1 | Der schiefe Wurf

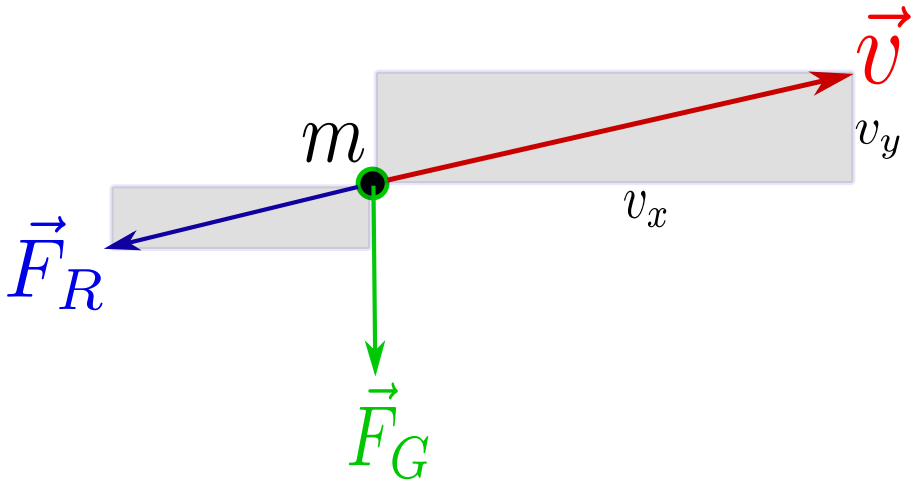


Abb.1 : angreifende Kräfte

Ausgangspunkt ist natürlich die Newton'sche Bewegungsgleichung:

$$\boxed{m_t\,\ddot{\vec{x}} = \sum_i \vec{F}_i} \tag{1.1}$$

$\vec{F}_i$ sind natürlich die angreifenden Kräfte. m_t ist die träge Masse des Körpers.
Wir betrachten hier nur den Luftwiderstand $\vec{F}_R$, der dem Geschwindigkeitsvektor $\vec{v}$ des Körpers entgegengerichtet ist und die Gravitationskraft $\vec{F}_G$. Mit dem Newton'schen Gravitationsgesetz folgt für die Gravitationsbeschleunigung in Richtung Erdzentrum

$$m_t\vec{a}_r(r) = \vec{F}_G = -G\frac{M_E\,m_t}{(R_E+r)^2}\vec{r}_0 \tag{1.2}$$

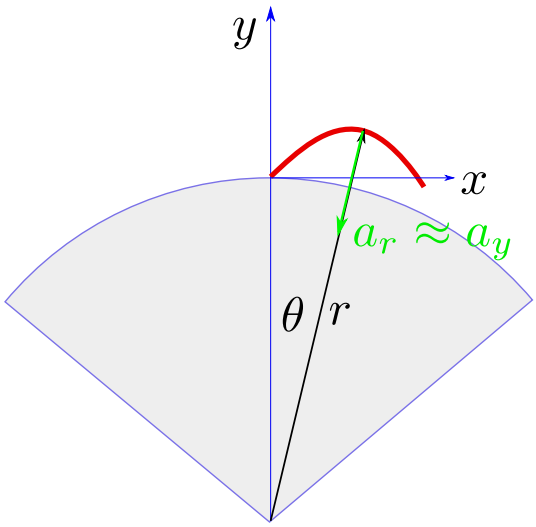


Abb.2 : Die Erde als Scheibe

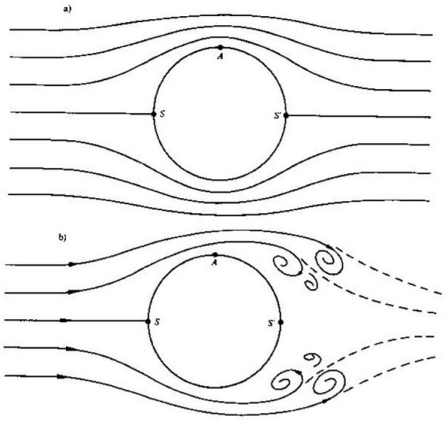


Abb.3 : laminare und turbulente Strömung

Wir machen einige Vereinfachungen:

- Die Wurfweiten w seien extrem klein gegenüber dem Erdradius R_E ($w \ll R_E$) - also wir machen aus der Erde eine Scheibe (eine über Jahrtausende beliebte Näherung). Damit können wir die Wurfbahn statt mit Polarkoordinaten $(r(t), \theta(t))$ mit kartesischen Koordianten (x, y) beschreiben.

- Die Wurfhöhen r (in obiger Formel) seien extrem klein gegenüber dem Erdradius R_E ($r \ll R_E$), damit können r in Formel 1.2 vernachlässigen und es ergibt sich:

$$\begin{cases} a_x = & 0 \\ \\ a_y = & -G\dfrac{M_E}{R_E^2} = -g \approx -9.81 \end{cases} \tag{1.3}$$

  Unsere Formeln sind daher für ballistische Geschoße und Satelliten vollkommen ungeeignet - das wäre wieder eine andere Geschichte!

- Wir betrachten keine Auftriebs- bzw. Abtriebskräfte - ganz gleich ob sie sich durch schnelle Rotation (Spin) (Frisbie-Scheibe) oder Tragflächen (Bumerang), rotierender Tennisball usw. ergeben.

- Als Näherungsstufen betrachten wir

  - $\vec{F}_R = 0$ (Bewegung im Vakuum, z.B. am Mond)
  - $|\vec{F}_R| \propto |\vec{v}|$ (laminare Strömung, langsame Geschwindigkeit z.B. Kugel im Honigglas, Auto in Schrittgeschwindigkeit - Karosserie möglichst ohne Kanten)
  - $|\vec{F}_R| \propto |\vec{v}|^2$ (turbulente Strömung - geworfene Steine, Geschoße, Autos bei "Reisegeschwindigkeit")

1.1 Ohne Luftwiderstand auf der Erdoberfläche

Uns bleibt einfach Gleichung 1.3 zu lösen:

$$\ddot{\vec{x}} = \begin{pmatrix} 0 \\ -g \end{pmatrix} \tag{1.4}$$

1.1.1 Mit Differentialgleichung und *wxMaxima*

Obige gewöhnliche Differentialgleichung 1.4 führt zu einem einfachen Anfangswertproblem:

$$\ddot{\vec{x}} = \begin{pmatrix} 0 \\ -g \end{pmatrix} \qquad \vec{x}(0) = \vec{0} \qquad \vec{v}(0) = \begin{pmatrix} v_{x0} \\ v_{y0} \end{pmatrix} \tag{1.5}$$

Dies lässt sich natürlich leicht "per Hand" integrieren, doch zur Übung machen wir es mit *wxMaxima*:

Keine Meldungen bei Umwandlungen von Brüchen in Dezimalzahlen

```
(%i1) ratprint:false$
```

Beschleunigungsvektor $\vec{a}(t)$ wird festgelegt

```
(%i2) a(t):=[0,-g]$
```

```
(%i3) define(v(t),integrate(a(t),t)+[v_x0,v_y0]);
```

$$\mathrm{v}(t) := [v_{x0}, v_{y0} - gt] \qquad (\%o3)$$

Zeit für Wurfhöhe t_H

```
(%i4) t_H:rhs(linsolve(v(t)[2],t)[1]);
```

$$\frac{v_{y0}}{g} \qquad (\mathrm{t\_H})$$

Bahn in Parameterform

```
(%i5) define(x(t),integrate(v(t),t));
```

$$\mathrm{x}(t) := [t\,v_{x0}, t\,v_{y0} - \frac{g\,t^2}{2}] \qquad (\%o5)$$

Wurfhöhe

```
(%i6) Height:subst(t_H,t,x(t)[2])$
```

von Parameterform in Funktionsdarstellung

```
(%i7) define(y_1(x),subst(x/v_x0,t,x(t)[2]));
```

$$y_1(x) := \frac{v_{y0}\,x}{v_{x0}} - \frac{g\,x^2}{2{v_{x0}}^2} \qquad (\%o7)$$

```
(%i8) eq1:(ratsimp(y_1(x)/x=0,x));
```

$$-\frac{gx - 2v_{x0}v_{y0}}{2{v_{x0}}^2} = 0 \qquad (\mathrm{eq1})$$

Berechnung der Wurfweite

```
(%i9) width:rhs(linsolve(eq1,x)[1])$
```

```
(%i10) display(Height,width)$
```

$$Height = \frac{{v_{y0}}^2}{2g} \qquad width = \frac{2v_{x0}v_{y0}}{g}$$

```
(%i11) t_max:width/v_x0;
```

$$\frac{2v_{y0}}{g} \qquad (\mathrm{t\_max})$$

statt der Wurfparameter v_{x0}, v_{y0} wechseln wir auf $|\vec{v}| = v$ und Steigungswinkel α

(%i12)
```
subst(v*cos(alpha),v_x0,y_1(x))$
```

(%i13)
```
define(y_2(x),trigsimp(subst(v*sin(alpha),v_y0,%)));
```

$$y_2(x) := -\frac{g\,x^2 - 2\cos(\alpha)\,\sin(\alpha)\,v^2 x}{2\cos(\alpha)^2\,v^2} \qquad (\%o13)$$

(%i16)
```
v_x0:6$v_y0:8$g:9.81$
```

(%i17)
```
v:sqrt(v_x0^2+v_y0^2)$
```

(%i18)
```
alpha:atan(v_y0/v_x0)$
```

Jetzt plotten wir alle 3 "Kurven" mit den Linienstärken 8,4 und 1 und den Farben 2,3 und 1

(%i19)
```
plot2d([[parametric,x(t)[1],x(t)[2],[t,0,t\_max],[nticks, 100]],y\_1(x),y\_2(x)],
       [x,0,width],[legend,"parametric","with v\_x,v\_y","with v and {/Symbol a}"],
       [style, [lines, 8,2],[lines, 4,3], [lines, 1,1]] )
```

(%i20)
```
ev(Height);
```
3.261977573904179

(%i21)
```
ev(width);
```
9.785932721712538

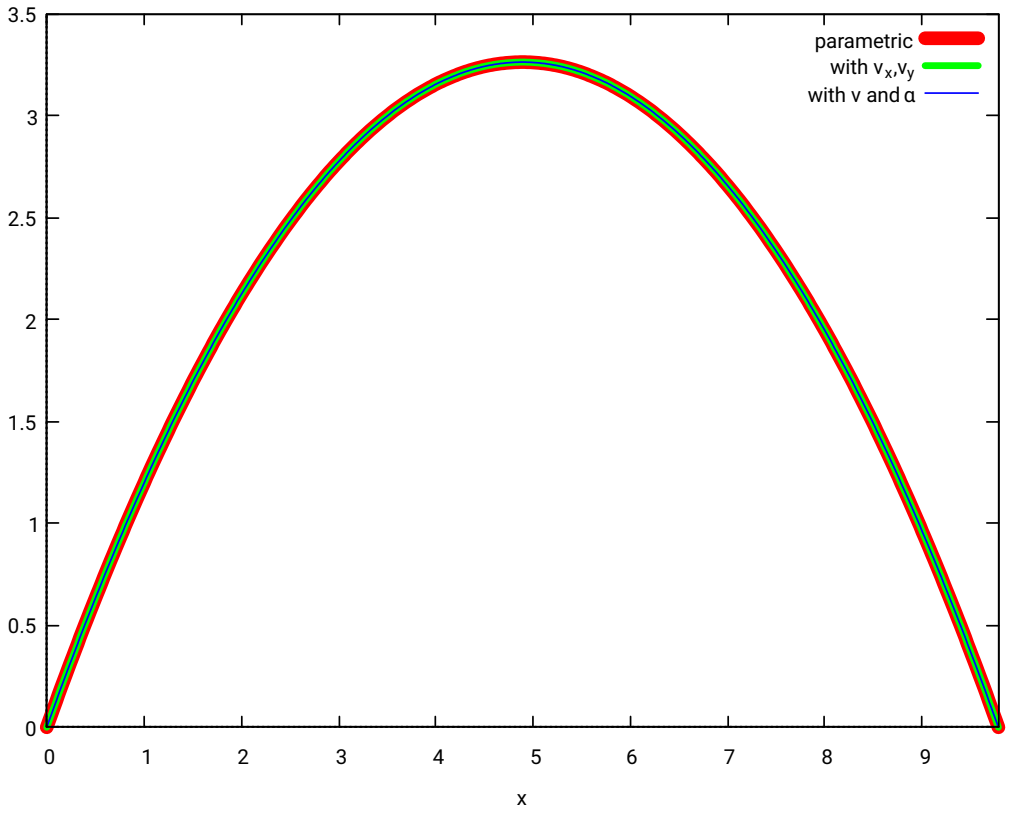


Abb.4 : alle 3 Darstellungen der Bahn

Alle 3 Graphen führen zum selben Ergebnis!

1.1.2 Mit Differenzengleichung und *wxMaxima*

$\Delta t \ll 1$ sei der diskrete Simulationszeitraum. Folgende Iterationsformeln finden Verwendung:

$$\frac{\vec{v}_{i+1}-\vec{v}_i}{\Delta t}=\vec{a}_i \quad \Rightarrow \quad \vec{v}_{i+1}=\vec{v}_i+\vec{a}_i\,\Delta t \tag{1.6}$$

$$\frac{\vec{x}_{i+1}-\vec{x}_i}{\Delta t}=\vec{v}_i \quad \Rightarrow \quad \vec{x}_{i+1}=\vec{x}_i+\vec{v}_i\,\Delta t$$

$$\vec{x}_0=\begin{pmatrix}0\\0\end{pmatrix} \qquad \vec{v}_0=\begin{pmatrix}v_{x0}\\v_{y0}\end{pmatrix}=\begin{pmatrix}6\\8\end{pmatrix} \qquad \vec{a}_0=\begin{pmatrix}0\\-g\end{pmatrix}$$

Natürlich lassen sich obige Vektorfolgen leicht in die entsprechenden Folgen für die Koordinaten zerlegen (so werden wir das in der Tabellenkalkulation machen) - in *wxMaxima* lassen sich allerdings obige rekursive Vektorfolgen durch Array-Funktionen implementieren. Hier eine kleine Demonstration:

Array Functions

```
(%i5)  cc[x,y]:= x/y;
```

$$cc_{x,y} := \frac{x}{y} \qquad (\%o1)$$

```
(%i2)  cc[u,z];
```

$$\frac{u}{z} \qquad (\%o2)$$

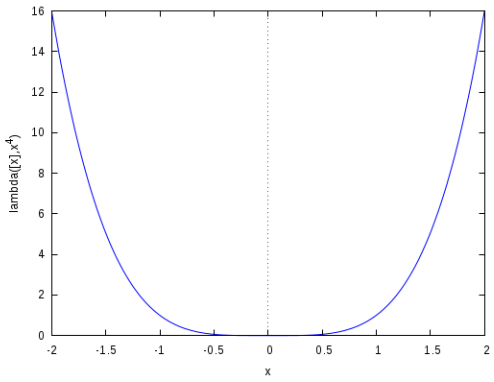
```
(%i4)  u:1$ z:2$
(%i5)  cc[u,z];
```

$$\frac{1}{2} \qquad (\%o5)$$

```
(%i6)  f[n](x):=x^n;
```

$$f_n(x) := x^n \qquad (\%o6)$$

```
(%i7)  wxplot2d(f[4],[x,-2,2]);
```



(%t7)

Recursion

```
(%i8)  v[i]:=v[i-1]+1;
(%i9)  v[0]:0;
(%i10) listarray(v);
(%i11) v[8];
(%i12) listarray(v);
```

$$v_i := v_{i-1}+1 \qquad (\%o8)$$

$$0 \qquad (\%o9)$$

$$[0] \qquad (\%o10)$$

$$8 \qquad (\%o11)$$

$$[0,1,2,3,4,5,6,7,8] \qquad (\%o12)$$

Aber jetzt zurück - der Lösung unseres Problems mit Differenzengleichung:

```
(%i3) Delta_t:0.001$ a:[0,-9.81]$ fpprintprec : 3 $
(%i4) v[i]:=v[i-1] + a * Delta_t;
```

$$v_i := v_{i-1} + a\,\Delta_t \qquad (\%o4)$$

```
(%i5) x[i]:=x[i-1]+v[i-1] * Delta_t;
```

$$x_i := x_{i-1} + v_{i-1}\,\Delta_t \qquad (\%o5)$$

```
(%i6) x[0]:[0,0]$
(%i7) v[0]:[6,8]$
(%i8) trajectory():=block([list:[x[0]]],
         for i:1 while x[i-1][2] >= 0 do
              list: cons(x[i],list),
         list
      )$

(%i9) plotlist:trajectory()$
(%i10) plot2d([discrete, plotlist])$
(%i11) Height:lmax(map(lambda([x],x . [0,1]),plotlist));
```

3.27 (Height)

```
(%i12) width:(plotlist[1][1]+plotlist[2][1])/2;
```

9.79 (width)

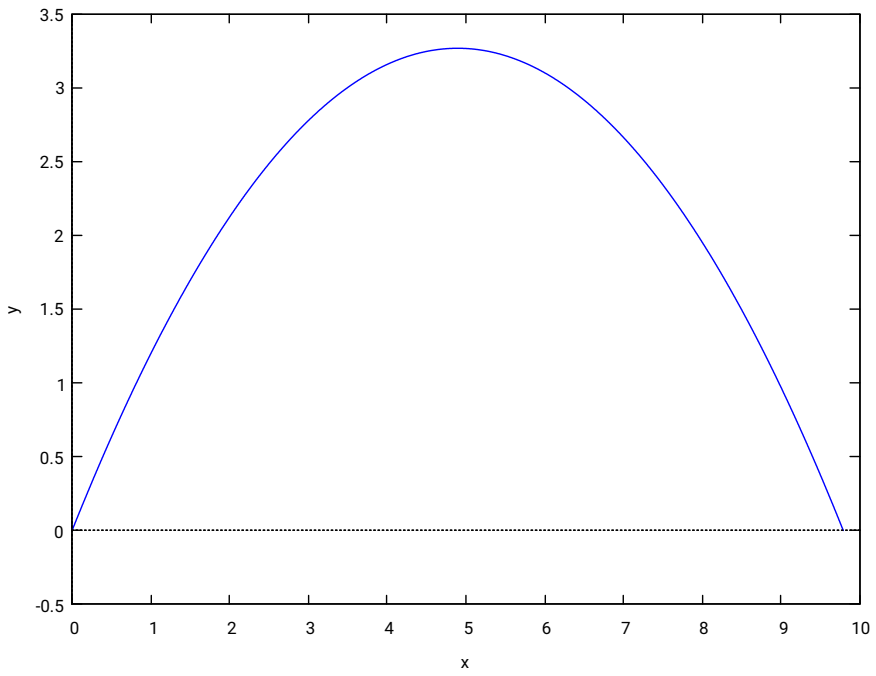


Abb.5 : Jetzt als Punktplot mit Differenzengleichung

1.1.3 Mit LibreOffice-Calc

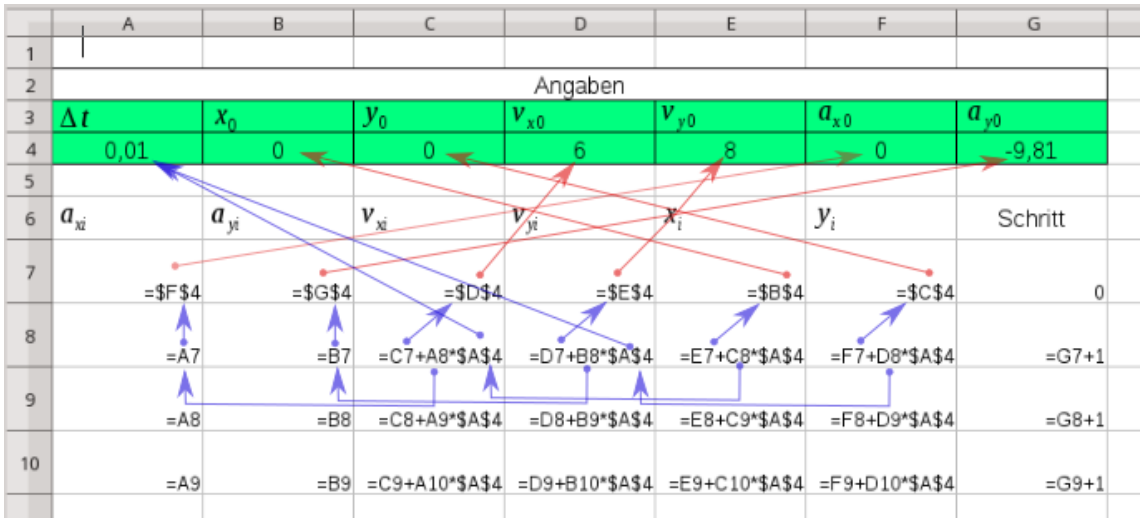


Abb.6 : Tabellenkalkulation

Bis zur 4.-ten Zeile sind die Angaben einzugeben. In Zeile 7 werden die Angaben kopiert. Zeile 8 ist die wichtigste - hier werden die Rekursionsformeln ausgedrückt. Diese Zeile wird nach unten ausgefüllt (beachte die absolute Adressierung für Δt) bis die y_i negativ werden. Anschließend wird für die x-y-Spalte ein x-y-Diagramm erstellt:

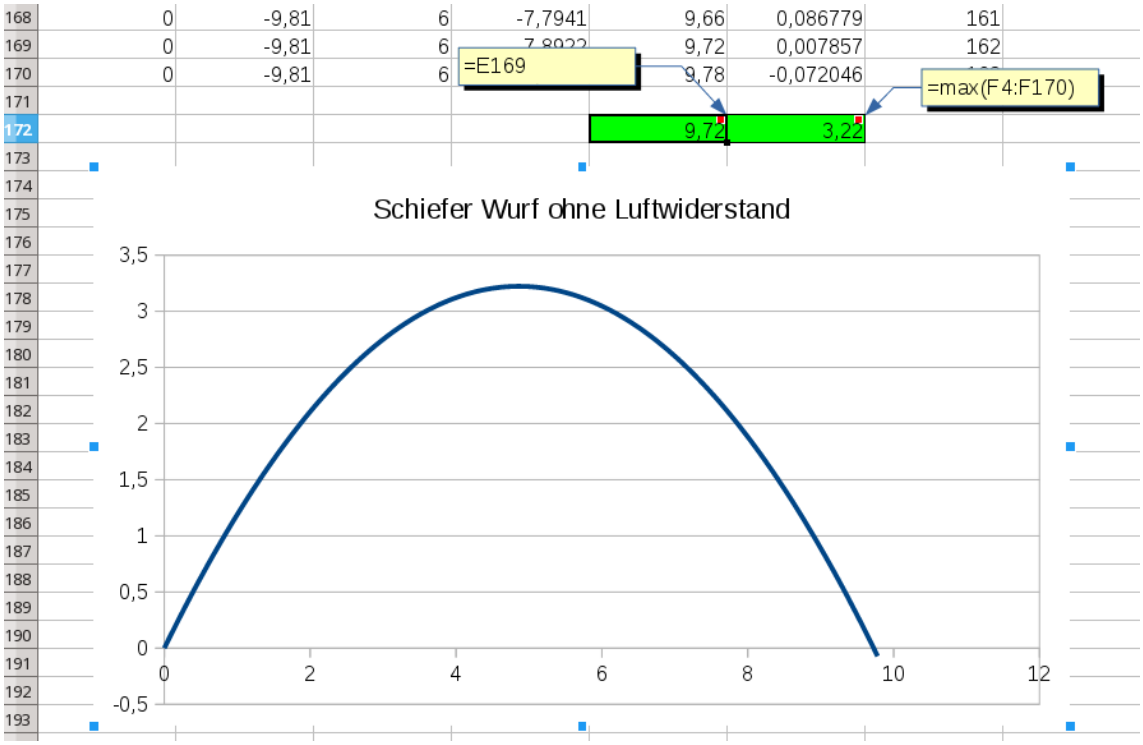


Abb.7 : Weite, Höhe und Bahn in LibreOffice-Calc

1.1.4 Mit Hand und Kettenregel

Wir schreiben 1.4 mit $\vec{v}$ an:

$$\begin{cases} \dot{v}_x = & 0 \\ \dot{v}_y = & -g \end{cases} \tag{1.7}$$

Wir versuchen gleich die Differentialgleichung für die Flugbahn $y(t(x))$ mit der Kettenregel herzuleiten:

$$\frac{dy}{dx} = \underbrace{\frac{dy}{dt}\frac{dt}{dx}}_{v_y/v_x} = \frac{dy}{dt}\frac{1}{\frac{dx}{dt}} \Rightarrow \boxed{\frac{d}{dx} = \frac{1}{v_x}\frac{d}{dt}} \tag{1.8}$$

Wir benutzen die letzte Operatorgleichung

$$\frac{d^2y}{dx^2} = \frac{d}{dx}\frac{dy}{dx} = \frac{1}{v_x}\frac{d}{dt}\frac{v_y}{v_x} = \frac{\dot{v}_y v_x - v_y \dot{v}_x}{v_x^3} \overbrace{=}^{1.7} = \frac{-g}{v_x^2} = \frac{-g}{v_{x0}^2} \tag{1.9}$$

Das letzte Gleichheitszeichen gilt, da v_x eine Konstante ist. Mit den Anfangsbedingungen

$$y(0) = 0 \qquad y'(0) = \tan\alpha = \frac{v_{y0}}{v_{x0}}$$

folgt durch einfaches Integrieren

$$\boxed{y(x) = x\,\tan\alpha - \frac{-g}{2v_{x0}^2}x^2} \tag{1.10}$$

Dies stimmt natürlich mit dem mit *wxMaxima* errechneten $y_1(x)$ überein!

1.2 Luftwiderstand $\propto |\vec{v}|$

Gleichung 1.4 wird jetzt zu

$$\vec{a} = \vec{g} - \frac{1}{\tau}\vec{v} \tag{1.11}$$

1.2.1 Mit Hand und Variablentransformation

Wie in 1.11 schon angedeutet, muss der Proportionalitätsfaktor τ die Dimension einer Zeit haben. Wir führen daher dimensionslose Größen ein (Genaueres im ANHANG 1.3.11:
τ als neue Zeiteinheit, $g\tau$ als neue Geschwindigkeitseinheit und $g\tau^2$ als neue Längeneinheit. Das bedeutet in Gleichung 1.11 wird g und τ durch 1 ersetzt!

$$T = \frac{t}{\tau} \Rightarrow t = T \cdot \tau \qquad (X(T), Y(T)) = \frac{1}{g\tau^2}\,(x(t(T)), y(t(T))) \tag{1.12}$$

damit können wir eine neue Geschwindigkeit festlegen

$$\frac{dX}{dT} = \frac{1}{g\tau^2}\frac{dx}{dt}\frac{dt}{dT} = \frac{1}{g\tau}v(t) \Rightarrow \boxed{V = \frac{1}{g\tau}v} \tag{1.13}$$

Dasselbe machen wir mit den Beschleunigungen indem wir 1.13 differenzieren:

$$\dot{V} = \frac{dV}{dT} = \frac{1}{g\tau}\frac{dv}{dt}\frac{dt}{dT} = \frac{1}{g}\dot{v}(t) \Rightarrow \boxed{\dot{V} = \frac{1}{g}\dot{v}} \tag{1.14}$$

Die x-Koordinate von 1.11 wird dann mit 1.13 und 1.14 zu

$$\dot{v}_x = g\dot{V}_x = -\frac{1}{\tau}g\tau V_x \Rightarrow \dot{V}_x = -V_x$$

und die y-Koordinate zu

$$\dot{v}_y = g\dot{V}_y = -g - \frac{1}{\tau}g\tau V_y \Rightarrow \dot{V}_y = -1 - V_x$$

wir haben also folgendes entkoppeltes Anfangswertproblem zu lösen

$$\begin{cases} \dot{V}_x = & -V_x & Y(0) = 0,\ X(0) = 0 \\ \dot{V}_y = & -1 - V_y & \dfrac{dY}{dX}(0) = \tan\alpha = \dfrac{V_0 \sin\alpha}{V_0 \cos\alpha} = \dfrac{V_{y0}}{V_{x0}} \end{cases} \tag{1.15}$$

Lösung über die Zeitdimension T durch Variablentrennung

x-Koordinate von 1.15

$$\int \frac{dV_x}{V_x} = \int -dT \Rightarrow \ln V_x = -T + C_1 \Rightarrow V_x(T) = e^{-T}e^{C_1} \Rightarrow V_x(T) = V_{x0}\, e^{-T}$$

$$X(T) = -\, V_{x0}\, e^{-T} + C_2 \Rightarrow \boxed{X(T) = V_{x0}\left(1 - e^{-T}\right)} \tag{1.16}$$

y-Koordinate von 1.15

$$\int \frac{dV_y}{1 + V_y} = \int -dT \Rightarrow \ln(1 + V_y) = -T + C_1 \Rightarrow 1 + V_y(T) = e^{-T}e^{C_1} \Rightarrow V_y(T) = (1 + V_{y0})e^{-T} - 1$$

$$Y(T) = -\,(1 + V_{y0})e^{-T} - T + C_2 \Rightarrow \boxed{Y(T) = (1 + V_{y0})\left(1 - e^{-T}\right) - T} \tag{1.17}$$

Um $Y(X)$ zu erhalten, machen wir aus 1.16 T explizit und setzen in 1.17 ein:

$$\boxed{Y(X) = (1 + V_{y0})\,\frac{X}{V_{x0}} + \ln\left(1 - \frac{X}{V_{x0}}\right)} \tag{1.18}$$

Gleichungen 1.16 bis 1.18 können wir mit den Transformationsgleichungen 1.12 und 1.13 in die üblichen physikalischen Einheiten umformen - als Beispiel sei das für $X(T)$ vorgeführt:

$$\frac{1}{g\tau^2}x(t) = \frac{1}{g\tau}v_{x0}\left(1 - e^{-\frac{t}{\tau}}\right)$$

Mit etwas Rechnen kommt man auf folgende Ergebnisse:

$$\begin{cases} x(t) = \tau\, v_{x0}\left(1 - e^{-\frac{t}{\tau}}\right) \\ y(t) = (g\tau^2 + \tau\, v_{y0})\left(1 - e^{-\frac{t}{\tau}}\right) - t\, g\, \tau \\ y(x) = (g\tau + v_{y0})\dfrac{x}{v_{x0}} + \ln\left(1 - \dfrac{x}{\tau v_{x0}}\right) g\tau^2 \end{cases} \tag{1.19}$$

Überprüfen Sie in 1.19 die einzelnen Terme auf ihre physikalische Dimension (Plausibilitätscheck!)
Für $\tau \to \infty$ müsste sich in 1.19 der reibungsfreie Fall ergeben also die Formeln von (%o5) im *wxMaxima*-Programm! Wir zeigen hier eine Möglichkeit:

Wir ersetzen die Exponentialfkt. durch eine Taylorreihe:

$$1 - e^{-\frac{t}{\tau}} = -\sum_{i=1}^{\infty}\left(-\frac{t}{\tau}\right)^i \frac{1}{i!} \quad \text{damit wird } x(t) \text{ von 1.19}$$

$$x(t) = v_{x0}t - \sum_{i=2}^{\infty}\left(-\frac{t}{\tau}\right)^i \frac{1}{i!} \quad \text{die Summe verschwindet für } \tau \to \infty$$

Für $y(t)$ machen spalten wir von der Taylorsumme die ersten 2 Summande an:

$$y(t) = (g\tau^2)\left[\frac{t}{\tau} - \left(\frac{t}{\tau}\right)^2\frac{1}{2} - \sum_{i=3}^{\infty}\left(-\frac{t}{\tau}\right)^i\frac{1}{i!}\right] - \cancel{(t\,g\,\tau)} + v_{y0}t - \sum_{i=2}^{\infty}\left(-\frac{t}{\tau}\right)^i\frac{1}{i!}$$

Nach dem Grenzübergang verschwinden jetzt ebenfalls die Summen und es bleibt

$$y(t) = v_{y0}t - \frac{1}{2}g\,t^2$$

Damit haben die Formeln von 1.19 einen weiteren Plausibilitätscheck bestanden!

Lösung von 1.15 über die Wurfbahn

Wir erhalten folgende Operatorengleichung mit der Kettenregel:

$$\frac{dY}{dX} = \frac{dY}{dT}\frac{dT}{dX} = \frac{dY}{dT}\frac{1}{\frac{dX}{dT}} \Rightarrow \boxed{\frac{d}{dX} = \frac{1}{V_x}\frac{d}{dT}} \tag{1.20}$$

andererseits aus dem ersten Teil obiger Gleichung

$$\frac{dY}{dX} = \frac{dY}{dT}\frac{dT}{dX} = \frac{dY}{dT}\frac{1}{\frac{dX}{dT}} = \frac{V_y}{V_x} \tag{1.21}$$

Nun kombinieren wir 1.20 und 1.21 zu

$$\frac{d^2Y}{dX^2} = \frac{d}{dX}\frac{dY}{dX} = \frac{1}{V_x}\frac{d}{dT}\frac{V_y}{V_x} = \frac{\dot{V}_y V_x - V_y \dot{V}_x}{V_x^3} \overbrace{=}^{1.15} -\frac{1}{V_x^2} \tag{1.22}$$

Wir benötigen also $V_x(X)$ um weiter zu kommen. Diese Funktion bekommen wir mit der Kettenregel und dem Anfangswertproblem1.15:

$$\begin{aligned} \frac{dV_x}{dX} = \frac{dV_x}{dT}\frac{dT}{dX} = \dot{V}_x\frac{1}{V_x} \overbrace{=}^{1.15} -1 \Rightarrow \\ \Rightarrow \quad V_x(X) = C - X = V_{x0} - X \end{aligned} \tag{1.23}$$

Damit wird 1.22 zu

$$\boxed{\frac{d^2Y}{dX^2} = -\frac{1}{(V_{x0} - X)^2}} \tag{1.24}$$

Gleichung 1.24 ist aber leicht durch Integrieren zu lösen:

$$\frac{dY}{dX} = (V_{x0} - X)^{-1} + C_1 \xrightarrow{Y'(0)=\tan\alpha} \frac{dY}{dX} = (V_{x0} - X)^{-1} + \underbrace{\frac{V_{y0}}{V_{x0}}}_{\tan\alpha} - \frac{1}{V_{x0}}$$

und neuerliches integrieren führt zu

$$Y(X) = \ln(V_{x0} - X) + X\tan\alpha - \frac{1}{V_{x0}}X + C_2 \xrightarrow{Y(0)=0} C_2 = -\ln V_{x0}$$

dies führt schließlich zum Endergebnis, welches Gott sei Dank mit 1.18 identisch ist:

$$Y(X) = \ln\left(1 - \frac{X}{V_{x0}}\right) + X\,\tan\alpha - \frac{X}{V_{x0}} \tag{1.25}$$

1.3 Luftwiderstand $\propto |\vec{v}|^2$

$$\vec{F}_R = -\underbrace{\frac{1}{2}\rho_L\, A\, c_W}_{p}\, v^2\, \vec{v}_0 \quad \Rightarrow \quad \vec{F}_R = -p\, v \begin{pmatrix} v_x \\ v_y \end{pmatrix}$$

A ist die "Schattenfläche", bei einer Kugel also die Fläche eines Großkreises
ρ_L ist die Dichte des Mediums (bei uns Luft) - diese schwankt etwas mit der Temperatur und der Seehöhe ($1{,}3\, kg/m^3$ ist aber hier ein brauchbarer Wert)

$$\frac{1}{2}\rho_L\, v^2 \quad \text{wird oft als Staudruck bezeichnet}$$

c_W ist eine dimensionslose Zahl, die von der Form des Körpers abhängt (auch von der Art der Strömung - laminar oder turbulent - sie hängt mit der Reynolds-Zahl Re zusammen. Am besten man bestimmt sie experimentell: für eine Kugel ist 0,4 ein brauchbarer Wert!)
mit $\mu = p/m_t$ und $\vec{F} = m_t\, \vec{a}$ folgt für die Beschleunigung

$$\vec{a} = \dot{\vec{v}} = \begin{pmatrix} 0 \\ -g \end{pmatrix} - \mu\, v \begin{pmatrix} v_x \\ v_y \end{pmatrix} \tag{1.26}$$

Das Produkt aus $\mu \cdot v$ ist eine Größe, die den Einfluss der Reibung beschreibt. Für $\mu = 0$ ergibt sich der reibungsfreie Wurf. Bei einer Billardkugel $m = 170\, g$ und $\varnothing \approx 58\, mm$ und unseren Anfangsgeschwindigkeit ist dieser Einfluss sehr gering! Aber schon der Austausch der Billardkugel durch eine aus Papier ($m \approx 10\, g$) würde diesen Einfluss erheblich vergrößern!
Während die exakte Lösung des Problems sich ziemlich verkompliziert hat, bleibt die Lösung mit Differenzengleichung beinahe gleich:
Man muss zu 1.6 nur Folgendens hinzufügen:

$$\vec{a}_i = -\mu |\vec{v}_i|\, \vec{v}_i + \begin{pmatrix} 0 \\ -g \end{pmatrix}$$

μ muss natürlich wie oben definiert sein!
Damit sich bei den Gleichungen keine zirkulären Definitionen einschleichen muss man etwas aufpassen:

$$\begin{aligned} \vec{a}_0 &= f_1(\vec{v}_0) \\ \vec{v}_1 &= f_2(\vec{v}_0, \vec{a}_0) \\ \vec{x}_1 &= f_3(\vec{x}_0, \vec{v}_0) \qquad \text{wobei } f_2 = f_3 \\ \vec{a}_1 &= f_1(\vec{v}_1) \\ \vec{v}_2 &= f_2(\vec{v}_1, \vec{a}_1) \quad \text{usw.} \end{aligned}$$

Beachte die übereinstimmenden Indices! In *wxMaxima* muss es daher heißen (beachte den Index bei der Beschleunigung!):

```
v[i]:= v[i-1] + a[i-1]*Delta_t
```

1.3.1 Mit *wxMaxima* 1 Bahnkurve

Bei einer Billardkugel ist der Einfluss der Luftreibung gering $\mu \approx 0$

(%i1) mu:0.5*1.3*30^2*10^(-6)*%pi*0.4/0.17,numer;

$$0.004324309887882421 \qquad \text{(mu)}$$

(%i2) a[i]:= -mu*sqrt(v[i] . v[i])*v[i] + g;

$$a_i := (-\mu)\,\sqrt{v_i.v_i}\,v_i + g \qquad \text{(\%o2)}$$

(%i5) Delta_t:0.001$ g:[0,-9.81]$ fpprintprec : 3 $

(%i6) v[i]:=v[i-1] + a[i-1] * Delta_t;

$$v_i := v_{i-1} + a_{i-1}\,\Delta_t \qquad \text{(\%o6)}$$

(%i7) x[i]:=x[i-1]+v[i-1] * Delta_t;

$$x_i := x_{i-1} + v_{i-1}\,\Delta_t \qquad \text{(\%o7)}$$

Interessanterweise wertet *wxMaxima* die Rekursionsformeln passend aus den Angaben selbst aus!

