Solving Cubic and Quartic Equations*

René Thiemann

March 19, 2025

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

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

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** $a4 :: 'a :: field\text{-}char\text{-}0$
assumes $a4$: $a4 \neq 0$
and b: $b = a3\ /\ a4$
and c: $c = a2\ /\ a4$
and d: $d = a1\ /\ a4$
and e: $e = a0\ /\ a4$
and p: $p = c - (3/8) * b\hat{}2$
and q: $q = (b\hat{}3 - 4*b*c + 8 * d)\ /\ 8$
and r: $r = (\ -3 * b\hat{}4 + 256 * e - 64 * b * d + 16 * b\hat{}2 * c)\ /\ 256$
and x: $x = y - b/4$
shows $a4 * x\hat{}4 + a3 * x\hat{}3 + a2 * x\hat{}2 + a1 * x + a0 = 0$
$\longleftrightarrow y\hat{}4 + p * y\hat{}2 + q * y + r = 0$
⟨*proof*⟩

lemma *biquadratic-solution*: **fixes** $p\ q :: 'a :: field\text{-}char\text{-}0$
shows $y\hat{}4 + p * y\hat{}2 + q = 0 \longleftrightarrow (\exists\ z.\ z\hat{}2 + p * z + q = 0 \wedge z = y\hat{}2)$
⟨*proof*⟩

1.2 Solving the depressed case via Ferrari's formula

lemma *depressed-quartic-Ferrari*: **fixes** $p\ q\ r :: 'a :: field\text{-}char\text{-}0$
assumes *cubic-root*: $8*m\hat{}3 + (8 * p) * m\hat{}2 + (2 * p\hat{}2 - 8 * r) * m - q\hat{}2 = 0$
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 $y\hat{}4 + p * y\hat{}2 + q * y + r = 0 \longleftrightarrow poly\ [:b1,a,1:]\ y = 0 \vee poly\ [:b2,-a,1:]\ y = 0$
⟨*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 :: $'a$ :: *field-char-0*
assumes a: $a \neq 0$
and xy: $x = y - b / (3 * a)$
and e: $e = (c - b\hat{}2 / (3 * a)) / a$
and f: $f = (d + 2 * b\hat{}3 / (27 * a\hat{}2) - b * c / (3 * a)) / a$
shows $(a * x \hat{} 3 + b * x^2 + c * x + d = 0) \longleftrightarrow y\hat{}3 + e * y + f = 0$
⟨*proof*⟩

2.2 Solving the depressed case in arbitrary fields

lemma *cubic-depressed*: **fixes** e :: $'a$ :: *field-char-0*
assumes yz: $e \neq 0 \Longrightarrow z\hat{}2 - y * z - e / 3 = 0$
and u: $e \neq 0 \Longrightarrow u = z\hat{}3$
and v: $v = - (e \hat{} 3 / 27)$
shows $y\hat{}3 + e * y + f = 0 \longleftrightarrow (\textit{if } e = 0 \textit{ then } y\hat{}3 = -f \textit{ else } u^2 + f * u + v = 0)$
⟨*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 :: *complex*
assumes $e0$: $e \neq 0$
and v: $v = - (e \hat{} 3 / 27)$
and u: $u\hat{}2 + f * u + v = 0$
shows $y\hat{}3 + e * y + f = 0 \longleftrightarrow (\exists\ z.\ z\hat{}3 = u \wedge y = z - e / (3 * z))$
⟨*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* ∗ *e*^*3* + *27* ∗ *f*^*2*)

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* ≡ *root 3 u* − *e* / (*3* ∗ *root 3 u*)
and Δ ≡ *discriminant-cubic-depressed e f*
assumes *e0*: *e* ≠ *0*
and *v*: *v* = − (*e* ^ *3* / *27*)
and *u*: u^2 + *f* ∗ *u* + *v* = *0*
shows *y1*^*3* + *e* ∗ *y1* + *f* = *0*
Δ ≠ *0* ⟹ *y*^*3* + *e* ∗ *y* + *f* = *0* ⟹ *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 Δ ≡ *discriminant-cubic-depressed e f*
assumes *Delta*: Δ ≥ *0*
and *rt*: *y*^*3* + *e* ∗ *y* + *f* = *0*
shows *y* ∈ 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-croots* :: *nat* ⇒ *complex* ⇒ *complex list* **where**
all-croots 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-croots*: **assumes** *n0*: *n* ≠ *0* **shows** *set* (*all-croots 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-croots*.

