5.3. Функциите на MATLAB за числено интегриране на системи ОДУ – Solvers

MATLAB ни предоставя седем основни функции за числено интегриране на системи ОДУ. В тях са реализирани различни алгоритми, включително и такива за решаване на така наречените *твърди* (Stiff) диференциални уравнения. *Твърди* се наричат тези уравнения, в решението на които се наслагват много бавно и бързо изменящи се компоненти. При решаването им с помощта на класическите методи се налага многократно намаляване стъпката на интегриране, което забавя значително изпълнението, а натрупването на грешки от закръгленията често прави резултатите неадекватни на изследвания процес. За интегрирането на такива уравнения са разработени специални алгоритми.

Ще си позволим по нататък да наричаме функциите за интегриране на ОДУ с английския термин solver-и. Solver-ите в MATLAB се делят на две основни групи:

- *За нетвърди диференциални уравнения* (Nonstiff ODE) **`ode45, ode23, ode113`**;
- *За твърди диференциални уравнения* (Stiff ODE) **`ode15s, ode23s, ode23t, ode23tb`**.

5.3.1. Кратка характеристика и съвети за избор на подходящи solver-и

☐ *За нетвърди диференциални уравнения*

ode45 – базира се на явния метод на Рунге-Кута от 4-ти и 5-ти ред. Това е едностъпков алгоритъм – за пресмятане на $y(t_n)$ е необходимо да знаем решението само в една предшестваща точка $y(t_{n-1})$. Този solver е най-често използвания при повечето от задачите. Затова при пристъпване към решаване на задача с неизвестно поведение се препоръчва да започваме опитите си с него. Осигурява от ниска до средна точност на решението.

ode23 – базира се на явния метод на Рунге-Кута от 2-ри и 3-ти ред. Едностъпков алгоритъм. Препоръчва се за решаване на не особено твърди диференциални уравнения при невисоки изисквания към точността, когато бързодействието е от решаващо значение за потребителя.

ode113 – прилага метода на Адамс-Бешфорт-Мултон с променлив ред. Методът е многостъпков – за пресмятане на текущото решение са нужни решенията от няколко предшестващи точки. Може да бъде по-ефективен от **ode45**, особено при високи изисквания към точността на пресмятанията и обемисти изрази в десните части на диференциалните уравнения.

☐ *За твърди диференциални уравнения*

ode15s – многостъпков адаптивен метод, базиращ се на метода на Gear. Ако считате вашата задача за твърда или опитите да я решите с **ode45** не са дали очакваните резултати, пробвайте с **ode15s**. Осигурява от ниска до средна точност.

ode23s – използва модифицирания метод на Розенброк от втори ред. Това е едностъпков метод и може да се окаже по-бърз от **ode15s** при невисоки изисквания към точността.

ode23t – реализира метод на трапеците с интерполация. Използва се при умерено твърди задачи. Дава добри резултати при решаване на уравнения, описващи почти хармонични трептения. Осигурява средна точност.

ode23tb – прилага неявната формула на Рунге-Кута. При невисоки изисквания към точността на решението може да се окаже по-ефективен от **ode15s**.

Забележка: Всички solver-и, с изключение на **ode23s,** могат да решават и системи ОДУ от неявен вид $M(t, y)\, y' = F(t, y)$, т.е. неразрешени спрямо производните. Тук M се нарича *масова* матрица (Mass matrix). Функцията **ode23s** може да работи само при $M = \text{const}$.

5.3.2. Базов синтаксис на solver-ите

Ако изключим някои специални опции, всички solver-и се викат по един и същи начин, т.е. имат един и същ базов синтаксис. Това облекчава използването им при решаване на непозната задача, за която не е очевидно кой от тях е най-подходящия. Достатъчно е да сменяме само името на solver-а и да го стартираме, докато открием най-добрия за случая.

Минималният брой на входящите аргументи, с които се вика всеки един от седемте solver-а, е три:

```
[t,y] = solver(@odefun,tspan,y0);
```

Тук **solver** е името на един от solver-ите.

Задължителните *входящи аргументи* са:

odefun – името на файл-функцията, пресмятаща десните части на канонизираната система диференциални уравнения;

tspan – вектор, задаващ интервала на интегриране. Първият елемент **tspan(1)** се приема за *началната* стойност на аргумента. В динамичните задачи това е момента t_0, за който са зададени *началните условия*. Интегрирането се извършва в интервала от **tspan(1)** до **tspan(end)**.

Ако **tspan** е вектор с два елемента **[t0 tend]**, solver-ът връща решението, пресметнато за *всяка* стъпка на интегриране.