(%i8) x[0]:[0,0]$

(%i9) v[0]:[6,8]$

Solange die y-Werte der Lagevektoren $\vec{x}$ größer/gleich Null sind, geben wir sie in eine Liste

```
(%i10) trajectory():=block([list:[x[0]]],
          for i:1 while x[i-1][2] >= 0 do
               list: cons(x[i],list),
          list
       )$
```

(%i11) plotlist:trajectory()$

Die Liste der (x, y)-Werte wird mit *discrete* ausgedruckt

(%i12) plot2d([discrete, plotlist])$

Wir ermitteln das Maximum der $y-$Werte - es weicht kaum von dem ohne Luftwiderstand ab!

(%i13) height:lmax(map(lambda([x],x . [0,1]),plotlist));

$$3.2 \qquad \text{(height)}$$

Bei der Weite nehmen wir einfach den Mittelwert der "zwei letzten" $x-$Werte

(%i14) width:(plotlist[1][1]+plotlist[2][1])/2;

$$9.45 \qquad \text{(width)}$$

1.3.2 Mit *wxMaxima* mehrere Bahnkurven

Jetzt wollen wir für verschiede Luftwiderstände die Graphen ausgeben, dafür schreiben wir unser Programm etwas um:

(%i4) Delta_t:0.0001$ g:[0,-9.81]$ fpprintprec:3$ stringdisp:true$

(%i7) radius:30*10^(-3)$c_W:0.4$m:0.17$

(%i8) mu(r,c,m):=0.5*1.3*r^2*3.1416*c/m$

Wir weisen den Einfluss der Luftreibung einfach der Variablen k zu!
(%i9) k:-float(mu(radius,c_W,m));

$$-0.00432 \tag{k}$$

Nun reservieren wir etwas Platz für die Fließkomma-Arrays a, v und x
(%i12) array (a, flonum,100000)$array (v, flonum,100000)$array (x, flonum,100000)$;

(%i13) v[0]:[6,8]$

Jetzt die Anfangsbedingungen
(%i14) x[0]:[0,0]$

Berechnungsmethode für die Beschleunigung
(%i15) get_acc(i):=k*sqrt(v[i] . v[i])*v[i] + g$

Berechnungsmethode für Ort und Geschwindigkeit
(%i16) next_xv(i,base, incr):=base[i-1]+incr[i-1]*Delta_t$

$a[0]$ gehört zwar nicht zu den Anfangsbedingungen wird aber für die Berechnung der ..[1]-Werte gebraucht!
(%i17) a[0]:get_acc(0);

$$[-0.259, -10.2] \tag{\%o17}$$

Liste für die "Legende" im Diagramm - wird in *trajectories* gleich mitausgefüllt!
(%i18) myLegend:[]$

Berechnung der Plotliste und der Legende; Parameter: Liste der zu berechnenden k-Werte
(%i19)
```
trajectories(k_list):=block([plist:[],list:[]],
          for j in k_list do block(
          k:-j,
          list:[x[0]],
          for i:1 while second(x[i-1]) >= 0 do block(
               v[i]:next_xv(i,v,a),
               x[i]:next_xv(i,x,v),
               a[i]:get_acc(i),
               list: cons(x[i],list)
       ),
          plist:cons(cons('discrete,[list]),plist),
          myLegend:cons(concat("k = ",j),myLegend)
          ),
          myLegend:cons('legend,myLegend),
          plist
       )$
```

Wir berechnen die Bahnen für $k = 0$(reibungsfrei),$k = 0.01$, $k = 0.1$ und $k = 1$(starke Luftreibung)
(%i20) plotlist:trajectories([0,0.01,0.1,1])$

(%i21) plot2d(plotlist,myLegend)$

Liste von Vektoren; Betrachte nur die Liste mit einer bestimmten -nr- Komponente

(%i22) get_component(nr, l):= map(nr,l)$

(%i23) max_y_ofList(l):=lmax(get_component(second,l))$

(%i24) removed_discrete:get_component(second,plotlist)$

(%i25) heights:map(max_y_ofList,removed_discrete);

$$[0.851, 2.32, 3.12, 3.26] \qquad \text{(heights)}$$

(%i26) width:get_component(first,(get_component(first,removed_discrete)));

$$[1.46, 5.59, 9.04, 9.79] \qquad \text{(width)}$$

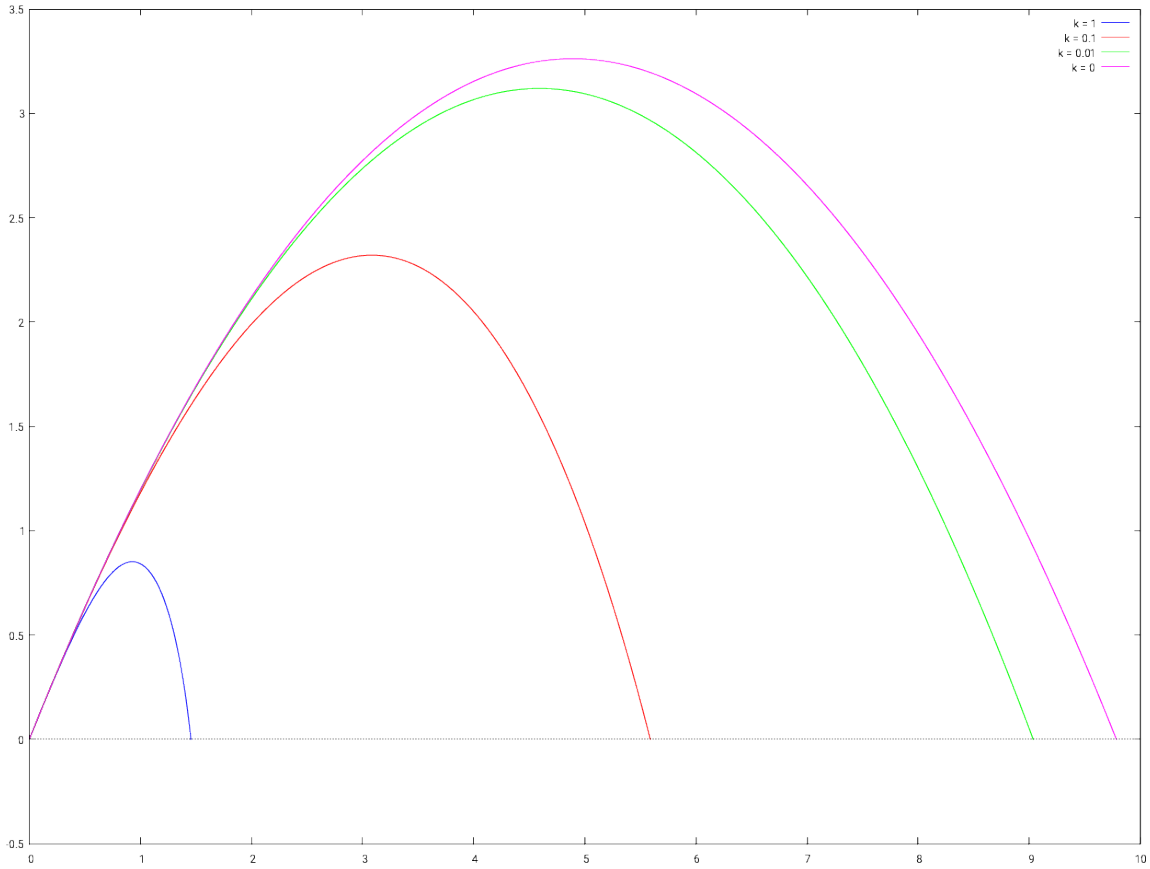


Abb.8 : Vergleich der Flugkurven - genaue Höhen und Weiten siehe in der Berechnung oben

Mit steigendem k geht die Symmetrie der Wurfbahn verloren und Höhen und Weiten werden geringer!
k ist proportional zu A, ρ_L und c_W und umgekehrt proportional zu m !

1.3.3 Mit *Geogebra* mehrere Bahnkurven

Vorweg - es ist mir nicht gelungen ohne etwas Javascript dieses Ziel mit Geogebra zu erreichen. Für mich war es daher auch sehr lehrreich das Zusammenspiel von *Geogebra* mit Javascript zu studieren. Dabei gab es für mich auch einige Überraschungen! Auf dem Arbeitsblatt werden auch untenstehende Differenzengleichungen und Flugbahn ohne Luftreibung angezeigt.
Zur Erinnerung noch einmal die Differenzengleichungen:

$$\overbrace{m\vec{a} = \vec{F}_R + \vec{F}_G}^{Newtons\ Law} = -\underbrace{\frac{1}{2}\rho_L |\vec{v}|^2}_{p_{Stau}} A\, c_W \frac{\vec{v}}{|\vec{v}|} + m\vec{g} \Rightarrow \vec{a} = -\underbrace{\frac{1}{2m}\rho_L A\, c_W}_{k} |\vec{v}|\, \vec{v} + \vec{g}$$

$$(1)\quad \vec{a}_i = -k|\vec{v}_i|\vec{v}_i + \vec{g} \qquad (2)\quad \vec{v}_{i+1} = \vec{v}_i + \Delta t \vec{a}_i \qquad (3)\quad \vec{x}_{i+1} = \vec{x}_i + \Delta t \vec{v}_i$$

(1) in (2) eingesetzt: (1.27)

$$\vec{v}_{i+1} = \vec{f}(\vec{v}_i) = (f_x, f_y)(v_{ix}, v_{iy}) \quad \text{wobei}$$

$$f_x(x,y) := x\left(1 - k\,\Delta t\sqrt{x^2+y^2}\right) \qquad f_y(x,y) := f_x(y,x) - g\,\Delta t$$

Um die Bahnkoordnaten $\vec{x}_i$ koordinatenweise zu berechnen definieren wir die Funktion:
$incrBy(x,y) := x + \Delta t\, y$
Diese 3 Funktionen sind sozusagen die Arbeitspferde. Die einzelnen Ergebnisse unserer Rekursionsschritte speichern wir in den Listen
vxList, vyList, xList, yList, PList
wobei *PList* die Punktliste aus *xList* und *yList* ist. Auf *PList* arbeitet ein *Polygon*-Befehl:
`trajectory = Polygon(PList)`
Die Vorgangsweise zum Zeichnen der Wurfbahn mit den aktuellen Parametern (k als Schieberegler implementiert) ist:

- Button erstellen → "Create Trajectory"
- Im "On Mouseclick" dieses Buttons folgende Tätigkeiten durchführen:
  - Formeln wegblenden und Flugbahn ohne Luftreibung einblenden
  - Die Listen *vxList, vyList, xList, yList, PList* leeren
  - Diese Listen mit den Startwerten füllen
  - Mit den Funktionen f_x, f_y und *incrBy* die Listen *vxList, vyList, xList, yList* berechnen bis ein Wert der *yList* negativ wird
  - Bei obiger Berechnung auch gleich die Punktliste *PList* erstellen
  - Gezeichnet wird dann automatisch wegen *trajectory = Polygon(PList)*

In Javascript (nicht vergessen umschalten von Geogebra-Script - notwendig, da in Geogebra-Script als einzige Schleife nur der *Folge*-Befehl existiert und wir brauchen eine Schleife mit Abbruchbedingung) schaut das dann so aus:

```
// hide explaination text
setVisibility(false);
// set vxList, vyList, xList, yList and PList to start-values
initLists();

//now do the recursion as long as body's height is positive - save the points in
    PList
// the content of PList is drawn by 'trajectory'
var i=1;
do {
   i = i+1;
   ggbApplet.evalCommand("SetValue(vxList," +i+", f_x(vxList("+(i-1)+"),vyList("
    +(i-1)+")) )");
  ggbApplet.evalCommand("SetValue[vyList," +i+", f_y(vxList("+(i-1)+"),vyList("
    +(i-1)+")) ]");
  ggbApplet.evalCommand("SetValue(xList," +i+", incrBy(xList("+(i-1)+"),vxList("
    +(i-1)+")) )");
  ggbApplet.evalCommand("SetValue(yList," +i+", incrBy(yList("+(i-1)+"),vyList("
    +(i-1)+")) )");
  currentWidth = ggbApplet.getListValue("xList", i);
  currentHeight = ggbApplet.getListValue("yList", i);
  ggbApplet.evalCommand("SetValue(PList," +i+",("+currentWidth+","+currentHeight
    +"))");
} while (currentHeight >= 0);
```

Wie man sieht, sind die Funktionen *setVisibility(boolean v)* und *initLists()* in den Reiter "global Javascript" ausgelagert, um den workflow klarer zu machen. Die Hauptarbeit leistet die *do - while*-Schleife.

Nicht vergessen: wir starten beim zweiten Listenelement, da im ersten Listenelement die Initialisierungen stehen!

Schauen wir uns die "global Javascript" näher an:
Da ist die *function initLists()* welche die Listen leert und anschl. initialisiert!

```
function initLists(){
  ggbApplet.evalCommand("vxList = {}");
  ggbApplet.evalCommand("vyList = {}");
  ggbApplet.evalCommand("xList = {}");
  ggbApplet.evalCommand("yList = {}");
  ggbApplet.evalCommand("PList = {}");

  ggbApplet.evalCommand("SetValue(vxList,1,v_{x0})");
  ggbApplet.evalCommand("SetValue[vyList,1,v_{y0}]");
  ggbApplet.evalCommand("SetValue(xList,1,x_{0})");
  ggbApplet.evalCommand("SetValue[yList,1,y_{0}]");
  ggbApplet.evalCommand("SetValue(PList,1,(x_{0},y_{0}))");
}
```

Das "Leeren" der Liste kann nicht weggelassen werden, da sonst nur das erste Listenelement verändert wird!

```
function toggleVisibility(){
  setVisibility(! ggbApplet.getVisible("Text1"));
}

function setVisibility(v){
  ggbApplet.setVisible("Text1", v);
  ggbApplet.setVisible("Text2", v);
  ggbApplet.setVisible("Text3", v);
  ggbApplet.setVisible("Text4", v);
  ggbApplet.setVisible("s", ! v);
}
```

Außerdem ist hier auch die *function setVisibility(boolean v)* implementiert - die den Text aus- bzw. Parameterwurfkurve (ohne Luftwiderstand) einblendet. Diese Funktion wird auch für die *function toggleVisibility()* verwendet (die Mausklick-Routine für einen Button)
So jetzt haben wir alles beieinander, um jeweils 1(!) Bahnkurve zu zeichnen (für das jeweilig mit dem Schieberegler eingestellte k). Wir wollen aber eine Kurve - wenn sie uns gefällt - weiter anzeigen (speichern) und den k-Wert dazuschreiben. Dazu benutzen wir folgende Dinge:

- `SavedPLists={}` eine Liste, in der wir die aktuelle *PList* einfügen (eine Liste von Listen)
- `SavedPoly=Folge(Polygonzug(SavedPLists(i)), i, 1, Länge(SavedPLists))` zeichnet die *SavedPLists*
- `MyLabels= {{"", (0, 0)}}` Liste, die Text und Punkte enthält - was wird wo geschrieben; ist mit einem Leerstring initialisiert, da sonst der nächste Befehl nich funktioniert!
- `display=Folge(Text(Element(MyLabels, i, 1), Element(MyLabels, i, 2)), i, 1, Länge(MyLabels))` schreibt die Einträge von *MyLabels* auf den Schirm (arbeitet NICHT auf einer leeren Liste!
- `beforeLastPos=Element(PList, Länge(PList) - 1)`der vorletzte Punkt der aktuellen Wurfbahn(zu Beginn undefiniert!) - das wird die Position, wo wir unseren k-Wert hinschreiben!
- `savedLength=Länge(SavedPLists)`die Anzahl der gespeicherten Kurven - in Javascript benötigt

Jetzt brauchen wir nur etwas Javascript-Kitt, um diese Objekte zu verbinden: *SavedPLists* und *MyLabels* füllen - den Rest erledigt Geogebra automatisch!

So jetzt erzeugen wir einen Button - ich habe ihn “Retain Trajectory” genannt - mit folgender *onMouseClick* Methode:

```
// get the length of list 'SavedPLists' and increment list-pointer
var savedLength = ggbApplet.getValue("savedLength");
savedLength++;

// PList is inserted by value NOT as pointer
ggbApplet.evalCommand("SetValue[SavedPLists,"+savedLength+",PList]");

// we construct the current label - but we cannot use the point 'beforeLastPos' directly,
// because it's a pointer NOT a value! So we have to construct it from it's coordinates!
var frictionCoeff=ggbApplet.getValue("k");
var myLabel = "k = " + frictionCoeff;
var xPos = ggbApplet.getXcoord("beforeLastPos");
var yPos = ggbApplet.getYcoord("beforeLastPos");
var pointPos = "("+xPos+","+yPos+")";

// note the double quotes - they are important here!
ggbApplet.evalCommand('SetValue[MyLabels,' +savedLength+ ', { " ' +myLabel+ ' "' + pointPos + '}]');
```

So - geschafft! Das Ergebnis können Sie bei https://www.geogebra.org/m/MsVRs4pn runterladen!

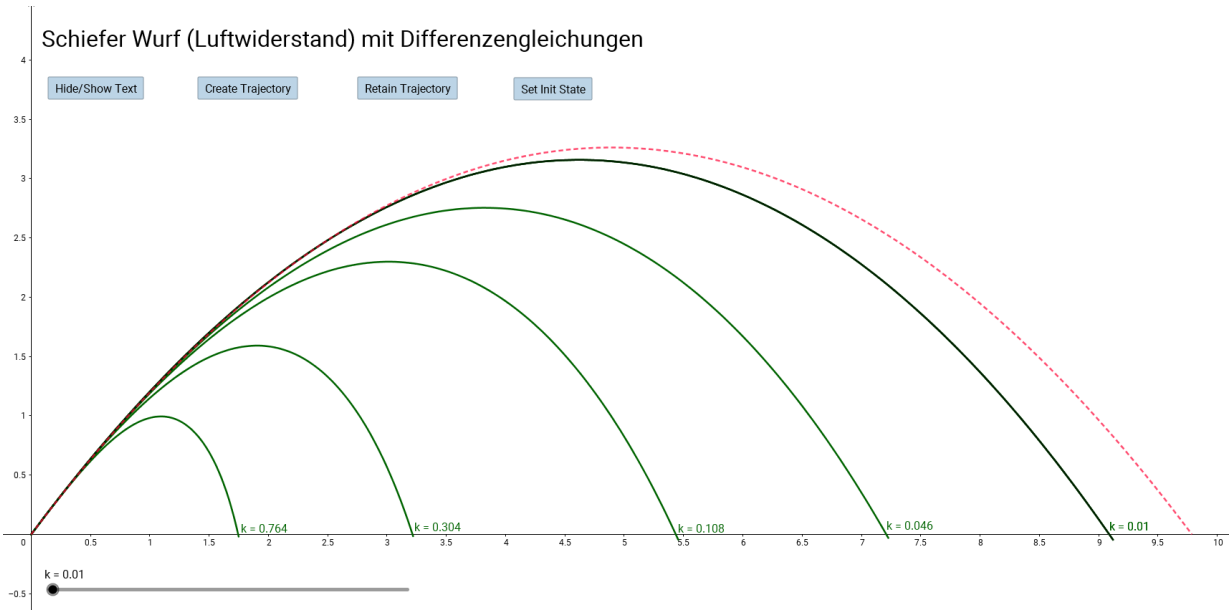


Abb.9 : Vergleich der Flugkurven in Geogebra

Den Aufwand mit *Geogebra* und *wxMaxima* vergleichend, scheint mir der in *Geogebra* höher zu sein - um eine Programmierung konnte ich mich in keinem von beiden drücken! Dies gelingt natürlich mit einer Tabellenkalkulation - aber der Preis dafür ist hoch. Hier die Übersicht über die vielen unterschiedlichen Zellen zu behalten, setzt eine gute Organsiation voraus!

