.

◦ *The difference between the operators* (:=) and (=): the operator `var:=expr` is used to assign `expr` to the variable `var`, and the operator `A=B` — to indicate equality (not assignment) between the left- and the right-hand sides (see `?rhs`), `Equation:=A=B; Equation; rhs(Equation); lhs(Equation);`

◦ *Statements* (we denote `stat`), are input instructions from the keyboard that are executed by Maple (e.g. `break`, `by`, `do`, `end`, `for`, `if`, `proc`, `restart`, `return`, `save`).

◦ *The new worksheet* (or the new problem) it is best to begin with the statement `restart` for cleaning Maple's memory. All examples and problems in the paper assume that they begin with `restart`.

◦ *The statement separators*: semicolon (;) and colon (:). The result of the statement will be displayed with semicolon (;), and it will not be displayed if followed by a colon (:), compare `plot(sin(x), x=0..Pi);` and `plot(sin(x), x=0..Pi):`

◦ *An expression* (`expr`) is a valid statement, and is formed as a combination of constants, variables, operators and functions.

◦ *Data types*: every expression is represented as a tree structure in which each node (and leaf) has a particular data type. For the analysis of any node and branch, the functions `type`, `whattype`, `nops`, `op` can be used.

◦ *A boolean expression* (`bexpr`) is formed with the *logical operators* and the `r`elational operators.

◦ *An equation* (`Eq`) is represented using the binary operator `=`, and has two operands, the left-hand-side `lhs` and the right-hand side `rhs`.

◦ *Inequalities* (`Ineq`) are represented using the relational operators and has two operands, the left-hand-side `lhs` and the right-hand side `rhs`.

◦ *A string* is a sequence of characters having no value other than itself, cannot be assigned to, and will always evaluate to itself. `x := "string";`, and `sqrt(x);` is an invalid function. Names and strings, can be used with `convert` and `printf`.

◦ If you get no response or an incorrect response, you may have entered or executed the statement incorrectly. Do correct the statement or interrupt the computation (the stop button in the Tool Bar menu).

◦ *Types of brackets*: parentheses `(expr)` — grouping, `(x+9)*3`, the function arguments, `sin(x)`; square brackets `[expr]` — lists, `[a,b,c,d]`, vectors, matrices, arrays; curly brackets {`expr`} — sets, {`a,b,c,d`}.

◦ *Types of quotes*: forward-quotes `'expr'` — to delay evaluation of expression, `'x+9+1'`, to clear variables, `x:='x'`; back-quotes `` `expr` `` — to form a symbol or a name, `` `the name:=7`;k:=5; print(`the value of k is`,k) ``; double-quotes `"expr"` — to create strings, a single `"` — to delimit strings.

◦ *Comments* can be included with the sharp sign `#` and all characters following it up the end of a line.

◦ *Maple source code* can be viewed for most of the functions (package functions): `interface(verboseproc=2);readlib(map);readlib(`plots/arrow`);`

◦ *Help system*: Maple contains a complete online help system. You can use: `?name`, `help(name)`, or the `Help` menu, or by highlighting a function and then pressing F1 (in Maple ≥ 9).

◦ *Maple worksheets*: Maple worksheets are files that document a working process and organize it as a collection of expandable groups (see `?worksheet`, `?shortcut`).

◦ *Palettes* can be used for building or editing mathematical expressions without remembering the Maple syntax (`View->Palettes->ShowAllPalettes`).

◦ *The Maplet User Interface* (available in Maple ≥ 8) consists of *Maplet applications* that are collections of windows, dialogs, actions (see `?Maplets`).

◦ *Packages*: in addition to the standard library functions, a number of specialized functions are available in various packages (subpackages) (see `?index[package]`). A package (subpackage) function can be loaded in the form (see `?with`):

```
with(package); function(arguments);
with(package[subpackage]); function(arguments);
```

◦ Previous results (during a session) can be used with symbols `%` (the last result), `%%` (the next-to-last result), `%%...%`, k times, (the kth previous result).

◦ *First steps*: we type the Maple function to the right of the prompt symbol >, and at the end of the command we place a semicolon, and then press `Enter` (or `Shift+Enter` to continue the function on the next line). Maple evaluates, displays the result, and inserts a new prompt.

1.4 Numerical evaluation

◦ Maple gives an exact answer to arithmetic expressions with integers and reduces fractions. When the result is an irrational number, the output has unevaluated form.

◦ Most computers represent both integer and floating point numbers internally using the binary number system. Maple represents the numbers in the decimal number system using a user-specified precision.

◦ *Numerical approximations*: global and local changing a user-specified precision, respectively, with the environment variable `Digits` (see `?Digits`, `?environment`) and with the function `evalf(expr,n)`.

1.5 Predefined constants and functions

◦ *Types of numbers*: integers, fractions, floating-point numbers, complex numbers, e.g. {`-55,5/6,3.4,-2.3e4,Float(23,-45),3-4*I,Complex(2/3,3)`}.

◦ *Predefined constants*: symbols for definitions of commonly used mathematical constants (e.g., `true,gamma,infinity,Pi,I,exp(1)`), see `?ininame`, `?constants`.

◦ *Active and inert functions*: the active functions (beginning with a lower-case letter) is used for computing; the inert functions (beginning with a capital letter) is used for showing steps in solving process (e.g., `diff, Diff`, `int, Int`, `limit, Limit`).

◦ *Library (predefined) functions and user-defined functions. Predefined functions:* most of the well known functions are predefined by Maple and they are known to some Maple functions (e.g., `diff`, `evalc`, `evalf`, `expand`). Numerous special functions are defined.

◦ *Elementary trascendental functions*: the exponential and logarithmic functions, the trigonometric and hyperbolic functions and their inverses,

```
 exp(x);     ln(x); log[b](x); log10(x);   sin(x);   cos(x);
 tan(x);    cot(x);    sec(x);   csc(x);  sinh(x);  cosh(x);
tanh(x);   coth(x);   sech(x);  csch(x);arcsin(x);arccos(x);
 arctan(x); arccot(x);   arcsec(x); arccsc(x);   arcsinh(x);
arccosh(x); arctanh(x); arccoth(x); arcsech(x);  arccsch(x);
```

◦ *Other useful functions:* `max/min`, maximum/minimum values of a sequence of real values; `round, floor, ceil, trunc, frac`, converting real numbers to nearby integers, and the fractional part or real numbers.

1.6 Expressions: evaluation and simplification

◦ *Evaluation*:

```
   subs(x=a, expr);       eval(expr, x=a);
   subs(x=a, y=b, expr);  eval(expr, {x=a, y=b});
```

◦ *Univariate, multivariate polynomials, and polynomials over a number ring and a field*, see `?polynomials`.

◦ *Manipulation with polynomials*: extract polynomial coefficients, `coeff`, `coeffs`, `lcoeff`, `tcoeff`; determine the degree and the lowest degree of a polynomial, `degree`, `ldegree`; group the coefficients of like terms together, `collect`; perform exact polynomial division, `divide`; find exact roots of an univariate polynomial over an algebraic number field, `roots`; sort a polynomial, `sort`; test for polynomials, `type[polynom]`,

```
coeff(p,x,n); coeffs(p,x);  lcoeff(p,x);  tcoeff(p,x);
degree(p,x);  ldegree(p,x); collect(p,x); divide(p,q);
roots(p, x);  sort(p, x);   type(p, polynom);
```

▶ Define the univariate polynomial $y = a_n x^n + \cdots + a_1 x + a_0$ $(n = 1, \ldots, 10)$:

```
n:=10;
y:=add(a||i*x^i,i=0..n); y:=add(cat(a,i)*x^i,i=0..n);
y:=a0+add(cat(a,i)*x^i,i=1..n); y:=a0+sum('cat(a,i)*x^i','i'=1..n);
y:=convert(['a||i*x^i'$'i'=0..n],'+');
a:=array(0..n); S:=a[0];
for i from 1 to n do S:=S + a[i]*x^i; od: y:=S;
```

◦ *Algebraic simplification*: factorize over an algebraic number field, `factor`; distribute products over sums, `expand`; combine terms into a single term, `combine`; evaluate in an algebraic number field, `evala`, `eval`; extract the numerator and the denominator of an expression, `numer`, `denom`; apply simplification rules to expressions, `simplify`; find factored normal form, `normal`; pattern matching, `match`; extension of the domain of computation for many functions, `frontend`; change the form of an expression, `convert`; manipulations of large expressions, `map`, manipulations with trigonometric expressions, `simplify`, `combine`, `testeq`,

```
factor(expr); expand(expr); combine(expr); evala(expr);
numer(expr); denom(f); simplify(expr); normal(expr);
match(expr=pattern,var,'s'); frontend(expr,var);
convert(expr,form,var); map(function,expr);
simplify(expr,trig); combine(expr,trig); testeq(expr1=expr2);
```

```
f := (x^3+2*x^2-x-2)/(x^3+x^2-4*x-4); F:=(x+y+1)^3*(x+3*y+3)^2;
factor(numer(f)); factor(denom(f)); simplify(f);
convert(f,parfrac,x); B:=collect(F,x); C:=map(factor,B);
assume(x>0, y>0); expand([ln(x*y), ln(x^y), (x*y)^z]);
assume(t>0); combine([exp(t)*exp(p), t^p*t^q, ln(t)+ln(p)]);
```