Ако **tspan** е вектор с повече от два елемента, solver-ът връща решението *само* за стойности на аргумента, съответстващи на елементите на вектора **tspan**. Елементите на **tspan** трябва да бъдат подредени във възходящ или низходящ ред.

Задаването на **tspan** с повече от два елемента оказва слабо влияние върху ефективността на пресмятанията, но при интегриране на големи системи ОДУ оказва влияние върху управлението на паметта.

y0 – вектор с началните условия на задачата [y10, y20, …, yn0].

Изходящите аргументи са:

t – вектор-стълб със стойностите на аргумента (обикновено времето), за които се съхранява решението в матрицата **y**;

y – матрица с решението на системата. Всеки стълб на тази матрица представлява решението за съответната функция на каноничната система: В първия стълб – $y_1 = y_1(t)$, във втория стълб – $y_2 = y_2(t)$ и т.н. Очевидно всеки ред на матрицата **y** представлява стойностите на функциите $y_1, y_2, \ldots, y_n$ за стойност на аргумента, равна на съответния елемент на вектора **t**.

При необходимост от промяна на някои подразбиращи се *свойства* на процеса на интегриране, се подава четвърти аргумент:

```
[t,y] = solver(@odefun,tspan,y0,options);
```

Тук аргументът **options** не е ключова дума, а име на запис, в който се съхраняват незадължителни параметри, които променят избрани от потребителя свойства на изчислителния процес (виж раздел 5.4).

Много често се използва и следната форма на извикване:

```
[t,y] = solver(@odefun,tspan,y0,options,p1,p2,…);
```

Тук **p1, p2,** … са физически *параметри*, които solver-ът подава на функцията **odefun** и на други функции, указани в аргумента **options**. В този случай файл-функцията **odefun** трябва да има следния вид:

```
function yt = odefun(t,y,p1,p2,…)
```

Важно: Ако нямате намерение да променяте подразбиращите се свойства на изчислителния процес, но желаете да подадете параметри към функцията **odefun** чрез списъка на аргументите, вместо аргумента **options** използвайте символа за празен масив:

```
[t,y] = solver(@odefun,tspan,y0,[],p1,p2,…);
```

Това са възможните вариации с входящите аргументи. Възможни са обаче и *вариации с изходящите аргументи* .

Броят на изходящите аргументи при извикване на кой да е от solver-ите може да бъде 0, 1, 2 или 5. Нека разгледаме първите два варианта:

☐ При отсъствие на изходящи аргументи се начертават автоматично в един общ прозорец графиките на *всички* функции $y_1, y_2, \ldots, y_n$ с помощта на системната функция **odeplot**. Отделните точки от решението се изобразяват с кръгчета, свързани с плавни линии. Макар да изглежда твърде привлекателен, този подход не е за препоръчване, тъй като **odeplot** използва един и същ мащаб за всички функции, при което някои от графиките се "сплескват" върху абсцисната ос!

☐ При извикване на някой от solver-ите с един единствен изходящ аргумент

```
sol = solver(@odefun,[a,b],y0,…)
```

на изхода получаваме *записа* sol, който има *две полета*:

sol.x – вектор-ред със стойностите на аргумента за отделните стъпки на интегриране. Името на полето е винаги x, независимо от това, какво е действителното име на аргумента в решаваната задача;

sol.y – матрица с n реда, в които се съхраняват стойностите на функциите $y_1, y_2, \ldots, y_n$ за съответните стойности на аргумента, съхранявани в sol.x.

Начертаването на графиката на i-тата функция y_i може да стане с командата

```
plot(sol.x,sol.y(i,:))
```

Записът sol ни дава възможност да пресметнем приблизителното решение в *произволна* точка x от интеграционния интервал [a, b] с помощта на функцията **deval**:

```
Sx = deval(sol,x);
```

Аргументът x може да бъде *скалар* или *вектор* с елементи, разположени в интервала [a, b]. Ако x е скалар, Sx ще бъде вектор-стълб със стойностите на функциите $y_1(x)$, $y_2(x)$, …, $y_n(x)$. Ако пък x е вектор, тогава Sx ще бъде матрица, в j-тия стълб на която се съхранява решението y(x(j)):

$Sx(1, j) = y_1(x(j));$
$Sx(2, j) = y_2(x(j));$
……...……
$Sx(n, j) = y_n(x(j)).$

Забележка: Името на изходящия аргумент sol е произволно, задавано от потребителя, но имената на полетата му x и y се задават от системата!

☐ Случаят с 5 изходящи аргумента е твърде специален и ще бъде разгледан в раздел 5.4.7.

