Blatt 13

Aufgabe 50

```
> restart:
> dgl := (x^2 * y(x) + y(x)^3) * diff(y(x), x) + x^3 + x * y(x)^2
  = 0;
> aw := y(1) = y1;
```

Lösungen für y0 = 1.

```
> y1 := 1;
> lsg := dsolve({ dgl, aw }, y(x));
> # Die beiden komplexen Lösungen erfüllen die Anfangsbedingung
  nicht!
> is(subs(x = 1, rhs(lsg[1])) - y1 = 0);
> is(subs(x = 1, rhs(lsg[2])) - y1 = 0);
> is(subs(x = 1, rhs(lsg[3])) - y1 = 0);
> u1 := unapply(rhs(lsg[3]), x);
> # Definitionsbereich -sqrt(2)..sqrt(2).
> plot(u1(x), x = -sqrt(2)..sqrt(2), color = red, legend = u1,
  scaling = constrained);
```

Lösungen für y0 = 0.

```
> y1 := 0;
> lsg := dsolve({ dgl, aw }, y(x));
> # Die beiden komplexen Lösungen erfüllen die Anfangsbedingung
  nicht!
> seq(print(rhs(l), subs(x = 1, rhs(l)), is(subs(x = 1, rhs(l) -
  y1 = 0))), l in lsg);
> u2 := unapply(rhs(lsg[3]), x);
> u3 := unapply(rhs(lsg[4]), x);
> # Definitionsbereich -1..1 für beide reellen Lösungen.
> plot([ u2(x), u3(x) ], x = -1..1, color = [ red, blue ], legend
  = [ u2, u3 ], scaling = constrained);
> # Lösungen in x = 1 diff'bar?
> subs(x = 1, diff(u2(x), x));
> subs(x = 1, diff(u3(x), x));
> # Nein!
```

Aufgabe 51

```
> restart:
> dgl := diff(y(x), x) - y(x)/x - x^2 - x^3 + x^4 = 0;
> # Alle Lösungen der DGL
> lsg_allg := dsolve(dgl, y(x));
> lsg_allg := rhs(dsolve(dgl, y(x))):
> aw := y(1) = y1;
```

```
> y1s := 1, 0, -1;
> for y1 in y1s do
    'y1' = y1;
    lsg[y1] := rhs(dsolve({ dgl, aw }, y(x)));
    # Wahlweise:
    C := solve({ subs(x = 1, lsg_allg) = y1 }, { _C1 });
    lsg2[y1] := subs(C[1], lsg_allg);
    lsg[y1] - lsg2[y1];
  end do;
> plot([ seq(lsg[y1], y1 in y1s) ], x = -1.5..2.5);
> # Die Graphen schneiden sich in der Null. Der Satz von
  # Picard-Lindelöff setzt aber Lipschitz-Stetigkeit im 2.
  # Argument voraus, um (lokale) Eindeutigkeit zu
  # garantieren. Dies ist in der Null nicht erfüllt.
```

Aufgabe 52

```
> restart:
> dgl := diff(y(x), x) = sqrt(x * y(x));
> aw := y(1) = 4;
> lsg := dsolve({ dgl, aw }, y(x));
> u1 := x -> rhs(lsg);
> test := simplify(subs(y(x) = u1(x), dgl));
> simplify(rhs(test) - lhs(test)) assuming x > 0;
> # Aha!
> a := x -> 0;
> b := x -> -sqrt(x);
> s := 1/2;
> dgl_neu := diff(y(x), x) + a(x) * y(x) + b(x) * y(x)^s = 0;
> # Test
> simplify(lhs(dgl_neu) - rhs(dgl_neu) - (lhs(dgl) - rhs(dgl)))
  assuming x > 0;
> # Korrespondierende DGL
> korr_dgl := diff(w(x), x) + (1 - s)*a(s)*w(x) + (1 - s)*b(x) =
  0;
> # Brauchen f^2(1) = y(1) = 4, also
> korr_aw := w(1) = sqrt(rhs(aw));
> f := rhs(dsolve({ korr_dgl, korr_aw }, w(x)));
> 'f^2' = expand(f^2);
> # Teste wieder die DGL:
> test2 := simplify(subs(y(x) = f^2, dgl));
> simplify(rhs(test2) - lhs(test2)) assuming x > 0;
```

Aufgabe 53

```
> restart:
> with(plots):
```

```
> dgl := diff(y(x), x) + y(x) * sin(x) + sin(2*x) = 0;
> aw := y(0) = y0;
> y0s := [ 2, 1, 0 ];
> lsg := dsolve({ dgl, aw }, y(x));
> lsg2 := unapply(rhs(lsg), (x, y0));
> us := map(Y0 -> (x -> lsg2(x, Y0)), y0s);
> # Richtungsfeld der DGL
> v := <1, rhs(isolate(dgl, diff(y(x), x)))>;
> # Vektorfeld normieren
> N := sqrt(v[1]^2 + v[2]^2);
> w := v/N;
> p_field := fieldplot(w, x = -Pi .. 2*Pi, y = -1 .. 3, color =
  blue):
> p_sol := plot([ seq(u(x), u in us) ],
    x = -Pi .. 2*Pi, color = [ red, green, yellow ]):
> display({ p_field, p_sol });
```

