Numerical Algorithms 3 (1992) 159–172

Multivariate rational data fitting: general data structure, maximal accuracy and object orientation

Annie Cuyt [1] and Brigitte Verdonk

Department of Mathematics and Computer Science, Universiteit Antwerpen (UIA), Universiteitsplein 1, B-2610 Wilrijk – Antwerpen, Belgium

Sections 1 and 2 discuss the advantages of an object-oriented implementation combined with higher floating-point arithmetic, of the algorithms available for multivariate data fitting using rational functions. Section 1 will in particular explain what we mean by "higher arithmetic". Section 2 will concentrate on the concepts of "object orientation". In sections 3 and 4 we shall describe the generality of the data structure that can be dealt with: due to some new results virtually every data set is acceptable right now, with possible coalescence of coordinates or points. In order to solve the multivariate rational interpolation problem the data sets are fed to different algorithms depending on the structure of the interpolation points in the n-variate space.

This text is a preparatory publication for the development of a scientific expert system for multivariate rational interpolation. The issues addressed are relevant to the implementation of such a system.

1. Scientific computation

Although computers were historically developed for scientific computation, extensive interest in the subject of computer arithmetic has mostly developed in the early eighties, yielding some important improvements in the accuracy of floating-point computations. We shall briefly describe these results. We believe that they should be taken into account if one wants to develop an expert system for scientific computation.

Let us first introduce some notations. Let $\mathbb{F} \subset \mathbb{R}$ denote the set of floating-point numbers and if $*$ is one of the operations $+$, $-$, $\times$, $/$ in $\mathbb{R}$, let $\circledast$ denote the corresponding floating-point operation. From a hardware point of view, two IEEE floating-point standards exist. Among others, these standards specify that [14]:

[1] Senior Research Associate NFWO.

© J.C. Baltzer A.G. Scientific Publishing Company

(1) The four basic operations $\oplus$, $\ominus$, $\otimes$ and $\oslash$ should be implemented with exact rounding, i.e.

$$x \circledast y = \circ(x * y) \quad * \in \{+, -, \times, /\}, \; x, y \in \mathbb{F},$$

where $\circ : \mathbb{R} \to \mathbb{F}$ must be such that if $\circ(r) = f$ then there does not exist another floating-point number between r and f. In other words, exact rounding guarantees maximally accurate floating-point arithmetic. Note that several mappings $\circ$ satisfy the aforementioned property, each corresponding to a particular rounding mode.

(2) Four types of rounding modes should be provided: round to the nearest floating-point number, round toward 0 (i.e. upward rounding for negative numbers and downward rounding for positive numbers), round toward $+\infty$ (i.e. always upward rounding), round toward $-\infty$ (i.e. always downward rounding).

The advantage of the IEEE standard is that when a program is ported from one machine to another, the results of the basic operations will be the same in every bit if both machines support the IEEE standard. However, as pointed out in [19], the specifications of the standard are not sufficient to guarantee maximal accuracy for many computations. Indeed, in [19] it is noted that most computations in numerical algorithms are defined in one of the spaces $\mathbb{VF}$, $\mathbb{MF}$, $\mathbb{IF}$, $\mathbb{IVF}$, $\mathbb{IMF}$ (floating-point subset of respectively vector$\mathbb{R}$, matrix$\mathbb{R}$, interval$\mathbb{R}$, ...) or $\mathbb{F}^2$, $\mathbb{VF}^2$, $\mathbb{MF}^2$, $\mathbb{IF}^2$, $\mathbb{IVF}^2$, $\mathbb{IMF}^2$ (floating-point subset of respectively $\mathbb{C}$, vector$\mathbb{C}$, matrix$\mathbb{C}$, interval$\mathbb{C}$, ...). In traditional floating-point arithmetic, the operations in these spaces are performed in terms of the elementary floating-point operations in $\mathbb{F}$. Consider for example the floating-point scalar product $\odot$ of two vectors x and y of floating-point numbers. Using the basic operations we obtain

$$x \odot y = \bigoplus_{i=1}^{n} x_i \otimes y_i.$$

Due to the accumulation of rounding errors of each of the floating-point operations $\oplus$ and $\otimes$, it is usually the case that

