# Exact Real Computation in Computer Algebra

Gábor Bodnár, Barbara Kaltenbacher, Petru Pau, Josef Schicho

SFB Numerical and Symbolic Computation
Johannes Kepler University, A-4040 Linz, Austria

**Abstract.** Exact real computation allows many of the advantages of numerical computation (e.g. high performance) to be accessed also in symbolic computation, providing validated results. In this paper we present our approach to build a transparent and easy to use connection between the two worlds, using this paradigm. The main discussed topics are: representation of exact real objects, operations on exact real matrices, polynomial greatest common divisor and root computation. Some of these problems are ill-posed; we use regularization methods to solve them.

## 1 Introduction

Many algorithms in computer algebra rely on exactness. The usual way to perform exact computation is to work with the rationals or with algebraic numbers. The latter are traditionally represented by the minimal polynomial and either an isolating interval or a sequence of signs in the real case (see [10]). Sometimes it can be a problem that these subfields are not closed under transcendental functions. Moreover, computations with algebraic numbers tend to be very expensive (see [17]).

The method of exact real computation seems to be a reasonable alternative: it provides mathematically consistent representation of elements of metric spaces. That is, the objects behave exactly like the represented mathematical entities. Recall that this does not hold, for instance, in the case of floating point numbers, which is one of the reasons why it is often so difficult to work with this representation in computer algebra.

Several ways to represent real numbers have already been invented; we can mention here infinite (lazy) strings of digits ([9], with golden ratio notation, [13], [8]) and functional representation, where the reals are given as functions that produce approximations on demand (see [5]). Those representations realize only very basic algorithms. On the other hand, there has been some recent research on numerical algorithms in algorithmic computer algebra, e.g. [18, 6, 16]. What lacks is an approach to exact real computation integrating up-to-date symbolic and numerical algorithms.

In this paper we present a new approach to exact real computation in a generic setup, where the represented elements come from metric spaces, and there exist numerical algorithms that can compute arbitrarily close approximates which are efficiently representable on the computer. The type specifications were designed to provide a transparent interface to the represented mathematical entities, while we endeavored to make it as easy as possible to define functions, like algebraic operations, on exact real objects. We used the paradigm of object oriented programming, obtaining a template parametrized by the class of underlying metric spaces.

In the type specifications we strictly separated the layer of the interface to the exact real objects from the layer of the interface to the numerical routines that compute approximates for the represented mathematical entity. This way the computations of approximates are separated from the computations related to error propagation; therefore an implementation of an exact real type is independent of the underlying numerical library. In this paper we focus on the representation of exact real objects together with the analysis of error propagation in a few selected computations. We do not discuss anything under the interface to the numerical algorithms.

For many problems the main task is to control error propagation. Often this information can be found in the literature of numerical analysis (see e.g. [11]). However, there are problems which lead to fundamental difficulties in numerical computation, because their solution does not depend continuously on the input data. They are called *ill-posed*, and some of them are in the center of our attention: pseudoinverse of rank-deficient matrices, polynomial greatest common divisor, multivariate polynomial factorization and curve and surface parameterization. Strictly speaking, these problems cannot be solved within the exact real paradigm (see [1]). As we definitely need a substitute for them, we apply the common technique of *regularization*, in which we replace the ill-posed problem by a nearby continuous one, and require the distance to the exact problem to be an additional input parameter. In this paper we apply Tikhonov regularization (see [2] and [7]) to the computation of pseudoinverses and GCD's.

The paper is organized as follows: In Sect. 2 we give the base type specification for exact real objects and provide some examples for the computable real numbers. Section 3 presents the analysis of the error propagation in some nontrivial operations on real matrices. In Sect. 4, we introduce polynomials with exact real coefficients with the most important operations.

We want to express our gratitude to R. Corless, S. Ratschan and H.-J. Stetter for inspiring remarks. This work has been supported by the Austrian science fund (FWF) in the frame of the special research area "numerical and symbolic scientific computing" (SFB013, projects F1303, F1308).

## 2 Type Specification for Exact Real Objects

Our final goal is to make powerful numerical methods readily available in symbolic computation, that is, to establish reliable and transparent connection between numeric and symbolic computation using the exact real paradigm.

In this paper we stay on higher abstraction levels treating the actual implementation of the underlying numerical algorithms as black boxes with which we communicate via standard interfaces. It is not our intention to look behind these interfaces and to discuss efficiency, stability, etc., of the possible implementations. However, as it will be shown in Sect. 3 and Sect. 4, as soon as we have numerical algorithms whose specifications fulfill the requirements of the interface, the reliability of the computations can be ensured by controlled error propagation, which can be sustained even in some ill-posed problems.

On the other hand, we cannot escape from choosing an appropriate representation for the approximates, which are produced on user request to provide information on the mathematical object in question. The approximating elements must be finitely representable (e.g. matrices of multi-precision floating point numbers), and they, together with the objects which they approximate, have to come from some metric space, so that the notion of distance or accuracy makes sense. This is perfectly conform with the theory of interval computation (see e.g. [14]).

We use the following notation: an abstract mathematical object is denoted by a standard italic character (e.g. $x \in \mathbf{R}$, say $x := \sqrt{2}$), a numerical computer representations by a typewriter character (e.g. $\mathtt{x} := 1.41$), which can be naturally considered as an element of the space of $x$ (i.e. $1.41 \in \mathbf{R}$), and an exact real object for $x$ is denoted by a calligraphic character $\mathcal{X}$.

