2 Die Newtonschen Gesetze

2.2 Vektoren

2.2.1 Grundbegriffe

In diesem Abschnitt besprechen wir die Grundbriffe der Vektorrechnung.

Eine Kraft ist ein typisches Beispiel eine Größe, die als Vektor dargestellt wird. Weitere Beispiele sind Geschwindigkeit und Beschleunigung. Der Begriff Vektor kann sich jedoch allgemeiner auf Systeme gleichartiger Objekte beziehen, beispielsweise auf Systeme linearer Gleichungen oder auf Systeme von Differentialgleichungen.

Wir wissen bereits, dass ein räumlicher Vektor ein Objekt ist, dem Länge, Richtung und Richtungssinn zukommt. (Parallele Geraden mit demselben Richtungssinn definieren einen einzigen Richtungssinn.) Zwei Vektoren sind *gleich,* wenn sie in allen drei Eigenschaften übereinstimmen.

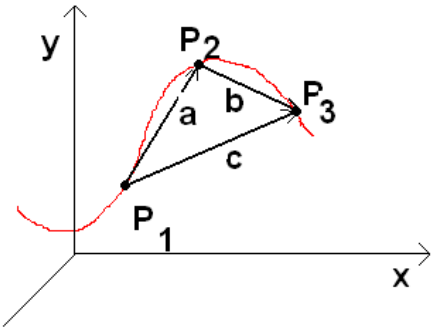

Fig.2.2.-1

Ein Verschiebungsvektor hängt nicht von der Bahn eines Teilchens ab, er wird nur durch Anfangs-und Endpunkt festgelegt. Die Verschiebung $\overrightarrow{c}$ ist äquivalent zu den aufeinander folgenden Verschiebungen $\overrightarrow{a}$ und $\overrightarrow{b}$, d.h. $\overrightarrow{c} = \overrightarrow{a} + \overrightarrow{b}$. Wir haben hier die *Summe* zweier Vektoren nach der Polygonregel gebildet, d.h. wir legen den Anfang eines Vektors in das Ende des vorhergehenden. Der Summenvektor geht vom Anfang des ersten bis zum Ende des letzten Vektors.

Ein Vektor kann auch als eine **Matrix** aufgefasst werden, die aus nur einer Spalte oder Zeile besteht. Wir werden diese Darstellung besonders oft benutzen.

2.2.2 Zeichnung von Vektoren

Will man in MuPAD einen Vektor darstellen, der seinen Ursprung in P = (1,2) und sein Ende in Q = (2.5,5) hat, so benutzt man den `plot::Arrow2d` - Befehl.

```
v1:=plot::Arrow2d([1,2],[2.5,5], Color=RGB::Red):
plot(v1)
```

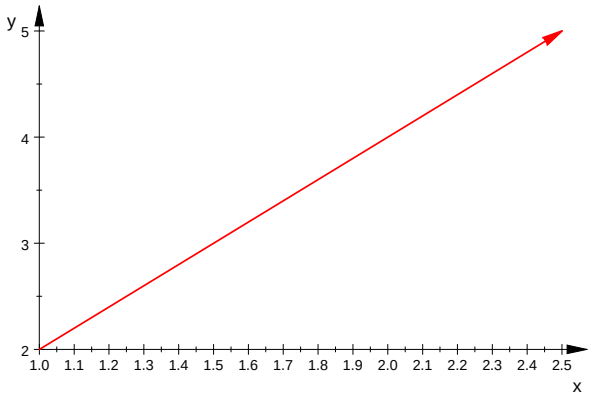

Fig.2.2-2

Man kann auch den bloßen Vektor zeichnen, d.h. ohne die Koordinatenachsen. Statt `plot(v1)` schreibt man `plot(v1,Axes=None)`.

Wir wollen nun drei Vektoren mit dem gleichen Ursprung O = (0,0) zeichnen (= Ortsvektoren), wobei wir verschiedene Stile benutzen.

```
plot(plot::Arrow2d([1, 1], Color = RGB::Red,
TipStyle = Open, TipLength = 10*unit::mm),
plot::Arrow2d([-1, 1], Color = RGB::Green,
LineWidth = 0.8*unit::mm,
TipStyle = Closed, TipAngle = PI/2),
plot::Arrow2d([0, -sqrt(2)], Color = RGB::Blue,
LineStyle = Dashed))
```

