Mobile Gleichungsgeschichte(n): Digitaler Baukasten

Rolf-P. Holzapfel

```
> with(plots) : with(plottools) : with(plots, intersectplot) : with(LinearAlgebra) : with(linalg) :
  with(plots, implicitplot) : with(GaussInt) :
```

A10 Digital gedrehte Kegelschnitte

```
> Eb := proc(u, v, w, p, q, r)
   option NormalenEbeneDurchVektor :
   local V :local N :
   V := ⟨u, v, w⟩ : N := ⟨p, q, r⟩ :
   BilinearForm(N, (⟨x, y, z⟩ - V), conjugate = false) :
   end proc:
> FarbLichtDreh := proc(F, a, b, C, L)
   option RaumdrehungKurve :
   plot3d(F, alpha = 0 ..2·Pi, x = a ..b, grid = [50, 50], coords = cylindrical, style = patchnogrid,
        scaling = constrained, lightmodel = L, color = C) :
   end proc:
```

```
> Cut := animate(intersectplot, [x^2 + y^2 - z^2 = 0, cos(t) (x - 1/3) + sin(t) z = 0, x = -1.2 ..1.2, y
        = -1.2 ..1.2, z = -1.2 ..1.2], t = 0 ..Pi, axes = box, linestyle = solid, thickness = 4) :
> Ef := Eb(1/3, 1, 0, cos(t), 0, sin(t)) :
  An := animate(implicitplot3d, [Ef = 0, x = -1.2 ..1.2, y = -1.2 ..1.2, z = -1.2 ..1.2], t = 0 ..Pi,
        color = blue, axes = boxed, style = patchnogrid) :
> K := FarbLichtDreh(x, -1.2, 1.2, cyan, light4) :
> S := array(1 ..2) : S[1] := display(An, K) : S[2] := Cut :
> display(S) :
```

A5 Tanz der Stäbe/Geaden auf dem Hyperboloid

```
> Hc := implicitplot3d(x^2 + y^2 - z^2 = 1, x = -sqrt(2) ..sqrt(2), y = -sqrt(2) ..sqrt(2), z = -1 ..1, style = contour,
        color = green, axes = none) :
  display(Hc, orientation = [-60, 75, 0]) :
> ST := array(1 ..2) :
> Stab := animate(spacecurve, [[cos(t) + (1 + u) sin(t), sin(t) - (1 + u) cos(t), -1 - u], u
        = -2 ..0, thickness = 3, color = red, axes = box], t = Pi/8 ..2 * Pi + Pi/8) :
  ST[1] := display(scale(Stab, 1, 1, 1), Hc, orientation = [20, 60, 15], axes = none) :
> Stab_ := animate(spacecurve, [[cos(t) + (1 + u) sin(t), -sin(t) + (1 + u) cos(t), -1 - u],
```

$u = -2..0,\ thickness = 3,\ color = blue,\ axes = box\Big],\ t = \frac{Pi}{8}..2 * Pi + \frac{Pi}{8}\Big)$:

> $ST_2 := display(scale(Stab\_, 1, 1, 1), Hc, orientation = [20, 60, 15], axes = none)$:

> $display(ST)$:

A6 Entgegengesetzt verdrehte Zylinder

> **for** k **from** 1 **to** 32 **do** $G_+[k] := animate\Big(spacecurve, [[\cos(t) + z\cdot\sin(t), \sin(t) - z\cdot\cos(t), -z], z = -1..1, thickness = 3, color = red, axes = box], t = \frac{k\cdot Pi}{16}..\frac{k\cdot Pi}{16} + 2\cdot Pi\Big)$ **od**:

> **for** k **from** 1 **to** 32 **do** $G\_[k] := animate\Big(spacecurve, [[\cos(-t) + z\cdot\sin(-t), -\sin(-t) + z\cdot\cos(-t), -z], z = -1..1, thickness = 3, color = blue, axes = box], t = \frac{k\cdot Pi}{16}..2 * Pi + \frac{k\cdot Pi}{16}\Big)$ **od**:

> $A := array(1..2) : A_1 := display(seq(G_+[k], k = 1..32), orientation = [-60, 75, 0])$:
$A_2 := display(seq(G\_[k], k = 1..32), \text{orientation} = [-60, 75, 0])$: $display(A)$:

A7 Hyperbolisches Getriebe

> Kvorn $:= intersectplot((x + 1)^2 + z^2 = 2.7, y = -1.4, x = -2.7..3, y = -1.7..1.7, z = -1.5..1.5, axes = none, color = blue, thickness = 4)$:

> $Koben := intersectplot((x - 1)^2 + y^2 = 2.7, z = 1.4, x = -2.4..3, y = -1.7..1.7, z = -1.5..1.5, axes = none, color = red, thickness = 4)$:

> **for** k **from** 1 **to** 32 **do** $Gy1\_[k] := animate\Big(spacecurve, [[-1 + \cos(-t) + y\cdot\sin(-t), -y, \sin(-t) - y\cdot\cos(-t)], y = -\sqrt{2}..\sqrt{2}, \text{thickness} = 3, \text{color} = \text{blue}, axes = box], t = \frac{k\cdot Pi}{16}..\frac{k\cdot Pi}{16} + 2\cdot Pi\Big)$ **od**:

> **for** k **from** 1 **to** 32 **do** $g1[k] := animate\Big(spacecurve, [[1 + \cos(t) + z\cdot\sin(t), \sin(t) - z\cdot\cos(t), -z], z = -\sqrt{2}..\sqrt{2}, thickness = 3, color = red, axes = box], t = \frac{k\cdot Pi}{16}..2 * Pi + \frac{k\cdot Pi}{16}\Big)$ **od**:

> $display(\text{Kvorn}, \text{Koben}, seq(Gy1\_[k], k = 1..32), seq(g1[k], k = 1..32), orientation = [-60, 75, 0], scaling = constrained, axes = box)$:

A8 Newton's mobile Kubik-Kurven

> $c0_+ := animate\Big(plot, \Big[\sqrt{(x + 1)(x^2 + t)}, x = -1.2..1.2\Big], t = -\frac{1}{2}..0, thickness = 2\Big)$:

$c0\_ := animate\Big(plot, \Big[-\sqrt{(x + 1)(x^2 + t)}, x = -1.2..1.2\Big], t = -\frac{1}{2}..0, thickness = 2\Big)$:

```
> c1_+ := animate( plot, [sqrt((x+1)(x^2+t)), x=-1.2..1.2], t=0.01..1, thickness=2):
  c1_- := animate( plot, [-sqrt((x+1)(x^2+t)), x=-1.2..1.2], t=0.01..1, thickness=2):
  display(c1_+, c1_-):

> c_+ := animate( plot, [sqrt((x+1)(x^2+t)), x=-1.2..1.2], t=0..0, thickness=2):
  c_- := animate( plot, [-sqrt((x+1)(x^2+t)), x=-1.2..1.2], t=0..0, thickness=2):
  display(c_+, c_-):

> Uv := textplot([-0.2,-1.2, typeset("Newton-Kurve mit Oval")], font=[TIMES, BOLD, 18],
  color=black):
  U1 := textplot([0,-1.5, typeset("einastige Newton-Kurve")], font=[TIMES, BOLD, 18],
  color=black):
  U0 := textplot([0,-1.2, typeset("singuläre Grenzkurve mit Doppelpunkt")], font=[TIMES,
  BOLD, 18], color=black):

> Nc := array(1..3): Nc_1 := display(c0_+, c0_-, Uv): Nc_2 := display(c_+, c_-, U0): Nc_3
  := display(c1_+, c1_-, U1): display(Nc):

> e0_+ := animate( plot, [sqrt(x·(x^2+t)), x=-1.2..1.2, color=blue], t=-1/2..0, thickness=2):
  e0_- := animate( plot, [-sqrt(x·(x^2+t)), x=-1.2..1.2, color=blue], t=-1/2..0, thickness
  =2):
  display(e0_+, e0_-):

> e1_+ := animate( plot, [sqrt(x(x^2+t)), x=-1.2..1.2, color=blue], t=0.05..1, thickness=2):
  e1_- := animate( plot, [-sqrt(x(x^2+t)), x=-1.2..1.2, color=blue], t=0.05..1, thickness=2):
  display(e1_+, e1_-):

> E1_+ := animate( plot, [sqrt(x(x^2+t)), x=-1.2..1.2, color=blue], t=0.2..1, thickness=2):
  E1_- := animate( plot, [-sqrt(x(x^2+t)), x=-1.2..1.2, color=blue], t=0.2..1, thickness=2):

> e_+ := animate( plot, [sqrt(x(x^2+t)), x=-1.2..1.2, color=blue], t=0..0, thickness=2):
  e_- := animate( plot, [-sqrt(x(x^2+t)), x=-1.2..1.2, color=blue], t=0..0, thickness=2):
  display(e_+, e_-):

> Ov := textplot([-.2, 1.2, typeset("Newton-Kurve mit Oval")], font=[TIMES, BOLD, 18],
  color=black):
  O1 := textplot([0, 1.5, typeset("einastige Newton-Kurve")], font=[TIMES, BOLD, 18], color
  =black):
  O0 := textplot([-0.2, 1.2, typeset("singuläre Grenzkurve mit Spitze")], font=[TIMES,
  BOLD, 18], color=black):

> NE_3 := display(E1_+, E1_-, O1): NE_3:

> Ne := array(1..3): Ne_1 := display(e0_+, e0_-, Ov): Ne_2 := display(e_+, e_-, O0): Ne_3
  := NE_3:

> display(Nc): display(Ne):
```