$$x \odot y \neq \circ(x \cdot y),$$

where $\circ(x \cdot y)$ is the exactly rounded result of $x \cdot y$. To overcome this shortcoming a new definition of the floating-point operations in the product spaces $\mathbb{VF}$, $\mathbb{MF}$, $\mathbb{F}^2$, $\mathbb{VF}^2$, ... is given in [18]. This definition is based on the principle of exact rounding. Coming back to our example, the scalar product of two vectors in $\mathbb{VF}$ is now defined as

$$x \odot y = \circ(x \cdot y), \quad x, y \in \mathbb{VF}.$$

The operations in the other product spaces are defined in the same way [18]. It is proven [18] that to implement the operations in the spaces $\mathbb{VF}$, $\mathbb{MF}$, $\mathbb{F}^2$, $\mathbb{VF}^2$,

... according to the new definition, only the following floating-point operations must be available on the computer:
(a) $\oplus$, $\ominus$, $\oslash$, $\otimes$ and scalar product $\odot$ with exact rounding to the nearest;
(b) $\oplus$, $\ominus$, $\oslash$, $\otimes$ and scalar product $\odot$ with exact rounding toward $-\infty$;
(c) $\oplus$, $\ominus$, $\oslash$, $\otimes$ and scalar product $\odot$ with exact rounding toward $+\infty$.

Note that, compared to the IEEE standard, the set of basic operations has been extended with the scalar product.

The advantages of highly accurate floating-point arithmetic can be combined with interval arithmetic to obtain self-validating numerical methods. Interval arithmetic is the only computational tool so far available that incorporates guarantees as part of the basic computational process. Interval arithmetic which has been around for a long time has often been criticized since its naive use may deliver bounds which are unreasonably large and thus do not contain much information about the solution of the problem. It is pointed out in [19] that by using the new instead of the traditional approach to floating-point arithmetic this criticism is superseded: one combines interval computation, the process of residual correction and the scalar product with exact rounding to obtain bounds with high or even maximum accuracy. In this approach the role of the optimally accurate scalar product is crucial. Application of these techniques has yielded linear algebra routines with verified and highly accurate solutions [22]. These techniques can also be used for the evaluation of arbitrary expressions with high accuracy. To this end the expression is reformulated as a system of linear equations. As an example, if a, b, d and e are floating-point numbers then evaluation of the expression $a+b-d/e$ is reformulated as follows:

$$x_1 = a,$$
$$x_2 = x_1 + b,$$
$$x_3 = d,$$
$$ex_4 = x_3,$$
$$x_5 = x_2 - x_4.$$

To solve this system, techniques which employ the scalar product with exact rounding, together with iterative defect correction methods are used to obtain optimally accurate results. Moreover, the availability of interval arithmetic allows for automatic validation of these results. For more details we refer to [19].

Using these results, programming languages can implement support for floating-point computation with maximal accuracy in different levels of enhancement. Programming languages which provide the 15 operations in (a.-c.) are said to support basic computer arithmetic [19].

At a next, or advanced computer arithmetic level, programming languages provide constructs to deal with computations in the product spaces $\mathbb{F}^2$, V$\mathbb{F}$, M$\mathbb{F}$, ... These languages use a type concept and an operator concept as well as overloading of (certain) function names. The type concept allows easy use of the

data types $\mathbb{F}^2$, V$\mathbb{F}$, M$\mathbb{F}$, ... The operator concept allows easy use of the optimally accurate arithmetic operations on these data types. As pointed out above, these operations can all be implemented with maximal accuracy using the 15 basic operations in (a.-c.). The types and operators for the sets $\mathbb{F}^2$, V$\mathbb{F}$, M$\mathbb{F}$, ... are available in predefined and precompiled form in programming languages supporting advanced computer arithmetic.

At the last, or higher computer arithmetic level, programming languages deal in an automatic way with the capability to evaluate to maximum accuracy expressions composed of the data types and operations in the two previous levels. The language extension consists in a new programming construct to specify that a certain program part or expression must be evaluated with maximal accuracy. Coming back to our example, the statement $f := eval(a + b - d/e)$ would indicate that the value of $a + b - d/e$ must be evaluated to full accuracy. As pointed out above, this can be achieved if the implementation of the *eval*-operator is such that the expression is reformulated as a system of linear equations and the optimally accurate scalar product, defect correction and interval arithmetic are used to solve this system.

