Zarrin news

Solving ODEs in MATLAB

En.Hadi Bahmani

Winter 1391

حل معادلات دیفرانسیل معمولی با استفاده از نرم افزار متلب

ترجمه و تالیف : مهندس هادی بهمنی

زمستان ۱۳۹۱

منبع : اخبار زرین دشت

فهرست

۱ -یافتن حل‌های صریح

نرم افزار متلب یک کتابخانه وسیعی از توابع برای حل معادلات دیفرانسیل معمولی دارد . در این نوشتار ، فقط ابتدایی‌ترین این توابع بررسی خواهند شد .

۱ ۱ - معادلات مرتبه اول

هر چند نرم افزار متلب مقدمتا یک بسته عددی است ، اما می‌تواند به طور سرراست معادلات دیفرانسیل را به شکل سمبولیک حل کند . به طور مثال فرض کنید ، هدف حل معادله دیفرانسیل مرتبه اول زیر باشد :

$$y'(x) = xy. \qquad (1.1)$$

برای حل معادله دیفرانسیل بالا می توان از تابع داخلی dsolve() استفاده کرد . ورودی و خروجی برای حل این مسئله در نرم افزار متلب در زیر آورده شده است .

```
>>y = dsolve('Dy = y*x','x')
y = C1*exp(1/2*x^2)
```

توجه شود که نرم افزار متلب از حرف بزرگ D برای اشاره کردن به مشتق استفاده می کند و نیاز دارد که کل معادله در بین دو علامت کوتیشن قرار گیرد . نرم افزار متلب متغیر t را به صورت پیش فرض به عنوان متغیر مستقل در نظر می گیرد ، اما در اینجا متغیر x به طور واضح به عنوان متغیر مستقل مشخص شده است . به عنوان راهکار دیگر برای تعریف معادله می توان معادله را به صورت زیر به عنوان یک متغیر تعریف کرد :

```
>>eqn1 = 'Dy = y*x'
eqn1 =
Dy = y*x
>>y = dsolve(eqn1,'x')
y = C1*exp(1/2*x^2)
```

همانطور که در بالا نشان داده شده است معادله دیفرانسیل به عنوان متغیر eqn1 تعریف شده است .

اما برای حل همان معادله دیفرانس‌یل به صورت یک مسئله مقدار اولیه یا شرط اولیه مثلا y(1)=1 ، به صورت زیر عمل می‌شود :

```
>>y = dsolve(eqn1,'y(1)=1','x')
y =
1/exp(1/2)*exp(1/2*x^2)
```

یا اینکه می‌توان به صورت زیر عمل کرد :

```
>>inits = 'y(1)=1';
>>y = dsolve(eqn1,inits,'x')
y =
1/exp(1/2)*exp(1/2*x^2)
```

اما تا الان شیوه حل معادله دیفرانسیل (۱.۱) با استفاده از نرم افزار متلب بیان شد . فرض می‌شود هدف رسم جواب معادله دیفرانسیل در یک بازه مشخص باشد تا از این طریق یک دید کلی از نحوه رفتار جواب معادله دیفرانسیل حاصل شود . کاربر نرم افزار برای رسم جواب معادله دیفرانسیل با دو مشکل مواجه می - شود : (۱) اول اینکه جواب معادله دیفرانسیل یعنی y(x) نمی‌تواند با عملگرهای آرایه ای نظیر (.^,./,.*) وفق داده شود و (۲) تابع y(x) که نرم افزار متلب به ما بر می گرداند در اصل یک متغیر سمبولیک است . اولین این موانع برای ر سم تابع y(x) به طور سرراست می تواند حل شود ، کافی است که از تابع vectorize() استفاده شود . برای حل مشکل دوم نیز دستور eval() به خدمت گرفته می شود . دستور اخیر رشته‌های حرفی را ارزیابی می‌کند . بنابراین از دستورات زیر جهت رسم تابع y(x) در یک بازه مشخص استفاده می‌شود :

```
>>x = linspace(0,1,20);
>>z = eval(vectorize(y));
>>plot(x,z)
```

نکته : دستور eval() رشته‌های حرفی را ارزیابی می کند ، اما دستور vectorize() متغیرهای سمبولیک را به رشته‌های حرفی تبدیل می‌کند . رشته‌های حرفی می‌توانند با عملگرهای آرایه‌ای وفق داده شوند .

۱ ۲ - معادلات مرتبه دوم و بالاتر

فرض می‌شود ، هدف حل و رسم جواب معادله دیفرانسیل مرتبه دوم زیر باشد :

$$y''(x) + 8y'(x) + 2y(x) = \cos(x);\ y(0) = 0,\ y'(0) = 1. \qquad (1.2)$$

از کد زیر برای حل معادله دیفرانسیل استفاده می‌شود :

```
>>eqn2 = 'D2y + 8*Dy + 2*y = cos(x)';
>>inits2 = 'y(0)=0, Dy(0)=1';
>>y=dsolve(eqn2,inits2,'x')
y =
1/65*cos(x)+8/65*sin(x)+(-1/130+53/1820*14^(1/2))*exp((-4+14^(1/2))*x)
-1/1820*(53+14^(1/2))*14^(1/2)*exp(-(4+14^(1/2))*x)
>>z = eval(vectorize(y));
>>plot(x,z)
```

درباره دستورات vectorize و eval قبلا توضیح داده شده است .

۳ ۱ - دستگاه معادلات دیفرانسیل

فرض می‌شود که هدف حل دستگاه معادلات دیفرانسیل متشکل از سه معادله دیفرانسیل مرتبه اول زیر و رسم جواب‌های آن باشد ،

$$\begin{aligned} x'(t) &= x(t) + 2y(t) - z(t) \\ y'(t) &= x(t) + z(t) \\ z'(t) &= 4x(t) - 4y(t) + 5z(t). \end{aligned} \qquad (3.1)$$

با توجه به کد زیر نتیجه گیری می‌شود که شیوه حل دستگاه معادلات دیفرانسیل نیز مانند حل یک معادله دیفرانسیل است با این تفاوت که هر معادله دیفرانسیل در داخل یک علامت کوتیشن قرار می‌گیرد :

```
>>[x,y,z]=dsolve('Dx=x+2*y-z','Dy=x+z','Dz=4*x-4*y+5*z')
x =
2*C1*exp(2*t)-2*C1*exp(t)-C2*exp(3*t)+2*C2*exp(2*t)-
1/2*C3*exp(3*t)+1/2*C3*exp(t)
y =
```

```
2*C1*exp(t)-C1*exp(2*t)+C2*exp(3*t)-C2*exp(2*t)+1/2*C3*exp(3*t)-
1/2*C3*exp(t)
z =
-4*C1*exp(2*t)+4*C1*exp(t)+4*C2*exp(3*t)-4*C2*exp(2*t)-
C3*exp(t)+2*C3*exp(3*t)
```

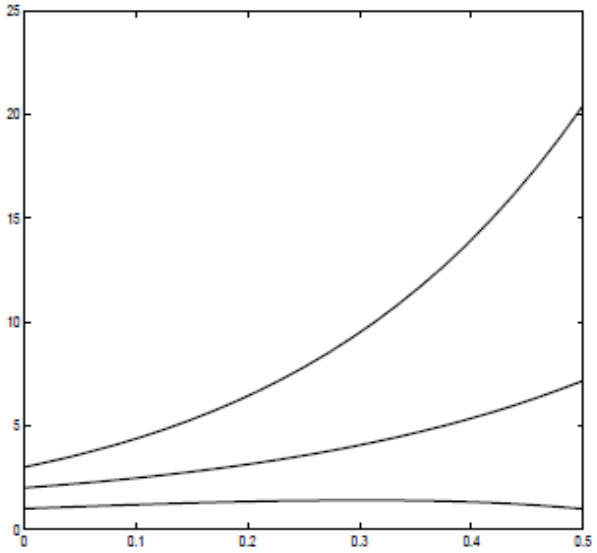
توجه کنید از آنجا که هیچ متغیر مستقلی توسط کاربر تعریف نشده است نرم افزار متلب از متغیر پیش‌فرض t به عنوان متغیر مستقل استفاده می‌کند .

