A Sage Companion to A. Reventós' "Affine Maps, Euclidean Motions and Quadrics"

Jaume Aguadé

1 introduction

This is the user manual for version 1.1 of a script[1] which defines a set of Sage ([3]) functions for solving all computational exercises in the book [5]. In these notes, we refer to [5] simply as "the book".We assume that the reader is familiar with the use of Sage in linear algebra as described in [1].

The author would like to mention that this script is meant as a help in teaching and learning the basics of affine geometry as explained in the book and so, even if the script may be a useful tool in research, this is not its primary scope.

2 Affine space

2.1 The base field

In principle, the base field may be any of the fields that Sage understands. The fields which are relevant to solve the exercises in the book are $\mathbb{Q}$, $GF(q)$ and the field of real algebraic numbers

```
sage:K=QQ
sage:L.<a>=FiniteField(q)
sage:R=AA
```

Actually, the computations in the book are supposed to be done in the real field. Fortunately, Sage can deal with the field of real algebraic numbers $\overline{\mathbb{Q}} \cap \mathbb{R}$ (denoted as `AA` in Sage) and, as we will discuss later, this is the appropriate field to solve the exercises in the book that go beyond the rationals.[2] In particular, if we need to deal with eigenvalues (for instance, in computing invariant lines or normal forms of an affinity), we should

[1] It may be noticed that this script has a strong Magma ([4]) flavor which is due to the fact that is comes from a preliminary version which was written in Magma. Hopefully, a future version of this script will be more true to the spirit (and capabilities) of Sage.

[2] This is one of the two major reasons to prefer Sage over Magma here. The other one is that Sage is open source.

choose `AA` as our base field. Notice that if a is an element of `AA`, then Sage prints it as a decimal number ending with a question mark:

```
sage:P=matrix([[1,2],[1,1]])
sage:P.eigenvalues()
[-0.4142135623730951?, 2.414213562373095?]
```

To print a real algebraic number using roots (if possible) we provide the function `Symbolize(-)`. For instance:

```
sage:P=matrix([[1,2],[1,1]])
sage:P.eigenvalues()
[-0.4142135623730951?, 2.414213562373095?]
sage:Symbolize(P.eigenvalues(),size='list')
sage:[-sqrt(2) + 1, sqrt(2) + 1]
```

This function has four values for the optional argument 'size': 'element' (default), 'list', 'matrix' and 'polynomial' (polynomials must be of degree one).

Anytime, we can ask Sage to output the results of our computations in TEX style, using the commands `view(-)` and `latex(-)`. For instance,

```
sage:view([-sqrt(2) + 1, sqrt(2) + 1])
```

produces $-\sqrt{2}+1 \quad \sqrt{2}+1$, while

```
sage:latex([-sqrt(2) + 1, sqrt(2) + 1])
```

produces TEX code for $-\sqrt{2}+1 \quad \sqrt{2}+1$.

In some cases when one or several indeterminate parameters appear we may use as base field sage's **symbolic ring** `SR`, but one has to notice that some functions (for instance `eigenspaces_left()`) are not available over the symbolic ring.

2.2 Points and vectors

The standard affine space of dimension d over a field K is defined by

```
sage:A,E=AffineSpace(K,d)
```

Here, A is the set of points and E is the K-vector space of dimension d acting on A. Actually, the two objects A and E are intrinsically the same, but we treat them as two separate entities, following the spirit of the book. The action of E on A is the standard one and other actions are not supported. This means that the above command defines the affine space $\mathbb{A}^d$.

Points of A and vectors of E are constructed giving their coordinates in the standard frame:

```
sage:P=A([0,-1,2])
sage:u=E([1,1,-1])
```

The base field and the dimension of the affine space can be recovered as

```
sage:K=E.base_field()
sage:d=E.dimension()
```

The action of the vector space E on the set A is given by addition while the vector $\overrightarrow{PQ}$ joining the points P and Q is computed by the function `Arrow`.

```
sage:Q=P+u
sage:PQ=Arrow(P,Q)
```

2.3 Affine varieties

An affine variety is determined by a point and a linear subspace of the vector space. For example

```
sage:X=Variety(P,E.subspace([u,v]))
```

defines the affine variety in A which contains the point P and has as direction the subspace of E generated by the vectors u and v.

The meaning of the following functions is straightforward

```
sage:BasePoint(X)
sage:Direction(X)
sage:AffineDimension(X)
sage:AffineCodimension(X)
sage:PointAsVariety(X)
sage:Belongs(P,X)
sage:Contained(X,Y)
sage:EqualityOfVarieties(X,Y)
sage:Parallel(X,Y)
```

Notice that a variety of dimension n is stored as a sequence consisting of a point and $n-1$ vectors which form a basis of the direction subspace of the variety.

