3.8 Verfolgungskurven

In diesem Kapitel werden wir einige klassische Probleme von „Verfolgungen“ betrachten, die bereits zu Zeiten Leonardo da Vincis untersucht wurden.
Es handelt sich darum, die Bahn eines Verfolgers zu berechnen, der sein Ziel so verfolgt, dass sein Geschwindigkeitsvektor in jedem Augenblick auf den momentanen Ort des verfolgten Objekts weist. Ein Beispiel ist ein Hund, der hinter seinem Herrn herläuft (oder einen Jogger angreifen will ...). Wir haben vor, die Wege von Hund und Herr in einer Animation darzustellen.

Das Wort *angreifen*, das wir eben benutzt haben, erinnert an ein anderes modernes Beispiel, nämlich an eine Verfolgungsrakete, die mit einem elektronischen Suchkopf versehen ist (mit einer "tête chercheuse"), um dem Gegner einen gezielten Schaden zuzufügen.

Eine Variation dieses Themas ist der Fall eines Schwimmers, der einen Fluss durchqueren will, indem er immer einen festen Punkt anpeilt, der auf der anderen Flussseite liegt. Wir werden sehen, dass die Bahn des Schwimmers eine Art Parabel ist.

Hier folgt die Lösung dieses Problems.

3.8.1 Die Durchquerung eines Flusses.

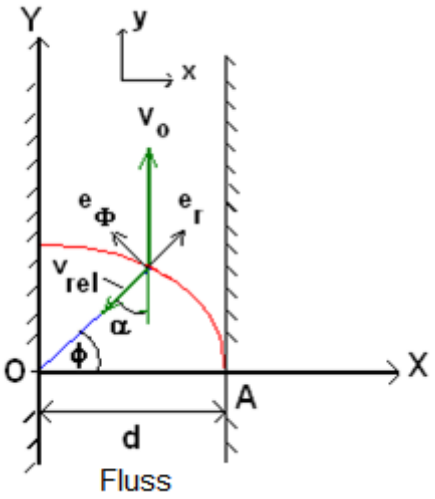

Fig.: 3.8-1

Der Schwimmer befindet sich auf der x-Achse im Punkt **A = (x_o,0).** Er durchschwimmt den Fluss mit der Relativgeschwindigkeit $\mathbf{v}_{rel}$. Dieser Vektor zeigt in jedem Augenblick zum Punkt O hin, wo sich z.B. seine Freundin befindet und ihn bei seiner Übung beobachtet. $\mathbf{v}_{rel} = -c\,\mathbf{e}_r$ ist die Geschwindigkeit des Schwimmers in Bezug auf den Fluss; $\mathbf{e}_r$ ist ein radialer Einheitsvektor. Die Entfernung OA ist d, und die auf (X,Y) bezogene Flussgeschwindigkeit ist $\mathbf{v}_0 = v_o\,\mathbf{j}$.

Welches ist also die vom Schwimmer beschriebene Bahn? In welchem Punkt **S** = (0,y_{end}) auf der gegenüberliegenden Flussseite wird er ankommen?

Lösung:

Wir wollen annehmen, dass das nichtinertiale System (x,y) sich mit dem Fluss bewegt, z.B. auf einem Boot befestigt, das sich mit der absoluten Geschwindigkeit $\mathbf{v}_o$ bewegt (gemessen vom Flussufer aus). In Gl. (4) des Paragraphen 3.5.1, d.h. $\mathbf{v}_{abs} = \mathbf{v}_{rel} + \mathbf{v}_o + \boldsymbol{\omega} \times \mathbf{r}$, setzen wir $\boldsymbol{\omega} = 0$ und schreiben vereinfachend $\mathbf{v}_{abs} := \mathbf{v}$.

Dann haben wir

$$\mathbf{v} = \mathbf{v}_{rel} + \mathbf{v}_o = -c\,\mathbf{e}_r + v_o\,\mathbf{j} \qquad (1)$$

Wir zerlegen $\mathbf{v}$ in seine radialen und transversalen Komponenten $\mathbf{v}_r$ und $\mathbf{v}_\varphi$: $\mathbf{v} = \mathbf{v}_r + \mathbf{v}_\varphi$.

Die Zerlegung von $\mathbf{v}_o$ gibt $\mathbf{v}_{o,r} = v_o \cos\alpha\,\mathbf{e}_r$ und $\mathbf{v}_{o,\varphi} = v_o \sin\alpha\,\mathbf{e}_\varphi$, wobei α der Winkel zwischen $\mathbf{v}_{rel}$ und ($-\mathbf{v}_o$) ist.