اما برای حل یک مسئله مقدار اولیه ، می توان به آسانی یک سری از مقادیر اولیه را تعریف کرد و آنها را به آخر دستور dsolve() اضافه کرد . فرض می‌شود ، x(0)=1 ، y(0)=2 و z(0)=3 باشد ، بنابراین می‌توان نوشت ،

```
>>inits='x(0)=1,y(0)=2,z(0)=3';
>>[x,y,z]=dsolve('Dx=x+2*y-z','Dy=x+z','Dz=4*x-4*y+5*z',inits)
x =
6*exp(2*t)-5/2*exp(t)-5/2*exp(3*t)
y =
5/2*exp(t)-3*exp(2*t)+5/2*exp(3*t)
z =
-12*exp(2*t)+5*exp(t)+10*exp(3*t)
```

در نهایت برای رسم جواب‌های دستگاه معادله دیفرانسیل بالا بر اساس مطالب گفته شده در قبل می‌توان به صورت زیر عمل کرد ،

```
>>t=linspace(0,.5,25);
>>xx=eval(vectorize(x));
>>yy=eval(vectorize(y));
>>zz=eval(vectorize(z));
>>plot(t, xx, t, yy, t, zz)
```



نمودار ۱-۱

۲ - یافتن حل‌های عددی

نرم افزار متلب تعدادی ابزار برای حل عددی معادلات دیفرانسیل معمولی دارد . در اینجا روی دو مورد اصلی آنها ، یعنی توابع توکار ode23 و ode45 تمرکز می‌شود که به ترتیب حل های عددی به روش های رانگ-کوتا مرتبه دوم - سوم و رانگ‌کوتای مرتبه چهارم - پنجم می‌شوند .

۲ ۱ - معادلات مرتبه اول با توابع Inline

مثال ۲-۱ . به صورت عددی جواب معادله دیفرانسیل مرتبه اول زیر را تقریب بزنید ،

$dy/dx=xy^2 + y ; y(0) =1, x \in [0, .5].$

برای هر معادله دیفرانسیل به شکل $y'=f(x,y)$ ،ابتدا باید بتوان تابع $f(x,y)$ را تعریف کرد . برای فقط یک معادله ، می‌توان تابع $f(x,y)$ را به عنوان تابع inline تعریف کرد . در اینجا ،

```
>>f=inline('x*y^2+y')
f =
Inline function:
f(x,y) = x*y^2+y
```

استفاده اساسی از حل کننده ode45 متلب به صورت زیر است :

ode45(function , domain , initial condition).

بنابراین در مورد معادله ما با شرایط بالا کاربرد این حل کننده به صورت زیر است :

```
>>[x,y]=ode45(f,[0 .5],1)
```

بعد از اجرای دستور بالا نرم افزار متلب دو بردار ستونی را خروجی می دهد . بردار ستونی اول مقادیر x و بردار ستونی دوم مقادیر y می‌باشد . (از آنجا که خروج ی نرم فزار نسبتا طولانی است در اینجا از آوردن خروجی صرفنظر شده است .) از آنجا که x و y بردارهایی با مولفه های متناظر هستند ، بنابراین می توان آنها را در مقابل هم رسم کرد :

```
>>plot(x,y)
```

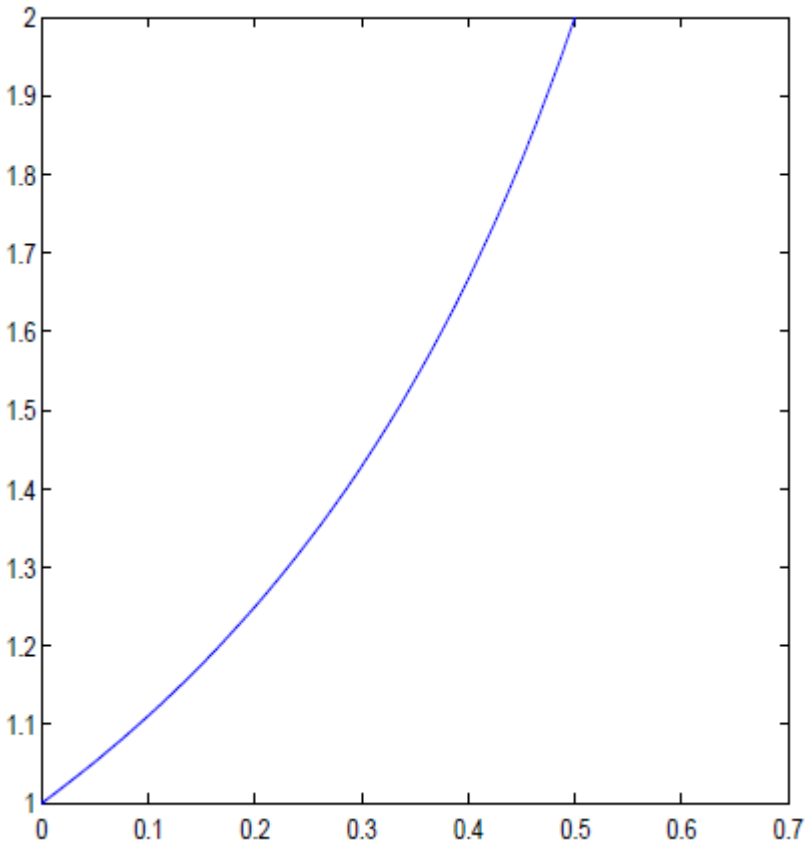
که نمودار ۲.۱ را خروجی می‌دهد .

انتخاب پارتیشن برای مقادیر x : در تقریب زدن جواب این معادله ، الگوریتم ode45 یک پارتیشن مشخصی را برای فاصله $[0,5]$ انتخاب کرده است ، و نرم افزار متلب مقادیری از y در هر نقطه‌ای در این پارتیشن را برگردانده است . در اغلب موارد کاربر دوست دارد که یک پارتیشن مشخصی را برای نرم افزار متلب مشخص کند و نرم افزار نیز مقادیر تقریبی را برای آن پارتیشن برگرداند . برای مثال اگر کاربر دوست داشته باشد که

مقادیر تقریبی را برای $y(0.1)$ ،$y(0.2)$ ....$y(0.5)$ داشته باشد ، کافی است که بردار مقادیر

[0,0.1,0.2,0.3,0.4,0.5] را به عنوان دامنه برای ode45 به عنوان دامنه مشخص کند . به این معنی که

```
>>xvalues=0:.1:.5
xvalues =
0 0.1000 0.2000 0.3000 0.4000 0.5000
>>[x,y]=ode45(f,xvalues,1)
x =
0
0.1000
0.2000
0.3000
0.4000
```



نمودار 2.1 : جواب معادله دیفرانسیل بالا

$$dy/dx=xy^2 + y \; ; \; y(0) =1, \; x \in [0, .5].$$

```
0.5000
y =
1.0000
```

1.1111
1.2500
1.4286
1.6667
2.0000

چندین آپشن (option) برای حل کننده ode45 متلب در دسترس می باشد ، که به کاربر امکان می دهد روی الگوریتم کنترل داشته باشد . دوتا از مهمترین این آپشن‌ها تلرانس خطاهای نسبی(RelTol) و مطلق (AbsTol) می‌باشند . در هر گام از الگوریتم ode45 ، یک خطا برای آن گام تخمین زده می شود . اگر y_k تقریب $y(x_k)$ در گام k ام باشد ، و e_k خطای تقریب در این گام باشد ، آنگاه نرم افزار متلب پارتیشن x را طوری در نظر می‌گیرد که نامساوی زیر ارضا شود ،

$$e_k \leq \max(\text{RelTol} * y_k, \text{AbsTol}),$$

که مقادیر پیش فرض برای RelTol=0.001 و AbsTol=0.000001 می‌باشد . به عنوان مثال برای موقعی که کاربر می خواهد این مقادیر پیش‌فرض را تغییر دهد ، مشاهده می کند که اگر y_k بزرگ شود ، آنگاه خطای e_k کاملا بزرگ خواهد شد . در این مورد ، کاربر می تواند مقدار تلرانس خطای RelTol را کاهش دهد .برای معادله دیفرانسیل $y'=xy^2+y$ ، با $y(0)=1$ ، مقادیر y برای وقتی که x به یک (۱) نزدیک می شود کاملا بزرگ می شود . در حقیقت ، با تلرانس خطاهای پیش فرض ، کاربر در می یابد که دستور :

```
>>[x,y]=ode45(f,[0,1],1);
```