The two main operations on varieties are the sum and the intersection. They can be computed as follows

```
sage:Z=AffineSum([X,Y,...])
sage:b,L=Meet(X,Y)
```

These functions are computed using propositions 1.11 and 1.13 in the book. The function `Meet(X,Y)` returns two values. The first one is a boolean value of `True` if the intersection of the two affine varieties is not empty and in this case the second value is the variety $X \cap Y$. If the two varieties do not meet, then the first value returns `False` and the second value is undefined.

2.4 Affine frames

An affine frame $\mathcal{R}$ consists of a point plus a basis of the vector space E. We can define a frame as follows

```
sage:Ref=Frame(P,B)
```

where $P \in A$ is a point and B is a basis of E. Given a frame, the coordinates of a point with respect to this frame are given by

```
sage:X=AffineCoordinates(P,Ref)
```

The function

```
sage:C=CanonicalFrame(A)
```

returns the canonical frame of the affine space A. There are functions to represent a frame by a matrix and to give the change of frame matrix according to section 1.10 in the book.

```
sage:M=FrameAsMatrix(Ref)
sage:T=ChangeOfFrameMatrix(Ref1,Ref2)
```

Given two frames `Ref1` and `Ref2` there are functions to perform change of coordinates from one frame to the other. The first one acts on points and the second one acts on affine varieties.

```
sage:Q=ChangeOfFramePoint(P,Ref1,Ref2)
sage:Y=ChangeOfFrameVariety(X,Ref)
```

2.5 Equations

If X is an affine variety in A, parametric equations for X can be read directly from `BasePoint(X)` and `Direction(X)`. To deal with Cartesian equations we first need to define a suitable polynomial ring containing the variables that we want to use in our equations. For example,

```
sage:Poly.<x,y,z,t>=PolynomialRing(A.base_field(),4)
```

Then, Cartesian equations for a variety X can be obtained in this way

```
sage:S=CartesianEquations(X,Poly)
```

Conversely, given a sequence $S = [f_1, \ldots, f_s]$ of polynomials of degree one, the affine subvariety of A that they define is

```
sage:V=VarietyByEquations(S,A)
```

2.6 Miscellaneous

If S is a sequence of points, then

`sage:Barycenter(S)`

returns the barycenter of the points in S. The function

`sage:Ratio(A,B,C)`

returns the simple ratio (A, B, C) of the points A, B, C which is the unique element in the base field such that $B = A + (A, B, C)\overrightarrow{AC}$ (see Definition 1.29 in the book). An error is raised if $A = C$ or the three points are not on a line.

2.7 Exercises

The file `Chapter_1` contains scripts to solve most computational exercises in Chapter 1 of the book. In particular, there are proofs of the theorems of Desargues and Pappus which deserve some explanation. As it is well known, Desargues theorem follows from the axioms of geometry in dimensions greater that two, while the two-dimensional proof requires the use of coordinates in a field. The proof of this theorem in projective geometry is short and elegant, but the affine proof, even in the generic case —when there are no points at infinity— is rather tedious and it makes sense to do it using sage. It goes as follows.

We start with the concurrent lines and two points in each line. We can take an affine frame such that the point where these three lines meet is the base point, and the lines are $x = 0$, $y = 0$ and $dx = cy$. Then, we consider the points $A_1 = (0, 1)$, $A_2 = (0, b)$, $B_1 = (1, 0)$, $B_2 = (a, 0)$, $C_1 = (c, d)$, $C_2 = (\lambda c, \lambda d)$. If we define

$$P = (B_1C_1) \cap (B_2C_2), \quad Q = (A_1C_1) \cap (A_2C_2), \quad R = (A_1B_1) \cap (A_2B_2)$$

then the theorem claims that the points P, Q, R lie on a line, for any choice of the parameters a, b, c, λ.

To prove this, we declare a, b, c, λ as variables and we consider the affine plane over the symbolic ring. Then, we check that the points P, Q and R generate a line. Then, a standard argument shows that for any value $a, b, c, \lambda \in \mathbb{R}$ such that the points P, Q, R exist in the affine plane over $\mathbb{R}$, they lie on a line.

