Asociación Argentina de Astronomía
Third La Plata International School on Astronomy and Geophysics:
Chaos, diffusion and non–integrability in Hamiltonian Systems
Applications to Astronomy , 2012
P. M. Cincotta, C. M. Giordano & C. Efthymiopoulos, eds.

LECTURE

Methods of algebraic manipulation in perturbation theory

A. Giorgilli[1,2], M. Sansottera[3]

(1) Dipartimento di Matematica, Università degli Studi di Milano,
Via Saldini 50, 20133 — Milano, Italy
(2) Istituto Lombardo Accademia di Scienze e Lettere
(3) naXys, Namur Center for Complex Systems, FUNDP,
Rempart de la Vierge 8, B5000 — Namur, Belgium

Abstract. We give a short introduction to the methods of representing polynomial and trigonometric series that are often used in Celestial Mechanics. A few applications are also illustrated.

1. Overview

Algebraic manipulation on computer is a tool that has been developed quite soon, about one decade after the birth of computers, the first examples dating back to the end of the fifties of the last century. General purpose packages began to be developed during the sixties, and include, e.g., Reduce (1968), Macsyma (1978), muMath (1980), Maple (1984), Scratchpad (1984), Derive (1988), Mathematica (1988), Pari/GP (1990) and Singular (1997) (the dates refer to the first release). However, most of the facilities of these general purpose manipulators are simply ignored when dealing with perturbation methods in Celestial Mechanics. For this reason, the job of developing specially devised manipulation tools has been undertaken by many people, resulting in packages that have limited capabilities, but are definitely more effective in practical applications. Producing a list of these packages is a hard task, mainly because most of them are not publicly available. A list of "old time" packages may be found in Henrard (1989) and Laskar (1989). In recent times a manipulator developed by J. Laskar and M. Gastineau has become quite known.

Finding references to the methods implemented in specially devised packages is as difficult as giving a list. We know only a few papers by Broucke and Garthwaite (1969), Broucke (1989), Rom (1970), Henrard (1986 and 1989), Laskar (1989), Jorba (1999) and Biscani (2009). A complete account of the existing literature on the subject goes beyond the limits of the present note. The present work introduces some ideas that have been used by the authors in order to implement a package named $X\varrho\acute{o}\nu o\varsigma$.

As a matter of fact, most of the algebraic manipulation used in Celestial Mechanics makes use of the so called "Poisson series", namely series with a general term of the form

$$x_1^{j_1} \cdot \ldots \cdot x_n^{j_n} \mathop{}_{\sin}^{\cos} (k_1\varphi_1 + \ldots + k_m\varphi_m) \ ,$$

(with obvious meaning of the symbols). Thus, a very minimal set of operations is required, namely sums, products and derivatives of polynomials and/or trigonometric polynomials. Traditionally, also the operation of inversion of functions, usually made again via series expansion, was required. However, the expansion methods based on Lie series and Lie transforms typically get rid of the latter operation (see, e.g., Henrard 1974).

Writing a program doing algebraic manipulation on series of the type above leads one to be confronted with a main question, namely how to represent a polynomial, trigonometric polynomial or Poisson series on a computer. The papers quoted above actually deal with this problem, suggesting some methods. In these lectures we provide an approach to this problem, followed by a few examples of applications.

In sect. 2 we include a brief discussion about the construction of normal form for a Hamiltonian system in the neighborhood of an elliptic equilibrium. We do not attempt to give a complete discussion, since it is available in many papers. We rather try to orient the reader's attention on the problem of representing perturbation series.

In sect. 3–7 we introduce a method which turns out to be quite useful for the representation of a function as an array of coefficients. The basic idea has been suggested to one of the authors by the paper of Gustavson (1966) (who, however, just mentions that he used an indexing method, without giving any detail about its implementation). One introduces an *indexing function* which transforms an array of exponents in a polynomial (or trigonometric polynomial) in a single index within an array. The general scheme is described in sect. 3. The basics behind the construction of an indexing function are described in sect. 4. The details concerning the representation of polynomials and trigonometric polynomials are reported in sects. 5 and 6, respectively. In sect. 7 we include some hints about the case of sparse series, that may be handled by combining the indexing functions above with a tree representation. Finally, sect. 8 is devoted to three applications, by giving a short account of the contents of published papers.

2. A common problem in perturbation theory

A typical application of computer algebra is concerned with the construction of first integrals or of a normal form for a Hamiltonian system. A nontrivial example, which however may be considered as a good starting point, is the calculation of a normal form for the celebrated model of Hénon and Heiles (1964), which has been done by Gustavson (1966). Some results on this model are reported in sect. 8.

We assume that the reader is not completely unfamiliar with the concept of normal form for a (possibly Hamiltonian) system of differential equations. Thus, let us briefly illustrate the problem by concentrating our attention on the algorithmic aspect and by explaining how algebraic manipulation may be introduced.

2.1. Computation of a normal form

Let us consider a canonical system of differential equations in the neighborhood of an elliptic equilibrium. The Hamiltonian may typically be given the form of a power series expansion

$$H(x,y) = H_0(x,y) + H_1(x,y) + \ldots\ , \qquad H_0(x,y) = \sum_{j=1}^{n} \frac{\omega_j}{2}(x_j^2 + y_j^2)\ , \tag{1}$$

where $H_s(x,y)$ for $s \geq 1$ is a homogeneous polynomial of degree $s+2$ in the canonical variables $(x,y) \in \mathbb{R}^{2n}$. Here $\omega \in \mathbb{R}^n$ is the vector of the frequencies, that are assumed to be all different from zero.

In such a case the system is said to be in Birkhoff normal form in case the Hamiltonian takes the form

$$H(x,y) = H_0(x,y) + Z_1(x,y) + Z_2(x,y) + \ldots \quad \text{with} \quad L_{H_0} Z_s = 0\ , \tag{2}$$

where $L_{H_0}\cdot = \{H_0, \cdot\}$ is the Lie derivative with respect to the flow of H_0, actually the Poisson bracket with H_0.

The concept of Birkhoff normal form is better understood if one assumes also that the frequencies are non resonant, i.e., if

$$\langle k, \omega \rangle \neq 0 \quad \text{for all } k \in \mathbb{Z}^n\ ,\ k \neq 0\ ,$$

where $\langle k, \omega \rangle = \sum_j k_j \omega_j$. For, in this case the functions $Z_s(x,y)$ turn out to be actually function only of the n actions of the system, namely of the quantities

$$I_j = \frac{x_j^2 + y_j^2}{2}\ , \quad j = 1, \ldots, n\ .$$

It is immediate to remark that $I_1, \ldots, I_n$ are independent first integrals for the Hamiltonian, an that they are also in involution, so that, by Liouville's theorem, the system turns out to be integrable. The definition of normal form given in (2) is more general, since it includes also the case of resonant frequencies.

The calculation of the normal form may be performed using the Lie transform method, which turns out to be quite effective. We give here the algorithm without proof. A complete description may be found, e.g., in (Giorgilli 1978), and the description of a program implementing the method via computer algebra is given in (Giorgilli 1979). The corresponding `FORTRAN` program is available from the CPC library.

The Lie transform is defined as follows. Let a *generating sequence* $\chi_1(x,y)$, $\chi_2(x,y), \ldots$ be given, and define the operator

$$T_\chi = \sum_{s \geq 0} E_s \tag{3}$$

where the sequence $E_0, E_1, \ldots$ of operators is recursively defined as

$$E_0 = 1\ , \quad E_s = \sum_{j=1}^{s} \frac{j}{s} L_{\chi_j} E_{s-j} \tag{4}$$

This is a linear operator that is invertible and satisfies the interesting properties

$$T_\chi\{f,g\} = \{T_\chi f, T_\chi g\}\ , \quad T_\chi(f \cdot g) = T_\chi f \cdot T_\chi g\ . \tag{5}$$

Let now $Z(x,y) = H_0(x,y) + Z_1(x,y) + Z_2(x,y) + \ldots$ be a function such that

$$T_\chi Z = H \ , \tag{6}$$

where H is our original Hamiltonian, and let Z possess a first integral Φ, i.e., a function satisfying $\{Z, \Phi\} = 0$. Then one has also

$$T_\chi\{Z,\Phi\} = \{T_\chi Z, T_\chi \Phi\} = \{H, T_\chi\Phi\} = 0 \ ,$$

which means that *if Φ is a first integral for Z then $T_\chi\Phi$ is a first integral for H.*

The question now is: *can we find a generating sequence χ_1, $\chi_2, \ldots$ such that the function Z satisfying* (6) *is in Birkhoff normal form?*

The answer to this question is in the positive, and the generating sequence may be calculated via an explicit algorithm that can be effectively implemented via computer algebra. We include here the algorithm, referring to, e.g., (Giorgilli 1978) for a complete deduction. Here we want only to stress that all operations that are required may be actually implemented on a computer.

The generating sequence is determined by solving for χ and Z the equations

$$Z_s - L_{H_0}\chi_s = H_s + Q_s \ , \quad s \geq 1 \ , \tag{7}$$

where Q_s is a known homogeneous polynomial of degree $s+2$ given by $Q_1 = 0$ and

$$Q_s = -\sum_{j=1}^{s-1}\bigl(E_j Z_{s-j} + \frac{j}{s}\{\chi_j, E_{s-j}H_0\}\bigr) \ , \quad s > 1 \ .$$

In order to solve (7) it is convenient to introduce complex variables ξ, η via the canonical transformation

$$x_j = \frac{1}{\sqrt{2}}(\xi_j + i\eta_j) \ , \quad y_j = \frac{i}{\sqrt{2}}(\xi_j - i\eta_j)$$

which transforms $H_0 = i\sum_j \omega_j\xi_j\eta_j$. In these variables the operator L_{H_0} takes a diagonal form, since

$$L_{H_0}\xi^j\eta^k = i\langle k-j, \omega\rangle\xi^j\eta^k \ ,$$

where we have used the multi-index notation $\xi^j = \xi_1^{j_1}\cdot\ldots\cdot\xi_n^{j_n}$, and similarly for η. Thus, writing the r.h.s. of (7) as a sum of monomials $c_{j,k}\xi^j\eta^k$ the most direct form of the solution is found by including in Z all monomials with $\langle k-j,\omega\rangle = 0$, and adding $\frac{c_{j,k}}{i\langle k-j,\omega\rangle}\xi^j\eta^k$ to χ_s for all monomials with $\langle k-j,\omega\rangle \neq 0$. This is the usual way of constructing a normal form for the system (1).

Let us now examine in some more detail the algebraic aspect. With a little amount of patience one can verify that (7) involves only homogeneous polynomials of degree $s+2$. Thus, one should be able to manipulate this kind of functions. Moreover, a careful examination of the algorithm shows that there are just elementary algebraic operations that are required, namely:

(i) sums and multiplication by scalar quantities;

(ii) Poisson brackets, which actually require derivatives of monomials, sums and products;

(iii) linear substitution of variables, which may still be reduced to calculation of sums and products without affecting the degree of the polynomial;

(iv) solving equation (7), which just requires a division of coefficients.

These remarks should convince the reader that implementing the calculation of the normal form via algebraic manipulation on a computer is just matter of being able of representing homogeneous polynomials in many variables and performing on them a few elementary operations, such as sum, product and derivative.

2.2. A few elementary considerations

In order to have an even better understanding the reader may want to consider the elementary problem of representing polynomials in one single variable. We usually write such a polynomial of degree s (non homogeneous, in this case) as

$$f(x) = a_0 + a_1 x + \ldots + a_s x^s \ .$$

A machine representation is implemented by storing the coefficients a_0, $a_1, \ldots,$ a_n as a one-dimensional array of floating point quantities, either real or complex. E.g., in `FORTRAN` language one can represent a polynomial of degree 100 by just saying, e.g., `DIMENSION F(101)` and storing the coefficient a_j as `F(j+1)` (here we do not use the extension of `FORTRAN` that allows using zero or even negative indices for an array). Similarly in a language like `C` one just says, e.g., `double f[101]` and stores a_j as `f[j]`.

The operation of sum is a very elementary one: if f, g are two polynomials and the coefficients are stored in the arrays `f,g` (in `C` language) then the sum h is the array `h` with elements `h[j] = f[j] + g[j]`. The derivative of f is the array `fp` with elements `fp[j] = (j+1)*f[j+1]`. In a similar way one can calculate the product, by just translating in a programming language the operations that are usually performed by hand.

The case of polynomials in two variables is just a bit more difficult. A homogeneous polynomial of degree s is usually written as

$$f(x,y) = a_{s,0}x^s + a_{s-1,1}x^s y + \ldots + a_{0,s}y^s \ .$$

The naive (not recommended) representation would use an array with two indices (a matrix), by saying, e.g., `DIMENSION F(101,101)` and storing the coefficient $a_{j,k}$ as `F(j+1,k+1)`. Then the algebra is just a straightforward modification with respect to the one-dimensional case.

Such a representation is not recommended for at least two reasons. The first one is that arrays with arbitrary dimension are difficult to use, or even not allowed, in programming languages. The second and more conclusive reason is that such a method turns out to be very effective in wasting memory space. E.g., in the two dimensional case a polynomial of degree up to s requires a matrix with $(s+1)^2$ elements, while only $(s+1)(s+2)/2$ are actually used. Things go much worse in higher dimension, as one easily realizes.

The arguments above should have convinced the reader that an effective method of representing polynomials is a basic tool in order to perform computer algebra for problems like the calculation of normal form. Once such a method is available, the rest is essentially known algebra, that needs to be translated in a computer language.

The problem for Poisson series is a similar one, as the reader can easily imagine. The following sections contains a detailed discussion of indexing methods particularly devised for polynomials and for Poisson series. The underlying idea is to represent the coefficients as a one-dimensional array by suitably packing them in an effective manner, so as to avoid wasting of space.

