Zwei springende Bälle

Zwei Bälle springen übereinander auf dem festen Boden auf und ab. Bei den Zusammenstößen und beim Bodenaufprall gelten der Energie- und der Impulserhaltungssatz.

Alle Reibungsverluste, auch Luftreibung werden vernachlässigt.

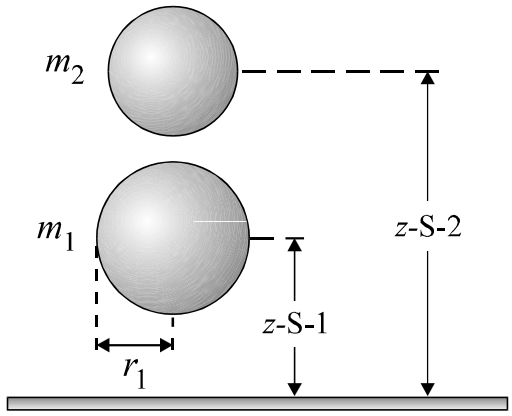

Abb. 1 Der untere springende Ball hat die Nr. 1; der obere, darüber springende Ball hat die Nr. 2.

Bei numerischen Berechnungen müssen die Anfangsbedingungen und Parameter folgende vier Bedingungen erfüllen:

$$z_{S1}(0) \geq 1\text{E}{-6}\ \text{m} \qquad z_{S1}(0) > r_1$$

$$z_{S2}(0) \geq 3\text{E}{-6}\ \text{m} \qquad z_{S2}(0) > z_{S1}(0) + r_1 + r_2$$

Falsche Eingaben werden von MECHANICUS automatisch gemeldet und abgelehnt.

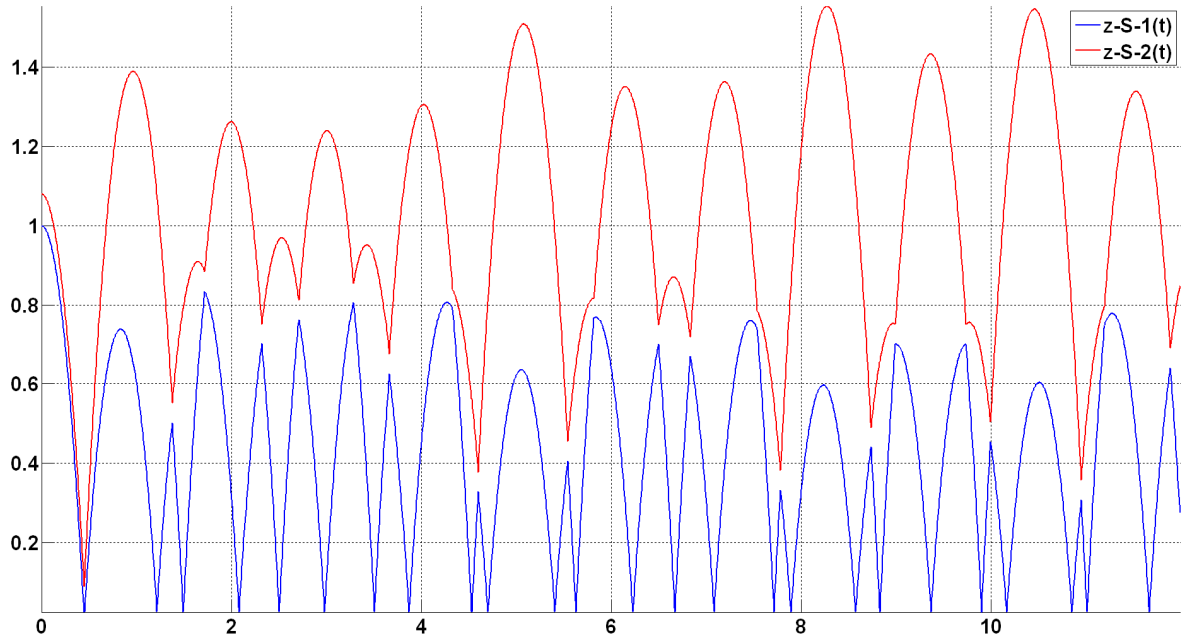

Abb. 2 Zwei Bälle springen mit zentralen, elastischen Stößen übereinander. Bei den Zusammenstößen der Bälle ist der Abstand der Kurven gleich dem Schwerpunktabstand $r_1 + r_2$ der Kugeln. Der untere Ball mit der Nr. 1 stößt auch elastisch auf den Boden. Die Anfangsbedingungen und Parameter lauten:

$$z_{S1}(0) = 1\ \text{m} \qquad \dot{z}_{S1}(0) = 0 \qquad z_{S2}(0) = 1{,}08\ \text{m} \qquad \dot{z}_{S2}(0) = 0$$

$$m_1 = 1\ \text{kg} \qquad r_1 = 0{,}025\ \text{m} \qquad m_2 = 0{,}85\ \text{kg} \qquad r_2 = 0{,}025\ \text{m}$$

