MCLF Documentation

Release 1.0

Stefan Wewers

Aug 09, 2017

Contents:

CHAPTER ONE

BERKOVICH CURVES

1.1 The Berkovich line

The Berkovich projective line over a discretely valued field.

Let K be a field and v_K a discrete valuation on *K*. Let *F=K(x)* be a rational function field over *K*. We consider *F* as the function field of the projective line *X* over *K*. Let X^{an} denote the *(K,v_K)*-analytic space associated to *X*. Then a point ξ on X^{an} may be identified with a (real valued) pseudo-valuation v_ξ on *F* extending *v_K*.

Note that we do not assume *K* to be complete with respect to *v_K*. Hence we can work with 'exact' fields, e.g. number fields.

There are only two kind of 'points' which are relevant for us and can be handled using the mac_lane infrastructure:

- **Type I, algebraic: these are the points that come from a closed point** on the (algebraic) projective line over the completed base field.
- **Type II: these are the points which correspond to discrete valuations** on the function field whose residue field is a function field over the residue base field

For both these kind of points, the corresponding pseudovaluation on *F* are directly realizable inside the `mac_lane` infrastructure.

If *v_xi(x)geq 0* we say that *xi* lies *in the unit disk*. Then the restriction of *v_xi* to *K[x]* is a discrete pseudo-valuation which can be realized either as an inductive valuation, or as a limit valuation.

If *xi* does not lie in the unit disk, then we use instead the restriction of *v_xi* to the polynomial ring *K[x^{-1}]* (internally, we use the ring *K[x]*, though).

By a result of Berkovich, the topological space *X^{an}* is a *simply connected quasi-polyhedron*. Among other things this means that for any two points *xi_1,xi2in X^{an}* there exists a unique closed subset

$$[\xi_1, \xi_2] \subset X^{an}$$

which is homeomorphic to the unit interval *[0,1]subsetRR* in such a way that *xi_1,xi_2* are mapped to the endpoints *0,1*.

Let *xi^gin X^{an}* denote the *Gauss point*, corresponding to the Gauss valuation on *F=K(x)* with respect to the parameter *x*. Then *X^{an}* has a unique partial ordering determined by the following two conditions: - *xi^g* is the smallest element - we have *xi_1<xi_2* if and only if *xi_2* lies in a connected component

of *X^{an}-{xi_1}* which does not contain *xi^g*.

A point *xi* of type II has a *discoid representation* as follows. If *xi=xi^g* then *D_xi* is defined as the closed unit disk. Otherwise, *D_xi* is defined of the set of all points *xi_1in X^{an}* such that *xileqxi_1*. Then *D_xi* is of the form

$$D_\xi = \{\xi_1 \mid v_{\xi_1}(f) \geq s\},$$

where *f* is a polynomial in *x* or in *x^{-1}*, irreducible over $\hat{\bar{K}}$ and *s* is a nonnegativ rational number. The pair *(f,s)* determines *xi*, but this representation is not unique.

Note that we can't simply extend the discoid representation to points of type I by allowing *s* to take the value *infty*.

AUTHORS:

- Stefan Wewers (2017-02-10): initial version

EXAMPLES:

```
<Lots and lots of examples>
```

TO DO:

- allow "virtual functions" for the evaluations of valuations (and derivations). "Virtual functions" are products of rational functions with possible rational exponents.
- use more cached functions; this may improve speed
- more systematic (and explicitly defined) relation between points and their discoid representation
- more doctests!

class `mclf.berkovich.berkovich_line.`**`BerkovichLine`**(*F*, *vK*)

The class of a Berkovich projective line over a discretely valued field.

Let *K* be a field and *v_K* a discrete valuation on *K*. Let *F=K(x)* be a rational function field over *K*. We consider *F* a the function field of the projective line *X* over *K*. Let *X^{an}* denote the *(K,v_K)*-analytic space associated to *X*. Then a point *xi* on *X^{an}* may be identified with a (real valued) pseudo-valuation *v_xi* on *F* extending *v_K*.

INPUT:

- `F` – a rational function field over a base field K
- `vK` – a discrete valuation on the base field K

`divisor`(*f*)

Return the divisor of a rational function *f*.