3. General scheme

The aim of this section is to illustrate how an appropriate algebraic structure may help in representing the particular classes of functions that usually appear in perturbation theory. We shall concentrate our attention only on polynomials and trigonometric polynomials, which are the simplest and most common cases. However, the reader will see that most of the arguments used here apply also to more general cases.

3.1. Polynomials and power series

Let $\mathcal{P}$ denote the vector space of polynomials in the independent variables $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$. A basis for this vector space is the set $\{u_k(x)\}_{k\in\mathbb{Z}^n_+}$, where

$$u_k(x) = x^k \equiv x_1^{k_1} \cdot \ldots \cdot x_n^{k_n} \ . \tag{8}$$

In particular, we shall consider the subspaces $\mathcal{P}_s$ of $\mathcal{P}$ that contain all homogeneous polynomials of a given degree $s \geq 0$; the subspace $\mathcal{P}_0$ is the one-dimensional space of constants, and its basis is $\{1\}$. The relevant algebraic properties are the following:

(i) every subspace $\mathcal{P}_s$ is closed with respect to sum and multiplication by a number, i.e., if $f \in \mathcal{P}_s \ \wedge \ g \in \mathcal{P}_s$ then $f + g \in \mathcal{P}_s$ and $\alpha f \in \mathcal{P}_s$;

(ii) the product of homogeneous polynomials is a homogeneous polynomial, i.e., if $f \in \mathcal{P}_r \ \wedge \ g \in \mathcal{P}_s$ then $fg \in \mathcal{P}_{r+s}$;

(iii) the derivative with respect to one variable maps homogeneous polynomials into homogeneous polynomials, i.e., if $f \in \mathcal{P}_s$ then $\partial_{x_j} f \in \mathcal{P}_{s-1}$; if $s = 0$ then $\partial_{x_j} f = 0$, of course.

These three properties are the basis for most of the algebraic manipulations that are commonly used in perturbation theory.

A power series is represented as a sum of homogeneous polynomials. Of course, in practical calculations the series will be truncated at some order. Since every homogeneous polynomial $f \in \mathcal{P}_s$ can be represented as

$$f(x) = \sum_{|k|=s} f_k u_k(x) \ ,$$

it is enough to store in a suitable manner the coefficients f_k. A convenient way, particularly effective when most of the coefficients are different from zero, is

Table 1. Illustrating the function representation for power series. A memory block is assigned to the function $f(x)$. The coefficient f_k of $u_k(x)$ is stored at the address resulting by adding the offset $I(k)$ to the starting address of the memory block.

$f_{(0,0,\ldots,0)}$	$\leftarrow 0 = I(k) \leftarrow k = (0,0,\ldots,0)$
$f_{(1,0,\ldots,0)}$	$\leftarrow 1 = I(k) \leftarrow k = (1,0,\ldots,0)$
$\cdots$ $\cdots$ $\cdots$	
$f_{(k_1,k_2,\ldots,k_n)}$	$\leftarrow\ I(k)\ \leftarrow k = (k_1,k_2,\ldots,k_n)$
$\cdots$ $\cdots$ $\cdots$	

based on the usual lexicographic ordering of polynomials (to be pedantic, inverse lexicographic). E.g., a homogeneous polynomial of degree s in two variables is ordered as

$$a_{s,0}x_1^s + a_{s-1,1}x_1^{s-1}x_2 + \ldots + a_{0,s}x_2^s \ .$$

The idea is to use the position of a monomial x^k in the lexicographic order as an index $I(k_1,\ldots,k_n)$ in an array of coefficients. We call I and *indexing function*. Here we illustrate how to use it, deferring to sect. 5 the actual construction of the function.

The method is illustrated in table 1. Let f be a power series, truncated at some finite order s. A memory block is assigned to f. The size of the block is easily determined as $I\bigl((0,\ldots,0,s)\bigr)$. For, $(0,\ldots,0,s)$ is the last vector of length s. The starting address of the block is assigned to the coefficient of $u_{(0,0,\ldots,0)}$; the next address is assigned to the coefficient of $u_{(1,0,\ldots,0)}$, because $(1,0,\ldots,0)$ is the first vector of length 1, and so on. Therefore, the address assigned to the coefficient of $u_{(k_1,\ldots,k_n)}$ is the starting address of the block incremented by $I\bigl((k_1,\ldots,k_n)\bigr)$. If f is a homogeneous polynomial of degree s the same scheme works fine with a few minor differences: the length of the block is $I\bigl((0,\ldots,0,s)\bigr) - I\bigl((0,\ldots,0,s-1)\bigr)$, the starting address of the block is associated to the coefficient of $u_{(s,0,\ldots,0)}$, and the coefficient of $u_{(k_1,\ldots,k_n)}$ is stored at the relative address $I\bigl((k_1,\ldots,k_n)\bigr) - I\bigl((0,\ldots,0,s-1)\bigr)$. This avoids leaving an empty space at the top of the memory block.

In view of the form above of the representation a function is identified with a set of pairs (k, f_k), where $k \in \mathbb{Z}_+^n$ is the vector of the exponents, acting as the label of the elements of the basis, and f_k is the numerical coefficient. Actually the vector k is not stored, since it is found via the index. The algebraic operations of sum, product and differentiation can be considered as operations on the latter set.

(i) If $f, g \in \mathcal{P}_s$ then the operation of calculating the sum $f + g$ is represented as

$$\left.\begin{matrix}(k,\, f_k)\\(k,\, g_k)\end{matrix}\right\} \mapsto (k, f_k + g_k)\ ,$$

to be executed over all k such that $|k| = s$.

(ii) If $f \in \mathcal{P}_r$ and $g \in \mathcal{P}_s$ then the operation of calculating the product fg is represented as

$$\left.\begin{matrix}(k,\, f_k)\\(k',\, g_{k'})\end{matrix}\right\} \mapsto (k + k', f_k g_{k'})\ ,$$

to be executed over all k, k' such that $|k| = r$ and $|k'| = s$.

(iii) If $f \in \mathcal{P}_s$ then the operation of differentiating f with respect to, e.g., x_1 is represented as

$$(k, f_k) \mapsto \begin{cases}\emptyset & \text{for } k_1 = 0\ ,\\ (k', k_1 f_k) & \text{for } k_1 \neq 0\ ,\end{cases}$$

where $k' = (k_1 - 1, k_2, \ldots, k_n)$.

It is perhaps worthwhile to spend a few words about how to make the vector k to run over all its allowed values. In the case of sum, we do not really need it: since the indexes of both addends and of the result are the same, the operation can actually be performed no matter which k is involved: just check that the indexes are in the correct range.[1] In order to perform product and differentiation it is essential to know the values of k and k'. To this end, we can either use the inverse of the indexing function, or generate the whole sequence by using a function that gives the vector next to a given k.

3.2. Fourier series

Let us denote by $\varphi = (\varphi_1 \ldots, \varphi_n) \in \mathbb{T}^n$ the independent variables. The Fourier expansion of a real function on $\mathbb{T}^n$ takes the form

$$f(\varphi) = \sum_{k\in\mathbb{Z}^n} \left(a_k \cos\langle k, \varphi\rangle + b_k \sin\langle k, \varphi\rangle\right)\ , \qquad (9)$$

where a_k and b_k are numerical coefficients. In this representation there is actually a lot of redundancy: in view of $\cos(-\alpha) = \cos\alpha$ and $\sin(-\alpha) = -\sin\alpha$ the modes $-k$ and k can be arbitrarily interchanged. On the other hand, it seems that we actually need two different arrays for the sin and cos components, respectively. A straightforward way out is to use the exponential representation $\sum_k a_k e^{i\langle k,\varphi\rangle}$, but a moment's thought leads us to the conclusion that the redundancy is not removed at all. However, we can at the same time remove the redundancy and reduce the representation to a single array by introducing a suitable basis $\{u_k(\varphi)\}_{k\in\mathbb{Z}^n}$. Let $k \in \mathbb{Z}^n$; we shall say that k is *even* if the first non zero

[1] For a homogeneous polynomial of degree s the first vector is $(s, 0, \ldots, 0)$, and the last one is $(0, \ldots, 0, s)$. The indexes of these two vectors are the limits of the indexes in the sum.

component of k is positive, and that k is *odd* if the first non zero component of k is negative. The null vector $k = 0$ is said to be even. Then we set

$$u_k(\varphi) = \begin{cases} \cos\langle k, \varphi\rangle & \text{for } k \text{ even },\\ \sin\langle k, \varphi\rangle & \text{for } k \text{ odd }. \end{cases} \tag{10}$$

This makes the representation $f(\varphi) = \sum_{k\in\mathbb{Z}^n} \varphi_k u_k(\varphi)$ unique and redundancy free. It may be convenient to remark that the notation for the sin function may create some confusion. Usually, working with one variable, we write $\sin\varphi$. The convention above means that we should rather write $-\sin(-\varphi)$, which is correct, but a bit funny. This should be taken into account when, after having accurately programmed all the operations, we discover that our manipulator says, e.g., that $\frac{\mathrm{d}}{\mathrm{d}\varphi}\cos\varphi = -\sin(-\varphi)$.

In view of the discussion in the previous section it should now be evident that a truncated Fourier expansion of a function $f(\varphi)$ can easily be represented by storing the coefficient of $u_k(\varphi)$ at an appropriate memory address, as calculated by the indexing function $I(k)$ of sect. 6.

The considerations of the previous section can be easily extended to the problem of calculating the sum and/or product of two functions, and of differentiating a function. Let us identify any term of the Fourier expansion of the function f with the pair (k, f_k). Let us also introduce the functions $\mathrm{od}(k)$ and $\mathrm{ev}(k)$ as follows: if k is odd, then $\mathrm{od}(k) = k$ and $\mathrm{ev}(k) = -k$; else $\mathrm{od}(k) = -k$ and $\mathrm{ev}(k) = k$. That, is, force k to be odd or even, as needed, by possibly changing its sign.

(i) Denoting by (k, f_k) and (k, g_k) the same Fourier components of two functions f and g, respectively, the sum is computed as

$$\left.\begin{matrix}(k,\, f_k)\\ (k,\, g_k)\end{matrix}\right\} \mapsto (k, f_k + g_k)\ . \tag{11}$$

(ii) Denoting by (k, f_k) and $(k', g_{k'})$ any two terms in the Fourier expansion of the functions f and g, respectively, the product is computed as

$$\left.\begin{matrix}(k,\, f_k)\\ (k',\, g_{k'})\end{matrix}\right\} \mapsto \begin{cases} \left(\mathrm{ev}(k+k'), \frac{f_k g_{k'}}{2}\right) \cup \left(\mathrm{ev}(k-k'), \frac{f_k g_{k'}}{2}\right) & \text{for } k \text{ even and } k' \text{ even},\\ \left(\mathrm{od}(k+k'), \frac{f_k g_{k'}}{2}\right) \cup \left(\mathrm{od}(k-k'), -\frac{f_k g_{k'}}{2}\right) & \text{for } k \text{ even and } k' \text{ odd},\\ \left(\mathrm{od}(k+k'), \frac{f_k g_{k'}}{2}\right) \cup \left(\mathrm{od}(k-k'), \frac{f_k g_{k'}}{2}\right) & \text{for } k \text{ odd and } k' \text{ even},\\ \left(\mathrm{ev}(k+k'), -\frac{f_k g_{k'}}{2}\right) \cup \left(\mathrm{ev}(k-k'), \frac{f_k g_{k'}}{2}\right) & \text{for } k \text{ odd and } k' \text{ odd}. \end{cases} \tag{12}$$

Remark that the product always produces two distinct terms, unless $k = 0$ or $k' = 0$.

(iii) Denoting by (k, f_k) any term in the Fourier expansion of a function f, differentiation with respect to, e.g., φ_1 is performed as

$$(k, f_k) \mapsto \begin{cases} (-k, -k_1 f_k) & \text{for } k \text{ even ,} \\ (-k, k_1 f_k) & \text{for } k \text{ odd .} \end{cases} \tag{13}$$

All these formulæ follow from well known trigonometric identities.

4. Indexing functions

The basic remark for constructing an index function is the following. Suppose that we are given a countable set $\mathcal{A}$. Suppose also that $\mathcal{A}$ is equipped with a relation of complete ordering, that we shall denote by the symbols $\prec$, $\preceq$, $\succ$ and $\succeq$. So, for any two elements $a, b \in \mathcal{A}$ exactly one of the relations $a \prec b$, $a = b$ and $b \succ a$ is true. Suppose also that there is a minimal element in $\mathcal{A}$, i.e., there is $a_0 \in \mathcal{A}$ such that $a \succ a_0$ for all $a \in \mathcal{A}$ such that $a \neq a_0$. Then an index function I is naturally defined as

$$I(a) = \#\{b \in \mathcal{A} \; : \; b \prec a\} \; . \tag{14}$$

If $\mathcal{A}$ is a finite set containing N elements, then $I(\mathcal{A}) = \{0, 1, \ldots, N-1\}$. If $\mathcal{A}$ is an infinite (but countable) set, then $I(\mathcal{A}) = \mathbb{Z}_+$, the set of non negative integers. For instance, the trivial case is $\mathcal{A} = \mathbb{Z}_+$ equipped with the usual ordering relation. In such a case the indexing function is just the identity.

Having defined the function $I(a)$, we are interested in performing the following basic operations:

(i) for a given $a \in \mathcal{A}$, find the index $I(a)$;

(ii) for a given $a \in \mathcal{A}$, find the element next (or prior) to a, if it exists;

(iii) for a given $l \in I(\mathcal{A})$, find $I^{-1}(l)$, i.e., the element $a \in \mathcal{A}$ such that $I(a) = l$.

The problem here is to implement an effective construction of the index for some particular subsets of $\mathbb{Z}^n$ that we are interested in. In order to avoid confusions, we shall use the symbols $\prec$, $\preceq$, $\succ$ and $\succeq$ when dealing with an ordering relation in the subset of $\mathbb{Z}^n$ under consideration. The symbols $<$, $\leq$, $\geq$ and $>$ will always denote the usual ordering relation between integers.