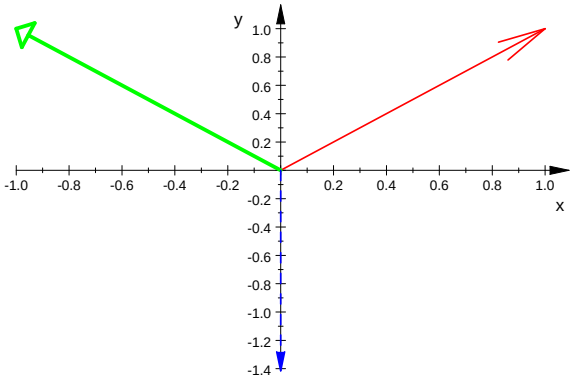

Fig.2.2-3

Der blaue Vektor ist gestrichelt gezeichnet: `LineStyle = Dashed`

Im folgenden Bild werden wir zwei Ortsvektoren mit Spitzen in x1 = (2,2) und x2 = (2.5,5) zusammen mit ihrer Summe x3 := x1 + x2 darstellen.

```
x1:=matrix([[2,2]]):x2:=matrix([[2.5,5]]):
x3:= x1+x2;
v1:=plot::Arrow2d([x1[1],x1[2]]):
```

```
v2:=plot::Arrow2d([x2[1],x2[2]]):
v3:=plot::Arrow2d([x3[1],x3[2]],Color=RGB::Red,TipStyle = Open):
info1:=plot::Text2d("Vektor 1", [x1[1],x1[2]],
HorizontalAlignment = Left,
VerticalAlignment = BaseLine):
info2:=plot::Text2d("Vektor 2", [x2[1],x2[2]],
HorizontalAlignment = Left,
VerticalAlignment = BaseLine):
info3:=plot::Text2d("Summe", [x3[1],x3[2]],
HorizontalAlignment = Left,
VerticalAlignment = BaseLine):
plot(v1,v2,v3,info1,info2,info3)
```

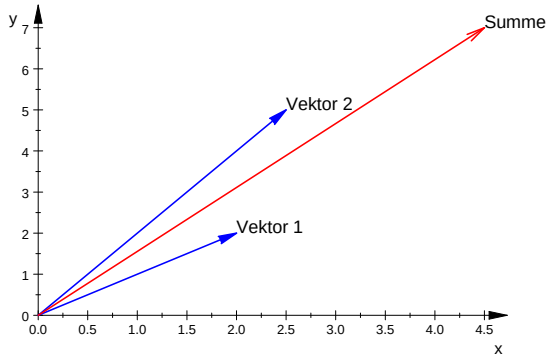

Fig.2.2-4

Beachten Sie, wie man in MuPAD Matrizen darstellt!

Die Beschriftung benötigt eine komplizierte `plot::Text2d`-Anweisung. Probieren Sie aus, welche Auswirkung andere `Alignement`-Angaben haben.

Für MuPAD ist ein Vektor eine Spalten-oder eine Zeilenmatrix. $\begin{pmatrix} 1 \\ -5 \\ 7 \end{pmatrix}$ ist einen Spaltenmatrix (n = 1) mit m = 3 Zeilen.

Man kann sie in MuPAD als `A:= matrix([[1],[-5],[7]])` darstellen, oder einfacher als `A:= matrix([1,-5,7])`.

Bei einer Zeilenmatrix benutzt MuPAD 2 Paare eckiger Klammern. Die Zeilenmatrix (3 -2 1), also m = 1 und n = 3, schreibt sich in MuPAD `A: matrix([[3,-2,1]])`.

2.2.3 Analytische Darstellung der Vektoren

Die graphische Darstellung von Vektoren ist selten gewünscht. In der Praxis beziehen wir einen Vektor auf eine vektorielle Basis und rechnen nur mit den Koordinaten des Vektors. Besonders bequem ist die Verwendung einer Basis, deren Vektoren senkrecht aufeinander stehen und die Länge 1 haben (orthonormale Basis). In zwei Dimensionen mit den Basisvektoren $\vec{i}$, $\vec{j}$ schreiben wir

einen beliebigen Vektor $\overrightarrow{a}$ in der Form $\overrightarrow{a} = a_x \overrightarrow{i} + a_y \overrightarrow{j}$.Die Zahlen a_x, a_y sind die Koordinaten des Vektors $\overrightarrow{a}$.(Statt Koordinaten sagt man oft Komponenten. Diese Ausdrucksweise ist nicht zu empfehlen, da Komponenten Vektoren sind. Z.B. ist $a_x \overrightarrow{i}$ die x-Komponente von $\overrightarrow{a}$ in der Basis $\left[\overrightarrow{i}, \overrightarrow{j}\right]$.)

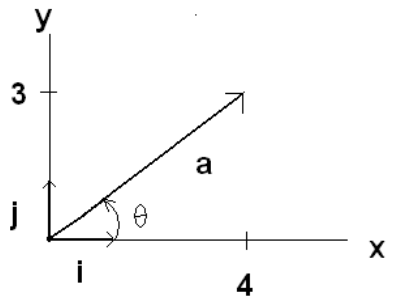

Fig.2.2-5

In der Abbildung haben wir den Vektor $\overrightarrow{a} = 4\overrightarrow{i} + 3\overrightarrow{j}$ dargestellt. Den Zusammenhang zwischen Länge a , Winkel θ und Koordinaten a_x, a_y sehen wir in den folgenden Formeln

$$\begin{aligned} a_x &= a\cos\theta \\ a_y &= a\sin\theta \\ a &= \sqrt{a_x^2 + a_y^2} \end{aligned} \tag{2.2-1}$$

