**SciLab. Функции.**
**Задача о теле, брошенном под углом к горизонту**

## Inline-функции

Пусть движение тела по оси **Oy** описывает изменение высоты над поверхностью земли, а по оси **Ox** – дальность полета. Соответствующие функции имеют вид:

$$h(t)=h_0 + v_0 \cdot \sin(\alpha) \cdot t - gt^2/2$$

$$s(t)=s_0 + v_0 \cdot \cos(\alpha) \cdot t$$

где $v_0$ – модуль вектора начальной скорости, $h_0$ и $s_0$ – начальная высота и смещение от точек отсчета по вертикали и горизонтали, соответственно, $\alpha$ – угол между вектором начальной скорости и горизонтом.

Эти функции довольно просты, поэтому можно, конечно, вводить их в виде отдельных выражений всякий раз, когда это потребуется. Однако, мы пойдем другим путем и запишем их в виде так называемых inline-функций, т.е. функций, описываемых в одну команду.

Пусть тело сброшено с высоты 2 км. Вот как можно узнать, на какой высоте оно окажется спустя 5, 10, 15 и 20 секунд.

```
-->deff('[y]=h(t)','y=h0+v0*sin(alpha)*t-g*t^2/2');
-->deff('[x]=s(t)','x=s0+v0*cos(alpha)*t');
-->h0=2000;alpha=(-90*%pi/180);v0=0;g=9.81;
-->t=[5,10,15,20];
-->h(t)
 ans =
  1877.375 1509.5 896.375 38.
```

Значение угла ($-90^o$) необходимо перевести в радианы: *SciLab*, как и многие системы, использует именно их. Если вы хотите работать только с градусами, то можете переопределить *h(t)* и *s(t)* следующим образом:

```
-->deff('[y]=h(t)','y=h0+v0*sin(alpha*%pi/180)*t-g*t^2/2');
-->deff('[x]=s(t)','x=s0+v0*cos(alpha*%pi/180)*t');
```

Однако далее мы будем работать с радианами.

Подобным образом можно описать любые некусочно заданные функции. Общий вид таков:

```
deff('[<результат> = <имя функции>(<входные переменные>)]',
                              '<мат.описание функции>','ключ')
```

Ключ – необязательный параметр, принимающий значения **c** или **n**. В первом случае функция будет скомпилирована, что повышает быстродействие, а во втором – нет; по умолчанию действует ключ **c**. Такие inline-функции годятся не только для простых случаев. Математическое описание функции, а также результат, могут представлять собой

матрицу и содержать многошаговые вычисления. Возьмем, например, функцию **Qroots()**, решающую квадратные уравнения:

```
-->deff('[x1, x2]=Qroots(Qa,Qb,Qc)',['D=Qb^2-4*Qa*Qc';'x1=(-
      Qb+sqrt(D))/(2*Qa)';'x2=(-Qb-sqrt(D))/(2*Qa)']);
```

Опробуем ее на нашем примере. Пусть начальная высота равна нулю, а скорость равна 20 м/с и направлена вверх ($90^o$). Определим, когда тело окажется на высоте 20 м. Если ось **Oy** направлена вертикально вверх, то уравнение будет иметь вид:

$$20 - 20t + 9{,}81\ t^2/2 = 0$$

Найдем его корни при помощи **Qroots**:

```
-->[h1,h2] = Qroots(9.81/2,-20,20)
 h2 =
    1.7577156
 h1 =
    2.3197563
```

Решим уравнение с другими коэффициентами:

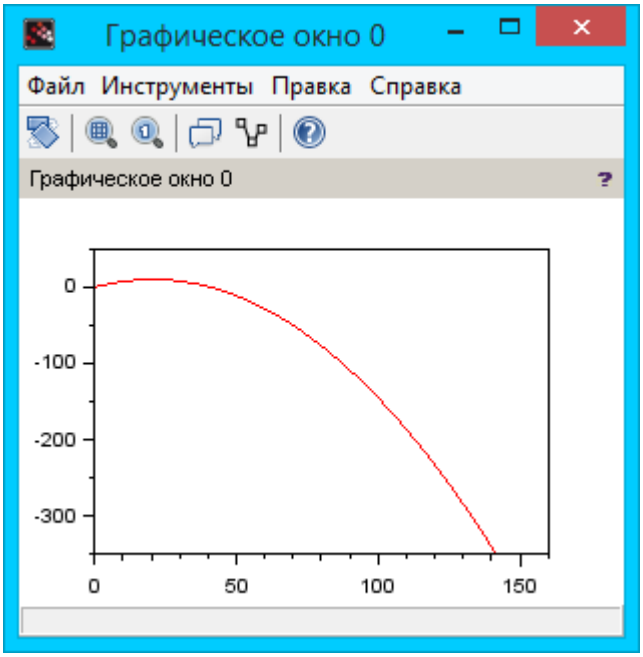
```
-->[q1,q2]=Qroots(1,2,3)
q2 =
  - 1. - 1.4142136i
q1 =
  - 1. + 1.4142136i
```

Как видно из этого примера, в тексте описания функции можно создавать внутренние переменные. И кроме того, нам не пришлось обрабатывать случай отрицательного дискриминанта, поскольку *SciLab* понимает мнимые числа и умеет работать с ними. Впрочем, он также умеет решать алгебраические и трансцендентные уравнения.

Давайте построим траекторию полета тела в течение первых 10 секунд при начальной скорости 20 м/с и угле $45^o$, считая, что $h_0=s_0=0$. Для этого нам необходимо создать вектор значений времени **(t)**, а затем построить график, где по оси **Ox** будет отложена **s(t)**, а по оси **Oy** – **h(t)**:

```
-->h0=0;s0=0;alpha=(45*%pi/180);v0=20;g=9.81;
-->t=0:0.1:10;
-->plot(s(t),h(t),'r-')
```

В результате мы получили траекторию полета тела, которая представляет собой график функции **h(s)**, заданной параметрически. Добавка **r-** в вызове **plot()** - это специальный параметр, регулирующий оформление (см. врезку Форматируем график на последней странице).



Траектория движения тела, брошенного под углом к горизонту и... ушедшего в недра земли.

Итак, наше тело ушло в недра земли. Чтобы этого не случилось, мы должны грамотно задавать диапазон расчета, т.е. определить, где высота становится равной нулю. Для этого пока воспользуемся встроенными средствами.

Функция **fsolve** позволяет решать линейные и нелинейные уравнения, а также их системы, заданные функциями, то есть находить нули функций. Поскольку ищется численное, а не аналитическое решение, то в результате мы получим одно число. Однако какое решение мы найдем, зависит от значения начального приближения. Так происходит потому, что, как правило, методы численного решения алгебраических уравнений позволяют найти ближайший к указанной точке корень.

Общий вид вызова функции таков:

```
eqAns = fsolve(vInit,F,Fj,e)
```

где **eqAns** – это переменная или вектор решений, **vInit** – начальное значение или вектор начальных значений, **F** – функция или список функций, представляющих собой левые части уравнений, **Fj** – это производная или список производных функций **F, e** – точность. Последние два параметра необязательны.

Чтобы корректно построить траекторию, нам необходимо найти нетривиальный положительный корень уравнения **h(t) = 0**. Зададим начальное приближение, равное **5.**

```
-->t_end=fsolve(5,h)
 t_end =
   2.8832081
-->t=0:0.01:t_end;
-->plot(s(t),h(t),'r-')
```

Выполните эти команды самостоятельно и посмотрите результат.

**Полиномы**

Если внимательно посмотреть на **h(t)**, то можно заметить, что она представляет собой полином второй степени относительно **t**. В *SciLab* встроено достаточно инструментов для работы с полиномами – в частности, есть функция **root**, возвращающая их корни.

