Αριθμητική Επίλυση Συνήθων Διαφορίκών Εξισώσεων
4ο Εργαστήριο

08/05/2015

1 Άκαμπτα Συστήματα στη Matlab

Έστω ότι έχουμε να λύσουμε το σύστημα που δίνεται από τις εξισώσεις van der Pol:

$$\begin{cases} y_1' = y_2 \\ y_2' = 1000(1 - y_1^2)y_2 - y_1 \end{cases} \tag{1}$$

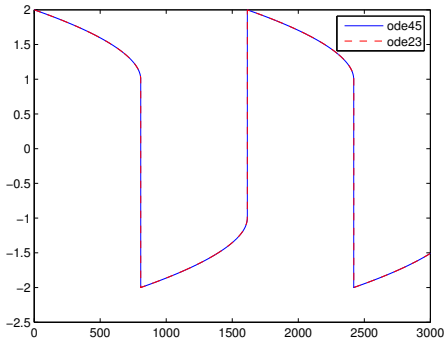
Το πρόβλημα αυτο είναι άκαμπτο και μπορεί να λύθει χρησιμοποιώντας τις έτοιμες συναρτήσεις της Matlab όπως την `ode45` και την `ode23` . Αν θέλουμε για παράδειγμα να λύσουμε με την `ode45`, τότε μπορούμε να την εκτελέσουμε όπως έχουμε δει στο προηγούμενο εργαστήριο, δηλαδή χρησιμοποιώντας τις εντολές:

```
>>opts=odeset('RelTol',1e-4);
>>[t,y]=ode45('f_stiff',[0 3000],[2 0],opts);
```

Εδώ ορίσαμε με την εντολή `odeset` την σχετική ακρίβεια να είναι της τάξης του 10^{-4}, ενώ η συνάρτηση `f_stiff` έχει οριστεί στο αρχείο `f_stiff.m`:

```
1 function dy = f_stiff(t,y)
2 dy = zeros(2,1);    % a column vector
3 dy(1) = y(2);
4 dy(2) = 1000*(1 - y(1)^2)*y(2) - y(1);
```

Εδώ αξίζει να σημειώσουμε πως η προσέγγιση της λύσης με την χρήση της `ode45` χρειάζεται κάτι περισότερο απο 5 λεπτά και 6.7 εκατομμύρια σημεία, ενώ η `ode23` χρειάζεται λίγο παραπάνω από 3 λεπτά και 2.2 εκατομμύρια σημεία. Ο λόγος πίσω από αυτό, είναι πως το σύστημά μας είναι 'πολύ' άκαμπτο καθώς περιοχές στις οποίες η λύση αλλάζει αργά εναλλάσονται με περιοχές που η λύση αλλάζει πολύ γρήγορα, όπως μπορούμε να δούμε και στο Σχήμα 1. Η `ode45`, λόγω της αυξημένης ακρίβειας της μεθόδου που υλοποιεί, χρειάζεται πολλά περισσότερα σημεία ώστε να υπολογίσει την λύση του συστήματος με την απαιτούμενη ακρίβεια. Οπότε, θυσιάζοντας ακρίβεια ώστε να μειώσουμε αρκετά τον υπολογιστικό χρόνο, προτείνουμε την χρήση της `ode23` που δίνει καλά αποτελέσματα όπως φαίνεται στο σχήμα 1.



Σχήμα 1: Προσεγγιστική λύση του y_1 με την ode45 (μπλε γραμμή) και την ode23 (κόκκινη γραμμή).

1.1 Ιδιοτιμές άκαμπτων συστημάτων

Ένας τρόπος για να αναγνωρίσουμε εάν ένα σύστημα είναι άκαμπτο είναι να ελέγξουμε τις ιδιοτιμές του. Για παράδειγμα, αν έχουμε το σύστημα διαφορικών πρώτου βαθμού,

$$\begin{cases} x_1' = x_2 \\ x_2' = -1001x_2 - 1000x_1 \end{cases}$$

το γράφουμε στη μορφή

$$x' = Ax,$$

όπου

$$A = \begin{pmatrix} 0 & 1 \\ -1000 & -1001 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

Τώρα, αν υπολογίσουμε τις ιδιοτιμές του A με την Matlab, χρησιμοποιώντας την εντολή

```
>>e = eig(A)
```

τότε αυτό θα μας επιστρέψει πως ο A έχει ιδιοτιμές $\lambda_1 = -1$ και $\lambda_2 = -1000$. Είναι εύκολο να δει κανείς πως ο λόγος τους είναι 1000, άρα οι δύο ιδιοτιμές απέχουν πολύ και συνεπως στο σύστημά μας είναι άκαμπτο.