Преди да продължим с описанието на възможните опции, нека разгледаме един пример, илюстриращ извикването на един от solver-ите с различен брой изходящи аргументи.

5.3.3. Пример – махало с вибрираща точка на окачване

Точката на окачване на математично махало вибрира във вертикално направление по закона

$$\xi = h \sin pt$$

При отчитане съпротивлението на средата, движението на махалото се описва от диференциалното уравнение

$$\ddot{\varphi} = -\frac{g}{\ell}\sin\varphi - b\dot{\varphi} + \frac{hp^2}{\ell}\sin\varphi\sin pt \quad ,$$

където:

$g \approx 9.81$ – земното ускорение;

ℓ – дължина на махалото;

φ – ъгъл на отклонение от *долното* равновесно положение;

b – коефициент на съпротивление;

h – амплитуда на вибриране на точката на окачване;

p – честота на вибриране на точката на окачване.

Да се изследва числено движението на махалото.

Тук ще покажем три различни извиквания на solver-а **ode45** – без изходящи аргументи, с един изходящ аргумент и с два такива.

Първо извършваме канонизирането, като положим:

$$\varphi = y_1 \quad \dot{\varphi} = y_2$$

В резултат на това получаваме следната система уравнения:

$$\begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = (hp^2 \sin pt - g)\dfrac{\sin y_1}{\ell} - by_2 \end{cases}$$

След това написваме файл-функцията, пресмятаща десните части на горната система диференциални уравнения. Тя може да има следния примерен вид:

```
function yt = matmah(t,y,l,h,p,b)
% Математично махало с вибрираща
% точка на окачване с честота p
yt = zeros(2,1);
yt(1) = y(2);
yt(2) = (h*p^2*sin(p*t)-9.81)...
        *sin(y(1))/l - b*y(2);
```

Следва главната програма **mahalo1**.