منجر به صدور خطا از جانب نرم افزار متلب می شود . علت این مسئله این است که با نزدیک شدن مقادیر x به یک (۱) مقادیر y افزایش می‌یابد . (کاربر می‌تواند مشاهده کند که در بالای بردار ستونی y ، این مقادیر در 10^{14} ضرب شده‌اند .) برای حل این مشکل ، کاربر می تواند برای RelTol یک مقدار کوچکتر را انتخاب کند .

```
>>options=odeset('RelTol',1e-10);
>>[x,y]=ode45(f,[0,1],1,options);
>>max(y)
```

```
ans =
2.425060345544448e+07
```

علاوه بر به خدمت گرفتن دستور آپشن ، مقدار ماکزیمم $y(x)$ نیز محاسبه شده است که نشان می‌دهد واقعا مقدار y خیلی بزرگ است ، هر چند به بزرگی مقدارهای قبلی (بدون استفاده از دستور آپشن) که برای y محاسبه شد ، نمی‌باشد .

۲ ۲ - معادلات دیفرانسیل مرتبه اول با M-files

به عنوان یک روش دیگر برای حل معادله دیفرانسیل مثال 2.1 می‌توان ابتدا با تعریف تابع $f(x,y)$ به عنوان یک M-file با نام first.m عمل کرد .

```
function yprime = firstode(x,y);
% FIRSTODE: Computes yprime = x*y^2+y
yprime = x*y^2 + y;
```

با این روش کاربر فقط نیاز به یک تغییر در دستور ode45 دارد : یعنی باید از علامت @ استفاده کند تا به نرم افزار بفهماند که باید از M-file استفاده کند . بنابراین کاربر باید از دستورات زیر استفاده کند ،

```
>>xspan = [0,.5];
>>y0 = 1;
>>[x,y]=ode23(@firstode,xspan,y0);
>>x
```

۲ ۳ - دستگاه معادلات دیفرانسیل مرتبه اول

حل یک دستگاه معادلات دیفرانسیل معمولی مرتبه اول کاملا مشابه حل فقط یک معادله دیفرانسیل است ، هر چند که یک دستگاه معادلات دیفرانسیل را نمی توان با استفاده از تابع Inline تعریف کرد ، و باید از یک M-file بهره گرفت .

مثال ۲-۲ دستگاه معادلات دیفرانسیل لورنز را حل کنید ،

$dx/dt=-\sigma x+\sigma y$

$dy/dt=\rho x-y-xz$

$dz/dt=-\beta z+xy,$

که برای اهداف این مثال ، مقدار $\sigma=10$ ،$\beta=8/3$ و $\rho=28$ در نظر گرفته می‌شود ، علاوه بر آن $x(0)=-8$ ،$y(0)=8$ و $z(0)=27$ در نظر گرفته می شود . M-file متلب که شامل معادلات لورنز می باشد به صورت زیر می‌باشد ،

```
function xprime = lorenz(t,x);
%LORENZ: Computes the derivatives involved in solving the
%Lorenz equations.
sig=10;
beta=8/3;
rho=28;
xprime=[-sig*x(1) + sig*x(2); rho*x(1) - x(2) - x(1)*x(3); -beta*x(3)
+x(1)*x(2)];
```

مشاهده می‌شود که x به عنوان $x(1)$ ، y به عنوان $x(2)$ و z به عنوان $x(3)$ ذخیره می‌شود . علاوه بر آن ، xprime یک بردار ستونی است ، همانطور که از علامت سمیکلون مشخص است . اگر در پنجره command window تایپ شود ،

```
>>x0=[-8 8 27];
>>tspan=[0,20];
>>[t,x]=ode45(@lorenz,tspan,x0)
```

هر چند در اینجا نشان داده نشده است ، خروجی دستور اخیر متشکل از یک بردار ستونی از زمان می باشد که با یک ماتریس متشکل از سه ستون دنبال می‌شود . اولین ستون این ماتریس متناظر است با مقادیر x در زمان‌های مربوطه ، و به طور مشابه ستون‌های دوم و سوم برای y و z هستند .

حال فرض می‌شود که هدف رسم مقادیر x در مقابل z باشد ،برای این هدف از دستور زیر استفاده می‌شود :

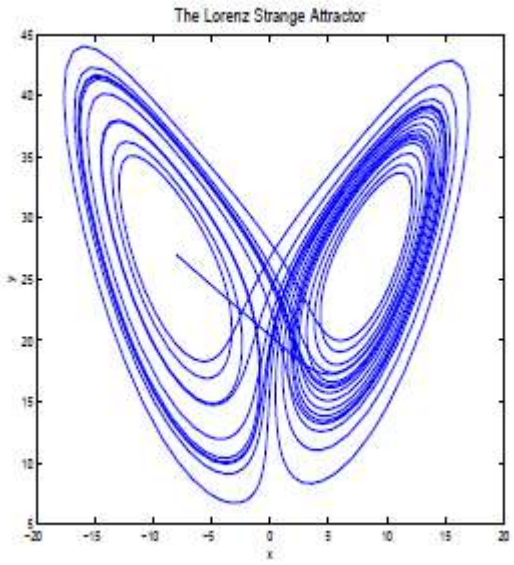
```
>>plot(x(:,1),x(:,3))
```

که خروجی این دستور در شکل ۲.۲ نشان داده شده است . البته برای رسم مقادیر x، y و z در مقابل زمان نیز می‌توان از کد زیر استفاده کرد :

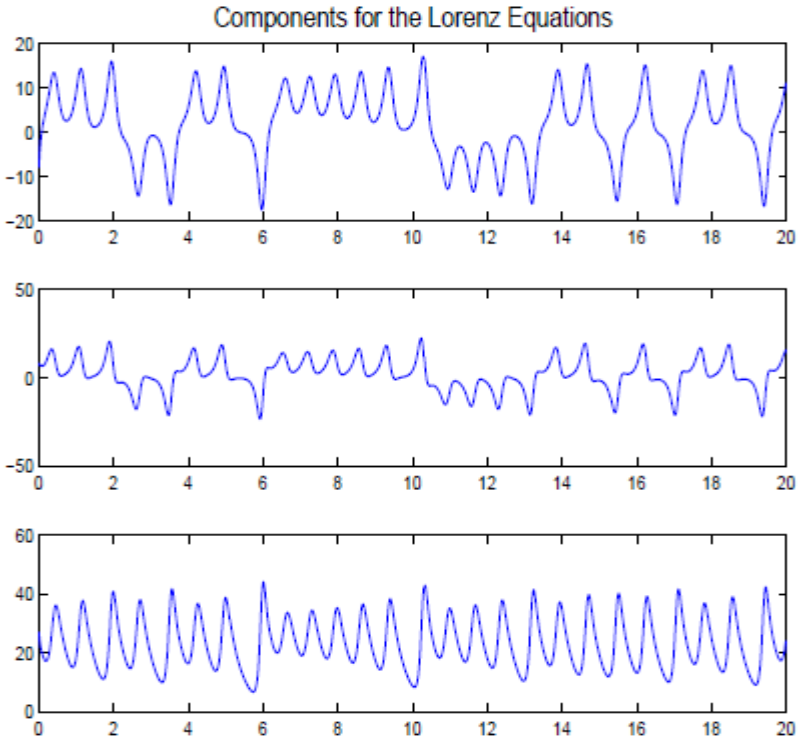
```
>>subplot(3,1,1)
>>plot(t,x(:,1))
```

```
>>subplot(3,1,2)
>>plot(t,x(:,2))
>>subplot(3,1,3)
>>plot(t,x(:,3))
```

نتیجه کد بالا نیز در شکل ۲.۳ نشان داده شده است .