1.3.4 Mit Runge-Kutta Verfahren

Wir können 1.27 (1) als Differentialgleichungssystem anschreiben:

$$\dot{\vec{v}} = -k|\vec{v}|\,\vec{v} + \vec{g} \tag{1.28}$$

aufgeschlüsselt in Komponenten ergibt das

$$\begin{aligned}\dot{v}_x &= -k\sqrt{v_x^2 + v_y^2}\;\; v_x \\ \dot{v}_y &= -k\sqrt{v_x^2 + v_y^2}\;\; v_y - 9.81\end{aligned}$$

Für obiges System können wir das in *wxMaxima* eingebaute Runge-Kutta-Verfahren benützen. Die Syntax ist einfach: `rk([eq1,eq2],[v1,v2],[x0,y0],[t,t0,tf,Delta_t])`
Übergeben werden die Liste Differentialgleichungen rechte Seite, Liste der abhängigen Variablen, Liste der Anfangswerte und die Liste der unabhängigen Variablen mit ihrem Anfangswert, Endwert und der Schrittweite. Als Output bekommt man eine Liste der Form:

```
[ [t0,v1(t0),v2(t0)], ... [ti,v1(ti),v2(ti)] ... [tf,v1(tf),v2(tf)] ]
```

Einige Probleme dieses Verfahrens liegen auf der Hand:

- Wie groß soll man t_f (t-final) wählen bei bestimmter Auftreffhöhe? (Bleibt nur Probieren)
- Wie groß soll die Schrittweite Δt für eine "genügende"(?) Genauigkeit gewählt werden? Strategie: $\Delta t_0 = 10$ und $\Delta t_{i+1} = \Delta t_i \cdot 2$ - das Verfahren für jedes Δt_i durchführen, die Wurhöhen h_i und Wurfweiten w_i bilden jeweils eine Folge. Abbruch wenn für beide folgen gilt
$$|a_{i+1} - a_i| < \varepsilon \qquad \text{Cauchy-Kriterium}$$
- Wie integriert man die Geschwindigkeiten $\vec{v}_i$? Von Unter- bzw. Obersumme, Trapez, Simpson,Romberg, Gauß und Spline Verfahren ist hier alles möglich - auch das wirkt sich natürlich auf obige Folgen aus!
Wir verwenden hier Unter- bzw. Obersumme - was ja gleichdedeutend ist mit unserer Differenzengleichung (bzw. Rekursionsformel):

$$\begin{aligned}\vec{x}_{i+1} &= \vec{x}_i + \vec{v}_i\,\Delta t \quad \Rightarrow \\ \vec{x}_1 &= \vec{x}_0 + \vec{v}_0\,\Delta t \qquad \vec{x}_2 = \vec{x}_1 + \vec{v}_1\,\Delta t = \vec{x}_0 + \vec{v}_0\,\Delta t + \vec{v}_1\,\Delta t \quad \text{also} \\ \vec{x}_{n+1} &= \vec{x}_0 + \Delta t \sum_{i=0}^{n} \vec{v}_i \quad \leftrightarrow \quad \dot{\vec{x}}(t) = \vec{v}(t) \Leftrightarrow \vec{x}(t) = \int_0^t \vec{v}(\tau)\,d\tau + \vec{x}_0\end{aligned} \tag{1.29}$$

Wenn man in der letzten Zeile linken und rechten Term vergleicht, erkennt man, dass unsere Summe eine Näherung des Integrals über $\vec{v}$ darstellt!
Wir implementieren das jetzt in *wxMaxima*:

Zuerst die Angaben, wobei Δt erst beim letzten Durchlauf so klein gewählt wird!

(%i6) Delta_t:0.001$ k:0.1$v_x0:6$v_y0:8$x[0]:[0,0]$g:9.81$targetHeight:0$

(%i8) eq1:-k*sqrt(v_x^2+v_y^2)*v_x; $$-0.1v_x\sqrt{{v_y}^2+{v_x}^2} \quad \text{(eq1)}$$

(%i9) eq2:-k*sqrt(v_x^2+v_y^2)*v_y-9.81; $$-0.1v_y\sqrt{{v_y}^2+{v_x}^2}-9.81 \quad \text{(eq2)}$$

die Liste der Stützpunkte - 3 dimensional! siehe Text!

(%i10) sol: rk([eq1,eq2],[v_x,v_y],[v_x0,v_y0],[t,0,1.4,Delta_t])$

die rekursive Methode und alternativ die Näherungssumme nach 1.29; *rest([1,2,3]→ [2,3]*

(%i11) x[i]:=x[i-1] + rest(sol[i])*Delta_t$

(%i12) y[i]:=x[0]+rest(lreduce("+",firstn(sol,i)))*Delta_t$

beide liefern natürlich das gleiche Ergebnis - hier ein Beispiel

(%i13) x[15]; $[0.08937836805567448, 0.1181460047265376]$ (%o13)

(%i14) y[15]; $[0.08937836805567449, 0.1181460047265376]$ (%o14)

(%i15) points:makelist(x[i],i,0,length(sol))$

(%i16) onlyPosY:sublist(points,lambda([x], (second(x)-targetHeight) >= 0))$

(%i17) width:lmax(map(lambda([x],first(x)), onlyPosY)); 5.5922 (width)

(%i18) flightTime:first(last(firstn(sol,length(onlyPosY)))); 1.372 (flightTime)

(%i19) height:lmax(map(lambda([x],second(x)),points)); 2.3241 (height)

(%i20) plot2d([discrete,onlyPosY])$

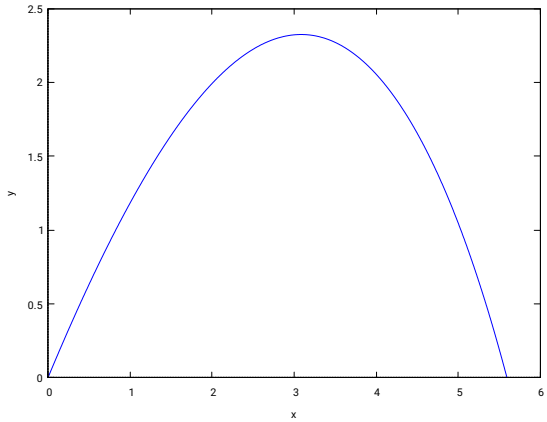


Abb.10 : Bahn mit Runge-Kutta und Differenzengleichung

1.3.5 Ermittlung der Bahnkurvengleichung (dimensionslos)

Wir schreiben 1.26 etwas anders an:

$$\vec{a} = \vec{g} - \frac{1}{\ell} v \vec{v} \qquad v := |\vec{v}| \tag{1.30}$$

Nach Division durch g (dimensionslos) ergibt sich folgendes System

$$\begin{cases} \dot{V}_x = & -V\,V_x & (V_x, V_y) := \dfrac{1}{\sqrt{g\ell}}\,(v_x, v_y) & V := |\vec{V}| \\ \dot{V}_y = & -1 - V\,V_y & (X, Y) := \dfrac{1}{\ell}(x, y) & T := t\sqrt{\dfrac{g}{\ell}} \end{cases} \tag{1.31}$$

(Durch Substitution zeigt man leicht $V_x \cdot T = X \Leftrightarrow v_x \cdot t = x$)
Jetzt gehen wir genau so vor wie bei 1.20 und 1.21 und erhalten

$$\begin{aligned} \frac{d^2Y}{dX^2} = \frac{d}{dX}\frac{dY}{dX} = \frac{1}{V_x}\frac{d}{dT}\frac{V_y}{V_x} == \frac{\dot{V}_y V_x - V_y \dot{V}_x}{V_x^3} = \\ = \frac{(-1 - V\,V_y)V_x - V_y(V\,V_x)}{V_x^3} = -\frac{1}{V_x^2} \end{aligned} \tag{1.32}$$

Leider ist jetzt V_x' keine Konstante mehr, sodass uns der Weg des linearen Falles versperrt bleibt:

$$V_x' = \frac{dV_x}{dX} = \frac{dV_x}{dT}\frac{1}{\frac{dX}{dT}} = \dot{V}_x \frac{1}{V_x} = -V(X) = -V_x\sqrt{1 + \left(\frac{V_y}{V_x}\right)^2} = -V_x\sqrt{1 + \left(\frac{dY}{dX}\right)^2}$$

Ein anderer Weg ergibt sich durch ableiten nach X von 1.32:

$$\frac{d^3Y}{dX^3} = \frac{2}{V_x^3}\frac{dV_x}{dX} = -\frac{2}{V_x^2}\sqrt{1 + \left(\frac{dY}{dX}\right)^2}$$

Wir benützen nocheinmal 1.32 un erhalten (zusammen mit den Anfangsbedingungen:

$$\frac{d^3Y}{dX^3} = 2\frac{d^2Y}{dX^2}\sqrt{1 + \left(\frac{dY}{dX}\right)^2} \qquad Y(0) = 0,\, Y'(0) = \tan\alpha,\, Y''(0) = -\frac{1}{V_{x0}^2} \tag{1.33}$$

Leider hat auch 1.33 keine geschlossene Lösungformel, aber es lässt sich leicht eine Näherung angeben, wenn $\dfrac{dY}{dX} \ll 1$ gilt - die sog. Kurzzeit("short-time" st) Lösung (Abschusswinkel annähernd Null und Flugzeit zu kurz, dass sich viel ändert!) - dann gilt

$$\frac{d^3Y_{st}}{dX^3} \approx 2\frac{d^2Y_{st}}{dX^2} \quad \textbf{short time approximation} \tag{1.34}$$

1.3.6 Lösung der short-time Näherung im Ortsraum

Wir lösen obige short-time-approximation mit *wxMaxima* und überprüfen sie mit Runge-Kutta.
Vorher aber einige erklärende Bemerkungen:

In *wxMaxima* werden gewöhnliche Differentialgleichungen erster und zweiter Ordnung (ordninary differential equations of first or second order) mit
`ode2(<eq>,<dvar>,<ivar>)` gelöst, dabei steht `<dvar>` bzw. `<ivar>` für die abhängige (dependent) bzw. unabhängige(independent) Variable.

Die Differentiation bei der Gleichung (`<eq>`) darf nicht ausgewertet werden (Apostroph), d. h. *wxMaxima* setzt hier nur das Operatorensymbol (noun form of operator) ohne es auszuwerten - da ja keine Funktionen eingesetzt werden (!) sondern nur Variable, würde die Differentiation einfach Null ergeben!

Um von der allgemeinen Lösung die *ode2* (die Integrationskonstante wird mit `%c` bezeichnet) liefert, auf die Lösung des Anfangswertproblems zu kommen, bedient man sich der Funktion `ic1(<gsol>,<ivar>=<ival>,<dvar>=<dval>)` – sie erzeugt aus der allgemeinen Lösung (general solution) die spezielle Lösung! Der "1"-er bedeutet, das es sich um eine Differentialgleichung 1.-er Ordnung handelt. Hier jetzt das Programm:

Bei Dezimalzahlen werden nur 3 Ziffern ausgegeben!

```
(%i29) fpprintprec:3$
```

$z_1 := Y''$ - wir holen uns Schritt für Schritt die nächste Funktion!

```
(%i2) diffEq1:'diff(z_1,X)=2*z_1;
```
$$\frac{d}{dX} z_1 = 2 z_1 \qquad \text{(diffEq1)}$$

```
(%i3) gSol1:ode2(diffEq1,z_1,X);
```
$$z_1 = \%c\, \%e^{2X} \qquad \text{(gSol1)}$$

```
(%i4) spSol1:ic1(gSol1,X=0,z_1=-1/V_X0^2);
```
$$z_1 = -\frac{\%e^{2X}}{{V_{X0}}^2} \qquad \text{(spSol1)}$$

$z_2 := Y'$

```
(%i5) diffEq2:'diff(z_2,X)=rhs(spSol1);
```
$$\frac{d}{dX} z_2 = -\frac{\%e^{2X}}{{V_{X0}}^2} \qquad \text{(diffEq2)}$$

```
(%i6) gSol2:ode2(diffEq2,z_2,X);
```
$$z_2 = \%c - \frac{\%e^{2X}}{2{V_{X0}}^2} \qquad \text{(gSol2)}$$

```
(%i7) spSol2:ic1(gSol2,X=0,z_2=tan(%alpha));
```
$$z_2 = -\frac{\%e^{2X} - 2\tan(\alpha)\, V_{X0}^2 - 1}{2V_{X0}^2} \qquad \text{(spSol2)}$$

$z_3 := Y$

```
(%i8) diffEq3:'diff(z_3,X)=rhs(spSol2);
```
$$\frac{d}{dX} z_3 = -\frac{\%e^{2X} - 2\tan(\alpha)\, V_{X0}^2 - 1}{2V_{X0}^2} \qquad \text{(diffEq3)}$$

(%i9) gSol3:ode2(diffEq3,z_3,X);

$$z_3 = \%c - \frac{\frac{\%e^{2X}}{2} - 2\tan(\alpha)\,{V_{X0}}^2 X - X}{2{V_{X0}}^2} \qquad \text{(gSol3)}$$

Hier haben wir jetzt die spezielle Lösung für unsere Funktion $Y(X)$

(%i10) spSol3:ic1(gSol3,X=0,z_3=0);

$$z_3 = -\frac{\%e^{2X} + \left(-4\tan(\alpha)\,{V_{X0}}^2 - 2\right)X - 1}{4{V_{X0}}^2} \qquad \text{(spSol3)}$$

Wir schreiben den Term um

(%i11) solEqXY:Y=expand(rhs(spSol3));

$$\boxed{Y = -\frac{\%e^{2X}}{4{V_{X0}}^2} + \frac{X}{2{V_{X0}}^2} + \tan(\alpha)X + \frac{1}{4{V_{X0}}^2}} \qquad \text{(solEqXY)}$$

Substituieren die "alten" Variablen

(%i12) solEqxy:subst(v_x0/sqrt(g*l),V_X0,(subst(y/l,Y,subst(x/l,X,solEqXY))));

$$\frac{y}{l} = -\frac{gl\,\%e^{\frac{2x}{l}}}{4{v_{x0}}^2} + \frac{gx}{2{v_{x0}}^2} + \frac{\tan(\alpha)x}{l} + \frac{gl}{4{v_{x0}}^2} \qquad \text{(solEqxy)}$$

Machen y explizit

(%i13) expand(solEqxy*l);

$$y = -\frac{g\,l^2\,\%e^{\frac{2x}{l}}}{4{v_{x0}}^2} + \frac{glx}{2{v_{x0}}^2} + \tan(\alpha)x + \frac{g\,l^2}{4{v_{x0}}^2} \qquad \text{(\%o13)}$$

Definieren y als Funktion

(%i14) define(y(x),rhs(%));

$$\mathrm{y}(x) := -\frac{g\,l^2\,\%e^{\frac{2x}{l}}}{4{v_{x0}}^2} + \frac{glx}{2{v_{x0}}^2} + \tan(\alpha)x + \frac{g\,l^2}{4{v_{x0}}^2} \qquad \text{(\%o14)}$$

Löschen die Variablen z_i - wir brauchen sie später für RUNGE-KUTTA

(%i15) kill(z_1,z_2,z_3)$

Abschusswinkel in Grad

(%i16) alphaDeg:5$

Winkel in Bogenmaß, Billardkugel: $V_{X0} = 1 \leftrightarrow v_{x0} \approx 50\,m/s$, Simulationsweite $0.4 \leftrightarrow\approx 900\,m$

(%i19) %alpha:alphaDeg*%pi/180$V_X0:1$simWidth:0.4$

Jetzt Runge-Kutta für Systeme (nur rechte Seite): $z_1' = z_2$; $z_2' = z_3$; $z_3' = 2z_3 \ldots$ mit Variablenliste und Anfangsbedingungen

(%i20) sol5:rk([z_2,z_3, 2*z_3*sqrt(1+z_2^2)],[z_1,z_2,z_3],[0,tan(%alpha),-1/V_X0^2],[x,0,simWidth,0.01])$

Zum Zeichnen brauchen wir nur X und $z_1 = Y$ - das sind die ersten 2 Listenelemente!

(%i21) plotPoints5:map(lambda([x],firstn(x,2)),sol5)$

Zum Vergleich unsere Näherungsfunktion!

