Computerpraktikum

Block I

Kurven im Komplexen und die Diskriminanteneinbettung

Matthias Künzer

Universität Stuttgart

Wintersemester 2018/19

Inhalt

Vorwort

Die Menge $\{ (x,y) \in \mathbf{R}^2 : \ y^2 = (x-1)(x+1)(x+3) \}$ ist eine Kurve in $\mathbf{R}^2$.

Wir wollen die Menge $\{ (x,y) \in \mathbf{C}^2 : \ y^2 = (x-1)(x+1)(x+3) \}$ untersuchen, noch um einen unendlichen Punkt ergänzt.

Es ist $\mathbf{C}^2$ komplex zweidimensional, also reell vierdimensional.

Da nun, separiert nach Real- und Imaginärteil, zwei reelle Gleichungen in $\mathbf{C}^2$ zu erfüllen sind, haben wir eine Fläche zu erwarten, die die obengenannte Kurve beinhaltet.

Nun müssen wir noch den unendlichen Punkt hinzunehmen und eine Projektion von $\mathbf{C}^2$ auf den dreidimensionalen Anschauungsraum vornehmen.

Ein Dank geht an Jens Künzer für eine Urversion der Graphiken, erstellt vor ein paar Jahren. Ein Dank für Korrekturen geht an Lorenz Jetter und Magnus Kühn.

Für weitere Hinweise auf Fehler und Unklarheiten bin ich dankbar.

Stuttgart, Herbst 2018

Matthias Künzer

1 Maple

Wir verwenden das Computeralgebra-System Maple.

Das Handbuch findet sich unter `www.maplesoft.com/support/help/`.

Eine nichtgraphische Oberfläche erhält man in einer Shell mittels `maple`.

Wir brauchen aber die graphische Oberfläche, die man durch Eingabe von `xmaple &` in einer Shell bekommt.

Sobald `xmaple` geöffnet ist, muß `New Worksheet` ausgewählt werden.

Damit die benötigten Pakete zur Verfügung stehen, ist zu Beginn

```
with(plots);
with(linalg);
```

einzugeben.

Eine normale Eingabeschrift erreicht man in `xmaple` übrigens mittels Tools, Options, Display, Input Display, Maple Notation, Apply Globally.

Einige der mittels Maple erstellten Bilder finden sich in den Galerien von `pnp.mathematik.uni-stuttgart.de/lexmath/kuenzer/compprak18/` in beweglicher Form.

2 Das reelle Bild

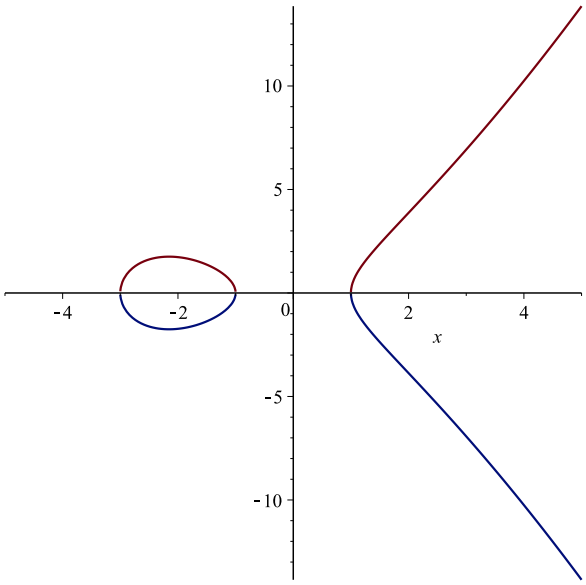
Das reelle Bild $\{\,(x,y) \in \mathbf{R}^2 :\; y^2 = (x-1)(x+1)(x+3)\,\}$ erhält man mittels

```
plot({sqrt((x-1)*(x+1)*(x+3)),-sqrt((x-1)*(x+1)*(x+3))},x=-5..5);
```

oder mittels

```
implicitplot(y^2 = (x-1)*(x+1)*(x+3), x = -5..5, y = -5..5);
```

Ersteres gibt ein Bild wie folgt.



3 Fundamentalsatz

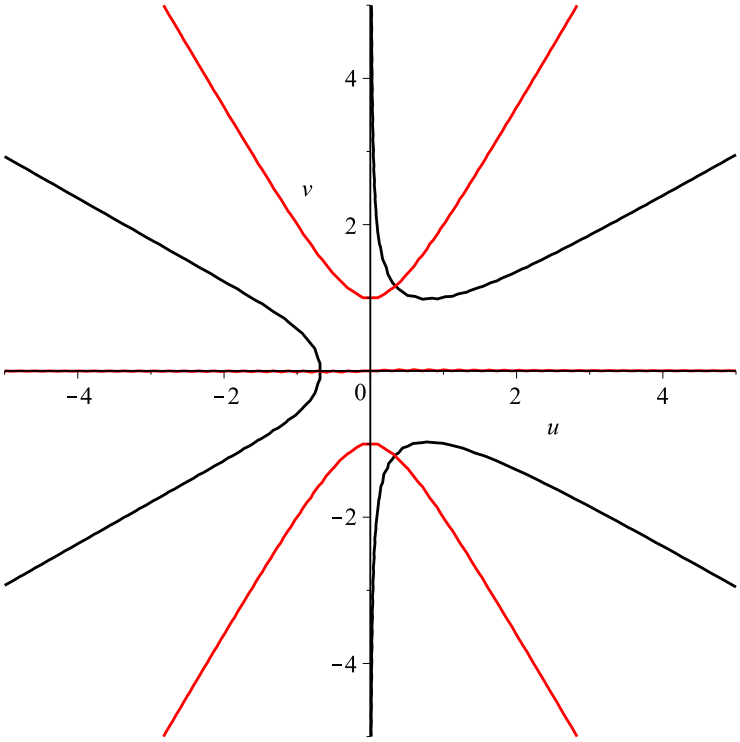
Mittels `implicitplot` kann man auch Nullstellenmengen in der komplexen Zahlenebene beschreiben. Sei e.g. folgendes Polynom betrachtet.

```
f := x -> x^3 + x + 1;
```

Die Nullstellen dieses Polynoms liegen in der Schnittmenge der Nullstellenmenge von `Re(f)` und `Im(f)`. Mittels

```
display(
implicitplot(Re(f(u + I*v)) = 0, u = -5..5, v = -5..5, grid = [50,50], color = "black"),
implicitplot(Im(f(u + I*v)) = 0, u = -5..5, v = -5..5, grid = [50,50], color = "red")
);
```

erhalten wir folgendes Bild.



Da sich für betragsgroße x das Polynom `f` im wesentlichen wie x^3 verhält, sind die Nullstellengeraden von $\mathrm{Re}(x^3)$ und $\mathrm{Im}(x^3)$ Asymptoten an die einzelnen Äste in diesem Bild. Dies wird auch gern als landläufige Begründung des Fundamentalsatzes der Algebra herangezogen, da es anhand des Bildes plausibel ist, daß eben diese Asymptoten es erzwingen, daß sich ein schwarzer Ast und ein benachbarter roter Ast in einem Punkt schneiden.

Wir können auch einen Parameter `t` einführen und ein bewegtes Bild erzeugen.

```
f_param := (x,t) -> x^3 + x + t;

display(
animate(implicitplot, [Re(f_param(u + I*v,t)) = 0, u = -5..5, v = -5..5,
                           grid = [50,50], color = "black"], t = -1..1),
animate(implicitplot, [Im(f_param(u + I*v,t)) = 0, u = -5..5, v = -5..5,
                           grid = [50,50], color = "red"], t = -1..1)
);
```

Das resultierende Bild kann animiert werden, die rechte Maustaste liefert einen entsprechenden Menüpunkt.

Es kann auch als bewegtes gif-Bild exportiert werden. Siehe Galerie 1, Figur 1.

4 Riemannsche Zahlenkugel

Wir betrachten die Abbildung

$$\begin{array}{ccccc} \mathbf{C} & \to & \mathbf{C} & \times & \mathbf{R} \\ z & \mapsto & (\frac{2z}{|z|^2+1} & , & \frac{|z|^2-1}{|z|^2+1}) \end{array}$$

Wir identifizieren $\mathbf{C} \times \mathbf{R}$ mit $\mathbf{R}^3$ entlang $(z, x) \mapsto (\mathrm{Re}(z), \mathrm{Im}(z), x)$.

Dabei liegen die Punkte $(z, 0)$, $(\frac{2z}{|z|^2+1}, \frac{|z|^2-1}{|z|^2+1})$ und $(0, 1)$ auf einer Geraden. Denn es ist

$$(z, 0) - (0, 1) \;=\; (z, -1)\ ,$$

es ist

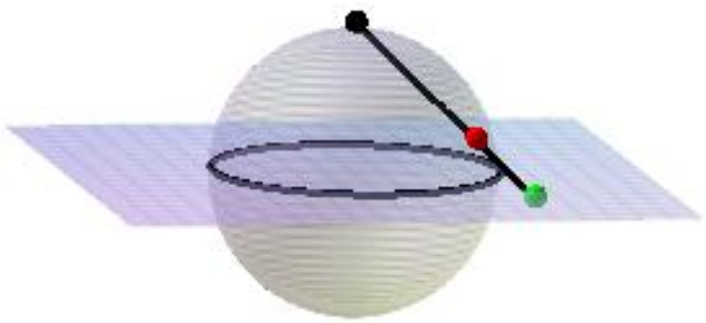
$$(\tfrac{2z}{|z|^2+1}\ ,\ \tfrac{|z|^2-1}{|z|^2+1}) - (0, 1) \;=\; (\tfrac{2z}{|z|^2+1}\ ,\ -\tfrac{2}{|z|^2+1})\ ,$$

und letzterer Verbindungsvektor ist der $\frac{2}{|z|^2+1}$-fache des ersteren.