B3

Fermat-Kurve 4. Grades:

x^4+y^4=1

Reelle Kurve ist ein Oval (rot)

Komplexe Kurve ist eine dreilöchrige Brezel

Existiert keine < -Lösung (außer x=0,y=±1; x=±1,y=0)
Fermat, 17. Jahrhundert

A3 Ei des Kolumbus

```
> o := x→sqrt(-x^2+sqrt(x)^3) : ei := evala( subs(x=-x, o(x))) :
> Tx := textplot( [0.3, 0, typeset("(x^2+y^2)^2 - x^3 = 0")], align=left, font=[TIMES, BOLD,
      20], color=blue) :
  tx := textplot( [0.1, 0.1, typeset("Eikurve:")], align=left, font=[TIMES, BOLD, 20], color
      =blue) :
> ei_+ := subs(x=x-0.5, ei) : Ei_+ := plot(ei_+, x=-1/2..1/2) : Ei_- := plot(-ei_+, x=-1/2
      ..1/2) :
  Ovoid := display(Ei_+, Ei_, Tx, tx, axes=none) : Ovoid :
> QuerEi := FarbLichtDreh(ei, -1, 0, gold, light4, orientation=[-115, 20, 130]) :
> HuhnEi := display(QuerEi, title="Hühnerei", titlefont=[TIMES, bold, 20], orientation=[
      -20, 40, -140]) : HuhnEi :
> ev := subs(x=x-1/40, ei) : SchnittEi := FarbLichtDreh(ev, -1+1/40, 0, gold, light4,
```

```
        orientation = [ -115, 20, 130 ] ) :
    SchnittEi :
> schnitt := changecoords( complexplot3d( 0, 0 .. 1/50 ·Pi + 2·Pi·I, view = -1 ..4, grid = [50, 50 ],
        style = surface, color = yellow, axes = none, lightmodel = light4 ), cylindrical ) :
> KolumbusEi := display( SchnittEi, schnitt, title = "Ei des Kolumbus", titlefont = [ TIMES, bold,
        20 ], orientation = [ -110, 45, 50 ] ) : KolumbusEi :
> Tr := array( 1 ..3 ) : Tr_1 := HuhnEi : Tr_2 := Ovoid : Tr_3 := KolumbusEi : display( Tr ) :
```