Die Orientation des Vektors finden wir mit

$$\tan\theta = \frac{a_x}{a_y} \tag{2.2-2}$$

In unserem Fall ist $a = 5$ und $\theta = \arctan\frac{3}{4} = 0.643\,5 * 180/\pi = 36.\,870°$

Bei MuPAD geben wir den Vektor als Matrix ein: `A:=matrix([[4,3]])`und bestimmen die sogenannte 2-Norm, d.h. den Betrag (Länge): `norm(A,2)`. Die ganze Rechnung sieht dann folgendermaßen aus;

```
A:=matrix([[4,3]]):
norm(A,2);
arctan(3/4);
float(%*180/PI)
```

Um einen Vektor mit einer Zahl zu multiplizieren, hat man die Koordinaten mit dieser Zahl zu multiplizieren:

$c\overrightarrow{a} = (ca_x)\overrightarrow{i} + (ca_y)\overrightarrow{j}$. Man addiert zwei Vektoren, indem man ihre Koordinaten addiert: $\overrightarrow{a} + \overrightarrow{b} = (a_x + b_x)\overrightarrow{i} + (a_y + b_y)\overrightarrow{j}$. Das Produkt zweier Vektoren kann zweifach definiert werde, nämlich als Skalar-Produkt oder als

Vektor-Produkt. Das Skalar-Produkt wird wesentlich häufiger benutzt als das vektorielle Produkt. Seine Definition lautet

$$\overrightarrow{a} \cdot \overrightarrow{b} = ab\cos\theta \tag{2.2-3}$$

Aus dieser Definition kann man die folgende herleiten

$$\overrightarrow{a} \cdot \overrightarrow{b} = a_x b_x + a_y b_y \tag{2.2-4}$$

In MuPAD gibt es eine "linalg"-Bibliothek, die eine große Anzahl von Formeln aus der linearen Algebra enthält, unter anderem die Funktion `linalg::scalarProduct(A,B):`

```
A:=matrix([-2,3]): B:=matrix([1.2,-2.5]):
linalg::scalarProduct(A,B)
```

```
-9.9
```

Dieses Ergebnis können wir auch erhalten durch Matrizen-Multiplikation. Wir können aber nicht einfach A*B schreiben, denn das liefert eine Fehlermeldung. Wir müssen vorher die erste Matrix als Zeilenmatrix schreiben, d.h. `matrix([[-2,3]])`, weil die Spaltenzahl n_1 der ersten Matrix gleich sein muss der Zeilenzahl m_2 der zweiten: (m_1,n_1)(m_2,n_2) = (1,**1**)(**1**,1) =1-1-Matrix.

```
A:=matrix([[-2,3]]): B:=matrix([1.2,-2.5]):
A*B
(-9.9)
```

Die Produktmatrix besteht aus einer einzigen Zahl, sie ist eine 1-1-Matrix.

Die Elemente c_{ik} der Produktmatrix berechnet man nach $c_{ik} = a_{ir} \cdot b_{rk}$. c_{ik} ist das Element in der i. Zeile und der k.Spalte der Produktmatrix. Man erhält c_{ik}, indem man die Elemente der i. Zeile der 1. Matrix mit den entsprechenden Elementen der k. Spalte der zweiten Matrix multipliziert. Beachte: n_1= Spaltenzahl der 1. Matrix muss gleich sein m_2 = Zeilenzahl der 2. Matrix.

$$A * B = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} * \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} =$$

$$\begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{pmatrix} \tag{2.2-5}$$

Hier folgt ein Zahlenbeispiel, das wir zunächst mit SWP behandeln. Wir geben die Matrix A ein

$$A = \begin{pmatrix} 3 & 1 \\ 5 & 2 \\ 4 & 3 \end{pmatrix}$$

und benutzen *Compute* > *Definitions*, um die Matrix A zu definieren. Anschließend geben wir B ein

$$B = \begin{pmatrix} -1 & -1 \\ 1 & 4 \end{pmatrix}$$

und definieren die Matrix B. Nun können wir mit *Compute* > *Evaluate* das Produkt AB berechnen:

$$AB = \begin{pmatrix} -2 & 1 \\ -3 & 3 \\ -1 & 8 \end{pmatrix}$$

Man kann sich auch die Matrixelemente mit *Compute* > *Evaluate* anzeigen lassen

$A_{2,2} = 2$, $B_{2,1} = 1$, $AB_{3,2}$ kann man sich erst anzeigen lassen, wenn man $C = AB$ definiert hat: $C_{3,2} = 8$

Nun folgen noch die MuPAD-Anweisungen:

```
A:=matrix([[3,1],[5,2],[4,3]]):
B:=matrix([[-1,-1],[1,4]]):
A*B;
C:=%//C=A*B
```

Die Matrixelemente lassen wir uns mit A[i,k] anzeigen, z.B. C[3,2] = 8

Mit Gl. 2.2-3 können wir den *Winkel* zwischen zwei Vektoren finden, aber es ist praktischer, die Funktion `linalg::angle` einzusetzen:

```
theta:=linalg::angle(
matrix([2,5]),matrix([-3,3])):
float(theta*180/PI)
```

```
66.80140949
```

