Управление процессом интегрирования

Интегрированные математические пакеты

Юдинцев В. В.

Кафедра теоретической механики. Самарский университет.

Остановка процесса интегрирования при выполнении условия

Определение момента падения на землю точки, движущейся по баллистической траектории

Уравнения движения точки в плоскости в однородном поле силы тяжести

In[691]:=

```
eq = {x''[t] == 0, y''[t] == -g };
```

Начальные условия

In[692]:=

```
ic = {x[0] == 0, y[0] == 0, x'[0] == V0 Cos[φ0], y'[0] == V0 Sin[φ0]};
```

Параметры

In[693]:=

```
p = {g → 9.81, V0 → 10, φ0 → 45 °};
```

In[694]:=

```
sol = NDSolve[{eq, ic} //. p, {x[t], y[t], x'[t], y'[t]}, {t, 0, 2}]
```

Out[694]=

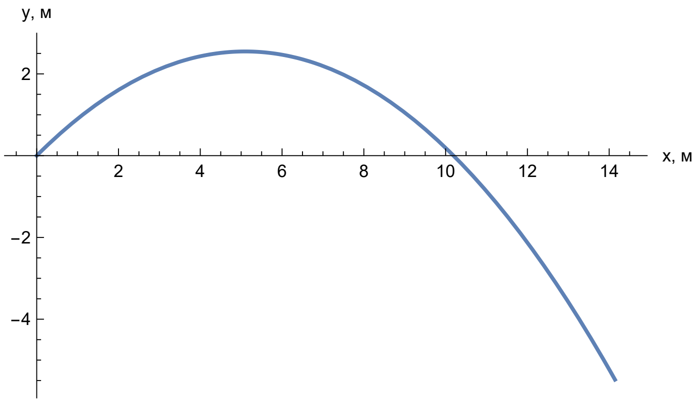
```
{{x[t] → InterpolatingFunction[ Domain: {{0., 2.}} Output: scalar ][t],
  y[t] → InterpolatingFunction[ Domain: {{0., 2.}} Output: scalar ][t],
  x′[t] → InterpolatingFunction[ Domain: {{0., 2.}} Output: scalar ][t],
  y′[t] → InterpolatingFunction[ Domain: {{0., 2.}} Output: scalar ][t]}}
```

Траектория точки

In[695]:=

```
ParametricPlot[{x[t], y[t]} /. sol, {t, 0, 2}, AxesLabel → {"x, м", "y, м"}]
```

Out[695]=



Траектория продолжилась и при y<0.

Как остановить процесс интегрирования при достижении нулевой высоты?

WhenEvent[условие, что делать]

или

WhenEvent[условие, {что делать,что делать,что делать,что делать}]

In[696]:=

```
sol2 = NDSolve[
  {
   eq,
   ic,
   WhenEvent[y[t] < 0, "StopIntegration"]
  } //. p,
  {x[t], y[t], x'[t], y'[t]}, {t, 0, 2}]
```

Out[696]=

```
{{x[t] → InterpolatingFunction[ Domain: {{0., 1.44}} Output: scalar ][t],
  y[t] → InterpolatingFunction[ Domain: {{0., 1.44}} Output: scalar ][t],
  x′[t] → InterpolatingFunction[ Domain: {{0., 1.44}} Output: scalar ][t],
  y′[t] → InterpolatingFunction[ Domain: {{0., 1.44}} Output: scalar ][t]}}
```

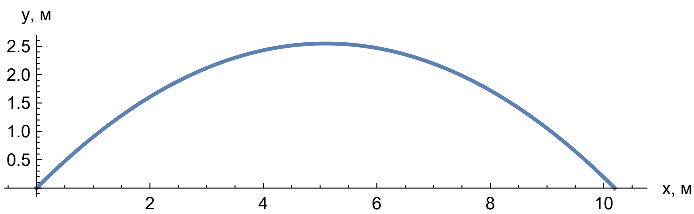
Процесс интегрирования остановился при t = 1.44 с, в момент достижения нулевой высоты.