Dabei liegt der Punkt $(\frac{2z}{|z|^2+1}, \frac{|z|^2-1}{|z|^2+1})$ auf der Einheitskugel mit Radius 1 um den Ursprung. Denn

$$\left|\tfrac{2z}{|z|^2+1}\right|^2 + \left(\tfrac{|z|^2-1}{|z|^2+1}\right)^2 \;=\; 1\ .$$

Unsere Abbildung nimmt also einen Punkt z aus $\mathbf{C}$, faßt ihn als $(z, 0)$ auf, legt eine Gerade durch $(z, 0)$ und $(0, 1)$ und projiziert $(z, 0)$ entlang dieser Geraden auf die Einheitskugel. So wird e.g. in folgendem Bild der grüne auf den roten Punkt abgebildet.



Diese Abbildung soll nun noch von $\mathbf{C}$ auf $\mathrm{P}^1(\mathbf{C})$ fortgesetzt werden.

Dabei war $\mathrm{P}^1(\mathbf{C})$ wie folgt konstruiert. Elemente (z', w'), $(z'', w'') \in \mathbf{C}^2 \setminus \{(0, 0)\}$ seien äquivalent, wenn es ein $\lambda \in \mathbf{C} \setminus \{0\}$ gibt mit $\lambda \cdot (z', w') = (z'', w'')$. Die Äquivalenzklasse eines solchen Elements (z, w) werde $(z : w)$ geschrieben. Kurz, für $(z, w) \in \mathbf{C}^2 \setminus \{(0, 0)\}$ ist

$$(z : w) \;=\; (\lambda z : \lambda w)$$

für $\lambda \in \mathbf{C} \smallsetminus \{0\}$.

Für $z,\ w\ \in \mathbf{C}$ schreiben wir $(z : w)$ für die Äquivalenzklasse von Elementen aus $\mathbf{C}^2 \smallsetminus \{0\}$, wenn wir Elemente (z', w'), $(z'', w'')\ \in\ \mathbf{C}^2 \smallsetminus \{0\}$ für äquivalent erklären, wenn es ein $\lambda \in \mathbf{C} \smallsetminus \{0\}$ gibt mit $\lambda \cdot (z', w') = (z'', w'')$.

Es wird nun $\mathbf{C}$ mit einer Teilmenge von $\mathrm{P}^1(\mathbf{C})$ identifiziert via $z \mapsto (z : 1)$. Der einzige Punkt von $\mathrm{P}^1(\mathbf{C})$, der nicht in dieser Teilmenge liegt, ist $(1 : 0)$. Ein sinnvolles Bild von $(1 : 0)$ unter der fortgesetzten Abbildung ist $(0, 1)$. Somit haben wir folgende fortgesetzte Abbildung.

$$\begin{array}{rcrcl} \mathrm{P}^1(\mathbf{C}) & \to & \mathbf{C} & \times & \mathbf{R} \\ (z : 1) & \mapsto & (\frac{2z}{|z|^2+1} & , & \frac{|z|^2-1}{|z|^2+1}) \\ (1 : 0) & \mapsto & (0 & , & 1) \end{array}$$

Dieselbe Abbildung läßt sich auch wie folgt schreiben.

$$\begin{array}{rcrcll} \mathrm{P}^1(\mathbf{C}) & \to & \mathbf{C} & \times & \mathbf{R} & \\ (z : w) & \mapsto & (\frac{2(z/w)}{|(z/w)|^2+1} & , & \frac{|(z/w)|^2-1}{|(z/w)|^2+1}) & \text{falls } w \neq 0 \\ (1 : 0) & \mapsto & (0 & , & 1) & \end{array}$$

Oder kurz wie folgt.

$$\begin{array}{rcrcl} \mathrm{P}^1(\mathbf{C}) & \to & \mathbf{C} & \times & \mathbf{R} \\ (z : w) & \mapsto & (\frac{2z\bar{w}}{|z|^2+|w|^2} & , & \frac{|z|^2-|w|^2}{|z|^2+|w|^2}) \end{array}$$

Diese bildet $\mathrm{P}^1(\mathbf{C})$ bijektiv auf die Einheitskugel ab.

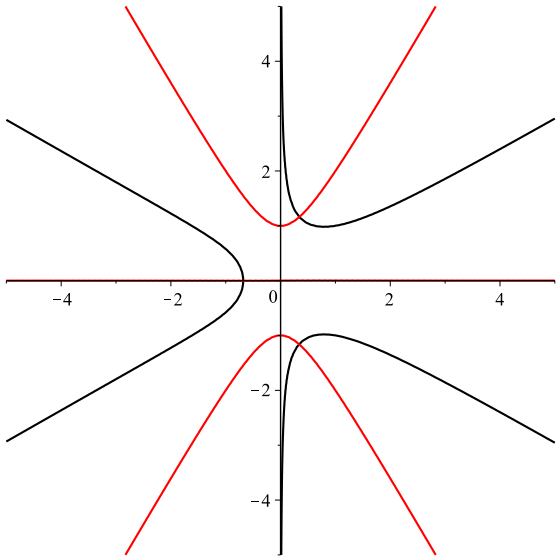
Wir greifen nun das Beispiel aus §3 wieder auf. Es wird nun zuerst das Resultat von `implicitplot` als Datensatz ausgegeben. Dieser wird dann auf zwei Weisen in eine Graphik umgesetzt.

```
f := x -> x^3 + x + 1;
ipre := implicitplot(Re(f(u + I*v)) = 0, u = -5..5, v = -5..5, grid = [100,100]):
ipim := implicitplot(Im(f(u + I*v)) = 0, u = -5..5, v = -5..5, grid = [100,100]):

pt_ipre := op([1,1..-2],ipre);
pt_ipim := op([1,1..-2],ipim);
```

Mittels

```
display(
 listplot(pt_ipre[1], symbol = point, color = "black")
,listplot(pt_ipre[2], symbol = point, color = "black")
,listplot(pt_ipre[3], symbol = point, color = "black")
,listplot(pt_ipim[1], symbol = point, color = "red")
,listplot(pt_ipim[2], symbol = point, color = "red")
,listplot(pt_ipim[3], symbol = point, color = "red")
);
```

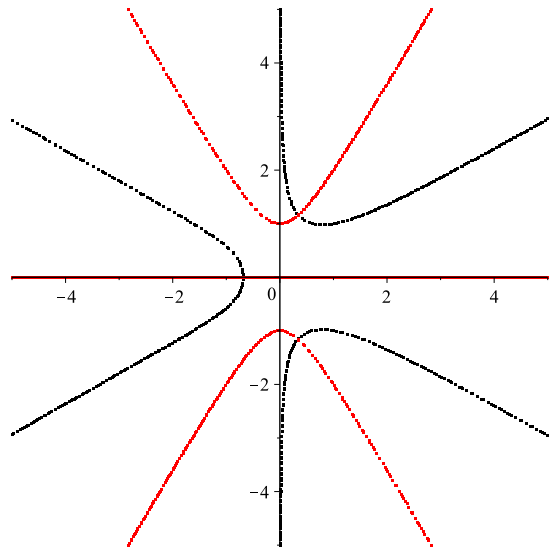
erhalten wir folgendes Bild.



Mittels

```
display(
 pointplot(pt_ipre[1], symbol = point, color = "black")
,pointplot(pt_ipre[2], symbol = point, color = "black")
,pointplot(pt_ipre[3], symbol = point, color = "black")
,pointplot(pt_ipim[1], symbol = point, color = "red")
,pointplot(pt_ipim[2], symbol = point, color = "red")
,pointplot(pt_ipim[3], symbol = point, color = "red")
);
```

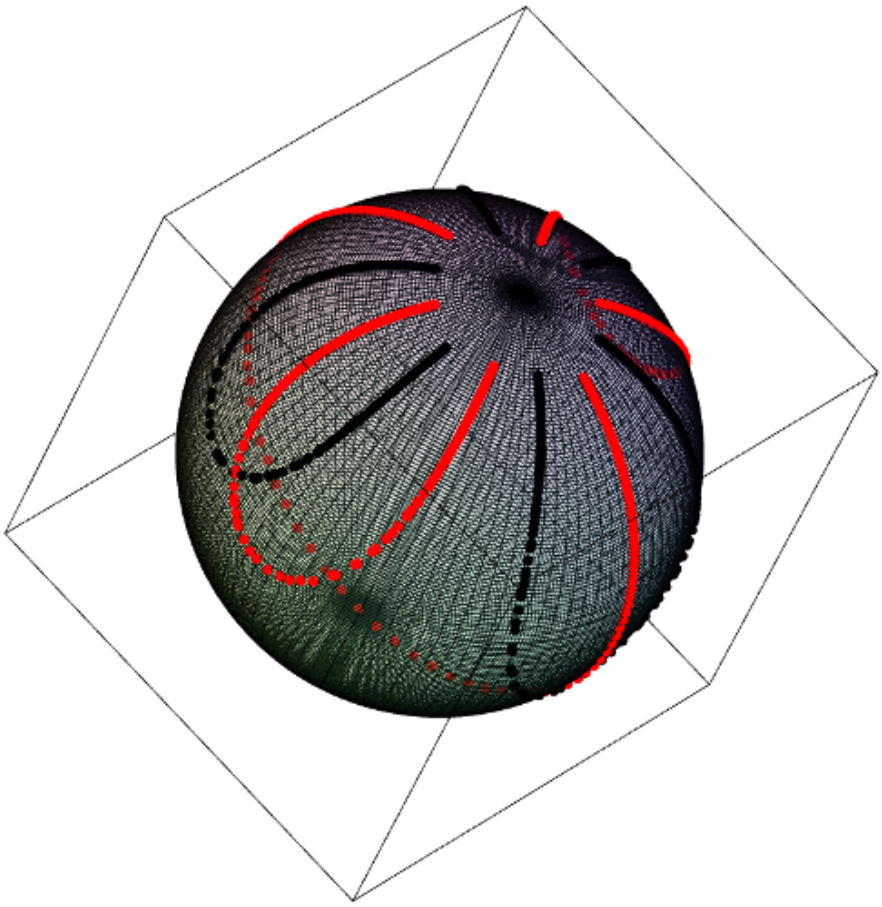
erhalten wir folgendes Bild.



