Napoleon's Theorem in Isabelle/HOL

Arthur Freitas Ramos David Barros Hulak
Ruy J. G. B. de Queiroz

September 18, 2026

Abstract

We formalize Napoleon's theorem in Isabelle/HOL. Given a non-degenerate Euclidean triangle, equilateral triangles constructed externally on its three sides have centres that form an equilateral triangle. The construction is represented in complex coordinates, and both orientations of the equilateral rotation are covered by one parametrized development.

Contents

1 Introduction

Napoleon's theorem is a classical triangle-geometry result [1]: the three equilateral triangles erected on the sides of an arbitrary triangle determine three centres which themselves are the vertices of an equilateral triangle. The orientation of the erected triangles matters. A coordinate formalization must therefore distinguish the two rotations through $\pm\pi/3$, and it must make explicit which orientation gives the external construction.

The exterior condition is part of the formal specification. For a directed side ab, the signed side orientation of a point x is $\mathrm{Im}((x-a)/(b-a))$. An apex is external precisely when its side orientation has strictly opposite sign to that of the third vertex.

The formalization uses the complex plane. If s is either $\mathrm{cis}(\pi/3)$ or $\mathrm{cis}(-\pi/3)$, the apex on the directed side from x to y is

$$E_s(x, y) = x + s(y - x).$$

The centre of the corresponding equilateral triangle is the average of its three vertices. The published AFP development of Morley's theorem [2] supplies the equilateral-triangle characterization used by the present entry; the present theory adds the cyclic centre calculation specific to Napoleon's configuration.

2 Definitions and construction correctness

The theory defines the oriented apex and the centre of an equilateral triangle on a directed side. For distinct endpoints, the rotation characterization immediately proves the two equal-length equations that certify the construction as equilateral. Keeping the rotation parameter explicit avoids duplicating the proof for clockwise and counterclockwise constructions. The predicate `external_side` states that two points lie strictly on opposite sides of the directed line. The theorem `napoleon_external_construction` proves this predicate for all three cyclic sides, in addition to the six equal-distance facts for the constructed equilateral triangles.

3 The centre calculation

Let N_{AB}, N_{BC}, and N_{CA} denote the three centres. The complex rotation parameter satisfies

$$s^2 = s - 1,$$

and the cyclic differences simplify to

$$N_{CA} - N_{AB} = (1 - s)(N_{BC} - N_{AB}).$$

Since $1 - s$ is the opposite $\pm\pi/3$ rotation, the two vectors from N_{AB} to the other two centres have equal lengths and meet at an equilateral angle. The final theorem is stated through the corresponding pairwise-distance characterization. The final bundled corollary `napoleon_external_theorem` packages this centre conclusion together with all three external equilateral constructions under the non-collinearity assumption, so the formal statement mirrors the complete English theorem.

4 Formalization notes

The entry imports the released `Morley_Theorem` session rather than duplicating its complex equilateral infrastructure. The `napoleon_centre` is the arithmetic mean of the three vertices, hence the centroid; on an equilateral triangle this agrees with the circumcenter, incenter, and orthocenter. The source theory is kept as a single top-level AFP theory, with the document containing the mathematical explanation and proof sketch.

```
theory Napoleon
  imports
    HOL−Decision-Procs.Commutative-Ring
    Morley-Theorem.Complex-Triangles
begin
```