Время остановки извлекается из решения

In[697]:=
```
sol2[[1, 1]]
```
Out[697]=
```
x[t] → InterpolatingFunction[ Domain: {{0., 1.44}} Output: scalar ][t]
```
In[698]:=
```
sol2[[1, 1, 2]]
```
Out[698]=
```
InterpolatingFunction[ Domain: {{0., 1.44}} Output: scalar ][t]
```
In[699]:=
```
sol2[[1, 1, 2, 0]]
```
Out[699]=
```
InterpolatingFunction[ Domain: {{0., 1.44}} Output: scalar ]
```
In[700]:=
```
sol2[[1, 1, 2, 0, 1]]
```
Out[700]=
```
{{0., 1.4416}}
```
In[701]:=
```
sol2[[1, 1, 2, 0, 1, 1, 2]]
```
Out[701]=
```
1.4416
```
In[702]:=
```
tk = sol2[[1, 1, 2, 0, 1, 1, 2]];
```

Траектория точки до момента падения на землю

In[703]:=
```
ParametricPlot[{x[t], y[t]} /. sol, {t, 0, tk}, AxesLabel → {"x, м", "y, м"}]
```
Out[703]=



Условие можно задать функцией, которая вычисляется только для аргумента-числа:

In[704]:=
```
h0Event[x_?NumericQ] := x < 0;
```

In[705]:=

```
sol2 = NDSolve[
   {
     eq,
     ic,
     WhenEvent[h0Event[y[t]], "StopIntegration"]
    } //. p,
   {x[t], y[t], x'[t], y'[t]}, {t, 0, 2}];
```

In[706]:=

Изменение динамических переменных

Отскок от земли

In[707]:=

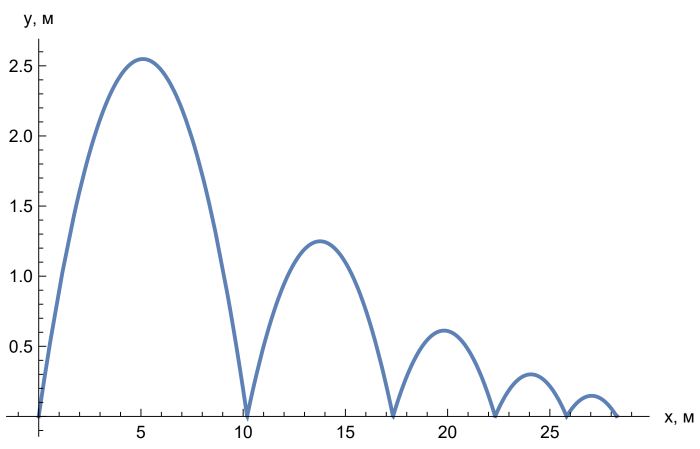
```
h0Event[x_?NumericQ] := x < 0;
```

При достижении нулевой высоты (ударе о землю) меняем направление вертикальной скорости на противоположное с коэффициентом меньше 1, т.е. моделируем мгновенный отскок с потерей кинетической энергии

In[708]:=

```
sol = NDSolve[
   {
     eq,
     ic,
     WhenEvent[h0Event[y[t]], y'[t] → -0.7 y'[t]]
    } //. p,
   {x[t], y[t], x'[t], y'[t]}, {t, 0, 4}];
ParametricPlot[{x[t], y[t]} /. sol, {t, 0, 4},
 AspectRatio → 1 / GoldenRatio, AxesLabel → {"x, м", "y, м"}]
```

Out[709]=



Сохраняем координаты x точек падения

Введем в уравнение еще одну переменную c[t], значение которой будет увеличивать на 1 при регистрации события. Эта переменная дискретная и будет изменяться скачкообразно (увеличиваться на 1) при наступлении события y<0. Переменная c[t] включается в список динамических переменных функции NDSolve, а также при помощи параметра **DiscreteVariables** указывается, что эта переменная является дискретной.