Letzteres ist weniger schön, läßt sich aber besser auf die Riemannsche Zahlenkugel transportieren. Dies geschieht wie folgt.

```
sphere := [seq([seq([Re(exp(2*Pi*I*s)*cos(Pi/2*t)),
                     Im(exp(2*Pi*I*s)*cos(Pi/2*t)),sin(Pi/2*t)],
                             s = 0..1, 0.01)],t = -1..1, 0.01)]:

proj := (ure,uim) -> 1/(ure^2 + uim^2 + 1) * [2 * ure, 2 * uim, ure^2 + uim^2 - 1];

ptre := NULL:
for k from 1 to 3 do
 ptre_inn := NULL:
 for l from 1 to rowdim(Matrix(pt_ipre[k])) do
  conv := convert(convert(pt_ipre[k],matrix),list):
  ptre_inn := ptre_inn,map(proj,op(conv[l])):
 od;
 ptre := ptre,[ptre_inn]:
od;
ptre_list := [ptre]:

ptim := NULL:
for k from 1 to 3 do
 ptim_inn := NULL:
 for l from 1 to rowdim(Matrix(pt_ipim[k])) do
  conv := convert(convert(pt_ipim[k],matrix),list):
  ptim_inn := ptim_inn,map(proj,op(conv[l])):
 od;
 ptim := ptim,[ptim_inn]:
od;
ptim_list := [ptim]:

display(
 surfdata(sphere, axes = boxed, tickmarks = [0,0,0], transparency = 0.8)
,pointplot3d(ptre_list[1], symbol = solidcircle, symbolsize = 10, color = "black")
,pointplot3d(ptre_list[2], symbol = solidcircle, symbolsize = 10, color = "black")
,pointplot3d(ptre_list[3], symbol = solidcircle, symbolsize = 10, color = "black")
,pointplot3d(ptim_list[1], symbol = solidcircle, symbolsize = 10, color = "red")
,pointplot3d(ptim_list[2], symbol = solidcircle, symbolsize = 10, color = "red")
,pointplot3d(ptim_list[3], symbol = solidcircle, symbolsize = 10, color = "red")
);
```

Dies liefert nach Export als “Extensible 3D” folgendes Bild.



Siehe auch Galerie 1, Figur 2.

5 Diskriminanteneinbettung

Es war $\mathrm{P}^2(\mathbf{C})$ wie folgt konstruiert. Elemente (u', v', w'), $(u'', v'', w'') \in \mathbf{C}^3 \setminus \{(0,0,0)\}$ seien äquivalent, wenn es ein $\lambda \in \mathbf{C} \setminus \{0\}$ gibt mit $\lambda \cdot (u', v', w') = (u'', v'', w'')$. Die Äquivalenzklasse eines solchen Elements (u, v, w) werde $(u : v : w)$ geschrieben. Kurz, für $(u, v, w) \in \mathbf{C}^3 \setminus \{(0,0,0)\}$ ist

$$(u : v : w) \;=\; (\lambda u : \lambda v : \lambda w)$$

für $\lambda \in \mathbf{C} \setminus \{0\}$.

Die naheliegende Verallgemeinerung der Abbildung

$$\begin{array}{rcccc} \mathrm{P}^1(\mathbf{C}) & \to & \mathbf{C} & \times & \mathbf{R} \\ (z : w) & \mapsto & \frac{1}{|z|^2+|w|^2}(2z\bar{w} & , & |z|^2 - |w|^2) \end{array}$$

aus §4 von $\mathrm{P}^1(\mathbf{C})$ nach $\mathrm{P}^2(\mathbf{C})$ ist folgende Abbildung, Diskriminanteneinbettung genannt.

$$\begin{array}{rcccccccccccc} \mathrm{P}^2(\mathbf{C}) & \to & & & \mathbf{C} & \times & \mathbf{C} & \times & \mathbf{C} & \times & \mathbf{R} & \times & \mathbf{R} & \times & \mathbf{R} \\ (u:v:w) & \mapsto & \frac{1}{|u|^2+|v|^2+|w|^2}(& 2u\bar{v} & , & 2u\bar{w} & , & 2v\bar{w} & , & |u|^2-|v|^2 & , & |u|^2-|w|^2 & , & |v|^2-|w|^2) \end{array}$$

Man beachte, daß diese Definition repräsentantenunabhängig ist.

Wir identifizieren $\mathbf{C}\times\mathbf{C}\times\mathbf{C}\times\mathbf{R}\times\mathbf{R}\times\mathbf{R}$ mit $\mathbf{R}^9$ via

$$(x,y,z,a,b,c) \;\mapsto\; (\mathrm{Re}(x),\mathrm{Im}(x),\mathrm{Re}(y),\mathrm{Im}(y),\mathrm{Re}(z),\mathrm{Im}(z),a,b,c)\ .$$

Es ist das Bild unserer Abbildung in der Sphäre in $\mathbf{R}^9$ mit Radius 1 enthalten, denn

$$\frac{1}{(|u|^2+|v|^2+|w|^2)^2}(|2u\bar{v}|^2+|2u\bar{w}|^2+|2v\bar{w}|^2+(|u|^2-|v|^2)^2+(|u|^2-|w|^2)^2+(|v|^2-|w|^2)^2) \;=\; 1\ .(^1)$$

Mittels einer Projektion $\mathbf{R}^9\to\mathbf{R}^3$ können wir also ganz $\mathrm{P}^2(\mathbf{C})$ in einem beschränkten Bereich betrachten und damit auch jede Teilmenge von $\mathrm{P}^2(\mathbf{C})$, wie zum Beispiel die Nullstellenmenge einer Gleichung.

Zeigen wir einmal die Injektivität unserer Abbildung.

Seien $(u':v':w')$ und $(u'':v'':w'')$ gegeben, die auf dasselbe Element abgebildet werden. O.E. ist $|u'|^2+|v'|^2+|w'|^2=|u''|^2+|v''|^2+|w''|^2$. Es ist also

$$\begin{array}{rccccccc} & (2u'\bar{v}', & 2u'\bar{w}', & 2v'\bar{w}', & |u'|^2-|v'|^2, & |u'|^2-|w'|^2, & |v'|^2-|w'|^2) \\ = & (2u''\bar{v}'', & 2u''\bar{w}'', & 2v''\bar{w}'', & |u''|^2-|v''|^2, & |u''|^2-|w''|^2, & |v''|^2-|w''|^2)\ . \end{array}$$

Zu zeigen ist $(u':v':w')\stackrel{!}{=}(u'':v'':w'')$.

Fall $u'\neq 0$, $v'\neq 0$, $w'\neq 0$. Wegen $0\neq 2u'\bar{v}'=2u''\bar{v}''$ und $0\neq 2u'\bar{w}'=2v''\bar{w}''$ ist auch $u''\neq 0$, $v''\neq 0$, $w''\neq 0$.

Schreibe $u_1:=u''/u'$, $v_1:=v''/v'$ und $w_1:=w''/w'$. Es genügt, $u_1\stackrel{!}{=}v_1\stackrel{!}{=}w_1$ zu zeigen.

Es ist $u_1\bar{v}_1=1$, $u_1\bar{w}_1=1$, $v_1\bar{w}_1=1$. Also ist in der Tat $u_1=v_1=w_1$.

Fall $u'\neq 0$, $v'\neq 0$, $w'=0$.

Es ist $u''\neq 0$ und $v''\neq 0$. Folglich ist $w''\neq 0$.

Schreibe $u_1:=u''/u'$ und $v_1:=v''/v'$. Zu zeigen genügt $u_1\stackrel{!}{=}v_1$.

Es ist $u_1\bar{v}_1=1$ und, wegen $|u'|^2=|u''|^2$, auch $u_1\bar{u}_1=1$. Also ist $\bar{u}_1=\bar{v}_1$.

Voriger Fall deckt wegen Symmetrie auch den Fall $u'\neq 0$, $v'=0$, $w'\neq 0$ und den Fall $u'=0$, $v'\neq 0$, $w'\neq 0$ ab.

---

[1] Juni 2020: Hier ist ein Rechenfehler enthalten, der dazu führt, daß das Bild in einer gestreckten Sphäre liegt. Will man, daß es in der Sphäre mit Radius 1 enthalten ist, muß man in der Abbildungsvorschrift einen Faktor $\sqrt{2}$ im Nenner des Vorfaktors ergänzen und den Faktor 2 in den ersten drei Einträgen ersetzen durch $\sqrt{6}$.

Fall $u' \neq 0$, $v' = 0$, $w' = 0$.

Wäre $v'' \neq 0$, dann wäre $u'' = 0$ und $w'' = 0$, was *nicht* geht.

Wäre $w'' \neq 0$, dann wäre $u'' = 0$ und $v'' = 0$, was *nicht* geht.

Also ist $u'' \neq 0$, $v'' = 0$, $w'' = 0$. Folglich ist $(u' : v' : w') = (u'' : v'' : w'')$.

Voriger Fall deckt wegen Symmetrie auch den Fall $u' = 0$, $v' \neq 0$, $w' = 0$ *und den Fall* $u' = 0$, $v' = 0$, $w' \neq 0$ *ab.*

Somit ist die Injektivität unserer Abbildung gezeigt.