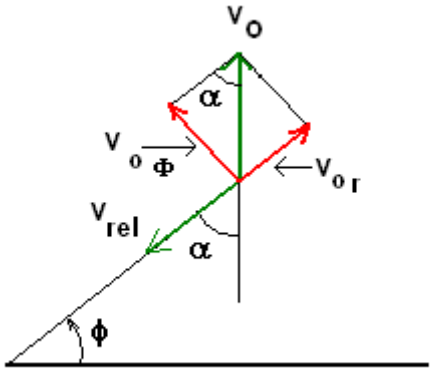

Fig.3.8-2

Der Vektor $\mathbf{v}_{rel}$ hat nur eine radiale Komponente: $\mathbf{v}_{rel} = -c\,\mathbf{e}_r = \mathbf{v}_r$. Also $\mathbf{v} = \mathbf{v}_r + \mathbf{v}_\varphi = (-c + v_o \cos\alpha)\,\mathbf{e}_r + v_o \sin\alpha \cdot \mathbf{e}_\varphi$. Damit folgt, vgl. 3.4.9, Gl. (4):

$$v_r = \frac{dr}{dt}\cdot\vec{e}_r = (-c + v_0\cos\alpha)\vec{e}_r$$
$$v_\varphi = r\frac{d\varphi}{dt}\cdot\vec{e}_\varphi = v_0\sin\alpha\cdot\vec{e}_\varphi \qquad (2)$$

In der zweiten Gleichung ersetzen wir $d\varphi/dt$ durch $-d\alpha/dt$, denn $\alpha = 90° - \varphi$ und $d\alpha/dt = -d\varphi/dt$. Wenn wir die erste Gl. durch die zweite teilen, ergibt sich $dr/r = (c - v_o cos\alpha)/(v_o\, sin\alpha)\cdot d\alpha$. Integration liefert

$$\ln(r) = (c/v_0)\cdot\ln(\tan(\alpha/2)) - \ln(\sin(\alpha)) + C \qquad (3)$$

Die Integrationskonstante C bestimmen wir mit Hilfe der Anfangsbedingungen α = α_o und r = d. Wir setzen r:= d und erhalten

$$r = d\,\frac{\sin\alpha_0}{\sin\alpha}\bullet\left(\frac{\tan\frac{\alpha}{2}}{\tan\frac{\alpha_0}{2}}\right)^{\frac{c}{v_0}} \qquad (4)$$

Wir setzen nun c = v_o und α_o = π/2. Dies ergibt

$$r = d\,\frac{\tan\frac{\alpha}{2}}{\sin\alpha} = \frac{d}{1+\cos\alpha} = \frac{d}{1+\sin\varphi} \qquad (5)$$

Diese Gleichung stellt eine Parabel dar. Das kann man leichter sehen, wenn man cartesische Koordinaten einführt: $r = (x^2+y^2)^{1/2}$ und sinφ = y/r.
Man erhält so: $d-y = (x^2 + y^2)^{1/2}$, d.h.

$$y = \frac{d}{2} - \frac{x^2}{2d} \qquad (6)$$

Also die Gleichung einer Parabel mit dem Scheitel in **S** = (0,d/2).

Im folgenden Programm 1 haben wir die Gl. (4) mit den Anfangsbedingungen d = 100 m und c = v_o = 5 m/s. α_o = PI/2. Das zweite Programm benutzt Gleichung (5).

Beachte in beiden Programmen die Verwendung der Anweisung
plot::Polar.

Programm 1

```
c:=5:v:=5:a0:=PI/2:d:=100://Schwimmer mit c =v
plot(plot::Polar([d*(sin(a0)/sin(a))*
  (tan(a/2)/tan(a0/2))^(c/v),a],a=0..a0))
```