In[710]:=

```
h0Event[q_?NumericQ] := q < 0;
```

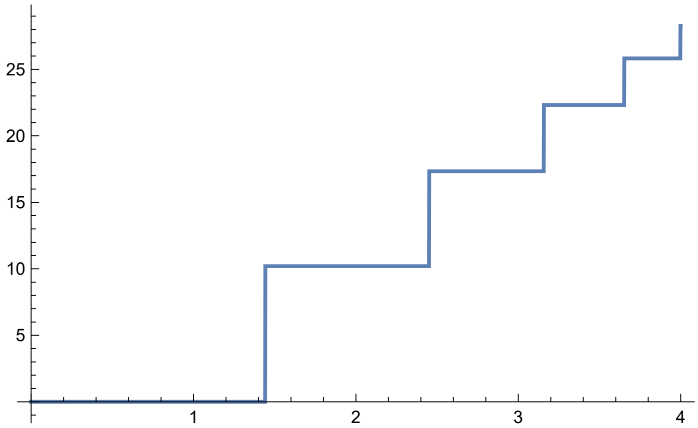
In[711]:=

```
sol = NDSolve[
   {
    eq,
    ic, c[0] == 0,
    WhenEvent[h0Event[y[t]], {y'[t] → -0.7 y'[t], c[t] → x[t]}]
   } //. p,
   {x[t], y[t], x'[t], y'[t], c[t]}, {t, 0, 10}, DiscreteVariables → c[t]];
```

In[712]:=

```
Plot[c[t] /. sol, {t, 0, 4}]
```

Out[712]=



Функция c[t] будет скачкообразно изменяться до значения координаты x точки падения тела.

Включение и выключение силы

Предположим, что тело падает на деформируемую поверхность **и при y < 0** на тело начинает действовать “выталкивающая” сила упругости, которая возникает при деформации поверхности. Предположим также, что эта сила пропорциональная глубине δ = -y и скорости деформации dδ/dt и направлена вверх:
Fn = k1*δ + k2*(dδ/dt) пока y<0.

Уравнения движения

В уравнение для y добавим силу Fn, которую умножим на дискретную переменную c[t], принимающую значения 1 или 0 (1 если y<0).

In[713]:=

```
eq = {x''[t] == 0, y''[t] == -g + c[t] * Fn / m};
```

Начальные условия

In[714]:=

```
ic = {x[0] == 0, y[0] == 0, x'[0] == V0 Cos[φ0], y'[0] == V0 Sin[φ0]};
```

Параметры

К списку параметров добавим массу, выражение для силы Fn, а также коэффициенты жесткости и демпфирования поверхности:

In[715]:=

```
p = {g → 9.81, V0 → 10, φ0 → 45 °, k1 → 300, k2 → 10, m → 1, Fn → k1 (-y[t]) + k2 (-y'[t])};
```

Для функции NDSolve укажем два события y<0 и y>0, при регистрации которых будет изменять значение дискретной переменной c[t] (признак действия силы Fn):

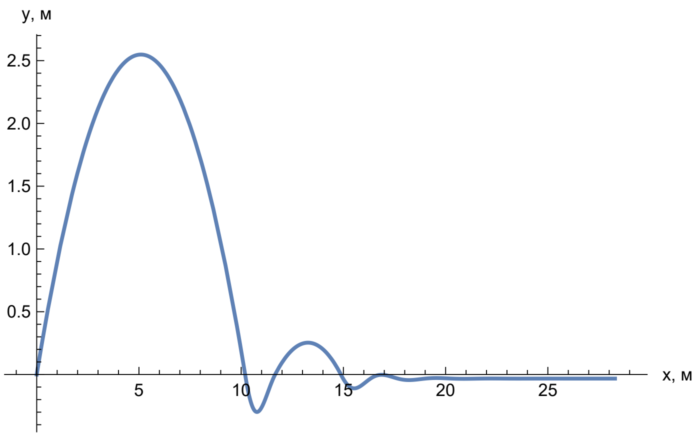
In[716]:=

```
sol = NDSolve[
   {
    eq,
    ic, c[0] == 0,
    WhenEvent[y[t] < 0, {c[t] → 1}],
    WhenEvent[y[t] > 0, {c[t] → 0}]
   } //. p,
   {x[t], y[t], x'[t], y'[t], c[t]}, {t, 0, 10}, DiscreteVariables → c[t]];
```