5 Napoleon's construction

A complex number *s* equal to *cis* (*pi*/*3*) or *cis* (−*pi*/*3*) rotates a directed side through one of the two equilateral-triangle angles. The apex and the centre of the resulting equilateral triangle are defined parametrically so that the same algebraic lemmas cover both orientations.

```
definition napoleon-apex :: complex ⇒ complex ⇒ complex ⇒ complex
  where napoleon-apex s x y = x + s ∗ (y − x)

definition napoleon-centre :: complex ⇒ complex ⇒ complex ⇒ complex
  where napoleon-centre s x y =
    (x + y + napoleon-apex s x y) / 3

lemma napoleon-apex-displacement:
  napoleon-apex s x y − x = s ∗ (y − x)
  by (simp add: napoleon-apex-def)

lemma equilateral-on-side:
  fixes x y s :: complex
  assumes x ≠ y
    and s = cis (pi / 3) ∨ s = cis (−pi / 3)
  shows
    cdist y x = cdist (napoleon-apex s x y) x ∧
     cdist (napoleon-apex s x y) x = cdist (napoleon-apex s x y) y
proof −
  have hyx: y ≠ x using assms(1) by auto
  have hrot:
    napoleon-apex s x y − x = cis (pi / 3) ∗ (y − x) ∨
       napoleon-apex s x y − x = cis (−pi / 3) ∗ (y − x)
    using assms(2) by (auto simp add: napoleon-apex-def)
  have h′ :
```

```
    cdist y x = cdist (napoleon-apex s x y) x ∧
     cdist (napoleon-apex s x y) x = cdist (napoleon-apex s x y) y
   using equilateral-caracterization[
     THEN iffD2, of y x napoleon-apex s x y, OF hyx] hrot by blast
  show ?thesis using h' by (simp add: cdist-commute)
qed
```

6 Algebra of the three centres

```
lemma cis-pi3-square:
  (cis (pi / 3) :: complex) ^ 2 = cis (pi / 3) − 1
  by (auto intro: complex-eqI
     simp: cis.code cos-60 sin-60 power2-eq-square algebra-simps)

lemma cis-minus-pi3-square:
  (cis (−pi / 3) :: complex) ^ 2 = cis (−pi / 3) − 1
  by (auto intro: complex-eqI
     simp: cis.code cos-60 sin-60 power2-eq-square algebra-simps)

lemma one-minus-cis-pi3:
  (1 :: complex) − cis (pi / 3) = cis (−pi / 3)
  by (auto intro: complex-eqI
     simp: cis.code cos-60 sin-60 algebra-simps)

lemma one-minus-cis-minus-pi3:
  (1 :: complex) − cis (−pi / 3) = cis (pi / 3)
  by (auto intro: complex-eqI
     simp: cis.code cos-60 sin-60 algebra-simps)

lemma napoleon-centre-cycle:
  fixes a b c s :: complex
  assumes s ^ 2 = s − 1
  shows
    napoleon-centre s c a − napoleon-centre s a b =
      (1 − s) ∗ (napoleon-centre s b c − napoleon-centre s a b)
proof −
  have hsquare': s ∗ s = s − 1
    using assms by (simp add: power2-eq-square)
  have hmult: s ∗ (s ∗ z) = (s − 1) ∗ z for z :: complex
    by (simp add: hsquare' mult.assoc[symmetric] algebra-simps)
  from hsquare'
  show napoleon-centre s c a − napoleon-centre s a b =
     (1 − s) ∗ (napoleon-centre s b c − napoleon-centre s a b)
    unfolding napoleon-centre-def napoleon-apex-def
    by (simp add: hmult algebra-simps field-simps)
qed

lemma napoleon-centres-equilateral:
  fixes a b c s :: complex
```

```
  assumes hsquare: s ^ 2 = s − 1
    and hroot: 1 − s = cis (pi / 3) ∨ 1 − s = cis (−pi / 3)
  shows
    cdist (napoleon-centre s b c) (napoleon-centre s a b) =
       cdist (napoleon-centre s c a) (napoleon-centre s a b) ∧
     cdist (napoleon-centre s c a) (napoleon-centre s a b) =
       cdist (napoleon-centre s c a) (napoleon-centre s b c)
proof −
  have hdiff:
      napoleon-centre s c a − napoleon-centre s a b =
        (1 − s) ∗ (napoleon-centre s b c − napoleon-centre s a b)
    by (rule napoleon-centre-cycle[OF hsquare])
  show ?thesis
  proof (cases napoleon-centre s b c = napoleon-centre s a b)
    case True
    have hca:
        napoleon-centre s c a = napoleon-centre s a b
      using hdiff True by simp
    show ?thesis by (simp add: True hca)
  next
    case False
    have hrot:
        napoleon-centre s c a − napoleon-centre s a b =
            cis (pi / 3) ∗ (napoleon-centre s b c − napoleon-centre s a b) ∨
         napoleon-centre s c a − napoleon-centre s a b =
            cis (−pi / 3) ∗ (napoleon-centre s b c − napoleon-centre s a b)
      using hdiff hroot by (auto simp add: hdiff)
    have h′ :
        cdist (napoleon-centre s b c) (napoleon-centre s a b) =
           cdist (napoleon-centre s c a) (napoleon-centre s a b) ∧
         cdist (napoleon-centre s c a) (napoleon-centre s a b) =
           cdist (napoleon-centre s c a) (napoleon-centre s b c)
      using equilateral-caracterization[
        THEN iffD2, of napoleon-centre s b c napoleon-centre s a b
           napoleon-centre s c a, OF False] hrot by blast
    show ?thesis by (rule h′)
  qed
qed
```