3 Affinities

3.1 Definition of affinities and basic representations

An affinity $f : A \to A$ in an affine space is defined giving a point Q which is the image of the base point and a linear map $h : E \to E$:

`sage:f=Affinity(A,Q,h)`

Notice that only self-maps are supported. Given an affinity f and a point $P \in A$ we can compute its image simply as `f(P)`. There are several functions to deal with different ways to represent an affinity:

```
sage:AffinityAsMatrix(f)
sage:MatrixAsAffinity(M,A)
sage:EquationsOfAffinity(f,Poly)
sage:AffinityByEquations(S,A)
```

Here `S=[p1,p2,..]` is a sequence of polynomials of degree one and this function returns the affinity given by $x' = p_1(x, y, \ldots)$, $y' = p_2(x, y, \ldots)$, etc.

```
sage:LinearPartOfAffinity(f)
```

returns the linear homomorphism $h : E \to E$ associated to f.

```
sage:IsBijective(f)
sage:CompositionOfAffinities(f,g)
sage:EqualityOfAffinities(f,g)
```

Notice that a boolean like `f==g` does not work.

```
sage:InverseOfAffinity(f)
sage:ImageOfVariety(f,X)
```

This function returns the variety $f(X)$.

```
sage:Conjugate(f,g)
```

This function returns the affinity $g^{-1}fg$.[3]

3.2 Fixed points and invariant lines

Given an affinity $f : A \to A$ the set of fixed points of f is a subvariety of A. It can be computed by the function

```
sage:FixedPoints(f)
```

There is a function to decide if a variety is invariant under f or is not:

```
sage:IsInvariant(X,f)
```

To compute the invariant lines of an affinity f we should proceed in two steps. Let $h : E \to E$ be the linear map associated to f. Then, for any fixed point P of f and any eigenvector $v \in E$ of h, the line $P + \langle v \rangle$ is invariant. In this way we find all invariant lines which meet the variety of fixed points. We provide a function to determine the invariant lines that we get in this way, if there is a finite number of them:

```
sage:SparseInvariantLines(f)
```

On the other side, let us call *special invariant lines* those which do not contain any fixed point. To compute these special invariant lines we can use the function

```
sage:SpecialInvariantLines(f)
```

This function needs some explanation. It is easy to prove that the special invariant lines

[3] Notice that we follow the book in inverting the left map instead of the right one, as done in many other textbooks.

are of the form $P + \langle \overrightarrow{Pf(P)} \rangle$ where P is not a fixed point and $\overrightarrow{Pf(P)}$ is an eigenvector of h of eigenvalue 1. Then, the admissible values of P form a set S such that if we add to S the fixed points of f we obtain a variety X. The above function returns this variety X.

3.3 Some types of affinities

Following the book, we introduce functions to define affinities of some type like translations, homotheties, symmetries or projections:

`sage:AffineTranslation(v)`

is the translation associated to the vector $v \in E$.

`sage:AffineHomothecy(r,P)`

is the homothecy of similitude ratio $r \neq 0, 1$ and center equal to P.

`sage:Symmetry(X,G)`

is the symmetry which fixes the points in the variety $X = P + F$ and acts as -1 in the linear subspace $G \subset E$. An error is raised unless $E = F \oplus G$.

`sage:AffineProjection(X,G)`

is the projection which fixes the points in the variety $X = P + F$ and acts trivially in the linear subspace $G \subset E$. An error is raised unless $E = F \oplus G$. Beside these functions to construct various types of affinities, there are functions to recognize if an affinity is of one of these types:

```
sage:IsTranslation(f)
sage:IsHomothecy(f)
sage:IsSymmetry(f)
sage:IsAffineProjection(f)
```

In each case, if any of these functions returns `true` then it returns as second and third values the parameters needed to identify these affinities. For instance, in the case of a translation, it returns the translation vector; in the case of a homothecy it returns the ratio and the center; in the case of a symmetry it returns the variety of fixed points X and the -1 eigenspace, and in the case of a projection it returns the variety of fixed points X and the kernel.

For each of these types of affinities, there is a function to construct a random example:

```
sage:RandomTranslation(A)
sage:RandomHomothecy(A)
sage:RandomSymmetry(A,d)
sage:RandomAffineProjection(A,d)
```

Here d is the dimension of the variety of fixed points.

3.4 Exercises

The file `Chapter_2` contains scripts to solve most computational exercises in Chapter 2 of the book. These scripts serve as examples on the usage of the functions discussed in this section.

4 Classification of affinities

Theorem 4.9 in the book tells us that two affinities are similar (i.e. are related by a change of frame) if and only if their linear parts are similar (i.e. conjugated as linear maps) and they have the same *invariance level*. The invariance level ρ is defined as the minimum dimension of an invariant variety. This elegant theorem settles the classification problem for affinities from a theoretical point of view, but to implement this classification in an algorithm me need to reinterpret it and we also need to understand and reformulate its proof.