**Definition 1.** *Let us fix some metric space $M$. We call a pair $(\mathtt{x}, \mathtt{e}) \in M \times \mathbf{R}_{\geq 0}$ an* approximate *for $x \in M$, if $|x - \mathtt{x}| \leq \mathtt{e}$. From a given list of approximates for $x$, $[(\mathtt{x}_1, \mathtt{e}_1), \ldots, (\mathtt{x}_n, \mathtt{e}_n)]$, $(\mathtt{x}_i, \mathtt{e}_i)$ the* most accurate *one if $\mathtt{e}_i \leq \mathtt{e}_j$ for all $j = 1, \ldots, n$ and $i$ is the minimal index with this property.*

At a given step of a computation, we can consider the list $L_x$ of approximates for an object $x$ that were computed so far, using temporal order. The most accurate approximate for $x$ at the given point of the computation is then the most accurate approximate for $x$ with respect to $L_x$.

### 2.1 Base Type of Exact Real Objects

The other main goal is to achieve transparency. For this, we have to provide a suitable abstract type specification with a fixed communication interface, basic and higher level constructors.

**Definition 2.** *An exact real object $\mathcal{X}$ that represents an element $x$ from some metric space $M$ with metric $d$, is defined by:*
*Data members:*

- *Name:* ***knownApprox****. At any point of the lifetime of the object it stores the most accurate approximate, of the form $(\mathtt{x}, \mathtt{e})$, $\mathtt{e} \in \mathbf{R}_{\geq 0}$, which was computed for $x$ till then.*

*Methods:*

- *Name:* ***epsint****. Input: a positive real value $\mathtt{e}$. Output: an approximating element $\mathtt{x}''$, such that $d(\mathtt{x}'', x) \leq \mathtt{e}$. Side effect: if the accuracy of $\mathtt{x}''$ is $\mathtt{e}''(\leq \mathtt{e})$ and $\mathtt{e}'' < \mathtt{e}'$, where $(\mathtt{x}', \mathtt{e}')$ is the* ***knownApprox*** *member of $\mathcal{X}$ at the time point of the procedure call, the* ***knownApprox*** *member of $\mathcal{X}$ becomes $(\mathtt{x}'', \mathtt{e}'')$.*

– *Name:* ***refine****. Input: none. Output: an approximate* $(\mathtt{x}, \mathtt{e})$ *of* $x$*, such that* $\mathtt{e} < \mathtt{e}'/2$ *if* $\mathtt{e}' > 0$*, where* $(\mathtt{x}', \mathtt{e}')$ *is the* ***knownApprox*** *member of* $\mathcal{X}$ *at the time point of the procedure call, or* $\mathtt{e} = \mathtt{e}'$ *if* $\mathtt{e}' = 0$*. Side effect: the* ***knownApprox*** *member of* $\mathcal{X}$ *becomes* $(\mathtt{x}, \mathtt{e})$*.*
– *Constructor. Input: an initial approximate* $(\mathtt{x}, \mathtt{e})$ *for* $x$ *and, if* $\mathtt{e} > 0$*, at least one procedure which matches one of the following specifications:*
  - *Input: a positive real value* $\mathtt{e}$*. Output: an approximate* $(\mathtt{x}', \mathtt{e}')$ *for* $x$*, such that* $\mathtt{e}' \leq \mathtt{e}$*. (Referred as:* ***approx****)*
  - *Input: an approximate* $(\mathtt{x}, \mathtt{e})$ *of* $x$*. Output: an approximate* $(\mathtt{x}', \mathtt{e}')$ *of* $x$*, such that* $\mathtt{e}' < \mathtt{e}$ *if* $\mathtt{e} > 0$*, or* $\mathtt{e}' = \mathtt{e}$ *if* $\mathtt{e} = 0$*. (Referred as:* ***iterate****)*

*Remark 1.* (i) The definition provides a template, with parameter $M$ ($d$ is given by $M$, but we do not denote it, e.g. by $d_M$, to simplify notation). (ii) In the specification of the iterative procedures we prescribe a lower bound on the improvement of the accuracy of approximation in order to ensure the termination of algorithms that need to produce approximating elements with given accuracies. (iii) When `epsint` is called with $\mathtt{e} \geq \mathtt{e}'$, $\mathtt{x}'$ is returned without any computation. (iv) The input initial approximate for the constructor has to be provided by the user. This is a trivial task if an `approx` procedure is available, however, it provides indispensable information when only an `iterate` procedure is given.

### 2.2 An Example for Real Numbers

In order to demonstrate how the above framework can be filled, we present a down-to-earth example in detail. We write the procedures in the programming language of the computer algebra system Maple, which also has a powerful numeric subsystem to exploit. This language does not support object oriented programming natively, thus class definitions and construction of objects has to be carried out implicitly. In particular, the methods need to be pointed explicitly to the object in context, which will be their first argument by default.

The simplest mathematical entities that offer themselves naturally to be represented in the exact real paradigm are the computable real numbers. The information defining the class is collected in the table `exactReal`, named after the class.

The constructor of the class just instantiates an object with the provided data members, from which `approx` and `iterate` are actually (references to) procedures. The `epsint` and `refine` methods provide the intended behavior relying on the `approx` and/or `iterate` members of the object in context.

```
exactReal := table([
"constructor" = proc(ka::list(float),ap,it)
 if ka[2]<0 then ERROR("Invalid approximate!"); fi;
 if ka[2]>0 and not (type(ap,procedure) or type(it,procedure)) then
  ERROR("At least one procedure must be given!");fi;
 RETURN(table(["approx"=ap, "iterate"=it, "knownApprox"=ka])); end,
"epsint" = proc(x::table, e::nonneg) local ap,ae;
 ap:=x["knownApprox"]; if ap[2]=0 then RETURN(ap[1]); fi; ae:=e;
```

```
 while ap[2]>e do
  if type(x["approx"],procedure) then ap:=x["approx"](ae);ae:=ae/10;
  else ap:=x["iterate"](ap); fi; od;
 x["knownApprox"]:=ap; RETURN(ap[1]); end,
"refine" = proc(x::table) local ap,e,ae;
 ap:=x["knownApprox"];
 if ap[2]=0 then RETURN(ap[1]); fi; e:=ap[2]/10; ae:=e;
 while ap[2]>e do
  if type(x["iterate"],procedure) then ap:=x["iterate"](ap);
  else ap:=x["approx"](ae); ae:=ae/10; fi;od;
  x["knownApprox"]:=ap; RETURN(ap[1]); end]);
```