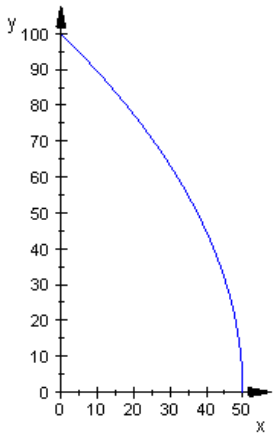

Fig.: 3.8-3

Programm 2

```
plot(plot::Polar([100/(1+sin(f)),f], f = 0..PI/2))
```

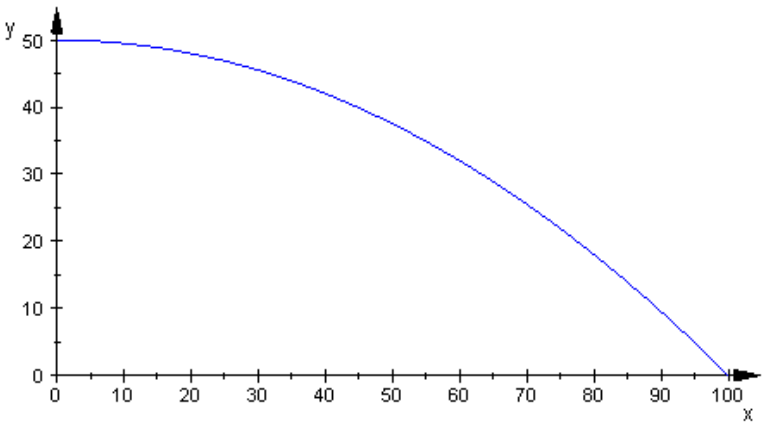

Fig.: 3.8-4

Wir wollen jetzt annehmen, dass das Geschwindigkeitsprofil des Flusses parabolisch ist: $\mathbf{V_o} = V(y)\,\mathbf{i} = (V - 4V(y-h/2)^2/h^2)\cdot\mathbf{i}$. An den Rändern des Flusses, wo y = 0 und y = h, haben wir die Geschwindigkeit 0. Im Zentrum, wo y = h/2 ist, hat der Fluss seine maximale Geschwindigkei V.

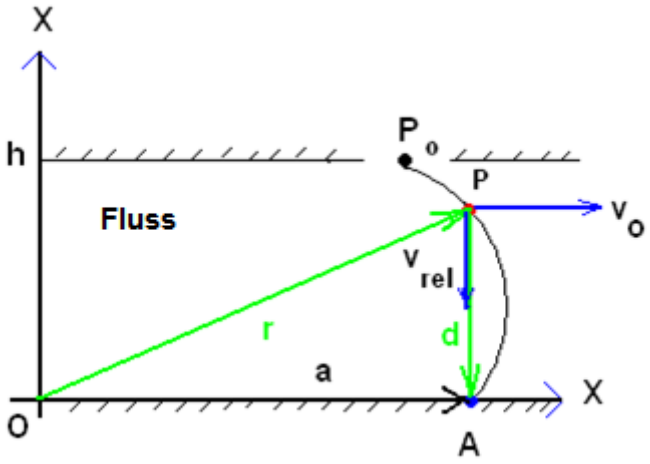

Fig.: 3.8-5

Die folgenden Beziehungen werden uns zu den Bewegungsgleichungen führen. $\mathbf{v} = \mathbf{v_{rel}} + \mathbf{V_o} = c\cdot\mathbf{c_o} + V(y)\cdot\mathbf{i}$, c = Geschwindigkeit des Schwimmers, $\mathbf{c_o}$ = Einheitsvektor in Richtung von $\mathbf{v_{rel}}$. Aus der Figur folgt $\mathbf{d} = \mathbf{a} - \mathbf{r}$, wo $|\mathbf{d}|$ die Entfernung zwischen P und A ist. Für den Betrag erhalten wir

$$d = |\mathbf{d}| = ((x_a - x)^2 + (y_a - y)^2)^{1/2}$$

Der Einheitsvektor ist $\mathbf{c_o} = \mathbf{d}/|\mathbf{d}| = (\mathbf{a} - \mathbf{r})/d$, und seine Komponenten sind

$$\mathbf{c_{o,x}} = (\mathbf{a}_x - \mathbf{r}_x)/d = (x_a - x)/d\cdot\mathbf{i} \quad \text{und} \quad \mathbf{c_{o,y}} = (\mathbf{a}_y - \mathbf{r}_y)/d = (y_a - y)/d\cdot\mathbf{j};\ y_a = 0$$

Für die Komponenten der absoluten Geschwindigkeit **v** erhalten wir jetzt

$$\mathbf{v}_x = c\cdot\,\mathbf{c_{o,x}} + V(y)\cdot\mathbf{i} = c(x_a - x)/d\cdot\mathbf{i} + V(y)\cdot\mathbf{i} = dx/dt\cdot\mathbf{i}$$

$$\mathbf{v}_y = c\cdot\,\mathbf{c_{o,y}} = c(y_a - y)/d\cdot\mathbf{j} = dy/dt\cdot\mathbf{j}$$

Das heißt:

$$\frac{dx}{dt} = c(x_a - x)/d + V(y)$$
$$\frac{dy}{dt} = c(y_a - y)/d$$

mit $y_a = 0$ (7)

Das sind die Bewegungsgleichungen des Schwimmers. Beide Gleichungen sind gekoppelt über d und V(y), denn d enthält x und y und V(y) enthält y.

Um das System (7) zweier Differentialgleichungen ersten Grades zu lösen, benutzen wir **numeric::ode2vectorfield** und **numeric::odesolve2**, wie wir es schon in den Abschnitten 2.5 e 3.2 taten. Eine weitere Methode werden wir in Paragraph 3.8.3 kennen lernen.

Das folgene Programm hat die Struktur des Programms in 2.5.1.

Die Anfangsbedingungen sind:
h = 15 m (Flussbreite)
x_a = 15m, y_a = 0 (Ort A des Ziels)
x_o = 5m, y_o = h (Startpunkt des Schwimmers)
c = 3 m/s (Relativgeschw. des Schwimmers in Bezug auf den Fluss)
V = 5 m/s (Maximalgeschwindigkeit des Flusses)

Der Schwimmer beginnt seine Bahn im Punkt $P_0(x_0,y_0)$, und in jedem Augenblick zeigt der Vektor $\mathbf{v}_{rel}$ des Schwimmers nach dem Zielort $A(x_a,y_a)$.