In[717]:=

```
ParametricPlot[{x[t], y[t]} /. sol, {t, 0, 4},
 AspectRatio → 1 / GoldenRatio, AxesLabel → {"x, м", "y, м"}]
```

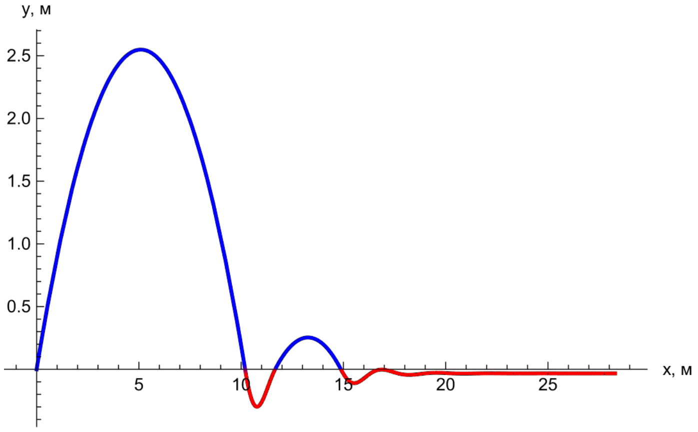
Out[717]=



In[718]:=

```
ParametricPlot[{x[t], y[t]} /. sol, {t, 0, 4},
 AspectRatio → 1 / GoldenRatio, AxesLabel → {"x, м", "y, м"},
 ColorFunction → Function[{x, y}, If[y < 0, Red, Blue]], ColorFunctionScaling → False]
```

Out[718]=



Функции **Sow** и **Reap**

Функции **Sow** (“засевать”) и **Reap** (“собирать урожай”) используются для сбора результатов вычислений.
Далее **Sow** используется для сохранения значения x[t] при наступлении события y[t]<0 после изменения знака вертикальной скорости. Чтобы результаты “посева” используется **Reap**, аргументом которой является выражение, в котором происходит “посев”. Функция Reap возвращает результат работы функции NDSlove и результаты “посева”, т.е. значения x[t] в момент наступления событий y[t]<0.

Уравнения

In[719]:=

```
eq = {x''[t] == 0, y''[t] == -g};
```

Начальные условия

In[720]:=

```
ic = {x[0] == 0, y[0] == 0, x'[0] == V0 Cos[φ0], y'[0] == V0 Sin[φ0]};
```

Параметры

In[721]:=

```
p = {g → 9.81, V0 → 10, φ0 → 45 °};
```

In[722]:=

```
sol = Reap[NDSolve[
   {
     eq,
     ic, c[0] == 0,
     WhenEvent[h0Event[y[t]], {y'[t] → -0.7 y'[t], Sow[x[t]]}]
    } //. p,
   {x[t], y[t], x'[t], y'[t]}, {t, 0, 4}]]
```

Out[722]=

```
{{{x[t] → InterpolatingFunction[ Domain: {{0., 4.}} Output: scalar ][t],
   y[t] → InterpolatingFunction[ Domain: {{0., 4.}} Output: scalar ][t],
   x′[t] → InterpolatingFunction[ Domain: {{0., 4.}} Output: scalar ][t],
   y′[t] → InterpolatingFunction[ Domain: {{0., 4.}} Output: scalar ][t]}},
 {{10.1937, 17.3293, 22.3242, 25.8206, 28.2681}}}
```

Решение дифференциального уравнения

In[723]:=

```
sol[[1]]
```

Out[723]=

```
{{x[t] → InterpolatingFunction[ Domain: {{0., 4.}} Output: scalar ][t],
  y[t] → InterpolatingFunction[ Domain: {{0., 4.}} Output: scalar ][t],
  x′[t] → InterpolatingFunction[ Domain: {{0., 4.}} Output: scalar ][t],
  y′[t] → InterpolatingFunction[ Domain: {{0., 4.}} Output: scalar ][t]}}
```