As a first elementary example, let us consider the case $\mathcal{A} = \mathbb{Z}$. The usual ordering relation $<$ does not fulfill our requests, because there is no minimal element. However, we can construct a different ordering satisfying our requests as follows.

Let $k, k' \in \mathbb{Z}$. We shall say that $k' \prec k$ in case one of the following relations is true:

(i) $|k'| < |k|$;

(ii) $|k'| = |k| \;\wedge\; k' > k$.

The resulting order is $0, 1, -1, 2, -2, \ldots$, so that 0 is the minimal element.

Constructing the indexing function in this case is easy. Indeed, we have

$$I(0) = 0 \ , \quad I(a) = \begin{cases} 2a - 1 & \text{for } a > 0 \ , \\ -2a & \text{for } a < 0 \ . \end{cases} \tag{15}$$

The inverse function is also easily constructed:

$$I^{-1}(0) = 0 \ , \quad I^{-1}(l) = \begin{cases} (l+1)/2 & \text{for } l \text{ odd} \ , \\ -l/2 & \text{for } l \text{ even} \ . \end{cases} \tag{16}$$

In the rest of this section we show how an indexing function can be constructed for two particularly interesting cases, namely polynomials and trigonometric polynomials. However, we stress that the procedure we are using is a quite general one, so it can be extended to other interesting situations.

5. The polynomial case

Let us first take $\mathcal{A}_n = \mathbb{Z}^n_+$, i.e., integer vectors with non negative components; formally

$$\mathcal{A}_n = \{k = (k_1, \ldots, k_n) \in \mathbb{Z}^n \ : \ k_1 \geq 0, \ldots, k_n \geq 0\} \ .$$

The index n in $\mathcal{A}_n$ denotes the dimension of the space. This case is named “polynomial” because it occurs precisely in the representation of homogeneous polynomials, and so also in the Taylor expansion of a function of n variables: the integer vectors $k \in \mathcal{A}_n$ represent all possible exponents.

We shall denote by $|k| = k_1 + \ldots + k_n$ the length (or norm) of the vector $k \in \mathbb{Z}^n_+$. Furthermore, to a given vector $k = (k_1, \ldots, k_n) \in \mathcal{A}_n$ we shall associate the vector $t(k) \in \mathcal{A}_{n-1}$ (the *tail* of k) defined as $t(k) = (k_2, \ldots, k_n)$. This definition is meaningful only if $n > 1$, of course.

5.1. Ordering relation

Pick a fixed n, and consider the finite family of sets $\mathcal{A}_1 = \mathbb{Z}_+, \ldots, \mathcal{A}_n = \mathbb{Z}^n_+$. *Let $k, k' \in \mathcal{A}_m$, with any $1 \leq m \leq n$. We shall say that $k' \prec k$ in case one of the following conditions is true:*

(i) $m \geq 1 \ \wedge \ |k'| < |k|$;

(ii) $m > 1 \ \wedge \ |k'| = |k| \ \wedge \ k'_1 > k_1$;

(iii) $m > 1 \ \wedge \ |k'| = |k| \ \wedge \ k'_1 = k_1 \ \wedge \ t(k') \prec t(k)$.

In table 2 the ordering resulting from this definition is illustrated for the cases $m = 2, 3, 4, 5$.

If $m = 1$ then only (i) applies, and this ordering coincides with the natural one in $\mathbb{Z}_+$. For $m > 1$, if (i) and (ii) are both false, then (iii) means that one must decrease the dimension n by replacing k with its tail $t(k)$, and retry the comparison. For this reason the ordering has been established for $1 \leq m \leq n$. Eventually, one ends up with $m = 1$, to which only (i) applies.

Table 2. Ordering of integer vectors in $\mathbb{Z}_+^m$ for $m = 2, 3, 4, 5$.

$I(k)$	$m = 2$	$m = 3$	$m = 4$	$m = 5$
0	$(0,0)$	$(0,0,0)$	$(0,0,0,0)$	$(0,0,0,0,0)$
1	$(1,0)$	$(1,0,0)$	$(1,0,0,0)$	$(1,0,0,0,0)$
2	$(0,1)$	$(0,1,0)$	$(0,1,0,0)$	$(0,1,0,0,0)$
3	$(2,0)$	$(0,0,1)$	$(0,0,1,0)$	$(0,0,1,0,0)$
4	$(1,1)$	$(2,0,0)$	$(0,0,0,1)$	$(0,0,0,1,0)$
5	$(0,2)$	$(1,1,0)$	$(2,0,0,0)$	$(0,0,0,0,1)$
6	$(3,0)$	$(1,0,1)$	$(1,1,0,0)$	$(2,0,0,0,0)$
7	$(2,1)$	$(0,2,0)$	$(1,0,1,0)$	$(1,1,0,0,0)$
8	$(1,2)$	$(0,1,1)$	$(1,0,0,1)$	$(1,0,1,0,0)$
9	$(0,3)$	$(0,0,2)$	$(0,2,0,0)$	$(1,0,0,1,0)$
10	$(4,0)$	$(3,0,0)$	$(0,1,1,0)$	$(1,0,0,0,1)$
11	$(3,1)$	$(2,1,0)$	$(0,1,0,1)$	$(0,2,0,0,0)$
12	$(2,2)$	$(2,0,1)$	$(0,0,2,0)$	$(0,1,1,0,0)$
13	$(1,3)$	$(1,2,0)$	$(0,0,1,1)$	$(0,1,0,1,0)$
14	$(0,4)$	$(1,1,1)$	$(0,0,0,2)$	$(0,1,0,0,1)$
15	$(5,0)$	$(1,0,2)$	$(3,0,0,0)$	$(0,0,2,0,0)$
16	$(4,1)$	$(0,3,0)$	$(2,1,0,0)$	$(0,0,1,1,0)$
17	$(3,2)$	$(0,2,1)$	$(2,0,1,0)$	$(0,0,1,0,1)$
18	$(2,3)$	$(0,1,2)$	$(2,0,0,1)$	$(0,0,0,2,0)$
19	$(1,4)$	$(0,0,3)$	$(1,2,0,0)$	$(0,0,0,1,1)$
20	$(0,5)$	$(4,0,0)$	$(1,1,1,0)$	$(0,0,0,0,2)$
...	...	...	...	...

It is convenient to define $\mathcal{P}_n(k)$ as the set of the elements which precede k; formally:

$$\mathcal{P}_n(k) = \{k' \in \mathcal{A}_n \ :\ k' \prec k\} \ .$$

With the latter notation the indexing function is simply defined as $I(k) = \#\mathcal{P}_n(k)$. The following definitions are also useful. Pick a vector $k \in \mathcal{A}_n$, and define the sets $\mathcal{B}_n^{(i)}(k)$, $\mathcal{B}_n^{(ii)}(k)$ and $\mathcal{B}_n^{(iii)}(k)$ as the subsets of $\mathcal{A}_n$ satisfying (i), (ii) and (iii), respectively, in the ordering algorithm above. Formally:

$$\begin{aligned} \mathcal{B}_n^{(i)}(0) = \mathcal{B}_n^{(ii)}(0) = \mathcal{B}_n^{(iii)}(0) = \mathcal{B}_1^{(ii)}(k) = \mathcal{B}_1^{(iii)}(k) &= \emptyset \ , \\ \mathcal{B}_n^{(i)}(k) &= \{k' \in \mathcal{A}_n \ :\ |k'| < |k|\} \ , \\ \mathcal{B}_n^{(ii)}(k) &= \{k' \in \mathcal{A}_n \ :\ |k'| = |k| \ \wedge \ k'_1 > k_1\} \ , \\ \mathcal{B}_n^{(iii)}(k) &= \{k' \in \mathcal{A}_n \ :\ |k'| = |k| \ \wedge \ k'_1 = k_1 \ \wedge \ t(k') < t(k)\} \ . \end{aligned} \tag{17}$$

The sets $\mathcal{B}_n^{(i)}(k)$, $\mathcal{B}_n^{(ii)}(k)$ and $\mathcal{B}_n^{(iii)}(k)$ are pairwise disjoint, and moreover

$$\mathcal{B}_n^{(i)}(k) \cup \mathcal{B}_n^{(ii)}(k) \cup \mathcal{B}_n^{(iii)}(k) = \mathcal{P}_n(k) \ .$$

This easily follows from the definition.

5.2. Indexing function for polynomials

Let $k \in \mathcal{A}_n$. In view of the definitions and of the properties above the index function, defined as in (14), turns out to be

$$I(0) = 0 \ , \quad I(k) = \#\mathcal{B}_n^{(i)}(k) + \#\mathcal{B}_n^{(ii)}(k) + \#\mathcal{B}_n^{(iii)}(k) \ . \tag{18}$$

Let us introduce the functions

$$\begin{aligned} J(n,s) &= \#\{k \in \mathcal{A}_n \ : \ |k| = s\} \ , \\ N(n,s) &= \sum_{j=0}^{s} J(n,j) \qquad\qquad \text{for } n \geq 1 \, , \ s \geq 0 \ . \end{aligned} \tag{19}$$

These functions will be referred to in the following as *J-table and N-table.* We claim that the indexing function can be recursively computed as

$$\begin{aligned} I(0) &= 0 \ , \\ I(k) &= \begin{cases} N(n,|k|-1) & \text{for } k_1 = |k| \ , \\ N(n,|k|-1) + I(t(k)) & \text{for } k_1 < |k| \ . \end{cases} \end{aligned} \tag{20}$$

The claim follows from

$$\#\mathcal{B}_n^{(i)}(k) = N(n,|k|-1) \ ; \tag{21}$$

$$\#\mathcal{B}_n^{(ii)}(k) = \begin{cases} 0 & \text{for } k_1 = |k| \ , \\ N(n-1,|k|-k_1-1) & \text{for } k_1 < |k| \ ; \end{cases} \tag{22}$$

$$\#\mathcal{B}_n^{(iii)}(k) = \begin{cases} I(t(k)) & \text{for } k_1 = |k| \ , \\ I(t(k)) - N(n-1,|k|-k_1-1) & \text{for } k_1 < |k| \ . \end{cases} \tag{23}$$

The equality (21) is a straightforward consequence of the definition of the N-table. The equality (22) follows from (17). Indeed, for $|k| = k_1$ we have $\mathcal{B}_n^{(ii)}(k) = \emptyset$, and for $|k| > k_1$ we have

$$\begin{aligned} \mathcal{B}_n^{(ii)}(k) &= \bigcup_{k_1 < j \leq |k|} \{k' \in \mathcal{A}_n \ : \ k'_1 = j \ \wedge \ |t(k')| = |k| - j\} \\ &= \bigcup_{0 \leq l < |k| - k_1} \{k' \in \mathcal{A}_n \ : \ k'_1 = |k| - l \ \wedge \ |t(k')| = l\} \ ; \end{aligned}$$

Coming to (23), first remark that

$$\mathcal{B}_n^{(iii)}(k) = \{k' \in \mathcal{A}_n \ : \ k'_1 = k_1 \ \wedge \ |t(k')| = |k| - k_1 \ \wedge \ t(k') \prec t(k)\} \ ,$$

so that

$$\#\mathcal{B}_n^{(iii)}(k) = \#\{\lambda \in \mathcal{A}_{n-1} \ :\ |\lambda| = |k| - k_1 \ \wedge\ \lambda \prec t(k)\} \ .$$

Then, the equality follows by remarking that

$$\begin{aligned}\mathcal{P}_{n-1}(t(k)) = \{\lambda \in \mathcal{A}_{n-1} \ :\ |\lambda| = |k| - k_1 \ \wedge\ \lambda \prec t(k)\} \\ \cup \{\lambda \in \mathcal{A}_{n-1} \ :\ |\lambda| < |k| - k_1\} \ .\end{aligned}$$

Adding up all contributions (20) follows.

5.3. Construction of the tables

In view of (19) and (20) the indexing function is completely determined in explicit form by the J-table. We show now how to compute the J-table recursively. For $n = 1$ we have, trivially, $J(1,s) = 1$ for $s \geq 0$. For $n > 1$ use the elementary property

$$\{k \in \mathcal{A}_n \ :\ |k| = s\} = \bigcup_{0\leq j\leq s} \{k \in \mathcal{A}_n \ :\ k_1 = s - j \ \wedge\ |t(k)| = j\} \ .$$

Therefore

$$\begin{aligned} J(1,s) &= 1 \ , \\ J(n,s) &= \sum_{j=0}^{s} J(n-1,j) \quad \text{for } n > 1 \ . \end{aligned} \tag{24}$$

This also means that, according to (19), we have $N(n,s) = J(n+1,s)$.

By the way, one will recognize that the J-table is actually the table of binomial coefficients, being $J(n,s) = \binom{n+s-1}{n-1}$.

5.4. Inversion of the index function

The problem is to find the vector $k \in \mathbb{Z}_+^n$ corresponding to a given index l.

For $n = 1$ we have $I^{-1}(l) = l$, of course. Therefore, let us assume $n > 1$. We shall construct a recursive algorithm which calculates the inverse function by just showing how to determine k_1 and $I\bigl(t(k)\bigr)$.

(i) If $l = 0$, then $k = 0$, and there is nothing else to do.

(ii) If $l > 0$, find an integer s satisfying $N(n, s-1) \leq l < N(n,s)$. In view of (20) we have $|k| = s$ and $I(t(k)) = l - N(n, s-1)$. Hence, by the same method, we can determine $\bigl|t(k)\bigr|$, and so also $k_1 = s - \bigl|t(k)\bigr|$.

5.5. An example of implementation

We include here an example of actual implementation of the indexing scheme for polynomials. This is part of a program for the calculation of first integrals that is fully described in (Giorgilli 1979). The complete computer code is also available from the CPC program library.

