FINITE FIELD EXPERIMENTS

JAKOB KRÖKER, MIKE STILLMANN, AND HANS-CHRISTIAN V. BOTHMER

1. INTRODUCTION

This article describes the packages `BlackBoxParameterSpaces` (rename Package) and `FiniteFieldExperiments` that are mostly used together, but the first one can also be used alone for some applications.

The purpose of these packages is to study moduli spaces or more generally parameter spaces X together with their universal families. This study comes in two flavours:

(a) The first situation is one, in which one has a unirational parametrisation

$$\phi\colon \mathbb{A}^n \to X$$

of some parameter space. In this situation one is often interested in the stratification of X by some property of the parametrised objects. Our running example for this type of question is the stratification of the space of cubic surfaces in $\mathbb{P}^3$ by singularity type. Here X is represented by a `BlackBoxParameterSpace`.

(b) In the second situation X is given as a subvariety of some larger unirational moduli space

$$X \subset \mathbb{A}^n.$$

Here one is interested in the components of X and their moduli interpretation. Our running example in this case is the variety of complexes. Here X is represented by a `BlackBoxIdeal`.

In both cases the study of X is done over a finite field by looking at a large number random points in $\mathbb{A}^n$. Since we are over a finite field, we have some non zero chance of finding points in interesting strata (in the first situation) or on interesting components (in the second case). Moreover the statistics of such a finite field experiment contain non trivial heuristic information about the stratification (in the first case) or the irreducible components of X (in the second situation). The interesting points and the statistical information generated are collected in an object of type `FiniteFieldExperiment`.

In our implementation the package `BlackBoxParameterSpaces` contains the tools to set up a parameter space X as above together with its universal family, so that it can be used in a finite field experiment. It also provides some black-box-algorithms that still work if instead of the equations of $X \subset \mathbb{A}^n$ only an algorithm to evaluate the equations is given. To explain the difference between an equation and an evaluation algorithm we look at the example of determinants.

The package `FiniteFieldExperiments` contains the tools to run a finite field experiment, collect the results and interpret the statistical information gathered.

People familiar with the idea of finite field experiments will notice that a specialised package for this type of study is not strictly necessary. The advantage of our implementation to such people is:

(a) The automatic bookkeeping during a finite field experiment: The algorithms provided will use heuristic methods to store interesting points while forgetting uninteresting ones. At the same time they will collect the stochastic information generated by the experiment.
(b) The possibility of continuing an experiment: If an experiment is run with a few points first, and then with increasingly more points, the algorithms will collect and combine the information found in all runs.
(c) The black-box-Algorithms

For novices, the structured way of setting up a finite field experiment with this package, in particular the `xxxAt`-notation seems to help to get into the philosophy of this type of study.

(put references to Frank and Tonoli here)

2. Black-Box-Parameter spaces

A `BlackBoxParameterSpace` represents an unirational parameter space

$$\mathbb{A}^n \to X$$

together with its universal family. While the parameter space $\mathbb{A}^n$ itself is trivial, the interesting information lies in the objects that are parametrized by this space. In this package these objects and their properties are modelled by `pointProperties`. Any function that depends only on the coordinated of a point $P \in \mathbb{A}^n$ can be a point property. Points are modelled by row vectors over a field, while point properties are functions that take a matrix as input and can have any type as output. We use the convention that the name of a point property is of the form `xxxAt`, because it computes a property of the prametrised object *at* a given point. Point properties need to be registered in a black box parameter space so they can later be used by a finite field experiment.

- `blackBoxParameterSpace` - make a new `BlackBoxParameterSpace`
- `registerPointProperty` or `rpp` - register a new `pointProperty`
- `updatePointProperty` or `upp` - change an existing `pointProperty`
- `pointProperties` - list the pointProperties of a black box
- `hasPointPropery` - check whether a point property is defined

As a running example we study the stratification of the space of cubic surfaces by singularity type.

(put the definition of the blackbox in the example of cubic surfaces here)

3. Finite field experiments with a Black-Box-Parameter space

In a `FiniteFieldExperiment` we take a `BlackBoxParameterSpace` and evaluate some of its `pointProperties` at a large number of random points. If we work over a finite field, we will find points on interesting strata with

a small but positive probability. Furthermore the number of points found in each stratum gives heuristic information about the codimension of each stratum.

Let's first explain the heuristic reasoning used in this package. Let's assume we work over a finite field $\mathbb{F}_q$ with q elements. If $f \in \mathbb{F}_q[x_1, \ldots, x_n]$ is a polynomial, and $x \in \mathbb{A}^n$ is a point, then

$$f(x) \in \mathbb{F}_q$$

For most polynomials each of the q possible values of $f(P)$ occurs with approximately the same probability. Therefore

$$f(x) = 0$$

occurs with probability about $1/q$, so we expect about q^{n-1} points on a hypersurface in $\mathbb{A}^n$, and this value does not depend on the degree of the hypersurface. A much more sophisticated analysis of the situation is described by the Weil-Conjectures

put some version of the Weil-Conjecture here.