نمودار 2.2



نمودار 2.3 نمودار مختصات دستگاه معادلات دیفرانسیل لورنز در مقابل زمان

۲ ۴ - پارامترهای گذر

در آنالیز دستگاه معادلات دیفرانسیل ، اغلب نیاز می شود که دستگاه معادلات دیفرانسیل با مقادیر متفاوت از پارامترها حل شود . برای مثال ، در مطالعه معادلات لورنز اغلب نیاز می شود رفتار دستگاه به عنوان تابعی از مقادیر σ ، β و ρ بررسی شود . البته یک راه برای تغییر دادن این مقادیر ، تغییر دستی این پارامترها در هر بار اجرای M-file لورنز می‌تواند باشد ، اما نه تنها این یک روش با سرعت پایین محسوب می شود ، بلکه این روش غیر اتوماتیک نیز هست . آنچه که می‌توان در عوض این روش انجام داد ، این است که مقادیر پارامتر را مستقیما در M-file از طریق دستور ode 45 وارد کرد . به منظور اینکه نشان داده شود چطور می توان این کار را کرد ، در ابتدا تابع lorenz.m به lorenz1.m تغییر نام داده می شود ، که تابع lorenz1.m یک برداری از پارامترها را که با p نشان داده می‌شود ، می‌پذیرد .

```
function xprime = lorenz1(t,x,p);
%LORENZ: Computes the derivatives involved in solving the
%Lorenz equations.
sig=p(1); beta=p(2); rho=p(3);
xprime=[-sig*x(1) + sig*x(2); rho*x(1) - x(2) - x(1)*x(3); -beta*x(3) +
x(1)*x(2)];
```

اکنون می توان مقادیر پارامترها را از طریق پنجره command window برای حل کننده ode 45 فرستاد ،

```
>>p=[10 8/3 28];
>>[t,x]=ode45(@lorenz1,tspan,x0,[],p);
```

همانطور که در بالا دیده می شود اکنون فقط کافی است که پارامترهای لازم را تنها در پنجره command window و از طریق بردار p کنترل کرد . توجه شود ، حتما لازم است قبل از بردار p ، بردار تهی در دستور فراخوان حل کننده ode 45 بکار رود .

۲ ۵ - معادلات دیفرانسیل مرتبه دوم

اولین قدم در حل معادلات دیفرانسیل معمولی مرتبه دوم یا بالاتر در نرم افزار متلب ، این است که معادله دیفرانسیل را به یک دستگاه معادلات دیفرانسیل مرتبه اول تبدیل نمود . به عنوان یک مثال ، دوباره به معادله دیفرانسیل مرتبه دوم مثال ۱.۲ از بخش ۱.۲ توجه می شود . برای تبدیل این معادله دیفرانسیل مرتبه دوم به دو معادله دیفرانسیل مرتبه اول ، $y_1(x)=y(x)$ و $y_2(x) = y'(x)$ قرار داده می‌شود ، بنابراین دستگاه معادله دیفرانسیل مرتبه اول زیر نتیجه می‌شود ،

$$y'_1(x) = y_2(x)$$

$$y'_2(x) = -8y_2(x) - 2y_1(x) + \cos(x) .$$

حال می‌توان از طریق تابع M-file زیر اقدام به حل معادله دیفرانسیل کرد ،

```
function  yprime = sorderd (y,x)
yprime =[ y(2) ; -8*y(2) – 2*y(1) + cos(x)]

>>xspan = [0,1];
>>y0=[0 1];
>>[x , y]= ode 45(@sorderd , xspan , y0)
```

۳ - معادلات دیفرانسیل معمولی با مقادیر مرزی (Boundary differential equations)

تا اینجا همه معادلات دیفرانسیلی را که بررسی شده اند ، معادلات با شرایط اولیه (initial conditions) بوده‌اند ، اما کلاس دیگری از معادلات دیفرانسیل معمولی که اغلب در عمل استفاده می‌شوند ، معادلات با مقدار مرزی می باشند (BVPs) . به عنوان نمونه به معادله دیفرانسیل زیر توجه شود ،

$$y''-3y'+2y=0$$
$$y(0)=0$$
$$y(1)=10,$$

که در اینجا شرایط $y(0)=0$ و $y(1)=10$ در مرز فاصله مورد نظر یعنی $[0,1]$ تعریف شده‌اند . (هر چند حل معادلات با شرایط مرزی می تواند برای مقادیر بیرون از بازه تعریف شده برای متغ یر مستقل نیز انجام شود ، اما رایج ترین سناریو برای حل این معادلات ، حل آنها برای مقادیر متغیر مستقل با بازه تعریف شده برای آن می باشد .) گام اول برای حل این نوع از معادله دیفرانسیل نوشتن آن به شکل

یک دستگاه معادله دیفرانسیل متشکل از دو معادله دیفرانسیل مر تبه اول می باشد ، با جایگزینی $y_1=y$ و $y_2=y'$ ، نتیجه می‌شود ،

$y'_1=y_2$

$y'_2 = -2y_1 +3y_2.$

اکنون این دستگاه معادلات دیفرانسیل در یک M-file با عنوان bvpexample.m ذخیره می‌شود ،

```
function yprime = bvpexample(t,y)
%BVPEXAMPLE: Differential equation for boundary value
%problem example.
yprime=[y(2); -2*y(1)+3*y(2)];
```

آنگاه شرایط مرزی در یک M-file با عنوان bc.m نوشته می شود ، این M-file ، باقیمانده های مرزی را ثبت می‌کند ،

```
function res=bc(y0,y1)
%BC: Evaluates the residue of the boundary condition
res=[y0(1);y1(1)-10];
```

توجه شود که در این M-file ، عنوان res برای بردار ثابت است و نمی توان عنوان آن را تغییر داد . هدف از تعریف این تابع محاسبه باقیمانده در شرایط مرزی می باشد . y0 و y1 بردارهای ستونی متناظر با $y(0)$ و $y(1)$ می‌باشد . عدد یک داخل پرانتزها دلالت کننده بر این و اقعیت هستند که شرایط مرزی برای تابع y تعریف شده‌اند . در حالتی که شرط مرزی دوم $y'(1) = 10$ باشد ، $y1(1)$ 10 – با $y1(2) – 10$ جایگزین می‌شود .

اکنون همه چیز برای حل معادله مقدار مرزی مهیاست . در کد زیر ، نخست ، یک شبکه از مقادیر x برای نرم افزار متلب مشخص شده است تا معادله دیفرانسیل برای آن بازه حل شود و نیز یک حدس اولیه برای برداری که برای یک مسئله مقدار اولیه $[y(0) , y'(0)]$ داده خواهد شد ،داده شده است . (البته مقدار $y(0)$ معلوم است و فقط باید برای مقدار $y'(0)$ یک حدس اولیه زده شود . به طور سربسته ، نرم افزار متلب یک خانواده ای از مسائل مقدار اولیه را با هدف پیدا کردن مسئله ای که برای آن شرایط مرزی صدق کند ، حل خواهد کرد .) این معادله دیفرانسیل مقدار مرزی با حل کننده داخلی bvp4c حل خواهد شد .

```
>>sol=bvpinit(linspace(0,1,25),[0 1]);
>>sol=bvp4c(@bvpexample,@bc,sol);
>>sol.x
ans =
Columns 1 through 9
0 0.0417 0.0833 0.1250 0.1667 0.2083 0.2500 0.2917 0.3333
Columns 10 through 18
0.3750 0.4167 0.4583 0.5000 0.5417 0.5833 0.6250 0.6667 0.7083
Columns 19 through 25
0.7500 0.7917 0.8333 0.8750 0.9167 0.9583 1.0000
>>sol.y
ans =
Columns 1 through 9
0 0.0950 0.2022 0.3230 0.4587 0.6108 0.7808 0.9706 1.1821
2.1410 2.4220 2.7315 3.0721 3.4467 3.8584 4.3106 4.8072 5.3521
Columns 10 through 18
1.4173 1.6787 1.9686 2.2899 2.6455 3.0386 3.4728 3.9521 4.4805
5.9497 6.6050 7.3230 8.1096 8.9710 9.9138 10.9455 12.0742 13.3084
Columns 19 through 25
5.0627 5.7037 6.4090 7.1845 8.0367 8.9726 9.9999
14.6578 16.1327 17.7443 19.5049 21.4277 23.5274 25.8196
```

مشاهده می‌شود که در این مورد ، نرم افزار متلب جواب معادله را با ساختاری که مولفه اول آن sol.x است و به طور ساده شامل مقادیر از x می‌شود که توسط کاربر تعریف شده است . مولفه دوم از ساختار ، sol.y ماتریسی است که ردیف اول آن مقادیر y(x) در نقاط متناظر با x ، و ردیف دوم آن نیز مقادیر متناظر برای y'(x) است .