6 Der Graph direkt projiziert

Folgendes Programm hat als Input eine Matrix `M` $\in \mathbf{R}^{m \times n}$, deren Spalten eine Basis eines Unterraums $U \subseteq \mathbf{R}^m$ sein sollen. Als Ausgabe `orth(M)` erhalten wir eine Matrix, deren Spalten eine Orthonormalbasis von U sind. Diese können wir also zur orthogonalen Projektion auf diesen Unterraum verwenden.

```
orth := proc(M)
 local Q0;
 QRdecomp(M,Q = Q0, fullspan = false);
 return map(evalf,Q0);
end proc;
```

In Galerie 2 betrachten wir die Graphen folgender Funktionen.

```
g1 := x -> x;
g2 := x -> x^2;
g3 := x -> exp(x);
g4 := x -> GAMMA(x-0.01); # kleine Verschiebung, um Singularitaeten nur anzunaehern
g5 := x -> Zeta(x-0.01);  # kleine Verschiebung, um Singularitaet nur anzunaehern
```

Hier konzentrieren wir uns auf `g2`. Zunächst wollen wir die Diskriminanteneinbettung nicht verwenden und direkt den Graphen von `g` als Teilmenge von $\mathbf{C}^2 = \mathbf{R}^4$ betrachten.

```
g := g2;
f := x -> matrix([[Re(x),Im(x),Re(g(x)),Im(g(x))]]);
```

Wir projizieren mittels folgender Matrix `P`.

```
submat := # Spalten spannen Teilraum auf
Matrix([
[ 1, 0, 0],
[ 0, 1, 0],
[ 0, 0, 1],
[ 0, 0, 1]
```

```
]):

P := orth(submat):
composite := x -> convert(convert(evalm(f(x) &* P),vector),list);
```

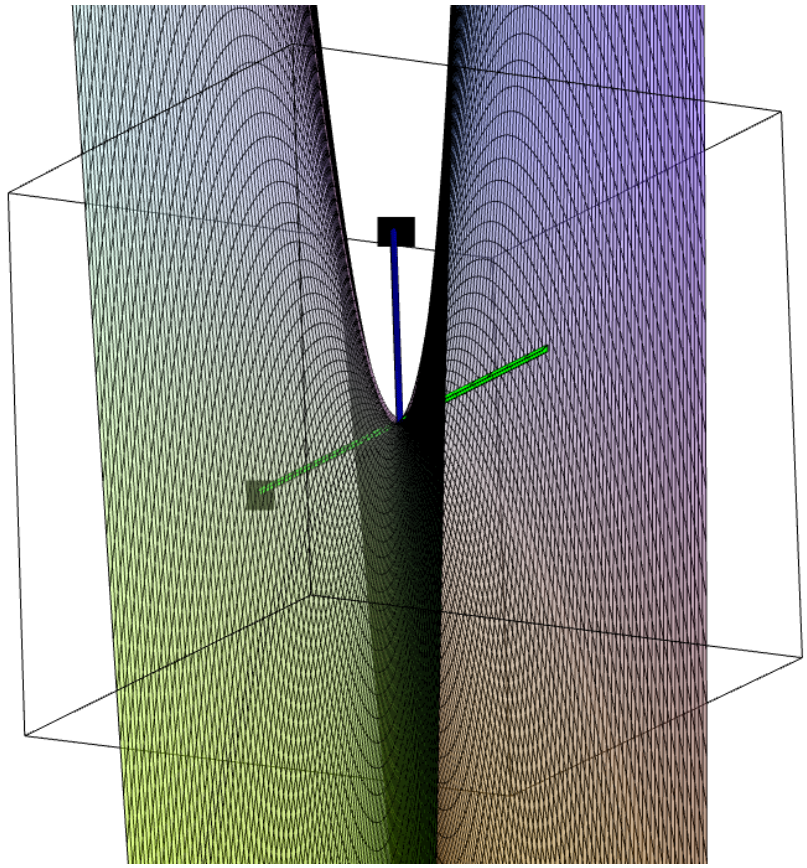
Auch die Achsen sollen eingezeichnet werden. Folgende Funktionen sind für reelles `x` gedacht.

```
axis_arg := x -> convert(convert(evalm(matrix([[x, 0, 0, 0]]) &* P),vector),list);
axis_val := x -> convert(convert(evalm(matrix([[0, 0, x, 0]]) &* P),vector),list);
```

Wir wählen als Definitionsbereich ein Gitter mit Schrittweite `1/st`, das die komplexen Zahlen mit Realteil und Imaginärteil zwischen -8 und 8 durchläuft.

```
netsize_im := 80 * 1/st:
netsize_re := 80 * 1/st:
st :=  10:
red := 1: # reduziert Achsenlaenge, passend justieren
points := [seq([seq(composite(u + I*v), u = -netsize_re..+netsize_re, 1/st)],
                                        v = -netsize_im..+netsize_im, 1/st)]:
points_real_axis_arg := [[seq(axis_arg(u),           u = -netsize_re..+netsize_re)],
                         [seq(axis_arg(u)+[0,0,0.2], u = -netsize_re..+netsize_re)],
         [seq(axis_arg(u)+[0,0.2,0], u = -netsize_re..+netsize_re)]]:
                                       # +[0,0,0.2] etc. = "poor man's bold"
points_real_axis_val := [[seq(axis_val(v),           v = -netsize_re/red..+netsize_re/red,1/st)],
                         [seq(axis_val(v)+[0,0,0.2], v = -netsize_re/red..+netsize_re/red,1/st)],
         [seq(axis_val(v)+[0,0.2,0], v = -netsize_re/red..+netsize_re/red,1/st)]]:
points_real_dir    := [[seq(composite(u),           u = -netsize_re..+netsize_re, 1/st)],
                       [seq(composite(u)+[0,0,0.2], u = -netsize_re..+netsize_re, 1/st)],
       [seq(composite(u)+[0,0.2,0], u = -netsize_re..+netsize_re, 1/st)]]:
labeltext := {[op(axis_arg(netsize_re)),"x"]
             ,[op(axis_val(netsize_re/red)),"xx"]
      }:

display(
 surfdata(points, axes = boxed, transparency = 0.5, view = -8..8)
,surfdata(points_real_dir, color = "violet", axes = boxed, tickmarks = [0,0,0], transparency = 0)
,surfdata(points_real_axis_arg, color = "green", axes = boxed, tickmarks = [0,0,0], transparency = 0)
,surfdata(points_real_axis_val, color = "blue",  axes = boxed, tickmarks = [0,0,0], transparency = 0)
,textplot3d(labeltext)
);
```

Dies liefert folgendes Bild; cf. Galerie 2, Figur 3.2.



7 Der Graph eingebettet und dann projiziert

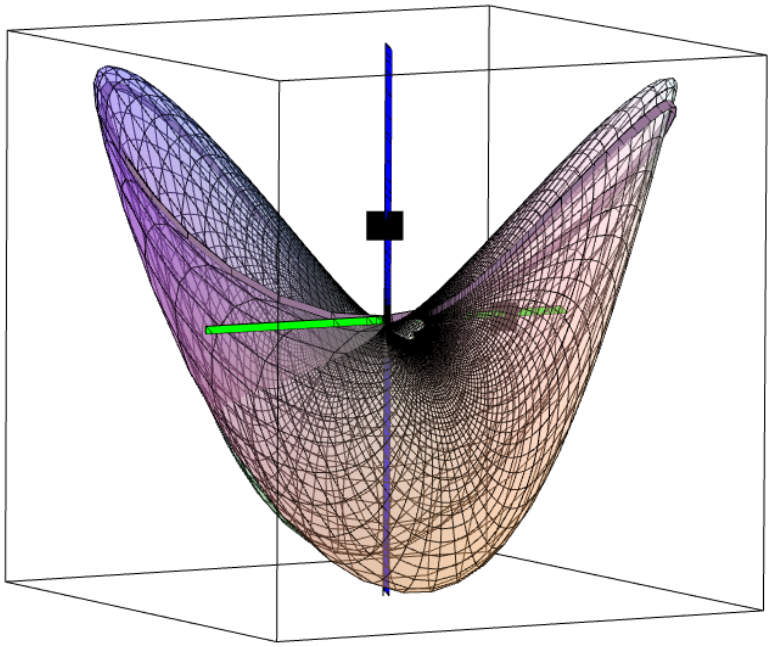
Nun versuchen wir, mittels der Diskriminanteneinbettung aus §5 die Graphiken nicht ins Unendliche laufen zu lassen. Der Effekt ist vergleichbar mit dem eines übertriebenen Fisheye-Objektivs.

```
orth := proc(M)
 local Q0;
 QRdecomp(M,Q = Q0, fullspan = false);
 return map(evalf,Q0);
end proc;

g1 := x -> x;
g2 := x -> x^2;
g3 := x -> exp(x);
g4 := x -> GAMMA(x-0.01); # kleine Verschiebung, um Singularitaeten nur anzunaehern
g5 := x -> Zeta(x-0.01);  # kleine Verschiebung, um Singularitaeten nur anzunaehern

g := g2;
```

```
lengthsq := x -> abs(x)^2 + abs(g(x))^2 + 1;

f := x -> evalm(1/lengthsq(x) * matrix([[Re(2*x*conjugate(g(x))), Im(2*x*conjugate(g(x))),
                                        Re(2*x), Im(2*x),
                                        Re(2*g(x)), Im(2*g(x)),
                                        abs(x)^2 - abs(g(x))^2, abs(x)^2 - 1, abs(g(x))^2 - 1]]));

submat := # Spalten spannen Teilraum auf
Matrix([
[ 2, 0, 0],
[ 1, 0, 0],
[ 0, 2, 0],
[ 0, 1, 0],
[ 0, 0, 2],
[ 0, 0, 1],
[ 0, 0, 0],
[ 0, 0, 0],
[ 0, 0, 0]
]):

P := orth(submat):

composite := x -> convert(convert(evalm(f(x) &* P),vector),list);
axis_arg := x -> convert(convert(evalm(1/(x^2 + 1)
                 * matrix([[0, 0, 2*x, 0, 0,   0, abs(x)^2,  abs(x)^2 - 1, -1]]) &* P),vector),list);
axis_val := x -> convert(convert(evalm(1/(x^2 + 1)
                 * matrix([[0, 0, 0,   0, 2*x, 0, -abs(x)^2, -1, abs(x)^2 - 1]]) &* P),vector),list);

# ohne Verbindung zum unendlichen Punkt, mit reellen Achsen

netsize_im := 80 * 1/st:
netsize_re := 80 * 1/st:
st :=  10:
red := 0.4: # reduziert Achsenlaenge, passend justieren
points := [seq([seq(composite(u + I*v), u = -netsize_re..+netsize_re, 1/st)],
                                        v = -netsize_im..+netsize_im, 1/st)]:
points_real_axis_arg := [[seq(axis_arg(u),              u = -netsize_re..+netsize_re)],
                         [seq(axis_arg(u)+[0,0,0.02],   u = -netsize_re..+netsize_re)],
         [seq(axis_arg(u)+[0,0.02,0],   u = -netsize_re..+netsize_re)]]:
points_real_axis_val := [[seq(axis_val(v),              v = -netsize_re/red..+netsize_re/red,1/st)],
                         [seq(axis_val(v)+[0,0,0.02],   v = -netsize_re/red..+netsize_re/red,1/st)],
         [seq(axis_val(v)+[0,0.02,0],   v = -netsize_re/red..+netsize_re/red,1/st)]]:
points_real_dir    := [[seq(composite(u),           u = -netsize_re..+netsize_re, 1/st)],
                       [seq(composite(u)+[0,0,0.02], u = -netsize_re..+netsize_re, 1/st)],
      [seq(composite(u)+[0,0.02,0], u = -netsize_re..+netsize_re, 1/st)]]:
labeltext := {[op(axis_arg(netsize_re/5)),"x"]
             ,[op(axis_val(netsize_re/red/10)),"xx"]
      }:
display(
 surfdata(points, axes = boxed, transparency = 0.5)
,surfdata(points_real_dir, color = "violet", axes = boxed, tickmarks = [0,0,0],
                                                           transparency = 0)
,surfdata(points_real_axis_arg, color = "green", axes = boxed, tickmarks = [0,0,0],
```

```
                                                                    transparency = 0)
,surfdata(points_real_axis_val, color = "blue",  axes = boxed, tickmarks = [0,0,0],
                                                                    transparency = 0)
,textplot3d(labeltext)
);
```

Dies liefert folgende Graphik; cf. Galerie 2, Figur 4.2.