Перед использованием полином необходимо определить. Делается это при помощи функции **poly**. Ее первый аргумент – это вектор-строка коэффициентов или корней полинома (смысл определяется флагом), второй – символьная переменная, а третий – флаг, принимающий значения **roots** или **coeff**. Например, пусть нам известно, что начальная высота полета равна 10 м, начальная скорость – 10 м/с и угол равен $30^o$. Требуется записать закон изменения высоты со временем.

```
-->hCoeff = [10, 10*sin(30*%pi/180),-9.81/2];
-->h=poly(hCoeff,'t','coeff')
 h=
 10 + 5t - 4.905t^2
```

Здесь мы указали флаг **coeff**, сообщающий, что первый параметр – это вектор коэффициентов полинома. А теперь определим, сколько секунд тело уже находится в полете и когда оно упадет на землю.

```
-->roots(h)
 ans =
  - 1.006401
    2.025769
```

То есть тело было брошено примерно секунду назад и упадет примерно через две секунды.

Теперь попробуем пойти другим путем. Известно, что тело находится в полете 3 секунды и через 5 секунд упадет на землю. Определить закон изменения высоты.

```
-->hRoots = [-3,5];
-->h=poly(hRoots,'t','roots')*(-9.81)/2
 h=
    73.575 + 9.81t - 4.905t^2
```

Таким образом, в настоящий момент времени тело находится на высоте 73,5 метра и имеет вертикальную скорость, равную 9,81 м/с. В данном случае флаг **roots** (используется по умолчанию) означает, что первый параметр – это корни полинома, который следует сформировать.

Естественно, работа с полиномами не ограничивается этими двумя функциями, их насчитывается более 35. Кроме того, с полиномами можно работать так же, как с обычными переменными, то есть вычитать, умножать, возводить в степень и т.д.

Например, определим два полинома, а затем выполним с ними все четыре арифметических действия, а также возведем один из них в квадрат.

```
 -->p1=poly([27,0,0,-8],'s','coeff')
 p1 =
    27 - 8s^3
-->p2=poly([9,6,4],'s','coeff')
 p2 =
    9 + 6s + 4s^2
-->p_add=p1+p2
 p_add =
    36 + 6s + 4s^2 - 8s^3
-->p_decr=p1-p2
 p_decr =
    18 - 6s - 4s^2 - 8s^3
-->p_div=p1/p2
 p_div =
    3 - 2s
    ------
      1
-->p_mult=p1*p2
 p_mult =
   243 + 162s + 108s^2 - 72s^3 - 48s^4 - 32s^5
-->p_degr=p2^2
 p_degr =
   81 + 108s + 108s^2 + 48s^3 + 16s^4
```

**Внешние функции**

У *SciLab* (как и *MatLab*) есть встроенный язык программирования. На самом деле, все функции, которые мы использовали, также входят в этот язык, наряду с обычными конструкциями (условие, цикл) и возможностями использования графики. Следовательно, программист, работая в *SciLab*, может не отвлекаться на реализацию численных методов, а использовать уже готовые наработки.

```
1  //функция для расчета дальности полета тела
2  //брошенного под углом к горизонту.
3  //Входные параметры:
4  //v0 - начальная скорость, alpha - угол в градусах
5  //h0 - начальная высота,
6  function [s] = FlyRange(h0,v0,alpha)
7    g = 9.81;
8    RadAngle = alpha*%pi/ 180
9
10   init_t = 0;
11   t=0;i=0;
12
13   while t<=0 do
14     t=init_t+10*i;
15     deff('x=h(t)','x=h0+v0*sin(RadAngle)*t - g*t^2/2');
16     t=fsolve(init_t,h);
17     i=i+1;
18   end
19
20   s = v0*cos(RadAngle)*t
21 endfunction
```

Встроенный редактор с текстом нашей функции.

Как и любая среда разработки, *SciLab* обладает текстовым редактором с подсветкой синтаксиса.