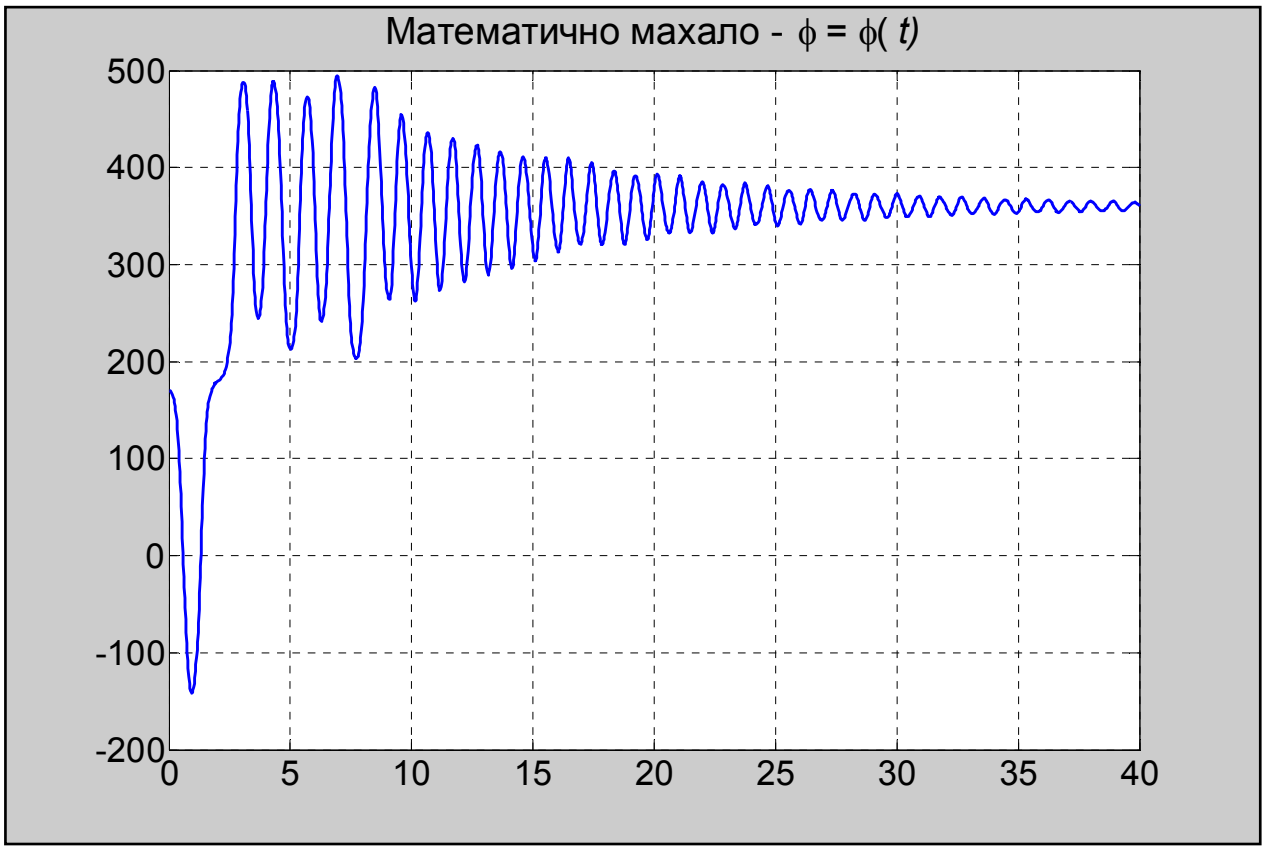
```
% ================================================
%                  Програма mahalo1
% ================================================
% Трептене на математично махало с вибрираща
% точка на окачване.
% Илюстрация на извикване на solver-а ode45
% с 0, 1 и 2 изходящи аргумента.

% Въвеждане стойности на параметрите
l = input(' Дължина на махалото l = ');
h = input(' Амплитуда на вибриране на точката на окачване h = ');
p = input(' Честота на вибриране на точката на окачване p = ');
b = input(' Коефициент на съпротивление b = ');
T = input(' Горна граница на интегриране T = ');
fi0 = input(' Начално отклонение в градуси fi0 = ');
w0  = input(' Начална ъглова скорост w0 = ');
eps = input(' Желана относителна точност eps = ');
fi0 = pi/180*fi0; % преминаване от градуси в радиани
```

```
% ------------------------------------------------
% А.) Извикване на ode45 без изходящи аргументи
% ------------------------------------------------
ode45(@matmah,[0,T],[fi0,w0],[],l,h,p,b)
grid on
disp(' Натиснете кой да е клавиш за продължение!')
pause % пауза за разглеждане на графиката
close % затваряне на прозореца
% ------------------------------------------------
% Б.) Извикване на ode45 с 1 изходящ аргумент
% ------------------------------------------------
opt = odeset('RelTol',eps);% настройка на точността
sol = ode45(@matmah,[0,T],[fi0,w0],opt,l,h,p,b);
sol.y = 180/pi*sol.y; % радиани --> градуси
    % Начертаване на графика fi = fi(t)
plot(sol.x, sol.y(1,:)), grid on
set(gca,'FontName','Arial Cyr','FontSize',14)
title('Математично махало - {\phi = \phi(\it t)}')
disp(' Натиснете кой да е клавиш за продължение!')
pause, close
    % Начертаване на графиката w = w(t)
plot(sol.x, sol.y(2,:)), grid on
set(gca,'FontName','Arial Cyr','FontSize',14)
title('Математическо махало - {\omega = \omega(\it t)}')
disp(' Натиснете кой да е клавиш за продължение!')
pause, close
    % Начертаване на фазовата траектория w = w(fi)
plot(sol.y(1,:), sol.y(2,:)), grid on
set(gca,'FontName','Arial Cyr','FontSize',14)
title('Фазова траектория - {\omega = \omega(\phi)}')
disp(' Натиснете кой да е клавиш за продължение!')
pause, close
% ------------------------------------------------
% В.) Извикване на ode45 с 2 изходящи аргумента -
%     стандартното, най-често използвано извикване.
% ------------------------------------------------
while 1 % безкраен цикъл
   [t,y] = ode45(@matmah,[0,T],[fi0,w0],opt,l,h,p,b);
   y = 180/pi*y; % радиани --> градуси
     % Начертаване на графиката fi = fi(t)
   plot(t,y(:,1)), grid on
   set(gca,'FontName','Arial Cyr','FontSize',14)
   title('Математично махало - {\phi = \phi(\it t)}')
   disp(' Натиснете кой да е клавиш за продължение!')
   pause, close
```