So in a first order approximation we have

$$|X(\mathbb{F}_q)| \approx q^{\dim X}.$$

If $X \subset \mathbb{A}^n$ and we choose random points in $\mathbb{A}^n$ then the probability of finding a point on X is approximately

$$P(x \in X) = \frac{|X(\mathbb{F}_q)|}{q^n} \approx q^{\dim X - n} = q^{-\operatorname{codim} X}.$$

Taking logarithms we get

$$-\log_q P(x \in X) \approx \operatorname{codim} X.$$

- `new FiniteFieldExperiment from BlackBoxParameterSpace`
- `watchProperty` - select a `pointProperty` from the Black box to be evaluated at random points
- `watchedProperites` - a list of currently evaluated point properties
- `run` - evaluate the watched properties at an additional number of points
- `trials` - the number of point evaluated so far
- `counts` - the number of points found on each stratum
- `estimatedCodimension` - estimate the codimension of each stratum using the heuristic explained above
- `pointsByKey` - a list of points on a given stratum.
- `tryProperty` - evaluate a new point property on the points found so far.
- `clear` - erase all points and statistics.
- possibly: `load` and `save`.

Returning to the cubic surfaces of the preceding section we set up a finite field experiment:

(put the finite field experiment in the example of cubic surfaces here)

4. Black-Box-Ideals

A `BlackBoxIdeal` represents a parameter space

$$X \subset \mathbb{A}^n$$

that is a subvariety of an unirational parameter space as in Section 2.

Let us assume for now, that the equations defining X are known. We discuss the BlackBox-Part of this implementation later in Section **??**.

In this situation some useful `pointProperties` can always be defined:

- `isZeroAt` - true when all equations of X vanish at the given point.
- `valuesAt` - returns the values of the equations of X at the given point.
- `jacobianAt` - evaluates the derivatives of the equations of X at the given point.
- `rankJacobianAt` - returns the rank of the Jacobian matrix of X at the given point. This is also the codimension of the tangent space of X at the given point.
- `isCertainlySingularAt` - tries to prove that X is singular at the given point. Returns wether such a proof was found.
- `isProbablySmoothAt` - true if `isCertainlySingular` is false.

As a running example we study the variety X of complexes

$$K^2 \xrightarrow{A} K^2 \xrightarrow{B} K.$$

It is the variety defined by the equation $AB = 0$ in the space of all pairs of matrices $(A, B) \in \mathbb{A}^6$.

(put the definition of the blackbox in the example of the variety of complexes here)

5. Finite Field Experiments with a Black-Box-Ideal

If we run a `FiniteFieldExperiment` with a `BlackBoxIdeal` defining X we are interested in the decomposition of X into irreducible components. Ideally we would like to find `pointProperties` that distinguish these components from each other. The statistics produced by a finite field experiment can help to guide this search.

Let's explain how we can obtain heuristic information on the number and codimension of the components of X. If we work with `BlackBoxIdeals` this information is much more precise than for the stratification of a `BlackBoxParameterSpace`, since we have access to the tangent space of X at each point $x \in X$ via the Jacobian matrix.

More precisely let $X = X_1 \cup \cdots \cup X_n$ where X_i is the union of all components of X that have codimension i. If c_i is the number of components of X_i and $\dim X_i \geq 1$ we know from the Weil conjectures that

$$\frac{|X_i(\mathbb{F}_q)|}{q^n} \approx c_i q^{-i}.$$

From this it follows that

$$P(x \in X_i) \cdot q^i \approx c_i.$$

Now we estimate the probability $P(x \in X_i)$ of finding a point on the union of codimension i components by looking at the probability of finding a point with a codimension i tangent space:

$$P(x \in X_i) \approx P(x \mid \operatorname{codim} T_{X,x} = i)$$

This gives us a heuristic estimate for the number of components of X in each codimension.

In this approximation all smooth points of X are counted in the correct codimension, but singular points will miscounted in a smaller codimension. If X is reduced, the set of singular points is of codimension at least 1 so presumably there are a lot more smooth points than singular ones, and this miscounting does not matter too much. If one worries that this might be a problem, one can run `tryProperty` with `isProbablySmoothAt` to see wether most of the found points $x \in X$ are indeed smooth.

Let's now do this for our variety of complexes

(put the finite field experiment in the example of cubic surfaces here)

6. Jets and Interpolation

Sometimes the equations of a parameter space $X \subset \mathbb{A}^n$ are not known explicitly, but only an algorithm to evaluate them is given. An example might be that X is defined by the vanishing of a determinant of a matrix of polynomials M. While it is easy to evaluate M at a point and then compute the determinant, it is sometimes very time and space consuming to compute the determinant of M first and then evaluate at a point.