A4 Exp-Log - Spiralen, - Wendeltreppe

```
> opts := thickness = 3, numpoints = 100, color = red :
  opt2 := thickness = 2, numpoints = 100, color = red :
> Exp := animate( spacecurve, [ [ cos( x·2 π ), sin( x·2 π ), t·2·Pi·x ], x = 0 ..6, opts ], t = 1 ..0 ) :
  Log := animate( spacecurve, [ [ cos( x·2 π ), sin( x·2 π ), t·2·Pi·x ], x = 0 ..6, opts ], t = 0 ..1 ) :
> Über1 : P := seq( [ 1, 0, k·2·Pi ], k = 0 ..6 ) : seq( evalf[3]( P_k ), k = 1 ..7 ) : Über_-1 : G := seq( [
        -1, 0, ( 2 k - 1 )·Pi ], k = 1 ..6 ) :

> t_0 := textplot3d( [ 1.3, 0, 0, typeset( 1 ) ], align = right, font = [ TIMES, BOLD, 20 ], color
        = blue ) :
  for j from 1 to 4 do t[ j ] := textplot3d( [ 1.15, 0, 2·π·j, typeset( j·2 π i ) ], align = right, font
        = [ TIMES, BOLD, 20 ], color = blue ) od:
  for j from 5 to 6 do t[ j ] := textplot3d( [ 1.05, 0, 2·π·j, typeset( j·2 π i ) ], align = right, font
        = [ TIMES, BOLD, 20 ], color = blue ) od:

> GrüneTexte :for j from 1 to 6 do g[ j ] := textplot3d( [ -1.1, 0, 2·π·j - π, typeset( ( 2·j - 1 )
        ·π i ) ], align = left, font = [ TIMES, BOLD, 20 ], color = green ) od:

> Down := textplot3d( [ -1.5, 0, 22, typeset( "exp↓" ) ], align = left, font = [ TIMES, BOLD, 20 ],
        color = blue ) :

> optb := thickness = 4, numpoints = 100, color = blue : optr := thickness = 1, numpoints = 100,
        color = red :
  Kreis := spacecurve( [ cos( x·2 π ), sin( x·2 π ), 0 ], x = 0 ..1, optb ) :
> Punkte := { seq( P[ k ], k = 1 ..7 ) } : Pp := pointplot3d( Punkte, title
        = "Komplexe Exponentialfunktion, eingeschränkt auf imaginäre Achse (zur Spirale
        gedreht)", titlefont = [ TIMES, bold, 15 ], axes = normal, color = blue, symbol = solidcircle,
        symbolsize = 20 ) : Sp := display( Exp, Pp, seq( t_k, k = 1 ..6 ), Kreis, Down ) :

> Up := textplot3d( [ 2, 0, 20, typeset( "log ↑" ) ], align = left, font = [ TIMES, BOLD, 20 ], color
        = green ) :
> GrünPunkte := { seq( G[ k ], k = 1 ..6 ) } : Pg := pointplot3d( GrünPunkte, title
        = "Komplexer Logarithmus, eingeschränkt auf  (blauen) Kreis  |z| = 1", titlefont = [ TIMES,
        bold, 15 ], axes = normal, color = green, symbol = solidcircle, symbolsize = 20 ) : sp
```

```
:= display(Log, Pg, seq(g_k, k = 1 ..6), Kreis, Up) :
```

```
> Lg := z→ln(z) : K6 := spacecurve([6·cos(x·2 π), 6·sin(x·2 π), 0], x = 0 ..1, optb) :
> W := changecoords(complexplot3d(Lg, 0 ..2·Pi + 6·Pi·I, title
      = "Wendeltreppenfläche des komplexen Logarithmus, eingeschränkt auf |z| ≤ 6", titlefont
      = [TIMES, bold, 15], view = -1 ..4, caption = "Erste Riemannsche Fläche", captionfont
      = [TIMES, bold, 15], grid = [50, 50], style = surface, axes = normal, lightmodel = light4),
    cylindrical) : display(W, K6) :
```

```
> Disk := changecoords(complexplot3d(0, 0 ..2·Pi + 6·Pi·I, view = -1 ..4, grid = [50, 50], style
      = surface, color = blue, axes = none, lightmodel = light4), cylindrical) :
> Wendel := changecoords(complexplot3d(Lg, 0 ..2·Pi + 6·Pi·I, title
      = "Riemannsche Fläche des Logarithmus", titlefont = [TIMES, bold, 15], view = -1 ..4, grid
      = [50, 50], style = surface, axes = none, lightmodel = light4), cylindrical) :
> Pf := textplot3d([-4, 0, 1, typeset("↳")], align = left, font = [TIMES, BOLD, 30], color
      = red) :
> WK6 := display(Wendel, Disk, orientation = [-71, 78, -3]) :
> S := array(1 ..3) : S_1 := Sp : S_2 := WK6 : S_3 := sp : display(S) :
```