lift-definition *all-croots-part1* :: *nat* ⇒ *complex* ⇒ *complex genuine-roots-aux* **is**
λ *n x*. *if n* = *0* ∨ *x* = *0* ∨ ¬ *algebraic x then* (*1*,[],*0*, *filter-fun-complex 1*)
else let p = *min-int-poly x*;
q = *poly-nth-root n p*;
zeros = *complex-roots-of-int-poly q*;
r = *Polynomial.monom 1 n* − [:*x*:]
in (*r*,*zeros*, *n*, *filter-fun-complex r*)
⟨*proof*⟩

lemma *all-croots-code*[*code*]:
all-croots n x = (*if n* = *0 then* [] *else if x* = *0 then* [*0*]
else if algebraic x then genuine-roots-impl (*all-croots-part1 n x*)
else Code.abort (*STR ''all-croots invoked on non−algebraic number''*) (λ -. *all-croots 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* :: *nat* ⇒ *complex* ⇒ *complex* **where**
croot n x = (*rcis* (*root n* (*cmod x*)) (*Arg x* / *of-nat n*))

lemma *croot-0*[*simp*]: *croot n 0* = *0 croot 0 x* = *0*
⟨*proof*⟩

lemma *croot-power*: **assumes** *n*: *n* ≠ *0*
shows (*croot n x*) ^ *n* = *x*
⟨*proof*⟩

lemma *Arg-of-real*: *Arg* (*of-real x*) =
(*if x* < *0 then pi else 0*)
⟨*proof*⟩

lemma *Arg-rcis-cis*[*simp*]: **assumes** $x > 0$
shows $Arg\ (rcis\ x\ y) = Arg\ (cis\ y)$
⟨*proof*⟩

lemma *cis-Arg-1*[*simp*]: $cis\ (Arg\ 1) = 1$
⟨*proof*⟩

lemma *cis-Arg-power*[*simp*]: **assumes** $x \neq 0$
shows $cis\ (Arg\ (x\ \widehat{}\ n)) = cis\ (Arg\ x * real\ n)$
⟨*proof*⟩

lemma *Arg-croot*[*simp*]: $Arg\ (croot\ n\ x) = Arg\ x\ /\ real\ n$
⟨*proof*⟩

lemma *cos-abs*[*simp*]: $cos\ (abs\ x :: real) = cos\ x$
⟨*proof*⟩

lemma *cos-mono-le*: **assumes** $abs\ x \leq pi$
and $abs\ y \leq pi$
shows $cos\ x \leq cos\ y \longleftrightarrow abs\ y \leq abs\ x$
⟨*proof*⟩

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

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

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*: $y\ \widehat{}\ n = x$
and *neq*: $y \neq croot\ n\ x$
shows $Re\ y < Re\ (croot\ n\ x) \vee Re\ y = Re\ (croot\ n\ x) \wedge Im\ y < Im\ (croot\ n\ x)$
⟨*proof*⟩

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

shows *croot n x = y*
⟨*proof*⟩

lemma *csqrt-is-croot-2*: *csqrt = croot 2*
⟨*proof*⟩

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

lemma *croot-impl*[*code*]: *croot n x = (if n = 0 then 0 else*
  arg-min-list (λ y. (− Re y, − Im y)) (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* :: *real ⇒ real ⇒ real option* **where**
  solve-depressed-cubic-Cardano-real e f = (
    if e = 0 then Some (root 3 (−f)) else
     let v = − (e ^ 3 / 27) in
     case rroots2 [:*v*,*f*,*1*:] *of*
       [*u*,-] ⇒ *let rt = root 3 u in Some (rt − e / (3 ∗ rt))*
     | - ⇒ *None*)

lemma *solve-depressed-cubic-Cardano-real*:
  assumes *solve-depressed-cubic-Cardano-real e f = Some y*
  shows {*y. y^3 + e ∗ y + f = 0*} = {*y*}
⟨*proof*⟩