Programm 3

```
reset();//parabolisches Profil; Punktgrafik
h:=15://Flussbreite
x0:=5:y0:=h://Anfangsort des Schwimmers
xa:=15:ya:=0://Zielort auf der x-Achse
c:=3:V:=5:/* c = vrel des Schwimmers,V =
Maximalgeschw. des Flusses*/
V(y):= V-4*V*(y-h/2)^2/h^2:
//Flussgeschw.in Bezug auf y
d:=sqrt((xa-x(t))^2+(ya-y(t))^2+0.0001):
// 0.0001, um Division durch Null zu vermeiden

     IVP:={x'(t)=c*(xa-x(t))/d+V(y),y'(t)=c*(ya-
     y(t))/d,
     x(0)=x0,y(0)=y0}:
     fields:=[x(t),y(t)]:
```

```
ivp:=numeric::ode2vectorfield(IVP, fields):
Y := numeric::odesolve2(ivp): Y(5);//x(5s), y(5s)
 //Animation

dt:=0.1:imax:=100:
plot(
plot::Point2d(Y(t)[1],Y(t)[2], Color = RGB::Blue,
VisibleFromTo = t..t + 0.99*dt,
PointSize = 2*unit::mm)
$ t in [i*dt $ i = 0..imax],//Schwimmer

plot::Point2d(xa,ya, Color = RGB::Green,
VisibleFromTo = t..t + 0.99*dt,PointSize =
2*unit::mm)
$ t in [i*dt $ i = 0..imax],//Ziel

plot::Point2d(Y(t)[1], Y(t)[2], Color = RGB::Red,
VisibleAfter = t,PointSize = 1*unit::mm)
$ t in [i*dt $ i = 1..imax],
ViewingBox=[0..25,0..15])
```

```
[21.48824241, 2.873975916] // Position nach 5 Sekunden.
(Doppelklick auf den Graphen, um die Animation zu sehen.)
```

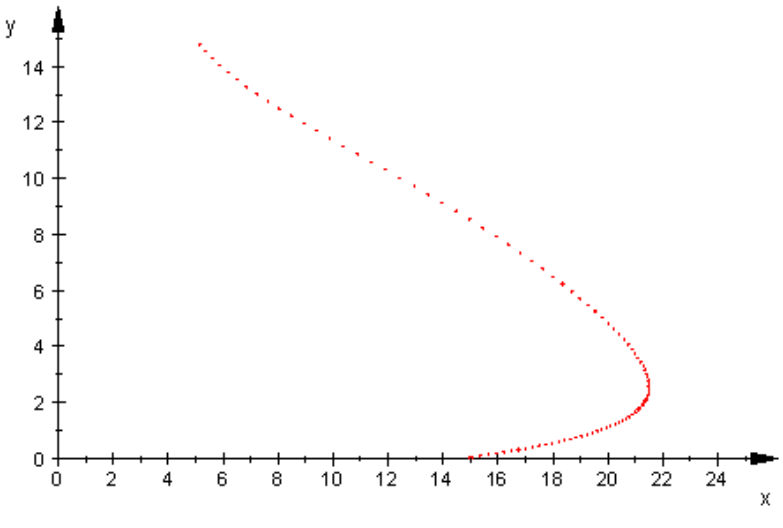

Fig.: 3.8-6

Wir haben die Punkte absichtlich nicht verbunden, um sehen zu können, wie sich die Geschwindigkeit des Schwimmers ändert. Er kann nur dann gegen den Strom ankommen, wenn die Flussgeschwindigkeit hinreichend klein ist, d.h. in der Nähe des Ufers. Dann kann der Schwimmer auch gegen die Strömung schwimmen und sein (grünes) Ziel in x_a=15 m auf der x-Achse erreichen.

Wenn wir eine durchgezogene Linie haben wollen, brauchen wir die vier letzten Programmzeilen nur durch die folgenden zu ersetzen:

```
plot::Line2d([Y(t - dt)[1], Y(t - dt)[2]],
[Y(t)[1], Y(t)[2]], Color = RGB::Red,
VisibleAfter = t)
$ t in [i*dt $ i = 1..imax],
ViewingBox=[0..25,0..15])
```