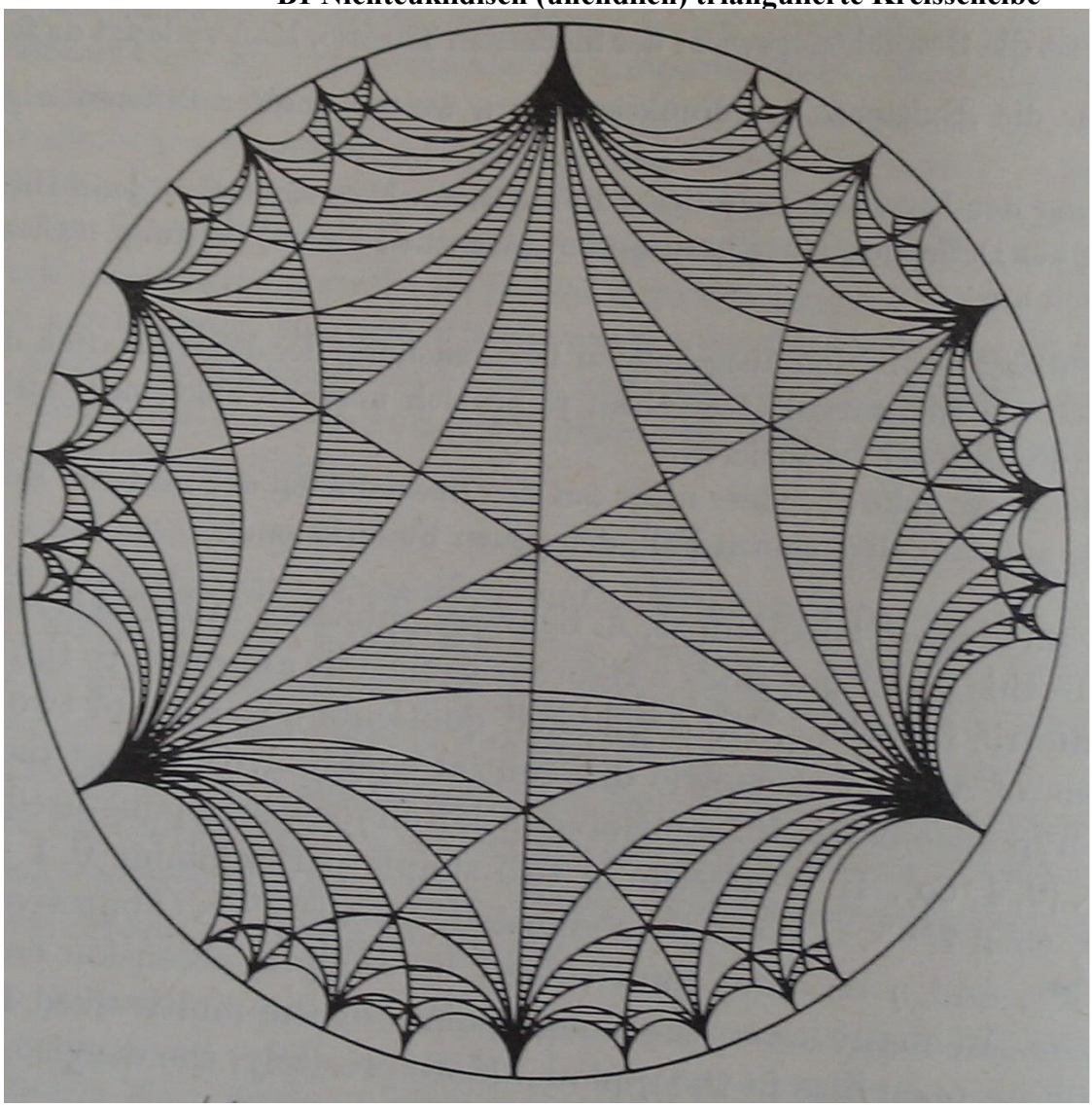
A4' Exp/Log - Fundamentalbogen

```
> opt2 := thickness = 2, numpoints = 100, color = red :
  ExpR := animate(spacecurve, [[cos(x·2 π), sin(x·2 π), t·2·Pi·x], x = 0 ..6, opt2], t = 1 ..0) :
> optb := thickness = 4, numpoints = 100, color = blue : optr := thickness = 1, numpoints = 100,
      color = red :
  Kreis := spacecurve([cos(x·2 π), sin(x·2 π), 0], x = 0 ..1, optb) :
> Windung := animate(spacecurve, [[cos(x·2 π), sin(x·2 π), t·2·Pi·x], x = 3 ..4, optb], t = 1
      ..0) :
> display(Windung, Kreis, ExpR, axes = normal, title
      = "Fundamental-Bogen der Exponentialabbildung, eingeschränkt auf die zur Spirale
    gedrehten imaginären Geraden", titlefont = [TIMES, bold, 15]) :
```

