

Solving Cubic and Quartic Equations*

René Thiemann

March 24, 2023

Abstract

We formalize Cardano's formula to solve a cubic equation

$$ax^3 + bx^2 + cx + d = 0,$$

as well as Ferrari's formula to solve a quartic equation [1]. We further turn both formulas into executable algorithms based on the algebraic number implementation in the AFP [2]. To this end we also slightly extended this library, namely by making the minimal polynomial of an algebraic number executable, and by defining and implementing n -th roots of complex numbers.

Contents

1	Ferrari's formula for solving quartic equations	2
1.1	Translation to depressed case	2
1.2	Solving the depressed case via Ferrari's formula	2
2	Cardano's formula for solving cubic equations	3
2.1	Translation to depressed case	3
2.2	Solving the depressed case in arbitrary fields	3
2.3	Solving the depressed case for complex numbers	3
2.4	Solving the depressed case for real numbers	4
3	n-th roots of complex numbers	4
3.1	An algorithm to compute all complex roots of (algebraic) complex numbers	4
3.2	A definition of <i>the</i> complex root of a complex number	5
4	Algorithms to compute all complex and real roots of a cubic polynomial	7

*Supported by FWF (Austrian Science Fund) project Y757.

5 Algorithms to compute all complex and real roots of a quartic polynomial

9

1 Ferrari's formula for solving quartic equations

theory *Ferraris-Formula*

imports

Polynomial-Factorization.Explicit-Roots
Polynomial-Interpolation.Ring-Hom-Poly
Complex-Geometry.More-Complex

begin

1.1 Translation to depressed case

Solving an arbitrary quartic equation can easily be turned into the depressed case, i.e., where there is no cubic part.

lemma *to-depressed-quartic*: **fixes** $a_4 :: 'a :: \text{field-char-0}$

assumes $a_4 \neq 0$

and $b: b = a_3 / a_4$

and $c: c = a_2 / a_4$

and $d: d = a_1 / a_4$

and $e: e = a_0 / a_4$

and $p: p = c - (3/8) * b^2$

and $q: q = (b^3 - 4*b*c + 8 * d) / 8$

and $r: r = (-3 * b^4 + 256 * e - 64 * b * d + 16 * b^2 * c) / 256$

and $x: x = y - b/4$

shows $a_4 * x^4 + a_3 * x^3 + a_2 * x^2 + a_1 * x + a_0 = 0$

$\longleftrightarrow y^4 + p * y^2 + q * y + r = 0$

<proof>

lemma *biquadratic-solution*: **fixes** $p q :: 'a :: \text{field-char-0}$

shows $y^4 + p * y^2 + q = 0 \longleftrightarrow (\exists z. z^2 + p * z + q = 0 \wedge z = y^2)$

<proof>

1.2 Solving the depressed case via Ferrari's formula

lemma *depressed-quartic-Ferrari*: **fixes** $p q r :: 'a :: \text{field-char-0}$

assumes *cubic-root*: $8*m^3 + (8 * p) * m^2 + (2 * p^2 - 8 * r) * m - q^2 = 0$

and $q0: q \neq 0$ — otherwise m might be zero, so a is zero and then there is a division by zero in b1 and b2

and *sqrt*: $a * a = 2 * m$

and $b1: b1 = p / 2 + m - q / (2 * a)$

and $b2: b2 = p / 2 + m + q / (2 * a)$

shows $y^4 + p * y^2 + q * y + r = 0 \longleftrightarrow \text{poly} [:b1, a, 1:] y = 0 \vee \text{poly} [:b2, -a, 1:] y = 0$

<proof>

end

2 Cardano's formula for solving cubic equations

theory *Cardanos-Formula*

imports

Polynomial-Factorization.Explicit-Roots

Polynomial-Interpolation.Ring-Hom-Poly

Complex-Geometry.More-Complex

Algebraic-Numbers.Complex-Roots-Real-Poly

begin

2.1 Translation to depressed case

Solving an arbitrary cubic equation can easily be turned into the depressed case, i.e., where there is no quadratic part.