In 2.1.1 sind wir auch schon dem **Vektorprodukt** zweier Vektoren $\overrightarrow{a}$, $\overrightarrow{b}$ begegnet. Es ist ein neuer Vektor, den wir mit $\overrightarrow{c} = \overrightarrow{a} \times \overrightarrow{b}$ bezeichnen wollen. $\overrightarrow{c}$ ist genau dann gleich dem Nullvektor, wenn $\overrightarrow{a}$, $\overrightarrow{b}$ kollinear sind. Sollten sie nicht kollinear sein, so ist der Betrag des Produktvektors gleich dem Betrag der Fläche des von $\overrightarrow{a}$, $\overrightarrow{b}$ aufgespannten Parallelogramms. D.h.

$$c = a \cdot b \cdot \sin\theta \tag{2.2-6}$$

Der Richtungssinn des Produktvektors $\overrightarrow{c}$ ist gegeben durch die Bewegung einer rechtsgängigen Schraube, wenn man den Vektor $\overrightarrow{a}$ auf kürzestem Weg in Richtung von Vektor $\overrightarrow{b}$ dreht. Diese der Mathematik fremde Definition zeigt schon, dass das Kreuzprodukt eigentlich ein *Pseudovektor* ist. Er kann sehr nützlich sein, auch wenn er kein eigentlich mathematisches Objekt ist.

Dreidimensionale Vektoren werden oft in der Standardbasis [$\overrightarrow{i}$, $\overrightarrow{j}$, $\overrightarrow{k}$]

geschrieben mit

$$\begin{aligned} \mathbf{i} &= (1,0,0) \\ \mathbf{j} &= (0,1,0) \\ \mathbf{k} &= (0,0,1) \end{aligned}$$

Das Kreuzprodukt (= Vektorprodukt) der beiden Vektoren $\overrightarrow{a} = a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}$ und $\overrightarrow{b} = b_1\mathbf{i} + b_2\mathbf{j} + b_3\mathbf{k}$ kann man nach dem folgenden Schema als Determinante berechnen:

$$\begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = \mathbf{i}\,(a_2b_3 - a_3b_2) - \mathbf{j}\,(a_1b_3 - b_1a_3) + \mathbf{k}\,(a_1b_2 - a_2b_1)$$

Hier ist ein MuPAD-Beispiel:

Gegeben sind die Vektoren $\overrightarrow{a} = 3\mathbf{i} + (-4)\mathbf{j} + 0\mathbf{k}$, sowie $\overrightarrow{b} = -2\mathbf{i} + 0\mathbf{j} + 3\mathbf{k}$. Berechne $\overrightarrow{c} = \overrightarrow{a} \times \overrightarrow{b}$

```
a:=matrix([[3,-4,0]]):
b:=matrix([[-2,0,3]]):
linalg::crossProduct(a,b)
```

```
(-12 -9 -8)
```

Also: $\overrightarrow{c} = -12\overrightarrow{i} - 9\overrightarrow{j} - 8\overrightarrow{k}$

Mit SWP errechnet man das Kreuzprodukt mit *Compute* > *Evaluate*:

$(3,-4,0) \times (-2,0,3) = \begin{bmatrix} -12 & -9 & -8 \end{bmatrix}$

das funktioniert auch in allgemeiner Form:

$(a_1,a_2,a_3) \times (b_1,b_2,b_3) = \begin{bmatrix} a_2b_3 - a_3b_2 & a_3b_1 - a_1b_3 & a_1b_2 - a_2b_1 \end{bmatrix}$

2.2.4 Vektoren und Parameterkurven

Oft werden wir eine ebene Kurve in Parameterform schreiben

$$x = x(t), y = y(t),\ a \le t \le b \tag{2.2-7}$$

wobei der Parameter t als Zeit interpretiert werden kann. Die Länge s der Kurve zwischen a und b ist dann der vom Massepunkt in dieser Zeit zurückgelegte Weg. Die Ableitungen dx/dt und dy/dt können als Koordinaten der Geschwindigkeit des sich bewegenden Punktes (x, y) aufgefasst werden.

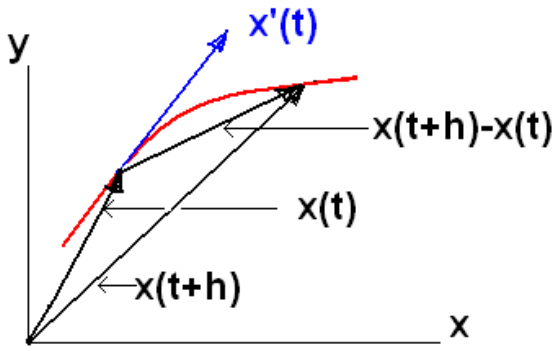

Fig.2.2-6

Der Geschwindigkeitsvektor $\overrightarrow{x'(t)}$ ist die Tangente im Zeitpunkt t an die von 2.2-7 definierte Kurve $\overrightarrow{x(t)}$. Der Betrag des Geschwindigkeitsvektors ist gegeben durch

$$\frac{ds}{dt} = \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2} \tag{2.2-8}$$