2 Functions and procedures

2.1 Definition, evaluation, and composition of functions

∘ *User-defined functions*: a function in general form `name(args)=expr` is defined as a functional operator (see `?->`). For instance, the functions of one or many variables $f(x)$ =`expr`, $f(x_1,\ldots,x_n)$ =`expr`, the vector functions of one or many variables $f(x)=(x_1,\ldots,x_n)$, $f(x_1,\ldots,x_n)=\langle f_1(x_1,\ldots,x_n),\ldots,f_n(x_1,\ldots,x_n)\rangle$ are defined as follows:

```
f:=x -> expr;  f:=(x1,...,xn) -> expr; f:=x -> [x1,...,xn];
f:=(x1,...,xn) -> [f1(x1,...,xn),...,fn(x1,...,xn)];
```

∘ *Alternative definitions of functions*: with the function `unapply` (that converts an expression to a function) and with the function `proc` (that defines a procedure),

```
f := unapply(expr, x);      f := proc(x) expr end;
```

∘ *Evaluation of functions*:

```
f(a);    subs(x=a, f(x));          eval(f(x),x=a);
f(a, b); subs(x=a, y=b, f(x, y)); eval(f(x, y),{x=a, y=b});
```

∘ *Composition operator*: `@`, e.g. the function compositions $(f_1\circ f_2\circ\cdots\circ f_n)(x)$ or $(f\circ f\circ\cdots\circ f)(x)$, n-times,

```
(f1 @ f2 @f3 @ ... @fn)(x);       (f @@ n)(x);
```

▶ Graph the real roots of the equation $x^3+(a-3)^3x^2-a^2x+a^3=0$ for $a\in[0,1]$:

```
Sol := [solve(x^3 +(a-3)^3*x^2 -a^2*x +a^3 =0, x)];
for i from 1 to 3 do  R || i   := unapply(Sol[i], a):
   print(plot(R || i(a), a = 0..1, numpoints=500));   od:
```

▶ Define the vector function $f(x,y)=\langle\cos(x^2-y^2),\sin(x^2-y^2)\rangle$ and calculate $f(1,2)$, $f(\pi,-\pi)$:

```
f:=(x,y)->[cos(x^2-y^2),sin(x^2-y^2)]; evalf(f(1,2)); f(Pi,-Pi);
```

▶ For the functions $f(x) = x^2$ and $g(x) = x + \sin x$ calculate the function compositions $(f \circ g \circ f)(x)$ and $(f \circ f \circ f \circ f)(x)$:

```
f := x -> x^2;  g := x -> x+sin(x);
f(g(f(x)));     (f@g@f)(x);  f(f(f(f(x))));  (f@@4)(x);
```

2.2 Piecewise continuous functions

∘ *Piecewise continuous functions* can be defined with `piecewise` or as a procedure with the control statement `if`:

```
f:=x -> piecewise(cond1,expr1,expr2);
g:=x -> piecewise(cond1,expr1,cond2,expr2,expr3);
f:=proc(x) if cond1 then expr1 else expr2 fi: end:
g:=proc(x) if cond1 then expr1 elif cond2 then expr2
                               else expr3 fi: end:
```

▶ Define and graph the function $g(x) = \begin{cases} 0, & |x| > 1 \\ 1 - x, & 0 \le x \le 1 \\ 1 + x, & -1 \le x \le 0 \end{cases}$, $x \in [-3, 3]$:

```
g:=x->piecewise(x>=-1 and x <=0,1+x,x>=0 and x<=1,1-x,0);
plot(g(x), x=-3..3);
g := proc(x) if   x >= -1 and x <= 0 then 1+x
             elif x >= 0 and x <= 1 then 1-x
             else 0      fi:      end;
plot(g, -3..3);
```

2.3 Procedures

∘ *A procedure* (see `?procedure`) is a block of statements which one needs to use repeatedly. A procedure can be used to define functions, matrices, graphs, etc.,

```
proc(args) local v1; global v2; options opts; stats; end proc;
```

where `args` is a sequence of arguments, `v1` and `v2` are the names of local and global variables, `opts` are special options (see `?options`), and `stats` are statements that are realized inside the procedure.

◦ *Recursive procedures* which calls itself until a condition is satisfied are defined with the option `remember`:

▶ Calculate the first 3000 Fibonacci numbers and the computation time, where $F(n) = F(n-1) + F(n-2)$, $F(0) = 0$, $F(1) = 1$:

```
Fib := proc(n::integer) option remember;
    if n<=1 then n else Fib(n-1) +Fib(n-2) fi; end;
NF:=NULL: for i from 10 to 40 do NF:=NF, Fib(i); od: NF:=[NF];
ti := time(): Fib(3000); tt := time()-ti;
```

2.4 Control structures

◦ In Maple language there are essentially the *two control structures*: the selection structure `if` and the repetition structure `for`:

```
if  cond1 then expr1 else expr2  fi;
if  cond1 then expr1 elif cond2 then expr2 else expr3  fi;
for  i from i_ini by step to i_last do stats od;
for  i from i_ini by step to i_last while cond1 do stats od;
```

where `cond1`, `cond2` are conditions, `expr1, expr2` are expressions, `stats` are statements, `i, i_ini, i_last` are the loop variable and the initial and the last values of `i`. These operators can be nested. The operators `break`, `next`, `while` inside the loop are used for breaking out of a loop, to proceed directly to the next iteration, or for an additional condition.

▶ Define the function *double factorial* for any integer n: $n!! = n(n-2)(n-4)\cdots(4)(2)$ if n is even, and $n!! = n(n-1)(n-3)\cdots(3)(1)$ if n is odd:

```
N1 := 20; N2 := 41;
DF := proc(N) local P, i1, i; P := 1;
      if modp(n, 2) =0 then i1:=2 else i1:=1 fi:
      for i from i1 by 2 to N do P:=P*i; od: end:
printf(" %7.0f!! = %20.0f", N1, DF(N1));
printf(" %7.0f!! = %20.0f", N2, DF(N2));
```

2.5 Basic objects and operations

◦ *Basic objects or data structures*, sequences, lists, sets, tables, arrays, vectors, matrices, are used for representing more complicated data.

```
Sequence1 := expr1, expr2, expr3, ..., exprn;
Sequence2 := seq(f(i), i=a..b);
List1   := [Sequence1]; Set1 := {Sequence1};
Table1  := table([expr1 = A1,..., exprN = AN]);
Array1  := array(n..m);
Vec1 := array(1..n, [al, al, ..., an]);
Vec2 := Vector(<a1,a2,a3,...,an>);
Mat1 := array(1..n,1..m,[[a11,...,alm],...,[an1,...,anm]]);
with(linalg): Mat2 := matrix(n, m, [a11, a12, ..., anm]]);
Mat_3:=Matrix(<<a11,a21,a31>|<a12,a22,a32>|<a13,a23,a33>>);
Mat4 := Matrix([[a11,a12],[a21,a22],[a31,a32]]);
```

◦ *Sequences, lists, sets*: are groups of expressions. Maple preserves the order and repetition in sequences and lists and does not preserves in sets. The order in sets can change during a Maple session. *A table* is a group of expressions represented in tabular form. Each entry has an index (an integer or any arbitrary expression) and a value (see `?table`). *An array* is a table with integer range of indices (see `?array`). In Maple arrays can be of any dimension (depending of computer memory). *A vector* is a one-dimensional array with integer positive range of indices (see `?vector`, `?Vector`). *A matrix* is a two-dimensional array with integer positive range of indices (see `?matrix`, `?Matrix`):

```
Seq1 := x, y, z, a, b, c;
List1 := [1,sin(x),cos(x),sin(2*x),cos(2*x),sin(3*x),cos(3*x)];
Set1  := {x, y, z};  Arr1:= array(-1..3);
Arr2 := array(1..4, [1, 2, 3, 4]); Arr3 := array([1, 2, 3, 4]);
```

◦ *Basic operations with objects.*

◦ *Create the empty structures*, `NULL`, `:=`, `[ ]`:

```
Seq1  := NULL;  List1 := [ ]; List_2 := NULL;  Set1 := {};
Tab1  := table();    Array1  := array(-10..10);
Vec1  := vector(10); Matrix1 := matrix(10, 10);
```

◦ *Concatenate two structures*, `||`, `op`, `[ ]`, `cat`:

```
Seq3   := Seq1 || Seq2; Seq3 := cat(Seq1, Seq2);
List 3 := [op(List1, List2)];
```

◦ *Extract an i'th element from a structure*, `[ ]`, `op`, `select, has`:

```
List1[i];      op(i, List1);   Array1[i, j];
op(i, Set1);   select(has, Set1, element);
```

∘ *Determine the number of elements in a structure*, `nops`:

```
nops(List1);   nops(Set1);
```

∘ *Create a substructure*, `op`, `[ ]`:

```
List2 := [op(n1..n2, List1)];   List2 := List1[n1..n2];
```

where $n_1 \leq n_2 \leq n$ and n is a number of elements of `List1`.