A2 Euklidischer Torus

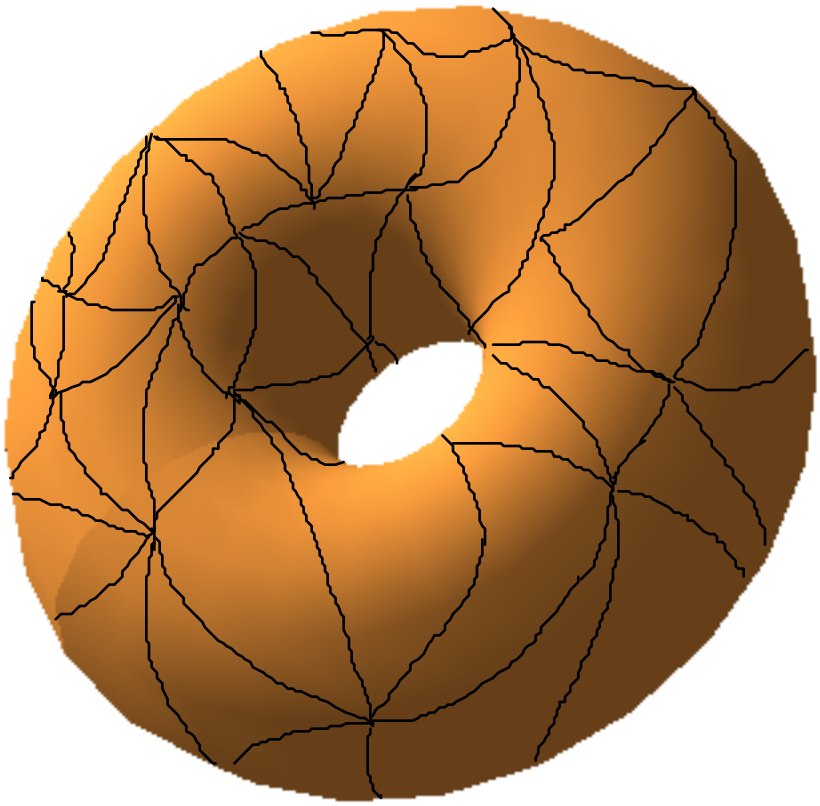
```
> C := array(1 ..3) :
> E := plot3d([x, y, 0], x = -1 ..1, y = -1 ..1) : C_1 := display(scale(E, 1, 1.5, 2), title
      = "  Fundamentalbereich einer doppelt-periodischen komplexen
    Funktion", titlefont = [TIMES, bold, 20], orientation = [180, 0, -180]) :
> cs := [cos(x), sin(x), 5·h] :
  C_2 := plot3d(cs, x = 0 ..2 * Pi, h = 0 ..1, grid = [24, 24], style = patch, lightmodel = light3, title
      = "wird doppelt zusammengerollt zum", titlefont = [TIMES, bold, 20], orientation = [
      -165, 40, 67]) :
> C_3 := plot3d([1, x, y], x = 0 ..2· Pi, y = 0 ..2 * Pi, coords = toroidal(10), style = patch, scaling
      = constrained, axes = none, title = "euklidisch gemaserten Torus  ", titlefont
      = [TIMES, bold, 20], orientation = [45, 20, -20]) :
> display(C) :
```

B1 Nichteuklidisch (unendlich) triangulierte Kreisscheibe



A1, B1 Nichteuklidisch triangulierter Torus

```
> NE := plot3d([1, x, y], x = 0..2·Pi, y = 0..2*Pi, coords = toroidal(10), style = patchnogrid,
      scaling = constrained, lightmodel = light3, color = gold, axes = none, orientation = [-75,
      -20, 150]) : NE :
```



(Torus mit Up-Pfeil vergrößern)

A11 Lösungszapfen der Weber-Gleichung

```
> P := Z→Z^5 + 5·Z + 5 :
> Eingabe : Polynom Q(Z) :
ReIm := proc(Q)
 option ZerlegungRealImaginärteil :
local s : s := expand(subs(Z = r·cos(φ) + r·I·sin(φ), Q)) :
[subs(I = 0, s), coeff(s, I)] :
end proc:
Ausgabe : Realteil, ImaginärteilPolynom :

> Eingabe : Polarkoordinaten :
xy := proc(r, φ)
 option PolarZuKartesischeKoordinaten :
r·cos(φ) + r·sin(φ)·I :
end proc:
Ausgabe : Gaußsche Zahl :
> t1 := textplot([[-0.9, -5, typeset(ζ1)]], font = [HELVETICA, bold, 20]) :
```

```
t1 := textplot3d([[-0.9, 0, -0.1, ς1]], axes = normal, font = [HELVETICA, bold, 20]) :
t2 := textplot3d([[-0.8, -1.2, 0, ς2]], axes = normal, font = [HELVETICA, bold, 20]) :
t3 := textplot3d([[-0.8, 1.2, 0.1, ς3]], axes = normal, font = [HELVETICA, bold, 20]) :
t4 := textplot3d([[1.25, -1.1, 0, ς4]], axes = normal, font = [HELVETICA, bold, 20]) :
t5 := textplot3d([[1.25, 1.1, 0.2, ς5]], axes = normal, font = [HELVETICA, bold, 20]) :
ti := textplot({[-0.05, 42, typeset(Kurven – Graphik)], [1.1, -20, typeset(y = x^5 + 5 x + 5)]}, font = [TIMES, BOLD, 23]) :
tit := textplot3d([[0, 0, 0, "."]], title = "Betrags-Fläche über der Gaußschen Zahlenebene", titlefont = [TIMES, BOLD, 20], axes = normal, font = [HELVETICA, bold, 20]) :
> BF := complexplot3d(P, -1.5 - 1.5 I..1.5 + 1.5 I, view = -1..4, grid = [50, 50], style = surface, axes = normal, lightmodel = light4) :
> BetragsFläche := display(BF, t1, t2, t3, t4, t5, tit, orientation = [-71, 78, -3]) :
> KG := plot(P(x), x = -2..2, thickness = 3, color = blue) : Graph := display(KG, ti, t1) :
> C := array(1..2) : C1 := Graph : C2 := BetragsFläche : GraphischeDarstellungen := display(C) : GraphischeDarstellungen :
```

