Aufgabe 55

```
> restart:
> with(LinearAlgebra):
> with(VectorCalculus):
> BasisFormat(false):
> A := < < 1, 0, 0, 0 > | < 1, 1, 0, 0 > | < 0, 1, 1, 0 > | < 0, 0, 0, 2 > >;
> y0 := < 1, 1, 1, 1 >;
> # Löse u1'(x) = A u(x), u(0) = y0.
> u1 := x -> MatrixExponential(A, x) . y0:
> u1(x);
> # Probe:
> diff(u1(x), x) - A . u1(x);
> # Inhomogene Gleichung
> g := x -> < sin(x), 0, x, 0 >;
> # Variation-der-Konstanten-Formel:
> u2 := x -> u1(x) + int(MatrixExponential(A, x - s) . g(s), s = 0 .. x);
> # Probe
> diff(u2(x), x) - (A . u2(x) + g(x));
> # Variante mit dsolve. Benötigt Maple-Version >= 18
> y := x -> < y1(x), y2(x), y3(x), y4(x) >;
> dgl := { diff(y(x), x) - A . y(x) };
> aw := { y(0) - y0 };
> v1 := x -> rhs(dsolve(dgl union aw, y(x)));
> v1(x);
> # Inhomogene Gleichung
> dgl2 := { diff(y(x), x) - A . y(x) - g(x) };
> v2 := x -> rhs(dsolve(dgl2 union aw, y(x)));
> v2(x);
> # Und natürlich wieder prüfen (s.o.) ...
```

Aufgabe 56

```
> restart:
> # Nun etwas allgemeiner, als in A 55.
> dgl := { diff(y(x), x) = A * y(x) + f(x, y(x)) };
> aw := { y(0) = y0 };
> Phi := exp(x * A) * y0 + int(exp((x - s) * A) * f(s, y(s)), s = 0 .. x);
> # Probe
> simplify(diff(Phi, x) - (A * Phi + f(x, y(x))));
```

Aufgabe 57

```
> restart:
> with(plots):
> dgl := {
    diff(y(x), x) = a * y(x)  - b * y(x) * z(x),
    diff(z(x), x) = -c * z(x) + d * y(x) * z(x)
  };
> aw := { y(0) = y0, z(0) = z0 };
> params := { y0 = 6000, z0 = 30, a = 1/5, b = 1/500, c = 1/10, d
  = 1/100000 };
> dgl2 := { op(subs(params, dgl)), op(subs(params, aw)) };
> loesung := dsolve(dgl2, { y(x), z(x) }, numeric, output =
  listprocedure);
> loesung(1);
> ly := x -> rhs(loesung[2](x));
> lz := x -> rhs(loesung[3](x));
> # Suche Plotbereich
> plot([ ly(x), 100 * lz(x), x = 0 .. 10]);
> plot([ ly(x), 100 * lz(x), x = 0 .. 20]);
> animate(plot, [ [ ly(x), 100 * lz(x), x = T - 10 .. T ], color
  = blue, legend = "y(x) vs. z(x)" ], T = 10 .. 60);
```

(b)

```
> eqs := seq(subs({ y(x) = y, z(x) = z }, rhs(dgl[kk]) = 0), kk =
  1..nops(dgl));
> gleichGewPunkte := solve({ eqs }, { y, z });
> # Probe:
> for ggp in gleichGewPunkte do
    awGG := subs(subs({ y = y0, z = z0 }, ggp), aw);
    dsolve({ op(dgl), op(awGG) }, { y(x), z(x) });
  end do;
> # Also Lösungen konstant!
```

(c)

```
> # Werte aus (a)
> # Richtungsfeld der DGL
> dgl2;
> v := <seq(subs({ y(x) = y, z(x) = z }, rhs(dgl2[kk])), kk = 1 .
  . nops(dgl)) >;
> # Vektorfeld normieren
> w := v / norm(v, 2);
> pf := fieldplot(w, y = 0..33000, z = 0 .. 250);
> psol := plot([ ly(x), lz(x), x = 0 .. 60 ], color = blue,
  thickness = 2);
> display(pf, psol);
```

Aufgabe 58

```
> restart:
```

```
> with(plots):
> dgl := {
    diff(y(x), x) = a * y(x) - k*y(x)^2 - b * y(x) * z(x),
    diff(z(x), x) = -c * z(x) + d * y(x) * z(x)
  };
> aw := { y(0) = y0, z(0) = 0 };
> loes := dsolve({ op(dgl), op(aw) }, { y(x), z(x) });
> limit(rhs(loes[1]), x = infinity) assuming a::positive;
```

(c)

```
> params := { y0 = 6000, z0 = 30, a = 1/5, b = 1/500, c = 1/10, d
  = 1/100000 };
> ks := { k = 1/10^5, k = 1/10^6 };
> aw2 := { y(0) = y0, z(0) = z0 };
> for kk in 1 .. nops(ks) do
    dgl2 := subs(params union {ks[kk]}, dgl) union subs(params,
  aw2):
    loesung := dsolve(dgl2, { y(x), z(x) }, numeric, output =
  listprocedure):
    ly := x -> rhs(loesung[2](x));
    lz := x -> rhs(loesung[3](x));
    print(plot([ ly(x), 100 * lz(x), x = 0 .. 300]));
  end do:
```

Aufgabe 59

```
> 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;
> param := t -> ( cos(t), sin(t) );
> h := t -> (f@param)(t);
> dh := D(h);
> d2h := D(dh);
> # h(0), dh(0), d2h(0);
> kritische_punkte := solve({ dh(t) = 0 }, { t }):
> # Und wie beim letzten Mal ist die Periodizität bei t = Pi/2
  anders, daher t = -Pi/2 hinzufügen
> kritische_punkte := kritische_punkte, { t = -Pi/2 };
> for kr in kritische_punkte do
    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:
> kritische_punkte := seq(kritische_punkte[ll], ll in [ 1, 2, 5,
  6 ]);
> # Jetzt das Innere betrachten.
> gradF := Gradient(f(x, y), [ x, y]);
> kritische_punkte_innen := solve({ gradF[1] = 0, gradF[2] = 0 },
  { x, y });
> for kr in allvalues(kritische_punkte_innen) do
    #xy := simplify(subs(kr, [param(t)])):
    xy := subs(kr, < x, y >);
    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;
  end do:
> # Also keine kritischen Punkte gefunden, also keine Extrema im
  Inneren.
> kp3d := [ seq(subs(kr, [ param(t), (f@param)(t) ]), kr in
  kritische_punkte) ]:
> 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 ):
> pp := pointplot3d(kp3d, symbol = circle,  symbolsize = 50,
  color = yellow);
> display([ pf, pf_constr, pp ]);
```