Cuprins

Ecuaţii diferenţiale în MATLAB

1. Probleme cu valori iniţiale

1.1 Rezolvitori

MATLAB are facilităţi foarte puternice de rezolvare a problemelor cu valori iniţiale pentru ecuaţii diferenţiale ordinare:

$$\frac{dy(t)}{dt} = f\big(t, y(t)\big), \qquad y(t_0) = y_0.$$

Cel mai simplu mod de a rezolva o astfel de problemă este de a scrie o funcţie care evaluează *f* şi de a apela unul dintre rezolvitorii MATLAB. Informaţia minimă pe care un rezolvitor trebuie să o primească este numele funcţiei, mulţimea valorilor lui *t* pe care se cere soluţia şi valoarea iniţială y_0. Rezolvitorii MATLAB acceptă argumente de intrare şi ieşire opţionale care permit să se specifice mai mult despre problema matematică şi modul de rezolvare a ei. Fiecare rezolvitor MATLAB este conceput să fie eficient în anumite situaţii, dar toţi sunt în esenţă interschimbabili. Toţi rezolvitorii

au aceeaşi sintaxă, ceea ce ne permite să încercăm diferite metode numerice atunci când nu ştim care ar fi cea mai potrivită. Sintaxa este

```
[t,y]=rezolvitor(@fun,tspan,y0,optiuni,p1,p2,...)
```

unde `rezolvitor` este unul din rezolvitorii daţi în tabela 1.

Rezolvitor	Tip problemă	Algoritm
`ode45`	Nonstiff	Pereche Runge-Kutta explicit˘a, cu ordinele 4 s¸i 5
`ode23`	Nonstiff	Pereche Runge-Kutta explicit˘a, cu ordinele 2 s¸i 3
`ode113`	Nonstiff	Metod˘a cu mai mult¸i pas¸i explicit˘a, cu ordin variabil, ordinele de la 1 la 13
`ode15s`	Stiff	Metod˘a cu mai mult¸i pas¸i implicit˘a, cu ordin variabil, ordinele de la 1 la 15
`ode23s`	Stiff	Pereche Rosenbrock modificat˘a (cu un pas), cu ordinele 2 s¸i 3
`ode23t`	Stiff	Regula implicit˘a a trapezului, cu ordinele 2 s¸i 3
`ode23tb`	Stiff	Algoritm Runge-Kutta implicit, ordinele 2 s¸i 3

Argumentele de intrare sunt

- `fun` – specifică funcţia din membrul drept. În versiunile 6.x este un handler de funcţie, iar în versiunile 5.x este un nume de funcţie (în acest caz se scrie `'fun'` nu `@fun`);
- `tspan` – vector ce specifică intervalul de integrare. Dacă este un vector cu două elemente `tspan=[t0 tfinal]`, rezolvitorul integrează de la `t0` la `tfinal`. Dacă `tspan` are mai mult de două elemente rezolvitorul returnează soluţiile în acele puncte. Abscisele trebuie ordonate crescător sau descrescător. Rezolvitorul nu îşi alege paşii după valorile din `tspan`, ci obţine valorile în aceste puncte prin prelungiri continue ale formulelor de bază care au acelaşi ordin de precizie ca şi soluţiile calculate în puncte.

- `optiuni` – opţiunile permit setarea unor parametrii ai rezolvitorului şi se crează cu `odeset`.

Parametrii de ieşire sunt:

- `t` – vectorul coloană al absciselor;
- `y` – tabloul soluţiilor: o linie corespunde unei abscise, iar o coloană unei componente a soluţiei.

După `optiuni` pot să apară parametrii variabili, `p1`, `p2`, `...` care sunt transmişi funcţiei `fun` la fiecare apel. De exemplu

```
[t,y]=rezolvitor(@fun,tspan,y0,optiuni,p1,p2,...)
```

apelează

```
fun(T,Y,flag,p1,p2,...).
```

1.2 Exemple

Să considerăm ecuaţia scalară

$$y'(t) = -y(t) + 5e^{-t}\cos 5t, \quad y(0) = 0.$$

pentru $t \in [0,3]$. Membrul drept este conţinut în fişierul `f1scal.m`:

```
function yder=f1scal(t,y)
%F1SCAL Exemplu de EDO scalara

yder = -y+5*exp(-t).*cos(5*t);
```