∘ *Determine total structure inside the other structure*, `op`, `evalm`, `print`:

```
op(List1);  evalm(Matrix1);  print(Array1);
```

∘ *Replace the i'th element of a structure*, `:=`, `[ ]`, `subsop`, `subs`, `evalm`:

```
List1[i] := val;  subsop(i=val, List1);
A1 := subs(A[i,j]=a+b, evalm(A));
```

∘ *Insert an element or some elements into a structure*, `[ ]`, `op`:

```
List2:=[op(List1), A1];  List2:=[A1, op(List1)];
List3:=[op(n1..n2,List1), A1, A2, A3, A4, op(List2)];
```

∘ *Create a structure according to a formula or with some properties* (e.g., `symmetric`, `identity`, `diagonal`, `sparse`), `seq`, `matrix`, `vector`, `array`:

```
List1   := [seq(f(i), i = n..m)];
Matrix1 := matrix(2, 2, (i,j) -> i+j);
Vector1 := vector(2, i -> i^2);
```

∘ *The union and intersection of sets, removing elements of sets, the sum, difference, multiplication, division, scalar multiplication of matrices*, `union`, `intersect`, `minus`, `remove,has`, `evalm`, `&*`:

```
Set3 := Set1 union Set2; Set3 := Set1 intersect Set2;
Set3 := Set1 minus Set2; Set2 := remove(has, Set1, A1);
Mat3 := evalm(M1 &* M2);
```

∘ *Apply a function to each element of a structure*, `map`:

```
Set2 := map(func, Set1);
```

▶ Observe the function behavior $y(x) = \cos(6(x - a\sin x))$, $x \in [-\pi, \pi]$, $a \in (\frac{1}{2}, \frac{3}{2})$:

```
with(plots): y := x -> cos(6*(x-a*sin(x))); G:=NULL; N:=20;
for i from 0 to N do a:=1/2+i/N;  G:=G,plot(y(x),x=-Pi..Pi); od:
G := [G]:  display(G, insequence=true);
```

3 Graphic visualization

3.1 Simple graphs and various options

∘ Graphs of real values of `expr` or the functions `f(x)`, `f(x, y)`, $x \in [x_1, x_2]$, $(x, y) \in [x_1, x_2] \times [y_1, y_2]$:

```
f:=expr;         plot(f, x1..x2, opts);
f:=x->expr;      plot(f(x), x= x1..x2, opts);
f:=(x,y)->expr; plot3d(f(x,y),x=x1..x2, y=y1..y2, opts);
```

∘ All the graphs can be drawn with various versions of `plot` and the package `plots`. The function `plot` has various forms (e.g., `logplot`, `odeplot`, `plot3d`, etc.) and various optional arguments which define the final figure (see `?plot[options]`, `?plot3d[options]`), e.g., light setting, legends, axis control, titles, gridlines, real-time rotation of 3D graphs, wide variety of coordinate systems, etc.

```
g := x ->  exp(-(x-3)^2*cos(Pi*x)^2);
plot(g(x), x=0..6, tickmarks=[4, 4], title=‘Graph of g(x)‘);
plot(tan(x), x=-2*Pi..2*Pi, y=-4..4, discont=true);
Points := [[1,2],[2,3],[3,5],[4,7],[5,11],[6,13],[7,17],[8,19]];
plot(Points, style = point); plot(Points, style = line);
```

∘ The global options for 2D and 3D graphs can be introduced with `setoptions`, `setoptions3d` (see `?plots`):

```
with(plots);
setoptions(axes=boxed, title=‘graph of f(x)‘);
setoptions3d(axes=normal, title=‘graph of g(x, y)‘);
```

3.2 Multiple graphs

∘ *A list or a set of functions in the same figure*:

```
plot([f1(x), f2(x), ..., fn(x)], x= a..b, options);
plot({f1(x), f2(x), ..., fn(x)}, x= a..b, options);
```

```
f := x -> sin(x)/x;      g := x -> cos(x)/x;
plot([f(x),g(x)], x=0..10*Pi, y=-1..2, linestyle=[SOLID,DOT],
      color=[red, blue]);
```

◦ *Merging various saved graphic structures* (`display/display3d`, `plots`):

```
with(plots);
f := x -> abs(sin(x));        g := x -> -cos(x);
G1 := plot(f(x), x=-Pi..Pi): G2 := plot(g(x), x=-Pi..Pi):
display({G1, G2}, title= "f(x) and g(x)");
```

3.3 Text in Graphs

◦ *Drawing text strings on 2D and 3D graphs*:

```
textplot([[x1, y1, String1], ...,  [xn, yn, Stringn]], opts);
textplot3d([[x1,y1,z1,String1],...,[xn,yn,zn,Stringn]],opts);
```

```
with(plots):  f := x -> 4*x^3 + 6*x^2 -9*x +2;
G21 := plot([f(x), D(f)(x), (D@@2)(f)(x)], x=-3..3):
G22 := textplot([1.2, 100, "f(x) and their derivatives"],
      font=[HELVETICA, BOLD, 13], color=plum): display([G21, G22]);
```

3.4 Special graphs

◦ *Coordinate lines*:

```
with(plots): conformal(z, z=z1..z2, opts):
coordplot(coordsystem, [xrange, yrange], opts);
```

► Graph $f(x) = x\sin(1/x)$ and $g(x) = \dfrac{x^2 - x + 1}{x^2 + x - 1}$ together with the corresponding coordinate lines:

```
with(plots):    A := 4:  f := x -> sin(1/x)*x;
G1:=plot(f(x), x=-Pi/2..Pi/2, color=blue, thickness=3):
M1:=conformal(z, z=-A-I..A+I, grid=[20,10], color=grey):
display([G1,M1]);
```

◦ *Bounded regions* (see `?plot[options]`, `inequal`):

```
plot(f(x), x=a..b, filled=true);
with(plots): inequal(ineqs, x=a..b, y=c..d, opts);
```

▶ Graph the region that satisfy the inequality $2x - 2y > 1$:

```
with(plots): Inequal := x -> 2*x-2*y>1;
A:=(color=blue); B:=(color=grey); C:=(color=green, thickness=10);
inequal(Inequal(x), x=-2..2, y=-2..2,
        optionsfeasible=A, optionsexcluded=B, optionsopen=C);
```

◦ *Logarithmic graphs*:

```
with(plots):                  logplot(f,range,opts);
semilogplot(f,range,opts);    loglogplot(f,range,opts);
```

```
with(plots); with(stats): al := stats[random, normald](20);
Points := [seq([0.2*i, exp(0.1*i)+0.1*al[i]], i=1..20)];
G1 := logplot(Points, style=point, color=red):
G2 := logplot(x+sin(x), x=0.5..3, style=line, color=green):
display({G1, G2});
```

◦ *2D and 3D parametric curves*:

```
      plot([x(t), y(t), t = t1..t2], horzl, vert, opts);
with(plots); spacecurve([x(t),y(t),z(t)],t=t1..t2,opts);
```

```
plot([t^2*sin(t), t^3*cos(t), t=-10*Pi..10*Pi], axes=boxed);
with(plots):x:=t->-1/2*cos(3*t);y:=t->-1/4*sin(3*t);z:=t->1/7*t;
spacecurve([x(t),y(t),z(t)],t=0..10*Pi,numpoints=400);
```

◦ *Tubes and knots around 3D parametric curve*:

```
with(plots): tubeplot([x(t),y(t),z(t)],t=t1..t2,
             radius=r,tubepoints=m,numpoints=n,opts);
```

```
with(plots): tubeplot([2*sin(t),cos(t)-sin(2*t),cos(2*t)],t=0..2*Pi,
         axes=boxed,radius=0.25,numpoints=100,scaling=constrained);
```

◦ *Conformal transformation* $z = f(Z)$ on a rectangular region and the Riemann sphere:

```
with(plots): f := z-> (5*z-1)/(5*z+1); z1 :=-1-2*I; z2 :=1+2*I;
conformal(f(z),z=z1..z2,-5-5*I..5+5*I,grid=[25,25],numxy=[140,140]);
conformal3d(cos(z)-sin(z), z=0-2*I..2*Pi+2*I, color=white,
                    grid=[20, 20], orientation=[17, 111]);
```

◦ *A density plot* of $f(x,y)$, $(x,y) \in [x_1,x_2] \times [y_1,y_2]$:

```
with(plots): densityplot(f(x,y),x=x1..x2,y=y1..y2,opts);
```

► Construct the density plot of $f(x,y) = xe^{-x^2-y^2}$, $(x,y) \in [-2,2] \times [-2,2]$, with color gradient and the corresponding legend:

```
with(plots):
A:=colorstyle=HUE,style=patchnogrid,numpoints=5000,axes=boxed;
G1:=densityplot((x,y)->x*exp(-x^2-y^2),-2..2,-2..2, A):
G2:=densityplot((x,y)->0.2*y, 3..3.5,-2..2, A): G3:=textplot(
    [seq([3.8,-1.95+i/8*3.9,sprintf("%.1f",-0.4+i/10)],i=0..8)]):
display({G1, G2, G3}, scaling=constrained);
```

3.5 Level curves and surfaces

◦ *Level curves and surfaces, graphs of implicit functions*:

```
with(plots): contourplot(f(x,y), x=x1..x2, y=y1..y2,opts);
contourplot3d(f(x,y,z),x=x1..x2,y=y1..y2, opts);
implicitplot(f(x, y)=c, x=x1..x2,y=y1..y2,opts);
implicitplot3d(f(x,y,z)=c,x=x1..x2,y=y1..y2,z=z1..z2,opts);
```

► Graph $h(x,y) = \dfrac{x-y}{x^2+y^2}$ and some level curves $(-2,-4,-6)$:

```
with(plots): h:=(x,y)->(x-y)/(x^2+y^2); R:=-4..4; C:=[-2,-4,-6];
plot3d(h(x,y),x=R,y=R,grid=[20,20],orientation=[15,67]);
contourplot(h(x,y),x=R,y=R,grid=[50,50],axes=boxed,contours=C);
```

3.6 Surfaces in the space

◦ *Surface parametrization*, $x = x(u,v)$, $y = y(u,v)$, $z = z(u,v)$, *implicit equation*, $f(x,y,z) = c$:

```
with(plots):
plot3d([x(u,v),y(u,v),z(u,v)],u=u1..u2,v=v1..v2,opts);
implicitplot3d(f(x,y,z)=c,x=x1..x2,y=y1..y2,z=z1..z2,opts);
```

▶ Graph the ellipsoid $\frac{x^2}{16}+\frac{y^2}{4}+z^2=1$:

```
with(plots):
x:=(u,v)->3*cos(u)*cos(v);y:=(u,v)->2*cos(u)*sin(v);
z:=(u,v)->sin(u);
plot3d([x(u,v),y(u,v),z(u,v)],u=-Pi/2..Pi/2,
  v=-Pi..Pi,axes=boxed, scaling=constrained, orientation=[58,60]);
```