به عنوان مثال دیگر معادله دیفرانسیل زیر نیز با استفاده از نرم افزار و مطالب گفته شده و بدون توضیح اضافی حل می‌شود ،

$$y''(x) + |y(x)| = 0$$

$$y(0) = 0$$

$$y(4) = -2 ;$$

باز نویسی معادله دیفرانسیل به شکل دو معادله مرتبه اول ،

$$y'_1 = y_2$$

$y'_2 = -|y_1|$

که در اینجا $y_1 = y$ و $y_2 = y'$ است .

```
function dydx = twoode(x,y)
  dydx = [ y(2) ; -abs(y(1))];


function res = twobc(ya,yb)
  res = [ ya(1) ; yb(1) + 2];
```

گرفتن جواب معادله ،

```
solinit = bvpinit(linspace(0,4,5),[0 0]);
```

در اینجا حدس اولیه برای$y(0) =0$ ، و برای $y'(0) = 0$ بوده است .

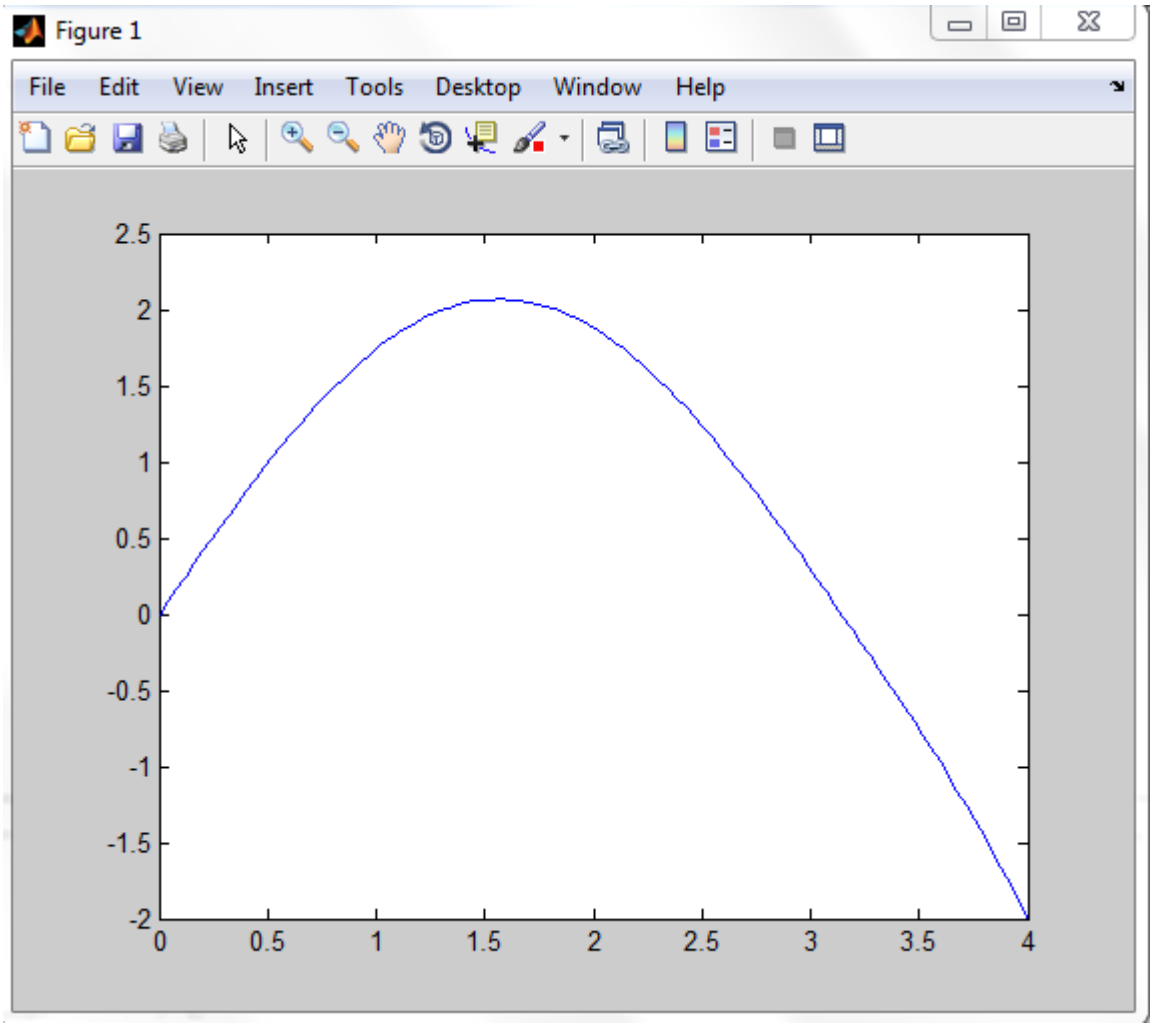
```
sol = bvp4c(@twoode,@twobc,solinit);
```

خروجی :

```
>> solinit = bvpinit(linspace(0,4,5),[0 0]);
>> sol = bvp4c(@twoode,@twobc,solinit)

sol =

    solver: 'bvp4c'
         x: [0 0.2500 0.5000 1 1.2500 1.5000 1.6250 1.7500 2 2.5000 2.7500
2.9306 3.1111 3.1389 3.1412 3.1415 3.1418 3.1901 3.2384 3.3333 3.6667 4]
         y: [2x22 double]
        yp: [2x22 double]
     stats: [1x1 struct]

>> sol.y

ans =

  Columns 1 through 16

         0    0.5114    0.9909    1.7392    1.9614    2.0617    2.0639    2.0338
1.8795    1.2371    0.7890    0.4332    0.0632    0.0058    0.0010    0.0004
    2.0669    2.0026    1.8139    1.1168    0.6518    0.1463   -0.1119   -0.3683   -
0.8600   -1.6558   -1.9104   -2.0210   -2.0659   -2.0669   -2.0669   -2.0669

  Columns 17 through 22
```

```
   -0.0002   -0.1001   -0.2003   -0.3985   -1.1356   -2.0000
   -2.0669   -2.0693   -2.0766   -2.1050   -2.3583   -2.8761
```

```
>> x = linspace(0,4);
y = deval(sol,x);
plot(x,y(1,:))
```



نمودار3-1

رسم نمودار جواب معادله مقدار مرزی

۴ - تعیین مکان زمانی یک رویداد

نوعا ، حل کننده های معادلات دیفرانسیل معمولی در نرم افزار متلب ، بعد از حل معادله روی بازه تعریف شده برای متغیر مستقل توسط کاربر ، از کار می‌ایستند . در کاربردهای گوناگون ، اما ، کاربر تمایل دارد که

حل معادله در یک مقدار مشخص از متغیر وابسته ، متوقف شود (مثلا موقعی که جمعیت از سرحد نهایی عبور کند) . به عنوان مثال ، فرض می شود هدف تعیین دوره تناوب نوسان یک پاندول باشد ، از آنجا که فاصله زمانی مناسب مشخص نیست (در حقیقت این ، همان چیزی است که کاربر به دنبال آن است) ، کاربر تمایل دارد که نرم افزار متلب ، معادله را حل کند تا اینکه پاندول به کسری از سیکل کامل نوسانش برسد و زمان متناظر با این رویداد را نمایش دهد . در این مورد ، زمانی که طول می‌کشد تا پاندول به پایین قوس نوسانش برسد ، ثبت می شود ، مسلما از ضرب این زمان در عدد چهار دوره تناوب نوسان پاندول بدست می‌آید . (در این روش ، رویداد مورد نظر مستقل از شرایط اولیه حاکم بر پاندول است)

معادله دیفرانسیل حاکم بر نوسان پاندول عبارت است از :

$$d^2\theta/dt^2 = -(g/l) \sin\theta$$

این معادله در یک M-file به نام pendode.m ذخیره می‌شود ، که در اینجا l=1 m فرض می‌شود .

```
function thetaprime=pendode(t,x)
thetaprime=[x(2);-(9.81/1)*sin(x(1))];
end
```

در تابع بالا $\theta = x(1)$ ، $d\theta/dt = x(2)$ فرض شده است .