При помощи текстового редактора мы сможем создать более элегантное решение для расчета дальности полета тела, брошенного под углом к горизонту. А именно, мы напишем внешнюю функцию, которую можно будет подгружать во время работы и пользоваться ею так же, как и встроенными. Именно так и выполнены все расширения и многие встроенные функции системы *SciLab*.

Вернемся к нашей задаче. Полный текст функции приведен на рисунке. Первые строки, начинающиеся с двойного слэша (//) – это комментарии. Затем идет заголовок функции:

```
function [<список выходных параметров>] = <Имя Функции>
                                (<список входных параметров>)
 <тело функции>
endfunction
```

В дальнейшем именно **Имя функции** и будет использоваться в окне *SciLab*. Все остальное является локальным содержимым и на рабочую среду оказывать влияния не будет. Следует отметить, что если при определении функции не указаны выходные параметры, то есть функция ничего не возвращает, то она будет являться процедурой. Как правило, в виде процедур оформляются такие участки программы, как отрисовка графика или картинки, ввод и проверка данных и т.д.

В следующих двух строках мы определяем значение ускорения свободного падения и переводим угол из градусов в радианы. Далее, поскольку мы будем использовать функцию **fsolve**, задается начальное приближение и устанавливается значение времени, равное нулю. Как уже говорилось ранее, **fsolve** ищет решение, ближайшее к начальному приближению, поэтому его необходимо задать так, чтобы найденное время было больше нуля. Для этого мы увеличиваем начальное приближение (переменная **Init_t**) на **10** в цикле до тех пор, пока найденное решение (переменная **t**) не станет положительным. Таким образом мы сможем найти момент, когда тело упадет на землю. А затем, по формуле для равномерного движения, мы вычисляем, какое расстояние пройдет тело за это время вдоль горизонтальной оси.

Результаты вычисления заносятся в переменные, являющиеся выходными параметрами. В нашем случае он один – **s**. На этом выполнение функции заканчивается, и результат передается на выход. Теперь нам осталось записать функцию в файл **(Ctrl+Shift+S),** сохранив файл в своем домашнем каталоге под именем **flyrange.sci**.

В следующий раз, когда нам понадобится эта функция, мы сможем подключить ее при помощи процедуры **getf**, которая принимает полный путь и название файла функции. Предположим, что вы сохранили файл в своем домашнем каталоге. Тогда команда будет такой: `getf h:/firstuser/flyrange.sci`

После ее выполнения вы сможете использовать вашу собственную функцию в вычислениях. Кроме того, функцию можно загрузить в *SciLab* непосредственно из редактора. После этого весь исходный код будет передан в *SciLab* и выполнен. Если вы создали в редакторе не функцию, а просто последовательность команд, она будет выполнена, а результат – выведен на экран.

**Форматируем график**

Опции форматирования линии двумерного графика в *SciLab* разбиты на три категории: цвет линии, тип линии, маркер значения. Они указываются в функции **plot** в кавычках в любом порядке после массивов с данными для построения. Ниже приведена таблица с описанием каждого из параметров.

| Цвет линии | | Тип линии | | Маркер | |
|---|---|---|---|---|---|
| **Опция** | **Описание** | **Опция** | **Описание** | **Опция** | **Описание** |
| r | красный | - | Сплош-ная | + | Знак плюс |
| g | зеленый | -- | Пунктир-ная | o | Круг |
| b | синий | : | Точками | * | Звездочка |
| c | голубой | -. | Пунктир с точкой | . | Точка |
| m | пурпур-ный | | | x | Крест |
| y | желтый | | | s | Квадрат |
| k | черный | | | d | Ромб |
| w | белый | | | ^ | Треугольник острием вверх |
| | | | | v | Треугольник острием вниз |
| | | | | > | Треугольник острием вправо |
| | | | | < | Треугольник острием влево |
| | | | | pentagram | Пятиконечная звезда |
| | | | | none | Нет (по умол-чанию) |