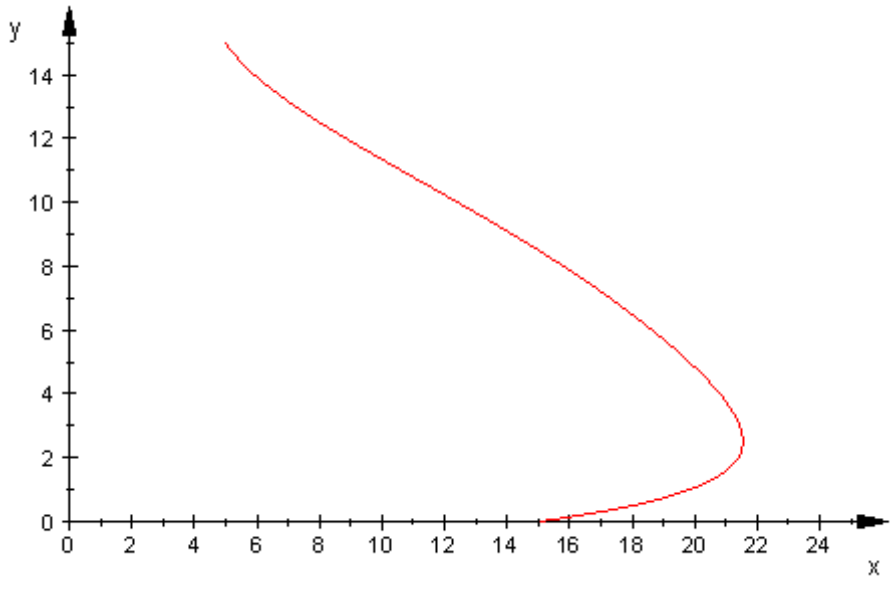

Fig.: 3.8-7

Wir werden nun das eigentliche Verfolgungsproblem behandeln: V ist Null und das Ziel bewegt sich mit der konstanten Geschwindigkeit c_2 = 1 m/s entlang der X-Achse. Den Schwimmer ersetzen wir durch einen Hund, der die Verfolgung seines Herrn im Punkt P_0 = (15,100) mit der Geschwindigkeit c_1 = 3 m/s beginnt. Im vorigen Programm haben wir x_a durch $x = c_2t$ zu ersetzen.

Programm 4:

```
reset();
//Herr(grün)und Hund(blau)sind beide sichtbar

    x0:=15:y0:=100:
    c1:=3:c2:=1://c1= vel. do cão
    d:=sqrt((c2*t-x(t))^2+y(t)^2+0.0001):

    IVP:={x'(t)=c1*(c2*t-x(t))/d,
    y'(t)=-c1*y(t)/d,x(0)=x0,y(0)=y0}:
    fields:=[x(t),y(t)]:
    ivp:=numeric::ode2vectorfield(IVP, fields):
    Y := numeric::odesolve2(ivp): Y(37.5);
    //Ort des Hundes nach 37.5 Sekunden

    //Animation

    dt:=0.5:imax:=100:
    plot(

    plot::Point2d(Y(t)[1],Y(t)[2], Color = RGB::Blue,
    VisibleFromTo = t..t + 0.99*dt,PointSize =
    2*unit::mm)
    $ t in [i*dt $ i = 0..imax],//Hund

    plot::Point2d(c2*t,0, Color = RGB::Green,
    VisibleFromTo = t..t + 0.99*dt,PointSize =
    2*unit::mm)
    $ t in [i*dt $ i = 0..imax],//Herr

    plot::Line2d([Y(t - dt)[1], Y(t - dt)[2]],
    [Y(t)[1], Y(t)[2]], Color = RGB::Red,

    VisibleAfter = t)

    $ t in [i*dt $ i = 1..imax],//Hund
    plot::Line2d([c2*(t - dt),0],
    [c2*t, 0], Color = RGB::Blue,VisibleAfter = t)
    $ t in [i*dt $ i = 1..imax],//Herr
    ViewingBox=[0..50,0..120])
```

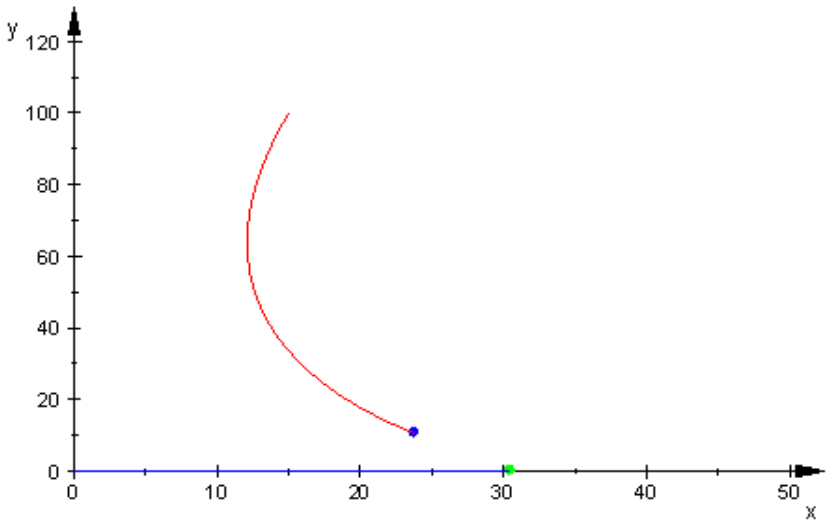

Fig.: 3.8-8

3.8.3 Die EULER-Methode

Man kann mit MUPAD eine Differentialgleichung lösen (integrieren) und sogar ein System derartiger Gleichungen. Es gibt dafür sehr komplexe Methoden. Die numerischen Ergebnisse können fast beliebig genau sein.