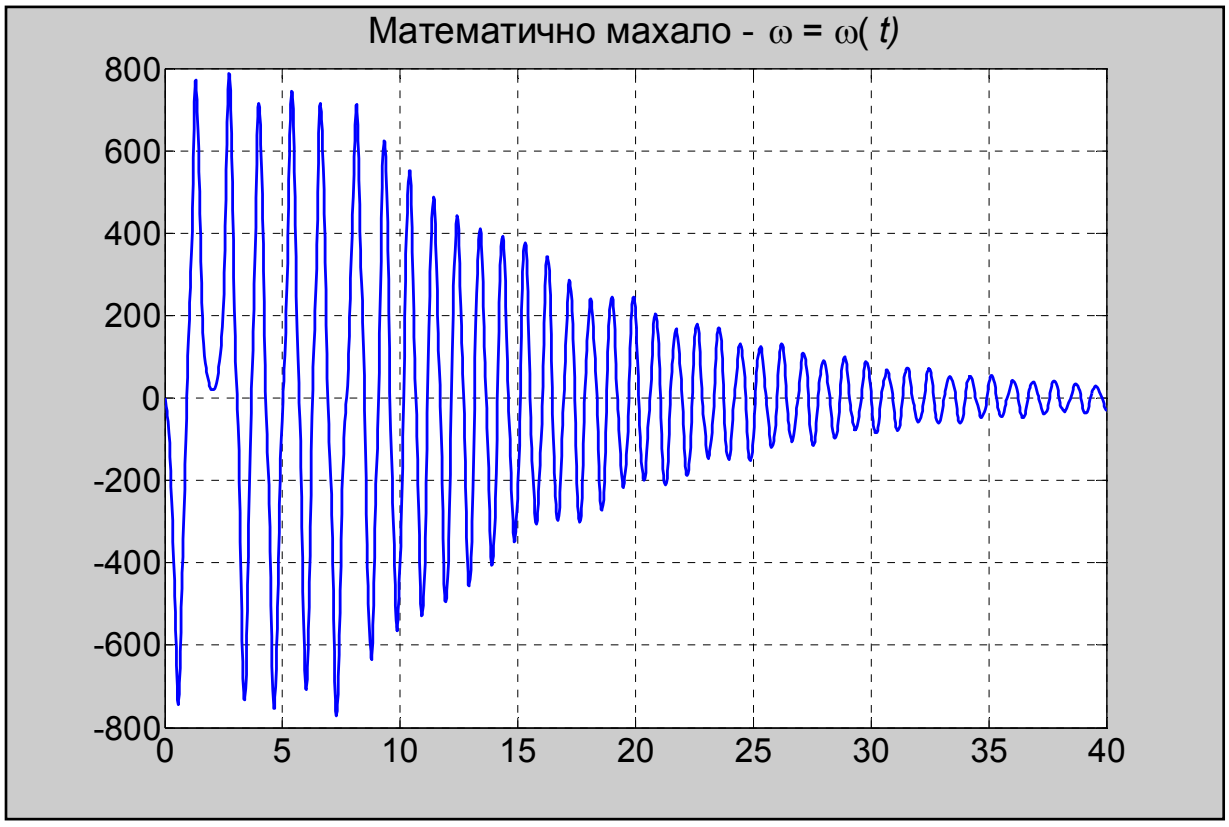
```
    % Начертаване на фазовата траектория w = w(fi)
  plot(y(:,1),y(:,2)), grid on
  set(gca,'FontName','Arial Cyr','FontSize',14)
  title('Фазова траектория - {\omega = \omega(\phi)}')
  disp(' Натиснете кой да е клавиш за продължение!')
  pause, close
  ans = input(' Желаете ли продължение? (Y/N): ','s');
  if ans == 'Y' | ans == 'y'
    p = input(' Честота на вибриране на окачването p = ');
    b = input(' Коефициент на съпротивление b = ');
    fi0 = input(' Начално отклонение в градуси fi0 = ');
    fi0 = pi/180*fi0; % градуси --> радиани
    w0  = input(' Начална скорост w0 = ');
   else
    break
  end % of if
end % of while
```