Das dort erkennbare Loch in der Oberfläche ist entstanden, weil wir uns nicht um Punkte im Unendlichen gekümmert haben, sondern nur das Argument x einigermaßen betragsgroß haben werden lassen.

8 Eine elliptische Kurve

Nun wollen wir die Teilmenge

$$\{\,(x,y) \in \mathbf{C}^2 :\ y^2 = (x-1)(x+1)(x+3)\,\} \ \subseteq\ \mathbf{C}^2$$

betrachten. Um die unendlichen Punkte ergänzt, ist wird dies zur Menge

$$\{\,(u:v:w) \in \mathrm{P}^2(\mathbf{C}) :\ v^2w = (u-w)(u+w)(u+3w)\,\} \ \subseteq \mathrm{P}^2(\mathbf{C})$$

Dazu wollen wir uns wieder der Diskriminanteneinbettung aus §5 bedienen. Danach soll wieder nach $\mathbf{R}^3$ projiziert werden, um das Resultat betrachten zu können.

Für die Projektion brauchen wir wieder die Funktion aus §6.

```
orth := proc(M)
 local Q0;
 QRdecomp(M,Q = Q0, fullspan = false);
 return map(evalf,Q0);
end proc;
```

Wir wollen die Punkte im Endlichen mittels $y = \pm\sqrt{(x-1)(x+1)(x+3)}$ berechnen. Die `sqrt`-Funktion hat aber einen Sprung beim Argument π: `sqrt(-1)` gibt `I`, aber `sqrt(-1 - 0.01*I)` gibt `0.004999937503 - 1.000012500*I`. Wir sollten aber `sqrt` sprungfrei zur Anwendung bringen. Dazu erstellen wir eine Wurzelfunktion, die einen Sprung beim Argument $2\pi t$ hat.

```
sqrt_rel := (x,t) -> evalf(exp(I*(2*Pi*t + Pi)/2) * sqrt(x * exp(-I*(2*Pi*t + Pi))));
```

Wir kürzen ab:

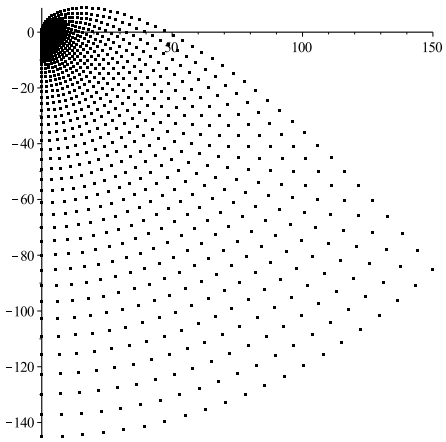
```
radicand := x -> (x-1) * (x+1) * (x+3);
```

Wir sollten nun den Definitionsbereich so unterteilen, daß wir in jedem Teil eine Wurzelfunktion anwenden können, die, wenn auf das Bild von `radicand` angewandt, keinen Sprung hervorbringt. Wir müssen also einen Teilbereich des Definitionsbereichs wählen, das Bild dieses Teilbereichts unter `radicand` von Maple graphisch ausgeben lassen und dann t so wählen, daß der Ursprungsstrahl in Richtung $2\pi t$ dieses Bild nicht schneidet.

So etwa liefert

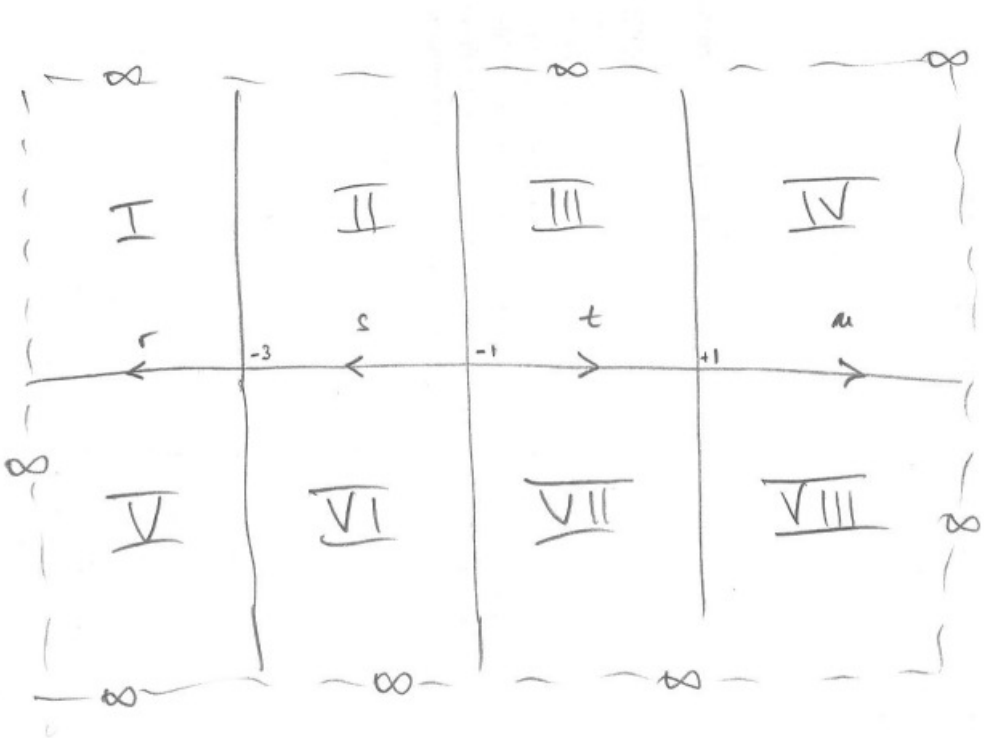
```
pointplot([seq(seq([Re(radicand(u + I*v)),Im(radicand(u + I*v))],
                u = -3..-1, 0.1), v =  0..5, 0.1)], symbol = point);
```

das Bild



Somit ist hier e.g. der Parameter $t = 0{,}5$ in Ordnung auf dem Teilbereich des Definitionsbereichs, der aus den komplexen Zahlen `u + I*v` besteht mit `u` $\in [-2, 0]$ und `v` $\in [0, 5]$, was uns gut genug ist, um dies auch für `u` $\in [-2, 0]$ und `v` $\in [0, +\infty)$ zu behaupten.

Insgesamt betrachten wir so folgende Unterteilung.




```
pointplot([seq(seq([Re(radicand(u + I*v)),Im(radicand(u + I*v))],
                u = -5..-3, 0.1), v =  0..+5, 0.1)], symbol = point); # gibt t = 0.75
pointplot([seq(seq([Re(radicand(u + I*v)),Im(radicand(u+I*v))],
                u = -3..-1, 0.1), v =  0..+5, 0.1)], symbol = point); # gibt t = 0.5
pointplot([seq(seq([Re(radicand(u + I*v)),Im(radicand(u+I*v))],
                u = -1..+1, 0.1), v =  0..+5, 0.1)], symbol = point); # gibt t = 0.0
pointplot([seq(seq([Re(radicand(u + I*v)),Im(radicand(u + I*v))],
                u = +1..+5, 0.1), v =  0..+5, 0.1)], symbol = point); # gibt t = 0.75
pointplot([seq(seq([Re(radicand(u + I*v)),Im(radicand(u + I*v))],
                u = -5..-3, 0.1), v = -5.. 0, 0.1)], symbol = point); # gibt t = 0.25
pointplot([seq(seq([Re(radicand(u + I*v)),Im(radicand(u+I*v))],
                u = -3..-1, 0.1), v = -5.. 0, 0.1)], symbol = point); # gibt t = 0.5
pointplot([seq(seq([Re(radicand(u + I*v)),Im(radicand(u+I*v))],
                u = -1..+1, 0.1), v = -5.. 0, 0.1)], symbol = point); # gibt t = 0.0
pointplot([seq(seq([Re(radicand(u + I*v)),Im(radicand(u + I*v))],
                u = +1..+5, 0.1), v = -5.. 0, 0.1)], symbol = point); # gibt t = 0.25
```

Will man noch spezielle Kurven auf der Oberfläche auszeichnen, dann kann man auch noch dafür den Parameter t abfragen. Folgendes ist e.g. die Abfrage für das Bild der reelle Achse sowie für das Bild der Geraden $\{\, -1 + \mathrm{i}v : \ v \in \mathbf{R} \,\}$.

```
pointplot([seq([Re(radicand(u)),Im(radicand(u))],
                   u =  -5.. 5, 0.1)], symbol = point); # gibt t = 0.25
pointplot([seq([Re(radicand(-1 + I*v)),Im(radicand(-1 + I*v))],
                   v =  -5.. 5, 0.1)], symbol = point); # gibt t = 0.5
```