(%i22) define(Y_5(X),rhs(ev(solEqXY)));

$$Y_5(X) := -\frac{\%e^{2X}}{4} + \tan\left(\frac{\pi}{36}\right)X + \frac{X}{2} + \frac{1}{4} \qquad \text{(\%o22)}$$

Jetzt dasselbe nocheinmal mit 15 Grad Abschusswinkel:

```
(%i24) alphaDeg:15$%alpha:alphaDeg*%pi/180$

(%i25) sol15:rk([z_2,z_3, 2*z_3*sqrt(1+z_2^2)],[z_1,z_2,z_3],[0,tan(%alpha),-1/V_X0^2],[x,0,simWidth,0.01])$

(%i26) plotPoints15:map(lambda([x],firstn(x,2)),sol15)$

->      define(Y_15(X),rhs(ev(solEqXY)));

(%i28) plot2d([[discrete,plotPoints5],[discrete,plotPoints15],Y\_5(X),Y\_15(X)],[X,0,simWidth],
       [legend, "   rk-5Grad","   rk-15Grad","   Näherung-5", "   Näherung-15"],
       [gnuplot_preamble, "set key bottom left; set xtics font \", 20\";
           set ytics font \", 20\"; set key font \", 20\" "]);
```

Wie man bei Abb. 11 sieht: Beim 5 Grad totale Überdeckung, bei 15 Grad bei genauem Hinsehen (Billardkugel bei ca. 50 m/s)

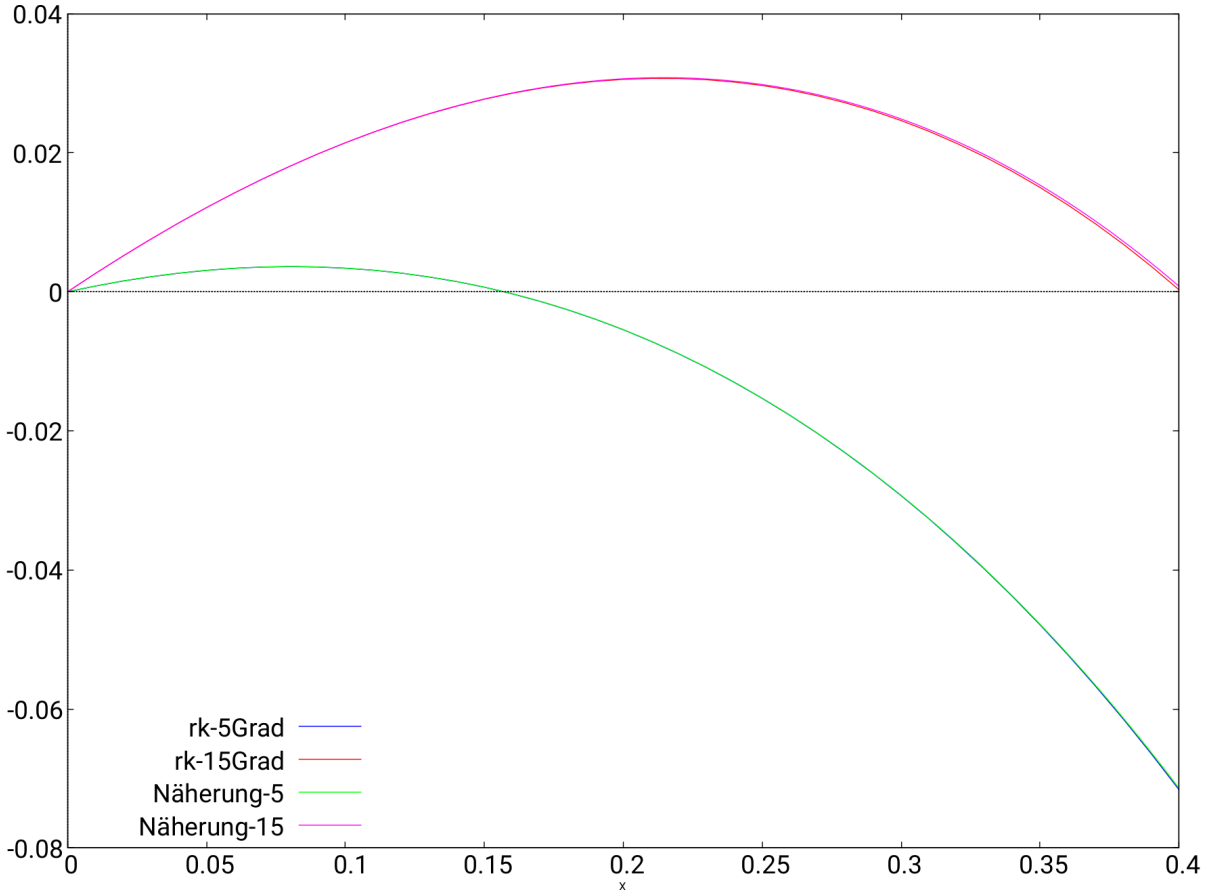


Abb.11 : Short-Time Näherungen vs. Runge-Kutta: kaum unterscheidbar

1.3.7 Herleitung und Lösung der short-time Näherung im Zeitraum

Ausgangspunkt sei wieder unser Differentialgleichungssystem 1.31:

$$\begin{cases} \dot{V}_x = & -V\,V_x \\ \\ \dot{V}_y = & -1 - V\,V_y \end{cases}$$

Beachte, dass es sowohl für V_x als auch für V_y stabile Endgeschwindigkeiten gibt, bei denen die Zeitableitung auf der linken Seite verschwindet - nennen wir sie V_{xf} und V_{yf} (f für "final"):

$$\begin{cases} 0 = & -V\,V_{xf} & \Rightarrow & V_{xf} = 0 \\ \\ 0 = & -1 - V\,V_{yf} & \Rightarrow & 1 = -|V_{yf}|V_{yf} \quad \Rightarrow V_{yf} = -1 \end{cases}$$

Die Horizontalgeschwindigkeit verschwindet und die Vertikalgeschwindigkeit strebt betragsmäßig gegen 1.
Für z.B. Hagelkörner macht das einen beträchtlichen Unterschied - nehmen wir eine Wolkenhöhe von $H = 10\,\text{km}$ an (was nicht ungewöhnlich ist für Gewittertürme) ergibt sich bei $r = 1\,\text{cm}$ ein

$v_{yf} \approx 22\,m/s$ gegenüber im Vakuum mit Überschallgeschwindigkeit $v_{vac} \approx 443\,m/s \approx 1.3c$

In Luft schmerzt es zwar, im Vakuum würde man von einer "Gewehrkugel" getroffen. Hier die Rechnung:

```
(%i9)  (%rho_Eis:0.92*10^3,r:10^(-2),c_W:0.4,%rho_Air:1.2,H:10^4,g:9.81)$
```

Berechnung von ℓ

```
(%i7)  (V:4/3*r^3*%pi,m:V*%rho_Eis, A:r^2*%pi, l:2*m/(%rho_Air*c_W*A))$
(%i10) v_yf:sqrt(g*l);                    22.39196284384198
(%i12) v_vac:box(sqrt(2*H*g));            442.944691807002
```

Zurück zur short-time Näherung: Ist $\dfrac{V_y^2}{V_x^2} \ll 1$ über große Teile der Geschoßbahn, kann man 1.31 wegen $V \approx V_x$ vereinfachen zu

$$\begin{cases} \dot{V}_x = & -V_x^2 \\ \\ \dot{V}_y = & -1 - V_x\,V_y \end{cases} \tag{1.35}$$

Die erste Gleichung von 1.35 ist jetzt entkoppelt und kann leicht durch Variablentrennung integriert werden.
Zur leichteren Lösung der Gleichung für V_y verwenden wir folgenden Trick (ohne Trick siehe ANHANG 1.3.14):

$$h(T) := (\tan\alpha)(T) = \frac{V_y}{V_x} \Rightarrow V_y = h \cdot V_x \Rightarrow \dot{V}_y = \dot{h}\,V_x + h\,\dot{V}_x = \dot{h}\,V_x + \left(\frac{V_y}{V_x}\right)\left(-V_x^2\right)$$

wobei beim letzten Gleichheitszeichen die Definition von h und die erste Zeile von 1.35 verwendet wurden. Vergleich mit der zweiten Zeile von 1.35 führt zu:

$$\dot{h}\, V_x = -1 \Rightarrow \boxed{h(T) = \frac{V_{y0}}{V_{x0}} - \int_0^T \frac{1}{V_x(t)}\, dt} \Rightarrow V_y(T) = h(T) \cdot V_x(T) \tag{1.36}$$

Die Ortskoordinaten bekommen wir mit den beiden Formeln

$$X(T) = X_0 + \int_0^T V_x(t)\, dt \qquad Y(T) = Y_0 + \int_0^T V_y(t)\, dt \tag{1.37}$$

Wenn wir jetzt noch den Zeitparameter elimieren, müssten wir die short-time Näherung im Ortsraum vor uns haben. Ziehen wir obigen Algorithmus in mit *wxMaxima* durch:

Formeln um die Ortskoordinaten zu berechnen

(%i1) `X(T):=X_0+integrate(V_x(t),t,0,T);` $\mathrm{X}(T) := X_0 + \int_0^T V_x(t)dt$ (%o1)

(%i2) `Y(T):=Y_0+integrate(V_y(t),t,0,T);` $\mathrm{Y}(T) := Y_0 + \int_0^T V_y(t)dt$ (%o2)

Die 1. Gleichung von 1.35 wird gelöst:

(%i3) `diffEq1:'diff(V_x,T)=-V_x^2;` $\frac{d}{dT}V_x = -{V_x}^2$ (diffEq1)

(%i4) `gSol1:ode2(diffEq1,V_x,T);` $\frac{1}{V_x} = T + \%c$ (gSol1)

(%i5) `spSol1:ic1(gSol1,T=0,V_x=V_x0);` $\frac{1}{V_x} = \frac{T\,V_{x0}+1}{V_{x0}}$ (spSol1)

V_x ist da!

(%i6) `define(V_x(T),rhs(linsolve (spSol1,V_x)[1]));` $V_x(T) := \frac{V_{x0}}{T\,V_{x0}+1}$ (%o6)

Jetzt unsere "Hilfsfkt" $h(T)$ aus 1.36 berechnet

(%i7) `define(h(T),V_y0/V_x0-integrate(1/V_x(t),t,0,T));` $\mathrm{h}(T) := \frac{V_{y0}}{V_{x0}} - \frac{T^2\,V_{x0}+2T}{2V_{x0}}$ (%o7)

Nun wird damit $V_y(T)$ bestimmt

(%i8) `define(V_y(T),ratsimp(V_x(T)*h(T)));` $V_y(T) := \frac{2V_{y0}-T^2\,V_{x0}-2T}{2T\,V_{x0}+2}$ (1.38)

Einige Voraussetzungen, um X bzw. Y zu berechnen

(%i9) `(X_0:0,Y_0:0)$`

(%i10) `assume(T>0,V_x0>0);` $[T>0, V_{x0}>0]$ (%o10)

Die X-Koordinate ist berechnet!
(%i11) eq1:X=X(T);

$$X = \log\left(T\,V_{x0}+1\right) \qquad \text{(eq1)}$$

Die Y-Koordinate wird berechnet und "zerteilt"!

(%i12) distrib(Y(T));

$$\frac{\log\left(T\,V_{x0}+1\right)V_{y0}}{V_{x0}}+\frac{\log\left(T\,V_{x0}+1\right)}{2{V_{x0}}^2}-\frac{T}{2V_{x0}}-\frac{T^2}{4} \qquad \text{(\%o12)}$$

Von obigem Ausdruck wird aus dem Term 1 und 2 herausgehoben
(%i13) h1:factor(part(distrib(Y(T)),1)+part(distrib(Y(T)),2));

$$\frac{\log\left(T\,V_{x0}+1\right)\left(2V_{x0}V_{y0}+1\right)}{2{V_{x0}}^2} \qquad \text{(h1)}$$

Wir zerteilen den zweiten Teil des obigen Bruches
(%i14) h2:expand(part(h1,1,2)/part(h1,2));

$$\frac{V_{y0}}{V_{x0}}+\frac{1}{2{V_{x0}}^2} \qquad \text{(h2)}$$

So - jetzt schreiben wir die Gleichung so an wie sie uns "gefällt"
(%i15) h3:Y=part(h1,1,1)*h2+part(distrib(Y(T)),3)+part(distrib(Y(T)),4);

$$Y = \log\left(T\,V_{x0}+1\right)\left(\frac{V_{y0}}{V_{x0}}+\frac{1}{2{V_{x0}}^2}\right)-\frac{T}{2V_{x0}}-\frac{T^2}{4} \quad \text{(h3)}$$

Wir eliminieren T: dafür holen wir es aus $X(T)$
(%i16) eq2:solve(eq1,T)[1];

$$T = \frac{\%e^X-1}{V_{x0}} \qquad \text{(eq2)}$$

Jetzt wir zweimal substituiert: das "Endergebnis" stimmt mit vorigem Abschnitt überein - Gott sei Dank!
(%i17) expand(subst(rhs(eq2),T,subst(X,log(T*V_x0+1),h3)));

$$\boxed{Y = -\frac{\%e^{2X}}{4{V_{x0}}^2}+\frac{V_{y0}X}{V_{x0}}+\frac{X}{2{V_{x0}}^2}+\frac{1}{4{V_{x0}}^2}} \qquad \text{(\%o17)}$$

1.3.8 Herleitung einer "long-time"-Näherung im Zeitraum

Um diese Gleichung herzuleiten brauchen wir einige Bausteine:

- Den zweiten Teil von 1.31 umgeschrieben:

$$\dot{V}_y = -1-V_y\left(V_x^2+V_y^2\right)^{\frac{1}{2}} \Rightarrow -\frac{\dot{V}_y+1}{V_yV_x}=\left[1+\left(\frac{V_y}{V_x}\right)^2\right]^{\frac{1}{2}} \qquad (1.39)$$

- Mit $h(T)$ von 1.36 wird daraus

$$-\frac{\dot{V}_y+1}{V_yV_x}=\left[1+\left(h(T)\right)^2\right]^{\frac{1}{2}} \Rightarrow -\frac{\dot{V}_y+1}{V_yV_x}=\left[1+\left(\frac{V_{y0}}{V_{x0}}-\int_0^T\frac{1}{V_x(t)}\,dt\right)^2\right]^{\frac{1}{2}} \qquad (1.40)$$

- Aus 1.31 können wir die "lästige" Wurzel V eliminieren:

$$\frac{\dot{V}_y + 1}{\dot{V}_x} = \frac{V_y}{V_x} \Rightarrow \frac{\dot{V}_y + 1}{V_y} = \frac{\dot{V}_x}{V_x} \tag{1.41}$$

  das setzen wir in die vorige Gleichung ein und erhalten

-

$$-\frac{\dot{V}_x}{V_x^2} = \left[1 + \left(\frac{V_{y0}}{V_{x0}} - \int_0^T \frac{1}{V_x(t)}\, dt\right)^2\right]^{\frac{1}{2}} \tag{1.42}$$

- Mit der Substitution $z(t) := (V_x(t))^{-1}$ wird aus 1.42

$$\boxed{\dot{z} = \left[1 + \left(\frac{V_{y0}}{V_{x0}} - \int_0^T z(t) dt\right)^2\right]^{\frac{1}{2}}} \tag{1.43}$$

Unsere Strategie ist jetzt:

- Lösung von 1.43 für $T \gg 1$
- $V_x(T) = z^{-1}$
-

$$V_y(T) = V_x(T)\, h(T) = V_x \left(\frac{V_{y0}}{V_{x0}} - \int_0^T z(t)\, dt\right)$$

$$\lim_{T\to\infty} V_x = 0 \Rightarrow \lim_{T\to\infty} z(T) = \infty \Rightarrow \lim_{T\to\infty} \int_0^T z(t) dt = \infty$$

Die rechte Seite von Gleichung 1.43 hat folgende Struktur für $Tgg1$:

$$\left[1 + (a - x)^2\right]^{\frac{1}{2}} \quad \text{wobei } x \text{ (das Integral) sehr groß ist!}$$

Wir können also obigen Ausdruck mit einer asymptotischen Taylorreihe ("Entwicklungspunkt ist ∞") annähern (siehe Anhang 1.3.12):

$$\left[1 + (a - x)^2\right]^{\frac{1}{2}} \approx (x - a) + \frac{1}{2\,x} + \frac{a}{2\,x^2} + \frac{4\,a^2 - 1}{8\,x^3} + \cdots \tag{1.44}$$

Wir brechen 1.44 bereits nach dem 1. Reihenglied ab und damit geht 1.43 über in

$$\dot{z} \approx \int_0^T z(t)\,dt - \frac{V_{y0}}{V_{x0}}$$

durch Differenzieren geht dies in folgendes Anfangswertproblem über

$$\ddot{z} = z \qquad \dot{z}(0) = -\frac{V_{y0}}{V_{x0}} \qquad z(0) = \frac{1}{V_{x0}} \tag{1.45}$$

So jezt die Implementation in *wxMaxima*:

(%i1) ratprint:false$;

Berechnungsformel für V_y

(%i2) V_y(T):=V_x(T)*(V_y0/V_x0 - integrate(z(t),t,0,T));

$$V_y(T) := V_x(T)\left(\frac{V_{y0}}{V_{x0}} - \int_0^T \mathrm{z}(t)dt\right)$$

Anfangswertproblem 1.45 wird gelöst

(%i3) eq:'diff(z,t,2) - 'diff(z,t,0) = 0;

$$\frac{d^2}{dt^2}z - z = 0 \qquad \text{(eq)}$$

(%i4) sol1:ode2(eq,z,t);

$$z = \%k1\ \%e^{t} + \%k2\ \%e^{-t} \quad \text{(sol1)}$$

(%i5) expr:(ic2(sol1,t=0,z=1/V_x0,'diff(z,t)=-V_y0/V_x0));

$$z = \frac{(V_{y0}+1)\,\%e^{-t}}{2V_{x0}} - \frac{(V_{y0}-1)\,\%e^{t}}{2V_{x0}}$$

Wir sustituieren die hyperbolischen Funktionen für die Exponentialfkt.

(%i6) define(z(t),rhs(expand(trigsimp(subst(cosh(t)+sinh(t),%e^t,(subst(-sinh(t)+cosh(t),%e^(-t),expr)))))));

$$\mathrm{z}(t) := \frac{\cosh(t)}{V_{x0}} - \frac{V_{y0}\sinh(t)}{V_{x0}}$$

V_x ist die reziproke Fkt. von z, V_y wurde schon festgelegt

(%i7) define(V_x(T),trigsimp(1/z(T)));

$$V_x(T) := -\frac{V_{x0}}{\sinh(T)\,V_{y0} - \cosh(T)} \qquad \text{(\%o7)}$$

(%i8) display(V_y(T))$

$$V_y(T) = -\frac{\cosh(T)\,V_{y0} - \sinh(T)}{\sinh(T)\,V_{y0} - \cosh(T)}$$

Jetzt RUNGE-KUTTA

(%i10) $V_x0:1$V_y0:0.2$

(%i11) eq1:-sqrt(V_x^2+V_y^2)*V_x$

(%i12) eq2:-sqrt(V_x^2+V_y^2)*V_y-1$

```
(%i14) Delta_t:0.01$ simWidth:10$

(%i15) sol_u: rk([eq1,eq2],[V_x,V_y],[V_x0,V_y0],[t,0,simWidth,Delta_t])$

(%i16) plotListVx:map(lambda([x],firstn(x,2)),sol_u)$

(%i17) plotListVy:map(lambda([x],[first(x),last(x)]),sol_u)$
```

Die short-time Näherungen zum Vergleich

```
(%i18) Vst_x(T):=V_x0/(T*V_x0+1);
```

$$Vst_x(T) := \frac{V_{x0}}{T\,V_{x0}+1}$$

```
(%i19) Vst_y(T):=(2*V_y0-T^2*V_x0-2*T)/(2*T*V_x0+2);
```

$$Vst_y(T) := \frac{2V_{y0}-T^2\,V_{x0}+(-2)\,T}{2T\,V_{x0}+2}$$

```
(%i20)  plot2d([Vst_y(x),V_y(x),[discrete,plotListVy]],[x,0,8],[y,-1.1,0.22],
           [gnuplot_term,"qt-0"], [title, "velocity in y-direction"],
           [legend, "   short-time-V_y","   long-time-V_y","   RUNGE-KUTTA-V_y"],
           [style,[lines,3,1,2], [lines,3,2,2],[lines,3,3,2]],
           [gnuplot\_preamble, "set key top right;
           set xtics font \", 15\"; set ytics font \", 15\";
           set key font \", 15\";  set title font \", 20\" "]
       )$

(%i21) plot2d([Vst_x(x),V_x(x),[discrete,plotListVx]],[x,0,8],
           [gnuplot_term,"qt-1"],  [title, "velocity in x-direction"],
           [legend, "   short-time-V_x","   long-time-V_x","   RUNGE-KUTTA-V_x"],
           [style,[lines,3,1,2], [lines,3,2,2],[lines,3,3,2]],
           [gnuplot_preamble, "set key top right;
           set xtics font \", 15\"; set ytics font \", 15\";
           set key font \", 15\"; set title font \", 20\" "]
       )$
```