7 External orientation and Napoleon's theorem

The sign of the oriented area of the triangle is the sign of the imaginary part of the quotient below. Choosing the opposite rotation makes the equilateral triangle on the directed side *a b* lie on the exterior side; the same choice works cyclically on *b c* and *c a*. This exteriority claim is formalized by *external-side*: the signed side orientations of the third vertex and the constructed apex have strictly negative product.

definition *napoleon-external-rotation* :: *complex* ⇒ *complex* ⇒ *complex* ⇒ *com-*

```
plex
  where
    napoleon-external-rotation a b c =
      (if 0 < Im ((c − a) / (b − a))
       then cis (−pi / 3)
       else cis (pi / 3))

definition side-orientation :: complex ⇒ complex ⇒ complex ⇒ real
  where side-orientation a b x = Im ((x − a) / (b − a))

definition external-side :: complex ⇒ complex ⇒ complex ⇒ complex ⇒ bool
  where
    external-side a b c x ⟷
       side-orientation a b c ∗ side-orientation a b x < 0

lemma napoleon-external-rotation-cases:
  napoleon-external-rotation a b c = cis (pi / 3) ∨
   napoleon-external-rotation a b c = cis (−pi / 3)
  unfolding napoleon-external-rotation-def
  by (split if-split; simp)

lemma napoleon-external-rotation-square:
  napoleon-external-rotation a b c ^ 2 =
     napoleon-external-rotation a b c − 1
proof (cases 0 < Im ((c − a) / (b − a)))
  case True
  show ?thesis
    unfolding napoleon-external-rotation-def
    using cis-minus-pi3-square True by (simp add: True)
next
  case False
  show ?thesis
    unfolding napoleon-external-rotation-def
    using cis-pi3-square False by (simp add: False)
qed

lemma napoleon-external-rotation-one-minus:
  1 − napoleon-external-rotation a b c = cis (pi / 3) ∨
   1 − napoleon-external-rotation a b c = cis (−pi / 3)
proof (cases 0 < Im ((c − a) / (b − a)))
  case True
  show ?thesis
    unfolding napoleon-external-rotation-def
    using one-minus-cis-minus-pi3 True by (simp add: True)
next
  case False
  show ?thesis
    unfolding napoleon-external-rotation-def
    using one-minus-cis-pi3 False by (simp add: False)
```

qed

lemma *oriented-area-cycle*:
  Im $((c - a) * cnj\ (b - a)) =$
    Im $((a - b) * cnj\ (c - b))$
  by (*simp add*: *algebra-simps*)

lemma *napoleon-external-rotation-cyclic*:
  napoleon-external-rotation $a\ b\ c =$
    napoleon-external-rotation $b\ c\ a$
proof −
  have *hleft*:
    $0 < Im\ ((c - a) / (b - a)) \longleftrightarrow$
     $0 < Im\ ((c - a) * cnj\ (b - a))$
    using *Im-complex-div-gt-0*[*of* $(c - a)\ (b - a)$] **by** *simp*
  have *hright*:
    $0 < Im\ ((a - b) / (c - b)) \longleftrightarrow$
     $0 < Im\ ((a - b) * cnj\ (c - b))$
    using *Im-complex-div-gt-0*[*of* $(a - b)\ (c - b)$] **by** *simp*
  have *harea*:
    Im $((c - a) * cnj\ (b - a)) =$
     Im $((a - b) * cnj\ (c - b))$
    by (*rule oriented-area-cycle*)
  have *hpos*:
    $0 < Im\ ((c - a) / (b - a)) \longleftrightarrow$
     $0 < Im\ ((a - b) / (c - b))$
    using *hleft hright harea* **by** (*simp add*: *harea*)
  show *?thesis*
    unfolding *napoleon-external-rotation-def*
    using *hpos* **by** *simp*
qed