Natürlich muß mit einer Wurzelfunktion, die für einen Teilbereich geeignet ist, auch immer noch ihr Negatives mitberücksichtigt werden. Wir kürzen wie folgt ab.

```
ecp0 := x -> +sqrt_rel(radicand(x), 0.00);
ecm0 := x -> -sqrt_rel(radicand(x), 0.00);
ecp1 := x -> +sqrt_rel(radicand(x), 0.25);
ecm1 := x -> -sqrt_rel(radicand(x), 0.25);
ecp2 := x -> +sqrt_rel(radicand(x), 0.50);
ecm2 := x -> -sqrt_rel(radicand(x), 0.50);
ecp3 := x -> +sqrt_rel(radicand(x), 0.75);
ecm3 := x -> -sqrt_rel(radicand(x), 0.75);
ecas := x -> abs(evalf(radicand(x)));
```

Nun die Funktionen, die auf den einzelnen Teilen des Definitionsbereichs gebraucht werden.

```
lengthsq := x -> abs(evalf(x))^2 + ecas(x) + 1;
fp0 := x -> evalm(1/lengthsq(x) * matrix([[Re(2*x*conjugate(ecp0(x))), Im(2*x*conjugate(ecp0(x))),
Re(2*x), Im(2*x), Re(2*ecp0(x)), Im(2*ecp0(x)), abs(x)^2 - ecas(x), abs(x)^2 - 1, ecas(x) - 1]]));
fm0 := x -> evalm(1/lengthsq(x) * matrix([[Re(2*x*conjugate(ecm0(x))), Im(2*x*conjugate(ecm0(x))),
Re(2*x), Im(2*x), Re(2*ecm0(x)), Im(2*ecm0(x)), abs(x)^2 - ecas(x), abs(x)^2 - 1, ecas(x) - 1]]));
fp1 := x -> evalm(1/lengthsq(x) * matrix([[Re(2*x*conjugate(ecp1(x))), Im(2*x*conjugate(ecp1(x))),
Re(2*x), Im(2*x), Re(2*ecp1(x)), Im(2*ecp1(x)), abs(x)^2 - ecas(x), abs(x)^2 - 1, ecas(x) - 1]]));
fm1 := x -> evalm(1/lengthsq(x) * matrix([[Re(2*x*conjugate(ecm1(x))), Im(2*x*conjugate(ecm1(x))),
Re(2*x), Im(2*x), Re(2*ecm1(x)), Im(2*ecm1(x)), abs(x)^2 - ecas(x), abs(x)^2 - 1, ecas(x) - 1]]));
fp2 := x -> evalm(1/lengthsq(x) * matrix([[Re(2*x*conjugate(ecp2(x))), Im(2*x*conjugate(ecp2(x))),
Re(2*x), Im(2*x), Re(2*ecp2(x)), Im(2*ecp2(x)), abs(x)^2 - ecas(x), abs(x)^2 - 1, ecas(x) - 1]]));
fm2 := x -> evalm(1/lengthsq(x) * matrix([[Re(2*x*conjugate(ecm2(x))), Im(2*x*conjugate(ecm2(x))),
Re(2*x), Im(2*x), Re(2*ecm2(x)), Im(2*ecm2(x)), abs(x)^2 - ecas(x), abs(x)^2 - 1, ecas(x) - 1]]));
fp3 := x -> evalm(1/lengthsq(x) * matrix([[Re(2*x*conjugate(ecp3(x))), Im(2*x*conjugate(ecp3(x))),
Re(2*x), Im(2*x), Re(2*ecp3(x)), Im(2*ecp3(x)), abs(x)^2 - ecas(x), abs(x)^2 - 1, ecas(x) - 1]]));
fm3 := x -> evalm(1/lengthsq(x) * matrix([[Re(2*x*conjugate(ecm3(x))), Im(2*x*conjugate(ecm3(x))),
Re(2*x), Im(2*x), Re(2*ecm3(x)), Im(2*ecm3(x)), abs(x)^2 - ecas(x), abs(x)^2 - 1, ecas(x) - 1]]));
```

Projiziert werden muß auch wieder.

```
submat := # Spalten spannen Teilraum auf
Matrix([
[ 1, 0, 0],
[ 1, 0, 0],
[ 0, 1, 0],
[ 0, 1, 0],
[ 0, 0, 1],
[ 0, 0, 1],
[ 0, 0, 0],
[ 0, 0, 0],
[ 0, 0, 0]
]):

P := orth(submat);

composite_p0 := x -> convert(convert(evalm(fp0(x) &* P),vector),list);
composite_m0 := x -> convert(convert(evalm(fm0(x) &* P),vector),list);
composite_p1 := x -> convert(convert(evalm(fp1(x) &* P),vector),list);
composite_m1 := x -> convert(convert(evalm(fm1(x) &* P),vector),list);
composite_p2 := x -> convert(convert(evalm(fp2(x) &* P),vector),list);
composite_m2 := x -> convert(convert(evalm(fm2(x) &* P),vector),list);
composite_p3 := x -> convert(convert(evalm(fp3(x) &* P),vector),list);
composite_m3 := x -> convert(convert(evalm(fm3(x) &* P),vector),list);
```

Netzgröße und -feinheit, Punkte im Unendlichen und die reellen Achsen werden spezifiziert.

```
netsize_im := 160 * 1/st:
netsize_re := 160 * 1/st:
st := 20:
point_infinity :=
convert(convert(evalm(Matrix([[0,0,0,0,0,0,-1,0,1]]) &* P),vector),list); # (u:v:w) = (0:1:0)
point_infinity_2 :=
convert(convert(evalm(Matrix([[0,0,0,0,0,0, 1,1,0]]) &* P),vector),list); # (u:v:w) = (1:0:0)
axis_arg := x -> convert(convert(evalm(1/(x^2 + 1)
* matrix([[0, 0, 2*x, 0, 0,
 0, x^2, x^2 - 1, -1]]) &* P),vector),list); # Bild von (x:0:1), x reell
axis_val := x -> convert(convert(evalm(1/(x^2 + 1)
* matrix([[0, 0, 0,
 0, 2*x, 0, -x^2, -1, x^2 - 1]]) &* P),vector),list); # Bild von (0:x:1), x reell
points_infinity_1 := [point_infinity, seq(point_infinity, u = -netsize_re..-3, 1/st)]:
points_infinity_2 := [seq(point_infinity, u = -3..-1, 1/st)]:
points_infinity_3 := [seq(point_infinity, u = -1..+1, 1/st)]:
points_infinity_4 := [seq(point_infinity, u = +1..+netsize_re, 1/st), point_infinity]:
```

Nun zur Berechnung der Punkte, einiger Hilfslinien und der improvisierten Achslabels.

```
points_p_i :=
[seq([point_infinity, seq(composite_p3(u + I*v), u = -netsize_re..-3, 1/st)],
v = 0..+netsize_im, 1/st), points_infinity_1]:
points_p_ii :=
[seq([seq(composite_p2(u + I*v), u = -3..-1, 1/st)],
v = 0..+netsize_im, 1/st), points_infinity_2]:
points_p_iii :=
[seq([seq(composite_p0(u + I*v), u = -1..+1, 1/st)],
v = 0..+netsize_im, 1/st), points_infinity_3]:
points_p_iv :=
[seq([seq(composite_p3(u + I*v), u = +1..+netsize_re, 1/st), point_infinity],
v = 0..+netsize_im, 1/st), points_infinity_4]:
points_p_v :=
[points_infinity_1, seq([point_infinity, seq(composite_p1(u + I*v), u = -netsize_re..-3, 1/st)],
v = -netsize_im..0, 1/st)]:
points_p_vi :=
[points_infinity_2, seq([seq(composite_p2(u + I*v), u = -3..-1, 1/st)],
v = -netsize_im..0, 1/st)]:
points_p_vii :=
[points_infinity_3, seq([seq(composite_p0(u + I*v), u = -1..+1, 1/st)],
v = -netsize_im..0, 1/st)]:
points_p_viii :=
[points_infinity_4, seq([seq(composite_p1(u + I*v), u = +1..+netsize_re, 1/st), point_infinity],
v = -netsize_im..0, 1/st)]:
points_m_i :=
[seq([point_infinity, seq(composite_m3(u + I*v), u = -netsize_re..-3, 1/st)],
v = 0..+netsize_im, 1/st), points_infinity_1]:
points_m_ii :=
[seq([seq(composite_m2(u + I*v), u = -3..-1, 1/st)],
v = 0..+netsize_im, 1/st), points_infinity_2]:
points_m_iii :=
[seq([seq(composite_m0(u + I*v), u = -1..+1, 1/st)],
v = 0..+netsize_im, 1/st), points_infinity_3]:
points_m_iv :=
[seq([seq(composite_m3(u + I*v), u = +1..+netsize_re, 1/st), point_infinity],
v = 0..+netsize_im, 1/st), points_infinity_4]:
points_m_v :=
[points_infinity_1, seq([point_infinity, seq(composite_m1(u + I*v), u = -netsize_re..-3, 1/st)],
v = -netsize_im..0, 1/st)]:
points_m_vi :=
[points_infinity_2, seq([seq(composite_m2(u + I*v), u = -3..-1, 1/st)],
v = -netsize_im..0, 1/st)]:
points_m_vii :=
[points_infinity_3, seq([seq(composite_m0(u + I*v), u = -1..+1, 1/st)],
v = -netsize_im..0, 1/st)]:
points_m_viii :=
[points_infinity_4, seq([seq(composite_m1(u + I*v), u = +1..+netsize_re, 1/st), point_infinity],
v = -netsize_im..0, 1/st)]:
```