Nun zeigt es sich aber, dass wir in vielen Fällen keine supergenauen Resultate benötigen, da wir oft nur einen flüchtigen Eindruck vom gewünschten Ergebnis erhalten wollen.

Um einen schnellen Überblick zu erhalten, ersetzen wir eine Ableitung dx/dt durch den Differenzenquotienten $\Delta x/\Delta t = (x(t + \Delta t)-x(t))/\Delta t$, was wir als $(x_{n+1} - x_n)/\Delta t$ schreiben wollen.

x_{n+1} ist die x-Koordinate eines Körpers im Augenblick $t+\Delta t$, und x_n ist die x-Koordinate im früheren Zeitpunkt t. Dann ist $(x_{n+1} - x_n)/\Delta t$ die mittlere Geschwindigkeit im Zeitintervall Δt.

Die mittlere Geschwindigkeit sagt uns nichts über die Geschwindigkeit des Körpers in einem bestimmten Augenblick. Aber wenn wir die mittlere Geschwindigkeit in immer kleineren Zeitintervallen Δt bestimmen, gelangen wir mit immer besserer Näherung zur wahren Geschwindigkeit im Zeitpunt t.

Die erste der Gleichungen (7) können wir dann näherungsweise ersetzen durch $(x_{n+1} - x_n)/\Delta t = c\,(x_a - x_n)/d + V(y_n)$, wobei wir Δt hinreichend klein wählen. Für die Entfernung d haben wir den Ausdruck

$$d = ((x_a - x_n)^2 + (h/2 - y_n)^2)^{1/2} .$$

(Manchmal ist es bequemer, die vorige Gleichung folgendermaßen zu schreiben: $(x_n - x_{n-1})/\Delta t = c\,(x_a - x_{n-1})/d + V(y_{n-1})$. In den folgenden Programmen werden wir diese Schreibweise benutzen.)

Um die Bahn des Körpers zu berechnen, beginnen wir die Rechnung im Punkt (x_0,y_0). Mit Hilfe der Gleichung

$$x_{n+1} = x_n + c\,(x_a - x_n)\,\Delta t/d + V(y_n)\,\Delta t \qquad (8)$$

können wir x_1 berechnen, da x_0 bekannt ist. Mit dem so gefundenen x_1 berechnen wir dann $x_2 = x_1 + c\,(x_a - x_1)\,\Delta t/d + V(y_1)\,\Delta t$ -danach x_3 etc.
Man nennt diese Art der Berechnung eine **Iteration**, und Gl. (8) heißt Iterationsgleichung.

Mit der y-Koordinate verfahren wir auf die gleiche Weise. Wir beginnen in y_0 und berechnen $y_1 = y_0 + c\,(y_a - y_0)\,\Delta t/d$, danach y_2, y_3 usw.

Dieses Verfahren zur Bestimmung einer Näherungslösung einer Differentialgleichung wird EULER-Verfahren genannt (*Leonhard Euler, 1707-1783*). Die von MUPAD benutzten Methoden sind natürlich viel exakter und komplizierter. Später werden wir uns genauer mit einigen dieser feineren Verfahren beschäftigen.

Um die Einfachheit des EULER-Verfahrens zu demonstrieren, schreiben wir ein kleines Programm für unseren Schwimmer im Fluss. Dieses Mal legen wir den Koordinatenursprung in die Mitte des Flusses mit der Breite h = 15 m. Der Schwimmer startet im Punkt P_0 = (5 m,-7.5 m). Das Ziel liegt in A = (15 m,7.5 m). Das Geschwindigkeitsprofil des Flusses ist wieder parabolisch: $V(y) = V - 4V/h^2 \cdot y^2$.