(some matrix example illustrating this)

In this situation one can still compute the derivatives of the equations of X in a specific point by evaluating over the ring $K[\epsilon]/\epsilon^2$, i.e. we use

$$f(x + \epsilon v) = f(x) + \epsilon f'_v(x)$$

where f_v is the derivative of f in the direction of v. This gives a way of defining `jacobianAt` even when only an evaluation algorithm is known. If

$$f(x + \epsilon v) = 0$$

we call $x + \epsilon v$ a *tangent vector* of X at x.

One can go further along this way, by evaluating the equations over the ring $K[\epsilon]/\epsilon^{d+1}$. We call $j = x + \epsilon x_1 + \ldots \epsilon^d x_d \in K[\epsilon]/\epsilon^{d+1}]$ a jet of length d starting at x. If all equations of X vanish on j we say that j lies on X. Geometrically j denotes a small curve germ on X starting at x.

Now, if $x \in X$ is a smooth point, then a jet j of arbitrary length d starting at x can be found by a variant of newtons method. All of this can be done only with an evaluation method for the equations of X.

- `Jet` - a type for handling jets.
- `epsRing` - get a ring with $\epsilon^d = 0$.
- `jetAt` - compute a random jet stating at a given point
- `continueJet` - increase the length of a given jet.
- `length` - the length of a jet.

- `substitute(Ideal,Jet)` - evaluate the generators of an ideal at the jet

Sometimes it is useful to consider a set of jets starting at the same point. It is usually faster to compute many short jets instead of one long jet at a given point. (so far this is used nowhere)

- `JetSet` - a type for handling a set of jets starting at the same point.
- `new JetSet from Jet` - make a one element JetSet.
- `size(JetSet)` - the number of Jets in a JetSet.
- `addElement` - adds a Jet to a JetSet.

The first application of this is a heuristic test for smoothness. For a given point $x \in X$ we try to construct a jet of length d on X starting at x. If this fails, X must be singular at x. If we succeed X is probably smooth at x. For large d the probability of a wrong answer will get arbitrarily small.

- `isCertainlySingularAt` - heuristic test of smoothness
- `isProbablySmoothAt` - heuristic test of smoothness
- `singularityTestOptions` - show current parameters for singularity test
- `setSingularityTestOptions` - change how singularities are detected

The second application of jets is finding equations that contain a given component of X. For this let $x \in X$ be a smooth point and $X' \subset X$ be the component of X that contains x. In this situation every equation in the ideal of X' must vanish on all jets j that lie on X and start at x. This is so because the computation of jets sees only the local geometry of X and therefore each jet on X must also lie on X' (and not on other components). If we want to find the equations of X' that involve a certain list of monomials. If r is the length of this list, we find a jet j of length $d \geq r$ and interpolate. For d large and j general enough the only polynomials that vanish on j are those that also vanish on X'.

- `InterpolatedComponent` - a type for handling partial information about irreducible components
- `interpolate` - find polynomials containing a given Jet or JetSet
- `interpolateComponentAt` - interpolate equations for the component of a variety defined by a BlackBoxIdeal at a given point. The BlackBox will add this component to its list of known components.
- `interpolateComponentsAt` - depreciate or rename
- `interpolatedComponentByName` - select an interpolated component of a given BlackBoxIdeal
- `renameInterpolatedComponent` - change the name of an interpolated component
- `refineInterpolation` - increases the maximal interpolation degree for all interpolated components of a BlackBoxIdeal.
- `resetInterpolation` - erases all interpolated components of a BlackBoxIdeal.

The last application of jets is a method to detect on which component of X a given point $x \in X$ lies. With the interpolation scheme above one is never sure wether one has found all equations of a component $X' \subset X$, but for the component membership test using jets it is not necessary to now all equations:

Assume that we know only one polynomial f that vanishes on a component X' of X. Lets furthermore assume that we have a point $x \in X$ such that $f(x) = 0$. We know want to know, wether x lies on X' or on some other component X''. To check this we compute a random jet j on X starting at x. If $X' = X''$ then j lies on X' and we have $f(j) = 0$. If $X' \neq X''$ and f does not vanish on all of X'' then f vanishes on a divisor $D \subset X''$ and $x \in D$. In this situation j will lie on X'', but, if it is general enough, it will not lie on D. Therefore $f(j) \neq 0$ in this case.

- `isOnInterpolatedComponent` - checks wether a point is probably on a given component
- `onComponentPrecision` - the length of the jets used for determining component membership
- `setOnComponentPrecision` - changes the length of the jets used for determining component membership
- `interpolatedComponentsAt` - lists all components that could possibly contain a given point
- `interpolatedComponentNamesAt` – lists the names of all components that could possibly contain a given point

The ideas of this section are modelled on the concept of witness points in numerical algebraic geometry [**?**].

Let's look at an example of all of this. Let X be the union of a line, a plane conic and a plane cubic where the conic and the cubic lie in the same plane, but the line does not.

(put the interpolation example here)

7. Handling Errors (??)

(explain how the throw/catch mechanism is used in a FiniteFieldExperiment?)