We should mention that the FORTRAN code included here has been written in 1976. Hence it may appear a little strange to programmers who are familiar with the nowadays compilers, since it does not use many features that are available in FORTRAN 90 or in the current versions of the compiler. It rather uses the standard of FORTRAN II, with the only exception of the statement PARAMETER that has been introduced later.

The PARAMETERs included in the code allow the user to control the allocation of memory, and may be changed in order to adapt the program to different needs.
NPMAX is the maximum number of degrees of freedom
NORDMX is the maximal polynomial degree that will be used
NBN1 and NBN2 are calculated from the previous parameters, and are used in order to allocate the correct amount of memory for the table of binomial coefficients. Here are the statements:

```
      PARAMETER (NPMAX=3)
      PARAMETER (NORDMX=40)
      PARAMETER (NBN2=2*NPMAX)
      PARAMETER (NBN1=NORDMX+NBN2)
```

As explained in the previous sections, the indexing function for polynomials uses the table of binomial coefficients. The table is stored in a common block named BINTAB so that it is available to all program modules. In the same block there are also some constants that are used by the indexing functions and are defined in the subroutine BINOM below. Here is the statement that must be included in every source module that uses these data:

```
      COMMON  /BINTAB/  IBIN(NBN1,NBN2),NPIU1,NMEN1,NFAT,NBIN
```

Subroutine BINOM fills the data in the common block BINTAB. It must be called at the beginning of the execution, so that the constants become available. Forgetting this call will produce unpredictable results. The calling arguments are the following.
NLIB : the number of polynomial variables. In the Hamiltonian case considered in the present notes it must be set as $2n$, where n is the number of degrees of freedom. It must not exceed the value of the parameter NPMAX.
NORD : the wanted order of calculation of the polynomials, which in our case is the maximal order of the normal form. It must not exceed the value of the parameter NFAT.

The subroutine checks the limits on the calling arguments; if the limits are violated then the execution is terminated with an error message. The calculation of the part of the table of binomial coefficients that will be used is based on well known formulæ.

```
      SUBROUTINE BINOM(NLIB,NORD)
C
C     Compute the table of the binomial coefficients.
C
      COMMON  /BINTAB/  IBIN(NBN1,NBN2),NPIU1,NMEN1,NFAT,NBIN
C
      NFAT=NORD+NLIB
      NBIN=NLIB
      IF(NFAT.GT.NBN1.OR.NBIN.GT.NBN2) GO TO 10
      NPIU1 = NLIB+1
      NMEN1 = NLIB-1
      DO 1 I=1,NFAT
      IBIN(I,1) = I
      DO 1 K=2,NBIN
      IF(I-K) 2,3,4
2     IBIN(I,K) = 0
      GO TO 1
3     IBIN(I,K) = 1
      GO TO 1
4     IBIN(I,K) = IBIN(I-1,K-1)+IBIN(I-1,K)
1     CONTINUE
      RETURN
10    WRITE(6,1000) NFAT,NBIN
      STOP
1000  FORMAT(//,5X,15HERROR SUB BINOM,2I10,//)
      END
```

Function `INDICE` implements the calculation of the indexing function for polynomials. The argument `J` is an integer array of dimension `NLIB` which contains the exponents of the monomial. It must contain non negative values with sum not exceeding the value `NORD` initially passed to the subroutine `BINOM`. These limits are not checked in order to avoid wasting time: note that this function may be called several millions of times in a program. The code actually implements the recursive formula (20) using iteration. Recall that recursion was not implemented in `FORTRAN II`.

```
      FUNCTION INDICE(J,NLIB)
C
C     Compute the relative address I corresponding to the
C     exponents J.
C
      COMMON  /BINTAB/  IBIN(NBN1,NBN2),NPIU1,NMEN1,NFAT,NBIN
      DIMENSION J(NLIB)
C
      NP=NLIB+1
      INDICE = J(NLIB)
      M = J(NLIB)-1
      DO 1 I=2,NLIB
      IB=NP-I
      M = M + J(IB)
      IB=M+I
      INDICE = INDICE + IBIN(IB,I)
 1    CONTINUE
      RETURN
      END
```

Subroutine `ESPON` is the inverse of the indexing function. Given the index `N` it calculates the array `J` of dimension `NLIB` which contains the exponents. The value of `N` must be positive (not checked) and must not exceed the maximal index implicitly introduced by the initial choice of NLIB and NORD passed to BINOM. The latter error is actually checked (this does not increase the computation time). The code implements the recursive algorithm described in sect. 5.4, again using iteration.

```
      SUBROUTINE ESPON(N,J,NLIB)
C
C     Compute the exponents J correponding to the
C     index N.
C
      COMMON  /BINTAB/  IBIN(NBN1,NBN2),NPIU1,NMEN1,NFAT,NBIN
      DIMENSION J(NLIB)
C
      NM=NLIB-1
      NP=NLIB+1
      DO 1 K=NLIB,NFAT
      IF (N.LT.IBIN(K,NLIB)) GO TO 2
 1    CONTINUE
      WRITE(6,1000)
      STOP
 2    NN = K-1
      M = N-IBIN(NN,NLIB)
      IF(NLIB-2) 8,6,7
 7    DO 3 I = 2,NM
      L = NP-I
      DO 4 K=L,NFAT
      IF(M.LT.IBIN(K,L)) GO TO 5
 4    CONTINUE
 5    IB=NLIB-L
      J(IB) = NN-K
      NN = K-1
      M = M - IBIN(NN,L)
 3    CONTINUE
 6    J(NM) = NN-M-1
      J(NLIB) = M
      RETURN
 8    J(1)=N
      RETURN
 1000 FORMAT(//,5X,15HERROR SUB ESPON,//)
      END
```

The code described here is the skeleton of a program performing algebraic manipulation on polynomial. Such a program must include a call to BINOM in order to initialize the table of binomial coefficients.

In order to store the coefficient of a monomial with exponents J (an integer array with dimension NLIB the user must include a statement like

```
      K = INDICE(J,NLIB)
```

and then say, e.g., `F(K)=...` which stores the coefficient at the address `K` of the array `F`.

Suppose instead that we must perform an operation on all coefficients of degree `IORD` of a given function `F`. We need to perform a loop on all the corresponding indices and retrieve the corresponding exponents. Here is a sketch of the code.

```
C     Compute the minimum and maximum index NMIN and NMAX
C     of the coefficients of order IORD.
C
      IB=IORD+NMEN1
      NMIN = IBIN(IB,NLIB)
      IB=IORD+NLIB
      NMAX = IBIN(IB,NLIB) - 1
C
C     Loop on all coefficients
C
      DO 1 N = NMIN,NMAX
      CALL ESPON(N,J,NLIB)
      ... more code to operate on the coefficient F(N) ...
 1    CONTINUE
```

Let us add a few words of explanation. According to (20), the index of the first coefficient of degree s in n variables is $I(s, 0, \ldots, 0) = N(n, s-1)$, and we also have $N(n, s-1) = \binom{n+s-1}{n}$ as explained at the end of sect. 5.3. This explains how the limits `NMIN` and `NMAX` are calculated as $N(n, s-1)$ and $N(n, s+1) - 1$, respectively. The rest of the code is the loop that retrieves the exponents corresponding to the coefficient of index `N`.

6. Trigonometric polynomials

Let us now consider the more general case $\mathcal{A}_n = \mathbb{Z}^n$. The index n in $\mathcal{A}_n$ denotes again the dimension of the space. The name used in the title of the section is justified because this case occurs precisely in the representation of trigonometric polynomials, as explained in sect. 3.2.

We shall now denote by $|k| = |k_1| + \ldots + |k_n|$ the length (or norm) of the vector $k \in \mathbb{Z}^n$. The tail $t(k)$ of a vector k will be defined again as $t(k) = (k_2, \ldots, k_n)$.

6.1. Ordering relation

Pick a fixed n, and consider the finite family of sets $\mathcal{A}_1 = \mathbb{Z}, \ldots, \mathcal{A}_n = \mathbb{Z}^n$.

Let $k, k' \in \mathcal{A}_m$, with any $1 \le m \le n$. We shall say $k' \prec k$ in case one of the following conditions is true:

(i) $m \ge 1 \ \wedge \ |k'| < |k|$;

(ii) $m > 1 \ \wedge \ |k'| = |k| \ \wedge \ |k'_1| > |k_1|$;

(iii) $m \ge 1 \ \wedge \ |k'| = |k| \ \wedge \ |k'_1| = |k_1| \ \wedge \ k'_1 > k_1$;

(iv) $m > 1 \ \wedge \ |k'| = |k| \ \wedge \ k'_1 = k_1 \ \wedge \ t(k') \prec t(k)$.

Table 3. Ordering of integer vectors in $\mathbb{Z}^m$ for $m = 2, 3, 4$.

$I(k)$	$m = 2$	$m = 3$	$m = 4$
0	$(0, 0)$	$(0, 0, 0)$	$(0, 0, 0, 0)$
1	$(1, 0)$	$(1, 0, 0)$	$(1, 0, 0, 0)$
2	$(-1, 0)$	$(-1, 0, 0)$	$(-1, 0, 0, 0)$
3	$(0, 1)$	$(0, 1, 0)$	$(0, 1, 0, 0)$
4	$(0, -1)$	$(0, -1, 0)$	$(0, -1, 0, 0)$
5	$(2, 0)$	$(0, 0, 1)$	$(0, 0, 1, 0)$
6	$(-2, 0)$	$(0, 0, -1)$	$(0, 0, -1, 0)$
7	$(1, 1)$	$(2, 0, 0)$	$(0, 0, 0, 1)$
8	$(1, -1)$	$(-2, 0, 0)$	$(0, 0, 0, -1)$
9	$(-1, 1)$	$(1, 1, 0)$	$(2, 0, 0, 0)$
10	$(-1, -1)$	$(1, -1, 0)$	$(-2, 0, 0, 0)$
11	$(0, 2)$	$(1, 0, 1)$	$(1, 1, 0, 0)$
12	$(0, -2)$	$(1, 0, -1)$	$(1, -1, 0, 0)$
13	$(3, 0)$	$(-1, 1, 0)$	$(1, 0, 1, 0)$
14	$(-3, 0)$	$(-1, -1, 0)$	$(1, 0, -1, 0)$
15	$(2, 1)$	$(-1, 0, 1)$	$(1, 0, 0, 1)$
16	$(2, -1)$	$(-1, 0, -1)$	$(1, 0, 0, -1)$
17	$(-2, 1)$	$(0, 2, 0)$	$(-1, 1, 0, 0)$
18	$(-2, -1)$	$(0, -2, 0)$	$(-1, -1, 0, 0)$
19	$(1, 2)$	$(0, 1, 1)$	$(-1, 0, 1, 0)$
20	$(1, -2)$	$(0, 1, -1)$	$(-1, 0, -1, 0)$
21	$(-1, 2)$	$(0, -1, 1)$	$(-1, 0, 0, 1)$
22	$(-1, -2)$	$(0, -1, -1)$	$(-1, 0, 0, -1)$
23	$(0, 3)$	$(0, 0, 2)$	$(0, 2, 0, 0)$
24	$(0, -3)$	$(0, 0, -2)$	$(0, -2, 0, 0)$
...	...	...	...

In table 3 the order resulting from this definition is illustrated for the cases $m = 2, 3, 4$.

If $m = 1$ this ordering coincides with the ordering in $\mathbb{Z}$ introduced in sect 4. For $m > 1$, if (i), (ii) and (iii) do not apply, then (iv) means that one must decrease the dimension n by replacing k with its tail $t(k)$, and retry the comparison. Eventually, one ends up with $m = 1$, falling back to the one dimensional case to which only (i) and (iii) apply.

The ordering in this section has been defined for the case $\mathcal{A}_n = \mathbb{Z}^n$. However, it will be useful to consider particular subsets of $\mathbb{Z}^n$. The natural choice will be to use again the ordering relation defined here. For example, the case of integer vectors with non negative components discussed in sect. 5.1 can be considered as a particular case: the restriction of the ordering relation to that case gives exactly the order introduced in sect. 5.1. Just remark that the condition (iii) above becomes meaningless in that case, so that it can be removed.

The set $\mathcal{P}_n(k)$ of the elements preceding $k \in \mathcal{A}^n$ in the order above is defined as in sect. 5.1. Following the line of the discussion in that section it is also convenient to give some more definitions. Pick a vector $k \in \mathcal{A}_n$, and define the sets $\mathcal{B}_n^{(i)}(k)$, $\mathcal{B}_n^{(ii)}(k)$, $\mathcal{B}_n^{(iii)}(k)$ and $\mathcal{B}_n^{(iv)}(k)$ as the subsets of $\mathcal{A}_n$ satisfying (i), (ii), (iii) and (iv), respectively, in the ordering algorithm above. Formally,

$$\begin{aligned}
\mathcal{B}_n^{(i)}(0) = \mathcal{B}_n^{(ii)}(0) = \mathcal{B}_n^{(iii)}(0) = \mathcal{B}_n^{(iv)}(0) = \mathcal{B}_1^{(ii)}(k) = \mathcal{B}_1^{(iv)}(k) &= \emptyset \ , \\
\mathcal{B}_n^{(i)}(k) &= \{k' \in \mathcal{A}_n \ : \ |k'| < |k|\} \ , \\
\mathcal{B}_n^{(ii)}(k) &= \{k' \in \mathcal{A}_n \ : \ |k'| = |k| \ \land \ |k'_1| > |k_1|\} \ , \\
\mathcal{B}_n^{(iii)}(k) &= \{k' \in \mathcal{A}_n \ : \ |k'| = |k| \ \land \ |k'_1| = |k_1| \ \land \ k'_1 > k_1\} \ , \\
\mathcal{B}_n^{(iv)}(k) &= \{k' \in \mathcal{A}_n \ : \ |k'| = |k| \ \land \ k'_1 = k_1 \ \land \ t(k') < t(k)\} \ .
\end{aligned} \tag{25}$$

The sets $\mathcal{B}_n^{(i)}(k)$, $\mathcal{B}_n^{(ii)}(k)$, $\mathcal{B}_n^{(iii)}(k)$ and $\mathcal{B}_n^{(iv)}(k)$ are pairwise disjoint, and moreover

$$\mathcal{B}_n^{(i)}(k) \cup \mathcal{B}_n^{(ii)}(k) \cup \mathcal{B}_n^{(iii)}(k) \cup \mathcal{B}_n^{(iv)}(k) = \mathcal{P}(k) \ .$$

This easily follows from the definition.