Now we can define, for instance, an exact real object for $\pi$, using the routine of Maple for numerical expansion, and we can use the queries of the class on it:

```
knownApprox := [3.14, 0.1];
approx := proc(eps::nonneg) [evalf(Pi,max(2,2-ilog10(eps))),eps] end;
exPi := exactReal["constructor"](knownApprox, approx, 0);
exactReal["epsint"](exPi, 1e-10);
                              3.14159265359
```

### 2.3 Higher Level Constructors

Higher level constructors are basically the procedures that implement functions on exact real objects. Let $f : M \to M'$ be a continuous function, $M, M'$ being metric spaces; in particular let $d'$ be the metric on $M'$. We assume that the numerical algorithm $\mathtt{F}$, which implements $f$, for any input $\mathtt{x} \in M$, returns a pair $(\mathtt{x}', \mathtt{e}') \in M' \times \mathbf{R}_{\geq 0}$, such that $d'(\mathtt{x}', f(\mathtt{x})) \leq \mathtt{e}'$.

Then a higher level constructor $\mathcal{F}$ that implements $f$, takes an exact real object $\mathcal{X}$ as argument, which represents a fixed argument $x$ of $f$, and builds an exact real object $\mathcal{X}'$ for $x' := f(x)$, using the basic constructor for the exact real type for $M'$. Note that $\mathcal{F}$ has to generate at least one procedure and an initial approximate (according to definition 2), and that in both of these tasks it has to rely on $\mathtt{F}$, the underlying numerical algorithm.

The most important task in the implementation of $\mathcal{F}$ is to reliably encode error propagation. The inaccuracy of the approximates of $x'$ originates not only in that $\mathtt{F}$ generates numerical error, but also in the fact that the input of $\mathtt{F}$ (i.e. an approximating element $\mathtt{x}$, obtained from $\mathcal{X}$ as `knownApprox` or by `epsint` or `refine`) is itself inaccurate.

The *forward error computation* determines for a given approximate $(\mathtt{x}, \mathtt{e})$ of a fixed element $x \in M$, an upper bound for $d'(f(x), f(\mathtt{x}))$. It is performed by $\mathcal{F}$ when it creates an initial approximate for $\mathcal{X}'$, and also by the `iterate` procedure of $\mathcal{X}'$, which is generated by $\mathcal{F}$.

The *backward error computation*, for fixed $x \in M$ and given positive real value $\mathtt{e}'$, determines $\mathtt{e}$ such that for all approximates $(\mathtt{x}, \mathtt{e})$ of $x$, $d'(f(x), f(\mathtt{x})) \leq \mathtt{e}'$. It is performed in the `approx` procedure of $\mathcal{X}'$, which is generated by $\mathcal{F}$.

More advanced examples of forward and backward error computation are presented in the consecutive sections. Here we give only a simple higher level constructor; the addition of real numbers as exact real objects. We insert `addReals` as a new method into the class definition of `exactReal`.

In this constructor we provide only one procedure with the input and output specification of `epsint`. The backward error computation is very simple: in case of exact addition we can just take approximating elements for the operands with `e`/2 accuracy. However, in general we also have to count with rounding error, thus, by default, we distribute the allowed error equally between the inaccuracies of the approximating elements and the rounding error of the addition.

The forward error is just the sum of inaccuracies of the approximates which we added to produce the initial approximation. This is the straightforward application of interval arithmetic (see [14]). Again, the rounding errors of the computations increase the inaccuracy of the initial approximating element of the sum.

```
"addReals" = proc(x1::table,x2::table)
local ka,approx,dx1,dx2;
 approx := proc(eps::nonneg) local ap,e,en;
  e:=divRD(eps,3); en:=e;
  ap:=numAdd(exactReal["epsint"](x1,e),exactReal["epsint"](x2,e));
  # if numAdd generates too much error we distribute eps unequally
  while ap[2]>en do
   e:=divRD(e,2); en:=subRD(subRD(eps,e),e);
   ap:=numAdd(exactReal["epsint"](x1,e),exactReal["epsint"](x2,e));
  od; RETURN([ap[1],eps]); end;
 dx1:=x1["knownApprox"]; dx2:=x2["knownApprox"];
 ka:=numAdd(dx1[1],dx2[1]);
 RETURN(exactReal["constructor"]([ka[1],addRU(ka[2],dx1[2],dx2[2])],
  approx,0)); end
```

Here `divRD` and `subRD` perform numerical division and subtraction respectively, introducing only rounding down errors, and `addRU` is numerical addition with only rounding up error. A simple procedure for numeric addition, which works well as long as the mantissas do not exceed the maximal allowed lengths in Maple, can be defined as:

```
numAdd := proc(x1::float,x2::float)
 RETURN([evalf(x1+x2, 1+max(length(op(1,x1))+op(2,x1),
  length(op(1,x2))+op(2,x2))-min(op(2,x1),op(2,x2))),0]); end;
```

As an additional remark to this section, let us note that often it is natural to generate iterative approximation procedures in the higher level constructors, applying only forward error computation, which can rely on firm interval analysis techniques. However, in critical situations one cannot always ensure sufficiently large improvement of the accuracy of the approximation in an iteration step because the intervals may have the tendency of blowing up. To prevent looping, one can encode heuristics in the `epsint` and `refine` procedures, that change from `iterate` to `approx` (if it is available) when a critical situation is predicted.