2 Η Κλασσική Μέθοδος Runge-Kutta

Μία οικογένεια αριθμητικών μεθόδων που παρουσιάζει ιδιαίτερο ενδιαφέρον, είναι οι μέθοδοι Runge-Kutta. Κάθε μέθοδος RK (Runge-Kutta) περιγράφεται από τις σταθερές a_{ij}, τ_i, b_i, οι οποίες έχει επικρατήσει να γράφονται στη μορφή το μητρώου Butcher

$$\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1q} & \tau_1 \\ a_{21} & a_{22} & \cdots & a_{2q} & \tau_2 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{q1} & a_{q2} & \cdots & a_{qq} & \tau_q \\ \hline b_1 & b_2 & \dots & b_q & \end{array}$$

Δεδομένης μίας προσέγγισης y^n στο t^n, υπολογίζουμε την προσέγγιση y^{n+1} χρησιμοποιώντας προσεγγίσεις $y^{n,i}$ στους ενδιάμεσους χρόνους $t^{n,i}$. Συγκεκριμένα, έχουμε ότι

$$t^{n,i} = t^n + \tau_i h, \qquad i = 1, \dots, q$$

$$y^{n,i} = y^n + h \sum_{j=1}^{q} a_{ij} f(t^{n,j}, y^{n,j}), \qquad i = 1, \dots, q$$

$$y^{n+1} = y^n + h \sum_{i=1}^{q} b_i f(t^{n,i}, y^{n,i}), \qquad i = 1, \dots, q$$

Υπάρχει μία μεγάλη συλλογή από μεθόδους Runge-Kutta, εμείς όμως θα επιλέξουμε να υλοποιήσουμε την αποκαλούμενη συχνά ως κλασσική μέθοδο Runge-Kutta 4ης τάξης η οποία έχει το παρακάτω μητρώο:

$$\begin{array}{cccc|c} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 & 1 \\ \hline \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} & \end{array}$$

Η μέθοδος με βάση το παραπάνω ορίζεται ως εξής:

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} \tag{2}$$

όπου τα k_1, k_2, k_3, k_4 είναι υπολογισμένα ως:

$$k_1 = hf(t_n, y_n)$$

$$k_2 = hf((t_n + \frac{h}{2}, y_n + \frac{k_1}{2}))$$

$$k_3 = hf((t_n + \frac{h}{2}, y_n + \frac{k_2}{2}))$$

$$k_4 = hf((t_n + h, y_n + k_3))$$

Θεωρούμε τώρα το πρόβλημα αρχικών τιμών:

$$\begin{cases} y'(t) = y + 4\pi t \cos(2\pi t^2) y & t \in [0, 2], \\ y(0) = 1. \end{cases} \tag{3}$$

Θα υλοποιήσουμε σε MatLab πρόγραμμα το οποίο θα επιλύει το πρόβλημα (3) με την μέθοδο Runge-Kutta που περιγράψαμε παραπάνω χρησιμοποιώντας ομοιόμορφο με βήμα $h = (2-0)/N$.

2.1 Υλοποίηση αλγόριθμου

Η ακριβής λύση του προβλήματος (3) μπορεί να υπολογισθεί σχετικά εύκολα και είναι η

$$y(t) = e^{t+\sin(2\pi t^2)} \tag{4}$$

Το επόμενο βήμα είναι να φτιάξουμε της απαραίτητες συναρτήσεις και η πρώτη θα αφορά την ακριβής λύση. Φτιάχνουμε ένα αρχείο με όνομα exact.m με το κώδικα:

```
function [z]=exact(t)
z=exp(t+sin(2*pi*t.^2));
```

Στη συνέχεια θα φτιάξουμε τη συνάρτηση του δεύτερου μέλους οπότε φτιάχνουμε ένα αρχείο f.m που θα περιέχει τον κώδικα:

```
function [z]=f(t,y)
z=y+4*pi*t*cos(2*pi*t.^2)*y;
```

Τώρα μένει να φτιάξουμε την συνάρτηση που θα υλοποιεί τη μέθοδο Runge-Kutta γιαυτό φτιάχνουμε ένα αρχείο rk4.m με τον κώδικα:

```
function [t,y]=rk4(a,b,ya,N)
h=(b-a)/N;
t=zeros(1,N+1);
y=zeros(1,N+1);
t(1)=a;
y(1)=ya;
for i=1:N
    k1=h*f(t(i),y(i));
```

```
    k2=h*f(t(i)+h/2,y(i)+k1/2);
    k3=h*f(t(i)+h/2,y(i)+k2/2);
    k4=h*f(t(i)+h,y(i)+k3);
    y(i+1)=y(i)+k1/6+k2/3+k3/3+k4/6;
    t(i+1)=t(i)+h;
end
```

Αυτά τα 3 αρχεία είναι αρκετά για να λύσουμε το πρόβλημα (3). Μπορείτε να τρέξετε το παρακάτω script για να δείτε τη λύση που προκύπτει:

```
fclear all;
y0=exact(0);
T=2;
t0=0.0;
N=512;
[t,y]=rk4(t0,T,y0,N);
plot(t,y,'*',t,exact(t))
legend('Proseggistiki lysi','Akrivis lysi')
```

2.2 Τάξη ακρίβειας

Για την τάξη ακρίβειας θα ορίζουμε το σφάλμα ως το μέγιστο σφάλμα της προσεγγιστικής από την ακριβή λύση, δηλαδή:

$$E_N = \max_{0 \le n \le N} |y^n - y(t^n)| \tag{5}$$

Τρέχουμε στην συνέχεια το παρακάτω script:

```
clear all; close all;
T=2;
y0=exact(0);
t0=0.0;
N=[8 16 32 64 128 256 512 1024];
m=length(N);
errors=zeros(1,m);
for i=1:m
    [t,y]=rk4(t0,T,y0,N(i));
    errors(i)=norm(abs(y-exact(t)),inf);
    if (i>1)
        p1(i)=(log(errors(i-1)/errors(i))/(log((T./N(i-1))/(T./N(i)))));
    end
end
p1
```

αφού τρέξει το script βγάζει τα εξής αποτελέσματα:

```
p1 =

         0    5.1239    2.5925    4.7090    4.1239    4.0498    4.0425    4.0242

>>
```