Vom folosi rezolvitorul `ode45`. Secvenţa de comenzi MATLAB

```
tspan = [0,3]; yzero=0;
[t,y]=ode45(@f1scal,tspan,yzero);
plot(t,y,'k--*')
xlabel('t'), ylabel('y(t)')
```

produce graficul din Figura 1. Soluţia exactă este $y(t) = e^{-t} \sin 5t$. Verificăm aceasta calculând maximul modulului diferenţelor dintre valorile furnizate de `ode45` şi valorile soluţiei exacte calculate în abscisele furnizate de `ode45`:

```
norm(y-exp(-t).*sin(5*t),inf)

ans =
  3.8416e-004
```

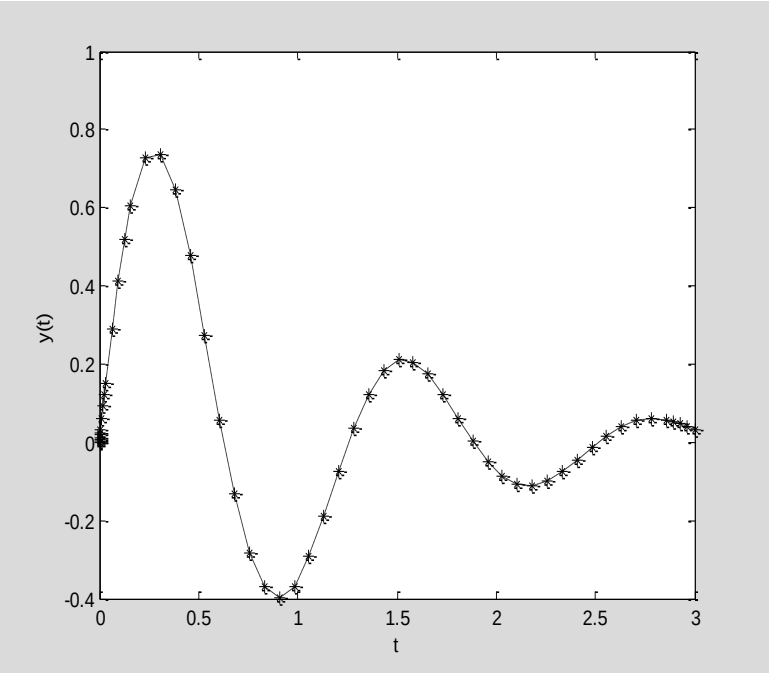


Figura 1 Graficul pentru ecuaţia scalară

Să considerăm acum ecuaţia pendulului simplu

$$\frac{d^2}{dt^2}\theta(t) = -\frac{g}{L}\sin\theta(t),$$

unde g este acceleraţia gravitaţională şi L este lungimea pendulului. Această ecuaţie de ordinul al doilea se poate transforma într-un sistem de ecuaţii de ordinul I, introducând necunoscutele $y_1 = \theta(t)$, $y_2 = d\theta/dt$.

2. Probleme cu valori pe frontieră

Funcţia `bvp4c` utilizează metoda colocaţiei pentru a rezolva sisteme de ecuaţii diferenţiale cu condiţii pe frontieră. Acestea pot fi scrise sub forma

$$\frac{d}{dx}y(x) = f(x, y(x), p), \qquad g(y(a), y(b), p) = 0.$$

y este un vector funcţie cu *m* componente, *f* este o funcţie de *x* şi *y* care are ca rezultat un *m*-vector, iar *p*, opţional, este un vector de parametrii care trebuie determinaţi. Soluţia se caută pe intervalul $a \le x \le b$, iar funcţia g, dată, specifică condiţiile pe frontieră. În general, problemele cu valori pe frontieră sunt mult mai dificil de rezolvat numeric decât problemele cu valori iniţiale. `bvp4c` necesită o soluţie iniţială de pornire. Soluţia iniţială şi cea finală se păstrează în structuri.

Vom începe cu o problemă scalară care descrie secţiunea transversală a unei picături de apă pe o suprafaţă plană:

$$\frac{d^2}{dx^2}h(x) + \big(1 - h(x)\big)\left(1 + \left(\frac{d}{dx}h(x)\right)^2\right)^{\frac{3}{2}} = 0, \qquad h(-1) = h(1) = 0.$$

$h(x)$ semnifică înalţimea picăturii în punctul x. Punem $y_1(x) = h(x)$, $y_2(x) = dh(x)/dx$ şi rescriem ecuaţia sub forma unui sistem de două ecuaţii de ordinul I

$$\frac{d}{dx}y_1(x) = y_2(x)$$

$$\frac{d}{dx}y_2(x) = (y_1(x) - 1)\Big(1 + y_2^2(x)\Big)^{\frac{3}{2}}$$

Acest sistem se reprezintă prin funcţia

```
function yprime=drop(x,y)
%DROP ODE/BVP picatura de apa
%apel yprime=drop(x,y)

yprime = [y(2); (y(1)-1)*((1+y(2)^2)^(3/2))];
```

Condiţiile iniţiale se dau sub formă reziduală

```
function res=dropbc(ya,yb)
%DROPBC ODE/BVP conditii pe frontiera picatura de apa
%apel res=dropbc(ya,yb)

res = [ya(1); yb(1)];
```

Vom alege ca valori de pornire (soluţii iniţiale) funcţiile $y_1(x) = \sqrt{1-x^2}$ şi $y_2(x) = \frac{-x}{0.1+\sqrt{1-x^2}}$.