Programming languages which support basic and advanced computer arithmetic features are often referred to as SC-languages, where SC stands for Scientific Computation. The first such language to be developed was Pascal-SC [17]. Since then, Pascal-XSC [24] (Pascal eXtension for Scientific Computation) was developed. It contains a number of new features which were not considered in Pascal-SC and is implemented using a Pascal-XSC to C precompiler. Scientific Computation features have also been added to other programming languages (FORTRAN-SC [13], Modula-SC [12], C-XSC [20]) or are available as libraries (ACRITH-XSC [2], Arithmos [3]).

Programming languages which support higher computer arithmetic features are under development with several approaches being considered for the *eval*-operator [16,23]. In the current SC-languages a kind of *eval*-operator can only be achieved for a number of specific problems by combining in a program the scalar product with exact rounding, the available interval arithmetic and the technique of defect correction. However, this *eval* is not at all automatic.

The choice to implement our expert system for scientific computation in Pascal-XSC (or alternatively C-XSC of which a prototype will be available by the time you read this) rather than in another SC-language, is motivated by the fact that both have an interface with C^{++} and hence allow to combine the advantages of scientific computation with the advantages of object orientation. We shall now describe this latter aspect in more detail.

2. Object orientation

There are several principles underlying object-oriented programming. We shall mainly concentrate on three of these, namely class or abstract data type,

inheritance and overloading which is closely linked with the principle of dynamic binding. Roughly speaking, a class or abstract data type is a type together with operations (also called methods) defined on that type. An abstract data type can only be accessed through the operations defined on it. In other words, the implementation of the type (its data structure) as well as the implementation of its operations are encapsulated within the abstract data type (hidden from the user). A typical example of an abstract data type is a stack, with the operations *push*, *pop*, *isfull* and *isempty*. Whether the stack is implemented as an array or a linked list is encapsulated within the abstract data type.

Another important principle of object orientation is the principle of inheritance. Different classes can be organized in a hierarchy, where more specific classes inherit functionality from more general ones. Consider for example the class of all closed curves *ClC* with two methods *area* and *circumference* defined in it to compute the area and the circumference of the closed curve respectively. More specific subclasses of *ClC* are for example the class of rectangles *ReC* and the class of circles *CiC*. These subclasses will inherit the methods *area* and *circumference* from the class *ClC*. However, since computing the area and circumference for a rectangle and circle is much simpler than for a general closed curve, it may be more appropriate to redefine the methods *area* and *circumference* in the class *ReC* and *CiC*. If this is the case, then to compute the area of a circle or rectangle the methods within those respective subclasses will be executed, and the general method defined in the class *ClC* of closed curves will not be applied.

This brings us to the last principle of object orientation which we discuss here, namely dynamic binding. Dynamic binding is a means to resolve the overloading of method names. A method name is said to be overloaded if different classes each define a method with that name. The system can resolve the overloading by determining at run time (dynamically) the class type of the argument to which the method is applied, and hence also the correct definition (implementation) of the method. In the example given above, when computing *area*(I), the system will determine to which class C the element I belongs and select the method *area* in that class. If no such method is defined in the class C, then the principle of inheritance is invoked and the method *area* in the superclass of C is invoked.

It is mainly these three features of object-oriented programming which will be put to good use when implementing a scientific expert system. For a specific problem domain (such as multivariate rational interpolation), usually different problem types can be identified, and with each problem type appropriate algorithms can be associated. The translation of this classification in an object-oriented programming language can thus easily be done. Indeed, each problem type can be implemented as a class, where the algorithms to solve that type of problem are encapsulated within the class. As such, each class certainly defines a method to "solve". The implementation of the *solve*-method in the class C_1 is

the algorithm to solve the problem of type C_1, whereas the implementation of the *solve*-method in class C_2 will be the algorithm to solve problems of type C_2. This classification and the partial order between the different problem classes is closely related to the particular problem domain. As pointed out above, the overloading of the method name *solve* can be resolved through dynamic binding. However, for numeric computations the default dynamic binding mechanism must be carefully examined. Indeed, it may be the case that although a particular problem belongs to class C_1 the *solve*-method in the superclass of C_1, instead of in C_1 itself, should be applied for reasons of stability and reliability. In traditional object-oriented programming languages all local methods have a priority higher than that of methods inherited from superclasses. It may be appropriate to overrule this default mechanism. The overloading of methods should be conditional on the performance of the algorithms for the particular data. The principles of object-orientation also allow to improve the programming methodology. Indeed, if more optimal algorithms or data structures can be developed for a particular class, only the implementation within that class needs to be modified, while the rest of the program remains unchanged.