3.7 Arrays and points in the space

```
with(plots):              pointplot(points, opts);
pointplot3d(points,opts); matrixplot(matrix,opts);
```

▶ Graph the points $(0,4,2)$, $(0,3,2)$, $(0,-1,1)$, $(0,3,1)$, $(0,-5,0)$, $(0,-1,0)$:

```
with(plots):
pointplot3d({[0,4,2],[0,3,2],[0,-1,1],[0,3,1],[0,-5,0],[0,-1,0]},
             axes=boxed,symbol=circle,symbolsize=20,color=green);
```

3.8 Various coordinate systems

◦ Curves in polar coordinates:

```
plot([r(t),theta(t),t=t1..t2],coords=polar,opts);
with(plots):polarplot([r(t),phi(t),phi=phi1..phi2],opts);
```

▶ Lissajous curves are defined by the parametric equations $x(t)=\sin(nt)$, $y(t)=\cos(mt)$, $t\in[0,2\pi]$, where (n,m) are different coprimes. Observe the forms of the *Lissajous curves* for various (n,m):

```
with(plots):
G:=[seq(plot([sin(ithprime(i)*t),cos(ithprime(i+1)*t),t=0..2*Pi],
    scaling=constrained),i=1..10)]:
display(G,insequence=true);
```

◦ *Surfaces in cylindrical and spherical coordinates*:

```
with(plots): Rtheta:=theta1..theta2; Rr:=r1..r2;
cylinderplot(f(r,theta), theta=Rtheta,r=Rr,opts);
cylinderplot([r(theta),theta,r],theta=Rtheta,r=Rr,opts);
sphereplot(f(z,r),z=z1..z2,r=Rr,opts);
sphereplot([z(theta),theta,z],z=z1..z2,theta=Rtheta,opts);
```

3.9 Vector fields

```
 with(plots):
 fieldplot([f(x,y),g(x,y)],x=x1..x2,y=y1..y2,opts);
 gradplot(f(x,y),x=x1..x2,y=y1..y2,opts);
 fieldplot3d([f(x,y,z),g(x,y,z),h(x,y,z)],
              x=x1..x2,y=y1..y2,z=z1..z2,opts);
 gradplot3d(f(x,y,z),x=x1..x2,y=y1..y2,z=z1..z2,opts);
```

```
with(plots): Rx=-5..5; Ry=-4*Pi..4*Pi;
contourplot(x*cos(y)+4*x-sin(y),x=Rx,y=Ry,grid=[50,50]);
fieldplot([x*sin(y)+cos(y),cos(y)+4], x=Rx, y=Ry,
                      grid=[30,30],arrows=slim,color=x);
fieldplot3d([x-20*y+20*z,x-4*y+20*z,x-4*y+20*z],x=-4..4,
  y=-8..8,z=-8..8,grid=[9,9,9],arrows=thick,scaling=constrained);
```

3.10 Animations

```
 with(plots):  animatecurve(f(x), x=a..b, opts);
 animate(f(x,t), x=a..b, t=t1..t2, opts);
 animate3d(f(x,y,t), x=a..b, y=c..d, t=t1..t2, opts);
 display([G1, G2, ..., GN],   insequence=true);
 display3d([G1, G2, ..., GN], insequence=true);
```

▶ Observe the Lissajous curves in polar coordinates:

```
with(plots):
animatecurve([sin(7*x),cos(11*x),x=0..2*Pi],coords=polar,
     numpoints=300, frames=300, color=blue, thickness=3);
```

4 Solving equations

4.1 Exact solutions

∘ *Exact solutions* of algebraic equations or systems of equations, `solve`; representations for roots of equations, `RootOf`, `allvalues`; isolating a subexpression to left side of an equation,

```
solve(Eq, var);   solve({Eq1, Eq2}, {var1, var2});
RootOf(expr, x); allvalues(expr, opts); isolate(eqn, expr);
```

▶ Find exact solutions of $3x + 11 = 5$, $\cos^2 x - \cos x - 1 = 0$. Solve the equation $V = \pi r^2 h$ for h. Find the inverse function of $f(x) = \dfrac{2x-3}{1-5x}$:

```
solve(3*x+11=5,x);          solve(cos(x)^2-cos(x)-1=0,x);
solve(v=Pi*r^2/h, h);       isolate(v=Pi*r^2/h, h);
F_inv := unapply(solve(y=(2*x-3)/(1-5*x), x), y); F_inv(x);
```

4.2 Numerical approximations

∘ *Numerical solutions of algebraic and trascendental equations*:

```
evalf(solve(Eq, var));       fsolve(Eq, var, opts);
fsolve(Eq, var=a..b, opts);  fsolve(Eq, var, complex);
```

▶ Approximate the values of x that satisfy the equation $x^5 - 2x^2 = 1 - x$. Find a numerical approximation to the solution of the equation $\sin x = x/2$ for $x \in (0, \pi]$:

```
map(evalf,[solve(x^5-2*x^2=1-x, x)]);
fsolve(sin(x)=x/2, x=0..Pi, avoid={x=0});
```

4.3 Analytical approximations

∘ In Maple there is no single function for finding approximate analytical solutions to equations, so various methods can be applied or developed new methods.

► Find approximate analytical solutions to the equation $F(x;\varepsilon)=0$, where F is a real function of x and a small parameter ε. According to the regular perturbation theory, the equation $F(x;0)=0$ has a solution x_0 and the solution of the perturbed equation is near x_0 and can be represented as a power series of ε. For instance, solve the equation $x^2-ax+\varepsilon=0$, $|\varepsilon|\ll 1$. If $\varepsilon=0$, the roots are $x_{01}=0$ and $x_{02}=a$. If $\varepsilon\to 0$, we have $X_1(\varepsilon)\to 0$, $X_2(\varepsilon)\to a$. If $|\varepsilon|\ll 1$, the roots are:

$$X_k = x_{0k}+\varepsilon x_{1k}+\cdots+\varepsilon^i x_{ik}+\cdots, \quad k=1,2, \quad i=2,3,\ldots,$$

where x_{ik} are the unknown coefficients to be determined. Substituting the series X_k into the original equation and matching the coefficients of like powers of ε, we arrive at the system of algebraic equations for the i th approximation, which can be solved for x_{ik}:

```
RegPertPoly := proc(Expr, var, param)
 global Sers: local i, j, Expr1, X, y, k, m:
 y  := var[0];  Expr1:= collect(Expr(y), param);
 X[0] := [solve(coeff(Expr1, param, 0), var[0])];
 k  := nops(X[0]);
 for j from 1 to k do  y[j] := X[0][j];
  for i from 1 to n do
    y[j]:=y[j]+param^i*var[i]; Expr1:=Expr(y[j]);
    X[i]:=solve(coeff(Expr1, param, i), var[i]);
    y[j]:=X[0][j] + add(X[m]*param^m, m=1..i); od:
  Sers := sort(convert(y, list)): od:
 RETURN(Sers): end:
n:=5; Eq:=x->x^2-a*x+epsilon; RegPertPoly(Eq,x,epsilon);
```

4.4 Differential equations

∘ In Maple there exists a large set of functions to solve (analytically, numerically, graphically) ordinary and partial differential equations or systems of differential equations.

∘ *Exact solutions to ordinary differential equations* (see `?dsolve`): find closed form solutions for a single ODE or a system of ODEs, solve ODEs or a system of them with given initial conditions, find formal power series solutions to a homogeneous linear ODE with polynomial coefficients, find series solutions to ODEs problems, find solutions using integral transforms, etc.

```
dsolve(ODE,y(x),opts); dsolve({ODEs},{funcs});
dsolve({ODEs,ICs},{funcs},opts);
dsolve(ODE,y(x),'formal_series','coeffs'=coeff_type);
dsolve({ODEs,ICs},{funcs},'series');
dsolve({ODEs,ICs}, {funcs}, method=transform, opts);
```

◦ *Explicit, implicit forms of the exact solutions, graphs of solutions*:

```
Sol_Exp := dsolve(diff(y(t),t)+t^2/y(t)=0, y(t));
Sol_Imp := dsolve(diff(y(t),t)+t^2/y(t)=0, y(t), implicit);
with(plots): G := subs({y(t)=y}, lhs(Sol_Imp));
Gs:= seq(subs(_C1=i, G), i=-5..5);
contourplot({Gs}, t=-5..5, y=-10..10, color=blue);
```

◦ *ODE classification and solution methods suggestion* (`?DEtools`, `?odeadvisor`): e.g. separation of variables, homogeneous equations, series solutions, exact equations, linear equations, etc.

```
with(DEtools): ODE1 := diff(y(t),t)=y(t)*sin(t)^2/(1-y(t));
Sol_ex:=dsolve(ODE1,y(t)); Sol_Imp:=dsolve(ODE1,y(t),implicit);
odeadvisor(ODE1);  Sol_sep:=separablesol(ODE1, y(t));
ODE2 := (t^2-y(t)*t)*diff(y(t),t)+y(t)^2=0; odeadvisor(ODE2);
Sol1 := dsolve(ODE2, y(t)); Sol2 := genhomosol(ODE2, y(t));
Sol_ser:= dsolve(ODE1,y(t),'series');
```