First of all, we notice that there is an alternate definition of the invariance level of an affinity which makes it easily computable.

Proposition 1 *Let f be a self-affinity of an affine space and let P be a point. Let $v = \overrightarrow{Pf(P)}$. Then, the invariance level $\rho(f)$ is the minimum integer r such that*

$$(\tilde{f} - I)^r(v) \in \mathrm{Im}(\tilde{f} - I)^{r+1}.$$

Proof Let Q be any other point and let $e = \overrightarrow{Qf(Q)}$. We have $e + \overrightarrow{PQ} = v + \tilde{f}(\overrightarrow{PQ})$. Hence

$$(\tilde{f} - I)^r(e) = (\tilde{f} - I)^r(v) + (\tilde{f} - I)^{r+1}(\overrightarrow{PQ})$$

and this proves that r does not depend on the choice of the point P. To compare r with $\rho(f)$, let P be the base point of a frame such that the matrix of f in this frame has the canonical form of theorem 4.16 in the book. If we use this point P to compute r and we use proposition 4.18 in the book, we see immediately that $r = \rho(f)$. □

Using this, we can compute $\rho(f)$ as

```
sage:InvarianceLevel(f)
```

This reduces the problem of deciding if two affinities are similar or not to the classical problem of deciding if two square matrices are similar or not. Fortunately, there are fast algorithms to construct the rational form of a given matrix and then two matrices are similar if and only if their rational forms coincide.[4] Hence, to decide if two affinities f and g are similar, we can use

[4] It seems, however, that sage's function `is_similar()` tries to compute the Jordan form of both matrices and so it fails if the eigenvalues are not in the base field and, even in this case, the algorithm is very slow.

```
sage:IsSimilar(f,g)
```

Moreover, affinities in dimensions 1 and 2 have special names as in chapter 3 of the book. These names can we recovered using

```
sage:Name(f)
```

Things get much more involved when, once we know that two affinities f and g are similar, we want to provide a third affinity h such that $g = h^{-1}fh$. This can be done using

```
sage:IsSimilar(f,g,transformation=True)
```

This function returns `False` or `True` and in this second case it returns also a suitable affinity h which conjugates f to g. It is worthwhile to discus how this affinity h is obtained, since this provides further insight on the classification theorem. It is clear that the only thing we need is a (computable) way to transform any affinity into its *normal form*

```
sage:NormalFormOfAffinity(f)
sage:NormalFormOfAffinity(f,transformation=True)
```

According to chapter 4 in the book, the normal form of f is a reference $\{O; e_1, \ldots, e_n\}$ such that

1. $f(O) = O + \epsilon e_1$ where $\epsilon = 0$ if $\rho(f) = 0$ and $\epsilon = 1$ if $\rho(f) > 0$.
2. $\mathcal{B} = \{e_1, \ldots, e_n\}$ is a *normal* basis of the underlying vector space, *adapted* to f. This means the following: $\mathcal{B}$ is any basis for any of the known normal forms for a square matrix. It may be the Jordan form (if it exists) or the rational form. Moreover, if $\rho(f) = r > 0$, then $e_1, \ldots, e_r$ form a Jordan basis for a Jordan block of eigenvalue equal to 1.[5]

In our case, the normal form of an endomorphism that we use is the following. We decompose the minimal polynomial in the real field (i.e. the field `AA` of real algebraic numbers) as a product $p(x)\,q(x)$ where $p(x)$ has only real roots and $q(x)$ is a product of quadratic polynomials without real roots. Then, we use the Jordan normal form for the kernel of $p(x)$ and the rational normal form for the kernel of $q(x)$.

Let us see how we proceed to find this adapted normal basis. If $\rho(f) = r$ we have that there is a vector v such that

$$(\overline{f} - I)^r(\overrightarrow{Of(O)}) = (\overline{f} - I)^{r+1}(v)$$

and r is minimal with this property. Let $f' = T^{-1}fT$ where T is the translation of vector $-v$. Then, f and f' have the same linear part and we consider the vector

$$e := \overrightarrow{Of'(O)} = \overrightarrow{Of(O)} - (\overline{f} - I)(v).$$

[5]Notice that we follow the book in considering that Jordan blocks are lower-triangular matrices instead of upper-triangular matrices as in many other books.

If $\rho(f) = 0$ then $e = 0$ and we just need to find a normal basis which will be automatically adapted to f'.