There are several object-oriented languages on the market today. These are, among others, C++, Smalltalk, and Eiffel. Although they do not all support object-orientation to the same degree, they all support the basic features of object-orientation described above. For the implementation of our system, we have chosen the language C++ because of its widespread use, but also for the important reason that it has an interface with Pascal-XSC and C-XSC, as mentioned in the previous paragraph.

3. The problem domain

A prototype of a scientific expert system based on the principles described above, will be implemented at the University of Antwerp (UIA) in C++ combined with Pascal-XSC or C-XSC, for the problem domain of multivariate rational Hermite interpolation, including Padé approximation. In this section, we shall briefly discuss how the general advantages of higher floating-point arithmetic and object-oriented programming are in particular applicable to our problem domain.

It is well-known that the problem of multivariate rational Hermite interpolation is in its most general way formulated as follows [8]. We restrict our presentation to the case of two variables. Given data

$$\frac{\partial^{k+l} f}{\partial x^k \, \partial y^l}(x_i, y_j),$$

where $k > 0$ or $l > 0$ indicate coalescence of coordinates or points, find polynomials

$$p(x, y) = \sum_{(i,j)\in N\subset\mathbb{N}^2} a_{ij}x^i y^j,$$

$$q(x, y) = \sum_{(i,j)\in D\subset\mathbb{N}^2} b_{ij}x^i y^j,$$

satisfying

$$\frac{\partial^{k+l}(fq-p)}{\partial x^k\,\partial y^l}(x_i, y_j) = 0, \tag{1}$$

where the total number of unknown coefficients in p and q equals the number of interpolation conditions plus a normalization condition. If $k > 0$ or $l > 0$ then each additional data point besides the value of f can be represented by another $(x_{i(k)}, y_{j(l)})$ where we let $x_{i(k)} \to x_i$ if $k > 0$ and $y_{j(l)} \to y_j$ if $l > 0$. For $(k, l) = (0, 0)$ we have of course $i(0) = i$ and $j(0) = j$. In the univariate case the rational Hermite interpolation problem is equivalent to the Newton–Padé approximation problem. In the multivariate case this is only true for particular data sets. Consequently the algorithms for rational Hermite interpolation and Newton–Padé approximation are in general not equivalent. Let us call the set of indices $(i(k), j(l))$ for all given i,j and k,l the data index set I. Clearly $\#I$ equals the number of interpolation conditions.

By total inclusion property of the data index set I we mean that when (i, j) belongs to I then all (i', j') with $0 \le i' \le i$ and $0 \le j' \le j$ also belong to I (this is the whole rectangle of indices emanating from the origin and with the given index (i, j) as its furthermost corner). Sometimes this can be achieved by a simple renumbering of the original data [9].

By partial inclusion property of the data index set I we mean that the amount of data given at each distinct point (x_i, y_j) is indexed by a set

$$I_{(i,j)} = \left\{(k, l)\middle|\, \frac{\partial^{k+l} f}{\partial x^k\,\partial y^l}(x_i, y_j) \text{ is given}\right\}, \tag{2}$$

which satisfies the total inclusion property and this for each distinct (x_i, y_j).

In fact the partial inclusion property is always true because one never jumps (partial) derivatives when letting interpolatory data coincide. Just look at the univariate case to see the logic. If besides the function value also higher derivatives are specified at an interpolation point, then these derivatives are of consecutive order.