◦ *Higher order ODE*: exact and numerical solutions, and their graphs

```
with(plots): setoptions(scaling=constrained,numpoints=200);
ODE:=diff(x(t),t$2)-diff(x(t),t)+(t-1)*x(t)=0;
ICs:=D(x)(0)=0,x(0)=1; Sol_ex := dsolve({ODE,ICs}, x(t));
Sol_num:=dsolve({EDO,ICs}, x(t), numeric);
G :=array(1..3);
G[1]:=odeplot(Sol_num, [t, x(t)],0..10,color=blue):
G[2]:=plot(rhs(Sol_ex), t=0..10, color=red):
G[3]:=odeplot(Sol_num,[x(t),diff(x(t),t)],0..10,color=magenta):
display(G);
```

◦ *Systems of ODEs*: exact solutions, laplace transforms, etc.

```
with(DEtools):  with(inttrans): ODE_s1:={D(x)(t)=-2*x(t)+5*y(t),
 D(y)(t)=4*x(t)-3*y(t)}; Sol_s1:=dsolve(ODE_s1, {x(t),y(t)});
A1 := array([[-2,5],[4,-3]]); matrixDE(A1,t);  ODE_s2 :=
  {diff(x(t),t)=-y(t)+cos(2*t),diff(y(t),t)=5*x(t)+2*sin(2*t)};
Eq1:=laplace(ODE_s2,t,p); Eq2:=subs({x(0)=2,y(0)=0},Eq1);
Eq3:=solve(Eq2,{laplace(x(t),t,p),laplace(y(t),t,p)});
Sol_sys2:=invlaplace(Eq3,p,t); assign(Sol_s2):
plot([x(t),y(t)],t=-3..3);
```

◦ *Numerical and graphic solutions to ordinary differential equations*: find numerical solutions to ODEs problems, `dsolve[numeric]`, graphs or animations of 2D and 3D solution curves obtained from the numerical solution, `odeplot`, phaseportraits for a system of first order differential equations or a single higher order differential equation with initial conditions, `phaseportrait`, vector fields for autonomous systems of first order differential equations, `DEplot`, etc.

```
with(plots); with(DEtools);
dsolve({ODEs}, numeric, {funcs}, opts);
dsolve(numeric, {funcs}, procopts, opts);
NS := dsolve({ODEs},numeric,{funcs},opts);
odeplot(NS, {funcs}, range, opts);
phaseportrait({ODEs}, {funcs}, range, {ICs}, opts);
DEplot({ODEs}, {funcs}, trange, opts);
```

```
with(linalg): with(DEtools): with(plots): with(student):
A:=array([[1,3],[-2,1]]): eigenvectors(A); matrixDE(A,t);
ODE_sys:=equate(array([[diff(x(t),t)],[diff(y(t),t)]]),
                        A &* array([[x(t)],[y(t)]]));
Sol_sys:=dsolve(ODE_sys,{x(t),y(t)}); assign(Sol_sys);
Curves:= {seq(seq(subs({_C1=i,_C2=j},[x(t),y(t),t=-3..3]),
                         i=-3..3),j=-3..3)}:
G1:=plot(Curves,view=[-10..10,-10..10],color=blue,axes=boxed):
unassign('x','y');
G2:=DEplot(ODE_sys,[x(t),y(t)],t=-3..3,x=-10..10,
                                 y=-10..10,color=red):
display({G1, G2});
```

▶ Construct a phaseportrait of the second-order dynamical system

$$\dot{v} = -\nu v + \varepsilon u[\delta_1 + \tfrac{1}{4} - \tfrac{1}{2}\phi_1 P(u,v) + \tfrac{1}{4}\phi_2 P(u,v)^2],$$
$$\dot{u} = -\nu u + \varepsilon v[-\delta_1 + \tfrac{1}{4} + \tfrac{1}{2}\phi_1 P(u,v) - \tfrac{1}{4}\phi_2 P(u,v)^2],$$

that describes the nonlinear motion of a fluid under conditions of subharmonic resonance and has been obtained by Shingareva (1995) applying averaging transformations with Maple. Here $P(u,v) = u^2 + v^2$, ν is the fluid viscosity, ε is the small parameter, ϕ_1 and ϕ_2 are, respectively, the second and the fourth corrections to the nonlinear frequency, δ_1 is the off-resonance parameter. For different regions where there exists a stable solution, we will have the corresponding phaseportraits, for instance:

```
with(plots):  with(DEtools):
delta_1:=-1/2:  phi_1:=1; phi_2:=1; nu:=0.005; epsilon:=0.1;
Ec1:=D(v)(t)=-nu*v(t)+epsilon*u(t)*(delta_1 + 1/4
             -phi_1/2*(u(t)^2+v(t)^2)+phi_2/4*(u(t)^2+v(t)^2)^2);
Ec2:=D(u)(t)=-nu*u(t)+epsilon*v(t)*(-delta_1 + 1/4
             +phi_1/2*(u(t)^2+v(t)^2)-phi_2/4*(u(t)^2+v(t)^2)^2);
Ecs:=[Ec1, Ec2]; CI :=[[0,0,1.1033],[0,0,-1.1033],[0,1.1055,0],
     [0,-1.1055,0],[0,0,1.613],[0,0,-1.613]]; var := [v(t), u(t)];
Opts:=arrows=medium,dirgrid=[20,20],stepsize=0.1,thickness=2,
      linecolour=blue, color=green;
phaseportrait(Ecs, var, t=-48..400, CI, Opts);
```

∘ *Analytical solutions to partial differential equations*: find analytical solutions for a partial differential equation (PDE) and systems of `PDEs`, `pdsolve`, declare functions and derivatives on the screen for a simple, compact display, `declare`, split into cases and sequentially decouple a system of differential equations, `casesplit`, determine under what conditions it is possible to obtain a complete solution through separation of variables, `separability`, etc.

```
with(PDEtools); declare({funcs}; ON; pdsolve(PDE,f,HINT,build);
pdsolve({PDEs},{fs},HINT);casesplit({PDEs});separability(PDE,f);
```

where `HINT=arg` are some hints, with `build` can be constructed an explicit expression for the indeterminate function `f`.

▶ Find a general solution to the wave equation $u_{tt} = c^2 u_{xx}$:

```
with(PDEtools); declare(u(x,t)); ON;
pde:=diff(u(x,t),t$2)=c^2*diff(u(x,t),x$2);
casesplit(pde); separability(pde,u(x,t)); pdsolve(pde,build);
```

∘ *Numerical and graphic solutions to PDEs* (see `?pdsolve[numeric]`): find numerical solutions to `PDE` or a system of `PDEs`. The solution obtained is represented as a module (similar to a procedure) which can be used for obtaining visualizations (`plot`,`plot3d`, `animate`, `animate3d`) and numerical values (`value`)

```
with(PDEtools): pdsolve({PDEs},{ICsBCs},numeric,{vars},opts);
Sol:=pdsolve({PDEs},{ICsBCs},numeric,{vars},opts);
Sol:-animate(var,t=t0..t1,x=x0..x1);Sol:-plot3d(var,t=t0..t1);
Num_vals := Sol :- value(); Num_vals(num1, num2);
```

where `ICsBCs` are initial and boundary conditions, `vars` are dependent variables.

▶ Solve (numerically and graphically) the initial boundary value problem for the wave equation in the domain $\mathcal{D} = \{0 < x < 1, 0 < t < \infty\}$: $u_{tt} = \frac{1}{100} u_{xx}$, $u(0,t) = 0$, $u(1,t) = 0$, $u(x,0) = 0$, $u_t(x,0) = \sin(2\pi x)$:

```
with(VectorCalculus): with(plots): with(PDEtools):
Eq:={diff(u(x,t),t$2)-0.01*Laplacian(u(x,t),'cartesian'[x])=0};
BC:={u(0,t)=0,u(1,t)=0}; IC:={D[2](u)(x,0)=sin(2*Pi*x),u(x,0)=0};
Opts:=spacestep=1/100, timestep=1/100;
Sol:=pdsolve(Eq, IC union BC, numeric, u(x,t), Opts);
Sol:-animate(u(x,t),t=0..5*Pi,x=0..1,frames=30,numpoints=100,
             thickness=3,color=blue);
Sol:-plot3d(u(x,t),t=0..5*Pi,shading=zhue,axes=boxed); Sol:-value();
Num_vals := Sol:-value();  Num_vals(1/2, Pi);
```

5 Calculus problems

◦ A lot of calculus functions are contained in the packages `student`, `Student`, `Student[Calculus1]`, `VectorCalculus`. The package `Student` contains functions covering the basic material of a single-variable calculus course, and with the package `VectorCalculus` can be performed multivariate and vector calculus operations.