The complex case

definition *solve-depressed-cubic-complex* :: *complex ⇒ complex ⇒ complex list*
where
  solve-depressed-cubic-complex e f = (*let*
        ys = (*if e = 0 then all-croots 3 (− f) else (let*
      u = hd (croots2 [: − (*e ^ 3 / 27*) ,*f*,*1*:]);
      zs = 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-croots 3 (− f) else (let
        f2 = f / 2;
        u = − f2 + csqrt (f2^2 + e ^ 3 / 27);
        zs = all-croots 3 u
        in map (λ z. z − e / (3 ∗ z)) zs))
  in remdups ys)
⟨proof⟩
```

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

$\longleftrightarrow (y\hat{}3 + e * y + f = 0)$
⟨*proof*⟩

For the general real case, we first try Cardano with negative discrimiant and only if it is not applicable, then we go for the calculation using complex numbers. Note that for 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* :: *real* ⇒ *real* ⇒ *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-croots 3 u))))
⟨proof⟩
```

lemma *solve-depressed-cubic-real*: $y \in set$ (*solve-depressed-cubic-real e f*)
$\longleftrightarrow (y\hat{}3 + e * y + f = 0)$
⟨*proof*⟩

Combining the various algorithms

lemma *degree3-coeffs*: *degree p* = *3* ⟹
  ∃ *a b c d*. *p* = [: *d*, *c*, *b*, *a* :] ∧ *a* ≠ *0*
  ⟨*proof*⟩

definition *roots3-generic* :: (*'a* :: *field-char-0* ⇒ *'a* ⇒ *'a list*) ⇒ *'a poly* ⇒ *'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 (λ *y*. *y* − *ba3*) *roots*)

lemma *roots3-generic*: **assumes** *deg*: *degree p* = *3*
  and *solver*: ⋀ *e f y*. *y* ∈ *set* (*depressed-solver e f*) ⟷ *y*^*3* + *e* ∗ *y* + *f* = *0*
  shows *set* (*roots3-generic depressed-solver p*) = {*x*. *poly p x* = *0*}
⟨*proof*⟩

definition *croots3* :: *complex poly* ⇒ *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* ⇒ *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* ⇒ *complex* ⇒ *complex* ⇒ *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 [: − (q^2), 2 * p^2 − 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* ∈ *set* (*solve-depressed-quartic-complex p q r*)
⟷ (*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* ⇒ *real* ⇒ *real* ⇒ *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 [: − (q^2), 2 * p^2 − 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* ∈ *set* (*solve-depressed-quartic-real p q r*)
⟷ (*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* ⟹
∃ *a b c d e. p = [: e, d, c, b, a :]* ∧ *a* ≠ *0*
⟨*proof*⟩

definition *roots4-generic* :: (*'a* :: *field-char-0* ⇒ *'a* ⇒ *'a* ⇒ *'a list*) ⇒ *'a poly* ⇒ *'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*: ⋀ *p q r y*. *y* ∈ *set* (*depressed-solver p q r*) ⟷ *y*^*4* + *p* ∗ *y*^*2* + *q* ∗ *y* + *r* = *0*
shows *set* (*roots4-generic depressed-solver p*) = {*x*. *poly p x* = *0*}
⟨*proof*⟩

definition *croots4* :: *complex poly* ⇒ *complex list* **where**
croots4 = *roots4-generic solve-depressed-quartic-complex*

lemma *croots4*: **assumes** *deg*: *degree p* = *4*
shows *set* (*croots4 p*) = { *x*. *poly p x* = *0*}
⟨*proof*⟩

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

lemma *rroots4*: **assumes** *deg*: *degree p* = *4*
shows *set* (*rroots4 p*) = { *x*. *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).

[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.