% ********** Край на програмата mahalo1 ***************

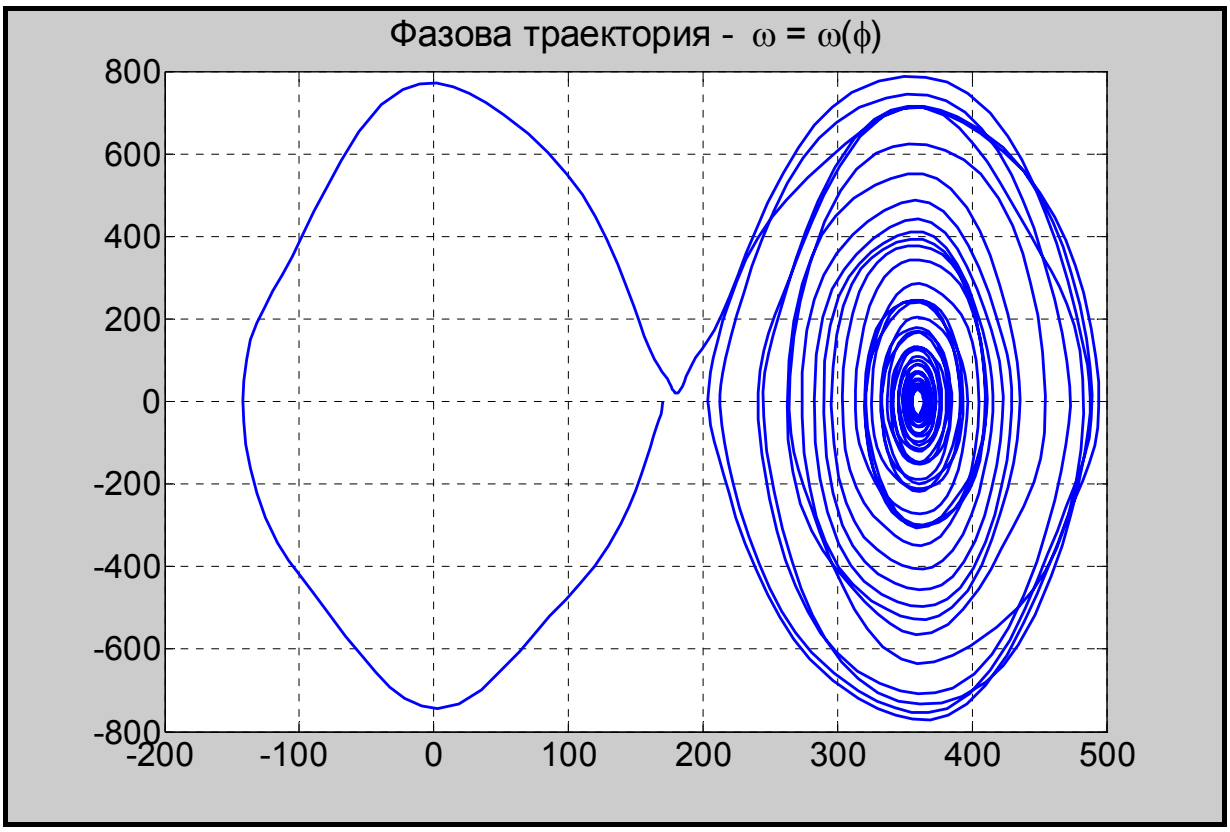
На фиг.5.1 - фиг.5.3 са показани част от резултатите от изпълнението на горната програма при следните данни: $\ell = 0.2$ m , $h = 0.01$ m , $p = 19$, $b = 0.2$, $T = 40$ s , $\varphi_o = 170^o$, $w_o = 0$.



Фиг. 5.1. Извикване на ode45 с един аргумент: закон на движение φ = φ(t)



Фиг. 5.2. Извикване на ode45 с един аргумент: ъглова скорост ω = ω(t)