lemma *to-depressed-cubic*: **fixes** $a :: 'a :: \text{field-char-0}$

assumes $a: a \neq 0$

and $xy: x = y - b / (3 * a)$

and $e: e = (c - b^2 / (3 * a)) / a$

and $f: f = (d + 2 * b^3 / (27 * a^2) - b * c / (3 * a)) / a$

shows $(a * x^3 + b * x^2 + c * x + d = 0) \longleftrightarrow y^3 + e * y + f = 0$
<proof>

2.2 Solving the depressed case in arbitrary fields

lemma *cubic-depressed*: **fixes** $e :: 'a :: \text{field-char-0}$

assumes $yz: e \neq 0 \implies z^2 - y * z - e / 3 = 0$

and $u: e \neq 0 \implies u = z^3$

and $v: v = -(e^3 / 27)$

shows $y^3 + e * y + f = 0 \longleftrightarrow (\text{if } e = 0 \text{ then } y^3 = -f \text{ else } u^2 + f * u + v = 0)$
<proof>

2.3 Solving the depressed case for complex numbers

In the complex-numbers-case, the quadratic equation for u is always solvable, and the main challenge here is prove that it does not matter which solution of the quadratic equation is considered (this is the `diff:False` case in the proof below.)

lemma *solve-cubic-depressed-Cardano-complex*: **fixes** $e :: \text{complex}$

assumes $e0: e \neq 0$

and $v: v = -(e^3 / 27)$

and $u: u^2 + f * u + v = 0$

shows $y^3 + e * y + f = 0 \longleftrightarrow (\exists z. z^3 = u \wedge y = z - e / (3 * z))$
<proof>

2.4 Solving the depressed case for real numbers

definition *discriminant-cubic-depressed* :: 'a :: comm-ring-1 \Rightarrow 'a \Rightarrow 'a where
discriminant-cubic-depressed e f = - (4 * e³ + 27 * f²)

lemma *discriminant-cubic-depressed*: **assumes** [-x,1:] * [-y,1:] * [-z,1:] =
[:f,e,0,1:]

shows *discriminant-cubic-depressed* e f = (x-y)² * (x-z)² * (y-z)²
<proof>

If the discriminant is negative, then there is exactly one real root

lemma *solve-cubic-depressed-Cardano-real*: **fixes** e f v u :: real

defines y1 \equiv root 3 u - e / (3 * root 3 u)

and $\Delta \equiv$ *discriminant-cubic-depressed* e f

assumes e0: e \neq 0

and v: v = - (e³ / 27)

and u: u² + f * u + v = 0

shows y1³ + e * y1 + f = 0

$\Delta \neq 0 \implies y^3 + e * y + f = 0 \implies y = y1$
<proof>

If the discriminant is non-negative, then all roots are real

lemma *solve-cubic-depressed-Cardano-all-real-roots*: **fixes** e f v :: real **and** y ::
complex

defines $\Delta \equiv$ *discriminant-cubic-depressed* e f

assumes Delta: $\Delta \geq 0$

and rt: y³ + e * y + f = 0

shows y \in \mathbb{R}

<proof>

end

3 n-th roots of complex numbers

theory *Complex-Roots*

imports

Complex-Geometry.More-Complex

Algebraic-Numbers.Complex-Algebraic-Numbers

Factor-Algebraic-Polynomial.Roots-via-IA

HOL-Library.Product-Lexorder

begin

3.1 An algorithm to compute all complex roots of (algebraic) complex numbers

definition *all-roots* :: nat \Rightarrow complex \Rightarrow complex list **where**

all-roots n x = (if n = 0 then [] else

if algebraic x then

(let p = min-int-poly x;

```

    q = poly-nth-root n p;
    xs = complex-roots-of-int-poly q
    in filter (λ y. y^n = x) xs)
else (SOME ys. set ys = {y. y^n = x}))

```

lemma *all-roots*: **assumes** $n \neq 0$ **shows** $\text{set } (\text{all-roots } n \ x) = \{y. y^n = x\}$

<proof>

TODO: One might change *complex-roots-of-int-poly* to *complex-roots-of-int-poly3* in order to avoid an unnecessary factorization of an integer polynomial. However, then this change already needs to be performed within the definition of *all-roots*.