# 3 Real Linear Algebra

In this section we present examples of the essential part of defining new exact real types, i.e. the formulas of forward and backward error computation for nontrivial operations on multidimensional exact real objects. We describe matrix inversion and the computation of the Moore-Penrose generalized matrix inverse (pseudoinverse) for matrices over the computable real numbers.

The straightforward way of describing multidimensional entities of real numbers would be to build them up from elementary exact real objects. However, in this case all the standard methods of linear algebra should be re-implemented to work with them, and the resulting exact real entries of vectors and matrices would be deeply cascaded, which would lead far from our prescribed goals.

In our approach, any multidimensional real entity is treated as an exact real object in its corresponding metric space. In this section the metric is induced by the 2-norm by default.

The usual arithmetic operations on real vectors and matrices can be easily described by means of forward and backward error analysis, analogously to the case of real numbers, thus we skip them here.

## 3.1 Inverse of a Nonsingular Matrix

Let $A \in \mathbf{R}^{n\times n}$ be a nonsingular matrix with known approximate $(\mathtt{A}, \mathtt{d})$. We have

$$\begin{aligned} \|A^{-1} - \mathtt{A}^{-1}\| &= \|A^{-1}A(A^{-1} - \mathtt{A}^{-1})\mathtt{A}\mathtt{A}^{-1}\| \leq \\ &\leq \|A^{-1}\|\|\mathtt{A} - A\|\|\mathtt{A}^{-1}\| \leq \mathtt{d}\|A^{-1}\|\|\mathtt{A}^{-1}\|. \end{aligned} \tag{1}$$

This yields an upper bound for the norm of the inverse: $\|A^{-1}\| \leq \|\mathtt{A}^{-1}\|/(1 - \mathtt{d}\|\mathtt{A}^{-1}\|)$ (note that $\mathtt{d}\|\mathtt{A}^{-1}\| < 1$ if $\mathtt{d}$ is small enough). Assuming that $(\mathtt{A}, \mathtt{d})$ fulfills this condition; for the forward error we get

$$\|A^{-1} - \mathtt{A}^{-1}\| \leq \frac{\mathtt{d}\|\mathtt{A}^{-1}\|^2}{1 - \mathtt{d}\|\mathtt{A}^{-1}\|}.$$

Let now a positive real number $\mathtt{e}$ be given; the goal is to find $\mathtt{d}_\mathtt{e}$ such that for all approximate $(\mathtt{A}_\mathtt{e}, \mathtt{d}_\mathtt{e})$ of $A$, we have $\|A^{-1} - \mathtt{A}_\mathtt{e}^{-1}\| \leq \mathtt{e}$. We apply (1) for $A$ and $\mathtt{A}_\mathtt{e}$, with the fact $\|\mathtt{A}_\mathtt{e}^{-1}\| \leq \|A^{-1}\| + \mathtt{e}$, also using the upper bound for $\|A^{-1}\|$. We get

$$\|A^{-1} - \mathtt{A}_\mathtt{e}^{-1}\| \leq \mathtt{d}_\mathtt{e} \frac{\|\mathtt{A}^{-1}\|}{1 - \mathtt{d}\|\mathtt{A}^{-1}\|}\Big(\frac{\|\mathtt{A}^{-1}\|}{1 - \mathtt{d}\|\mathtt{A}^{-1}\|} + \mathtt{e}\Big).$$

We have to choose $\mathtt{d}_\mathtt{e}$ small enough to get the right hand side smaller than $\mathtt{e}$; for instance, we may set

$$\mathtt{d}_\mathtt{e} := \mathtt{e}/\Big(\frac{\|\mathtt{A}^{-1}\|}{1 - \mathtt{d}\|\mathtt{A}^{-1}\|}\Big(\frac{\|\mathtt{A}^{-1}\|}{1 - \mathtt{d}\|\mathtt{A}^{-1}\|} + \mathtt{e}\Big)\Big) = \frac{\mathtt{e}(1 - \mathtt{d}\|\mathtt{A}^{-1}\|)^2}{\|\mathtt{A}^{-1}\|^2 + \mathtt{e}\|\mathtt{A}^{-1}\| - \mathtt{e}\,\mathtt{d}\|\mathtt{A}^{-1}\|^2}.$$

These error bounds are valid only if the inverse computation on a nonsingular numeric matrix is exact, otherwise $\mathtt{e}$ has to be distributed between the approximation error and the rounding errors.

The computations of this subsection are valid under any consistent norm.

### 3.2 Pseudoinverse

Given $A \in \mathbf{R}^{m\times n}$, the *pseudoinverse* of $A$ is the unique matrix $A^\dagger \in \mathbf{R}^{n\times m}$, which, for any $b \in \mathbf{R}^m$, provides $x := A^\dagger b$ in $\mathbf{R}^n$ that minimizes $\|Ax - b\|$ with the smallest $\|x\|$. In other terms, our main interest is to find the "best" solution for a linear equation system in a numerically stable way. If $A$ is nonsingular, the pseudoinverse coincides with the inverse. For the extensive study of the topic we refer to [3] and [20].

In the general case, pseudoinverse computation is an ill-posed problem; in order to solve it we use a special technique, known as *Tikhonov regularization*.

We solve the following optimization problem:

$$\text{for } \alpha > 0 \text{ find } x \text{ such that } \|Ax - b\|^2 + \alpha\|x\|^2 \text{ is minimal.} \tag{2}$$

For all $\alpha > 0$ problem (2) is continuous in $A, b$, and the solution can be expressed as $A^\alpha b$, where, as $\alpha \to 0$, $A^\alpha$ converges to $A^\dagger$.