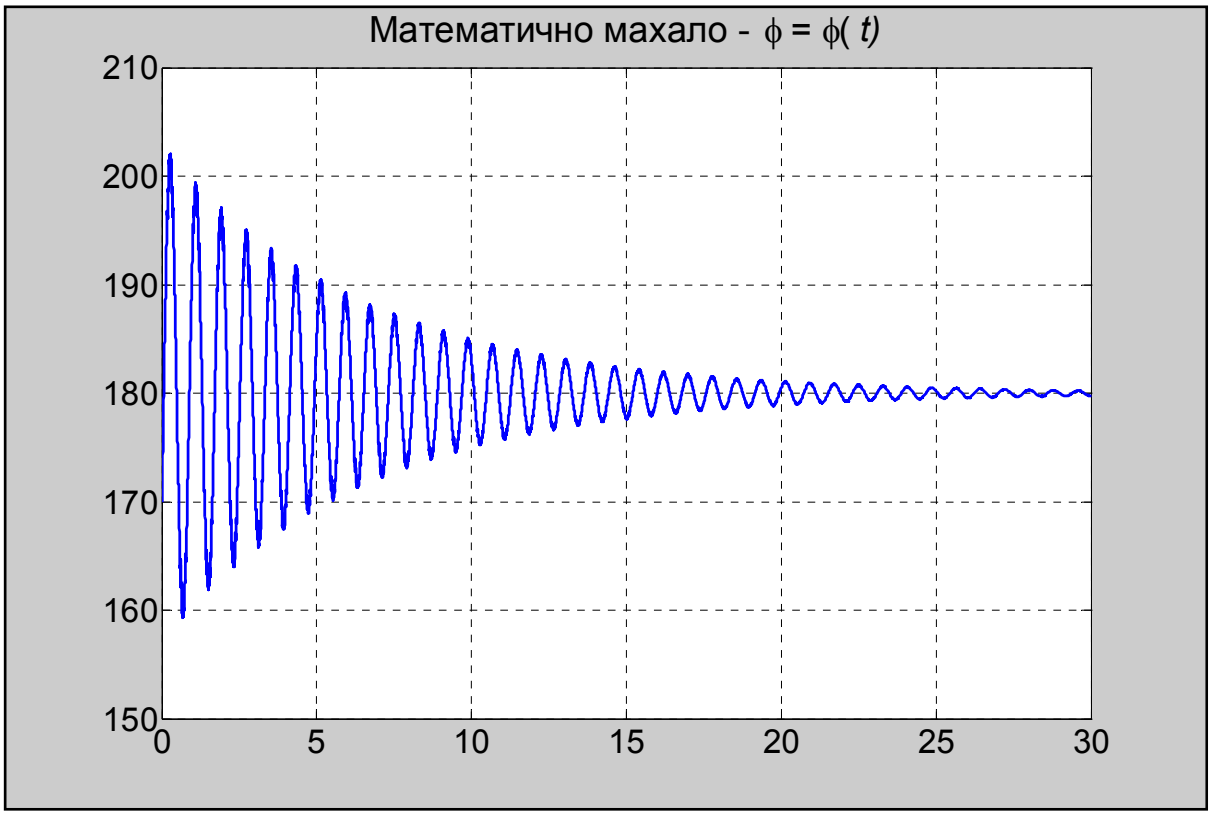


Фиг. 5.3. Извикване на ode45 с един аргумент: фазова траектория ω = ω(φ)

Графиката, построена при извикването на **ode45** без изходящ аргумент, не е показана, тъй като е твърде непрегледна – отделните точки на графиката са толкоз нагъсто, че кръговете, с които се изобразяват, се сливат в една непрекъсната дебела линия, а самите линии – в една непрекъсната боядисана площ. При други задачи картината не е толкова лоша. Може да пробвате. Графиките, получени при извикване на solver-а с два аргумента, по нищо не се различават от приложените по-горе.

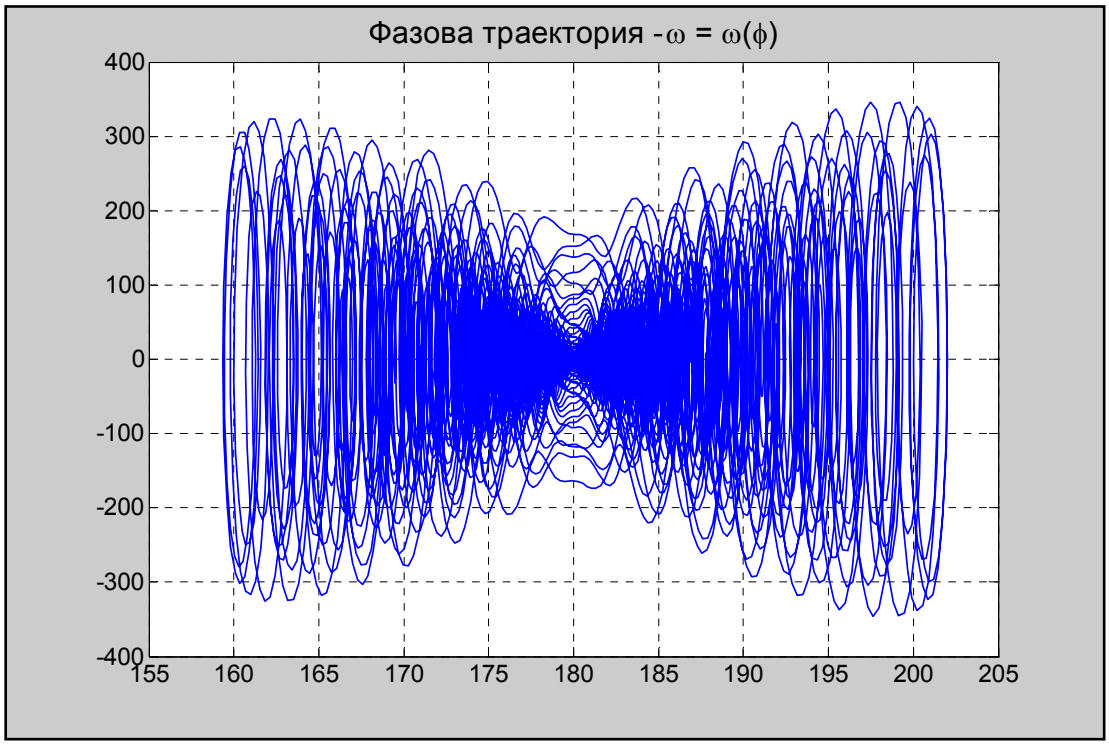
А сега ще станем свидетели на едно много интересно явление. Нека построим графиката на координатата $\varphi = \varphi(t)$ и фазовата траектория $\omega = \omega(\varphi)$ при следните данни:

$$\ell = 0.2 \text{ m} , h = 0.01 \text{ m} , p = 300 , b = 0.3 , T = 30 \text{ s} , \varphi_o = 170^o , w_o = 0.$$