Programm 5

```
reset():EULER-Verfahren
x0:=5:xa:=15:v:=5:h:=15:c:=3:
final:=100://Zahl der Punkte
DIGITS:=4:
x[0]:=x0:y[0]:=-h/2:dt:=0.1:
coord:=array(1..final,1..3)://Array der Koordinaten
for n from 1 to final do

     d:=((xa-x[n-1])^2+(h/2-y[n-1])^2)^0.5:
     x[n]:=x[n-1]+(xa-x[n-1])*c*dt/d +(v-
     (4*v/h^2)*y[n-1]^2)*dt:
     y[n]:=y[n-1]+(h/2-y[n-1])*c*dt/d:

     coord[n,1]:= n://Elemente des Arrays
     coord[n,2]:=x[n-1]:
     coord[n,3]:=y[n-1]:

     end_for:

     for n from 1 to final do
        print(n,x[n],y[n])
     end_for:

     plot(plot::Point2d([x[n],y[n]])$ n=1..final,
     ViewingBox =[0..25,-10..10])
```

```
0, 5, -15/2 // Ergebnisse: n, x(n), y(n)
1, 5.166, -7.25
2, 5.366, -7.001
3, 5.596, -6.751
4, 5.856, -6.5
5, 6.144, -6.249
6, 6.46, -5.997
7, 6.8, -5.744
8, 7.165, -5.489
9, 7.552, -5.232
10, 7.961, -4.973
```

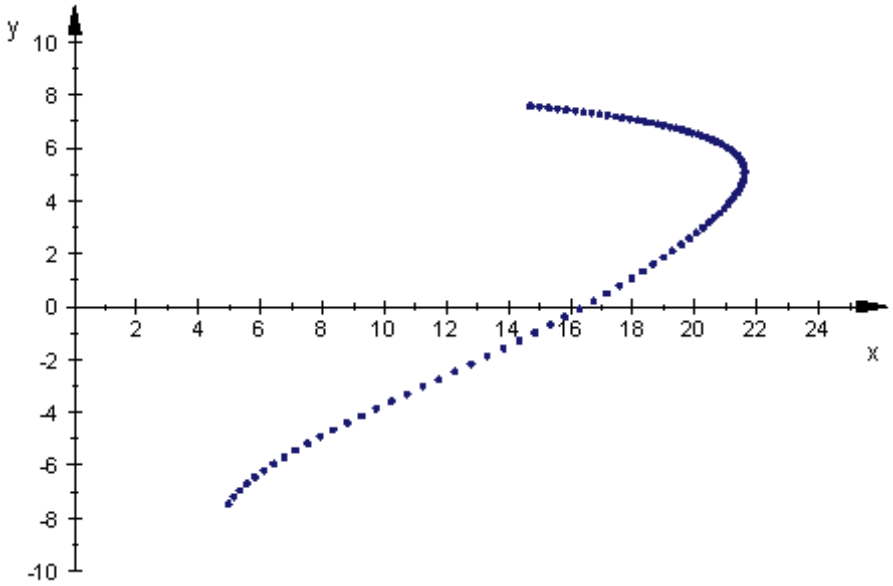

Fig.3.8-9

Man muss diese Trayektorie mit Fig. 3.8-6 vergleichen, die dieselbe Bewegung beschreibt, aber ein anderes Koordinatensystem benutzt.
Übrigens können Sie die Konstanten auch interaktiv eingeben:

```
input("x0=",x0,"xa=",xa,"v=",v,"h=",h,"c=",c,"final",final)
:
```

Im folgenden Programm wurden einige Kontrollstrukturen eingeführt wie

```
if (float(d) < 0.1) then final :=n;
```

und `if (n=0) or(modp(n,10)=0) then`

wie auch Anfangspunkt und Endpunkt.

Das erste `if ...then` zusammen mit `break` unterbricht die Rechnungen, wenn die Entfernung kleiner als 0.1 ist.

Die zweite Bedingung sorgt dafür, dass nur die Anfangsbedingungen und nur jedes zehnte Resultat ausgegeben werden.

Programm 6

```
reset():EULER-Methode
xo:=5:xa:=15:v:=5:h:=15:c:=3:
final:=100://Anzahl der Punkte
DIGITS:=4:

x[0]:=xo:y[0]:=-h/2:dt:=0.1:

coord:=array(1..final,1..3)://Array der Koordinaten

for n from 1 to final do

d:=((xa-x[n-1])^2+(h/2-y[n-1])^2)^0.5:
x[n]:=x[n-1]+(xa-x[n-1])*c*dt/d +(v-(4*v/h^2)*y[n-1]^2)*dt:
y[n]:=y[n-1]+(h/2-y[n-1])*c*dt/d:

          /*coord[n,1]:= n:
          coord[n,2]:=x[n-1]:
          coord[n,3]:=y[n-1]:*/

if (float(d) < 0.1) then final :=n;

print("ENDE, n = ",n); break;

end_if:
end_for:

for n from 0 to final do
if (n=0) or (modp(n,10)=0) then
print(n,x[n],y[n]);
end_if://Nur jedes zehnte Ergebnis wird ausgeben
end_for:

      //Graf

          plot(
          plot::Point2d([x[n],y[n]])$ n=0..final,
          plot::Point2d([x[final],y[final]],
          PointSize =2*unit::mm,Color=RGB::Green),
```

```
            plot::Point2d([x[0],y[0]],PointSize
            =2*unit::mm,Color=RGB::Red),

        plot::Text2d("Start",[x[0],y[0]],
        HorizontalAlignment=Right),
        plot::Text2d("Ziel",[x[final],y[final]],
        HorizontalAlignment=Right),
        ViewingBox =[0..25,-10..10])

  "ENDE, n = ", 94

0, 5, -15/2
  10, 7.961, -4.973
  20, 12.83, -2.224     etc.
```