Numerische Lösung der Dgln. mit Schaltfunktionen (Event-functions)

Die Dgln. werden am besten mit dem Verfahren ode45 gelöst.

Das folgende MatLab-Programm zeigt, wie die Bewegung der Bälle und vor allem die Stöße berechnet werden. Der grüne Text hinter dem %-Zeichen enthält Kommentare. Das Programm kann mit Copy und Paste in den MatLab-Editor übernommen werden.

function Springende_Baelle

```
%  Diese Funktion berechnet und zeichnet die Bewegung von zwei übereinander springenden Bällen. Jeder
%  Bodenaufprall und jeder Zusammenstoß ist zentral und elastisch.
global  r1  r2

%-------------------------------------------------------------------------------------------------------------------
%  Anfangsbedingungen  &  Parameter
%-------------------------------------------------------------------------------------------------------------------
tstart  =  0.0;                          %  Startzeit

z1  =  0.3;                               %  Anfangshöhe       von m1.
v1  =  0.0;                               %  Anfangsgeschw.   von m1.
z2  =  0.42;                              %  Anfangshöhe       von m2.
v2  =  0.0;                               %  Anfangsgeschw.   von m2.

y0  =  [ z1   v1   z2   v2 ];            %  Anfangsbedingungen im Vektor y0

m1  =  1.0;                               %  Masse von m1.
m2  =  0.9;                               %  Masse von m2.
r1   =  0.05;                             %  Radius von m1.
r2  =  r1 * ( m2 / m1 )^(1/3);            %  Radius von m2.
%-------------------------------------------------------------------------------------------------------------------


%  Die numerische Lösung der Dgl wird beendet, wenn die Zahl der Stöße oder wenn  tfinal  erreicht wird.
Stosszahl  =  100;                  %  Zahl der Stöße
tfinal       =   10;                  %  End-Zeit.
Delta_t  =  0.01;                    %  Ausgabeschrittweite des ode-Verfahrens.

%-------------------------------------------------------------------------------------------------------------------
options  =  odeset( 'RelTol' ,  1E-6,  'AbsTol',  1E-8,  ...
                          'Events' ,  @Eventfunktion_Springende_Baelle );
%-------------------------------------------------------------------------------------------------------------------

%  Sei  N = Stosszahl. Dann werden drei Größen wie folgt definiert:
%
%  T_hit :  T_hit  ist ein Vektor der Länge N. Er enthält die N Zeiten, zu denen ein Bodenaufprall oder ein
%           Zusammenstoß der beiden Bälle erfolgte.
%  Y_hit :  Y_hit  ist eine Nx4-Matrix mit den 2 Koordinaten und den 2 Geschwindigkeitten zu den N Zeiten
%           der Stöße.
%  i_hit   : i_hit  ist ein Vektor der Länge N. Seine Elemente sind beim Bodenaufprall gleich 1  und  beim
%           Zusammenstoß der beiden Bälle gleich  2  --  entsprechend der Festlegung in der unten
%           definierten Funktion  "Eventfunktion_Springende_Baelle".
T  =  tstart;
Y  =  y0;
T_hit  =  [ ];
Y_hit  =  [ ];
```

```
i_hit   =  [ ];

%-------------------------------------------------------------------------------------------------------------------
%  Jeder Durchlauf der for-Schleife berechnet die Bewegungen der beiden Bälle in einem Zeitintervall
%  zwischen zwei aufeinander folgenden Stößen. Die  for-Schleife  wird vorzeitig beendet, wenn im Laufe
%  ihres Durchgangs "tstart  >=  tfinal"  wird, wenn also die Endzeit  'tfinal'  erreicht wird, bevor die Zahl
%  der Stöße den Endwert  'Stosszahl'  erreicht.
%-------------------------------------------------------------------------------------------------------------------
for i = 1 : Stosszahl
   try
      %-------------------------------------------------------------------------------------------------------------
      %        [ T_local , Y_local , te , ye , ie ]  =  ...
      %              ode45( @Zwei_springende_Baelle_Dgl,  tstart : Delta_t : tfinal, y0, options );
      %
      %  löst die Dgl bis entweder die Endzeit  'tfinal'  erreicht wird oder bis eine der beiden in der
      %  event-function
      %
      %    [value, isterminal, direction ]  =  Eventfunktion_Springende_Baelle(T, Y )
      %
      %  definierten Funktionen
      %            Y(1) - r1       &       Y(3) - Y(1) - (r1 + r2)
      %  eine Nullstelle erreicht. ( Y(1) = z1  und  Y(3) = z2 ) Beim Bodenaufprall hat die 1-te Funktion eine
      %  Nullstelle, beim Zusammenstoß der beiden Bälle hat die 2-te Funktion eine Nullstelle.
      %
      %  Für jede der beiden Nullstellen geben die Variablen  'isterminal'  und  'direction'  an, ob beim Durch-
      %  laufen die Integration beendet werden muss und ob dabei die Richtung, in der die Nullstelle durch-