lift-definition *all-roots-part1* :: $\text{nat} \Rightarrow \text{complex} \Rightarrow \text{complex}$ *genuine-roots-aux* **is**
 $\lambda n \ x. \text{if } n = 0 \vee x = 0 \vee \neg \text{algebraic } x \text{ then } (1, [], 0, \text{filter-fun-complex } 1)$
 else let $p = \text{min-int-poly } x;$
 $q = \text{poly-nth-root } n \ p;$
 $\text{zeros} = \text{complex-roots-of-int-poly } q;$
 $r = \text{Polynomial.monom } 1 \ n - [:x:]$
 in $(r, \text{zeros}, n, \text{filter-fun-complex } r)$

<proof>

lemma *all-roots-code*[code]:

```

all-roots n x = (if n = 0 then [] else if x = 0 then [0]
  else if algebraic x then genuine-roots-impl (all-roots-part1 n x)
  else Code.abort (STR "all-roots invoked on non-algebraic number") (λ -.
all-roots n x))

```

<proof>

3.2 A definition of *the* complex root of a complex number

While the definition of the complex root is quite natural and easy, the main task is a criterion to determine which of all possible roots of a complex number is the chosen one.

definition *croot* :: $\text{nat} \Rightarrow \text{complex} \Rightarrow \text{complex}$ **where**
 $\text{croot } n \ x = (\text{rcis } (\text{root } n \ (\text{cmod } x)) (\text{Arg } x / \text{of-nat } n))$

lemma *croot-0*[simp]: $\text{croot } n \ 0 = 0$ $\text{croot } 0 \ x = 0$

<proof>

lemma *croot-power*: **assumes** $n: n \neq 0$

shows $(\text{croot } n \ x) ^ n = x$

<proof>

lemma *Arg-of-real*: $\text{Arg } (\text{of-real } x) =$

$(\text{if } x < 0 \text{ then } \pi \text{ else } 0)$

<proof>

lemma *Arg-rcis-cis[simp]*: **assumes** $x > 0$
shows $\text{Arg} (\text{rcis } x \ y) = \text{Arg} (\text{cis } y)$
 $\langle \text{proof} \rangle$

lemma *cis-Arg-1[simp]*: $\text{cis} (\text{Arg } 1) = 1$
 $\langle \text{proof} \rangle$

lemma *cis-Arg-power[simp]*: **assumes** $x \neq 0$
shows $\text{cis} (\text{Arg} (x \wedge^n)) = \text{cis} (\text{Arg } x * \text{real } n)$
 $\langle \text{proof} \rangle$

lemma *Arg-croot[simp]*: $\text{Arg} (\text{croot } n \ x) = \text{Arg } x / \text{real } n$
 $\langle \text{proof} \rangle$

lemma *cos-abs[simp]*: $\cos (\text{abs } x :: \text{real}) = \cos x$
 $\langle \text{proof} \rangle$

lemma *cos-mono-le*: **assumes** $\text{abs } x \leq \pi$
and $\text{abs } y \leq \pi$
shows $\cos x \leq \cos y \longleftrightarrow \text{abs } y \leq \text{abs } x$
 $\langle \text{proof} \rangle$

lemma *abs-add-2-mult-bound*: **fixes** $x :: 'a :: \text{linordered-idom}$
assumes $xy: |x| \leq y$
shows $|x| \leq |x + 2 * \text{of-int } i * y|$
 $\langle \text{proof} \rangle$

lemma *abs-eq-add-2-mult*: **fixes** $y :: 'a :: \text{linordered-idom}$
assumes $\text{abs-id}: |x| = |x + 2 * \text{of-int } i * y|$
and $xy: -y < x \leq y$
and $i: i \neq 0$
shows $x = y \wedge i = -1$
 $\langle \text{proof} \rangle$

This is the core lemma. It tells us that *croot* will choose the principal root, i.e. the root with largest real part and if there are two roots with identical real part, then the largest imaginary part. This criterion will be crucial for implementing *croot*.