We find the optimum of the quadratic function in $x$, at the zero of the derivative $(2\langle A\dot{x}, Ax - b\rangle + 2\alpha\langle \dot{x}, x\rangle = 2\langle \dot{x}, A^T(Ax - b) + \alpha x\rangle)$ which vanishes if $A^T(Ax - b) + \alpha x = 0$, or in other terms at

$$x = (A^T A + \alpha I^{n\times n})^{-1} A^T b,$$

where $I^{n\times n}$ is just the identity in $\mathbf{R}^{n\times n}$. Because the solution of problem (2) approximates $A^\dagger b$ as $\alpha$ goes to zero, we have $\lim_{\alpha\to 0} A^\alpha = A^\dagger$, where $A^\alpha := (A^T A + \alpha I^{n\times n})^{-1} A^T$. We also remark that $A^T A + \alpha I^{n\times n} \in \mathbf{R}^{n\times n}$ is an invertible matrix.

Let $A_0$ approximate $A$, and let $\sigma$ be the minimum of the positive singular values of $A$; assume that we have knowledge of this value. We want to determine how good does $A_0^\alpha$ approximate $A^\dagger$. We have

$$\|A^\dagger - A_0^\alpha\| \leq \|A^\dagger - A^\alpha\| + \|A^\alpha - A_0^\alpha\|, \tag{3}$$

where the second summand can be bounded by standard error bounds computed for the basic operations on matrices. In the following, we determine an upper bound for the first summand.

Let $A = U\Delta V$ be a singular value decomposition, where $U \in \mathbf{R}^{m\times m}, V \in \mathbf{R}^{n\times n}$ are orthogonal matrices, and $\Delta \in \mathbf{R}^{m\times n}$ is a diagonal matrix, with elements $(\sigma_1, \ldots, \sigma_i, \ldots, 0)$. Using this decomposition of $A$, we have

$$\begin{aligned} A^\alpha = (V^T \Delta^2 V + \alpha I^{n\times n})^{-1} V^T \Delta U^T &= (V^T(\Delta^2 + \alpha I^{n\times n})V)^{-1} V^T \Delta U^T = \\ &= V^T(\Delta^2 + \alpha I^{n\times n})^{-1} \Delta U^T. \end{aligned}$$

Thus, by the convergence of $A^\alpha$ to $A^\dagger$, we have $A^\dagger = V^T \Delta^\dagger U^T$, and since we use 2-norm we can write

$$\begin{aligned} \|A^\dagger - A^\alpha\| &= \|V^T(\Delta^\dagger - (\Delta^2 + \alpha I^{n\times n})^{-1}\Delta)U^T\| = \\ &= \|\Delta^\dagger - (\Delta^2 + \alpha I^{n\times n})^{-1}\Delta\|. \end{aligned}$$

We have that

$$(\Delta^2 + \alpha I^{n\times n})^{-1}\Delta = \operatorname{diag}\Big(\frac{\sigma_i}{\sigma_i^2+\alpha}\Big).$$

On the other hand, we have $\Delta^\dagger = \operatorname{diag}(\sigma_i^\dagger)$, where

$$\sigma_i^\dagger = \begin{cases} 1/\sigma_i & \text{if } \sigma_i \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Therefore we get

$$\Delta^\dagger - (\Delta^2 + \alpha I^{n\times n})^{-1}\Delta = \operatorname{diag}\Big(\sigma_i^\dagger - \frac{\sigma_i}{\sigma_i^2+\alpha}\Big) = \operatorname{diag}(\delta_i),$$

where

$$\delta_i = \begin{cases} \frac{\alpha}{\sigma_i(\sigma_i^2+\alpha)} & \text{if } \sigma_i \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Finally, we get

$$\|A^\dagger - A^\alpha\| = \|\operatorname{diag}(\delta_i)\| = \max_{\sigma_i>0} \frac{\alpha}{\sigma_i(\sigma_i^2+\alpha)} = \frac{\alpha}{\sigma(\sigma^2+\alpha)}.$$

The first step in the forward and backward error computation is to find a lower bound $\mathtt{s}$ for $\sigma$. Such an $\mathtt{s}$ is called regularization information for the problem, and it usually depends on $A$. This information, from our point of view, has to come from outside, that is, the user has to provide it. If the given $\mathtt{s}$ is too big, the exact real matrix of the solution may represent a different matrix than $A^\dagger$.

Let $(\mathtt{A}, \mathtt{d})$ be an approximate for $A$, and let $\mathtt{s}$ be a positive lower bound for $\sigma$. Let $\mathtt{e_d}$ denote the forward error of the computation $(\mathtt{A}^T\mathtt{A} + \alpha' I^{n\times n})^{-1}\mathtt{A}^T$ (which can be determined from the forward errors of the basic matrix operations). The forward error of the pseudoinverse computation is then

$$\frac{\alpha'}{\mathtt{s}(\mathtt{s}^2+\alpha')} + \mathtt{e_d}.$$

We would like to determine a close to optimal value for $\alpha'$, which minimizes the forward error. Under the assumptions $\alpha' \le \|\mathtt{A}\|^2$ and $\mathtt{d} \ll \mathtt{s} \ll 1$, by expanding $\mathtt{e_d}$, we get a formula of the following form $\alpha'/(\mathtt{s}^3 + \mathtt{s}\alpha') + \mathtt{d}/\alpha' + c$, where $c$ is constant in $\alpha'$. After simplifying the denominator of this expression, we have to minimize $\alpha'/\mathtt{s}^3 + \mathtt{d}/\alpha'$. We get that a good choice is $\alpha' := \sqrt{\mathtt{s}^3\mathtt{d}}$.

In the backward error computation, for given positive $\mathtt{e}$, we distribute the error between the parts $\|A^\dagger - A^\alpha\|$ and $\|A^\alpha - A_0^\alpha\|$ in (3) equally. Then, we have to determine a suitably small $\alpha'$, so that we have

$$\frac{\alpha'}{\mathtt{s}(\mathtt{s}^2+\alpha')} \le \frac{\mathtt{e}}{2}.$$

For instance, if $\mathtt{e}\,\mathtt{s} < 2$, we may choose $\alpha' = \mathtt{e}\,\mathtt{s}^3/(2 - \mathtt{e}\,\mathtt{s})$.