Aufgabe 54

```
> restart:
> with(plots):
> with(VectorCalculus):
> with(LinearAlgebra):
> BasisFormat(false):
> f := (x, y) -> x^2 * (y + 1) + y/2;
> g := (x, y) -> x^2 + y^2 - 1;
> # Parametrisierung
> param := t -> ( cos(t), sin(t) );
> (f@param)(t);
> # Ansehen
> pf := plot3d(f(x, y), x = -1.1..1.1, y = -1.1..1.1):
> pf_constr := spacecurve([ param(t)[1], param(t)[2], (f@param)
  (t), t = -Pi..Pi ], x = -1..1, y = -1..1, thickness = 3, color
  = black ):
> display([ pf, pf_constr ]);
> L := (x, y, lambda) -> f(x, y) + lambda * g(x, y);
> dL := (x, y, lambda) -> < seq(D[ll](L)(x, y, lambda), ll = 1.
  .3) >;
> d2L := (x, y, lambda) -> <seq(
     Transpose(< seq(D[ll,kk](L)(x, y, lambda), ll = 1..3) >),
    kk = 1..3) >;
> #L(1,1,1), dL(1,1,1), d2L(1,1,1);
> kritische_punkte := seq(allvalues(s),
    s in solve({ seq(dL(x, y, lambda)[ll] = 0, ll = 1..3) },
      { x, y, lambda }));
> for kr in kritische_punkte do
```

```
    #kr;
    print('<x, y>' = subs(kr, [x, y]));
    if (not is(subs(kr, x), real)) or (not is(subs(kr, y), real))
  then
     # Komplexe Werte überspringen
     print("Komplexe kritische Stelle ignoriert");
      next;
    end if;
    #'<x, y>' = evalf(subs(kr, [x, y]));
    print('f(x, y)' = subs(kr, f(x, y)));
   # Hessematrix
   d2L_val := subs(kr, d2L(x, y, lambda)):
   print(Re(evalf(simplify(Eigenvalues(d2L_val)))));
  end do:
> # Hmm, alle indefinit ... dann optisch ablesen
> # Filtere komplexe Lösungen
> kritische_punkte := seq(kritische_punkte[ll], ll = 1..4):
> kp3d := [ seq(subs(kr, [ x, y, f(x, y) ]), kr in
  kritische_punkte) ];
> pp := pointplot3d(kp3d, symbol = circle,  symbolsize = 50,
  color = yellow);
> display([ pf, pf_constr, pp ]);
```

Alternative: 1D Funktion

```
> h := t -> (f@param)(t);
> dh := D(h);
> d2h := D(dh);
> # h(0), dh(0), d2h(0);
> kritische_punkte := solve({ dh(t) = 0 }, { t });
> # Maple findet die kritische Stelle bei t = -Pi/2 nicht, fügen
  wir diese hinzu ...
> dh(-Pi/2);
> kritische_punkte := kritische_punkte, { t = -Pi/2 };
> # Oder vielleicht doch dabei?!?
> solve({ dh(t) = 0 }, { t }, AllSolutions=true) ;
> about(_Z1); about(_Z2);
> # Aha, die letzte Lösung ist Pi-periodisch, die anderen alle 2*
  Pi-periodisch.
> # Da unsere Parametrisierung 2*Pi-periodisch ist, sind es
  tatsächlich zwei verschiedene Lösungen!
> # (d.h. man muss die zweite Lösung von Hand zu obiger Liste
  hinzufügen)
> minMax[-1] := "Maximum":
> minMax[1] := "Minimum":
> minMax[0] := "???": # Bei 2. Ableitung = 0 müssen weitere
  Kriterien hinzugenommen werden
> for kr in kritische_punkte do
    #kr;
```

```
    xy := simplify(subs(kr, [param(t)])):
    print('t' = simplify(subs(kr, t)));
    print('<x, y>' = xy);
    if (not is(xy[1], real)) or
       (not is(xy[2], real)) then
     # Komplexe Werte überspringen
      print("Komplexe kritische Stelle ignoriert");
       next;
    end if;
    print('f(x, y)' = f(xy[1], xy[2]));
    # Kriterium 2. Ordnung
    d2h_val := simplify(subs(kr, d2h(t)));
    print('diff(f@param, t)'('t') = d2h_val);
    #typ := minMax[sign(d2h_val)];
    #print(typ);
  end do:
> # Und hier auch noch einmal zeichnen:
> kritische_punkte := seq(kritische_punkte[ll], ll in [ 1, 2, 5,
  6 ]);
> kp3d := [ seq(subs(kr, [ param(t), (f@param)(t) ]), kr in
  kritische_punkte) ]:
> pp := pointplot3d(kp3d, symbol = circle,  symbolsize = 50,
  color = yellow);
> display([ pf, pf_constr, pp ]);
```

Weitere Alternative: Lass Maple eine (stückweise) Parametrisierung finden

```
> solve(g(x, y) = 0, x);
> # Dann die beiden Lösungen als Parametrisierungen benutzen.
```