Сохраненные функцией **Sow** результаты

In[724]:=

```
sol[[2]]
```

Out[724]=

```
{{10.1937, 17.3293, 22.3242, 25.8206, 28.2681}}
```

Сечения Пуанкаре

В теории динамических систем, разделе математики, отображение Пуанкаре (также отображение последования, отображение первого возвращения) — это проекция

некоторой площадки в фазовом пространстве на себя (или на другую площадку) вдоль траекторий (фазовых кривых) системы.
https://ru.ruwiki.ru/wiki/Отображение_Пуанкаре

Уравнение Дуффинга

In[725]:=

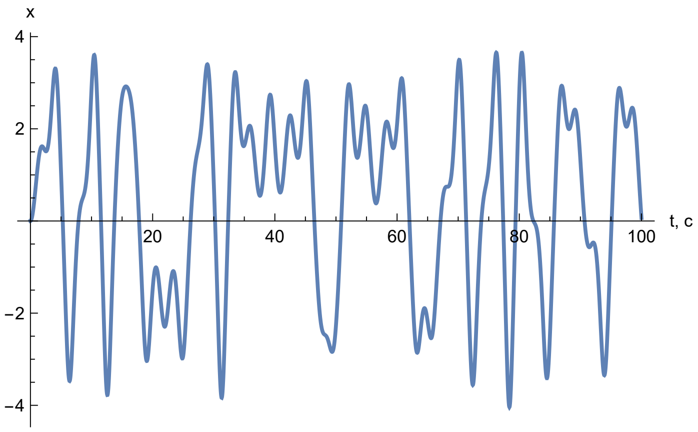
```
eq = x''[t] + δ x'[t] + α x[t] + β x[t]^3 == γ Cos[ω t];
p = {α → -1, β → 0.25, δ → 0.1, γ → 2.5, ω → 2};

sol = NDSolve[{eq, x[0] == 0, x'[0] == 0} /. p, {x[t], x'[t]}, {t, 0, 100}];
```

In[728]:=

```
Plot[x[t] /. sol, {t, 0, 100}, AxesLabel → {"t, c", "x"}]
```

Out[728]=

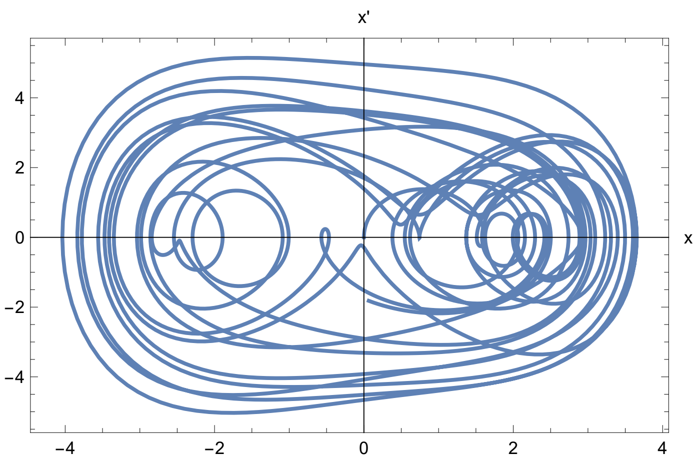


Фазовый портрет

In[729]:=

```
ParametricPlot[{x[t], x'[t]} /. sol, {t, 0, 100},
 AspectRatio → 1 / GoldenRatio, Frame → True, AxesLabel → {"x", "x'"}]
```

Out[729]=



Сохраним пары координат точки в фазовом пространстве (x[t], x’[t]) в моменты времени, когда время кратно 2 $\frac{\pi}{\omega}$ и покажем эти точки на графике.

```
WhenEvent[Mod[t, 2 π/ω], Sow[{x[t], x'[t]}]]
```

In[730]:=

```
solp = Reap[
   NDSolve[{eq, x[0] == 0, x'[0] == 0, WhenEvent[Mod[t, 2 π/ω], Sow[{x[t], x'[t]}]]} /. p,
    {x[t], x'[t]}, {t, 0, 20000}]];
```

In[731]:=

```
ListPlot[solp[[2, 1]], Frame → True, AxesLabel → {"x", "x'"}]
```

Out[731]=