5.1 Differential and Integral calculus

◦ *The limit* of $f(x)$ when x tends to x_0, *the derivatives* of $f(x)$ with respect to x:

```
limit(f(x), x=x_0); diff(f(x), x);  Diff(f(x), x); D(f)(x);
diff(f(x), x$n); Diff(f(x), x$n); (D@@n)(f)(x);
```

▶ Graph $f(x)=\dfrac{x^3-9x}{x^3-x}$, $x \in [-2,2]$, and evaluate $\lim\limits_{x\to 0} f(x)$, $\lim\limits_{x\to \pm 1} f(x)$:

```
f:=x->(x^3-9*x)/(x^3-x); plot(f(x),x=-2..2,-100..100,discont=true);
limit(f(x), x=0);  limit(f(x), x=1); limit(f(x), x=-1);
```

▶ Let $f(x) = x^3 - 4x^2 + 8x - 2$. Calculate $f'(x)$, $f'(\frac{7}{3})$, find the value of x for which $f'(x) = 10$:

```
f:=x->x^3-4*x^2+8*x-2; D(f)(x); D(f)(7/3); solve(D(f)(x)=10,x);
```

◦ *Integral calculus*: construction of Riemann sums, `leftbox, rightbox`, `leftsum, rightsum`; evaluation of indefinite and definite integrals, `int, Int`, `changevar`, `intparts`, `value`; approximations of definite integrals, `simpson`, `trapezoid`, etc.

```
with(student); with(Student); with(Student[Calculus1]);
leftbox(f(x), x=a..b, opts); rightbox(f(x), x=a..b, opts);
leftsum(f(x), x=a..b,n); rightsum(f(x), x=a..b, n);
int(f(x), x);  int(f(x), x=a..b, opts); I1 := Int(f(x), x);
changevar(f(x)=u, I1); intparts(I1, u); value(I1);
simpson(f(x), x=a..b, n); trapezoid(f(x), x=a..b, n);
```

▶ Evaluate the indefinite and definite integrals:

```
f :=x->exp(-x)*cos(x); I1:=int(f(x),x=0..sqrt(Pi)); evalf(I1);
F:=x->exp(2*x)*sin(2*x); factor(int(F(x),x));
evalf(int(cos(4*x)*exp(-4*x^2),x=-2*Pi..2*Pi));
```

5.2 Series

◦ *Manipulation of power series*: calculation of power series sums, `sum`; the environment variable `Order`; generalized series expansion, `series`; the Taylor polynomials, `convert,polynom`; Taylor and Maclaurin series expansion, `taylor`; multivariate Taylor series expansion, `mtaylor`; a coefficient in the (multivariate) Taylor series, `coeftayl`; formal power series package, `powerseries`; multiplicative inverse of a formal power series, `inverse`; multiplication of power series, `multiply`, etc,

```
sum(f(i),i); sum(f(i),i=a..b); Order; series(f(x),x=a,n);
convert(ser,polynom);taylor(expr,x=a,n);taylor(expr,x=0,n);
mtaylor(expr, vars, n); coeftayl(expr, eqn, k);
with(powseries): inverse(expr); multiply(ser1, ser2);
```

► Calculate the series sum $\sum_{n=1}^{\infty} \frac{1}{2n^2+9n+10}$, $\sum_{n=1}^{\infty} x^{kn}$, $\sum_{n=1}^{10000} \frac{\cos n}{n}$:

```
sum(1/(2*n^2+9*n+10), n=1..infinity);  sum(x^(k*n), n=1..infinity);
Ser := n -> cos(n)/n; Points := [seq([i,Ser(i)],i=7000..10000)]:
plot(Points,style=POINT,color=blue,symbol=circle,symbolsize=10);
evalf(sum(Ser(i), i=1..10000));
```

◦ There is no single function in Maple for finding an expansion of a function in terms of a set of complete functions, or one of the simplest class of the Fourier expansions, an expansion in terms of the trigonometric functions $1, \cos(x)$, $\sin(x)$, $\cos(2x)$, $\sin(2x), \ldots$, or their complex equivalents $\phi_n = e^{-inx}$ $(n = 0, \pm 1, \pm 2, \ldots)$, which are complete and orthogonal on the interval $(-\pi, \pi)$ (or any interval of length 2π).

► Approximate the square wave waveform of period 2π:

```
h := Heaviside(t-Pi); plot(h, t=0..2*Pi, scaling=constrained);
A := i -> 1/Pi*int(h*cos(i*t), t=0..2*Pi);
B := i -> 1/Pi*int(h*sin(i*t), t=0..2*Pi);
A0:= 1/(2*Pi)*int(h, t=0..2*Pi);
F := n -> evalf(A0+sum(B(i)*sin(i*t), i=1..n));
plot(F(30), t=0..2*Pi, scaling=constrained,color=blue);
```

5.3 Multivariate calculus

◦ Numerous functions for performing multivariate calculus are contained in the packages `VectorCalculus`, `linalg`, `LinearAlgebra`: multivariate functions, `->`; par-

tial differentiation, `diff, Diff`; multiple integrals, `Doubleint`, `Tripleint`; iterative integrals, `int`; relative extrema, `extrema`; vector calculus, `Gradient`, `Nabla`, `Laplacian`, `VectorField`, `Diverge`, `Curl`, `DotProduct`, etc.,

```
with(student); with(VectorCalculus); with(linalg);
with(LinearAlgebra); f := (x,y) -> expr;
D(f); D[i](f); D[i,j](f); D[i](D[j,i](f));
diff(f(x1,...,xn),x1,...,xn); diff(f(x1,...,xn), x1$n);
int(...int(int(f, x1), x2)..., xn); extrema(expr,cond,vars);
value(Doubleint(f(x,y), x=x1..x2, y=y1..y2));
value(Tripleint(f(x,y,z),x=x1..x2, y=y1..y2, z=z1..z2));
Gradient(expr,[x,y]);Nabla(expr,[x,y]);Laplacian(expr,[x,y]);
VF := VectorField(<x,y>, coordsys);  Diverge(VF); Curl(VF);
DotProduct(<x,y>, <x,y>);  <x,y> . <x,y>;
```

▶ Let $f(x,y,z) = -(xy)^2\,\mathbf{i} + \cos^2(xyz)\,\mathbf{j} + \sin^2 z\,\mathbf{k}$. Find $\operatorname{curl} f$, $\operatorname{div} f$, $\Delta(\operatorname{div} f)$, $\operatorname{grad}(\Delta(\operatorname{div} f))$:

```
with(VectorCalculus): SetCoordinates('cartesian'[x,y,z]);
f := VectorField(<-(x*y)^2, cos(x*y*z)^2, sin(z)^2>);
curl_f          := map(factor, Curl(f));
div_f           := combine(Divergence(f));
lap_div_f       := combine(Laplacian(div_f));
grad_lap_div_f := map(simplify, Gradient(lap_div_f));
```

6 Standing waves in strings

6.1 Statement of the problem

◦ We consider a string that is initially stretched between two points with the equilibrium position of the string lying along the x-axis. After the string is plucked, let $u(x,t)$ be the displacement of the string at position x and time t. We use the following assumptions: uniform string, planar vibrations, uniform tension, no other external forces, small vibrations.

◦ The partial derivatives represent the vertical velocity — $u_t(x,t)$, acceleration of the point on the string at position x — $u_{tt}(x,t)$, the slope of the string at position x — $u_x(x,t)$, the concavity of the string at position x — $u_{xx}(x,t)$.

▶ The following problem is an example of an initial boundary value problem for the wave equation in domain $\mathcal{D} = \{0 < x < L, 0 < t < \infty\}$, $u_{tt} = 4u_{xx}$, $u(0,t) = 0$, $u(L,t) = 0$, $u(x,0) = \cos(x)$, $u_t(x,0) = 1$:

```
with(plots):
n := 2: L := 1: c := 2: N := 300:
u:=(x,t)->(A*cos(n*Pi*c*t/L)+B*sin(n*Pi*c*t/L))*sin(n*Pi*x/L);
Eq1:=u(x,0)=cos(x); Eq2:=evala(subs(t=0,diff(u(x,t),t)))=1;
S1:=solve({Eq1,Eq2},{A,B}); u:=unapply(subs(S1,u(x,t)),x,t);
N:=24: animate(u(x,t),x=-Pi..Pi,t=0..2,frames=N);
```

◦ If $f(z)$ is any nonconstant twice differentiable function, then $u(x,t) = f(x - ct)$ and $u(x,t) = f(x + ct)$ are travelling wave solutions of the wave equation and the parameter c ($c^2 = T/\rho$) is the speed at which any travelling wave will propagate along the string:

```
with(plots): N:=300: u1:=(x,t)->cos(x-2*t); u2:=(x,t)->cos(x+2*t);
A:=array(1..2): A[1]:=animate(u1(x,t),x=0..4*Pi,t=1..10,frames=N):
A[2]:=animate(u2(x,t),x=Pi/2..9*Pi/2,t=1..10,frames=N, color=blue):
display(A);
```

6.2 Solutions of the wave equation

◦ *The general solution* of the wave equation can be obtained as follows:

```
with(PDEtools); declare(u(x,t));  ON;
pde :=diff(u(x,t), t$2)=c^2*diff(u(x,t), x$2);
casesplit(pde); separability(pde, u(x,t));
pdsolve(pde, build);
```

◦ *The d'Alembert solution* of the wave equation is based on the observation that the general solution could be decomposed into the sum of two travelling waves (each travelling with speed c in opposite directions) and can be obtained (for infinite string) as follows:

```
u:=(x,t)->F(x-c*t)+G(x+ c*t);  Ec1:=u(x,0)=f(x);
Ec2 :=(unapply(diff(u(x, t), t), x, t))(x, 0)=g(x);
Ec21:=value(map(Int,subs(x=s,factor(Ec2/c)),s=0..x));
Sol:= solve({Ec1, Ec21}, {F(x), G(x)});
F1 := unapply(op(1,Sol),x); G1:=unapply(op(2,Sol),x);
 u := combine(F1(x-c*t) + G1(x+c*t));
```

◦ *Standing waves solution.* Another approach for solving the wave equation involves decomposing the solution $u(x,t)$ into the sum of *standing waves*. Function

$$u(x,t) = \begin{cases} (C_1 + C_2 x)(C_3 + C_4 t), \\ \left(C_1 e^{rx/c} + C_2 e^{-rx/c}\right)\left(C_3 e^{rt} + C_4 e^{-rt}\right), \\ \left(C_1 \cos(rx/c) + C_2 \sin(rx/c)\right)\left(C_3 \cos(rt) + C_4 \sin(rt)\right), \end{cases}$$

describes all possible standing waves which are the solutions of the waves equation.