علاوه بر این فایل ، فایل رویداد ، با عنوان pendevent.m ، نیز ساخته می شود که مشخص کننده رویدادی است که کاربر آن را جستجو می کند ،

```
function [lookfor stop direction]=pendevent(t,x)
%PENDEVENT: MATLAB function M-file that contains the event
%that our pendulum reaches its center point from the right
lookfor = x(1) ; %Searches for this expression set to 0
stop = 1; %Stop when event is located
direction = -1; %Specifiy direction of motion at event
```

در M-file بالا ، سطر lookfor = x(1) مشخص می کند که نرم افزار متلب ، باید رویداد $x(1) = 0$ را جستجو کند (به این معنی که ، $x(t) = 0$) . (اگر کاربر بخواهد رویداد $x(t) = 1$ را جستجو کند ، باید از lookfor = x(1) – 1 استفاده کند) سطر stop = 1 به نرم افزار متلب آموزش می دهد که حل را در موقعی که رویداد رخ می‌دهد ، متوقف کند ، و دستور direction = -1 به نرم افزار آموزش می دهد که

فقط رویدادهایی را قبول کند که برای آنها x(2) (x') منفی باشد . (اگر پاندول از سمت راست مرکز شروع به نوسان نماید (بدون سرعت اولیه) ،از زمانی که شروع به نوسان می کند تا زمانی که از مرکز عبور می‌کند جهت حرکت آن منفی است ، یا سرعت آن منفی است .)

اکنون معادله دیفرانسیل مرتبه اول بالا ، از زمان شروع نوسان آن تا موقعی که از مرکز عبور می کند ، حل می‌شود ، برای حل دستورات زیر در پنجره دستور صادر می شود :

```
>>options=odeset('Events',@pendevent);
>>x0=[pi/4 0];
>>[t, x, te, xe, ie]=ode45(@pendode, [0, 10], x0, options);
>>te
te =
0.5215
>>xe
xe =
-0.0000 -2.3981
```

در اینجا ،x0 یک بردار از داده های اولیه می‌باشد ، که برای شروع نوسان پاندول در نظر گرفته شده است ، یعنی زاویه شروع نوسان پاندول $\pi/4$ و سرعت اولیه صفر آن به ترتیب در این بردار وارد می شود . خروجی-های دستور ode45() عبارتند از : بردار زمان (t) ، یک ماتریس از متغیرهای وابسته x ، زمانی که در آن رویداد مورد نظر رخ می دهد یعنی te و مقادیر x در زمان رخداد رویداد (صفر شدن زاویه نوسان پاندول) یعنی xe . اگر رویداد مورد نظر خود یک بردار ی از چند رویداد باشد ، بردار شاخص ie نشان می‌دهد که کدام یک از رویدادها در هر زمان رخ می دهد . (البته در این مورد ، بردار رویداد ما فقط شامل یک رویداد است) . در اینجا فقط یک رویداد مشخص شده است ، بنابراین ie = 1 . در این مورد دیده می‌شود که زمان رخداد این رویداد در زمان t = 0.5215 ثانیه می‌باشد ، و بنابراین دوره تناوب برابر است با : P = 2.086 ثانیه (البته با احتساب خطاهای عددی در محاسبه مشتق) . هر چند که تعیین تناوب پاندول به روش عددی خیلی مشکل است ، اما محاسبه ت حلیلی دوره تناوب پاندول با فرض اینکه برای زوایای خیلی کم sinθ برابر با θ باشد اصلا مشکل نیست . با این فرض دوره تناوب پاندول تقریبا برابر است با :

$P = 2\pi\sqrt{\frac{l}{g}}$ که در اینجا P=2.001 ثانیه . (در حالی که تقریب زاویه کوچک یک دوره تناوب مستقل از زاویه θ را ارائه می‌کند ، دوره تناوب پاندول واقعا به زاویه θ بستگی دارد .)

برای اینکه درک بیشتری از مفهوم شاخص ایجاد شود ، در این مرحله ثابت می شود که زمان لازم برای هر ربع نوسان پاندول یکسان است . یعنی زمان‌هایی که در آن زاویه θ برابر صفر است و علاوه بر آن زمان-هایی که در آنها $\theta'=0$ است ، تعیین می‌شوند و به زمان های بین رخداد این حالت ها توجه می‌شود . در این مورد pendevent.m با pendevent1.m جایگزین می‌شود .

```
function [lookfor stop direction]=pendevent1(t,x)
%PENDEVENT1: MATLAB function M-file that contains the event
%that our pendulum returns to its original postion pi/4
lookfor = [x(1);x(2)]; %Searches for this expression set to 0
stop = [0;0]; %Do not stop when event is located
direction = [0;0]; %Either direction accepted
```

در این مورد ، وقوع دو رویداد جستجو می‌شود ، و بنابراین متغیرهای داخل pendevent1.m بردارهایی از دو مولفه خواهند بود ، هر کدام یک از آنها متناظر با یک رویداد . در این مورد بعد از رخداد هر رویداد محاسبات متوقف نمی‌شود ، و هیچ جهتی مشخص نمی‌شود . در پنجره دستور ، کد زیر تایپ می‌شود :

```
>>options=odeset('Events',@pendevent1);
>>x0=[pi/4 0];
>>[t, x, te, xe, ie]=ode45(@pendode,[0 2],x0,options);
>>te
te =
0.0000
0.5216
1.0431
1.5646
>>xe
xe =
0.7854   -0.0000
-0.0000   -2.3972
-0.7853  0.0000
0.0000    2.3970
>>ie
ie =
2
```

1
2
1

دیده می شود که در بازه زمانی [0 , 2] ، زمان های رخداد رویدادها به صورت تقریبی عبارتند از : 0 ، 0.5216 ، 1.0431 و 1.5646 . با نگاه کردن به ماتریس xe ، که برای این ماتریس مقدار اول در هر ردیف مکان زاویه ای و مقدار دوم سرعت زاویه ای است ، می توان فهمید که رویداد اول متناظر با موقعیت مکانی شروع نوسان است ، رویداد دوم متناظر با حالتی است که پاندول کاملا در امتداد قائم است (زاویه صفر) ، رویداد سوم نیز متناظر با حالتی است که پاندول کاملا در سمت مخالف قرار دارد و در نهایت رویداد چهارم نیز متناظر با حالتی است که پاندول در برگشت از حالت تعادل عبور می کند (زاویه صفر) . الان کاملا روشن شده است که شاخص ie چگونه کار می‌کند : مقدار این شاخص برای موقعی که رویداد دوم رخ می‌دهد برابر ۲ ، و برای موقعی که رویداد اول رخ می‌دهد برابر با ۱ است .

۵ - روش‌های عددی

هر چند می‌توان با استفاده از نرم افزار متلب ، معادلات دیفرانسیل معمولی را بدون داشتن اطلاعاتی راجع به روش‌های عددی حل این معادلات ، حل کرد ، اغلب درک اصول زیربنایی روش های عددی می تواند مفید باشد . در این قسمت ، از تئوری تیلور جهت استخراج روش های تقریبی حل معادلات دیفرانسیل استفاده می‌شود .

۵ ۱ - روش اویلر

معادله دیفرانسیل مرتبه اول کلی زیر را در نظر بگیرید ،

$$\frac{dy}{dx} = f(x,y)\ ; y(x_0) = y0 \qquad (6.1)$$

فرض کنید ، هدف حل این معادله روی بازه ای از مقادیر x یعنی $[x_0 , x_n]$ باشد ، به عبارتی دیگر هدف تعیین مقادیر تابع $y(x)$ به ازای هر مقدار x در پارتیشن $P=[x_0 ,x_1,x_2,\ldots,x_n]$ باشد . از آنجا که مقدار $y(x_0)$ داده شده است ، اولین مقداری از x که باید تخمین زده شود ، $y(x_1)$ است . با استفاده از تئوری تیلور می‌توان نوشت :

$y(x_1) = y(x_0) + y'(x_0)(x_1 - x_0) + y'(c)\ (x_1 - x_0)^2 / 2$,

که در اینجا ، $c \in (x0, x1)$ است .