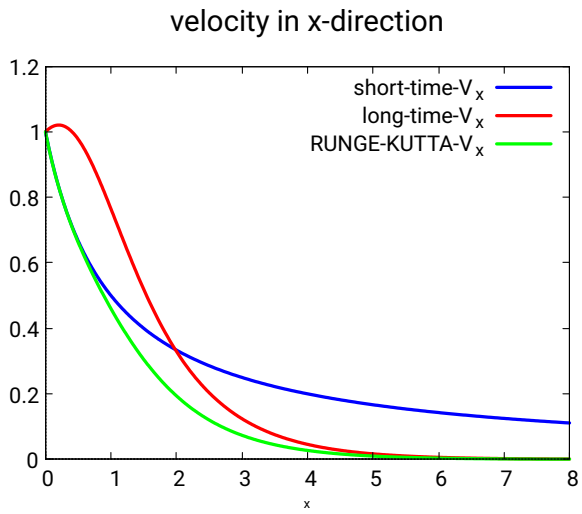


Abb.12 : Vergleich x

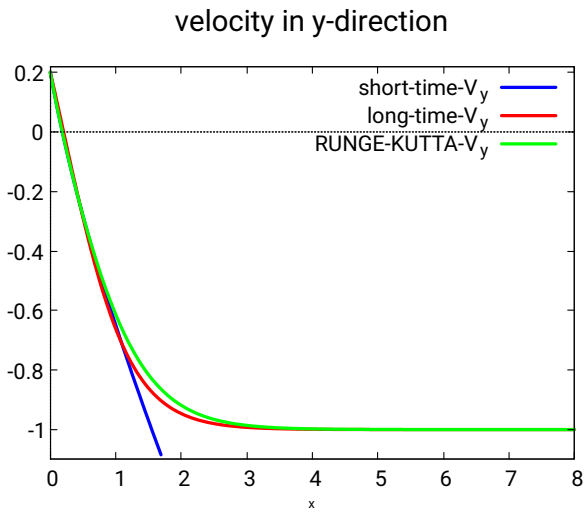


Abb.13 : Vergleich y

Wir haben vorausgesetzt, dass für $T \gg 1$ das Integral $\int_0^T z\,dt$ extrem groß wird, das schauen wir uns noch genauer an:

$$\int_0^T z\,dt \approx \frac{e^T \overbrace{\left(1 - V_{y0}\right)}^{!!}}{2\,V_{x0}} - \frac{e^{-T}\,\left(V_{y0}+1\right)}{2\,V_{x0}} + \frac{V_{y0}}{V_{x0}}$$

Also V_{y0} sollte kleiner 1 sein, wenn eine "ordentliche" Konvergenz gewährleistet sein soll! Der Graph für $V_{y0} = 0.95$ zeigt gleich, wie schlecht die long-time-Näherung dann gleich ist:

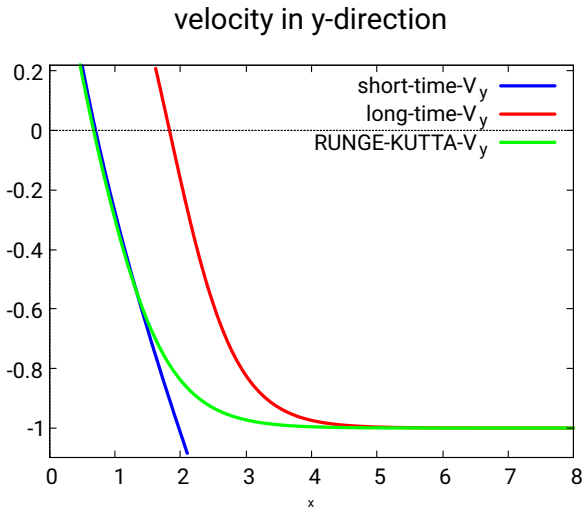


Abb.14 : V_x für $V_{y0} = 0.95$

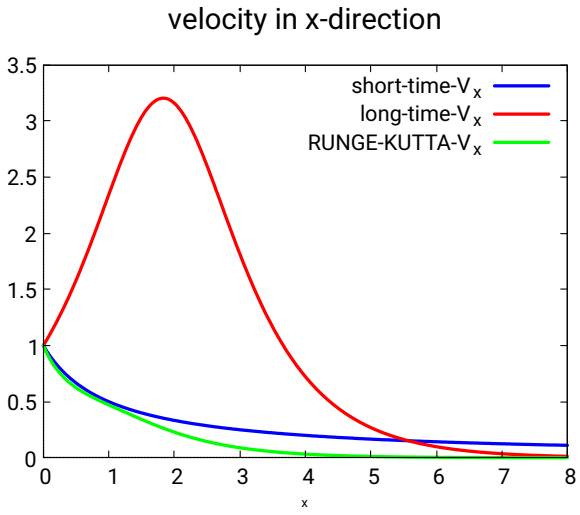


Abb.15 : V_x für $V_{y0} = 0.95$

Man sieht an beiden Graphen wie spät jetzt die long-time-Näherung sich an die tatsächliche Lösung schmiegt - die anfangs steigende x-Geschwindigkeit ist natürlich total daneben.

Eine Trajektorie mit Hilfe der long-time-Näherung ist nicht möglich, da sich der anfängliche Fehler so akkumliert, sodass die Formeln 1.37 kein sinnvolles Ergebnis liefern. Nur wenn man zu einem späten Zeitpunkt t_1 die Koordinaten X_1 und Y_1 kennen würde, könnte man diese Formeln sinnvoll anwenden!

1.3.9 Implizite Lösung der Bahnkurvengleichung im Zeitraum

Wir starten bei 1.43:

$$\dot{z} = \left[1 + \left(\frac{V_{y0}}{V_{x0}} - \int_0^T z(t)dt\right)^2\right]^{\frac{1}{2}} \tag{1.46}$$

und wählen folgenden Substitution

$$w(T) = \int_0^T z(t)dt - \left(\frac{V_{y0}}{V_{x0}}\right) \qquad \text{damit gilt } \dot{w} = z \tag{1.47}$$

damit wird 1.46 zu

$$\ddot{w} = \left[1 + w^2\right]^{\frac{1}{2}} \tag{1.48}$$

multiplizieren wir mit $\dot{w}$ ergibt sich folgender Ausdruck für die noch unbekannte Funktion $f(w)$:

$$\ddot{w}\,\dot{w} = \left[1 + w^2\right]^{\frac{1}{2}} \dot{w} \Leftrightarrow \frac{1}{2}\frac{d}{dT}\left(\dot{w}^2\right) = \frac{df}{dw}\underbrace{\frac{dw}{dT}}_{\dot{w}} \Rightarrow \frac{df}{dw} = \left[1 + w^2\right]^{\frac{1}{2}} \tag{1.49}$$

$$\Rightarrow f(w) = \frac{1}{2}\left(asinh(w) + w\sqrt{1 + w^2}\right) + C \tag{1.50}$$

Mit der Integration von 1.49 ergibt sich

$$\frac{1}{2}\frac{d}{dT}\left(\dot{w}^2\right) = \frac{df}{dw}\frac{dw}{dT} \quad \Big| \int_0^t . \, dT$$

$$\Rightarrow \frac{1}{2}[\dot{w}^2 - \dot{w}_0^2] = f(w) - f(w_0) \Rightarrow \boxed{\dot{w} = \underbrace{\sqrt{2f(w) + \dot{w}_0^2 - 2f(w_0)}}_{s(w)}} \tag{1.51}$$

dabei gilt

$$w_0 = w(0) \overset{1.47}{=} -\left(\frac{V_{y0}}{V_{x0}}\right) \qquad \dot{w}_0 = \frac{1}{z_0} = \frac{1}{V_{x0}} \qquad \dot{w} = s(w)$$

1.51 können wir mit Variablentrennung integrieren und erhalten

$$\int_{w_0}^{w} \frac{dv}{s(v)} = T \Leftrightarrow S(w) = T \nrightarrow w = S^{-1}(T) \tag{1.52}$$

Selbst wenn es gelingen sollte S (das Integral) zu berechnen, ist die Umkehrung unmöglich! Numerisch ist es aber möglich (siehe ANHANG 1.3.13)

$$T \overset{1.52}{\longrightarrow} w \overset{s(w)=\dot{w}}{\longrightarrow} \dot{w} \overset{\dot{w}=z}{\longrightarrow} z \overset{z=V_x^{-1}}{\longrightarrow} V_x \overset{1.36}{\longrightarrow} V_y \Rightarrow \begin{cases} T \longrightarrow V_x \\ \\ T \longrightarrow V_y \end{cases}$$

Diese Möglichkeit schauen wir uns nun in *wxMaxima* an und überprüfen das Ergebnis mit dem Resultat des ursprünglichen Differentialgleichungssystems:

```
(%i2) V_x0:1$V_y0:0.2$

(%i5) w_0:-V_y0/V_x0$w_dot_0:1/V_x0$w_f:150$

      invert(l):=map(lambda([x],[second(x),first(x)]),l)$ /* was never
(%i6) used */
```

Differentialgleichung für $f(w)$ wird gelöst

```
(%i7) eq:'diff(f,w)=sqrt(1+w^2);
```

$$\frac{d}{dw}f = \sqrt{w^2+1} \qquad \text{(eq)}$$

```
(%i8) logarc:true$ /*asinh is substuted by log */;
```

```
(%i9) sol:ode2(eq,f,w);
```

$$f = \frac{\log\left(\sqrt{w^2+1}+w\right)}{2} + \frac{w\sqrt{w^2+1}}{2} + \%c \qquad \text{(sol)}$$

$\longrightarrow$ Integrationsconstante C wegen Differenz unwichtig!

```
(%i10) sol1:expand(ic1(sol,w=0,f=0));
```

$$f = \frac{\log\left(\sqrt{w^2+1}+w\right)}{2} + \frac{w\sqrt{w^2+1}}{2} \qquad \text{(sol1)}$$

```
(%i11) define(f(w), rhs(sol1));
```

$$\mathrm{f}(w) := \frac{\log\left(\sqrt{w^2+1}+w\right)}{2} + \frac{w\sqrt{w^2+1}}{2} \qquad \text{(\%o11)}$$

```
(%i12) s(w):=sqrt(2*f(w)+w_dot_0^2-2*f(w_0));
```

$$\mathrm{s}(w) := \sqrt{2\,\mathrm{f}(w) + w\_dot_0{}^2 + (-2)\,\mathrm{f}\left(w_0\right)} \qquad \text{(\%o12)}$$

Wir bekommen die $w \leftrightarrow T$ Tabelle von Runge-Kutta

```
(%i13) wT_list:rk(1/s(w),y,0,[w,w_0,w_f,0.1])$
```

zur Erinnerung: $\dot{w} = s(w) = 1/V_x$

```
(%i14) get_T_To_Vx(rk_list):=map(lambda([x],[second(x),1/s(first(x))]),rk_list)$
```

zur Erinnerung: $V_y = -w/\dot{w} = -w/s(w)$

```
(%i15) get_T_To_Vy(rk_list):=map(lambda([x],[second(x),-first(x)/s(first(x))]),rk_list)$
```

```
(%i16) T_max:last(last(wT_list));
```

$$5.40561416361086 \qquad \text{(T\_ max)}$$

Zum Vergleich lösen wir die originalen Differentialgleichungen

```
(%i17) eq1:-sqrt(v_x^2+v_y^2)*v_x;
```

$$-v_x\sqrt{{v_y}^2+{v_x}^2} \qquad \text{(eq1)}$$

```
(%i18) eq2:-sqrt(v_x^2+v_y^2)*v_y-1;
```

$$-v_y\sqrt{{v_y}^2+{v_x}^2}-1 \qquad \text{(eq2)}$$

```
(%i19) sol_rk: rk([eq1,eq2],[v_x,v_y],[V_x0,V_y0],[t,0,T_max,0.1])$
```

Wir extrahieren V_x bzw. V_y

```
(%i20) only_t_Vx(l):=map(lambda([x],firstn(x,2)),l)$
```

```
(%i21) plot2d([[discrete,get_T_To\_Vx(wT_list)],[discrete,only_t_Vx(sol\_rk)] ],
          [y,0,1],[x,0,T\_max], [gnuplot_term,"qt-1"],  [title, "velocity in x-direction"],
          [legend, "   V_x-via implicit equation","   V_x-via original diffEqu  "],
          [style,[lines,6,1,2], [lines,3,2,2]], [gnuplot\_preamble, "set key top right;
            set xtics font\", 15\"; set ytics font\", 15\"; set key font\", 15\";
            set title font\", 20\" "])$
```

```
(%i22) only_t_Vy(l):=map(lambda([x],[first(x),third(x)]),l)$
```

```
(%i23) plot2d([[discrete,get_T_To_Vy(wT_list)],[discrete,only_t_Vy(sol_rk)] ],
          [gnuplot_term,"qt-2"],[x,0,T_max], [title, "velocity in y-direction"],
          [legend, "   V_y-via-implicit","   V_y-via-diffEq"],
          [style,[lines,6,1,2], [lines,3,2,2]],  [gnuplot_preamble, "set key top right;
          set xtics font\", 15\"; set ytics font\", 15\"; set key font\", 15\";
          set title font\", 20\" "])$
```

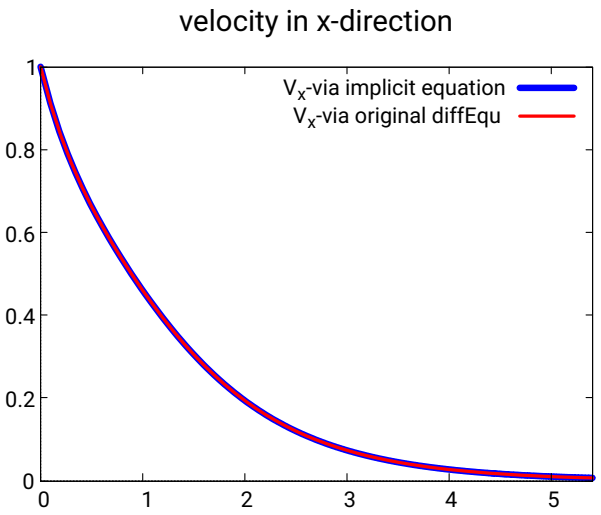


Abb.16 : Übereinstimmung der beiden Lösungen

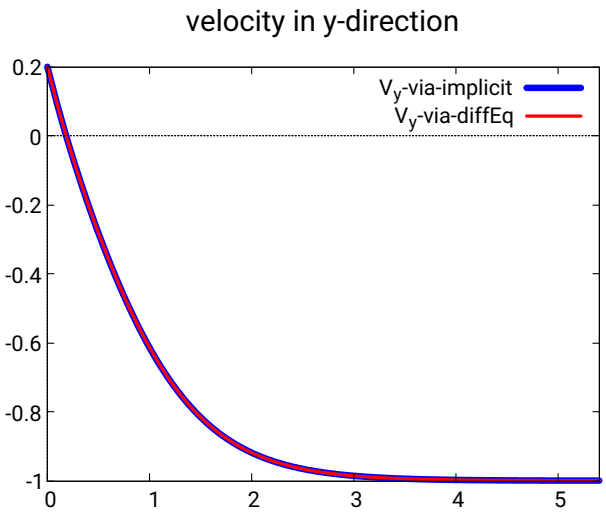


Abb.17 : Übereinstimmung der beiden Lösungen

Wir haben früher bei einer Billardkugel berechnet $\frac{1}{\ell} \approx 4 \cdot 10^{-3} \Rightarrow \ell \approx 250$
Bei unseren Anfangsbedingungen
$V_x = 1$, $V_y = 0.2 \Rightarrow \alpha \approx 11^\circ$ wäre bei $T \approx 5 \Rightarrow t \approx 5\,T = 25\,s$
der Endzustand erreicht:

$$v_x \approx 0 \qquad v_y = \sqrt{g\ell}(-1) \approx -50\,m/s$$

Aber jetzt noch zur Bahnkurve: Natürlich geht es auch wieder mit 1.29, aber es geht auch anders:

$$V_x = \frac{dX}{dt} = \frac{dX}{dw}\frac{dw}{dt} \Rightarrow \frac{dX}{dw} = \frac{V_x}{\dot{w}} \quad \text{außerdem gilt} \quad V_x = \frac{1}{\dot{w}} = \frac{1}{s(w)} \tag{1.53}$$

damit ergibt sich

$$\int_0^T V_x(t)\,dt = \int_{w_0}^{w} \frac{dX}{dw}\,dw = \int_{w_0}^{w} \frac{V_x}{\dot{w}}\,dw = \int_{w_0}^{w} \frac{dw'}{[s(w')]^2} \Rightarrow$$

$$X(T(w)) = X_0 + \int_0^T V_x(t)\,dt = X_0 + \int_{w_0}^{w} \frac{dw'}{[s(w')]^2} \tag{1.54}$$

Bei $Y(T)$ läuft es ähnlich, nur gilt hier $V_y = -\frac{w}{\dot{w}}$ damit ergibt sich

$$Y(T(w)) = Y_0 + \int_{w_0}^{w} \frac{V_y}{\dot{w}}dw = -\int_{w_0}^{w} \frac{w'}{[s(w')]^2}dw' \tag{1.55}$$

wobei die bijektive Abbildung w zu T bzw. umgekehrt nur als Tabelle vorliegt. Aber wir können für die Berechnung der Bahn nur auf die w-Werte bzw. $s(w)$-Werte zurückgreifen. Als "Probe" benutzen wir die *short-time* Approximation.
Diese Methode testen wir mit *wxMaxima*:

(%i2) V_x0:1$V_y0:0.2$

Hier nocheinmal die short-time Näherung

(%i3) Y_st(X):=-%e^(2*X)/(4*V_x0^2)+X/(2*V_x0^2)+V_y0/V_x0*X+1/(4*V_x0^2);

$$Y_{st}(X) := \frac{-\%e^{2X}}{4{V_{x0}}^2} + \frac{X}{2{V_{x0}}^2} + \frac{V_{y0}}{V_{x0}}X + \frac{1}{4{V_{x0}}^2} \tag{\%o3}$$

(%i6) w_0:-V_y0/V_x0$w_dot_0:1/V_x0$w_f:0.4$

(%i7) f(w):=log(sqrt(w^2+1)+w)/2+(w*sqrt(w^2+1))/2; $\mathrm{f}(w) := \frac{\log\left(\sqrt{w^2+1}+w\right)}{2} + \frac{w\sqrt{w^2+1}}{2}$

(%i8) s(w):=sqrt(2*f(w)+w_dot_0^2-2*f(w_0)); $\mathrm{s}(w) := \sqrt{2\,\mathrm{f}(w) + \dot{w}_0^2 + (-2)\,\mathrm{f}(w_0)}$

Wir berechnen die Tabellen mit Runge-Kutta

(%i9) wX_list:rk(1/(s(w))^2,X,0,[w,w_0,w_f,0.01])$

(%i10) wY_list:rk(-w/(s(w))^2,Y,0,[w,w_0,w_f,0.01])$

Wir stellen die Plot-Liste zusammen

```
(%i11) getXY_list(xl,yl):=block([xy_list:[]],
        for i thru length(xl) do
              xy_list:cons([second(xl[i]),second(yl[i])],xy_list),
       reverse(xy_list))\$
```

(%i12) XY_list:getXY_list(wX_list,wY_list)$

```
(%i13) plot2d([Y_st(w),[discrete,XY_list]],[w,0,0.4],
       [title, "implicit vs. short-time trajectory"],
           [legend, "   short-time","   implicit"], [style,[lines,6,1,2], [lines,3,2,2]],
           [gnuplot\_preamble, "set key top right; set xtics font \", 15\";
           set ytics font \", 15\"; set key font \", 15\"; set title font \", 20\" "])$
```