(fisierul `dropinit.m`)

```
function yinit = dropinit(x)
%DROPINIT ODE/BVP solutii initiale picatura de apa
%apel yinit = dropinit(x)

yinit = [sqrt(1-x^2); -x/(0.1+sqrt(1-x^2))];
```

Urmeaza scriptul care rezolvă problema şi desenează soluţia (`waterdrop.m`)

```
%WATERDROP - water droplet BVP
```

```
solinit = bvpinit(linspace(-1,1,20),@dropinit);
sol = bvp4c(@drop, @dropbc, solinit);
fill(sol.x, sol.y(1,:),[0.7, 0.7, 0.7])
axis([-1, 1, 0, 1])
xlabel('x', 'Fontsize',16)
ylabel('y','Rotation',0,'FontSize',16)
```

Exemplu de utilizare, vezi figura 1.
waterdrop

În general, `bvp4c` se apelează sub forma

```
sol = bvp4c(@odefun, @bcfun, solinit, options)
```

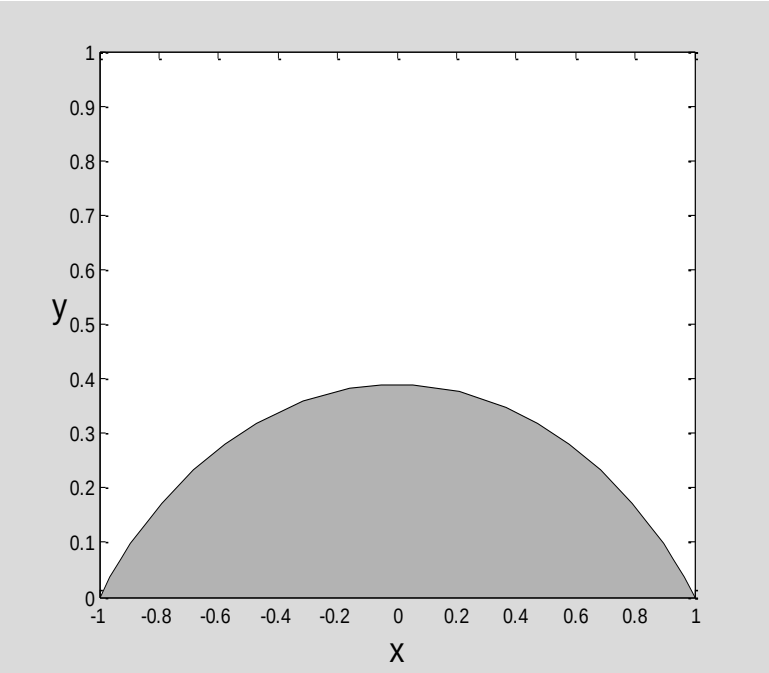


Figura 22

Funcţia `odefun` descrie ecuaţia diferenţială, iar `bcfun` dă reziduurile pentru condiţiile iniţiale. Ele returnează vectori coloană. Forma generală a lui `odefun` este

```
yprime = odefun(x,y)
```

iar a lui `bcfun`

```
res=bcfun(ya,yb).
```

Structura pentru soluţia iniţială, `solinit`, are două câmpuri `solinit.x` şi `solinit.y`. Valorile `solinit.x` dau abscisele soluţiei iniţiale, `solinit.x(1)` şi `solinit.x(end)`

conţinând *a* şi respectiv *b*. Corespunzător, `solinit.y(:,i)` dă soluţia iniţială în `solinit.x(i)`. Funcţia auxiliară bvpinit crează o structură pentru soluţia iniţială. Celelalte argumente ale lui `bvp4c` sunt opţionale. Structura `options` pemite specificarea diverşilor parametrii ai algoritmului de colocaţie, cum ar fi toleranţele, valori iniţiale, numărul maxim de puncte. Ea poate fi creată cu `bvpset`, similară cu `odeset`. (vezi `doc bvpset`).
Parametrul de ieşire sol este o structură.
`sol.x(i)` abscisa,
`sol.y(:,i)` ordonata,
`sol.yp(:,i)` aproximarea derivatei