با مشاهده معادله می‌توان نتیجه‌گیری کرد که : $y'(x0) = f(x0, y(x0)),$

و بنابراین :

$$y(x_1) = y(x_0) + f(x_0, y(x_0))(x_1 - x_0) + y'(c)(x_1 - x_0)^2/2$$

اگر پارتیشن P دارای زیر بازه های کوچکی باشد ، آنگاه x_1-x_0 کوچک خواهد بود ، و می توان کمیت کوچک‌تر $y'(c)(x1 - x0)^2/2$ را به عنوان ترم خطا در نظر گرفت . به این معنی که می‌توان نوشت :

$$y(x_1) \approx y(x_0) + f(x_0, y(x_0))(x_1 - x_0). \qquad (6.2)$$

اکنون می‌توان مقدار $y(x_2)$ را نیز با روشی مشابه با استفاده از تئوری تیلور محاسبه کرد ، یعنی ،

$$y(x_2) = y(x_1) + y'(x_1)(x_2 - x_1) + y'(c)(x_2 - x_1)^2/2.$$

دوباره ، از فرم کلی معادله دیفرانسیل می توان نوشت : $y'(x_1) = f(x_1, y(x_1)),$ و بنابراین ،

$$y(x_2) = y(x_1) + f(x_1, y(x_1))(x_2 - x_1) + y'(c)(x_2 - x_1)^2/2.$$

در این مورد نیز با صرفنظر از ترم $y'(c)(x_2 - x_1)^2/2$ به عنوان خطا ، می‌توان نوشت :

$$y(x_2) \approx y(x_1) + f(x_1, y(x_1))(x_2 - x_1),$$

که در اینجا باید برای مقدار $y(x_1)$ از مقدار بدست آمده در رابطه (6.2) استفاده کرد . به طور کلی ، برای هر $k = 1,2,\ldots,n\text{-}1$ می‌توان $y(x_{k+1})$ را از رابطه زیر تخمین زد :

$$y(x_{k+1}) \approx y(x_k) + f(x_k, y(x_k))(x_{k+1} - x_k),$$

که $y(x_k)$ از محاسبه قبلی (مرحله قبلی) مشخص خواهد شد . البته در روش‌های عددی انتگرال‌گیری در عمل مرسوم است که کل بازه از زیر بازه‌هایی با عرض برابر تشکیل شود ،

$$(x_{k+1} - x_k) = \Delta x = (x_n - x_0)/n .$$

(در مطالعه روش‌های عددی برای حل معادلات دیفرانسیل ، اغلب این کمیت با h نشان داده می‌شود .) در این مورد ، رابطه کلی زیر نتیجه می‌شود :

$$y(x_{k+1}) \approx y(x_k) + f(x_k, y(x_k))\Delta x.$$

اگر مقادیر y_0 ، y_1 ، y_2 ، ...، y_n تقریبات تابع y در نقاط x_0 ، x_1 ، ... ، x_n باشد (یعنی ، $y_0 = y(x_0)$ ، $y_1 \approx y(x_1)$ و غیره) ، آنگاه می‌توان $y(x)$ را روی پارتیشن P با محاسبه تکراری زیر تقریب زد :

$$y_{k+1} = yk + f(x_k, y_k)\Delta x. \qquad (6.3)$$

مثال ۶ – ۱ . برای حل معادله دیفرانسیل زیر به روش عددی از روش اویلر با $n= 10$ استفاده می‌شود ،

$$\frac{dy}{dx} = \sin(xy)\,; y(0) = \pi\,;$$

اما بازه انتگرال گیری نیز $[0\ ,\ 1]$ انتخاب می‌شود . در اینجا جهت اطلاع خواننده چند تکرار نخست برای حل معادله دیفرانسیل انجام می شود ، و برای حل کامل معادله روی بازه تعیین شده از M-file نرم افزار متلب استفاده می‌شود . در ابتدا ، مقدار اولیه $y(0) = \pi$ مقدار تابع y را در نقطه صفر تعیین می کند . اگر بازه کل از زیر بازه‌هایی به طول مساوی ۰.۱ (زیرا $n=10$ می‌باشد) تشکیل شده باشد ، آنگاه بر اساس رابطه (6.3) می‌توان نوشت :

$$y_1 = y_0 + \sin(x_0y_0)\Delta x = \pi + \sin(0) \times 0.1 = \pi\ .$$

اکنون نقطه $(x_1\ ,\ y_1) = (0.1\ ,\ \pi)$ مشخص شد ، و می‌توان با استفاده از این نقطه و رابطه (6.3) دوباره محاسبه را ادامه داد :

$$y_2 = y_1 + \sin(x_1y_1)\Delta x = \pi + \sin(0.1\pi)(0.1) = 3.1725$$

اکنون نقطه $(x_2\ ,\ y_2) = (0.2\ ,\ 3.1725)$ تعیین شده است ، و می‌توان دوباره محاسبات را ادامه داد :

$$y_3 = y_2 + \sin(x_2y_2)\Delta x = 3.1725 + \sin(.2(3.1725))(.1) = 3.2318.$$

اما برای حل کامل معادله می‌توان از M-file های زیر استفاده کرد :

```
function [xvalues, yvalues] = euler(f,x0,xn,y0,n)
%EULER: MATLAB function M-file that solve the
%ODE y'=f, y(x0)=y0 on [x0,y0] using a partition
%with n equally spaced subintervals
dx = (xn-x0)/n;
x(1) = x0;
y(1) = y0;
for k=1:n
x(k+1)=x(k) + dx;
y(k+1)= y(k) + f(x(k),y(k))*dx;
end
```

```
xvalues = x';
yvalues = y';
```

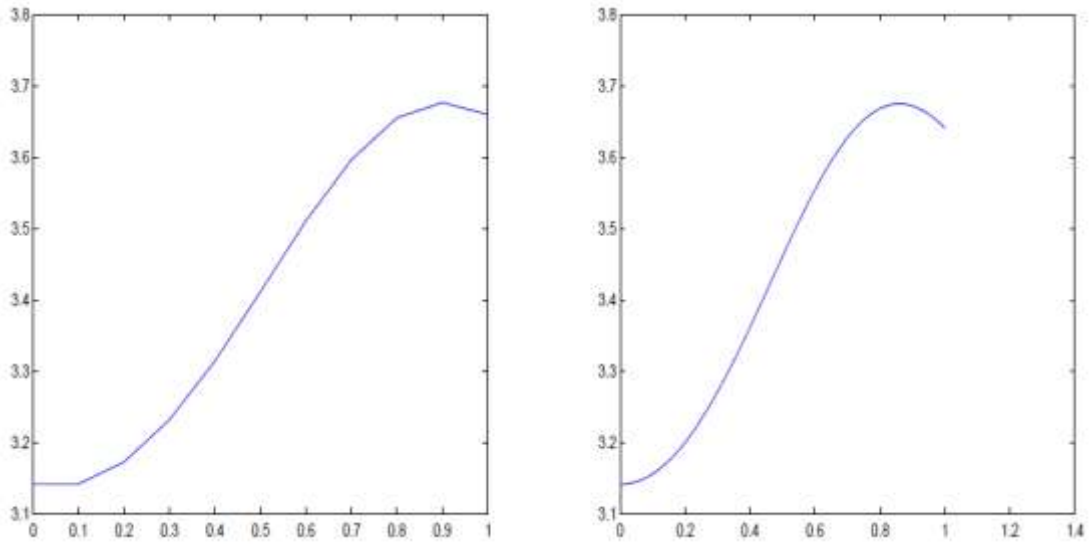
جهت کامل کردن حل این معادله می توان از کد زیر در پنجره دستور بهره برد که در نهایت نمودار 6.1 را ایجاد می‌کند ،

```
>>f=inline('sin(x*y)')
f =
Inline function:
f(x,y) = sin(x*y)
>>[x,y]=euler(f,0,1,pi,10)
x =
0
0.1000
0.2000
0.3000
0.4000
0.5000
0.6000
0.7000
0.8000
0.9000
1.0000
y =
3.1416
3.1416
3.1725
3.2318
3.3142
3.4112
3.5103
3.5963
3.6548
3.6764
3.6598
>>plot(x,y)
>>[x,y]=euler(f,0,1,pi,100);
>>plot(x,y)
```