lemma *noncollinear-distinct*:
  assumes $\neg$ *collinear* $a\ b\ c$
  shows $a \neq b \land b \neq c \land c \neq a$
proof −
  have *hab*: $a \neq b$
    using *assms* **unfolding** *collinear-def* **by** *auto*
  have *hbc*: $b \neq c$
  proof
    assume $b = c$
    with *hab* **have** *collinear* $a\ b\ c$
      by (*simp add*: *collinear-def*)
    with *assms* **show** *False* **by** *contradiction*
  qed
  have *hca*: $c \neq a$
  proof
    assume $c = a$
    with *hab* **have** *collinear* $a\ b\ c$

```
    by (simp add: collinear-def)
   with assms show False by contradiction
  qed
  show ?thesis by (intro conjI; fact)
qed

lemma side-orientation-nonzero:
  assumes hncol: ¬ collinear a b c
  shows side-orientation a b c ≠ 0
  using hncol unfolding side-orientation-def collinear-def by auto

lemma side-orientation-apex:
  fixes a b s :: complex
  assumes a ≠ b
  shows
   side-orientation a b (napoleon-apex s a b) = Im s
proof −
  have hba: b − a ≠ (0 :: complex) using assms by auto
  have hratio:
    (napoleon-apex s a b − a) / (b − a) = s
   unfolding napoleon-apex-def
   using hba by (simp add: field-simps)
  show ?thesis
   unfolding side-orientation-def
   using hratio by simp
qed

lemma Im-cis-pi3-pos:
  0 < Im (cis (pi / 3) :: complex)
proof −
  have hroot: 0 < sqrt (3 :: real)
   by (rule real-sqrt-gt-zero) simp
  show ?thesis by (simp add: cis.code sin-60 hroot)
qed

lemma Im-cis-minus-pi3-neg:
  Im (cis (−pi / 3) :: complex) < 0
proof −
  have hroot: 0 < sqrt (3 :: real)
   by (rule real-sqrt-gt-zero) simp
  show ?thesis by (simp add: cis.code sin-60 hroot)
qed

lemma napoleon-external-side:
  fixes a b c :: complex
  assumes hncol: ¬ collinear a b c
  shows
   external-side a b c
     (napoleon-apex (napoleon-external-rotation a b c) a b)
```

```
proof –
  obtain hab hbc hca where a ≠ b b ≠ c c ≠ a
    using noncollinear-distinct[OF hncol] by blast
  have hside: side-orientation a b c ≠ 0
    by (rule side-orientation-nonzero[OF hncol])
  show ?thesis
    unfolding external-side-def
  proof (cases 0 < side-orientation a b c)
    case True
    have hcond: 0 < Im ((c − a) / (b − a))
      using True by (simp add: side-orientation-def)
    have hrot:
        napoleon-external-rotation a b c = cis (−pi / 3)
      unfolding napoleon-external-rotation-def side-orientation-def
      using hcond by simp
    have hapex:
        side-orientation a b
          (napoleon-apex (napoleon-external-rotation a b c) a b) < 0
      using side-orientation-apex[OF ‹a ≠ b›]
        Im-cis-minus-pi3-neg hrot by simp
    show side-orientation a b c ∗
        side-orientation a b
         (napoleon-apex (napoleon-external-rotation a b c) a b) < 0
      by (rule mult-pos-neg[OF True hapex])
  next
    case False
    have hside-neg: side-orientation a b c < 0
      using hside False by linarith
    have hcond: ¬ 0 < Im ((c − a) / (b − a))
      using False by (simp add: side-orientation-def)
    have hrot:
        napoleon-external-rotation a b c = cis (pi / 3)
      unfolding napoleon-external-rotation-def side-orientation-def
      using hcond by simp
    have hapex:
        0 < side-orientation a b
          (napoleon-apex (napoleon-external-rotation a b c) a b)
      using side-orientation-apex[OF ‹a ≠ b›]
        Im-cis-pi3-pos hrot by simp
    show side-orientation a b c ∗
        side-orientation a b
         (napoleon-apex (napoleon-external-rotation a b c) a b) < 0
      by (rule mult-neg-pos[OF hside-neg hapex])
  qed
qed

lemma napoleon-external-construction:
  assumes hncol: ¬ collinear a b c
  shows
```

```
    cdist b a =
      cdist (napoleon-apex (napoleon-external-rotation a b c) a b) a ∧
    cdist (napoleon-apex (napoleon-external-rotation a b c) a b) a =
      cdist (napoleon-apex (napoleon-external-rotation a b c) a b) b ∧
    cdist c b =
      cdist (napoleon-apex (napoleon-external-rotation a b c) b c) b ∧
    cdist (napoleon-apex (napoleon-external-rotation a b c) b c) b =
      cdist (napoleon-apex (napoleon-external-rotation a b c) b c) c ∧