The interesting case happens when $\rho(f) = r > 0$. In this case, the minimal polynomial for $\overline{f}$ is divisible by $(x-1)^N$ for some maximal $N \geq r$. Let V be the maximal invariant subspace killed by $(x-1)^N$. Clearly, $(\overline{f} - I)^{r-1}(e)$ is an eigenvector in V. Consider this set of linearly independent vectors

$$e, (\overline{f} - I)(e), (\overline{f} - I)^2(e), \ldots, (\overline{f} - I)^{r-1}(e).$$

This set can be completed to a Jordan basis for V in such a way that they form a Jordan block. The problem of extending a set of linearly independent vectors to a Jordan basis was studied in [2] where necessary and sufficient condition for extendability were found. One sees easily that these conditions are satisfied in this case.

So, we need an algorithm to effectively complete these vectors to a Jordan basis of V. The standard way to find a Jordan basis is by induction, since it is rather easy to obtain a Jordan basis for an invariant subspace V once we have a Jordan basis for the subspace $(\overline{f} - I)(V)$. Hence, we just need to adapt this inductive algorithm to the case in which we already have a vector e which we want to be the top vector of a Jordan block. To learn more about the details of the function `NormalFormOfAffinity(f)` one can check its code since we will not go deeper into it here.

5 Euclidean Affine Spaces

By default, the basic function `AffineSpace(K,d)` constructs $\mathbb{A}^d$ with the inner product given by the identity matrix. Hence, given vectors u, v we can compute $\langle u, v \rangle$ by `u.inner_product(v)`. If, instead of the standard inner product we want to use the inner product given by the symmetric matrix M, we can use

```
sage:AffineSpace(K,d,metric=M)
```

5.1 Distance

The main function in this section is the one which computes the distance between two varieties:

```
sage:SquareDistance(X,Y)
```

```
sage:SquareDistance(X,Y,points=True)
```

Here X, Y are varieties or points and the first function returns the square of the distance between X and Y. If we set the optional argument as True, the function also returns points $x \in X$, $y \in Y$ at the minimum distance.

```
sage:OrthogonalVariety(P,X)
```

yields the variety Y which is orthogonal to X and such that $X \cap Y = P$. The boolean

```
sage:AreOrthogonal(X,Y)
```

decides if the varieties X and Y are orthogonal. Finally, given a variety X, we can construct the orthogonal symmetry with respect to X and the orthogonal projection to X:

```
sage:OrthogonalSymmetry(X)
```

```
sage:OrthogonalProjection(X)
```

5.2 Euclidean motions

The main topics in this section are to decide if an affinity is an euclidean motion and to decide if two euclidean motions are similar through an euclidean motion. We provide the following functions:

```
sage:IsEuclideanMotion(f,orthobase=True)
```

decides if f is an euclidean motion or not. If the optional argument `orthobase` is set to True, then this function also returns an orthonormal basis in which the linear part of f has normal form consisting of diagonal entries ± 1 and rotation blocks.

According to theorem 6.23 in the book, two euclidean motions are (euclidean) similar if and only if the linear parts are euclidean similars and the glide vectors have the same length. The *glide vector* is computed by

```
sage:GlideVector(f)
```

and the function

```
sage:IsEuclideanSimilar(f,g,transformation=True)
```

decides if the euclidean motions f and g are euclidean similar or not. If the optional argument `transformation` is set to True, then this function also returns an euclidean motion which conjugates f to g. There is also the related function

```
sage:EuclideanNormalFormOfAffinity(f,transformation=True)
```

which displays the normal form of f (as defined in the book) and also the matrix of an affinity which conjugates f to this normal form. The function

```
sage:RotationMatrix(P,theta)
```

provides the matrix of a (plane) rotation with center P and angle theta, which is given as an element in the symbolic ring, like pi/5, 2*pi/7, etc. Conversely, the function

```
sage:IsRotation(M,params=True)
```

decides if a matrix 2 by 2 matrix M is a rotation or not and if it is, then it returns its center and its angle (if possible).

References

[1] R.A. Beezer, *Sage for Linear Algebra. A Supplement to A First Course in Linear Algebra*. Available at `linear.ups.edu`. GNU Free Documentation License. 2011

[2] R. Bru, M. López Pellicer, *Extensions of algebraic Jordan basis*. Glas. Mat. Ser. III 20(40) (1985), no. 2, 289–292.

[3] `sagemath.org`

[4] W. Bosma, J. J. Cannon, C. Fieker, A. Steel (eds.), *Handbook of Magma functions*, Edition 2.20 (2014).

[5] A. Reventós, *Affine Maps, Euclidean Motions and Quadrics*, Springer Undergraduate Mathematics Series. 2011.