INPUT:

- *f* – a nonzero element of the function field of self

OUTPUT:

the divisor of *f*, as a list of pairs *(xi, m)*, where *xi* is a point of type I and m is the multiplicity of *xi*.

`find_zero`(*xi1*, *xi2*, *f*)

Return the point between `xi1` and `xi2` where `f` has valuation *0*.

INPUT:

- **`xi1`, `xi2` – points on the Berkovich line such that** `\xi_1``<``xi2`
- **`f` – a nonconstant rational function; it is assumed that the signs** of the valuations of `f` at `xi1` and `xi2` are different

OUTPUT:

The smallest point between `xi1` and `xi2` where the valuation of `f` is zero.

We are assuming for the moment that the function

$$v \mapsto v(f)$$

is affine (i.e. has no kinks) on the interval *[xi_1,xi_2]*.

gauss_point ( )

Return the Gauss point of self.

The Gauss point is the type-II-point corresponding to the Gauss valuation on *K[x]*.

infty ( )

Return the point *infty*.

make_polynomial (*f*, *in_unit_disk=True*)

Turn `f` into a polynomial.

INPUT:

- `f` – an element of *F=K(x)* or of *K[x]*
- `in_unit_disk` – boolean

OUTPUT:

Either *f* of *f(1/x)*, considered as a polynomial in *K[x]*, and depending on whether `in_uni_disk` is true or false.

point_from_discoid (*phi*, *s*, *in_unit_disk=True*)

Return the point on `self` determined by a discoid.

INPUT:

- `phi` – a monic and integral polynomial in *K[x]*
- `s` – a nonnegative rational number, or *infty*
- `in_unit_disk` – a boolean (default: True)

OUTPUT:

a point *xi* on the Berkovich line `self` which is the unique boundary point of the discoid *D* determined as follows: if `in_unit_disk` is true then *D* is the set of valuations *v* on *K[x]* such that *v(phi)geq s*. Otherwise, it is the image of this point under the automorphism *xto 1/x*.

If *D* defined above is not irreducible (and hence not a discoid) then an error is raised.

point_from_polynomial_pseudovaluation (*v*, *in_unit_disk=True*)

Return the point corresponding to a pseudo-valuation on a poylnomial ring.

INPUT:

- **v – a discrete pseudo-valuation on the polynomial ring *K[x]*,** extending the base valuation *v_K*
- `in_unit_disk` (default=True) – boolean

OUTPUT:

The point on the unit disk corresponding to `v` (if `in_unit_disk` is true), or the point on the inverse unit disk corresponding to `v` .

point_from_pseudovaluation (*v*)

Return the point on the Berkovich line corresponding to the pseudovaluation `v` .

INPUT:

- **v – a discrete pseudovaluation on the function field of `self` ,** extending the base valuation *v_K*

OUTPUT:

The point *xi* on the Berkovich line *X* =‘‘self‘‘ corresponding to the pseudo valuation `v` on the function field of *X*.

polynomial_divisor (*f*, *m*)

Return the divisor of zeroes of a squarefree polynomial.

INPUT:

- **`f` – a squarefree polynomial in the generator of the function** field of self
- `m` – an integer

OUTPUT:

The divisor of *f*, multiplied with *m*.

NOTE:

At the moment, we must require that the Newton polygon of *f* has either only nonpositive or only positive slopes. So the zeroes of *f* lie all inside the closed unit disk, or all outside.

class `mclf.berkovich.berkovich_line.`**PointOnBerkovichLine**

A point on a Berkovich projective line.

We only allow two different types of points:

- **Type I, algebraic: these are the points that come from a closed point** on the (algebraic) projective line over the completed base field.
- **Type II: these are the points which correspond to discrete valuations** on the function field whose residue field is a function field over the residue base field

In particular, the Gauss valuation on *F=K(x)* with respect to the parameter *x* corresponds t a point *xi^g* of type II on *X^{an}* which we call the *Gauss point*.

The set *X^{an}* has a canonical partial ordering in which the Gauss point is the smallest elements. All point of type I are maximal elements.

is_strictly_less (*xi1*)

return True if self is strictly smaller than xi1.

make_polynomial (*f*)

Return the polynomial corresponding to `f`.