      %  laufen wird, von Bedeutung ist oder nicht. Mit  k=1,2  gilt im Einzeln:
      %
      %  Für  'isterminal( k ) = 1  wird die Integration beendet    , wenn die k-te Funktion eine Nullstelle erreicht.
      %  Für  'isterminal( k ) = 0  wird die Integration fortgesetzt, wenn die k-te Funktion eine Nullstelle erreicht.
      %
      %  Für  'direction( k ) = 0'    werden alle Nullstellen beachtet (the default).
      %  Für  'direction( k ) = +1'   werden nur die Nullstellen beachtet, die mit positiver Steigung durchlaufen
      %                               werden.
      %  Für  'direction( k ) = -1'   werden nur die Nullstellen beachtet, die mit negativer Steigung durchlaufen
      %                               werden.
      %
      %  Die letzten drei Ausgabewerte der ode-Funktion sind:
      %
      %  te  :  Zeit beim Durchlaufen der beachteten Nullstelle.
      %  ye :  4-dim. Lösungsvektor beim Durchlaufen der beachteten Nullstelle.
      %  ie  :  Es gilt:  'ie = 1'  bzw.  'ie=2' , wenn die 1-te bzw. die 2-te Funktion eine Nullstelle hat.
      %-------------------------------------------------------------------------------------------------------------

      [ T_local , Y_local , te , ye , ie ]  =  ...
                    ode45( @Zwei_springende_Baelle_Dgl ,  tstart : Delta_t : tfinal ,  y0 ,  options );

      %-------------------------------------------------------------------------------------------------------------
      %  Die Ausgaben von ode45 werden nacheinander zusammengesetzt.
      T  =  [ T ;  T_local( 2 : end      ) ];
      Y  =  [ Y ;  Y_local( 2 : end , : ) ];

      %  T_hit  und  Y_hit  enthalten die zusammengesetzten Zeiten und die zusammengesetzten Koordinaten,
      %  bei denen die numerische Rechnung infolge eines Stoßes, d. h. eines Nulldurchganges unterbrochen
      %  wurde.
      T_hit   =  [ T_hit ;  te ];
      Y_hit   =  [ Y_hit ;  ye ];
      i_hit    =  [ i_hit   ;  ie ];

      %  Neue Anfangsbedingungen für den nächsten Durchlauf der for-Schleife.
      tstart  =  T_local( end );
```

```
      if  ie  ==  1
        %  Bodenaufprall. Hier ist  ie = 1. Die Geschwindigkeit des unteren Balles Nr. 1 wird umgekehrt.
        y0(1)  =     Y_local( end , 1 );
        y0(2)  =  -  Y_local( end , 2 );
        y0(3)  =     Y_local( end , 3 );
        y0(4)  =     Y_local( end , 4 );
      else
        %  Zusammenstoß der beiden Bälle. Hier ist  ie = 2. Die Geschwindigkeiten nach dem Stoß werden mit
        %  den bekannten Gln. für den zentralen, elastischen Stoß berechnet.
        y0(1)  =  Y_local( end , 1 );
        y0(2)  =  1/(m1+m2) * (  (m1-m2)  * Y_local(end , 2)  +  2*m2 * Y_local(end , 4)  );
        y0(3)  =  Y_local( end , 3 );
        y0(4)  =  1/(m1+m2) * (  (m2-m1)  * Y_local(end , 4)  +  2*m1 * Y_local(end , 2)  );
      end

    catch
      %  Die  for-Schleife  wurde auf Grund eines Fehlers beendet  --  z. B. weil  ‚tstart  >=  tfinal'  ist. In diesem
      %  Fall ist  'tstart : Delta_t : tfinal'  kein gültiges Zeitintervall mehr, so dass die ode-function einen Fehler
      %  meldet. Der Fehler führt aber wegen des try-catch-Blockes nicht zum Absturz des Programms.
      %  Vielmehr wird das Programm nach dem Auftreten des Fehlers mit den Befehlen, die hinter  "catch"
      %  stehen, fortgesetzt.
      break                    %  Abbruch der  for-Schleife.

    end                        %  Ende des  "try-catch-Blocks".

end                            %  Ende der for-Schleife.
%-------------------------------------------------------------------------------------------------------------------

plot( T , Y(:, 1) , 'b' ,  T , Y(:, 3) , 'r' ,  'LineWidth', 2 );
axis  tight
legend( [ 'z1(t)' ; 'z2(t)' ] ,  'Location',  'NorthEast' );
xlabel('Zeit');
ylabel('Höhe');
title('Balltrajektorien');
grid  on


%-------------------------------------------------------------------------------------------------------------------
function  yPkt  =  Zwei_springende_Baelle_Dgl( t , y )
yPkt  =  zeros( 1 , 4 );
yPkt  =  [ y(2)
           - 9.81
           y(4)
           - 9.81 ] ;


% -----------------------------------------------------------------------------------------------------------------
function  [ value, isterminal, direction ] =  ...
                                  Eventfunktion_Springende_Baelle( T , Y )
global  r1  r2
value       =  [ Y(1) - r1 ;  Y(3) - Y(1) - (r1 + r2) ] ;      %  Höhe oder Entfernung  =  0.
isterminal  =  [ 1 ;  1 ] ;                                    %  Ende der Integration
direction   =  [ - 1 ;  -1 ] ;                                 %  Negative Richtung
```