▶ Construct standing wave solutions to the wave equation $u_{tt} = 9u_{xx}$:

```
with(plots): setoptions(axes=boxed,scaling=constrained,
             thickness=3,tickmarks=[3,3]);
N:=100: c:=3: r:=2: G:=array(1..3);
C[1,1] := 1:     C[1,2]:= -1:    C[1,3] := 1:    C[1,4]:=-1:
C[2,1] := exp(x):C[2,2]:=exp(-x): C[2,3]:= 1/20: C[2,4]:=1:
C[3,1] := cos(x):C[3,2] := 0: C[3,3] := 1: C[3,4] := 1:
u[1]:=(x,t)->(C[1,1]+C[1,2]*x)*(C[1,3] + C[1,4]*t);
u[2]:=(x,t)->(C[2,1]*exp(r*x/c)+C[2,2]*exp(-r*x/c))*
             (C[2,3]*exp(r*t) + C[2,4]*exp(-r*t));
u[3]:=(x,t)->(C[3,1]*cos(r*x/c)+C[3,2]*sin(r*x/c))*
             (C[3,3]*cos(r*t)  +C[3,4]*sin(r*t));
G[1]:=animate(u[1](x,t),x=-10..10,t=0..10,frames=N,color=blue):
G[2]:=animate(u[2](x,t),x=-1..1,t=0..2,frames=N,color=green):
G[3]:=animate(u[3](x,t),x=-Pi..Pi,t=0..10,frames=N,color=plum):
display(G);
```

6.3 Standing waves in a fix string

∘ In particular applications, only a small subset of these solutions may be physically realistic. As a particular application, we obtain the standing waves for a string of finite length L in which both ends of the string are fixed, $u_{tt} = 4u_{xx}$, $u(0,t) = 0$, $u(1,t) = 0$, $u(x,0) = \sin(\pi x)$, $u_t(x,t) = 0$, $\mathcal{D} = \{0 < x < 1, t > 0\}$:

```
with(plots): setoptions(axes=boxed, scaling=constrained,
             thickness=3, tickmarks=[3,3]);
n := 2: L := 1: c := 2: N := 300: omega := n*Pi*c/L:
u := (x,t)->(A*cos(omega*t)+B*sin(omega*t))*sin(n*Pi*x/L);
Eq1:=u(x,0)=sin(Pi*x); Eq2:=evala(eval(diff(u(x,t),t),t=0))=0;
Sol:=solve({Eq1,Eq2},{A,B}); u:=unapply(eval(u(x,t),Sol),x,t);
G1 := animate(u(x, t),x=0..L,t=0..2,frames=N,color=blue):
G2 := plot(0, x=0..L, color=green,linestyle=DOT):
display([G1, G2]);
```

6.4 Superposition of standing waves and Fourier series

∘ We show that the boundary value problem for a fix string $u_{tt} = c^2 u_{xx}$, $0<x<L$, $t>0$, $u(0,t)=0$, $u(L,t)=0$ has the property that the superposition of two solutions $U(x,t) = Au_1(x,t) + Bu_2(x,t)$ is a solution of the problem:

```
with(PDEtools): Ec1:=diff(u1(x,t),t$2)=c^2*diff(u1(x,t),x$2);
u1(0, t) := 0;   u1(L, t) := 0;   declare(Ec1);   Ec1;
Ec2 := diff(u2(x, t), t$2) = c^2*diff(u2(x, t), x$2);
```

```
u2(0, t) := 0;   u2(L, t) := 0; declare(Ec2); Ec2;
U :=  A*u1(x, t)+ B*u2(x, t);  EDP := diff(U, t$2);
EDP := combine(factor(subs({Ec1, Ec2}, EDP)), diff);
Diff(U, t$2) = Diff(int(int(EDP, x), x), x);
eval(U, x=0);      eval(U, x=L);
```

∘ In modern mathematics, a Fourier Analysis is an important result and show that there exists an expansion of a function in terms of a set of complete functions. Following the Fourier theory, we solve the initial boundary value problem that describes the movement of a fix string, $u_{tt} = \frac{1}{16\pi^2} u_{xx}$, $0 < x < 0.5$, $t > 0$, $u(0,t) = 0$, $u(0.5,t) = 0$, $u(x,0) = 0$, $u_t(x,0) = \sin(4\pi x)$:

```
with(plots): L := 0.5; c := 1/(4*Pi); M := 10;
f := x -> 0;  g := x -> sin(4*Pi*x);
an := proc(n) option remember;
      2/L*evalf(int(f(x)*sin(n*Pi*x/L), x=0..L)); end;
bn := proc(n) option remember;
      evalf(2/L*evalf(int(g(x)*sin(n*Pi*x/L), x=0..L))); end;
un := proc(x, t, n)  (an(n)*cos(n*Pi*c*t/L)
      +bn(n)/(c*n*Pi/L)*sin(n*Pi*c*t/L))*sin(n*Pi*x/L);  end;
AprF:=proc(x, t)  evalf(add(un(x, t, k), k=1..M));  end;
plot(AprF(x,0.5), x=0..L, color=blue, thickness=3);
```

7 Standing waves in fluids

7.1 Statement of the problem

∘ The classical two-dimensional standing wave problem consists of solving the Euler equations for a one- or two-layer fluid with free boundary conditions (free fluid surface). The assumption is made that the flow is irrotational. The boundary value problem needs to be solved in a flow domain, for example, $\mathcal{D} = \{0 \le x \le L, -h \le y \le \eta(x,t)\}$ for the surface elevation $\eta(x,t)$ and the velocity potential $\phi(x,y,t)$. The fluid depth h, and the horizontal size of the domain L are given. We study periodic solutions (in x and t) of standing wave problem.

∘ In general, there are two ways for representing the fluid motion: the Eulerian approach, in which the coordinates are fixed in the reference frame of the observer, and the Lagrangian approach, in which the coordinates are fixed in the reference frame of the moving fluid. We construct analytically approximate solutions in Lagrangian variables. We develop the analytic Lagrangian approach proposed by Sekerzh-Zenkovich (1947) for constructing approximate solutions for nonlinear waves. We generalize the solution method that allows one to solve a set of problems, for example, the problems of infinite- and finite-depth surface standing waves and infinite- and finite-depth internal standing waves. This method can be useful for

extending a series solution to high order, solving a problem that is not solvable in Eulerian formulation, or solving another set of problems. We develop computer algebra procedures to aid in the construction of higher-order approximate analytical solutions.

∘ Note that most of the approximate analytic solutions were obtained using the Eulerian formulation, the present paper deals with the alternative formulation, which deserves to be better known. Therefore, we compare the analytic frequency-amplitude dependences obtained in Lagrangian variables with the corresponding ones known in Eulerian variables. The analysis has shown that the analytic frequency-amplitude dependences are in complete agreement with previous results obtained by Rayleigh (1915), Penney and Price (1952), Aoki (1980), Tadjbakhsh and Keller (1960), Okamura (1997) in Eulerian variables, and by Shingareva (1995), Shingareva et al. (2002), Shingareva and Lizárraga (2004a) in Lagrangian variables.

The analysis of solutions has shown that the use of the Lagrangian approach to solve standing waves problems presents some advantages with respect to the Euler formulation, particulary because it allows to simplify the boundary conditions (the unknown free boundary is a line), the radius of convergence of an expansion parameter is bigger than in the Eulerian variables (this allows one to observe steep standing waves).

7.2 Asymptotic solution

∘ We consider two-dimensional nonlinear wave motions in the fluid domain $\mathcal{D} = \{0 \le x \le L, -\infty \le y \le \eta(x,t)\}$. On the free surface the pressure is constant and equal to zero. We consider this as a basic model and other models can be derived on its basis. We choose a rectangular system of coordinates xOy in the plane of motion so that (i) the x-axis coincides with the horizontal level of fluid at rest and (ii) the y-axis is directed vertically upwards so that the unperturbed free surface has coordinates $y = 0$ and $x \in [0, L]$.

∘ We transform variables from Eulerian (x, y) to Lagrangian (a, b) adding the following requirements: the Jacobian $J = \dfrac{\partial(x,y)}{\partial(a,b)} = 1$, the free surface $y = \eta(x,t)$ is equivalent to the parametric curve $\{x(a,0,t), y(a,0,t)\}$, at $t = 0$ the free surface is $\{x(a,0,0), y(a,0,0)\}$, at the vertical lines $a = 0$ and $a = L$ the horizontal velocity $x_t = 0$, and the infinite depth is $b = -\infty$. The Lagrange equations for wave motions in fluid plus the continuity equation and the boundary conditions are then given as:

$$x_{tt}x_a + (y_{tt} + g)y_a + \frac{p_a}{\rho} = 0, \quad x_{tt}x_b + (y_{tt} + g)y_b + \frac{p_b}{\rho} = 0, \ \frac{\partial(x,y)}{\partial(a,b)} = 1,$$
$$x(0,b,t) = 0, \ x(L,b,t) = L, \ \ y(a,-\infty,t) = -\infty, \ \ p(a,0,t) = 0,$$

where $x(a,b,t)$ and $y(a,b,t)$ are the coordinates of an individual fluid particle in motion, $p(a,b,t)$ is the pressure in the fluid, ρ is the fluid density.