Der Vektor $\overrightarrow{x(t+h)} - \overrightarrow{x(t)}$ ist eine Sekante, die für $\lim_{h\to 0}$in die Tangente in $\overrightarrow{x(t)}$ übergeht. (Diese Grenzlage der Sekante definiert eigentlich die Tangente.) Das Produkt aus $\overrightarrow{x(t+h)} - \overrightarrow{x(t)}$ und h^{-1} ist ein Vektor und sein Grenzwert ist der Vektor $d\ \overrightarrow{x}/dt$. Wir schreiben dies wie folgt

$$\frac{d\ \overrightarrow{x}}{dt} = \lim_{h\to 0} \frac{\overrightarrow{x(t+h)} - \overrightarrow{x(t)}}{h} \tag{2.2-9}$$

Man bildet die Ableitung einer vektoriellen Funktion, indem man jede Koordinate einzeln ableitet.

Ein **Kreis** mit dem Radius r ist in Parameterform gegeben durch

$$x(t) = r\sin(t), y(t) = r\cos(t);\ 0 \le t < 2\pi \tag{2.2-10}$$

Eine **Ellipse** mit Halbachsen a und b hat die Darstellung

$$x(t) = a\cos(t), y(t) = b\sin(t);\ 0 \le t < 2\pi \tag{2.2-11}$$

Beispiel: Wir zeichnen nun einen Kreis mit $r = 2$ und daruf den Punkt, mit $t = \pi/4$. Die Koordinaten dieses Punktes sind $x(\pi/4) = y(\pi/4) = 1.4142$.

In diesem Punkt zeichnen wir die Vektoren $\overrightarrow{v}$ und $\overrightarrow{a}$. Mit Hilfe eines Skalierungsfaktors reduzieren wir die Längen der Vektoren. Beide haben eine feste Länge, aber verändern fortwährend die Richtung. Sie stehen stets senkrecht aufeinander. Die folgende Figur zeigt einen Massepunkt, der sich im Uhrzeigersinn auf einer Kreisbahn bewegt. (Eine Bewegung im Gegenuhrzeigersinn wird von $x(t) = r\cos(t)$ und $y(t) = r\sin(t)$ beschrieben.)

Programm 1 (Momentaufnahme mit $t = \pi/4$)

```
x:=t->2*sin(t):
y:=t->2*cos(t):
curve:=plot::Curve2d([x(t),y(t)],t= 0..2*PI):
pos:=t->(x(t),y(t)):Ortsvektor
vel:=t->(x'(t),y'(t)):Geschwindigkeitsvektor
acel:=t->(x''(t),y''(t)):Beschleunigung
t1:=PI/4:Zeitpunkt
scale:=0.6:Skalenfaktor
x1:=pos(t1)[1]:Anfang des Vektors (x'(t),y'(t))
y1:=pos(t1)[2]:fuer t=t1
v1:=vel(t1)[1]:Geschw. im Punkt (x(t),y(t))
v2:=vel(t1)[2]:fuer t=t1
a1:=acel(t1)[1]:Beschleunigung
a2:=acel(t1)[2]:
x2:=x1+v1*scale:Spitze des Vektors (x'(t),y'(t))
y2:=y1+v2*scale:
x3:=x1+a1*scale:Spitze des Vektors (x''(t),y''(t))
y3:=y1+a2*scale:
p:=plot::Point2d(x1,y1,Color=RGB::Green,PointSize=3*unit::mm):
ve:=plot::Arrow2d([x1,y1],[x2,y2],Color=RGB::Red):
ac:=plot::Arrow2d([x1,y1],[x3,y3],Color=RGB::Green):
plot(curve,p,ac,ve,Scaling = Constrained)
```

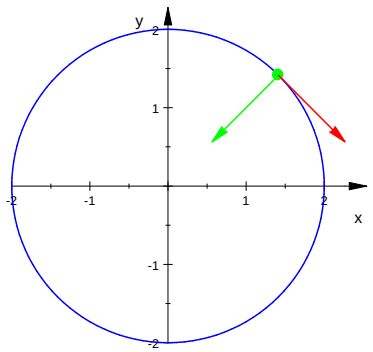

Fig.2.2-7

Mit Hilfe der Instruktion `Scaling=Constrained` erreichen wir, dass die Darstellung nicht verzerrt erscheint. Beim Zeichnen einer Flugbahn wollen wir das u.Umst. jedoch nicht, da Höhe und Weite sehr verschieden sein können. In diesem Fall werden wir diese Istruktion weglassen. Beachten Sie, dass der grüne Beschleunigungsvektor auf das Bewegungszentrum hinweist, während die rote Geschwindigkeit tangential in Bewegungsrichtung zeigt.