Finally, we set up an exact real matrix $\mathcal{A}^{\alpha'}$ for $(A^TA + \alpha' I^{n\times n})^{-1}A^T$ by means of the basic matrix operations, and ask for an $\mathtt{e}/2$ approximation: $\mathtt{A}^{\alpha'}_{\mathtt{e}/2}$. According to inequality 3, by the choice of $\alpha'$, $(\mathtt{A}^{\alpha'}_{\mathtt{e}/2}, \mathtt{e})$ is an approximate for $A^\dagger$, provided that the regularization information is correct.

## 4 Real Polynomials

In this section, we describe some basic algorithms for univariate polynomials with real coefficients. From our point of view, the polynomials of degree at most $d$ are just the exact objects in the metric space $\mathcal{P}_d := \mathbf{R}^{d+1}$. For many operation on polynomials, we also need the metric space $\mathcal{M}_d := \mathbf{R}^d$ of monic polynomials of degree $d$. We have some obvious inclusion maps, and a partial map *normation*: $\mathcal{P}_d \to \mathcal{M}_d$ which is defined iff the formal leading coefficient is not zero.

It is rather straightforward to develop algorithms for addition, multiplication, composition, evaluation, division by monics and derivative. In fact, all these functions could be implemented using exact real arithmetic for the coefficients; however, we a get better performance if we implement these operations from the scratch, using the scheme described in Sect. 2 with more accurate forward and backward error. For instance, the forward error of evaluation predicted by the mean value theorem of calculus $f(b) - f(a) = (b-a)f'(\xi)$ is much smaller than the the accumulation of the forward errors arising by Horner's scheme.

In the following subsections, we discuss some problems that cannot be solved solely by arithmetic.

### 4.1 GCD and Squarefree Factorization

The GCD operation is a map from $\mathcal{M}_m \times \mathcal{M}_n$ to $\bigcup_{i=0}^{\min(m,n)} \mathcal{M}_i$. Clearly, the function is discontinuous, because the degree is discrete and cannot depend continuously on the input. Thus, the problem of GCD computation is ill-posed. To solve it, we use a regularization technique similar to the one for computing the pseudo-inverse in Sect. 3, and which was proposed in [6].

Let $P \in \mathcal{M}_m, Q \in \mathcal{M}_n$ be monic polynomials, $m \le n$. The Sylvester matrix $S = S(P,Q)$ is an $(m+n)\times(m+n)$ matrix with entries from the set of coefficients of $P$ and $Q$ (we refer to [21] for a more precise definition).

Let $V := \mathbf{R}^{m+n}$ be endowed with the $\infty$-norm. The function $sv : \mathbf{R}^{N\times N} \to \mathbf{R}^N$ computing the ordered $N$-tuple of singular values is continuous; in fact we have $\|sv(A) - sv(B)\| \le \|A - B\|$, which gives a very easy formula for the forward and backward error (see [11]). Suppose that we have a lower bound $\mathtt{s}$ for the smallest nonzero singular value of the Sylvester matrix $S$. Then we can compute the rank of $S$ by computing an $\mathtt{s}/2$-estimate of $sv(S)$: the singular values with approximating element less than $\mathtt{s}/2$ are zero, and the singular values with approximating element greater than $\mathtt{s}/2$ must be nonzero. If the rank is $m+n-r$, then $r$ is the degree of the GCD (see [21]).

But if we know the degree of the GCD, then we can compute the GCD by solving a system of linear equations. If $R$ is the GCD, and $\deg(R) = k$, then there is a unique pair of polynomials $U \in \mathcal{P}_{n-k-1}$, $V \in \mathcal{P}_{m-k-1}$, such that $UP+VQ=R$. The polynomial $UP+VQ$ has formal degree $m+n-k-1$, and we know the coefficients of $x^{m+n-k-1}$, $x^{m+n-k-2}$, …, $x^k$ (namely $0,\dots,0,1$). This gives a system $(*)$ of $m+n-2k$ linear equations in equally many unknowns. The system is nonsingular, so we could solve it exactly by the methods described in Sect. 3. However, we prefer to solve it via the pseudo-inverse, i.e. treating it as a least squares problem. The reason is the following.

The regularization information $\mathtt{s}$ has to come from the user. If the smallest nonzero singular value was estimated too low, then $\mathtt{s}$ is still a lower bound, and the computed result is still exact (the effect giving too small bounds just leads to a worse performance, because the singular values must be computed more accurate than necessary). If the smallest nonzero singular value was estimated too high, then the rank of $S$ is larger than expected, and the degree of the GCD is smaller than expected. The system $(*)$ becomes singular, and the exact algorithm for resolving it does not terminate. On the other hand, if we treat $(*)$ as a least squares problem, then we get a solution anyway. The computed GCD $R'$ is still a linear combination of $P$ and $Q$ but it is not an exact divisor of $P$ and $Q$. However, the remainder of $P$ or $Q$ modulo $R'$ is small when the the given $\mathtt{s}$ is small.

Squarefree factorization is another ill-posed problem, but we can reduce it to GCD computation. There are several ways to do this for polynomials over the rational numbers (or over any other computable field). Not all these reductions work also in the exact real case; we have to be careful that we never estimate the degree of an intermediate result too high, whereas an error in the other direction is not so disturbing (note that wrong degree estimates are unavoidable in the case that the regularization information was incorrect). Algorithm 1 works because it has this monotonicity property (see also [1]).