∘ We consider weakly nonlinear standing waves or waves of small amplitude and steepness, for which the amplitude and the ratio of wave height to wavelength is assumed to be of order ε, where ε is a small parameter. We introduce the dimensionless amplitude ε, the wave phase ψ, Lagrangian variables α, β (instead of a, b), and space coordinates and pressure ξ, η, and σ (instead of x, y, and p):

$$\kappa A = \varepsilon, \quad \psi = \omega t, \quad \alpha = a\kappa, \quad \beta = b\kappa,$$
$$\kappa x = \alpha + \varepsilon\xi, \quad \kappa y = \beta + \varepsilon\eta, \quad \kappa^2 p = -\kappa(\rho g)\kappa y + \varepsilon\rho\omega_{(0)}^2\sigma,$$

where $\kappa = \pi n/L$ is the wave number (n is the number of nodes of the wave), ω is the nonlinear frequency, $\omega_{(0)}^2 = g\kappa$ is the dispersion relation for linear periodic waves, and g is the acceleration due to gravity. In terms of the dimensionless variables, the equations of motion and the boundary conditions can be rewritten in the form

$$\mathcal{L}^1(\xi,\sigma) = -\varepsilon(\xi_{\psi\psi}\xi_\alpha + \eta_{\psi\psi}\eta_\alpha),\ \mathcal{L}^2(\eta,\sigma) = -\varepsilon(\xi_{\psi\psi}\xi_\beta + \eta_{\psi\psi}\eta_\beta),\ \mathcal{L}^3(\xi,\eta) = -\varepsilon\frac{\partial(\xi,\eta)}{\partial(\alpha,\beta)},$$
$$\xi(0,\beta,\psi) = 0,\ \xi(\pi n,\beta,\psi) = 0,\ \eta(\alpha,-\infty,\psi) = 0,\ \sigma(\alpha,0,\psi) - \eta(\alpha,0,\psi) = 0,$$

where linear differential operators $\mathcal{L}^i$ $(i = 1, 3)$ are

$$\mathcal{L}^1(\xi,\sigma) = \xi_{\psi\psi} + \sigma_\alpha, \quad \mathcal{L}^2(\eta,\sigma) = \eta_{\psi\psi} + \sigma_\beta, \quad \mathcal{L}^3(\xi,\eta) = \xi_\alpha + \eta_\beta.$$

∘ The construction of asymptotics is based on the perturbation theory. Defining the formal power series in the amplitude parameter ε

$$u = \mathcal{F}^{111}(u) + \sum_{i=2}^{N} \varepsilon^{i-1} u^{(i)} + O(\varepsilon^N), \quad u = \xi, \eta, \sigma,$$

where $\xi^{(i)}, \eta^{(i)}$, and $\sigma^{(i)}$ $(i = 2, \ldots, N)$ are unknown 2π-periodic in ψ functions of variables α, β, and ψ, and the linear terms $\mathcal{F}^{111}(u)$, $u = \xi, \eta, \sigma$, are defined by the following expressions:

$$\mathcal{F}^{111}(\xi) = -\sin(\alpha)e^\beta\cos(\psi), \quad \mathcal{F}^{111}(\eta) = \mathcal{F}^{111}(\sigma) = \cos(\alpha)e^\beta\cos(\psi).$$

Considering weakly nonlinear standing waves, we can assume that the nonlinear wave frequency ω is close to the linear wave frequency $\omega_{(0)}$:

$$\omega(\varepsilon) \equiv \psi_t = \omega_{(0)} + \sum_{i=1}^{N-1} \varepsilon^i \omega_{(i)} + O(\varepsilon^N),$$

where $\omega_{(i)}$ are new unknown corrections to the nonlinear wave frequency.

∘ *Algebraic procedure*: substituting these expansions into the equations of motion and the boundary conditions and matching the coefficients of like powers of ε,

we arrive at the linear inhomogeneous system of partial differential equations and boundary conditions for the i th approximation. We look for a solution, 2π-periodic in ψ, to this system of equations and the boundary conditions in the form:

$$v^{(i)}= \sum_{j,k,l=0}^{i} V_i^{jkl} \mathcal{F}^{jkl}(v), \quad v = \xi, \eta, \sigma, \quad V = \Xi, \mathrm{H}, \Sigma,$$

where Ξ_i^{jkl}, H_i^{jkl}, and Σ_i^{jkl} are unknown constants. Substituting these series into the linear system of equations and boundary conditions for the ith approximation and using the orthogonality conditions for periodic solutions, we obtain a family of systems of linear algebraic equations with respect to the unknown coefficients Ξ_i^{jkl}, H_i^{jkl}, and Σ_i^{jkl}. By using formulas described above, we obtain the asymptotic solution of the order of $O(\varepsilon^N)$ and the unknown corrections to the nonlinear wave frequency $\omega_{(i)}$. Setting $\beta = 0$ in the parametric equations for κx and κy, we can obtain the profiles of surface standing waves in Lagrangian variables for the ith approximation $(i = 2, \ldots, N)$. Changes in the amplitude ε influence the surface configuration by changing both the shape of the surface and the amplitude of motion.

∘ As an example, we show here how to find the second derivative with respect to dimensionless time ψ for the variables $\xi(\alpha, \beta, t)$ and $\eta(\alpha, \beta, t)$:

```
NA:=2; NP:=NA-1; NN:=NA+1; TZ:=-delta/(delta-1):lambda:=1+TZ;
psiT := omega+add(epsilon^i*omega||i, i= 1..NP);
setsub := {diff(psi(t), t) = psiT};  subt := {psi(t) = psi};
 xi    := (x, y, z) -> -sin(x)*exp(y)*cos(z);
 eta   := (x, y, z) ->  cos(x)*exp(y)*cos(z);
Fxi:=(x->xi(alpha,beta,psi(x)));Feta:=(x->eta(alpha,beta,psi(x)));
dxi   := diff(Fxi(t), t);   deta :=diff(Feta(t), t);
xi1T  := subs(setsub, dxi); xi2T:=subs(setsub, diff(xi1T, t));
eta1T := subs(setsub, deta);eta2T:= subs(setsub, diff(eta1T, t));
Xi    := xi(alpha, beta, psi); Eta := eta(alpha, beta, psi);
```

∘ Based on the perturbation solution obtained for a particular case, for instance, for the infinite-depth standing wave problem, we write out the frequency-amplitude dependence:

$$\frac{\omega}{\omega_{(0)}} = 1 - \frac{1}{8}\varepsilon^2 - \frac{23}{256}\varepsilon^4 + O(\varepsilon^5).$$

The dependence is equal to the previous results obtained by Rayleigh (1915) in Eulerian variables, where $\omega_1 = 0$ and $\omega_2 = -\dfrac{1}{8}A^2\omega_0$. This expression coincides with the results obtained by Penney and Price (1952) and Aoki (1980) in Eulerian variables, where $\omega_3 = 0$ and $\omega_4 = -\dfrac{15}{256}A^4\omega_0$.

References

Akritas, A.G. 1989 Elements of Computer Algebra with Applications. *Wiley, NY*

Aoki, H. 1980 Higher order calculation of finite periodic standing waves by means of the computer. *J. Phys. Soc. Jpn.* **49** 1598–1606

Calmet, J. and van Hulzen, J.A. 1983 Computer algebra systems. *Computer Algebra: Symbolic and Algebraic Computations*/eds. B. Buchberger, G.E. Collins, and R. Loos, 2d ed. *Springer-Verlag, New-York*

Char, B.W., Geddes, K.O., Gonnet, G.H., Monagan, M.B., and Watt, S.M. 1990 Maple Reference Manual, 5th ed. *Waterloo Maple Publishing, Canada*

Corless, R.M. 1995 Essential Maple. *Springer-Verlag, Berlin*

Concus, P. 1962 Standing capillary-gravity waves of finite amplitude. *JFM* **14** 568–576.

Davenport, J.H., Siret, Y. and Tournier, E. 1993 Computer Algebra Systems and Algorithms for Algebraic Computation. *Academic Press, London*

Heck, A. 1996 Introduction to Maple. *Springer, New York*

Geddes, K.O., Czapor, S.R., and Labahn, G. 1992 Algorithms for Computer Algebra. *Kluwer Academic Publishers, Boston*

Grosheva, M.V. and Efimov, G.B. 1988 On systems of symbolic computations (in Russian). *Applied Program Packages. Analytic Transformations* ed. A.A. Samarskii *Nauka, Moscow, 30–38*

Okamura, M. 1997 Resonant standing waves on water of uniform depth. *J. Phys. Soc. Jpn.* **66** 3801–3808

Penney, W. G. and Price, A. T. 1952 Finite periodic stationary gravity waves in a perfect liquid. Part 2. *Phil. Trans. R. Soc. Lond.* A **224** 254–284

Rayleigh, Lord 1915 Deep water waves, progressive or stationary, to the third order of approximation. *Phil. Trans. R. Soc. Lond.* A **91** 345–353

Sekerzh-Zenkovich, Ya. I. 1947 On the theory of standing waves of finite amplitude. *Doklady Akad. Nauk. USSR* **58** 551–554

Shingareva, I. 1995 Investigation of Standing Surface Waves in a Fluid of Finite Depth by Computer Algebra Methods. PhD thesis, *IPM, RAS, Moscow*

Shingareva, I., Sekerzh-Zenkovich, S.Ya., García A., M. G. 2002 Ninth-Order Analytic Solution of Free Standing Gravity Waves in Fluid of Infinite Depth. *Bull. ERCOFTAC* **52** 37–41

Shingareva, I. and Lizárraga-Celaya, C. 2004 High Order Asymptotic Solutions to Free Standing Water Waves by Computer Algebra. *Proceedings of the Maple Summer Workshop, Waterloo, Ontario, Canada, R.J. Lopez (Editor)* 1–28

Shingareva, I. y Lizárraga-Celaya, C. 2004 Curva Crítica en Estructuras de Fluidos. *Aportaciones Matemáticas, SMM, México* **34** 57–72

Shingareva, I., Lizárraga Celaya, C., and Ochoa Ruiz, A.D. 2006 Maple and Standing Waves. Problems and Solutions. *Editorial Unison (Universidad de Sonora), Hermosillo, México (in Spanish)* 180 p

Tadjbakhsh, I. and Keller, J. B. 1960 Standing surface waves of finite amplitude. *JFM* **8** 442–451

Wester, M.J. 1999 Computer Algebra Systems: a Practical Guide. *John Wiley, Chichester, UK*