lemma *croot-principal*: **assumes** $n: n \neq 0$
and $y: y \wedge^n = x$
and $\text{neq}: y \neq \text{croot } n \ x$
shows $\text{Re } y < \text{Re} (\text{croot } n \ x) \vee \text{Re } y = \text{Re} (\text{croot } n \ x) \wedge \text{Im } y < \text{Im} (\text{croot } n \ x)$
 $\langle \text{proof} \rangle$

lemma *croot-unique*: **assumes** $n: n \neq 0$
and $y: y \wedge^n = x$
and $y\text{-max-Re-Im}: \bigwedge z. z \wedge^n = x \implies \text{Re } z < \text{Re } y \vee \text{Re } z = \text{Re } y \wedge \text{Im } z \leq \text{Im } y$

shows $\text{croot } n \ x = y$
 ⟨proof⟩

lemma *csqrt-is-croot-2*: $\text{csqrt} = \text{croot } 2$
 ⟨proof⟩

lemma *croot-via-root-selection*: **assumes** *roots*: set $ys = \{ y. y^n = x \}$
and $n: n \neq 0$
shows $\text{croot } n \ x = \text{arg-min-list } (\lambda y. (- \text{Re } y, - \text{Im } y)) \ ys$
 (**is** $- = \text{arg-min-list } ?f \ ys$)
 ⟨proof⟩

lemma *croot-impl*[code]: $\text{croot } n \ x = (\text{if } n = 0 \text{ then } 0 \text{ else}$
 $\text{arg-min-list } (\lambda y. (- \text{Re } y, - \text{Im } y)) (\text{all-croots } n \ x))$
 ⟨proof⟩

end

4 Algorithms to compute all complex and real roots of a cubic polynomial

theory *Cubic-Polynomials*

imports

Cardanos-Formula

Complex-Roots

begin

The real case where a result is only delivered if the discriminant is negative

definition *solve-depressed-cubic-Cardano-real* :: $\text{real} \Rightarrow \text{real} \Rightarrow \text{real option}$ **where**
 $\text{solve-depressed-cubic-Cardano-real } e \ f = ($
 $\text{if } e = 0 \text{ then } \text{Some } (\text{root } 3 \ (-f)) \text{ else}$
 $\text{let } v = - (e \wedge 3 / 27) \text{ in}$
 $\text{case } \text{roots2 } [:v,f,1:] \text{ of}$
 $[u,-] \Rightarrow \text{let } rt = \text{root } 3 \ u \text{ in } \text{Some } (rt - e / (3 * rt))$
 $| - \Rightarrow \text{None})$

lemma *solve-depressed-cubic-Cardano-real*:

assumes *solve-depressed-cubic-Cardano-real* $e \ f = \text{Some } y$

shows $\{y. y^3 + e * y + f = 0\} = \{y\}$

⟨proof⟩

The complex case

definition *solve-depressed-cubic-complex* :: $\text{complex} \Rightarrow \text{complex} \Rightarrow \text{complex list}$
where

$\text{solve-depressed-cubic-complex } e \ f = (\text{let}$
 $ys = (\text{if } e = 0 \text{ then } \text{all-croots } 3 \ (-f) \text{ else } (\text{let}$
 $u = \text{hd } (\text{roots2 } [: - (e \wedge 3 / 27) ,f,1:]);$
 $zs = \text{all-croots } 3 \ u$

```

in map (λ z. z - e / (3 * z)) zs)
in remdups ys)

```

lemma *solve-depressed-cubic-complex-code*[code]:

```

solve-depressed-cubic-complex e f = (let
  ys = (if e = 0 then all-roots 3 (- f) else (let
    f2 = f / 2;
    u = - f2 + csqrt (f2^2 + e ^ 3 / 27);
    zs = all-roots 3 u
    in map (λ z. z - e / (3 * z)) zs))
in remdups ys)
⟨proof⟩

```

lemma *solve-depressed-cubic-complex*: $y \in \text{set } (\text{solve-depressed-cubic-complex } e \ f)$

```

⟷ (y^3 + e * y + f = 0)
⟨proof⟩

```

For the general real case, we first try Cardano with negative discriminant and only if it is not applicable, then we go for the calculation using complex numbers. Note that for non-negative delta no filter is required to identify the real roots from the list of complex roots, since in that case we already know that all roots are real.