Wir berechnen das Integral $\int_{w_0}^{w_f} 1/s(w)\,dw = T_f$ um die Endzeit zu bestimmen

print("w=0.4 entspricht die Zeit: T =",quad_qag(1/s(w),w,w_0,w_f,3)[1])$

w=0.4 entspricht die Zeit: T = 0.4825063259906707

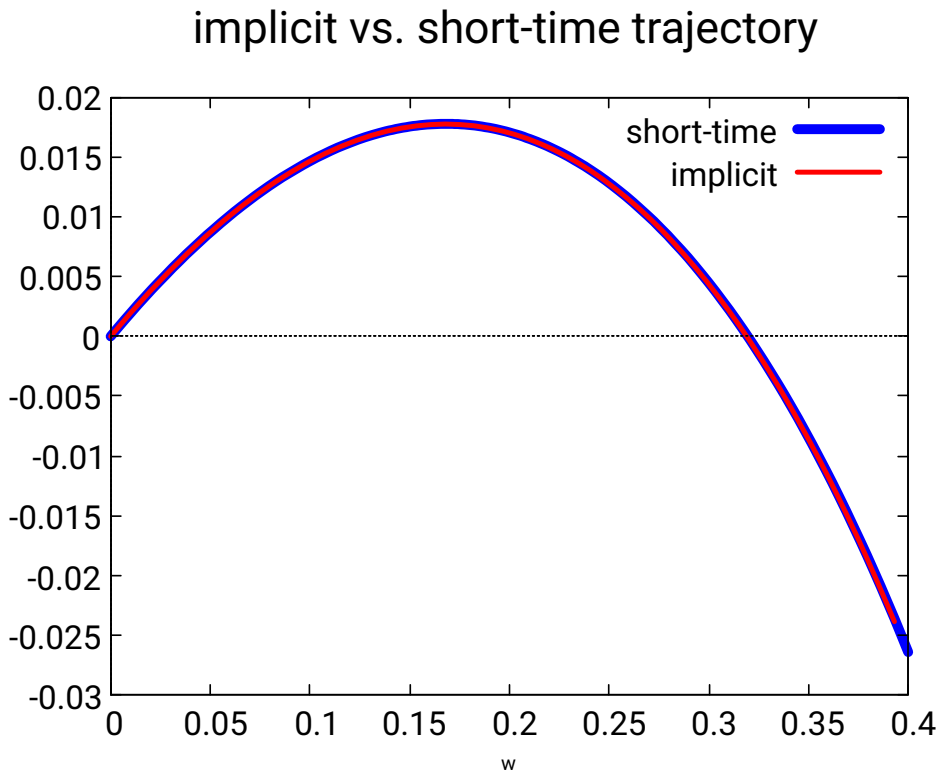


Abb.18 : Short-time vs. implicit trajectory

1.3.10 Allgemeine Lösung der Bahnkurvengleichung im Ortsraum

Wie schon gesagt, besitzt 1.33 keine geschlossene Lösung, aber es ist eine Reihenlösung a la Taylor angebbar. Mit CAS-Tools wie *wxMaxima* sind ziemlich genaue Ergebnisse erzielbar. Aber jetzt der "Reihe" nach:

$$Y(X) = \sum_{k}^{\infty} \frac{a_k}{k!} X^k \qquad \text{mit} \quad a_k := Y^{(k)}(0)$$

wobei a_0, a_1 und a_2 bereits von 1.33 bekannt sind. Die folgenden kann man mit 1.33 rekursiv berechnen:

$$a_3 = 2a_2\sqrt{1 + a_1^2}$$

um a_4 zu erhalten, muss man 1.33 ableiten:

$$Y^{(4)} = 2\frac{Y^{(3)}(1 + Y'^2) + Y''\,Y'}{\sqrt{1 + Y'^2}} \Rightarrow a_4 = 2\frac{a_3(1 + a_1^2) + a_2\,a_1}{\sqrt{1 + a_1^2}}$$

Für a_5 wieder ableiten und die bisher bekannten a_i einsetzen.
Bezeichnen wir 1.33 mit `eq[1]` und Ableitung davon mit `eq[2]` und Ableitung davon mit `eq[3]` ergibt sich folgendes Schema (`calcCoeff(till)`):

```
a[1], a[2]          →  eq[1]  →  a[3]
a[1], ...  a[3]     →  eq[2]  →  a[4]
a[1], ...  a[4]     →  eq[3]  →  a[5]
a[1], ...  a[5]     →  eq[4]  →  a[6]
⋮
```

Dieses Verfahren ist zwar mühsam, eignet sich aber gut für ein Computer Algebra System (CAS) wie *wxMaxima*. Also muss ein Programm her - einige Besonderheiten vorweg:

`a:make_array(hashed,30)` reserviert ein Array mit maximal 30 Feldern, deren Daten mathematische Ausdrücke sind - keine Zahlen

`depends(Y,X)` gibt *wxMaxima* zu verstehen, dass Y von X abhängt, obwohl wir diese Abhängigkeit nicht kennen.

Hier nun der Vergleich zwischen dem numerischen RUNGE-KUTTA-Verfahren, unserer Potenzreihe bis zum Grad 16 und der short-time Näherung für 30 Grad - wir sehen in Abb. 19, das zwischen Potenzreihe und Runge-Kutta kein wahrnehmbarer Unterschied ist und auch die short-time-Näherung noch beachtlich gut ist.
Wäre der Abschusswinkel flacher könnte auch sie gut mithalten!

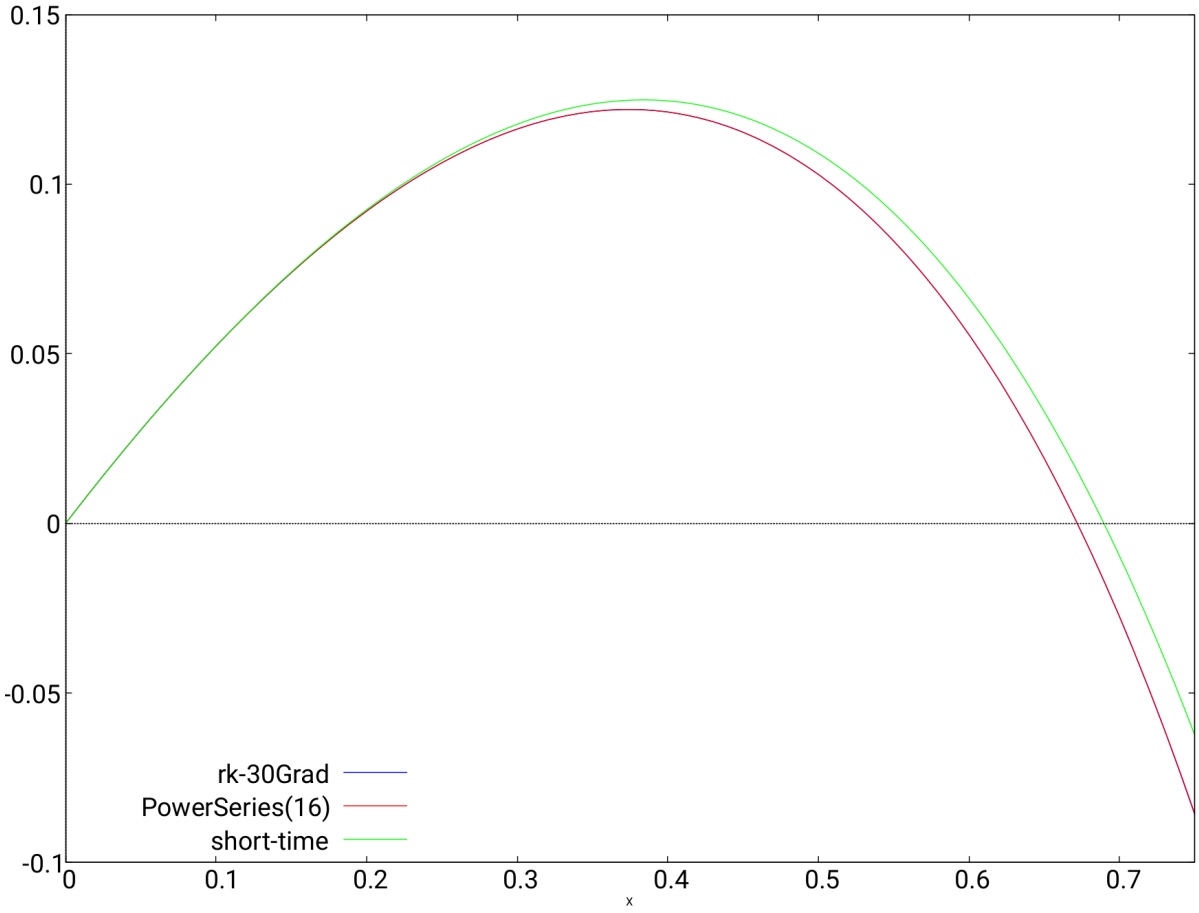


Abb.19 : Vergleich shorttime-Näherung, Potenzreihe und Runge-Kutta

Rechnet man die dimensionslosen Größen für eine Billardkugel um, ergeben die Wurfweite, Wurfhöhe und Anfangsgeschwindigkeit in x-Richtung für obige Bahn folgende Zahlen Zahlen:

$$\ell = \frac{1}{k} \approx 230 \qquad X = 0.67 \;\Rightarrow\; x = X \cdot \ell \approx 154\,m$$

$$Y = 0.12 \;\Rightarrow\; y = Y \cdot \ell \approx 28\,m$$

$$V_{X0} = 1 \;\Rightarrow\; v_{x0} = 1\sqrt{g\ell} \approx 48\,m/s \approx 171\,km/h$$

Hier nun der Code für obige Bahnkurve:

Wir reservieren “hashed-arrays” - siehe oben im Text!

```
(%i3)  a:make_array(hashed,30)$eq:make_array(hashed,30)$fpprintprec:2$
```

Die Anfangswerte für die Potenzreihe werden festgelegt

```
(%i6)  a[0]:0$a[1]:tan(%alpha)$a[2]:-1/V_X0^2$
```

Y ist abhängig von X; mit *dependencies* werden alle Abhängigkeiten aufgelistet

(%i8)
```
depends(Y,X)$dependencies;
```

$$[\mathrm{Y}(X)] \tag{\%o8}$$

Die Ausgangsdifferentialgleichung wird festgelegt

(%i9)
```
eq[1]:'diff(Y,X,3)=2*'diff(Y,X,2)*sqrt(1+'diff(Y,X,1)^2);
```

$$\frac{d^3}{dX^3}Y = 2\sqrt{\left(\frac{d}{dX}Y\right)^2+1}\left(\frac{d^2}{dX^2}Y\right) \tag{\%o9}$$

Ersetzt in *eq* der Reihe nach $Y^{(i)}$ durch $a_i = Y^{(i)}(0)$ mit $i \in \{1, 2, \ldots n\}$ und berechnet a_{n+1}

(%i10)
```
substn(eq,n):=block([res:rhs(eq),k:n+1],
    assume(cos(\%alpha)\ensuremath{>}0),
    for i:1 thru n do res:subst(a[i],'diff(Y,X,i),res),
    a[k]:expand(ratsimp(trigsimp(res)))
)$
```

Die neue Differentialgleichung wird berechnet (Doppelapostroph) und mit der alten neuer Koeffizient berechnet - siehe obiges Schema!

(%i11)
```
calcCoeff(till):=block(
     for j:2 thru till do (
         eq[j]:ratsimp(''diff(eq[j-1],X,1)),
         substn(eq[j-1],j)
     )
)$
```

Berechnung bis a_{16}

(%i12)
```
calcCoeff(15);
```

Die short-time Näherung - die brauchen wir zum Vergleich später!

(%i14)
```
Y_st:-%e^(2*X)/(4*V_X0^2)+X/(2*V_X0^2)+tan(%alpha)*X+1/(4*V_X0^2);
```

$$-\frac{\%e^{2X}}{4{V_{X0}}^2}+\frac{X}{2{V_{X0}}^2}+\tan(\alpha)X+\frac{1}{4{V_{X0}}^2} \tag{Y\_st}$$

Die Koeffizienten "unserer" Potenzreihe im Vergleich mit der short-time Näherung

(%i15)
```
displayDiff(till):=block(
     b:make_array(hashed,30),
     Y_st_taylor: taylor(Y_st,X,0,20),
    print("Coeff of X^i","     ", "short-time", "    ", "power-series"),
    for i thru till do (
      b[i]: coeff(Y_st_taylor,X,i),
      print("i =",i,"            ",b[i],"           ", expand(ratsimp(a[i]/i!)))
    )
)$
```

Jetzt die Ausführung: Bis $\mathcal{O}(2)$ herrscht Übereinstimmung

(%i16) displayDiff(4);

Coeff of X^i	short-time	power-series
i=1	$\tan(\alpha)$	$\tan(\alpha)$
i=2	$-\frac{1}{2{V_{X0}}^2}$	$-\frac{1}{2{V_{X0}}^2}$
i=3	$-\frac{1}{3{V_{X0}}^2}$	$-\frac{1}{3\cos(\alpha){V_{X0}}^2}$
i=4	$-\frac{1}{6{V_{X0}}^2}$	$\frac{\sin(\alpha)}{12{V_{X0}}^4}-\frac{1}{6\cos(\alpha)^2{V_{X0}}^2}$

Wir machen eine Liste mit den einzelnen Termen und ...

(%i17) powerList:makelist(ratsimp((a[i]/i!)*X^i),i,0,16)$

... reduzieren obige Liste mit "+" - bilden also die Summe!

(%i18) powerSeries:lreduce("+",powerList)$

Jetzt legen wir die Werte fest: Abschusswinkel 30°, $V_{X0}=1$ und die Simulationsweite (durch Probieren)

(%i21) %alpha:%pi/6$V_X0:1$simWidth:0.75$

Wir legen obige Potenreihe als Funktion $Y(X)$ für den Plotbefehl fest

(%i22) define(Y(X),float(powerSeries))$

Nur wegen der Neugier schauen wir uns die Potenzreihe auch mal an!

(%i23) float(Y(X));

$$-0.0025X^{16}+0.0029X^{15}+0.0063X^{14}+0.0072X^{13}+0.003X^{12}-0.0087X^{11}-0.026X^{10}-0.041X^9$$
$$-0.038X^8-0.014X^7+0.0049X^6-0.036X^5-0.18X^4-0.38X^3-0.5X^2+0.58X \tag{1.56}$$

Nun das RUNGE-KUTTA Verfahren für die ursprüngliche Diffglg. dritter Ordnung - gleich wie bei short-time Näherung

(%i24) sol30:rk([z_2,z_3, 2*z_3*sqrt(1+z_2^2)],[z_1,z_2,z_3],[0,tan(%alpha),-1/V_X0^2],[x,0,simWidth,0.01])$

Für den Plotbefehl benötigen wir nur (x, z_1) - das sind jeweils die ersten 2 Komponenten der Lösungsliste

(%i25) plotPoints30:map(lambda([x],firstn(x,2)),sol30)$

Jetzt sehen wir und die 3 Funktionen an: Runge-Kutta, Potenzreihe und short-time-Näherung

```
(%i26) plot2d([[discrete,plotPoints30],Y(X),Y\_st],[X,0,simWidth],
          [legend, "   rk-30Grad","   PowerSeries(16)", "   short-time"],
          [gnuplot\_preamble, "set key bottom left; set xtics font \", 20\";
                              set ytics font \", 20\"; set key font \", 20\" "]);
```

Wie man bei 1.56 (das Näherungspolynom) erkennen kann, zwingt die Fakultät im Nenner keineswegs die Koeffizienten gegen Null, d.h. eine schnelle Konvergenz kann man nur bei

$$X < 1 \quad \Rightarrow \quad x < \ell = \frac{1}{k} = \frac{2m}{\rho_M c_W A}$$

erwarten. Also das Verhältnis der reibungsbestimmenden Größen (Dichte des Mediums, Querschnittfläche, Widerstandsbeiwert) zur Masse des Körpers sollte groß sein.
Die Flugbahn eines Papierblatts bleibt weiterhin unberechenbar!

Zur Demonstration hier noch die Flugbahnberechnungen für $X > 1$. Man sieht dass unsere Potenzreihennäherung den Einfluss der Luftreibung unterschätzt (erst recht natürlich die short-time-Näherung). Die Bewegung geht im Grenzfall zu einem reinen vertikalen Fall mit konstanter Geschwindigkeit über!

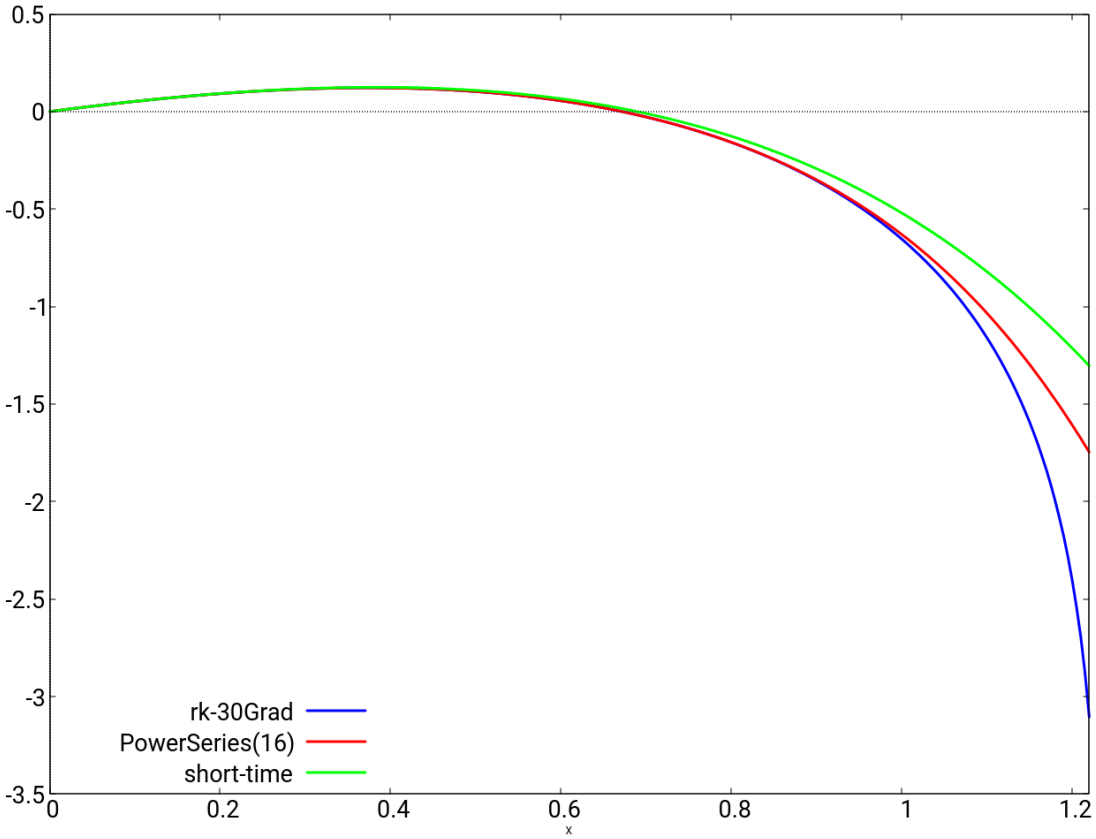


Abb.20 : Übergang in reine Fallbewegung

Damit ist unser Ausflug zum "Werfen" und "Schießen" in Luft zu Ende. Folgende Quellen wurden verwendet:

G.W. Parker: "Projectile motion with air resistance quadratic in the speed", North Carolina, 1977
Riccardo Borghi: "Trajectory of a body in a resistant medium: an elementary derivation" ,
2013 Eur. J. Phys. 34 359

1.3.11 ANHANG: dimensionslose Variablen

Um das Prinzip zu erklären, nehmen wir zu Beginn eine einfache Gleichung:

$$s = s_0 - v\,t_0 \tag{1.57}$$

s_0 und t_0 seien Konstanten aus $\mathbb{R}$ mit den Einheiten "Weg" bzw. "Zeit". Die Variablen s und v besitzen die Einheiten "Weg" bzw. Geschwindigkeit, sodass $v\,t_0$ ebenfalls die Einheit eines "Weges" hat - sonst könnte man die Subtraktion ja nicht ausführen("Zwetschkenknödel minus Äpfel"?). Um auf dimensionslose Einheiten zu kommen, dividieren wir 1.57 durch s_0:

$$\frac{s}{s_0} = 1 - v\,\frac{t_0}{s_0} \tag{1.58}$$

Wir können jetzt eine dimensionslose "Wegeienheit" $S := \frac{s}{s_0}$ und "Geschwindigkeitseinheit" $V := v\frac{t_0}{s_0}$ festlegen, sodass 1.58 sich vereinfacht zu

$$S = 1 - V \tag{1.59}$$

Durch diese Einheitenumstellung bleibt auch die Zeit nicht unberührt, denn es gilt:

$$\frac{S}{V} = T = \frac{s\,s_0}{v\,s_0\,t_0} = \frac{t}{t_0} \tag{1.60}$$

Was ist dadurch gewonnen? Wie fast immer gibt es pros und cons:

- Da die Konstanten verschwunden sind, lässt sich 1.59 sicher leichter und universeller (für alle Konstanten) lösen als 1.57
- Nachdem 1.59 gelöst ist, muss man die Lösungen ins SI-System zurückverwandeln - das ist mit den Transformationsgleichungen nicht schwer aber doch Arbeit.
- Es kommt also darauf an: Ist die Ausgangsgleichung schwer zu knacken (denken Sie an eine heikle Differentialgleichung) wird die Vereinfachung zu dimensionslosen Einheiten- vermutlich schwerer wiegen, als die zusätzliche Arbeit anschließend für die Umwandlung.