6.2. Indexing function for trigonometric polynomials

Let $k \in \mathcal{A}_n$. In view of the definitions and of the properties above the index function, defined as in (14), turns out to be

$$I(0) = 0 \ , \quad I(k) = \#\mathcal{B}_n^{(i)}(k) + \#\mathcal{B}_n^{(ii)}(k) + \#\mathcal{B}_n^{(iii)}(k) + \#\mathcal{B}_n^{(iv)}(k) \ . \tag{26}$$

Let us introduce the J-table and the N-table as

$$\begin{aligned}
J(n,s) &= \#\{k \in \mathcal{A}_n \ : \ |k| = s\} \ , \\
N(n,s) &= \sum_{j=0}^{s} J(n,j) \qquad\qquad \text{for } n \geq 1 \ , \ s \geq 0 \ .
\end{aligned} \tag{27}$$

We claim that the index function can be recursively computed as

$$I(0) = 0 \ ,$$

$$I(k) = \begin{cases} N(n,|k|-1) & \text{for } |k_1| = |k| \\ & \quad \text{and } k_1 \geq 0 \ , \\ N(n,|k|-1)+1 & \text{for } |k_1| = |k| \\ & \quad \text{and } k_1 < 0 \ , \\ N(n,|k|-1) + N(n-1,|k|-|k_1|-1) + I(t(k)) & \text{for } |k_1| < |k|, \\ & \quad \text{and } k_1 \geq 0 \ , \\ N(n,|k|-1) + N(n-1,|k|-|k_1|) + I(t(k)) & \text{for } |k_1| < |k| \\ & \quad \text{and } k_1 < 0 \ , \end{cases} \tag{28}$$

This formula follows from

$$\#\mathcal{B}_n^{(i)}(k) = N(n,|k|-1) \ ; \tag{29}$$

$$\#\mathcal{B}_n^{(ii)}(k) = \begin{cases} 0 & \text{for } |k_1| = |k| \ , \\ 2N(n-1,|k|-|k_1|-1) & \text{for } |k_1| < |k| \ ; \end{cases} \tag{30}$$

$$\#\mathcal{B}_n^{(iii)}(k) = \begin{cases} 0 & \text{for } |k_1| \leq |k| \ \wedge \ k_1 \geq 0 \ , \\ J(n-1,|k|-|k_1|) & \text{for } |k_1| \leq |k| \ \wedge \ k_1 < 0 \ ; \end{cases} \tag{31}$$

$$\#\mathcal{B}_n^{(iv)}(k) = \begin{cases} I(t(k)) & \text{for } |k_1| = |k| \ , \\ I(t(k)) - N(n-1,|k|-|k_1|-1) & \text{for } |k_1| < |k| \ . \end{cases} \tag{32}$$

The equality (29) is a straightforward consequence of the definition (25). The equality (30) follows by remarking that for $|k_1| = |k|$ we have $\mathcal{B}_n^{(ii)}(k) = \emptyset$, and for $|k_1| < |k|$ we have

$$\mathcal{B}_n^{(ii)}(k) = B_n^{+}(k) \cup B_n^{-}(k) \ , \quad B_n^{+}(k) \cap B_n^{-}(k) = \emptyset \ ,$$

with

$$B_n^{+}(k) = \bigcup_{0 \leq l < |k|-|k_1|} \{k' \in \mathcal{A}_n \ : \ k'_1 = |k| - l \ \wedge \ |t(k)| = l\} \ ,$$

$$B_n^{-}(k) = \bigcup_{0 \leq l < |k|-|k_1|} \{k' \in \mathcal{A}_n \ : \ k'_1 = l - |k| \ \wedge \ |t(k)| = l\} \ ;$$

use also $\#B_n^{+}(k) = \#B_n^{-}(k)$. The equality (31) follows from

$$\mathcal{B}_n^{(iii)}(k) = \begin{cases} \emptyset & \text{for } |k_1| = |k| \\ & \quad \text{and } k_1 \geq 0 \ , \\ \{k' \in \mathcal{A}_n \ : \ k'_1 = |k_1| \ \wedge \ |t(k')| = |k| - |k_1|\} & \text{for } |k_1| = |k| \\ & \quad \text{and } k_1 < 0 \ . \end{cases}$$

Coming to (32), remark that

$$\mathcal{B}_n^{(iv)}(k) = \{k' \in \mathcal{A}_n \ :\ |t(k')| = |k| - |k_1| \ \wedge\ t(k') \prec t(k)\} \ .$$

Proceeding as in the polynomial case we find again

$$\#\mathcal{B}_n^{(iv)}(k) = \#\{\lambda \in \mathcal{A}_{n-1} \ :\ |\lambda| = |k| - |k_1| \ \wedge\ \lambda \prec t(k)\} \ ,$$

and (32) follows by remarking that

$$\begin{aligned}\mathcal{P}_{n-1}(t(k)) = \{\lambda \in \mathcal{A}_{n-1} \ &:\ |\lambda| = |k| - |k_1| \ \wedge\ \lambda \prec t(k)\} \\ \cup \{\lambda \in \mathcal{A}_{n-1} \ &:\ |\lambda| < |k| - |k_1|\} \ .\end{aligned}$$

Adding up all contributions (28) follows.

6.3. Construction of the tables

We show now how to construct recursively the J-table, so that the N-table can be constructed, too. For $n = 1$ we have, trivially, $J(1,0) = 1$ and $J(1,s) = 2$ for $s > 0$. For $n > 1$ use the elementary property

$$\{k \in \mathcal{A}_n \ :\ |k| = s\} = \bigcup_{-s \le j \le s} \{k \in \mathcal{A}_n \ :\ k_1 = j \ \wedge\ |t(k)| = s - |j|\} \ .$$

Therefore

$$\begin{aligned} J(1,0) &= 1 \ , \\ J(1,s) &= 2 \ , \\ J(n,s) &= \sum_{j=-s}^{s} J(n-1, s-|j|) \quad \text{for } n > 1 \ . \end{aligned} \tag{33}$$

This completely determines the J-table.

6.4. Inversion of the index function

The problem is to find the vector k of given dimension n corresponding to the given index l. For $n = 1$ the function $I(k)$ and its inverse $I^{-1}(l)$ are given by (15) and (16). Therefore in the rest of this section we shall assume $n > 1$. We shall give a recursive algorithm, showing how to determine k_1 and $I\bigl(t(k)\bigr)$.

(i) If $l = 0$ then $k = 0$, and there is nothing else to do.

(ii) Assuming that $l > 0$, determine s such that

$$N(n, s-1) \le l < N(n,s) \ .$$

From this we know that $|k| = s$.

(iii) Define $l_1 = l - N(n, s-1)$, so that $I\bigl(t(k)\bigr) \le l_1$ by (28). If $l_1 = 0$ set $s_1 = 0$; else, determine s' such that

$$N(n-1, s'-1) \le l_1 < N(n-1, s') \ ;$$

and let $s_1 = \min(s', s)$. In view of $I\bigl(t(k)\bigr) \le l_1$ we know that $|t(k)| \le s_1$. Remark also that $s_1 = 0$ if and only if $l_1 = 0$. For, if $s_1 \ge 1$ then we have $l_1 \ge N(n-1, 0) = 1$.

(iv) If $l_1 = 0$, then by the first of (28) we conclude

$$k_1 = |k| = s \ , \quad t(k) = 0 \ ,$$

and there is nothing else to do.

(v) If $l_1 = 1$, then by the second of (28) we conclude

$$k_1 = -|k| = -s \ , \quad t(k) = 0 \ ,$$

and there is nothing else to do.

(vi) If $l_1 > 1$ and $s_1 > 0$, we first look if we can set $0 \le k_1 < |k|$. In view of the third of (28) we should have

$$|k| - k_1 = s_1 \ , \quad \bigl|t(k)\bigr| = s_1 \ , \quad I\bigl(t(k)\bigr) = l_1 - N(n-1, s_1-1) \ .$$

This can be consistently made provided the conditions

$$s_1 > 0 \quad \text{and} \quad I\bigl(t(k)\bigr) \ge N(n-1, s_1-1)$$

are fulfilled. The condition $s > 0$ is already satisfied. By (28), the second condition is fulfilled provided $l_1 \ge 2N(n-1, s_1-1)$. This has to be checked.

(vi.a) If the second condition is true, then set $k_1 = |k| - s_1$, and recall that $\bigl|t(k)\bigr| = s_1$. Hence, we can replace n, l, and s by $n-1$, $l_1 - N(n-1, s_1-1)$ and s_1, respectively, and proceed by recursion restarting again from the point (iii).

(vi.b) If the second condition is false, then we proceed with the next point.

(vii) Recall that $l_1 > 1$, and remark that we have also $s_1 > 1$. Indeed, we already know $s_1 > 0$, so we have to exclude the case $s_1 = 1$. Let, by contradiction, $s_1 = 1$. Then we have $l_1 \ge 2 = 2N(n-1, s_1-1)$, which is the case already excluded by (vi). We conclude $s_1 > 1$. We look now for the possibility of setting $|k_1| < |k|$ and $k_1 < 0$. In view of the fourth of (28) we should have

$$|k| + k_1 = s_1 - 1 \ , \quad \bigl|t(k)\bigr| = s_1 - 1 \ , \quad I\bigl(t(k)\bigr) = l_1 - N(n-1, s_1-1) \ .$$

This can be consistently made provided the conditions

$$s_1 > 1 \quad \text{and} \quad I\bigl(t(k)\bigr) \ge N(n-1, s_1-2)$$

are fulfilled. The condition $s_1 > 1$ is already satisfied. As to the second condition, by (28) it is fulfilled provided $l_1 > N(n-1, s_1-1) + N(n-1, s_1-2)$. This has to be checked.

(vii.a) If the second condition is true, then set $k_1 = -|k| + s_1 - 1$, and recall that $|t(k)| = s_1 - 1$. Hence, we can replace n, l, and s by $n-1$, $l_1 - N(n-1, s_1-2)$ and $s_1 - 1$, respectively, and proceed by recursion restarting again from the point (iii).

(vii.b) If the second condition is false we must decrease s_1 by one and start again with the point (vi); remark that $s_1 > 1$ implies $s_1 - 1 > 0$, which is the first of the two conditions to be satisfied at the point (vi), hence the recursion is correct.

Since $l_1 > 1$ we have $l_1 > 2N(n-1, 0)$, so that the conditions of point (vi) are satisfied for $s = 1$. Hence the algorithm above does not fall into an infinite loop between points (vi) and (vii). On the other hand, for $n = 1$ either (iii) or (iv) applies, so that the algorithm stops at that point.

7. Storing the coefficients for sparse functions

The method of storing the coefficient using the index, as illustrated in sect. 3, is the most direct one, but reveals to be ineffective when most of the coefficients of a function are zero (sparse function). For, allocating memory space for all coefficients results in a wasting of memory.

A method that we often use is to store the coefficients using a tree structure based on the index. However we should warn the reader that the method described here has the advantage of being easily programmed, but does not pretend to be the most effective one. Efficient programming of tree structure is described, e.g., in the monumental books *The art of computing programming*, by D.E. Knuth (1968).

7.1. The tree structure

The first information we need is how many bits are needed in order to represent the maximum index for a function. We shall refer to this number as the *length of the index*. In the scheme that we are presenting here this is actually the length of the path from the root of the tree to its leave, where the coefficient is found.

In fig. 1 we illustrate the scheme assuming that 4 bits are enough, i.e., there are at most 16 coefficients indexed from 0 to 15. The case is elementary, of course, but the method is the general one, and is extended to, e.g., several millions of coefficients (with a length a little more than 20) in a straightforward manner. The bits are labeled by their position, starting from the less significant one (choosing the most significant one as the first bit is not forbidden, of course, and sometimes may be convenient). The label of the bit corresponds to a level in the tree structure, level 0 being the root and level 3 being the last one, in our case. At level zero we find a cell containing two pointers, corresponding to the digit 0 and 1, respectively. To each digit we associate a cell of level 1, which contains a pair of pointers, and so on until we reach the last level (3 in our case). Every number that may be represented with 4 bits generates a unique path along the tree, and the last cell contains pointers to the coefficient. The example in the figure represents the path associated with the binary index 1010, namely 10 in decimal notation.

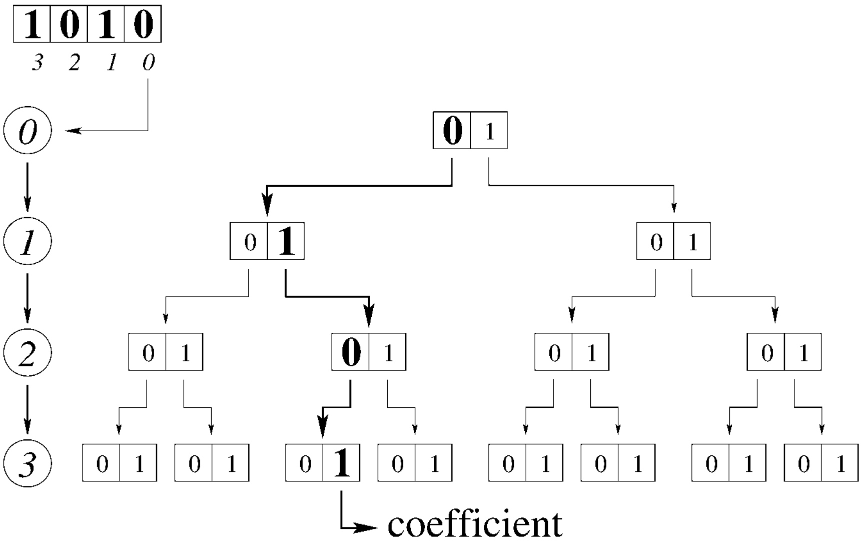

Figure 1. Illustrating the tree structure for a 4-bit long index (see text).