definition *solve-depressed-cubic-real* :: $\text{real} \Rightarrow \text{real} \Rightarrow \text{real list}$ **where**

```

solve-depressed-cubic-real e f = (case solve-depressed-cubic-Cardano-real e f
of Some y ⇒ [y]
| None ⇒ map Re (solve-depressed-cubic-complex (of-real e) (of-real f)))

```

lemma *solve-depressed-cubic-real-code*[code]: *solve-depressed-cubic-real* e f =

```

(if e = 0 then [root 3 (-f)] else
let v = e ^ 3 / 27;
  f2 = f / 2;
  f2v = f2^2 + v in
if f2v > 0 then
  let u = -f2 + sqrt f2v;
    rt = root 3 u
  in [rt - e / (3 * rt)]
else
let ce3 = of-real e / 3;
  u = - of-real f2 + csqrt (of-real f2v) in
map Re (remdups (map (λrt. rt - ce3 / rt) (all-roots 3 u))))
⟨proof⟩

```

lemma *solve-depressed-cubic-real*: $y \in \text{set } (\text{solve-depressed-cubic-real } e \ f)$

```

⟷ (y^3 + e * y + f = 0)
⟨proof⟩

```

Combining the various algorithms

lemma *degree3-coeffs*: *degree* $p = 3 \implies$
 $\exists a b c d. p = [: d, c, b, a :] \wedge a \neq 0$
 ⟨*proof*⟩

definition *roots3-generic* :: (*'a* :: *field-char-0* \Rightarrow *'a* \Rightarrow *'a list*) \Rightarrow *'a poly* \Rightarrow *'a list*
where

roots3-generic depressed-solver $p =$ (*let*
 $cs = coeffs\ p;$
 $a = cs\ !\ 3; b = cs\ !\ 2; c = cs\ !\ 1; d = cs\ !\ 0;$
 $a3 = 3 * a;$
 $ba3 = b / a3;$
 $b2 = b * b;$
 $b3 = b2 * b;$
 $e = (c - b2 / a3) / a;$
 $f = (d + 2 * b3 / (27 * a^2) - b * c / a3) / a;$
 $roots = depressed-solver\ e\ f$
in map ($\lambda y. y - ba3$) *roots*)

lemma *roots3-generic*: **assumes** *deg*: *degree* $p = 3$
and *solver*: $\bigwedge e f y. y \in set\ (depressed-solver\ e\ f) \iff y^3 + e * y + f = 0$
shows $set\ (roots3-generic\ depressed-solver\ p) = \{x. poly\ p\ x = 0\}$
 ⟨*proof*⟩

definition *croots3* :: *complex poly* \Rightarrow *complex list* **where**
 $croots3 = roots3-generic\ solve-depressed-cubic-complex$

lemma *croots3*: **assumes** *deg*: *degree* $p = 3$
shows $set\ (croots3\ p) = \{x. poly\ p\ x = 0\}$
 ⟨*proof*⟩

definition *rroots3* :: *real poly* \Rightarrow *real list* **where**
 $rroots3 = roots3-generic\ solve-depressed-cubic-real$

lemma *rroots3*: **assumes** *deg*: *degree* $p = 3$
shows $set\ (rroots3\ p) = \{x. poly\ p\ x = 0\}$
 ⟨*proof*⟩

end

5 Algorithms to compute all complex and real roots of a quartic polynomial

theory *Quartic-Polynomials*
imports
Ferraris-Formula
Cubic-Polynomials
begin

The complex case is straight-forward

definition *solve-depressed-quartic-complex* :: complex \Rightarrow complex \Rightarrow complex \Rightarrow complex list **where**

solve-depressed-quartic-complex p q r = remdups (if q = 0 then
 (concat (map (λ z. let y = csqrt z in [y, -y]) (croots2 [:r,p,1:]))) else
 let cubics = croots3 [- (q²), 2 * p² - 8 * r, 8 * p, 8];
 m = hd cubics; — select any root of the cubic polynomial
 a = csqrt (2 * m);
 p2m = p / 2 + m;
 q2a = q / (2 * a);
 b1 = p2m - q2a;
 b2 = p2m + q2a
 in (croots2 [:b1,a,1:] @ croots2 [:b2,-a,1:])))

lemma *solve-depressed-quartic-complex*: $x \in \text{set } (\text{solve-depressed-quartic-complex } p \ q \ r)$