Animationen

Die numerischen Lösungsverfahren der Dgln. geben *bei jedem Stoß* die Zeit, die zwei Koordinaten und die zwei Geschwindigkeiten, die beim Stoß vorliegen, aus. Daher ist die *Ausgabeschrittweite*, die eigentlich konstant sein sollte, bei Stößen (in der Regel) *verkleinert*, evtl. sogar *stark verkleinert*. Bei der Darstellung von Kurven ist die Ausgabe jedes Stoßes erfreulich. Schwankende Ausgabeschrittweiten stören bei Kurven überhaupt nicht.

Auf die Animation aber hat die Verkürzung der Ausgabeschrittweite einen ungünstigen Einfluss: Bei Stößen laufen *Animationen verlangsamt und stockend* ab. Wenn z. B. die untere Kugel ‚verfrüht' auf den Boden stößt, dann erfährt die Bewegung der oberen Kugel eine evtl. gut sichtbare Verzögerung.

Leider kann dieses Problem nicht befriedigend gelöst werden. Wenn die Ausgabeschrittweite mit zusätzlichem Programmieraufwand konstant gemacht wird, dann sind die Bodenaufpralle und Zusammenstöße in der Regel nicht in den berechneten Ausgabewerten enthalten. Das bedeutet z. B., dass ein fallender Ball bei der Animation nicht bis zum Boden kommt, sondern schon vorher umkehrt.

Erfreulicherweise ist das Problem umso kleiner, je schneller der Rechner Animationen abspielen kann, je kleiner also der Anwender – vor der numerischen Berechnung – die Schrittweite wählen kann.

Mechanicus versucht, die Animationen etwas zu verbessern, indem die vom Anwender gewählte Schrittweite vor den numerischen Lösungen der Dgln. automatisch durch 10 geteilt wird. Bei der Betrachtung von Kurven wird jede Ausgabe gelesen, bei der Animation aber wird automatisch nur jede zehnte Ausgabe gelesen und animiert.

Hinweise: **1)** Bei Sprunghöhen, die viel größer sind als die Ballradien, können die Bälle wegen ihrer relativen Winzigkeit bei der Animation nicht erkannt werden. **2)** Zusammenstöße können in kleinen Zeitabständen folgen, wenn z. B. die untere Kugel viel leichter ist als die obere Kugel ($m_1 << m_2$).

Literatur

Literatur ist mir nicht bekannt.