A9 Lösungs-Drohnen einer algebraischen Gleichung

```
> Eingabe : Polynom Q(Z) :
ReIm := proc(Q)
 option ZerlegungRealImaginärteil :
local s : s := expand(subs(Z = r·cos(φ) + r·I·sin(φ), Q)) :
[subs(I = 0, s), coeff(s, I)] :
end proc:
Ausgabe : Realteil, ImaginärteilPolynom :
```

Proceduren:

```
> Eingabe : Polarkoordinaten :
xy := proc(r, φ)
 option PolarZuKartesischeKoordinaten :
r·cos(φ) + r·sin(φ)·I :
end proc:
Ausgabe : Gaußsche Zahl :
> Eingabe : PolynomQ(Z), Stelleς :
Newt := proc(Q, ς)
option NäherungsschrittNewtonVerfahren :
 local g : g := diff(Q, Z) :
 subs(Z = ς, Z - Q/g) :
 end proc:
Ausgabe : Näherungswert nach einem Näherungsschritt :
> Eingabe : PolynomQ(Z), Stelleς :
Newton := proc(Q, ς) option ApproximationkStellenNachKomma :
local S, i :
S := [0, ς] :
for i from 1 to 5 do
S := [S2, evalf(Newt(Q, S2))]
```

```
    end do: S2
end proc:
    Ausgabe : Näherungswert nach einem Näherungsschritt :
> Eingabe : StartradiusA, EndradiusB, Startwinkelα, Endwinkelβ :
Welle := proc(A, B, α, β) option EingeschränkteAnimationsWelle :
animate( plot, [[ReP, ImP, φ = α..β]], r = A..B, scaling = constrained, frames = 50) :
end proc:
    Ausgabe : Animationswelle der BetragsFunktion des PolynomsP
     in den vier Eingabegrenzen :
> Eingabe : GrenzradiusR :
RadiusWelle := proc(R) option VolleAnimationsWelle :
 Welle(0, R, 0, 2·π) :
end proc:
    Ausgabe : Animationswelle der BetragsFunktion des PolynomsP von0 bis R :
> Eingabe : Grenzradien r und R :
 TeilWelle := proc(r, R) option AnimationsTeilWelle :
Welle(r, R, 0, 2·π) :
end proc:
    Ausgabe : Animationswelle der BetragsFunktion des PolynomsP
     in den zwei Eingabegrenzen :
> Eingabe : RoutenRadiusR :
Route := proc(R) option WellenEinfrierungBeiRadiusR :
plot([subs(r = R, ReP), subs(r = R, ImP), φ = 0..2·Pi]) :
end proc:
    Ausgabe : Route der BetragsWerte entlang des Kreises mit RadiusR :
> Eingabe : Kartesische Koordinaten eines Punktes der Ebene :
Pt := proc(x, y) option PunktMarkierung :
  plots[pointplot]([[x, y]], symbol = solidcircle, symbolsize = 30, color = blue) :
end proc:
    Ausgabe : Markierung als fetten Punkt (Drohne) :
> Eingabe : RoutenRadiusR :
Drohne := proc(R) option DrohnenFlugAnimation :
plots[animate](Pt, [subs(r = R, ReP), subs(r = R, ImP)], φ = 0..2·π, frames = 50, color
     = blue, axes = normal) :
 end proc:
    Ausgabe : Animation des Drohnenflugs entlang der Route mit RoutenRadiusR :
> Eingabe : Routenradiusρ, AnfangsWinkelα, EndWinkelβ :
route := proc(ρ, α, β) option TeilrouteMitWinkelGrenzen :
plot([subs(r = ρ, ReP), subs(r = ρ, ImP), φ = α..β]) :
end proc:
    Ausgabe : Teilroute in den genannten Grenzen :
> Eingabe : Routenradiusρ, AnfangsWinkelα, EndWinkelβ :
drohne := proc(ρ, α, β) option DrohnenFlugAnimation :
plots[animate](Pt, [subs(r = ρ, ReP), subs(r = ρ, ImP)], φ = α..β, frames = 50, color = blue,
     axes = normal) :
end proc:
    Ausgabe : Animation der Drohne entlang der angegebenen Grenzen :
```