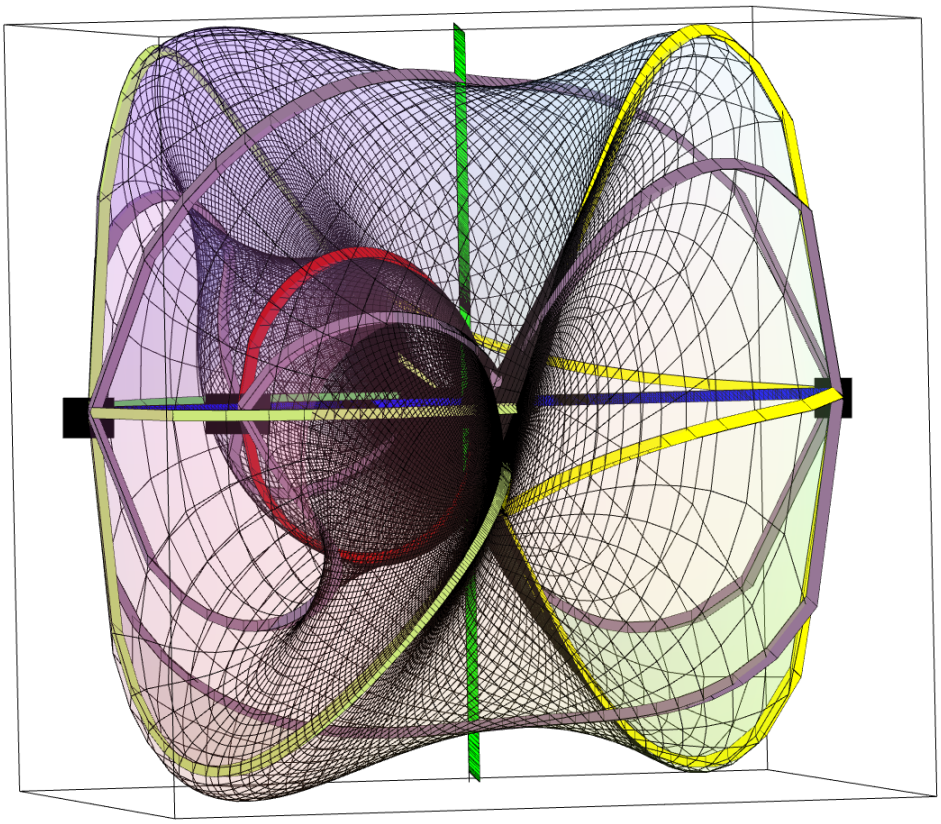
```
pmb1 := [0,0,0.02]: # +[0,0,0.02] etc. = poor man's bold
pmb2 := [0,0.02,0]:
points_real_axis_arg :=
[[point_infinity_2, seq(axis_arg(u),      u = -netsize_re..+netsize_re,1/st), point_infinity_2],
 [point_infinity_2, seq(axis_arg(u)+pmb1, u = -netsize_re..+netsize_re,1/st), point_infinity_2],
 [point_infinity_2, seq(axis_arg(u)+pmb2, u = -netsize_re..+netsize_re,1/st), point_infinity_2]]:
points_real_axis_val :=
[[point_infinity, seq(axis_val(v),      v = -netsize_re..+netsize_re,1/st), point_infinity],
 [point_infinity, seq(axis_val(v)+pmb1, v = -netsize_re..+netsize_re,1/st), point_infinity],
 [point_infinity, seq(axis_val(v)+pmb2, v = -netsize_re..+netsize_re,1/st), point_infinity]]:
points_real_dir_p :=
[[point_infinity, seq(composite_p1(u),      u = -netsize_re..+netsize_re, 1/st), point_infinity],
 [point_infinity, seq(composite_p1(u)+pmb1, u = -netsize_re..+netsize_re, 1/st), point_infinity],
 [point_infinity, seq(composite_p1(u)+pmb2, u = -netsize_re..+netsize_re, 1/st), point_infinity]]:
points_real_dir_m :=
[[point_infinity, seq(composite_m1(u), u = -netsize_re..+netsize_re, 1/st), point_infinity],
 [point_infinity, seq(composite_m1(u)+pmb1, u = -netsize_re..+netsize_re, 1/st), point_infinity],
 [point_infinity, seq(composite_m1(u)+pmb2, u = -netsize_re..+netsize_re, 1/st), point_infinity]]:
points_im_dir_p_1 :=
[[seq(composite_p3(1 + I*v),      v = -netsize_im..+netsize_im, 1/st), point_infinity],
 [seq(composite_p3(1 + I*v)+pmb1, v = -netsize_im..+netsize_im, 1/st), point_infinity],
 [seq(composite_p3(1 + I*v)+pmb2, v = -netsize_im..+netsize_im, 1/st), point_infinity]]:
points_im_dir_m_1 :=
[[seq(composite_m3(1 + I*v),      v = -netsize_im..+netsize_im, 1/st), point_infinity],
 [seq(composite_m3(1 + I*v)+pmb1, v = -netsize_im..+netsize_im, 1/st), point_infinity],
 [seq(composite_m3(1 + I*v)+pmb2, v = -netsize_im..+netsize_im, 1/st), point_infinity]]:
points_im_dir_p_n1o := # n = negative
[[point_infinity, seq(composite_p1(-1 + I*v),      v = 0..+netsize_im, 1/st), point_infinity],
 [point_infinity, seq(composite_p1(-1 + I*v)+pmb1, v = 0..+netsize_im, 1/st), point_infinity],
 [point_infinity, seq(composite_p1(-1 + I*v)+pmb2, v = 0..+netsize_im, 1/st), point_infinity]]:
points_im_dir_m_n1o :=
[[point_infinity, seq(composite_m1(-1 + I*v),      v = 0..+netsize_im, 1/st), point_infinity],
 [point_infinity, seq(composite_m1(-1 + I*v)+pmb1, v = 0..+netsize_im, 1/st), point_infinity],
 [point_infinity, seq(composite_m1(-1 + I*v)+pmb2, v = 0..+netsize_im, 1/st), point_infinity]]:
points_im_dir_p_n1u :=
[[point_infinity, seq(composite_p3(-1 + I*v),      v = -netsize_im..0, 1/st), point_infinity],
 [point_infinity, seq(composite_p3(-1 + I*v)+pmb1, v = -netsize_im..0, 1/st), point_infinity],
 [point_infinity, seq(composite_p3(-1 + I*v)+pmb2, v = -netsize_im..0, 1/st), point_infinity]]:
points_im_dir_m_n1u :=
[[point_infinity, seq(composite_m3(-1 + I*v),      v = -netsize_im..+netsize_im, 1/st), point_infinity],
 [point_infinity, seq(composite_m3(-1 + I*v)+pmb1, v = -netsize_im..+netsize_im, 1/st), point_infinity],
 [point_infinity, seq(composite_m3(-1 + I*v)+pmb2, v = -netsize_im..+netsize_im, 1/st), point_infinity]]:
points_im_dir_p_n3 :=
[[seq(composite_p3(-3 + I*v),           v = 0..+netsize_im, 1/st), point_infinity],
 [seq(composite_p3(-3 + I*v)+[0,0,0.02], v = 0..+netsize_im, 1/st), point_infinity],
 [seq(composite_p3(-3 + I*v)+[0,0.02,0], v = 0..+netsize_im, 1/st), point_infinity]]:
points_im_dir_m_n3 :=
[[seq(composite_m3(-3 + I*v),           v = 0..+netsize_im, 1/st), point_infinity],
 [seq(composite_m3(-3 + I*v)+[0,0,0.02], v = 0..+netsize_im, 1/st), point_infinity],
 [seq(composite_m3(-3 + I*v)+[0,0.02,0], v = 0..+netsize_im, 1/st), point_infinity]]:
```

```
labeltext :=
{[op(composite_p3(1)),"x"]
,[op(composite_p3(-1)),"xx"]
,[op(composite_p3(-3)),"xxx"]
,[op(point_infinity),"xxxx"]
}:
```

Darstellen lassen wir dies wie folgt.

```
display(
surfdata({
  points_p_i
 ,points_p_ii
 ,points_p_iii
 ,points_p_iv
 ,points_p_v
 ,points_p_vi
 ,points_p_vii
 ,points_p_viii
 ,points_m_i
 ,points_m_ii
 ,points_m_iii
 ,points_m_iv
 ,points_m_v
 ,points_m_vi
 ,points_m_vii
 ,points_m_viii
}, axes = boxed, tickmarks = [0,0,0], transparency = 0.8)
,surfdata({points_real_dir_p,points_real_dir_m},
color = "violet", axes = boxed, tickmarks = [0,0,0], transparency = 0)
,surfdata({points_im_dir_p_1,points_im_dir_m_1},
color = "yellow", axes = boxed, tickmarks = [0,0,0], transparency = 0)
,surfdata({points_im_dir_p_n1o,points_im_dir_m_n1o},
color = "ForestGreen", axes = boxed, tickmarks = [0,0,0], transparency = 0)
,surfdata({points_im_dir_p_n1u,points_im_dir_m_n1u},
color = "YellowGreen", axes = boxed, tickmarks = [0,0,0], transparency = 0)
,surfdata({points_im_dir_p_n3,points_im_dir_m_n3},
color = "DarkRed", axes = boxed, tickmarks = [0,0,0], transparency = 0)
,textplot3d(labeltext)
,surfdata(points_real_axis_arg,
color = "blue", axes = boxed, tickmarks = [0,0,0], transparency = 0)
,surfdata(points_real_axis_val,
color = "green", axes = boxed, tickmarks = [0,0,0], transparency = 0)
);
```

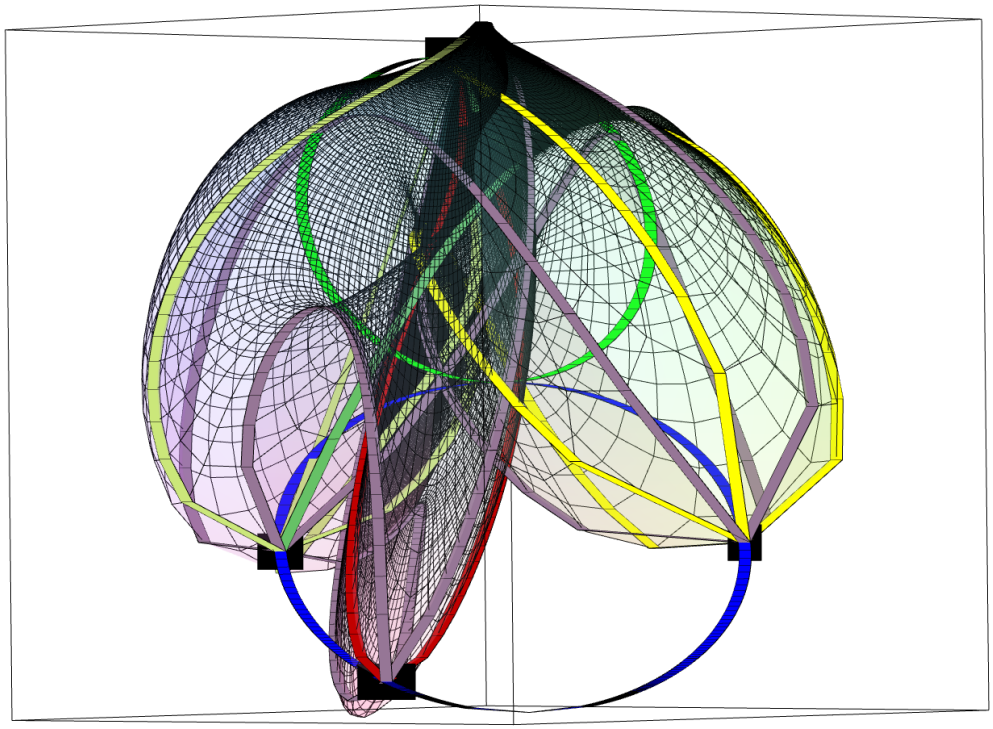
Dies liefert folgende Graphik; cf. Galerie 3, Figur 5.1.