INPUT:

- `f` – an element of *F=K(x)*

OUTPUT:

If `f` is an element of the function field *F=K(x)* the we return

- f as an element of *K[x]* if possible and `self` lies in the unit disk
- f(1/x) as an element of *K[x]* if possible and ‘‘lies outside the unit disk

Otherwise an error is raised.

This function is useful to converting elements of the function field to elements of the domain of the MacLane valuation underlying `self`.

class `mclf.berkovich.berkovich_line.`**TypeIIPointOnBerkovichLine** (*X*, *v*)

A point of type II on a Berkovich line. INPUT:

- `X` – a Berkovich line over a valued field K
- `v` – a discrete valuation on the function field of X extending the base valuation

approximation ()

Return an approximation of `self`. For a point of type II, `self` is already an approximation of itself.

discoid (*certified_point=None*)
Return a representation of the discoid of which `self` is the unique boundary point.

INPUT:

- **`certified_point` (default=None) – this argument is not used** for type-II-points

OUTPUT:

A pair *(f, s)*, where *f* is a polynomial in the generator *x* of the function field of *X*, or a polynomial in *1/x*, and where *s* is a nonnegative rational number. This data represents a discoid *D* via the condition *v_xi(f)geq s*.

Then `self` is the unique boundary point of *D*, and if, moreover, `self` is not the Gauss point then *D* contains precisely the points *xi* which are greater or equal to `self`. If `self` is the Gauss point then *D* is the standard closed unit disk, *f=x* and *s=0*.

improved_approximation ( )
Return an improved approximation of `self`. This is meaningless for type-II-points, so self is returned.

infimum (*xi2*)
Return the infimum of self and xi2.

INPUT:

- `xi2` – a point of type I or II on the Berkovich line

OUTPUT:

The infimum of self and *xi_2* (w.r.t. to the natural partial ordering). Unless *xi_2=infty* or self is equal to *xi_2*, the result is a point of type II.

is_equal (*xi*)
Return True if self is equal to `xi`.

is_inductive ( )
True if `self` corresponds to an inductive pseud-valuation. This is always true for points of type II.

is_leq (*xi*)
Return True if self is less or equal to xi.

INPUT:

- `xi` – a point of type I or II

OUTPUT:

True if self is less or equal to `xi` (w.r.t. the natural partial order on *X*)

is_limit_point ( )
True is `self` corresponds to a limit valuation. This is never true for points of type II.

point_in_between (*xi1*)
Return a point in between `self` and `xi1`.

INPUT:

- `xi1` – a point which is strictly smaller than `self`

OUTPUT: a point which lies strictly between `self` and `xi1`

pseudovaluation_on_polynomial_ring ( )
Return the pseudo-valuation on the polynomial ring 'K[y]' corresponding to `self`, where *y* is either *x* or *1/x* depending on whether self lies in the standard closed unit disk or not.

type ( )
Return the type of self.

v (*f*)
Evaluate element of the function field on the valuation corresponding to self.

INPUT:

- `f` – an element of the function field of the underlying projective line

OUTPUT:

the value *v(f)*, where *v* is the valuation corresponding to self

class `mclf.berkovich.berkovich_line.` **TypeIPointOnBerkovichLine** (*X*, *v*)
An algebraic point of type I.

INPUT:

- *X* – a Berkovich projective line over a valued field *K*
- *v* – an infinite pseudo-valuation on the function field of *X*

OUTPUT:

The point on *X* corresponding to *v*.

approximation (*certified_point=None*)
Return an approximation of `self`.

INPUT:

- `certified point` (default=None) – a point on the Berkovich line

OUTPUT:

An inductive point which approximates `self`, in such a way that we can distinguish `self` from `certified point`.

If `self` is an inductive point, then `self` is returned. Otherwise, `self` is a limit point, and the output is a point of type II greater or equal to `self` (i.e. corresponding to a discoid containing `self`). If `certified_point` is not None and distinct from `self`, then the output is not greater or equal to `certified_point`.

TO DO : We should also make sure that the approximation has the same degree as the point itself. If the point is generated as part of the support of a principal divisor, then this should be ok, because of the default "require_final_EF=True" in "vK.approximants(f)".

discoid (*certified_point=None*)
Return a representation of a discoid approximating `self`.