Kreis-Markierungen:

```
> Eingabe : Kartesisches Koordinaten – Paar eines Punktes in der Ebene :
circ := proc(x, y) option KleinerKreisUmPunkt :
plots[pointplot]([[x, y]], symbol = circle, symbolsize = 30, color = blue) :
end proc:
    Ausgabe : Offener Mini-Kreis um den Punkt :
> Eingabe : Kartesisches Koordinaten – Paar eines Punktes in der Ebene :
Circ := proc(x, y) option FüllPunkt :
textplot([[x, y, typeset(o)]], font = [HELVETICA, bold, 25], color = blue) :
end proc:
    Ausgabe : Gefüllter Kreis um den Punkt :
```

Pfeil-Markierungen

```
> Eingabe : Kartesische Koordinaten der Mitte eines ebenen Pfeiles :
no := proc(x, y) option NordOstPfeil :
textplot([[x, y, typeset("↗")]], font = [HELVETICA, bold, 50], color = red) :
end proc:
    Ausgabe : Pfeil in Nord-Ost – Richtung :
> Eingabe : Kartesische Koordinaten der Mitte eines ebenen Pfeiles :
sw := proc(x, y) option SüdWestPfeil :
textplot([[x, y, typeset("↙")]], font = [HELVETICA, bold, 50], color = red) :
end proc:
    Ausgabe : Pfeil in Süd-West – Richtung :
> Eingabe : Kartesische Koordinaten der Mitte eines runden Richtungs – Pfeiles :
BowL := proc(x, y) option NachLinksBogenPfeil :
textplot([[x, y, typeset("↶")]], font = [HELVETICA, bold, 25], color = blue) :
end proc:
    Ausgabe : Richtungs – Pfeil entgegen dem Uhrzeiger – Sinn an der vorgesehenen Stelle :
> Eingabe : Kartesische Koordinaten der Mitte eines runden Richtungs – Pfeiles :
BowU := proc(x, y) option NachUntenBogenPfeil :
textplot([[x, y, typeset("↳")]], font = [HELVETICA, bold, 25], color = blue) :
end proc:
    Ausgabe : Richtungs – Pfeil im Uhrzeiger – Sinn an der vorgesehenen Stelle :

> P := Z→Z^5 + 5·Z + 5 : g := x→P(x) : ReP := ReIm(P(Z))_1 : ImP := ReIm(P(Z))_2 :

> w1 := textplot({[-30, 30, typeset(Radius-Welle)], [25, 30, typeset(bis r = 1.8)]}, font
    = [TIMES, BOLD, 20]) :
  w2 := textplot({[-10, 13, typeset(Teil – Welle)], [17, 13, typeset(r ∈ [1.3, 1.46])]}, align
    = above, font = [TIMES, BOLD, 20]) :
  Pfe_1 := no(20, 13), sw(-11, -11) : Pfe_2 := no(8, 5), sw(-4, -4) :
> W := array(1..2) : W_1 := display(RadiusWelle(1.8), w1, Pfe_1) : W_2
    := display(TeilWelle(1.3 , 1.46), w2, Pfe_2) : Radiuswellen := display(W) :
    Radiuswellen :

> Dr := textplot([-5, 12, typeset(Drohne)], font = [TIMES, BOLD, 20]) :
> R := array(1..3) : R_1 := display(Route(0.89), Drohne(0.89), Circ(0, 0.1), BowL(5, 4.5)) :
  R_2 := display(Route(1.43), Drohne(1.43), Circ(0, 0.3), BowL(5, 12), BowU(-5.5, -7),
    Dr) :
```

```
R3 := display(Route(1.655), Drohne(1.655), Circ(0, 0.5), BowL(5, 19)) : ArcusDrohnen
    := display(R) :

> RF := textplot([1, -25, typeset(Routen – Finder)], font = [TIMES, BOLD, 15]) :
> display(Welle(0.6, 1.8, 0, 2·π), RF) : ArcusDrohnen :
```

Polarkoordinaten (r,φ) ablesen:

```
> pk1 := 0.89, -π : pk2 := 1.43, 2.18 : pk4 :=  1.655, 0.72 : pk3 := 1.43, -2.18 : pk5 := 1.655,
      -0.72 :
> c := seq(evalf[3]( xy(pk_i) ), i = 1..5) : ς := seq(Newton(P(Z), c_i, 9), i = 1..5) :

>
```

Animations-Clips auf Facebook-Seite,
Bilder-Galerie und Vortragsartikel auf Homepage.