Let us also illustrate how this structure may be represented in memory, trying to avoid wasting of space. We use two separate arrays, the first one for pointers and the second one for the coefficients, as illustrated in fig. 2. The cells containing pairs of pointers are allocated in the first array, the root of the tree having label zero. The label of a cell is always even: the first element corresponds to the zero bit, the next one (with odd label) to the bit one.

The arrays are initially allocated with appropriate size, and are cleared. A good method is to fill the array of pointers with -1 (denoting an unused pointer) and the coefficients table with zeros. We also keep track of the first unused cell in the array, which initially is set to 2 because the root cell is considered to be in use, and of the first free coefficient, which initially is 0.

We shall use the following notations: $\texttt{cell}(2j)$ is the cell with even label $2j$ in the array; $\texttt{cell}(2j, 0)$ and $\texttt{cell}(2j, 1)$ are the pointers corresponding to a bit 0 or 1 which are stored at locations $2j$ and $2j + 1$, respectively, in the array of pointers; $\texttt{coef}(j)$ is the j-th element of the array of coefficients; `cc` is the current cell and `cb` is the current bit (see below for the meaning); `fp` is the label of the first free (unused) cell of pointers; `fc` is the label of the first free coefficient; ℓ is the length of the index.

7.2. Storing the first coefficient

Let us describe how the first coefficient is stored. Suppose we want to store the value x as the coefficient corresponding to a given index. Here is the scheme.

(i) *Initialization:* set $\texttt{cc} = 0$ and $\texttt{cb} = 0$. The values of $\texttt{fp} = 2$ and $\texttt{fc} = 0$ have already been set when during the array allocation.

(ii) *Creating a path:* repeat the following steps until `cb` equals $\ell - 1$:

(ii.a) if the bit at position `cb` in the index is 0, then redefine $\texttt{cell}(\texttt{cc}, 0) = \texttt{fp}$; else redefine $\texttt{cell}(\texttt{cc}, 1) = \texttt{fp}$;

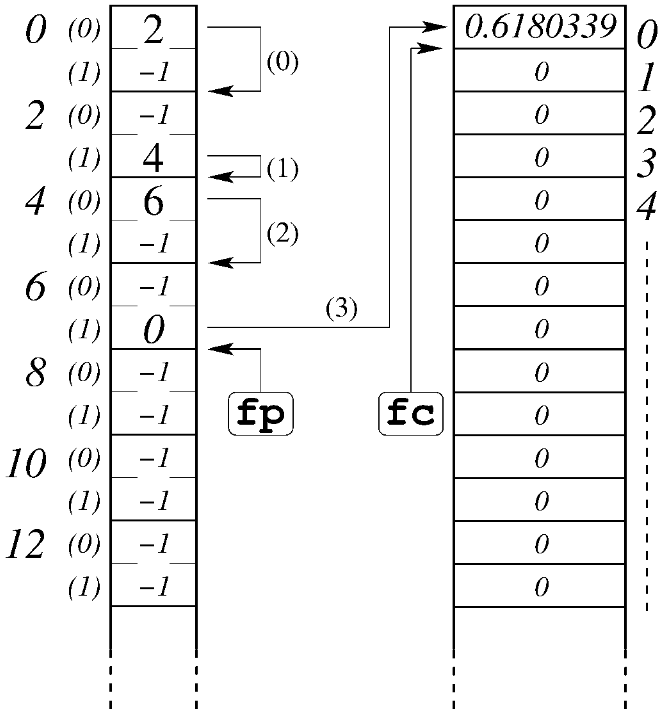

Figure 2. Illustrating how the tree structure is stored in memory (see text).

(ii.b) set $\texttt{cc} = \texttt{fp}$ and increment $\texttt{fp}$ by 2 (point to the next free cell);

(ii.c) increment $\texttt{cb}$ by 1 (next bit).

(iii) *Store the coefficient:*

(iii.a) if the bit at position $\texttt{cb}$ in the index is 0, then redefine $\texttt{cell}(\texttt{cc}, 0) = \texttt{fc}$; else redefine $\texttt{cell}(\texttt{cc}, 1) = \texttt{fc}$;

(iii.b) set $\texttt{coef}(\texttt{fc}) = x$;

(iii.c) increment $\texttt{fc}$ by 1 (point to the next free coefficient).

Programming this algorithm in a language such as `C` or `FORTRAN` requires some 10 to 20 statements.

Let us see in detail what happens if we want to store the coefficient 0.6180339 with index 1010 and $\ell = 4$, as illustrated in fig. 2. Here is the sequence of operations actually made

step (i):	$\texttt{cc} = 0$,	$\texttt{cb} = 0$,	$\texttt{fp} = 2$,	$\texttt{fc} = 0$;	
step (ii):	$\texttt{cell}(0,0) = 2$,	$\texttt{cc} = 2$,	$\texttt{fp} = 4$,	$\texttt{cb} = 1$,	then ,
	$\texttt{cell}(2,1) = 4$,	$\texttt{cc} = 4$,	$\texttt{fp} = 6$,	$\texttt{cb} = 2$,	then ,
	$\texttt{cell}(4,0) = 6$,	$\texttt{cc} = 6$,	$\texttt{fp} = 8$,	$\texttt{cb} = 3$,	end of loop ;
step (iii):	$\texttt{cell}(6,1) = 0$,	$\texttt{coef}(0)$=0.6180339 ,		$\texttt{fc} = 1$,	end of game .

After this, the contents of the arrays are as represented in fig. 2.

7.3. Retrieving a coefficient

The second main operation is to retrieve a coefficient, which possibly has never been stored. In the latter case, we assume that the wanted coefficient is zero. Here is a scheme.

(i) *Initialization:* set $\mathtt{cc} = 0$ and $\mathtt{cb} = 0$.

(ii) *Follow a path:* repeat the following steps until $\mathtt{cb}$ equals ℓ:

- **(ii.a)** save the current value of $\mathtt{cc}$;
- **(ii.b)** if the bit at position $\mathtt{cb}$ in the index is 0, then redefine $\mathtt{cc}$ as $\mathtt{cell}(\mathtt{cc}, 0)$; else redefine $\mathtt{cc}$ as $\mathtt{cell}(\mathtt{cc}, 1)$;
- **(ii.c)** if $\mathtt{cc} = -1$ then the coefficient is undefined. Return 0 as the value of the coefficient;
- **(ii.d)** increment $\mathtt{cb}$ by 1 (next bit).

(iii) *Coefficient found:* return the coefficient $\mathtt{coef}(\mathtt{cc})$.

Let us give a couple of examples in order to better illustrate the algorithm. Suppose that we are looking for the coefficient corresponding to the binary index 1010. By following the algorithm step by step, and recalling that in our example the length of the index is 4, the reader should be able to check that the sequence of operations is the following:

$$
\begin{array}{llll}
\text{step (i):} & \mathtt{cc} = 0\ , & \mathtt{cb} = 0\ ; & \\
\text{step (ii):} & \mathtt{cc} = 2\ , & \mathtt{cb} = 1\ , & \text{then}\ , \\
 & \mathtt{cc} = 4\ , & \mathtt{cb} = 2\ , & \text{then}\ , \\
 & \mathtt{cc} = 6\ , & \mathtt{cb} = 3\ , & \text{then}\ , \\
 & \mathtt{cc} = 0 & \mathtt{cb} = 4\ , & \text{end of path}\ ; \\
\text{step (iii):} & & & \text{return } 0.6180339\ .
\end{array}
$$

The returned value is that of $\mathtt{coef}(0)$, stored in the location 0 of the coefficients array.

Suppose now that we are looking for the coefficient corresponding to the binary index 1110. Here is the actual sequence of operations:

$$
\begin{array}{llll}
\text{step (i):} & \mathtt{cc} = 0\ , & \mathtt{cb} = 0\ ; & \\
\text{step (ii):} & \mathtt{cc} = 2\ , & \mathtt{cb} = 1\ , & \text{then}\ , \\
 & \mathtt{cc} = 4\ , & \mathtt{cb} = 2\ , & \text{then}\ , \\
 & \mathtt{cc} = -1\ , & \mathtt{cb} = 2\ , & \text{return zero}\ .
\end{array}
$$

Here the algorithm stops because a coefficient has not been found.

7.4. Other operations

Having implemented the two operations above, the reader should be able to implement also the following operations:

(i) storing a new coefficient corresponding to a given index;

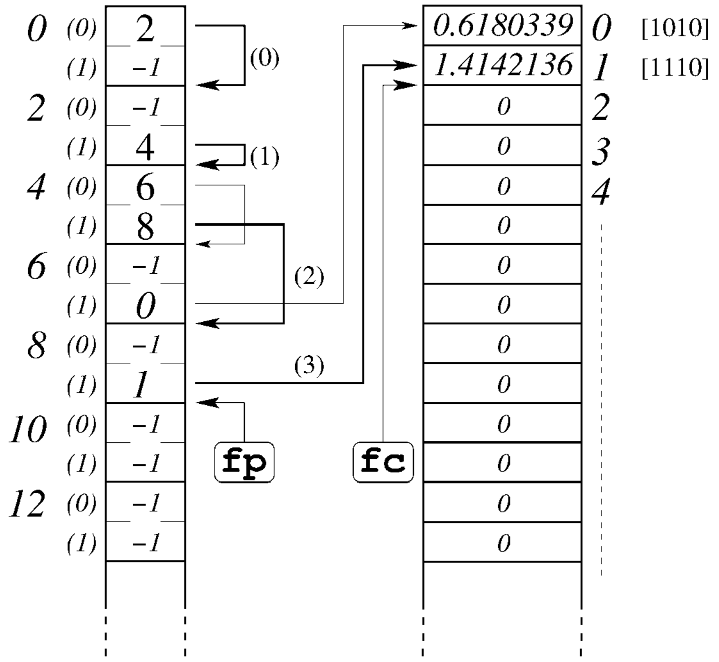

Figure 3. Inserting a new coefficient in a tree structure (see text).

(ii) adding something to a given coefficient;

(iii) multiplying a given coefficient by a number.

These are the basic operations that we need in order to perform an elementary computer algebra. Let us add a few hints.

Storing a new coefficient requires perhaps some moment of thinking. Using the index, one should follow the corresponding path in the tree (as in the operation of retrieving) until either happens: the coefficient is found, or the search fails at some point. If the coefficient is found, then it can be overwritten if the new value has to replace the old one. On failure, the path must be completed by appropriately defining the pointers (as in the case of the first coefficient), and then the coefficient can be stored in the appropriate location. As an exercise, suppose that we want to store the coefficient 1.4142136 corresponding to the binary index 1110. After completing the operation the memory should look as in fig. 3.

Adding something to a given coefficient is not very different from the previous operation. Just follow the path. If the coefficient is found, then add the wanted value to it. On failure, just change the "add" operation to a "store" one, and proceed as in the case (i).

Multiplying a coefficient by a constant is even easier. If the coefficient is found, then do the multiplication. On failure, just do nothing.

Further operations can be imagined, but we think that we have described the basic ones. There are just a couple of remarks.

The method illustrated here uses an amount of memory that clearly depends on the number of non zero coefficients of a function. However, this amount is typically not known in advance. Thus, enough memory should be allocated at the beginning in order to assure that there is enough room. When a function is filled, and we know that it will not be changed, the excess of memory can be freed and reused for other purposes. Every operating system and language provides functions that allow the programmer to allocate memory blocks and resize them on need.

A second remark is that other storing methods can be imagined. E.g., once a function is entirely defined it may be more convenient to represent it as a sequential list of pairs (index, coefficient). This is definitely a very compact representation for a sparse function (although not the best for a crowded one).

8. Applications

We report here some examples of application of algebraic manipulation that have been obtained by implementing the formal algorithm of sect. 2. We consider three cases, namely the model of Hénon and Heiles, the Lagrangian triangular equilibria for the Sun-Jupiter system and the planetary problem including Sun, Jupiter, Saturn and Uranus (SJSU).

8.1. The model of Hénon and Heiles

A wide class of canonical system with Hamiltonian of the form

$$H(x,y) = \frac{\omega_1}{2}(y_1^2 + x_1^2) + \frac{\omega_2}{2}(y_2^2 + x_2^2) + x_1^2 x_2 \tag{34}$$

has been studied by Contopoulos, starting at the end of the fifties, for different values of the frequencies. This approximates the motion of a star in a galaxy, at different distances from the center. A wide discussion on the use of these models in galactic dynamics and on the construction of the so called “third integral” can be found in the book of Contopoulos (2002). The third integral is constructed as a power series $\Phi = \Phi_2 + \Phi_3 + \ldots$ where Φ_s is a homogeneous polynomial of degree s which is the solution of the equation $\{H, \Phi\} = 0$, where $\{\cdot, \cdot\}$ is the Poisson bracket (see, e.g., Whittaker 1916 or Contopoulos 1960). A different method is based on the construction of the Birkhoff normal form (Birkhoff 1927).

A particular case with two equal frequencies and Hamiltonian

$$H(x,y) = \frac{1}{2}(y_1^2 + x_1^2) + \frac{1}{2}(y_2^2 + x_2^2) + x_1^2 x_2 - \frac{1}{3}x_2^3 \tag{35}$$

has been studied by Hénon and Heiles in 1964. This work has become famous since for the first time the existence of a chaotic behavior in a very simple system has been stressed, showing some figures. It should be remarked that the existence of chaos had been discovered by Poincaré in his memory on the problem of three bodies (Poincare 1889), but it had been essentially forgotten.