    cdist a c =
      cdist (napoleon-apex (napoleon-external-rotation a b c) c a) c ∧
    cdist (napoleon-apex (napoleon-external-rotation a b c) c a) c =
      cdist (napoleon-apex (napoleon-external-rotation a b c) c a) a ∧
    external-side a b c
      (napoleon-apex (napoleon-external-rotation a b c) a b) ∧
    external-side b c a
      (napoleon-apex (napoleon-external-rotation a b c) b c) ∧
    external-side c a b
      (napoleon-apex (napoleon-external-rotation a b c) c a)
proof −
  obtain hab hbc hca where a ≠ b b ≠ c c ≠ a
    using noncollinear-distinct[OF hncol] by blast
  have hs:
     napoleon-external-rotation a b c = cis (pi / 3) ∨
      napoleon-external-rotation a b c = cis (−pi / 3)
    by (rule napoleon-external-rotation-cases)
  have hab′:
     cdist b a =
        cdist (napoleon-apex (napoleon-external-rotation a b c) a b) a ∧
      cdist (napoleon-apex (napoleon-external-rotation a b c) a b) a =
        cdist (napoleon-apex (napoleon-external-rotation a b c) a b) b
    using equilateral-on-side[of a b napoleon-external-rotation a b c]
     ‹a ≠ b› hs by blast
  have hbc′:
     cdist c b =
        cdist (napoleon-apex (napoleon-external-rotation a b c) b c) b ∧
      cdist (napoleon-apex (napoleon-external-rotation a b c) b c) b =
        cdist (napoleon-apex (napoleon-external-rotation a b c) b c) c
    using equilateral-on-side[of b c napoleon-external-rotation a b c]
     ‹b ≠ c› hs by blast
  have hca′:
     cdist a c =
        cdist (napoleon-apex (napoleon-external-rotation a b c) c a) c ∧
      cdist (napoleon-apex (napoleon-external-rotation a b c) c a) c =
        cdist (napoleon-apex (napoleon-external-rotation a b c) c a) a
    using equilateral-on-side[of c a napoleon-external-rotation a b c]
     ‹c ≠ a› hs by blast
  have hside-ab:
     external-side a b c
        (napoleon-apex (napoleon-external-rotation a b c) a b)
```

```
    by (rule napoleon-external-side[OF hncol])
  have hncol-bca: ¬ collinear b c a
    using hncol by (metis collinear-sym1 collinear-sym2)
  have hncol-cab: ¬ collinear c a b
    using hncol by (metis collinear-sym1 collinear-sym2)
  have hrot-bc:
      napoleon-external-rotation a b c =
       napoleon-external-rotation b c a
    by (rule napoleon-external-rotation-cyclic)
  have hrot-ca:
      napoleon-external-rotation a b c =
       napoleon-external-rotation c a b
    using napoleon-external-rotation-cyclic[of a b c]
      napoleon-external-rotation-cyclic[of b c a] by simp
  have hside-bc0:
      external-side b c a
         (napoleon-apex (napoleon-external-rotation b c a) b c)
    by (rule napoleon-external-side[OF hncol-bca])
  have hside-bc:
      external-side b c a
         (napoleon-apex (napoleon-external-rotation a b c) b c)
    using hside-bc0 hrot-bc by (simp add: hrot-bc)
  have hside-ca0:
      external-side c a b
         (napoleon-apex (napoleon-external-rotation c a b) c a)
    by (rule napoleon-external-side[OF hncol-cab])
  have hside-ca:
      external-side c a b
         (napoleon-apex (napoleon-external-rotation a b c) c a)
    using hside-ca0 hrot-ca by (simp add: hrot-ca)
  show ?thesis using hab' hbc' hca' hside-ab hside-bc hside-ca by blast
qed

theorem napoleon-theorem:
  fixes a b c :: complex
  assumes hncol: ¬ collinear a b c
  shows
    cdist (napoleon-centre (napoleon-external-rotation a b c) b c)
       (napoleon-centre (napoleon-external-rotation a b c) a b) =
       cdist (napoleon-centre (napoleon-external-rotation a b c) c a)
         (napoleon-centre (napoleon-external-rotation a b c) a b) ∧
     cdist (napoleon-centre (napoleon-external-rotation a b c) c a)
       (napoleon-centre (napoleon-external-rotation a b c) a b) =
       cdist (napoleon-centre (napoleon-external-rotation a b c) c a)
         (napoleon-centre (napoleon-external-rotation a b c) b c)
  using napoleon-centres-equilateral[
    OF napoleon-external-rotation-square
       napoleon-external-rotation-one-minus]
  by blast
```