Alt exemplu este

$$\frac{d^2}{dx^2}\theta(x) + \lambda \sin\theta(x)\cos\theta(x) = 0, \;\; \theta(-1) = \theta(1) = 0;$$

el apare în teoria cristalelor lichide.

```
function lcrun
%LCRUN   Liquid crystal BVP.
% Solves the liquid crystal BVP for four different lambda values.

lambda_vals = [2.4, 2.5, 3, 10];
lambda_vals = lambda_vals(end:-1:1); %Necessary order for
%continuation.
```

```
solinit = bvpinit(linspace(-1,1,20),@lcinit);
lambda = lambda_vals(1); sola = bvp4c(@lc,@lcbc,solinit);
lambda = lambda_vals(2); solb = bvp4c(@lc,@lcbc,sola);
lambda = lambda_vals(3); solc = bvp4c(@lc,@lcbc,solb);
lambda = lambda_vals(4); sold = bvp4c(@lc,@lcbc,solc);
plot(sola.x,sola.y(1,:),'-', 'LineWidth',4), hold on
plot(solb.x,solb.y(1,:),'--','LineWidth',2)
plot(solc.x,solc.y(1,:),'--','LineWidth',4)
plot(sold.x,sold.y(1,:),'--','LineWidth',6), hold off
legend([repmat('\lambda = ',4,1) num2str(lambda_vals')])
xlabel('x','FontSize',16)
ylabel('\theta','Rotation',0,'FontSize',16)
ylim([-0.1 1.5])

    function yprime = lc(x,y)
    %LC       ODE/BVP liquid crystal system.
    yprime = [y(2); -lambda*sin(y(1))*cos(y(1))];
    end

end

function res = lcbc(ya,yb)
%LCBC     ODE/BVP liquid crystal boundary conditions.
```

```
res = [ya(1); yb(1)];
end

function yinit = lcinit(x)
%LCINIT   ODE/BVP liquid crystal initial guess.
yinit = [sin(0.5*(x+1)*pi); 0.5*pi*cos(0.5*(x+1)*pi)];
end
```

lcrun

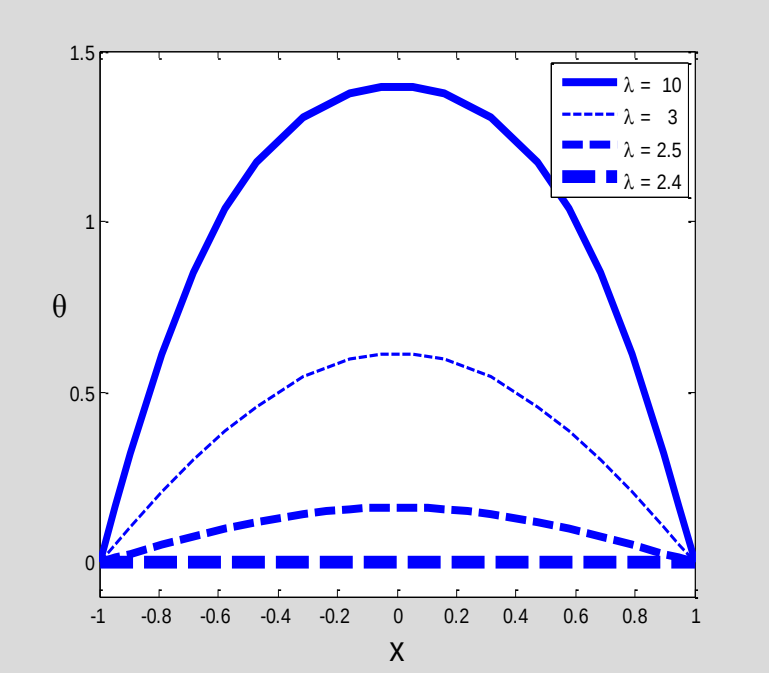


Figura 3. Problema cu cristale lichide pentru diverse valori ale lui *λ*

Am ales să scriem codul pentru soluţie sub forma unui singur fişier .m şi să scriem odefun, bcfun şi solinit sub forma unor funcţii imbricate. În acest mod λ este accesibil în toate funcţiile. Valorile pentru λ sunt ordonate descrescător. Soluţia de pornire pentru $\lambda = 10$ este $y_1(x) = \sin\frac{(x+1)\pi}{2}$ şi $y_2(x) = \frac{\pi}{2}\cos\frac{(x+1)\pi}{2}$. Pentru celelate valori ale lui λ folosim ca soluţie de pornire soluţia finală pentru valoarea precedentă. Tehnica se numeşte *continuare* în parametrul λ şi este o metodă utilă în cazul problemelor grele. Dacă λ sunt ordonate crescător se obţine de fiecare dată soluţia trivială.