$\longleftrightarrow (x^4 + p * x^2 + q * x + r = 0)$
 <proof>

The main difference in the real case is that a specific cubic root has to be used, namely a positive one. In the soundness proof we show that such a cubic root always exists.

definition *solve-depressed-quartic-real* :: real \Rightarrow real \Rightarrow real \Rightarrow real list **where**

solve-depressed-quartic-real p q r = remdups (if q = 0 then
 (concat (map (λ z. rroots2 [-z,0,1:]) (rroots2 [:r,p,1:]))) else
 let cubics = rroots3 [- (q²), 2 * p² - 8 * r, 8 * p, 8];
 m = the (find (λ m. m > 0) cubics); — select any positive root of the
 cubic polynomial
 a = sqrt (2 * m);
 p2m = p / 2 + m;
 q2a = q / (2 * a);
 b1 = p2m - q2a;
 b2 = p2m + q2a
 in (rroots2 [:b1,a,1:] @ rroots2 [:b2,-a,1:])))

lemma *solve-depressed-quartic-real*: $x \in \text{set } (\text{solve-depressed-quartic-real } p \ q \ r)$

$\longleftrightarrow (x^4 + p * x^2 + q * x + r = 0)$
 <proof>

Combining the various algorithms

lemma *numeral-4-eq-4*: 4 = Suc (Suc (Suc (Suc 0)))

<proof>

lemma *degree4-coeffs*: degree p = 4 \implies

$\exists a \ b \ c \ d \ e. p = [: e, d, c, b, a :] \wedge a \neq 0$

<proof>

definition *roots4-generic* :: ('a :: field-char-0 \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a list) \Rightarrow 'a poly \Rightarrow 'a list **where**

roots4-generic depressed-solver p = (let

```

cs = coeffs p;
cs = coeffs p;
a4 = cs ! 4; a3 = cs ! 3; a2 = cs ! 2; a1 = cs ! 1; a0 = cs ! 0;
b = a3 / a4;
c = a2 / a4;
d = a1 / a4;
e = a0 / a4;
b2 = b * b;
b3 = b2 * b;
b4 = b3 * b;
b4' = b / 4;
p = c - 3/8 * b2;
q = (b3 - 4*b*c + 8 * d) / 8;
r = (-3 * b4 + 256 * e - 64 * b * d + 16 * b2 * c) / 256;
roots = depressed-solver p q r
in map (λ y. y - b4') roots)

```

lemma *roots4-generic*: **assumes** *deg*: degree $p = 4$
and *solver*: $\bigwedge p\ q\ r\ y. y \in \text{set } (\text{depressed-solver } p\ q\ r) \longleftrightarrow y^4 + p * y^2 + q * y + r = 0$
shows $\text{set } (\text{roots4-generic depressed-solver } p) = \{x. \text{poly } p\ x = 0\}$
<proof>

definition *roots4* :: *complex poly* \Rightarrow *complex list* **where**
roots4 = *roots4-generic solve-depressed-quartic-complex*

lemma *roots4*: **assumes** *deg*: degree $p = 4$
shows $\text{set } (\text{roots4 } p) = \{x. \text{poly } p\ x = 0\}$
<proof>

definition *rroots4* :: *real poly* \Rightarrow *real list* **where**
rroots4 = *roots4-generic solve-depressed-quartic-real*

lemma *rroots4*: **assumes** *deg*: degree $p = 4$
shows $\text{set } (\text{rroots4 } p) = \{x. \text{poly } p\ x = 0\}$
<proof>

end

References

- [1] G. Cardano. *Ars Magna, The Great Art or the Rules of Algebra*. 1545. [https://en.wikipedia.org/wiki/Ars_Magna_\(Cardano_book\)](https://en.wikipedia.org/wiki/Ars_Magna_(Cardano_book)).
- [2] R. Thiemann, A. Yamada, and S. Joosten. Algebraic numbers in Isabelle/HOL. *Archive of Formal Proofs*, Dec. 2015. https://isa-afp.org/entries/Algebraic_Numbers.html, Formal proof development.