theorem *napoleon-external-theorem*:
  fixes *a b c* :: *complex*
  assumes *hncol*: ¬ *collinear a b c*
  shows
    (*cdist b a* =
      cdist (*napoleon-apex* (*napoleon-external-rotation a b c*) *a b*) *a* ∧
    cdist (*napoleon-apex* (*napoleon-external-rotation a b c*) *a b*) *a* =
      cdist (*napoleon-apex* (*napoleon-external-rotation a b c*) *a b*) *b* ∧
    cdist c b =
      cdist (*napoleon-apex* (*napoleon-external-rotation a b c*) *b c*) *b* ∧
    cdist (*napoleon-apex* (*napoleon-external-rotation a b c*) *b c*) *b* =
      cdist (*napoleon-apex* (*napoleon-external-rotation a b c*) *b c*) *c* ∧
    cdist a c =
      cdist (*napoleon-apex* (*napoleon-external-rotation a b c*) *c a*) *c* ∧
    cdist (*napoleon-apex* (*napoleon-external-rotation a b c*) *c a*) *c* =
      cdist (*napoleon-apex* (*napoleon-external-rotation a b c*) *c a*) *a* ∧
    external-side a b c
      (*napoleon-apex* (*napoleon-external-rotation a b c*) *a b*) ∧
    external-side b c a
      (*napoleon-apex* (*napoleon-external-rotation a b c*) *b c*) ∧
    external-side c a b
      (*napoleon-apex* (*napoleon-external-rotation a b c*) *c a*)) ∧
    (*cdist* (*napoleon-centre* (*napoleon-external-rotation a b c*) *b c*)
      (*napoleon-centre* (*napoleon-external-rotation a b c*) *a b*) =
      cdist (*napoleon-centre* (*napoleon-external-rotation a b c*) *c a*)
        (*napoleon-centre* (*napoleon-external-rotation a b c*) *a b*) ∧
     cdist (*napoleon-centre* (*napoleon-external-rotation a b c*) *c a*)
      (*napoleon-centre* (*napoleon-external-rotation a b c*) *a b*) =
      cdist (*napoleon-centre* (*napoleon-external-rotation a b c*) *c a*)
        (*napoleon-centre* (*napoleon-external-rotation a b c*) *b c*))
  using *napoleon-external-construction*[*OF hncol*]
    napoleon-theorem[*OF hncol*]
  by *blast*

end

References

[1] R. Honsberger. *Episodes in Nineteenth and Twentieth Century Euclidean Geometry*, volume 37 of *New Mathematical Library*. Mathematical Association of America, 1995.

[2] B. Puyobro. Morley's theorem. Archive of Formal Proofs. Available at https://isa-afp.org/entries/Morley_Theorem.html.