Only for data sets satisfying the total inclusion property the rational Hermite interpolation problem (1) can be reformulated as a Newton–Padé approximation problem. Given the (x_i, y_j), construct basis functions

$$B_{ij}(x, y) = \prod_{k=0}^{i-1}(x - x_k)\prod_{l=0}^{j-1}(y - y_l)$$

and find polynomials

$$p(x, y) = \sum_{(i,j)\in N\subset\mathbb{N}^2} a_{ij}B_{ij}(x, y),$$

$$q(x, y) = \sum_{(i,j)\in D\subset\mathbb{N}^2} b_{ij}B_{ij}(x, y),$$

satisfying

$$(fq - p)(x, y) = \sum_{(i,j)\in\mathbb{N}^2\setminus I} d_{ij}B_{ij}(x, y), \tag{3}$$

with $N \subset I$ and $\#I = \#N + \#D - 1$. Examples of the different situations are given in the next section with a complete set of references, also to the most recently developed techniques.

Remark that instead of fitting p/q to f we consider the modified problem of fitting $fq - p$ to zero. A solution of the modified problem is also a solution of the true problem except when both p and q have interpolation points as common zeros. It is well-known that conditions (1) or (3) result in defining systems of linear equations. Linear systems of equations occur in many other aspects of multivariate rational interpolation, when using Sylvester's identity in recursive schemes [5,6] or for the evaluation of continued fractions [11,21]. This is an important advantage for our implementation, as we recall briefly.

It has been mentioned in section 1 on scientific computation that in higher computer arithmetic, optimally accurate results can be obtained for different floating-point expressions by reformulating them in terms of systems of linear equations. These linear systems then have to be solved using defect correction methods and automatic validation as described in [19]. We therefore expect that especially the higher computer arithmetic tools of Pascal-XSC or C-XSC will prove to be very effective and accurate when implementing algorithms for multivariate rational interpolation.

Concerning the object-oriented aspects, the classification of the problem domain is mainly based on the structure of the data. Let us now focus on this classification. As already pointed out, the most general situation to be considered is the one with N and D general and I satisfying the partial inclusion property. If all data are given at distinct points then the problem is a pure rational interpolation problem. If data with coalescent coordinates occur then a combination with algorithms of the Newton–Padé case is necessary to compute intermediate results. For the use of certain recursive algorithms it is necessary that one chooses the numerator and denominator index sets N and D to satisfy the total inclusion property.

If I also satisfies the total inclusion property and $N \subset I$, then the problem can be solved by some Newton–Padé algorithms. If all data are coalescent the problem is in fact merely a Padé approximation problem. If all data are given at distinct points, then all divided differences in the formal Newton series have to be calculated, else some are replaced by given partial derivatives.

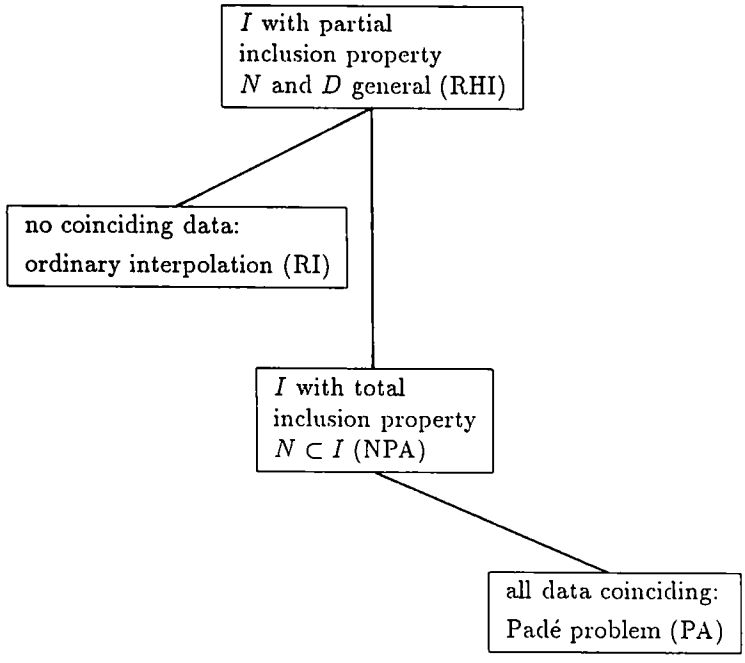

Fig. 1.