Wir nehmen nun die 7. Zeile (`t1:=PI/4`) weg und ersetzen die drei Plot-Zeilen für p, ve, ac am Ende des Programms durch

```
p:=plot::Point2d([x1,y1],t1=0..2*PI,Color=RGB::Black,
```

```
PointSize=3*unit::mm):
ve:=plot::Arrow2d([x1,y1],[x2,y2],t1=0..2*PI,Color=RGB::Red):
ac:=plot::Arrow2d([x1,y1],[x3,y3],t1=0..2*PI,Color=RGB::Green):
```

Das Ergebnis ist eine **Animation** der Bewegung. Wir können die Bewegung mit verschiedenen Geschwindigkeiten ablaufen lassen. Der Trick besteht darin, bei den Plot-Anweisungen das Zeitintervall `t1=0..2*PI` einzuführen. (Erst ab Version 3 gibt es bei MuPAD die Animation.)

Wir wollen uns nun auch davon überzeugen, dass sich die Beträge von $\overrightarrow{v}$ und $\overrightarrow{a}$ während der Kreisbewegung nicht ändern.

Programm 2 (Die Länge der Pfeile auf der Kreisbahn)

```
x:=t->2*sin(2*t):
y:=t->2*cos(2*t):
v:=matrix([[x'(t),y'(t)]]):Geschwindigkeitsvektor
a:=matrix([[x''(t),y''(t)]]):Vektor der Beschleunigung
v0:=sqrt(linalg::scalarProduct(v, v)):Betrag von v
a0:=sqrt(linalg::scalarProduct(a, a)):Betrag von a
v1:=plot::Function2d(v0(t),t=0..2*PI, Color=RGB::Red):
a1:=plot::Function2d(a0(t),t=0..2*PI,Color=RGB::Green):
plot(v1,a1)
```

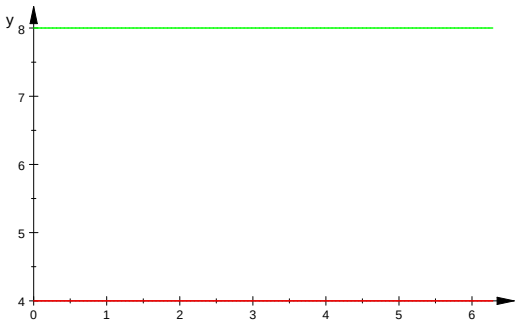

Fig.2.2-8

Die grüne Horizontale durch y = 8 stellt den Beschl.-Betrag dar.

Wir müssen erwarten, dass sich bei der Bewegung auf einer Ellipse ander Ergebnisse ergeben. Der Beschleunigungsvektor kann nicht mehr fortwährend senkrecht auf der Geschwindigkeit stehen. Was wird aus den parallelen Geraden in Fig. 2.2-8? Untersuchen Sie einfach die Angelegenheit mit MuPAD! (Ersetzen Sie z.B. einfach die 2. Zeile durch `y:=t->3*cos(t)`.)

2.2.5 Der schräge Wurf im Vakuum

Es wird kein Problem für uns sein, die Bahn einer schräg abgeschossenen Kanonenkugel zu berechnen. Wenn die Kugel das Rohr verlassen hat, steht sie nur mit der Erde in WW und erfährt die Beschleunigung $\overrightarrow{a} = \overrightarrow{g}$.In der Animation werden wir sehen, dass der Beschleunigungsvektor immer nach unten orientiert

ist. Im nächsten Abschnitt werden wir sehen, dass die "Kugel"-Koordinaten durch folgende Gleichungen gegeben sind

$$x(t) = v_0 t\cos(\alpha);\ y(t) = -\frac{g}{2}t^2 + v_0 t\sin(\alpha) \tag{2.2-12}$$

Wir wählen $v_0 = 70m/s$ und $\alpha = 30°$ (Abschusswinkel). Wir haben Programm 1 nur ein wenig zu ändern, um die Trajektorie der Kugel zusammen mit den Vektoren $\overrightarrow{v}$ und $\overrightarrow{a}$ darzustellen.

Programm 3 (Schiefer Wurf)

```
v0:=70://Projektil-Anfangsgeschw.
g:=9.81:
alpha:=PI/6:
x:=t->v0*t*cos(alpha):
y:=t->-g/2*t^2+v0*t*sin(alpha):
curve:=plot::Curve2d([x(t),y(t)],t= 0..7.13):
pos:=t->(x(t),y(t)):Ortsvektor
vel:=t->(x'(t),y'(t)):Vektor der Geschw.
acel:=t->(x''(t),y''(t)):Beschleunigung
scale:=0.6:Skalierungsfaktor
x1:=pos(t1)[1]:Anfang des Vektors(x'(t),y'(t))
y1:=pos(t1)[2]:fuer t=t1
v1:=vel(t1)[1]:Geschw. des Punktes (x(t),y(t))
v2:=vel(t1)[2]:fuer t=t1
a1:=acel(t1)[1]:Beschleunigung
a2:=acel(t1)[2]:
x2:=x1+v1*scale:Spitze von Vektor (x'(t),y'(t))
y2:=y1+v2*scale:
x3:=x1+a1*3*scale:Der Skalenfaktor wurde vergroessert,
y3:=y1+a2*3*scale: um den Beschl.Vektor deutlicher zu sehen.
p:=plot::Point2d([x1,y1],t1=0..7.13,Color=RGB::Black,
  PointSize=3*unit::mm):
ve:=plot::Arrow2d([x1,y1],[x2,y2],t1=0..7.13,Color=RGB::Red):
ac:=plot::Arrow2d([x1,y1],[x3,y3],t1=0..7.13,Color=RGB::Green):
plot(curve,p,ac,ve)
```