**Algorithm 1** Squarefree($P$)

```
if deg(P) = 1 then
  return ((P, 1));
else
  P' := derivative of p divided by deg(P);
  Q := GCD(P, P');
  L := Squarefree(Q);
  for all (F, e) in L do
    P := quotient(P, F^(e+1));
    in L, replace (F, e) by (F, e + 1);
  end for
  append (P, 1) to L;
  return L
end if
```

If the given regularization information (used in the GCD computation) was correct, then the algorithm returns a list $((F_1,e_1),\dots,(F_r,e_r))$, such that the

$F_i$ are squarefree and relatively prime, and $F_1^{e_1} \dots F_r^{e_r} = P$. Otherwise, the $F_i$ are still squarefree and relatively prime, but $F_1^{e_1} \dots F_r^{e_r}$ is not exactly equal to $P$ (but close if the estimated lower bound $\mathtt{s}$ is small). In this case, the sum of the degrees of the $F_i{}'$ is smaller than the number of distinct complex roots of $P$ (in other words, we have made the mistake of identifying two roots which are closely together).

### 4.2 Roots Computation

Let $\mathbf{C}$ be the space of complex numbers, endowed with the Euclidean norm. Any polynomial of degree $n$ has $n$ complex roots, counted with multiplicity; but there is no continuous map $\mathcal{M}_n \to \mathbf{C}^n$, and the reason is that there is no continuous way of selecting one particular root. To avoid this difficulty, we introduce the space $\mathcal{S}_n$ as the set of orbits under the permutation group $S_n$ permuting the coordinates of $\mathbf{C}^n$. The metric is defined by $d([x],[y]) := \min_{\sigma \in S_n} d_0(x, \sigma(y))$, where $d_0$ is the metric induced by the $\infty$-norm in $\mathbf{C}^n$. The roots map $rt : \mathcal{M}_n \to \mathcal{S}_n$ is continuous, and we want to compute it in the sense of exact real computation. It turns out that it is more practical to create this exact real object by an `iterate` function rather than by an `approx` function, because the a posteriori error bound is typically much smaller than the a priori error bound of a root, i.e. we can find a more realistic accuracy when we already know the approximating value.

For the moment, let us assume that the input polynomial $P$ is squarefree. Then the approximating polynomial $\mathtt{P}$ is also squarefree, if the accuracy $\mathtt{e}$ is sufficiently small. We compute its roots by a numerical algorithm. For each computed root $\mathtt{z_i}$ (which is in general not an exact root of $\mathtt{P}$), we expect a nearby root of the exact polynomial $P$. If $P'(\mathtt{z_i}) \neq 0$, then there is a root with distance at most $\frac{|P(\mathtt{z_i})|}{|nP'(\mathtt{z_i})|}$ (see [19]). This does not necessarily imply that we have information about every root of $P$: it could be that a root of $P$ lies in two of the disks with midpoint $\mathtt{z_i}$ and radius $\mathtt{r_i} \geq \frac{nP(\mathtt{z_i})}{P'(\mathtt{z_i})}$, and there is another root of $P$ which is outside of all disks. But for small $\mathtt{e}$, the radii become small, and the disks will be disjoint. Thus, we have computed an approximate $((\mathtt{z_1}, \dots, \mathtt{z_n}), \max_{i=1}^n \mathtt{r_i})$ of $rt(P)$.

Now, let us drop the assumption that $P$ is squarefree. Then we can compute its squarefree decomposition with an arbitrary regularization parameter $\mathtt{s} > 0$ (we take $\mathtt{s} = 10^{-5}$ by default). If the regularization information is correct, i.e. if $s$ is a lower bound for all positive singular values in all Sylvester matrices occurring in algorithm 1, then we get the correct decomposition $P = F_1^{e_1} \dots F_r^{e_r}$. Otherwise, we get the squarefree decomposition of a nearby polynomial $P_1$, and we know that the number of distinct roots of $P_1$ is smaller than the number of distinct roots of $P$. By decreasing $\mathtt{s}$, we can make the distance $\|P - P_1\|$ arbitrarily small.

Now we compute the roots of the $F_1, \dots, F_r$ by the above algorithm for the squarefree case. Let $\mathtt{z_{i,j}}$ be a computed root of $F_i$. We expect a cluster of $e_i$ roots, counted with multiplicity, near $\mathtt{z_{i,j}}$. There are several methods for cluster validation, see [15, 12, 1, 4]; below we do the validation by a simple formula.

In the case $P^{e_i}(\mathtt{z_{i,j}}) \neq 0$, the distance can be estimated by the formula below. This inequation is true for small regularization parameter: if our decomposition is exact, then it is certainly true, otherwise we have ${P_1}^{e_i}(\mathtt{z_{i,j}}) \neq 0$ and this implies the claim whenever $\|P - P_1\|$ is sufficiently small. Here is the precise statement.

**Lemma 1.** *Let $P$ be a monic polynomial of degree $n$. Let $z \in \mathbf{C}$ be such that $P^{(k)}(z) \neq 0$. Then there are at least $k$ roots, counted with multiplicity, in the disk around $z$ with radius*

$$M_{P,z,k} := 2n \max_{j=0}^{k-1} \sqrt[k-j]{\frac{kk!|P^{(j)}(z)|}{j!|P^{(k)}(z)|}}.$$

*Proof.* Without loss of generality, we assume $z = 0$. Let $p_0, \ldots, p_{n-1}, p_n = 1$ be the coefficients of $P$. We assume also that $p_0 \neq 0$; the case $p_0 = 0$ follows by continuity.