INPUT:

- `certified_point` (default=None) – a point of type II

OUTPUT:

A pair *(f, s)*, where *f* is a polynomial in the generator *x* of the function field of *X*, or *f=1/x*, and where *s* is a nonrational number, or is equal to *infty*. This data represents a (possibly degenerate) discoid *D* via the condition *v_xi(f)geq s*.

D as above is either the degenerate discoid with *s=infty* which has `self` as the unique point, or *D* is an approximation of `self` (this simply means that `self` is contained in *D*). If `certified_point` is given then it is guaranteed that it is not contained in *D*.

improved_approximation ( )
Return an improved approximation of `self` .

infimum (*xi2*)
Return the infimum of self and *xi_2*.

INPUT:

- `xi2` – a point of type I or II on the Berkovich line

OUTPUT:

The infimum of self and *xi_2*. Unless self or *xi_2* are equal, this is a point of type II.

is_equal (*xi*)
Return True is self is equal to `xi` .

is_gauss_point ( )
Return True if self is the Gauss point.

is_in_unit_disk ( )
True is self is contained in the unit disk.

is_inductive ( )
True if `self` corresponds to an inductive MacLane valuation.

is_leq (*xi*)
Return True is self is less or equal to xi.

INPUT:

- `xi` – a point of type I or II

OUTPUT:

True if self is less or equal to `xi` (w.r.t. the natural partial order on *X* for which the Gauss pont is the smallest element). Since self is a point of type I and hence maximal, this is never true unless *xi* is equal to self.

is_limit_point ( )
True if `self` corresponds to a limit valuation.

pseudovaluation_on_polynomial_ring ( )
Return the MacLane valuation representing `self` .

OUTPUT:

A MacLane valuation on the polynomial ring *K[x]* representing `self` . It is either inductive or a limit valuation.

type ( )
Return the type of self

v (*f*)
Evaluate the pseudo-valuation corresponding to self on `f` .

INPUT:

- **f – an element of the function field *K(x)*** (or of the polynomial ring *K[x]*).

OUTPUT:

The value *v(f)*, where v is the pseudo-valuation corresponding to `self` . If *f* is in *K[x]* then we take for *v* the MacLane pseudo-valuation corresponding to `self` .

mclf.berkovich.berkovich_line.**equality_of_pseudo_valuations**(*v1*, *v2*)
Decide whether two pseudo-valuations are equal.

INPUT:

- **v1 , v2 – two pseudo-valuations on the same rational** function field *F=K(x)*

OUTPUT:

True if `v1` is equal to `v2`, False otherwise.

We actually assume that the restriction of `v1` and `v2` to the constant base field are equal.

mclf.berkovich.berkovich_line.**mac_lane_infimum**(*v1*, *v2*)
Return the infimum of *v_1* and *v_2*.

INPUT:

-`v1`, `v2` – MacLane valuations on *K[x]*‘

OUTPUT:

on the set of all MacLane valuations on *K[x]*.

mclf.berkovich.berkovich_line.**normalized_reduction**(*v*, *f*)
Return the normalized reduction of f with respect to v.

INPUT:

- v – type II valuation on a rational function field F = K(x)
- f – a strongly irreducible polynom in K[x], or 1/x

OUTPUT:

a nonzero element fb in the residue field of v ??? Precise definition needed!

mclf.berkovich.berkovich_line.**valuation_from_discoid**(*vK*, *f*, *s*)
Return the inductive valuation corresponding to a discoid.

INPUT:

- `vK` – a discrete valuation on a field *K*
- `f` – a nonconstant monic integral polynomial over *K*
- `s` – a nonnegative rational number, or *infty*

OUTPUT:

an inductive valuation `v` on the domain of `f`, extending `vK`, corresponding to the discoid *D* defined by *w(f)geq s*. In other words, this means that *D* defined above is irreducible (and hence a discoid), and *v* is its unique boundary point.

If *D* is not irreducible, an error is raised.

1.2 Berkovich trees

Finite children of the Berkovich projective line

Let *K* be a field and *v_K* a discrete valuation on *K*. Let *X=mathbb{P^1_K}* be the projective line over *K*. Let *X^{an}* denote the *(K,v_K)*-analytic space associated to *X*. We call *X^{an}* the *Berkovich line* over *K*.