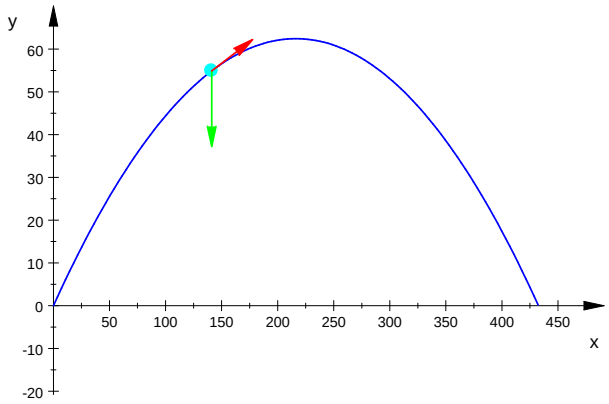

Fig.2.2-9

Die Flugzeit beträgt 7.136s (wurde vorher berechnet). In der Animation sieht man, wie sich der Geschwindigkeitsvektor fortwährend ändert. Im höchsten Punkt ist er horizontal, denn die vertikale Geschw.-Komponente ist im höchsten Punkt Null (die horizontale Komponente ändert sich nicht).

Es ist instruktiv, auch die Geschwindigkeits-Komponenten in der Animation zu verfolgen. Man hat nur noch die folgenden Programmzeilen mit aufzunehmen:

```
vx:=plot::Arrow2d([x1,y1],[x2,y1],t1=0..7.13,
Color=RGB::Black):
vy:=plot::Arrow2d([x1,y1],[x1,y2],t1=0..7.13,
Color=RGB::Blue):
plot(curve,p,ac,ve,vx,vy)
```

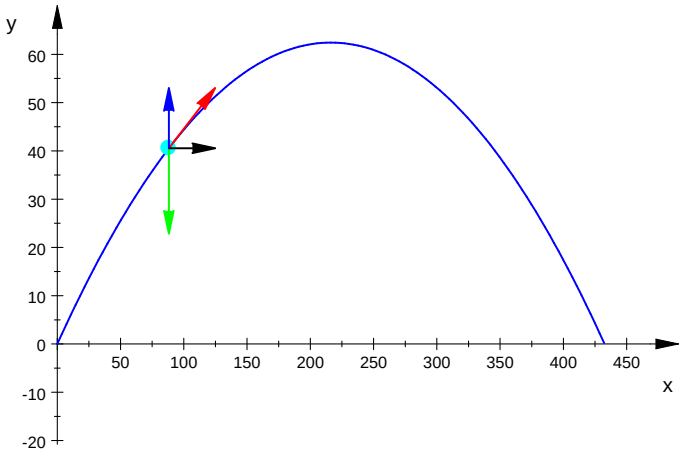

Fig.2.2.10

Um die Flugzeit zu berechnen, setzen wir $y = 0$

```
v0:=70:g:=9.81:
```

```
alpha:=PI/6:
x:=t->v0*t*cos(alpha):
y:=t->-g/2*t^2+v0*t*sin(alpha):
solve(y(t)=0,t)[2]

7.13558//Flugzeit in Sekunden
```

Die Steigzeit ist nat. die Hälfte davon. Dies können wir wir auch aus der Bedingung $v_y = 0$ erhalten

```
vy:=y'(t):
solve(vy=0,t)
{3.56778...}
```

Die maximale Höhe von 62.44 m finden wir mit

```
solve(vy=0,t):
y(solve(vy=0,t)[1])

62.436//Hoehe in m
```

2.2.6 Ausgaben in Tabellenform

Wen wir die Koordinaten des Projektils entlang seiner Bahn in numerischer Form sehen wollen, ist es geschickt, die Programmausgabe in Form einer Tabelle zu verlangen. Der MuPAD-Befehl dafür ist `output::tableForm` .

```
v0:=70:g:=9.81:alpha:=PI/6:
DIGITS :=6:
x:=t->v0*t*cos(alpha):
y:=t->-g*t^2/2+v0*t*sin(alpha):
output::tableForm([[t $ t = 1..5,"t"]]):
output::tableForm([[float(x(t)) $ t = 1..5,"x"]]):
output::tableForm([[float(y(t)) $ t = 1..5,"y"]]):

[1, 2, 3, 5, "t"]
[60.6218, 121.244, 181.865, 242.487, 303.109, "x"]
[30.095, 50.38, 60.855, 61.52, 52.375, "y"]
```