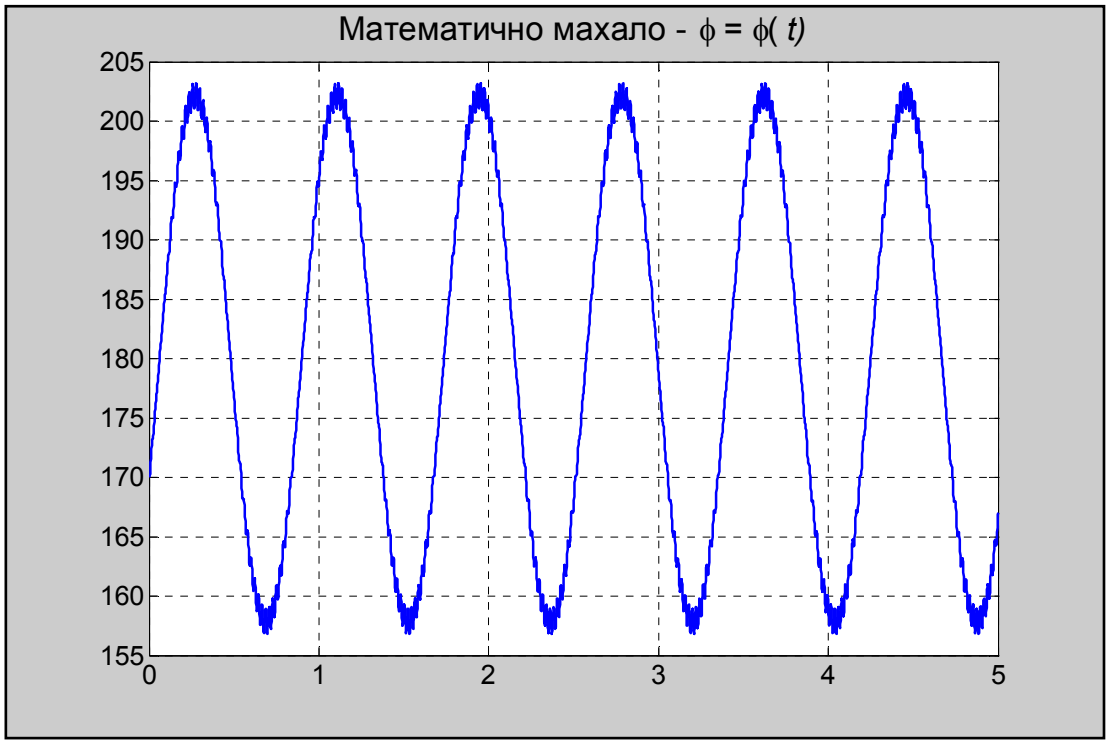
Фиг. 5.4. Трептене около горното равновесно положение – $\varphi = 180^o$

Както виждате на фиг. 5.4 и фиг. 5.5, при определени условия, вибрирането на точката на окачване във вертикално направление със сравнително висока честота превръща вертикалното неустойчиво равновесно положение на обикновеното математично махало в устойчиво. Може да не вярвате на очите си, но това е факт. Двете фигури показват, че трептенията на махалото затихват при ъгъл 180^o, т.е. около горното вертикално положение! Ако поекспериментирате малко, ще се убедите, че трептенето на махалото е извънредно чувствително към малки изменения на параметрите и на началните условия. При по-голяма настойчивост бихте могли да откриете и други стойности на параметрите, при които горното вертикално положение е устойчиво.



Фиг. 5.5. Фазова траектория при устойчиво горно вертикално положение

Някои от читателите могат да помислят, че тайната на феномена се крие в демпфирането (съпротивлението на средата), но това не е така. Главният фактор е честотата p на вибриране на точката на окачване. На фиг. 5.6. виждате трептения на същото махало около горното равновесно положение при отсъствие на съпротивление.



Фиг. 5.6. Трептене около горното положение без демпфиране