A program for the construction of the third integral has been implemented by Contopoulos since 1960. He made several comparisons between the level lines of the integral so found on the surface of constant energy and the figures given by the Poincaré sections of the orbits. A similar calculation for the case of Hénon and Heiles has been made by Gustavson (1966), who used the normal form

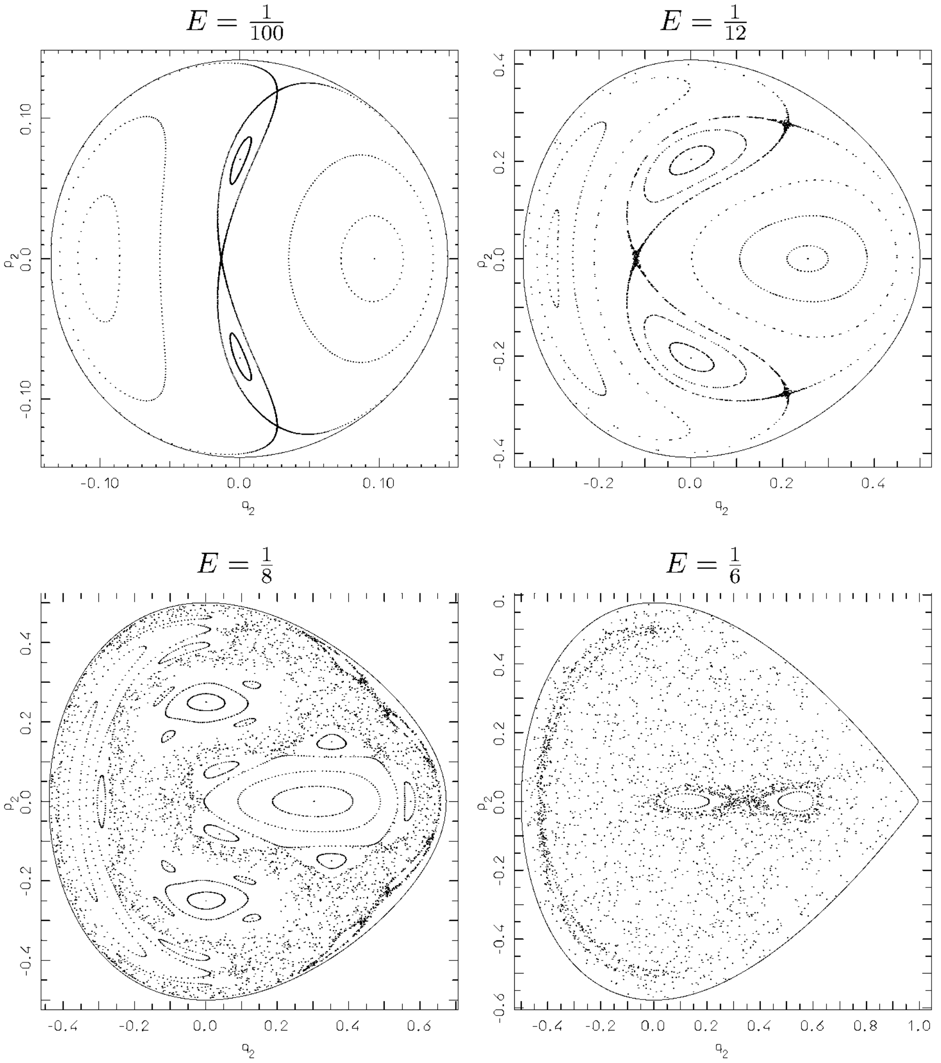

Figure 4. Poincaré sections for the Hénon and Heiles model. The energies are as in the original paper.

method. The third integral was expanded up to order 8, which may seem quite low today, but it was really difficult to do better with the computers available at that time. Here we reproduce the figures of Gustavson extending the calculation up to order 58, which is now easily reached even on a PC.

In fig. 4 we show the Poincaré sections for the values of energy used by Hénon and Heiles in their paper. As stressed by the authors, an essentially ordered motion is found for $E < \frac{1}{12}$, while the chaotic orbits become predominant at higher energies.

The comparison with the level lines of the third integral at energy $E = \frac{1}{100}$ is reported in fig. 5. The correspondence with the Poincaré sections is evident even at order 8, as calculated also by Gustavson. We do not produce the figures

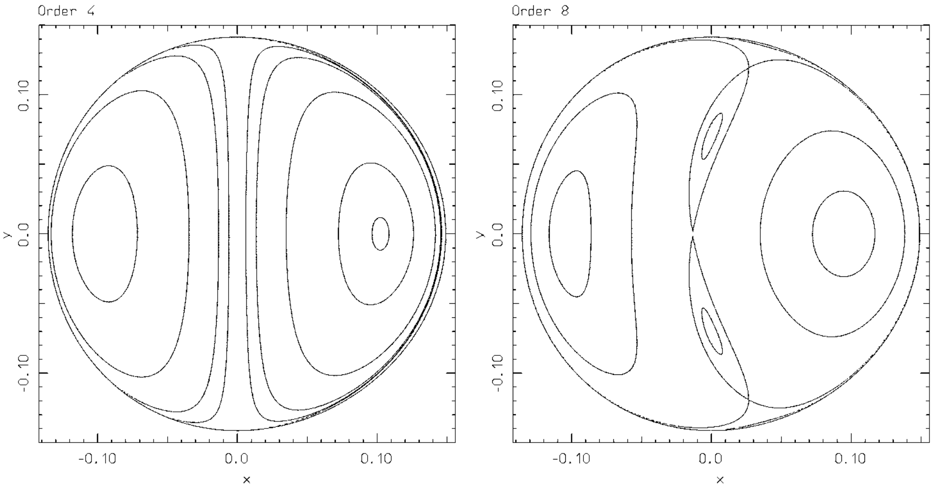

Figure 5. Level lines of the first integral truncated at orders 4 and 8, for energy $E = \frac{1}{100}$. The figure for truncation orders up to 58 are actually the same as for order 8.

for higher orders because they are actually identical with the one for order 8. This may raise the hope that the series for the first integral is a convergent one.

Actually, a theorem of Siegel states that for the Birkhoff normal form divergence is a typical case (Siegel 1941). A detailed numerical study has been made in (Contopolus 2003 and 2004), showing the mechanism of divergence. Moreover, it was understood by Poincaré that perturbations series typically have an asymptotic character (see Poincare 1892, Vol. II). Estimates of this type have been given, e.g., in (Giorgilli 1988 and 1989).

For energy $E = \frac{1}{12}$ (fig. 6) the asymptotic character of the series starts to appear. Indeed already at order 8 we have a good correspondence between the level lines and the Poincaré section, as was shown also Gustavson's paper. If we increase the approximation we see that the correspondence remains good up to order 32, but then the divergence of the series shows up, since at order 43 an unwanted "island" appears on the right side of the figure which has no correspondent in the actual orbits, and at order 58 a bizarre behavior shows up.

The phenomenon is much more evident for energy $E = \frac{1}{8}$ (fig. 7). Here some rough correspondence is found around order 9, but then the bizarre behavior of the previous case definitely appears already at order 27.

The non convergence of the normal form is illustrated in fig. 8. Writing the homogeneous terms of degree s of the third integral as $\Phi_s = \sum_{j,k} \varphi_{j,k} x^j y^k$, we may introduce the norm

$$\|\Phi_s\| = \sum_{j,k} |\varphi_{j,k}| \ .$$

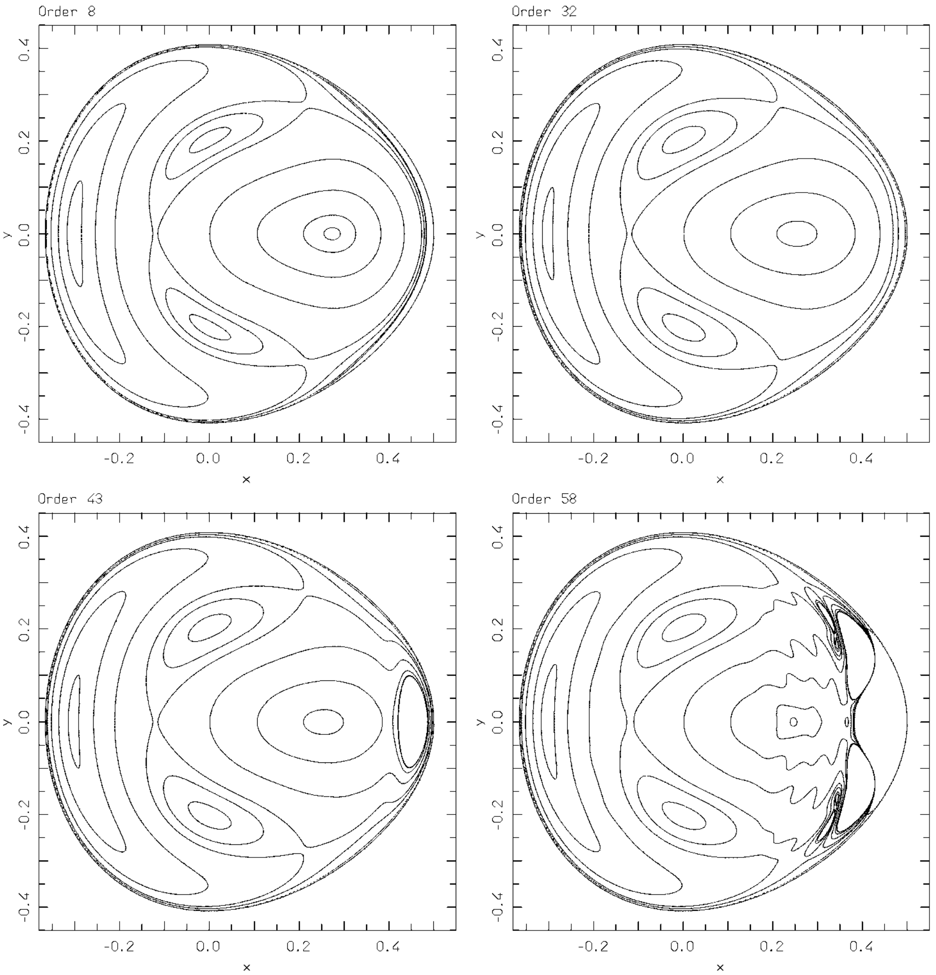

Figure 6. Level lines of the first integral truncated at orders 8, 32, 43 and 58, for energy $E = \frac{1}{12}$. A good correspondence with the Poincaré sections is found at orders, roughly, 8 to 32. Then the level lines start to disprove, in agreement with the asymptotic character of the series.

Then an indication of the convergence radius may be found by calculating one of the quantities

$$\|\Phi_s\|^{1/s}\ ,\quad \frac{\|\Phi_s\|}{\|\Phi_{s-1}\|}\ ,\quad \left(\frac{\|\Phi_s\|}{\|\Phi_{s-2}\|}\right)^{1/2}\ .$$

The first quantity corresponds to the root criterion for power series. The second one corresponds to the ratio criterion. The third one is similar to the ratio criterion, but in the present case turns out to be more effective because it takes into account the peculiar behavior of the series for odd and even degrees. The values given by the root criterion are plotted in the left panel of fig. 8. The data for the ratio criterion are plotted in the right panel, where open dots and solid

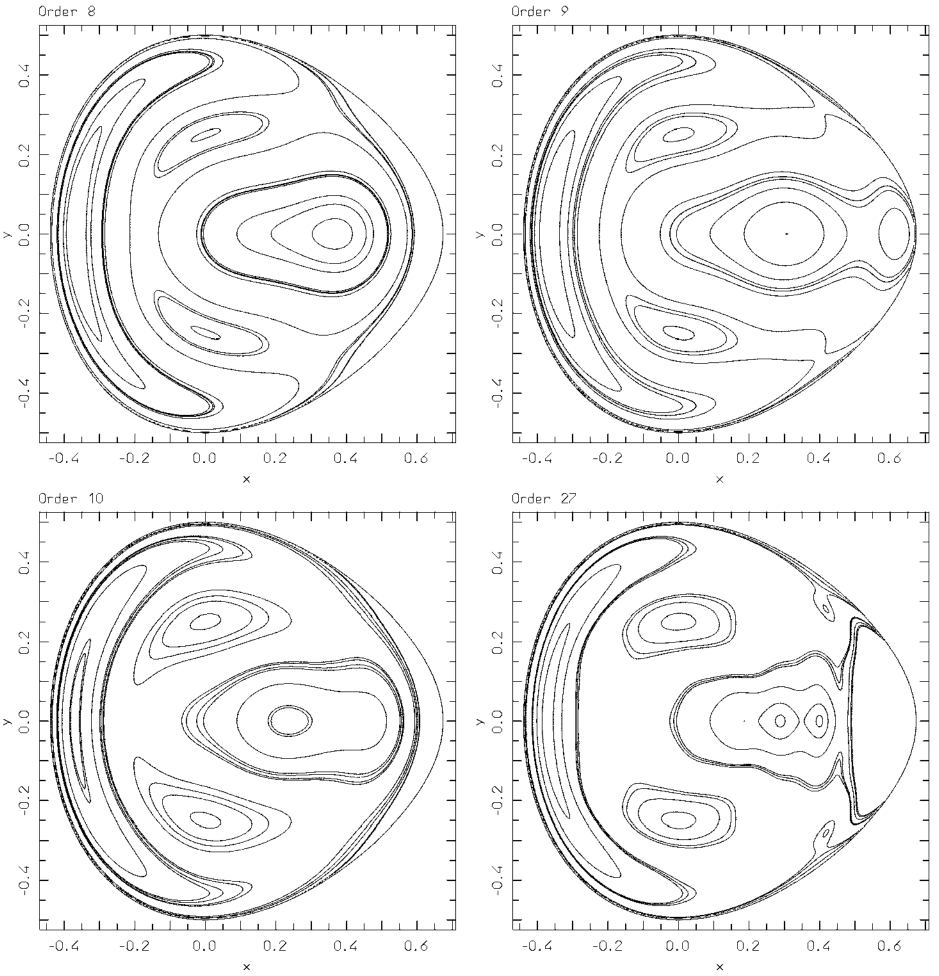

Figure 7. Level lines of the first integral truncated at orders 8, 9, 10 and 27, for energy $E = \frac{1}{8}$. Some correspondence with the Poincaré sections is found around the order 9. Then the level lines are definitely worse, making even more evident the asymptotic character of the series.

dots refer to the second and third quantities in the formula above, respectively. In all cases it is evident that the values steadily increase, with no tendency to a definite limit. The almost linear increase is consistent with the behavior $\|\Phi_s\| \sim s!$ predicted by the theory.