Let $y_1, \ldots, y_n$ be the roots of the reverse polynomial $\bar{P}(x) := x^n P(1/x)$, ordered by absolute value, largest first. Assume, indirectly, that there are no $k$ roots of $P$ in the circle of radius $M := M_{P,0,k}$. Then $|y_i| < \frac{1}{M}$ for $i = k, \ldots, n$. For $i = 0, \ldots, k$, we define $A_i$ as the sum of all products of $i$ roots from $y_1, \ldots, y_{k-1}$, and $B_i$ as the sum of all products of $i$ roots from $y_k, \ldots, y_n$ ($A_0 = B_0 = 1$), and $C_i := \sum_{j=0}^{i} \left|\frac{p_j}{p_0}\right| \left(\frac{2n}{M}\right)^{i-j}$. The $B_i$ can be estimated by

$$B_i < \binom{n-k+1}{i} \frac{1}{M^i} \leq \left(\frac{n}{M}\right)^l.$$

We claim that $|A_i| \leq C_i$ for $i = 0, \ldots, k-1$. From Vieta's formula, we obtain

$$\left|\frac{p_i}{p_0}\right| = \left|\sum_{j=0}^{i} A_{i-j} B_j\right|$$

and by induction on $i$, we get

$$\begin{aligned} B_i \leq \left|\frac{p_i}{p_0}\right| + \sum_{j=0}^{i-1} |A_j B_{i-j}| \leq \left|\frac{p_i}{p_0}\right| + \sum_{j=0}^{i-1} R_j \left(\frac{n}{M}\right)^{i-j} = \\ = \left|\frac{p_i}{p_0}\right| + \sum_{j=0}^{i-1} \left|\frac{p_j}{p_0}\right| (2^{i-j} - 1) \left(\frac{n}{M}\right)^{i-j} \leq R_i, \end{aligned}$$

which proves the claim. Now we get

$$\left|\frac{p_k}{p_0}\right| \leq \sum_{i=0}^{k-1} R_i \left(\frac{n}{M}\right)^{k-i} = \sum_{i=0}^{k-1} \left|\frac{p_i}{p_0}\right| (2^{k-i} - 1) \left(\frac{n}{M}\right)^{k-i} < \sum_{i=0}^{k-1} \left|\frac{p_i}{p_0}\right| \left(\frac{2n}{M}\right)^{k-i}.$$

By the choice of $M$, we have $\left(\frac{2n}{M}\right)^{k-i} \leq \left|\frac{p_0}{kp_i}\right|$ for all $i = 0, \ldots, k-1$. So we get $\left|\frac{p_k}{p_0}\right| < \left|\frac{p_k}{p_0}\right|$, which is a contradiction.

As the regularization parameter and the accuracy tend to zero, the radii $M_{P,\mathbf{z_{i,j}},e_i}$ become arbitrary small. Eventually, one of two things will happen. Either the disks will become disjoint. In this case, we have computed an approximate of $rt(P)$. Or the rank of the Sylvester matrix increases in one of the GCD computations required in the squarefree computation, which means that one of the clusters splits into several. The second can only happen a finite number of times, so that we will always come to an approximate of $rt(P)$.

## References


[1] Aberth, O. (1998): Precise numerical methods using C++. Academic Press Inc., San Diego, CA

[2] Arsenin, V., and Tikhonov, A. N. (1977): Solutions of ill-posed problems. Wiley

[3] Ben-Israel, A., and Greville, T. N. E. (1974): Generalized Inverses: Theory and Applications. Wiley

[4] Bini, D., and Fiorentino, G. (2000): Design, analysis, and implementation of a multiprecision polynomial rootfinder. Num. Alg. 23, 127–173

[5] Boehm, H., and Cartwright, R. (1990): Exact real arithmetic, formulating real numbers as functions. In: Research Topics in Functional Programming, Turner D. (ed.) Addison-Wesley, 43–64

[6] Corless, R., Gianni, P., Trager, B., and Watt, S. (1995): The singular value decomposition for polynomial systems. In: Proc. ISSAC'95, ACM Press, 195–207

[7] Engl, H. (1999): Regularization methods for solving inverse problems. In: Proc. ICIAM 99

[8] Escardó, M. H. (2000): Exact numerical computation. In: ISSAC'00, ACM Press, tutorial.

[9] Gianantonio, P. (1993): Real number computability and domain theory. In: Proc. Math. Found. Comp. Sci. '93, Springer, 413–422

[10] Gonzalez-Vega, L., Rouillier, F., Roy, M.-F., and Trujillo, G. (2000): Symbolic recipes for real solutions. Algorith. Comp. Math. 4, 121–167

[11] Higham, N. J. (1993): Accuracy and stability of numerical algorithms. SIAM

[12] Hribernig, V., and Stetter, H.-J. (1997) Detection and validation of clusters of polynomial zeroes. Journal of Symbolic Computation 24, 667–682

[13] Ménissier-Morain, V. (2001): Arbitrary precision real arithmetic: design and algorithms. Preprint

[14] Moore, R. E. (1966): Interval Anaysis. Prentice-Hall, Englewood Cliffs, N.J.

[15] Neumaier, A. (1988): An existence test for root clusters and multiple roots. ZAMM 68, 257–259

[16] Noda, M. T., and Sasaki, T. (1989): Approximate square-free decomposition and rootfinding for ill-conditioned algebraic equations. J. Inf. Proc. 12, 159–168

[17] Roy, M.-F., and Szpirglas, A. (1990): Complexity of computation of real algebraic numbers. Journal of Symbolic Computation 10, 39–51

[18] Stetter, H.-J. (2000): Condition analysis of overdetermined algebraic problems. In: Proc. CASC 2000, 345–365.

[19] Strzeboński, A. (1997): Computing in the field of complex algebraic numbers. Journal of Symbolic Computation 24, 647–656

[20] Tran, Q.-N. (1996): A Hybrid Symbolic-Numerical Approach in Computer Aided Geometric Design (CAGD) and Visualization. PhD thesis, Johannes Kepler University, RISC-Linz

[21] Winkler, F. (1996): Polynomial algorithms in computer algebra. Springer