Mit der alternativen Projektion

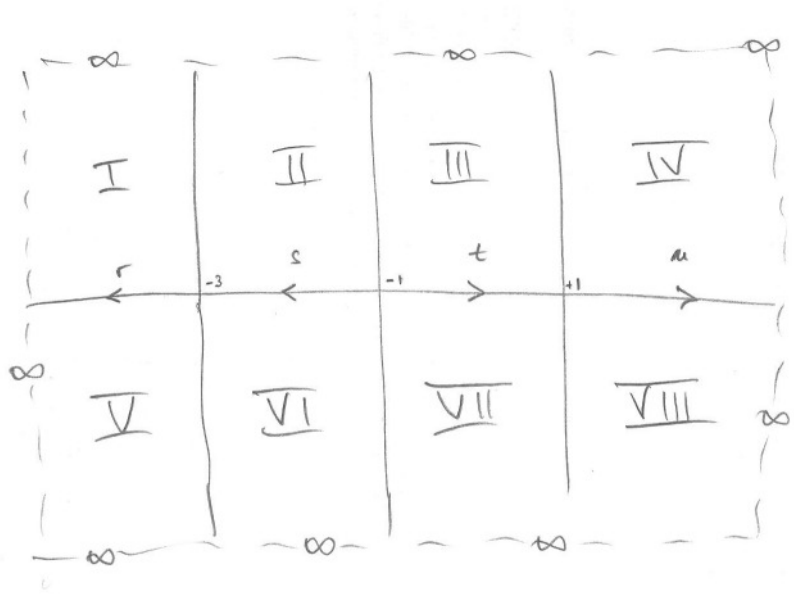
```
submat := # Spalten spannen Teilraum auf
Matrix([
[ 1, 0, 0],
[ 1, 0, 0],
[ 0, 1, 0],
[ 0, 1, 0],
[ 0, 0, 1],
[ 0, 0, 1],
[ 3, 3, 3],
[ 0, 0, 0],
[ 0, 0, 0]
]):
P := orth(submat);
```

liefert dies folgende Graphik; cf. Galerie 3, Figur 5.2.



9 Topologische Überlegungen

Zu jedem Bereich des unterteilten Definitionsbereichs

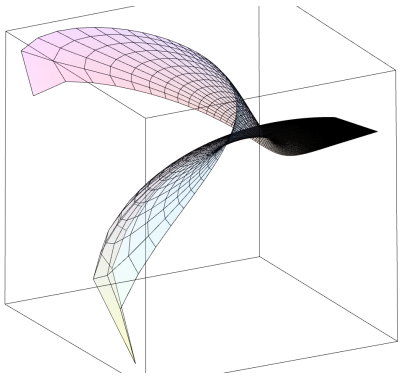


gehören zwei Bereiche auf der elliptischen Kurve. Wir stellen mittels graphischer Ausgabe fest, welche zusammengehören.

So etwa liefert unter Verwendung der alternativen Projektion aus §8 der Befehl

```
surfdata({
  points_p_iii
 ,points_m_iv
}, axes = boxed, tickmarks = [0,0,0], transparency = 0.8);
```

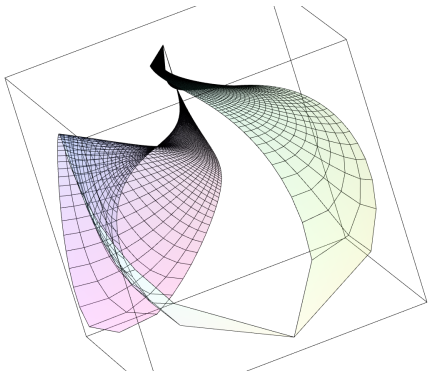
die folgende Graphik; cf. Galerie 3, Figur 5.3.



Dies zeigt, daß `points_p_iii` und `points_m_iv` einen gemeinsamen Rand haben.
Gegenprobe: Es liefert der Befehl

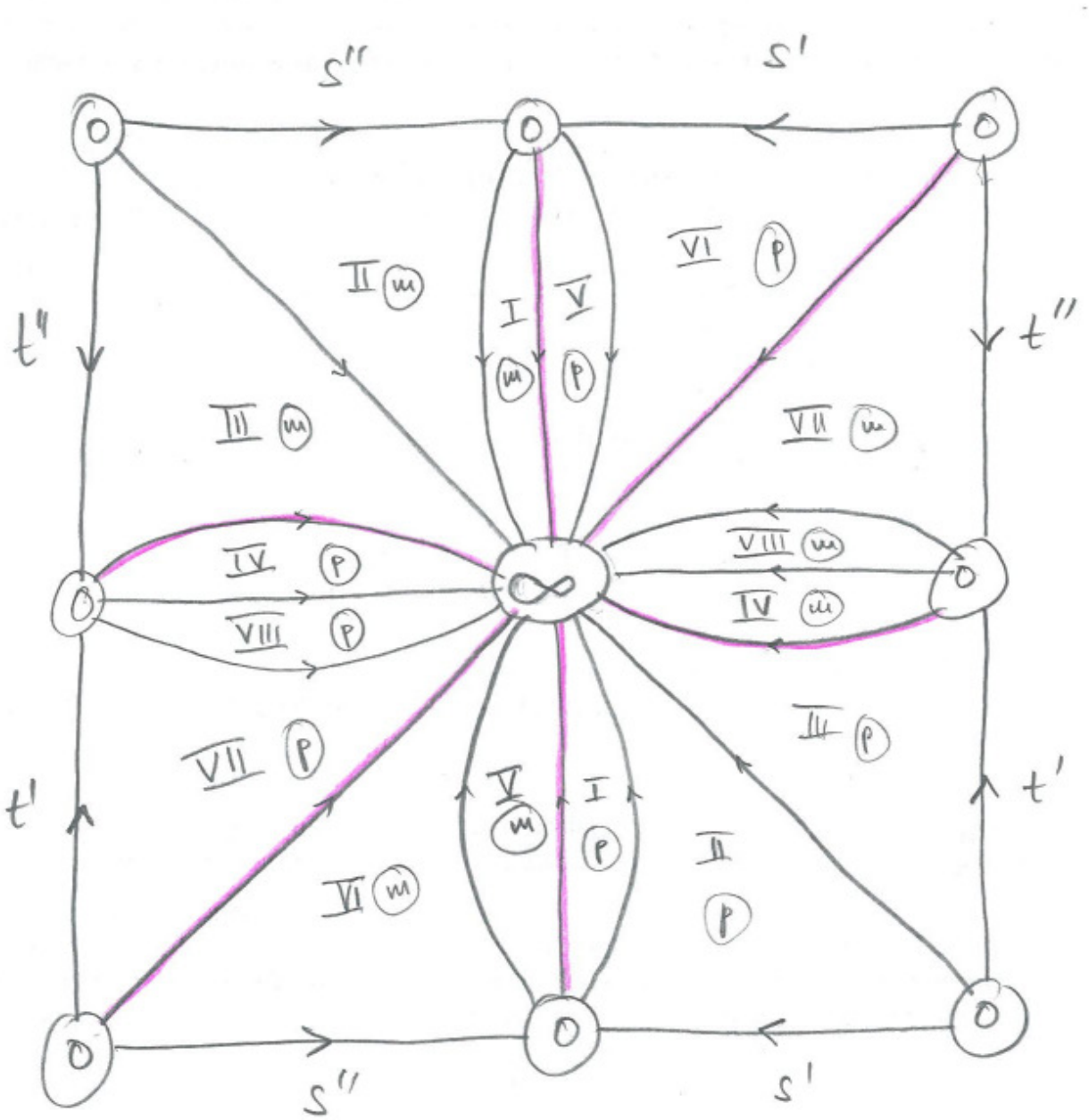
```
surfdata({
  points_p_iii
 ,points_p_iv
}, axes = boxed, tickmarks = [0,0,0], transparency = 0.8);
```

die folgende Graphik; cf. Galerie 3, Figur 5.4.



Dies zeigt, daß `points_p_iii` und `points_p_iv` keinen gemeinsamen Rand haben.

Dies können wir für alle Bereiche durchführen und erhalten folgendes Bild.



Hierzu wurde auch noch herangezogen, daß `points_p_iii` und `points_p_vii` einen gemeinsamen Rand haben, daß `points_m_iii` und `points_m_vii` einen gemeinsamen Rand haben, daß `points_p_ii` und `points_p_vi` einen gemeinsamen Rand haben und daß `points_m_ii` und `points_m_vi` einen gemeinsamen Rand haben, wie man mit derselben Methode feststellt.

Die Kanten, an denen `p` und `m` verbunden sind, wurden lila hervorgehoben.

Dieses Bild liefert einen Torus, wie man nach Verklebung der jeweils gegenüberliegenden Quadratskanten feststellt.

10 Aufgabenstellung

Erstellung eigener Beispiele, in Anlehnung an die oben angeführten.

Beschreibung der entstandenen Bilder: Erklärungen für auffällige Phänomene.