8.2. The Trojan asteroids

The asymptotic behavior of the series lies at the basis of Nekhoroshev theory on exponential stability. The general result, referring for simplicity to the case above, is that in a ball of radius ϱ and center at the origin one has

$$\left|\Phi(t) - \Phi(0)\right| < O(\varrho^3) \quad \text{for } |t| < O(\exp(1/\varrho^a)) \ ,$$

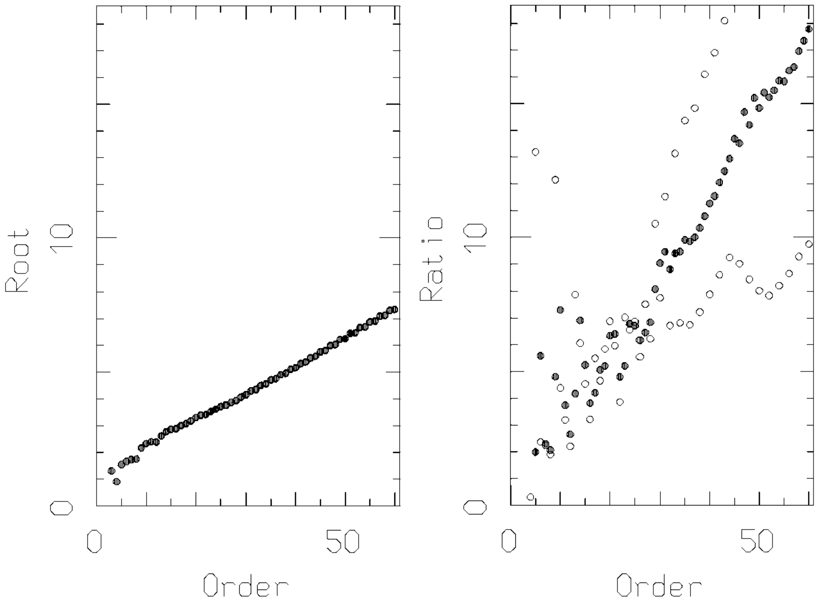

Figure 8. The convergence radius evaluated with the ratio (left) and the root (right) criterion. In both cases the non convergence of the series is evident.

for some positive $a \le 1$. This is indeed the result given by the theory (see, e.g., Giorgilli 1988). In rough terms the idea is the following. Due to the estimate $\|\Phi_s\| \sim s!$ and remarking that $\dot{\Phi} = \{H, \Phi\}$ starts with terms of degree $s+1$, one gets $|\dot{\Phi}| = O(s!\varrho^{s+1})$. Then one looks for an optimal degree s which minimizes the time derivative, i.e., $s \sim 1/\varrho$. By truncating the integrals at the optimal order one finds the exponential estimate.

However, the theoretical estimates usually give a value of ϱ which is useless in practical applications, being definitely too small. Realistic results may be obtained instead if the construction of first integrals for a given system if performed by computer algebra. That is, one constructs the expansion of the first integral up to an high order, compatibly with the computer resources available, and then looks for the optimal truncation order by numerical evaluation of the norms.

The numerical optimization has been performed for the expansion of the Hamiltonian in a neighborhood of the Lagrangian point L_4, in the framework of the planar circular restricted problem of three bodies in the Sun-Jupiter case. This has a direct application to the dynamics of the Trojan asteroids (see Giorgilli 1997).

The two first integrals which are perturbations of the harmonic actions have been constructed up to order 34 (close to the best possible with the computers at that time). The estimate of the time of stability is reported in fig. 9. The lower panel gives the optimal truncation order vs. $\log_{10} \varrho$. In the upper panel we calculate the stability time as follows: for an initial datum inside a ball of radius ϱ_0 we determine the minimal time required for the distance to increase up to $2\varrho_0$. Remark that the vertical scale is logarithmic. The units are chosen

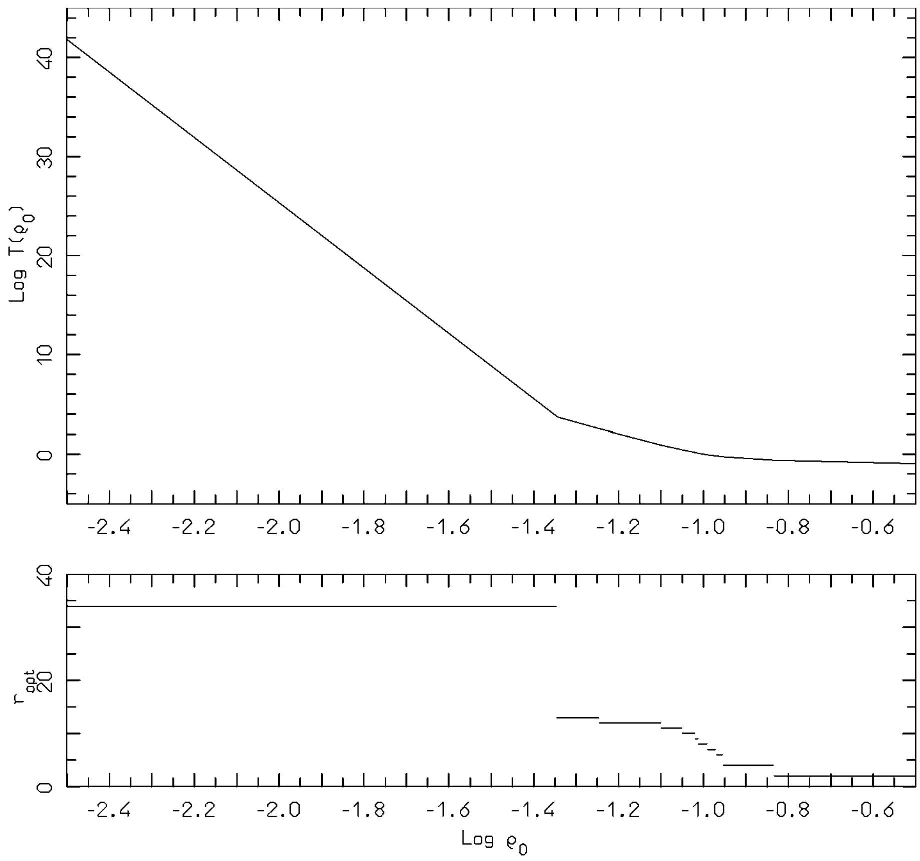

Figure 9. The estimated stability time and the optimal truncation order for the L_4 point of the Sun-Jupiter system.

so that $\varrho = 1$ is the distance of Jupiter from the Sun, and $t = 2\pi$ is the period of Jupiter. With this time unit the estimated age of the universe is about 10^9. The figure shows that the obtained data are already realistic, although, due to the unavoidable approximations, only four of the asteroids close to L_4 known at the time of that work did fall inside the region of stability for a time as long as the age of the Universe.

8.3. The SJSU system

As a third application we consider the problem of stability for the planar secular planetary model including the Sun and three planets, namely Jupiter, Saturn and Uranus. The aim is evaluate how long the semi-major axes and the eccentricities of the orbits remain close to the current value (see, Sansottera 2010).

The problem here is much more difficult than in the previous cases. The Hamiltonian must be expanded in Poincaré variables, and is expressed in action-angle variables for the fast motions and in Cartesian variables for the slow motions, for a total of 9 polynomial and 3 trigonometric variables. The expansion of the Hamiltonian in these variables clearly is a major task, that has been handled via computer algebra.

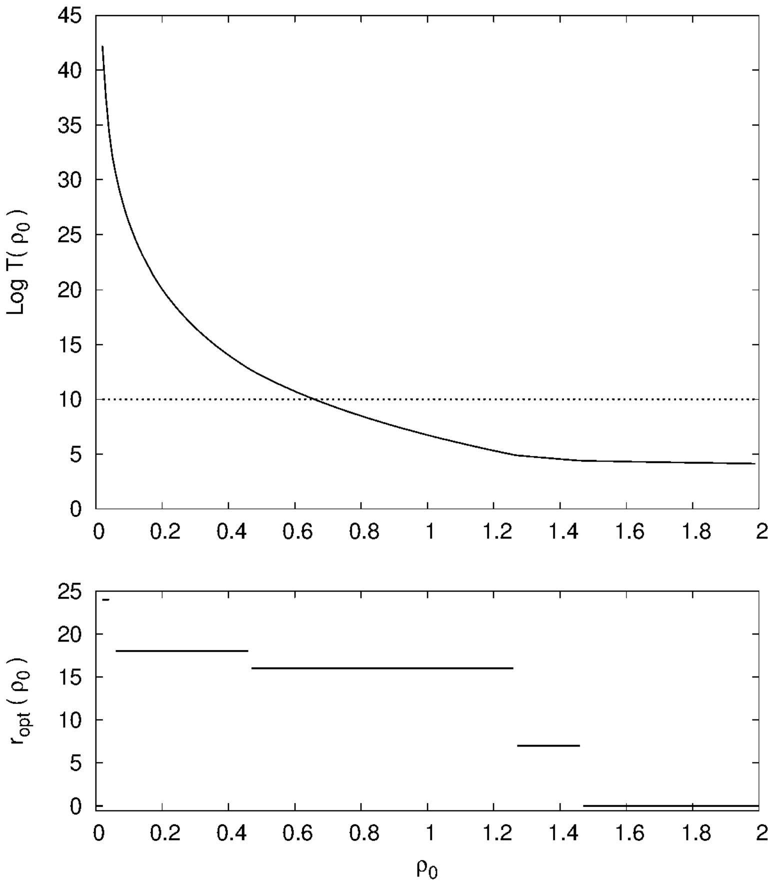

Figure 10. The estimated stability time and the optimal truncation order for the SJSU planar system. The dashed line corresponds to the estimated age of the Universe.

The reduction to the secular problem actually removes the fast motions, so that we get an equilibrium corresponding to an orbit of eccentricity zero close to a circular Keplerian one, and a Hamiltonian expanded in the neighborhood of the equilibrium, which is still represented as a system of perturbed harmonic oscillators, as in the cases above. Thus, after a long preparatory work, we find a problem similar to the previous one, that can be handled with the same methods.

The results are represented in fig. 10, where we report again the optimal truncation order and the estimated stability time, in the same sense as above. The time unit here is the year, and the distance is chosen so that $\varrho_0 = 1$ corresponds to the actual eccentricity of the three planets. The result is still realistic, although a stability for a time of the order of the age of Universe holds only inside a radius corresponding roughly to 70% of the real one.

Acknowledgments. The work of M. S. is supported by an FSR Incoming Post-doctoral Fellowship of the Académie universitaire Louvain, co-funded by the Marie Curie Actions of the European Commission.

References

Birkhoff, G. D. 1927, *Dynamical systems*, New York

Biscani, F. 2009, arXiv:0907.2076

Broucke, R. & Garthwaite, K. 1969, Cel. Mech., 1, 271

Broucke, R. 1989, Cel. Mec., 45, 255

Contopoulos, G. 1960, 49, 273

Contopoulos, G., 2002 *Order and Chaos in Dynamical Astronomy*, Springer

Contopoulos, G., Efthymiopoulos, C. & Giorgilli, A. 2003, J. Phys. A: Math. Gen., 36, 8639

Contopoulos, G., Efthymiopoulos, C. & Giorgilli, A. 2004, J. Phys. A: Math. Gen., 37, 10831

Giorgilli, A. & Galgani, L. 1978, Cel. Mech., 17, 267

Giorgilli, A., 1979, Comp. Phys. Comm., 16, 331

Giorgilli, A. 1988, Ann. Ist. H. Poincaré, 48, 423

Giorgilli, A., Delshams, A., Fontich, E., Galgani, L. & Simó, C. 1989, J. Diff. Eqs., 20

Giorgilli, A. & Skokos, Ch. 1997, Astron. Astroph., 317, 254

Gustavson, F. G., 1966, Astron. J., 71, 670

Hénon, M. and Heiles, C., 1964, Astron. J., 69, 73

Henrard, J., 1974, Cel. Mech., 10, 497

Henrard, J. 1986, : *Algebraic manipulations on computers for lunar and planetary theories*, Relativity in celestial mechanics and astrometry, 59

Henrard, J., 1989, Cel. Mech., 45, 245

Jorba, J. 1999, Experimental Mathematics, 8, 155

Knuth, D. 1968, *The Art of Computer Programming*, Addison-Wesley

Laskar, J, 1989, D. Benest & C. Froeschlé (eds.), *Les méthodes modernes de ma méchanique céleste (Goutelas, 1989)*, Ed. Frontières, 89

Poincaré, H. 1890, Acta Mathematica

Poincaré, H. 1892, *Les méthodes nouvelles de la mécanique céleste*, Gauthier–Villars, Paris

Rom, A. 1970, Cel. Mech., 1, 301

Sansottera, M., Locatelli U., & Giorgilli, A. 2010, Math. Comput. Simul., doi:10.1016/j.matcom.2010.11.018.

Siegel, C. L. 1941, Ann. Math., 42, 806

Whittaker, E. T. 1916, Proc. Roy Soc. Edinburgh, Sect. A, 37, 95