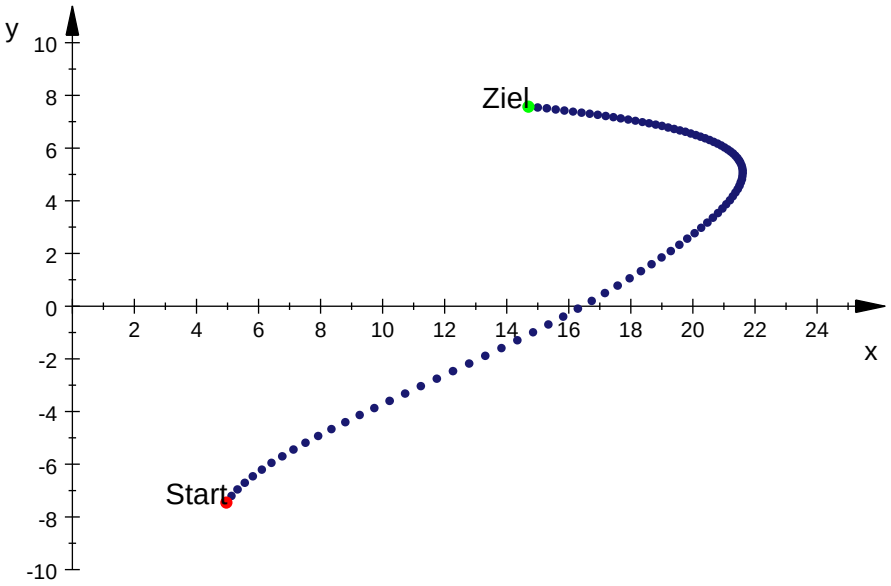

Fig.: 3.8-10

3.8.4 Mit Bleistift und Papier

Ein Boot startet im Punkt A(0,0), um einen Fluss zu überqueren, der mit konstanter und gleichförmiger Geschwindigkeit Vo = 5 ft/s fließt. (Mit den Einheiten ft und ft/s erhalten wir einfache Ergebnisse.) Die Flussbreite beträgt 100 ft. Das Boot hat in Bezug auf den Fluss die Geschw. c = 10 ft/s.

Welche Zeit wird das Boot brauchen, um auf geradliniger Bahn von A bis C(50,100) zu gelangen?

Lösung:

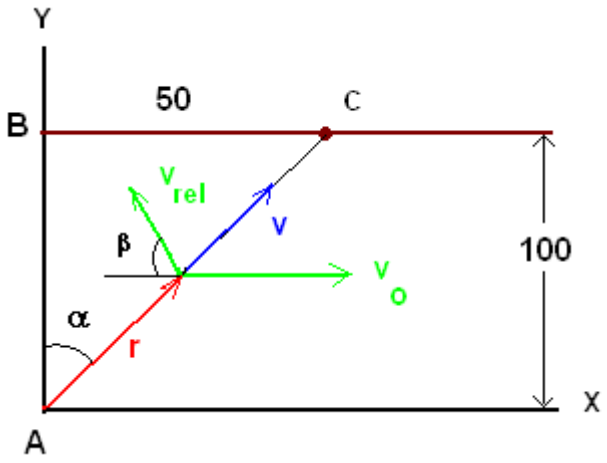

Fig.: 3.8-11

Die Absolutgeschwindigkeit **v** muss die Richtung des Vektors **AC** haben, der eine Länge von 111,80 ft hat. Die Relativgeschwindigkeit des Bootes ist $\mathbf{v_{rel}}$ = -10 cosβ **i** + 10 senβ **j.**
Für die Absolutgeschwindigkeit erhalten wir

$$\mathbf{v} = v\,\text{sen}\alpha\,\mathbf{i} + v\,\cos\alpha\,\mathbf{j} = -10\,\cos\beta\,\mathbf{i} + 10\,\text{sen}\beta\,\mathbf{j} + 5\mathbf{i}$$

Durch Einsetzen der Zahlenwerte erhalten wir die folgenden Ausdrücke für v:

$v \cdot 0{,}447 = -\cos\beta + 5$ e $v \cdot 0{,}894 = 10 \cdot \operatorname{sen}\beta$.

Quadrieren und addieren liefert:

$$v^2 - 4{,}47 \cdot v + 25 = 100.$$

Die Lösung dieser Gleichung ergibt v = 11,18 ft/s, und die gesuchte Zeit beträgt t = 111,80/11,18 = 10 s.