Let *xi^g* be the *Gauss point* on *X^{an}*, corresponding to the Gauss valuation on the function field of *X* with respect to the canonical parameter *x*. Then *X^{an}* has a natural partial ordering for which *xi^g* is the smallest element. With respect to this partial ordering, any two elements have a unique infimum.

A *Berkovich tree* is a finite (nonempty) subset *T* with the property that for any two elements in *T* the infimum is also contained in *T*. In particular, a *T* has a least element, called the *root* of the tree.

Given any finite subset *S* of *X^{an}*, there is now a unique minimal subtree *T* contaning *S*. We call *T* the tree *spanned* by *S*.

This module realizes finite subtrees of *X^{an}* as combinatorial objects, more precisely as *finite rooted combinatorial trees*. So a tree consists of a root, and a list of children. If the tree is a subtree of another tree, then there is a link to its parent.

AUTHORS:

- Stefan Wewers (2017-02-13): initial version

EXAMPLES:

```
<Lots and lots of examples>
```

class `mclf.berkovich.berkovich_trees.`**`BerkovichTree`** (*X*, *root=None*, *children=None*, *parent=None*)

Create a new Berkovich tree *T*.

INPUT:

- `X` – a Berkovich line
- `root` – a point of `X` (default: None)
- `children` – a list of Berkovich trees on `X` (default = None)
- `parent` – a Berkovich tree or None (default: None)

OUTPUT:

A Berkovich tree *T* on *X* with root `root`, children `children` and parent `parent`. *T* may be empty (no root and no children), but if there are children then there must be root.

`adapt_to_function` (*f*)

Add all zeroes and poles of *f* as leaves of the tree.

INPUT:

- `f` – a rational function on *X*

OUTPUT:

the new tree obtained by adding all zeroes and poles of *f* as vertices to the old tree.

`add_point` (*xi*)

Return the tree spanned by self and the point xi.

INPUT:

- xi – A point of type I or II on X

OUTPUT:

T1, T2, where

- T1 is the tree obtained from T0=self after inserting xi as a vertex.
- T2 is the subtree of T1 with root xi

If T0 has a parent, then the root of T0 must be less than xi. Therefore, the parent of T1 will be the original parent of T0.

Note that this command may change the tree T0! For instance, xi may become the root of T1 and then T0 has T1 as new parent.

children ( )
Return the list of all children.

copy ( )
Return a copy of self.

find_point (*xi*)
Find subtree with root `xi` .

INPUT:

- `xi` – a point on the Berkovich line underlying `self`

OUTPUT:

The subtree *T* of `self` with root `xi` , or `None` if `xi` is not a vertex of `self` .

graph ( )
Return a graphical representation of self.

OUTPUT:

G, vert_dict,

where G is a graph object and vert_dict is a dictionary associating to a vertex of G the corresponding vertex of self.

has_parent ( )
Return True if self has a parent.

is_leaf ( )
Return True if self is a leaf.

leaves ( )
Return the list of all vertices.

make_parent (*parent*)
add `parent` as parent of self.

parent ( )
Return the parent of self.

paths ( )
Return the list of all directed paths of the tree.

OUTPUT:

the list of all directed paths of the tree, as a list of pairs *(xi_1,xi_2)*, where *xi_2* is a child of *xi_1*.

position (*xi*)
Find the position of `xi` in the tree T=self.

INPUT:

- `xi` – a point on the Berkovich line underlying T

OUTPUT:

xi1, T1, T2, is_vertex,

where - xi1 is the image of xi under the retraction map onto the total space of T

- T1 is the smallest subtree of T whose total space contains xi1
- T2 is the child of T1 such that xi1 lies on the edge connecting T1 and T2 (or is equal to T1 if xi1 is the root of T1)
- **is_vertex is True if xi1 is a vertex of T (which is then the root** of T1) or False otherwise

print_tree (*depth=0*)
Print the vertices of the tree, with identation corresponding to depth.

It would be nicer to plot the graph and then list the points corresponding to the vertices.

root ()
Return the root of the tree.

subtrees ()
Return the list of all subtrees.

vertices ()
Return the list of all vertices.

mclf.berkovich.berkovich_trees.**create_graph_recursive** (*T*, *G*, *vertex_dict*, *root_index*)
Create recursively a graph from a Berkovich tree.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

L

M

N

P

R

S

T

V