Eine sehr kurze Fassung mit Hilfe einer **Matrix** zeigte mir Wolfgang Lindner:

```
v0:=70:g:=9.81:alpha:=PI/6:
DIGITS :=6:
x:=t->v0*t*cos(alpha):
y:=t->-g*t^2/2+v0*t*sin(alpha):
matrix([[T,X,Y],[t,float(x(t)),float(y(t))] $ t=0..8])
```

```
[T      X         Y]
[0     0.0      0.0]
[1    60.6218 30.095]
[2   121.244   50.38]usw.
```

Auch mit SWP kann man Funktionstabellen erstellen:

1. Zunächst definiert man mit *Compute* > *Definitions* Konstanten und Funktionen.

$a = \pi/6$

$v = 70$ oder $v_0 = 70$

$g = 9.81$

$x(t) = vt\cos(a)$

$y(t) = -gt^2/2 + vt\sin(a)$

2. Man hole sich das Klammersymbol und stelle eine einspaltige Matrix hinein (das wird die t-Spalte werden)

$$x\begin{pmatrix}0\\1\\2\\3\\4\\5\end{pmatrix} = \begin{pmatrix}0\\60.622\\121.24\\181.87\\242.49\\303.11\end{pmatrix}$$

Mitt *Compute* > *Evaluate Numerically* erhalten wir die x-Spalte (am Schluss schreiben wir t und x in die obersten Zellen.

3. Nun wird die t-Spalte markiert und links neben die x-Spalte kopiert. Mit *Matrices* > *concatenate* werden beide Spalten verschmolzen.

4. Die gleiche Prozedur mit der y-Spalte machen.

$$x\begin{pmatrix}0\\1\\2\\3\\4\\5\end{pmatrix} = \begin{pmatrix}0\\1\\2\\3\\4\\5\end{pmatrix}\begin{pmatrix}0\\60.622\\121.24\\181.87\\242.49\\303.11\end{pmatrix}, \text{ concatenate: } \begin{pmatrix}0 & 0\\1 & 60.622\\2 & 121.24\\3 & 181.87\\4 & 242.49\\5 & 303.11\end{pmatrix}$$

$$y\begin{pmatrix}0\\1\\2\\3\\4\\5\end{pmatrix} = \begin{pmatrix}0\\30.095\\50.38\\60.855\\61.52\\52.37\end{pmatrix}, \text{ concatenate: } \begin{pmatrix}t & x & y\\1 & 60.622 & 30.095\\2 & 121.24 & 50.38\\3 & 181.87 & 60.855\\4 & 242.49 & 61.52\\5 & 303.11 & 52.37\end{pmatrix}$$

Man kann die Tabelle aber auch sehr schön gestalten, wenn man ein "**Array**" innerhalb einer "**Prozedur**" benutzt. (Wir werden diese Begriffe später ausführlich besprechen; an dieser Stelle wollen wir nur zeigen, wie man mit ihrer Hilfe wirklich nützliche Tabellen erstellen kann. Im übrigen sind diese Begriffe praktisch mit den entsprechenden Ausdrücken in anderen Computersprachen, z.B. Pascal, identisch.)

Programm 4 (Tabelle mit Prozedur)

```
Projektil:=proc(t0,schritte)
local t,x,y,tabelle;
begin
v0:=70:g:=9.81:alpha:=PI/6:
DIGITS:=6:
t:=t0:
tabelle:=array(0..schritte, 1..3):
for t from t0 to schritte do
 x:=float(v0*t*cos(alpha)):
 y:=float(-g*t^2/2+v0*t*sin(alpha)):
 if t = t0 then
 tabelle[t,1]:=T/s:
 tabelle[t,2]:=X/m:
 tabelle[t,3]:=Y/m:
else
 tabelle[t,1]:=t:
 tabelle[t,2]:=x:
 tabelle[t,3]:=y:
 end_if:
end_for:
return(tabelle)
end_proc:
Projektil(0,8)

[T/s  X/m      Y/m ]
[1  60.6218  30.095]
[2  121.244   50.38]
[3  181.865  60.855] usw.
```

2.2.7 Die Länge einer Trajektorie (Bogenlänge)

Schließlich wollen wir auch noch mit Gleichung 2.2-8 die Länge des Wegstücks berechnen, das die Kugel vom Anfang bis zum Ende der Bahn zurückgelegt hat. Aus Gl. 2.2-8 erhalten wir zunächst

$$ds = \sqrt{x'(t)^2 + y'(t)^2}dt$$

Durch Integration zwischen $t = a$ und $t = b$ erhalten wir dann die gesuchte Streckenlänge s

$$s = \int_a^b \sqrt{x'(t)^2 + y'(t)^2}dt \tag{2.2-13}$$

Wir benutzen die numerische Integration, da das Integral sich nicht exakt auswerten lässt.

```
alpha:=PI/6:
x:=t->v0*t*cos(alpha):
y:=t->-g*t^2/2+v0*t*sin(alpha):
s:=numeric::int(sqrt(x'(t)^2+y'(t)^2),t=0..7.13558)

455.525// Weg s in m
```