With each of the classes, different algorithms can be associated [8]. The *solve*-method will implement the algorithms for each particular class. Since in many cases also several algorithms exist for a particular class (recursive algorithms, continued fraction representation of the rational function), sufficient theoretical expertise must be included in the system so that the appropriate algorithm is selected within the class. Criteria to be taken into account are reliability (true or modified rational interpolant [4]) and stability (reasonable error bounds [15]), as well as choosing the appropriate degree in numerator and denominator with respect to consistency (data from a rational function itself [1]) and convergence (data from a meromorphic or Stieltjes function [7]). This will make the scientific expert system a "true" expert system.

4. Classification details

Let us work our way up by first presenting an interpolation problem that can be reformulated as a Newton–Padé approximation problem. Consider the data set from fig. 2: for all indicated pairs of indices (i, j) the function value $f(x_i, y_j)$ is given, at (x_0, y_1) also the partial derivatives $\partial f/\partial x$, $\partial f/\partial y$ and $\partial^2 f/\partial y^2$ are known and at (x_1, y_1) the partial derivative $\partial f/\partial x$ is specified. If we proceed as

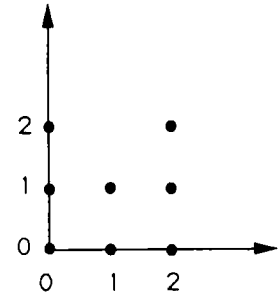

Fig. 2.

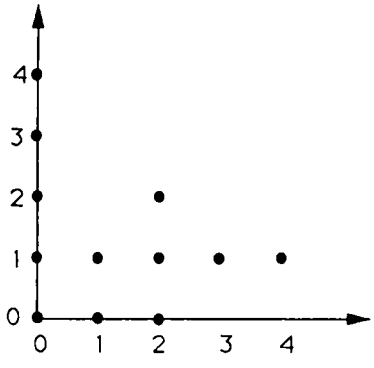

Fig. 3.

outlined in the previous section, we represent all these data as indicated in fig. 3 where we let $y_3 \to y_1$, $y_4 \to y_1$, $x_3 \to x_0$ and $x_4 \to x_1$.

By renumbering we can represent this data set by the index set I drawn in fig. 4 and satisfying the total inclusion property. Hence we can compute all bivariate divided differences occurring in the Newton interpolating series

$$\sum_{(i,j)\in I} f[x_0,\ldots,x_i][y_0,\ldots,y_j]B_{ij}(x, y) + \ldots$$

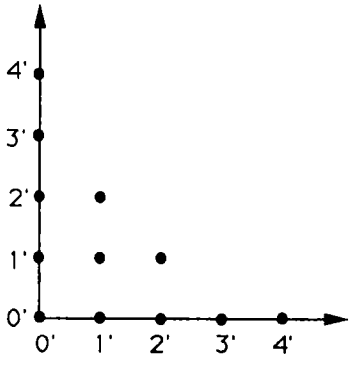

Fig. 4.

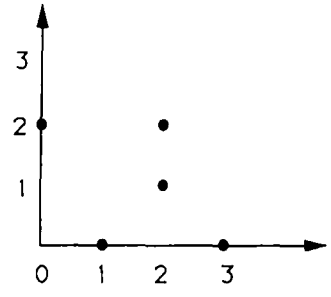

Fig. 5.

and approximate this series in a Padé-like fashion, satisfying some accuracy-through-order conditions. Choose for instance

$$N = \{(i, j) \mid 0 \le i + j \le 3\} \subset I,$$

$$D = \{(i, j) \mid 0 \le i + j \le 1\},$$

and write down the linear systems of eqs. (3) for the unknown coefficients in p and q indexed by N and D respectively.

Next we consider an interpolation problem that cannot be reformulated as an approximation problem. Consequently only the methods from the RHI-class are applicable. For the previous problem we used the NPA-formulation though the RHI-approach remained valid. We already remarked that in the multivariate case the two formulations are in general not equivalent anymore. Take the data set from fig. 5: for all (i, j) the function value $f(x_i, y_j)$ is given while at (x_3, y_0) also $\partial f/\partial y$ is known. This data set is the limit situation of the one given in fig. 6 where we let $y_3 \to y_0$.