Je schwieriger die Ausgangsgleichung, umso mehr spricht für dimensionslose Größen!

Nehmen Sie die Pendelgleichung:

$$m\ddot{x} = -m\,g\,\sin\theta \qquad x(\theta) = L\,\theta \Rightarrow \ddot{x} = \frac{d}{dt}\frac{dx}{d\theta}\frac{d\theta}{dt} = L\frac{d^2\theta}{dt^2}$$

$$\frac{d^2\theta}{dt^2} - \frac{g}{L}\sin\theta = 0 \leftrightarrow \frac{d^2\theta}{dT^2} - \sin\theta = 0$$

1.3.12 ANHANG: asymptotische Taylorreihe

Gibt man für den Entwicklungspunkt einer Funktion "∞" in *wxMaxima* ein, gibt es keine Fehlermeldung, sondern einen Term, der auch negative Exponenten enthält. Für $x \gg 1$ liefert das eine ziemliche gute Näherung - man spricht von **asymptotischer Näherung** durch eine Taylorreihe. Aber was macht *wxMaxima* da genau?

```
(%i3)  f(x):=sqrt(1+x^2)$
(%i9)  exptdispflag:false$
```

(%i10) `define(t(x),taylor(f(x),x,inf,4));` $\quad t(x) := x + \frac{x^{-1}}{2} - \frac{x^{-3}}{8} + \ldots$ (%o10)/T)

(%i17) `float(f(100));` 100.0049998750062 (%o17)

(%i18) `float(t(100));` 100.004999875 (%o18)

Es passiert folgendes:

- In $f(x)$ wird x durch ξ^{-1} ersetzt (Beachte: $x \to \infty \Leftrightarrow \xi \to 0$)

$$f(\frac{1}{\xi}) = \sqrt{1+\xi^{-2}}$$

- Diese Funktion wird jetzt mit Entwicklungspunkt 0 "taylorisiert":
  `define(g(%xi),taylor(f(1/%xi),%xi,0,4))`

$$\xi^{-1} + \frac{\xi}{2} - \frac{\xi^3}{8} + \cdots$$

- In g wird jetzt wieder ξ durch x^{-1} ersetzt:
  `expand(g(1/x))`

$$x + \frac{x^{-1}}{2} - \frac{x^{-3}}{8}$$

  Also halten wir fest:

$$f(x) \approx g\left(\frac{1}{x}\right) \quad \text{für } x \gg 1$$

$$\text{mit } g(x) := taylor(f(1/x), x, 0, n))$$

1.3.13 ANHANG: Invertierung einer Funktion mit Runge-Kutta

Angenomen wir hätten eine Funktion $y(w)$ mit

$$y(w) := w_0 + \int_{w_0}^{w} f(x)\,dx \quad \text{mit } w_0 \in \mathbb{R} \quad \Leftrightarrow y = F(w) \tag{1.61}$$

- wir suchen aber die Umkehrfunktion $w = F^{-1}(y)$. Bei etwas komplexeren f müssen wir froh sein, wenn wir das Integral "knacken" können - von der anschließenden Invertierung von F ganz zu schweigen. Wenn wir allerdings mit einer Tabelle zufrieden sind, die "jedem" w ein y zuordnet, dann können wir es numerisch schaffen!
Zuerst verwandeln wir obige Funktion y in ein Anfangswertproblem:

$$\frac{dy}{dw} = f(w) \qquad y(w_0) = w_0$$

Mit `rk(f(w),y,w_0,[w,w_0,2,0.1])` erstellen wir eine Liste und vertauschen die Komponenten. Das implementieren wir jetzt in *wxMaxima*:

(%i1) `m(x):=x; /* die 1. Mediane als Bezugsgerade */` $\quad m(x) := x$ (%o1)

Das "Versuchskaninchen" - einfach genug, um y^{-1} zu bestimmen

(%i2) `f(x):=x^2;` $\quad f(x) := x^2$ (%o2)

(%i7) `w_0:1$ /* Setzen irgendneinen Wert */`

(%i3) `define(y(w),w_0+integrate(f(x),x,1,w));` $\quad y(w) := \frac{w^3}{3} + \frac{2}{3}$ (%o3)

(%i4) `eq:y(w)=x;` $\quad \frac{w^3}{3} + \frac{2}{3} = x$ (eq)

Nur Lösung [3] ist reell und unsere Umkehrfkt.

(%i5) `sol:solve(eq,w);` $[w = \frac{\left(\sqrt{3}\%i - 1\right)\,(3x-2)^{\frac{1}{3}}}{2}, w = -\frac{\left(\sqrt{3}\%i + 1\right)\,(3x-2)^{\frac{1}{3}}}{2}, w = (3x-2)^{\frac{1}{3}}]$

(%i6) `y_inv(x):=rhs(sol[3])$`

Jetzt die Tabelle $y \leftrightarrow w$

(%i8) `plotList:rk(f(w),y,w_0,[w,w_0,2,0.1])$`

Vertauschen der Komponenten einer Liste aus Paaren

(%i9) `invert(l):=map(lambda([x],[second(x),first(x)]),l)$`

(%i10)
```
plot2d([[discrete,plotList],[discrete,invert(plotList)],y(x),y_inv(x),m(x)],
    [x,1,2],[y,1,2.5], [same_xy, true], [legend, "   R-K","   R-K-invers",
      "   y - analytisch", "   y^{}{-1} - analytisch","   Mediane"],
    [gnuplot\_preamble, "set key top left; set xtics font \", 10\";
      set ytics font \", 10\"; set key font \", 15\" "],
    [style,[lines,6,1,2], [lines,6,2,2],[lines,3,3,2],[lines,3,6,2], [lines,1,5,2]])$
```

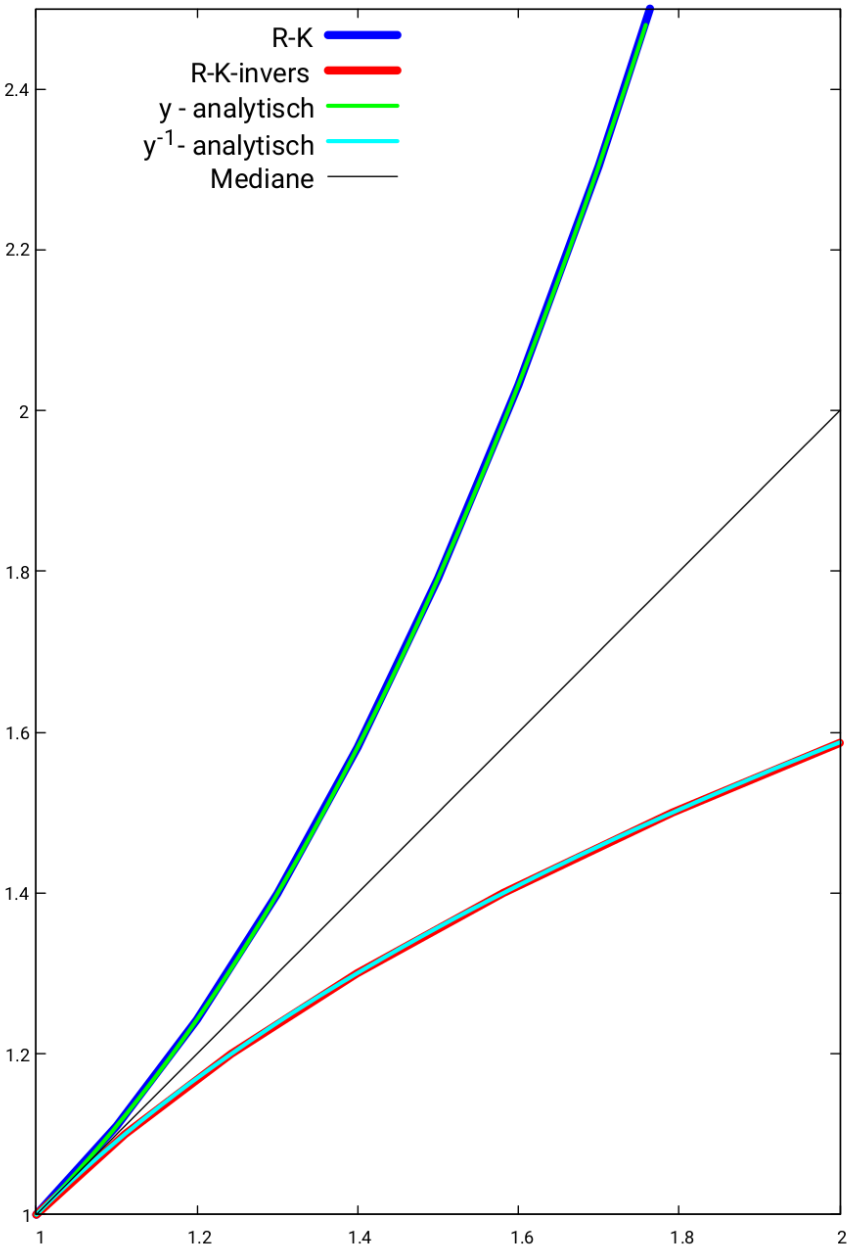


Abb.21 : Invertierung der Funktionstabelle vs. analytische Methode

Runge-Kutta Verfahren und anschl. Vertauschung liefert tabellarisch denselben Graph wie die analytische Methode (die nur bei "einfachen" Funktionen f gelingt).
Außerdem können wir das Runge-Kutta Verfahren auch dazu verwenden (wie das bei den *wxMaxima*-Programmen geschehen ist), um tabellarisch die Integralgleichung 1.61 zu lösen!

1.3.14 ANHANG: Lineare Differentialgleichung erster Ordnung

$$\dot{u} = a(t)u \qquad\qquad \dot{u} = a(t)u + s(t) \tag{1.62}$$

Differentialgleichungen obiger Struktur heißen *homogene bzw. inhomogene lineare Differentialgleichungen erster Ordnung*. Die zweite Gleichung von 1.35 ist von dieser Gestalt mit

$$a(t) := -V_x(t) \quad \text{und} \quad s(t) := -1$$

Die homogene Gleichung

Wie meist führt die Lösung der inhomogenen Gleichung über die Lösung der homogenen. Also knöpfen wir uns diese vor:
Wenn $a(t)$ stetig ist besitzt sie auch eine Stammfunktion z.B.

$$A(t) := \int_{t_0}^{t} a(x)\,dx \quad \text{bzw. irgendeine Stammfkt.} \int a(t)\,dt$$

Dann sind alle Lösungen der homogenen linearen Gleichung von der Gestalt

$$\boxed{y(t) := C \exp\left(\int a(x)\,dx\right)} \quad \text{mit } C \in \mathbb{R} \text{ Anpassungsfaktor für Anfangswertproblem}$$

Sei $z(t)$ irgendeine andere Lösung der homogenen Gleichung 1.62, dann gilt

$$\frac{d}{dt}\frac{y}{z} = \frac{\dot{y}z - y\dot{z}}{z^2} = \frac{ayz - yaz}{z^2} = 0 \Rightarrow y = Cz \quad \text{mit } C \in \mathbb{R}$$

Beispiel 1.1

$$\dot{u} = \sin(t) \cdot u \qquad u(0) = 1 \Rightarrow u(t) = e^{1-\cos(t)}$$

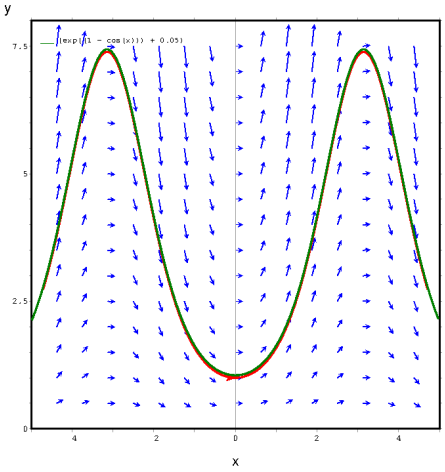


Abb.22 : Direction-Field mit Trajektorie

```
plotdf(sin(x)*y,[trajectory_at,0,1],
  [xfun,"exp(1-cos(x))+0.05"],
  [y,0,8],[x,-5,5]);
```

Die Funktion $\exp(1-\cos(x))+0.05$ wurde um 0.05 Einheiten nach oben verschoben - damit sie die andere Bahn nicht überdeckt!

Die inhomogene Gleichung

Die allgemeine Lösung des Anfangswertproblems

$$\dot{u}(t) = a(t) \cdot u(t) \quad u(0) = u_0 \tag{1.63}$$

lautet also

$$u_g(t) = u_0 \exp\left(\int_{t_0}^{t} a(x)\,dx\right)$$

Das inhomogene Anfangswertproblem lautet

$$\dot{u}(t) = a(t) \cdot u(t) + s(t) \quad u(0) = u_0 \tag{1.64}$$

Theorem 1.2

$$\left.\begin{array}{l}\text{Sei } u_p(t) \text{ eine beliebige partikuläre Lösung von 1.64}\\ \\ u(t) \text{ sei eine beliebige Lösung von 1.64}\end{array}\right\} \Rightarrow (u - u_p) \text{ ist Lösung von 1.63}$$

Beweis:

$$\frac{d}{dt}(u - u_p) = \dot{u} - \dot{u}_p = a(t) \cdot u + s(t) - (a(t) \cdot u_p + s(t)) = a(t) \cdot (u - u_p)$$

$$\Rightarrow \boxed{u = u_g + u_p}$$

□

Wir brauchen also die allgemeine Lösung des homogenen Anfangswertproblems (AWP) und eine partikuläre Lösung des inhomogenen Anfangswertproblems! Man bekommt diese mit der Methode *Variation der Konstanten*:

$$u_p(t) = C(t) \exp\left(\int a(x)\,dx\right)$$

Die Konstante C wird so lange variiert, bis sie die inhomogene Differentialgleichung erfüllt:

$$\frac{d}{dt}\left[C(t) \exp\left(\int a(x)\,dx\right)\right] = a(t)\left[C(t) \exp\left(\int a(x)\,dx\right)\right] + s(t) \Rightarrow \dot{C} = s(t) \exp\left(-\int a(x)\,dx\right)$$

Mit $\boxed{C(t) = \int s(t) \exp\left(-\int a(x)\,dx\right) dt}$ haben wir C und damit u_p gefunden

Dies Ergebnis benutzen wir jetzt um mit *wxMaxima* die *short-time*-Näherung zu finden:

Zuerst lösen wir die Gleichung für V_x

(%i1) diffEq1:'diff(V_x,T)=-V_x^2; $$\frac{d}{dT}V_x = -{V_x}^2 \qquad \text{(diffEq1)}$$

(%i2) gSol1:ode2(diffEq1,V_x,T); $$\frac{1}{V_x} = T + \%c \qquad \text{(gSol1)}$$

(%i3) spSol1:ic1(gSol1,T=0,V_x=V_x0); $$\frac{1}{V_x} = \frac{T\,V_{x0}+1}{V_{x0}} \qquad \text{(spSol1)}$$

Jetzt haben wir die Lösung für $V_x(T)$

(%i4) define(V_x(T),rhs(linsolve (spSol1,V_x)[1])); $$V_x(T) := \frac{V_{x0}}{T\,V_{x0}+1} \qquad \text{(\%o4)}$$

Bei unserer zweiten inhomogenen linearen Dfglg ist $a := -V_x$, $s = -1$

(%i5) a(t):=-V_x(t); $$\mathrm{a}(t) := -V_x(t) \qquad \text{(\%o5)}$$

(%i6) assume(T>0,V_x0>0); $$[T>0, V_{x0}>0] \qquad \text{(\%o6)}$$

(%i7) define(d(t),exp(integrate(a(t),t))); $$\mathrm{d}(t) := \frac{1}{V_{x0}t+1} \qquad \text{(\%o7)}$$

Wir haben jetzt die allgemeine("general") Lösung der homogenen Gleichung:

(%i8) define(V_yg(T),V_y0*d(T)); $$V_{yg}(T) := \frac{V_{y0}}{T\,V_{x0}+1} \qquad \text{(\%o8)}$$

(%i9) define(b(t),exp(integrate(-a(t),t))); $$\mathrm{b}(t) := V_{x0}t+1 \qquad \text{(\%o9)}$$

Jetzt bestimmen wir unsere Konstante C

(%i10) define(C(t),integrate((-1)*b(t),t)); $$\mathrm{C}(t) := -\frac{V_{x0}t^2}{2} - t \qquad \text{(\%o10)}$$

Damit haben wir jetzt die partikuläre Lösung für V_y

(%i11) define(V_yp(t),ratsimp(C(t)*d(t))); $$V_{yp}(t) := -\frac{V_{x0}t^2+2t}{2V_{x0}t+2} \qquad \text{(\%o11)}$$

Die Summe der allgemeinen Lösung der homogenen und partikulären der inhomogenen Gleichung

(%i13) define(V_y(T),ratsimp(V_yg(T)+V_yp(T))); $$\boxed{V_y(T) := \frac{2V_{y0} - T^2\,V_{x0} - 2T}{2T\,V_{x0}+2}} \qquad \text{(\%o13)}$$

Obiges Ergebnis ist eine Bestätigung von 1.38 !