Ultimul exemplu este o problemă de valori proprii referitoare la o problemă cu valori pe frontieră. Ecuaţia diferenţială este

$$\frac{d^2y(x)}{dx^2} + \mu y(x) = 0,$$

cu condiţiile

$$y(0) = 0, y'(0) = 0, y(1) + y'(1) = 0.$$

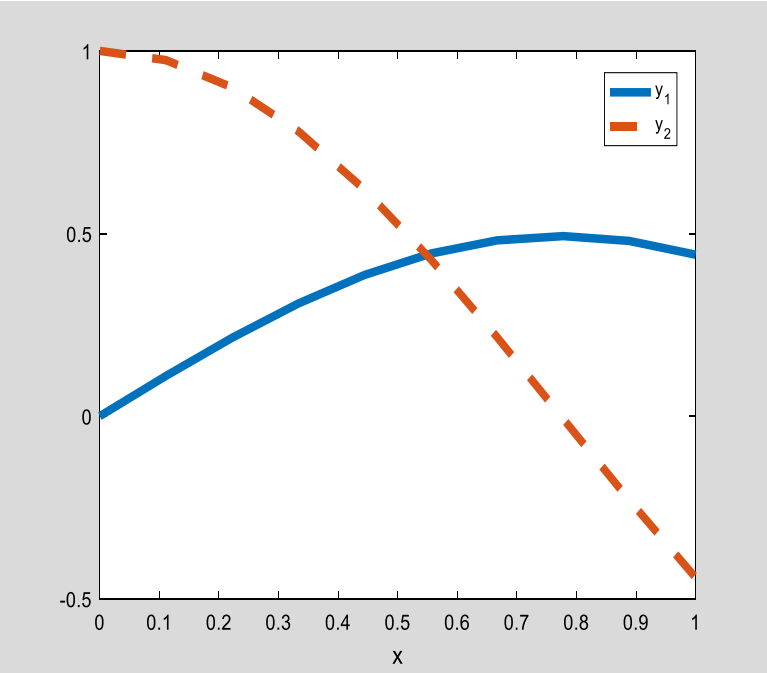
Această ecuaţie modelează deplasarea unei frânghii fixate în $x = 0$, cu suport elastic în $x = 1$ şi se roteşte cu viteză unghiulară uniformă în jurul poziţiei de echilibru de pe axa *Ox*. Problema este o problemă de valori proprii deoarece trebuie să găsim o valoare a parametrului μ pentru care există o soluţie. Putem privi cele două condiţii în $x = 0$ ca definid o problemă cu valori iniţiale şi să încercăm să determinăm valorile lui μ pentru care soluţia verifică condiţia în $x = 1$. Problema se rescrie sub forma unui sistem de două EDO de ordinul I, iar soluţiile de pornire sunt $y_1(x) = \sin x$, $y_2(x) = \cos x$. Valorea de pornire $\mu = 5$ se specifică ca parametru suplimentar în `bvpinit`.

```
function sol = skiprun

%SKIPRUN  Skipping rope BVP/eigenvalue example.

solinit = bvpinit(linspace(0,1,10),@skipinit,5);
sol = bvp4c(@skip,@skipbc,solinit);
plot(sol.x,sol.y(1,:),'-', sol.x,sol.yp(1,:),'--', 'LineWidth',4)
xlabel('x','FontSize',12)
legend('y_1','y_2')

% ------------------------ Subfunctions ----------------------
function yprime = skip(x,y,mu)
%SKIP     ODE/BVP skipping rope example.
%        YPRIME = SKIP(X,Y,MU) evaluates derivative.
yprime = [y(2); -mu*y(1)];

function res = skipbc(ya,yb,mu)
%SKIPBC   ODE/BVP skipping rope boundary conditions.
%        RES = SKIPBC(YA,YB,MU) evaluates residual.
res = [ya(1); ya(2)-1; yb(1)+yb(2)];

function yinit = skipinit(x)
%SKIPINIT ODE/BVP skipping rope initial guess.
```

```
%          YINIT = SKIPINIT(X) evaluates initial guess at X.
yinit = [sin(x); cos(x)];
```

skiprun

```
ans =
  struct with fields:

        solver: 'bvp4c'
        x: [0 0.1111 0.2222 0.3333 0.4444 0.5556 0.6667 0.7778
0.8889 1]
             y: [2×10 double]
            yp: [2×10 double]
    parameters: 4.1159
         stats: [1×1 struct]
```


1
0.5
0
-0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
x
y1
y2