No renumbering of the data index set resulting in total inclusion is possible. Hence we can choose N and D satisfying the total inclusion property, while I is general,

$$N = \{(0, 0), (1, 0), (0, 1), (1, 1)\},$$

$$D = \{(0, 0), (1, 0), (0, 1)\},$$

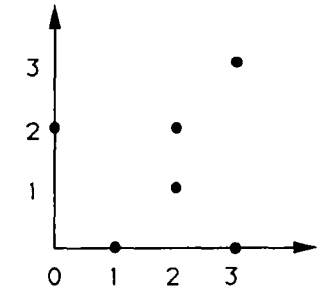

Fig. 6.

and solve the problem using methods described in [10]. The linear system of eqs. (1) for p and q indexed by N and D is

$$(fq-p)(x_i, y_j)=0, \quad (i, j)\in I\setminus\{(3, 3)\},$$

$$\frac{\partial(fq-p)}{\partial y}(x_3, y_0)=0.$$

Up to now the definition of the multivariate rational interpolant has not been constructive. For both approaches a recursive computation scheme exists. To this end we need an enumeration of the index points in $\mathbb{N}^2$ that preserves the inclusion property when taking initial subsets of an index set I that has the total inclusion property (several choices for such an enumeration are possible).

For the Newton–Padé approximant this then results in writing I as

$$I=\cup_{k=0}^{n+m} I_k, \quad \#I_k=k+1;$$

$$I_k\setminus I_{k-1}=\{(i_k, j_k)\}, \quad I_{-1}=\emptyset,$$

where each I_k satisfies the total inclusion property. Since $N\subset I$ we define $N_k=I_k$ for $k=0,\ldots,n$. We can also enumerate the points in D in the order of the enumeration of $\mathbb{N}^2$ to obtain $D_k=\{(d_0, e_0),\ldots,(d_k, e_k)\}$ for $k=0,\ldots,m$. Now that we have fixed an order in which the terms are added in p and q and the data are treated, we can interpret a recursive computation scheme in terms of "previous" and "next" interpolants. For our data set depicted in fig. 4, the Newton–Padé approximant will be denoted by

$$[N/D]_I=[N_9/D_2]_{I_{11}}=E_2^{(9)}.$$

This solution can be computed by a generalization of the ϵ-algorithm [6] where in general $E_m^{(n)}$ is specified in terms of $E_{m-1}^{(n)}$ and $E_{m-1}^{(n+1)}$. Starting values for the recursive scheme are the

$$E_0^{(k)}=\sum_{(i,j)\in I_k} f[x_{d_0},\ldots,x_i][y_{e_0},\ldots,y_j]B_{d_0 i, e_0 j}(x, y), \quad k=0,\ldots,n+m,$$

where

$$B_{ki,lj}(x, y)=\frac{B_{ij}(x, y)}{B_{kl}(x, y)}.$$

For the rational Hermite interpolation problem with I only satisfying the partial inclusion property, we enumerate

$$I=\{(k_0, l_0),\ldots,(k_{n+m}, l_{n+m})\},$$

$$N=\{(i_0, j_0),\ldots,(i_n, j_n)\},$$

$$D=\{(d_0, e_0),\ldots,(d_m, e_m)\}.$$

The same computation scheme as above, but with different starting values and a different interpretation, can yield the solution [8,10] which we denote for

the data set of fig. 6 by

$$[N/D]_I = [N_3/D_2]_{I_5} = E_5^{(0)}.$$

The superscript in the E-value indicates the starting interpolation point, in this case indexed (k_0, l_0), and the subscript gives the total number of interpolation conditions imposed from there on. The fact that $E_m^{(n)}$ can be calculated from $E_{m-1}^{(n)}$ and $E_{m-1}^{(n+1)}$ now means that we use a Bulirsch–Stoer type computation where $E_5^{(0)}$ is obtained from dividing the set of data into two subsets: in $E_4^{(0)}$ we discard the last interpolation point and in $E_4^{(1)}$ we omit the first one. Starting values are now $E_0^{(h)} = f(x_{k_h}, y_{l_h})$ which indeed interpolate only at the hth point. Intermediate values where coalescent data points occur, such as $E_1^{(4)}$ in the example above, have to be computed using the NPA-approach. This is possible since I satisfies the partial inclusion property, meaning that at every distinct data point (x_k, y_l) the set $I_{(k,l)}$ defined by (2) satisfies the total inclusion property. Remark that if only noncoalescent data occur all $I_{(k,l)} = \{(0, 0)\}$.

In a straightforward object-oriented approach, multivariate rational data fitting problems of class NPA will automatically be solved by NPA-methods because the methods of the superclass RHI are superseded. However, for a scientific computation problem one first needs certainty about the fact that the NPA-*solve* is also numerically "preferable" over the RHI-*solve*. This still has to be investigated, along with the other open problems mentioned in section 3 to make the system a true expert system, namely reliability, stability, consistency and convergence of the multivariate rational interpolant.

References

[1] J. Abouir and A. Cuyt, Multivariate partial Newton–Padé and Newton–Padé type approximants, J. Approx. Th. (to appear).

[2] Acrith-XSC: IBM High Accuracy Arithmetic – Extended Scientific Computation. Version 1, Release 1, IBM Deutschland GmbH, Department 3282, Böblingen.

[3] ARITHMOS (BS2000): Benutzerhandbuch, SIEMENS AG, Bereich Datentechnik.

[4] H. Arndt, Ein verallgemeinter Kettenbruch-Algorithmus zur rationalen Hermite-Interpolation, Numer. Math. 36 (1980) 99–107.

[5] C. Brezinski, A general extrapolation algorithm, Numer. Math. 35 (1980) 175–187.

[6] A. Cuyt, A recursive computation scheme for multivariate rational interpolants, SIAM J. Num. Anal. 24 (1987) 228–238.

[7] A. Cuyt, Extension of "A multivariate convergence theorem of the de Montessus de Ballore type" to multipoles, J. Comp. Appl. Math. 41 (1992) 323–330.

[8] A. Cuyt, Multivariate rational interpolation: old and new results, in preparation.

[9] A. Cuyt and B. Verdonk, General order Newton–Padé approximants for multivariate functions, Numer. Math. 43 (1984) 293–307.

[10] A. Cuyt and B. Verdonk, Different techniques for the construction of multivariate rational interpolants, in: *Nonlinear Numerical Methods and Rational Approximation*, ed. A. Cuyt (1988) pp. 167–190.

[11] A. Cuyt and B. Verdonk, Evaluation of branched continued fractions using block-tridiagonal linear systems, IMA J. Numer. Anal. 8 (1988) 209–217.

[12] C. Falcó Korn, S. Gutzwiller, S. König and Ch. Ullrich, Modula-SC. Motivation, language definition and implementation, IMACS Ann. Comput. Appl. Math. 12 (1992) 161–179.
[13] FORTRAN for scientific computation. Language description and sample programs, Institute for Applied Mathematics, University of Karlsruhe.
[14] D. Goldberg, What every computer scientist should know about floating-point arithmetic, ACM Comp. Surv. 23 (1991) 5–48.
[15] P. Graves-Morris, Practical, reliable, rational interpolation, J. Inst. Math. Appl. 25 (1980) 267–286.
[16] W. Krämer, Highly accurate evaluation of program parts with applications, IMACS Ann. Comput. Appl. Math. 7 (1990) 397–409.
[17] U. Kulisch (ed.), *Pascal-SC: Information Manual and Floppy Disks* (Wiley–Teubner, Stuttgart, 1987).
[18] U. Kulisch and W. Miranker, *Computer Arithmetic in Theory and Practice* (Academic Press, New York, 1981).
[19] U. Kulisch and W. Miranker, The arithmetic of the digital computer: a new approach, SIAM Rev. 28 (1986) 1–36.
[20] C. Lawo, C-XSC, A programming environment for eXtended Scientific Computation, *Proc. 13'h IMACS World Congress*, vol. 1 (1991) p. 34.
[21] J. Miklosko, Investigation of algorithms for numerical computation of continued fractions, USSR Comp. Math. Math. Phys. 16 (1976) 1–12.
[22] Ch. Ullrich and J. Wolff von Gudenberg (eds.), *Accurate Numerical Algorithms: A Collection of Research Papers* (Springer, Berlin, 1989).
[23] J. Wolff von Gudenberg, Object-oriented concepts for scientific computation, IMACS Ann. Comput. Appl. Math. 12 (1992) 181–192.
[24] R. Klatte, V. Kulisch, M. Neaga, D. Ratz and C. Ullrich (eds.), *Pascal-XSC. Language Reference with Examples* (Springer, Berlin, 1992).