برای مقایسه ، مقادیر دقیق برای کلیه مقادیر x در بازه مورد نظر داده شده است ،

```
x =
0
```

0.1000
0.2000
0.3000
0.4000
0.5000
0.6000
0.7000
0.8000
0.9000
1.0000
y =
3.1416
3.1572
3.2029
3.2750
3.3663
3.4656
3.5585
3.6299
3.6688
3.6708
3.6383



نمودار 6.1 ، نمودار سمت چپ ناشی از حل تقریبی معادله به روش اویلر و $\Delta x=0.1$

نمودار سمت راست، ناشی از حل تقریبی معادله به روش اویلر و $\Delta x = 0.01$

۶-۲- روش‌های تیلور مرتبه بالاتر

ایده زیربنایی روش اویلر می تواند به طور سرراست و با استفاده از چند جمله ای‌های تیلور مرتبه بالاتر بهبود یابد . در استخراج روش اویلر ، $y(x)$ با یک چند جمله‌ای مرتبه اول تخمین زده می‌شود . به طور کلی ، اگر چند جمله‌ای تیلور مرتبه n استفاده شود ، روش تیلور مرتبه n بدست می‌آید . برای نشان دادن نحوه این کار ، د ر اینجا روش تیلور مرتبه دوم استخراج می‌شود . (در واقع روش اویلر همان روش تیلور مرتبه یک می‌باشد .)

در اینجا نیز فرض می شود ، $P = [x_0, x_1, ..., x_n]$ نشان دهنده یک پارتیشن از بازه $[x_0 , x_n]$ باشد که قصد شده است معادله دیفرانسیل 6.1 روی آن حل شود . نقطه شروع برای روش تیلور مرتبه دوم این است که چند جمله ای تیلور از مرتبه دوم (با باقیمانده) برای $y(x_{k+1})$ حول نقطه x_k نوشته شود . یعنی ، بر اساس تئوری تیلور ،

$$y(x_{k+1}) = y(x_k) + y'(x_k)(x_{k+1} - x_k) + y''(x_k)\,(x_{k+1} - x_k)^2 / 2 + y'''(c)\,(x_{k+1} - x_k)^3 / 3!$$

که در اینجا نیز $c \in (x_k, x_{k+1})$. همانند روش اویلر ، از ترم خطا (که مقدار کوچکی است) صرفنظر می-شود ، و تخمین برای $y(x_{k+1})$ به صورت زیر نتیجه می‌شود :

$$y(x_{k+1}) \approx y(x_k) + y'(x_k)(x_{k+1} - x_k) + y''(x_k)(x_{k+1} - x_k)^2 / 2 \ ;$$

همانند روش اویلر در اینجا نیز مشتق $y'(x_k)$ می‌تواند با $f(x_k , y(x_k))$ جایگزین شود . علاوه بر آن ، برای روش تیلور مرتبه دوم نیاز به تخمین $y''(x_k)$ می‌باشد . برای استخراج رابطه‌ای جهت تخمین این مشتق ، می‌توان از مشتق‌گیری معادله اصلی یعنی $y'(x) = f(x, y(x))$ استفاده کرد . یعنی ،

$$y''(x) = \frac{d}{dx}y'(x) = \frac{d}{dx}f(x, y(x)) = \frac{\partial f}{\partial x}(x, y(x)) + \frac{\partial f}{\partial y}(x, y(x))\frac{dy}{dx} \quad ;$$

که این رابطه از تعمیم قاعده زنجیری مشتق‌گیری برای تابع دو متغیره نتیجه شده است . از رابطه اخیر می-توان نوشت :

$$y''(x_k) = \frac{\partial f}{\partial x}(x_k, y(x_k)) + \frac{\partial f}{\partial y}(x_k, y(x_k))y'(x_k) =$$

$$\frac{\partial f}{\partial x}(x_k, y(x_k)) + \frac{\partial f}{\partial y}(x_k, y(x_k))f(x_k, y(x_k)) \quad ;$$

با جایگزینی $y''(x_k)$ با عبارت سمت راست این رابطه ، و جایگزینی $y'(x_k)$ با $f(x_k , y(x_k))$ ، می‌توان نتیجه‌گیری کرد :

$$y(x_{k+1}) \approx y(x_k) + f\big(x_k, y(x_k)\big)(x_{k+1} - x_k) + \left[\frac{\partial f}{\partial x}\big(x_k, y(x_k)\big) +\right.$$

$$\partial f \partial y x k, y x k f x k, y x k (x k+1 - x k) 22 \ ;$$

اگر زیربازه‌های با عرض برابر $\Delta x = (x_{k+1} - x_k)$, در رابطه بالا جایگزین شود ، رابطه نهایی زیر نتیجه می‌شود :

$$y(x_{k+1}) \approx$$

$$y(x_k) + f\big(x_k, y(x_k)\big)(\Delta x) +$$

$$\left[\frac{\partial f}{\partial x}\big(x_k, y(x_k)\big) + \frac{\partial f}{\partial y}\big(x_k, y(x_k)\big)f\big(x_k, y(x_k)\big)\right]\frac{\Delta x^2}{2} \quad ;$$

مثال ۶-۲- برای حل معادله دیفرانسیل زیر از روش تیلور مرتبه دوم استفاده می‌شود ، n=10 در نظر گرفته می‌شود .

$\frac{dy}{dx} = \sin(xy)\,; y(0) = \pi$;

$x \in [0,1]$

برای اطلاع خوانندگان فقط برای چند مقدار نخست x در این بازه ، معادله به روش تیلور مرتبه دوم حل می‌شود و ادامه حل به نرم افزار متلب سپرده می‌شود . برای شروع مشاهده می‌شود که ،

$f(x,y) = \sin(xy)$,

$\frac{\partial f}{\partial x}(x,y) = y\cos(xy)$,

$\frac{\partial f}{\partial y}(x,y) = x\cos(xy)$,

اگر y_k تقریب $y(x_k)$ فرض شود ، روش تیلور مرتبه دوم نتیجه می دهد :

$$y_{k+1} = y_k + \sin(x_k y_k)(0.1) + \left[y_k \cos(x_k y_k) + x_k \cos(x_k y_k) \sin(x_k y_k) \right] (0.1)^2 / 2 .$$

با استفاده از نقطه شروع $(x_0 , y_0) = (0,\pi)$ ، می‌توان نوشت :

$$y_1 = \pi + \pi (0.005) = 3.1573 ,$$

که به مقدار دقیق یعنی 3.1572 بسیار نزدیک است . اکنون با معلوم شدن نقطه

$(x_1 , y_1) = (0.1 , 3.1573)$ می‌توان محاسبات را ادامه داد ،

$$y_2 = 3.1573 + \sin(0.1 \times 3.1573)(0.1) + \left[3.1573 \cos(0.1\times 3.1573) + 0.1\cos(0.1\times 3.1573) \sin(0.1\times 3.1573)\right] (0.1^2) / 2 = 3.2035 ,$$

این مقدار نیز به مقدار دقیق 3.2029 نزدیک است ، توجه شود که مقدار بدست آمده در این زمان از روش اویلر برابر با 3.1725 بدست آمده است . اکنون می‌توان درک کرد که روش تیلور مرتبه دوم نتایج دقیق‌تری نسبت به روش اویلر ارائه می‌دهد .

اکنون همین معادله با استفاده از دو M-file ساده نرم افزار متلب حل می‌شود ،

```
function F=taylor(x,y)
f=sin(x*y);
fx=y*cos(x*y);%derivation of f function to x.
fy=x*cos(x*y);%derivation of f function to y.
F=[f;fx;fy];
```

```
clc,close all,clear all;
xn=1;
n=10;
x=zeros(1,n+1);y=zeros(1,n+1);
x(1)=0;
y(1)=pi;
dx=(xn-x(1))/n;
for i=1:n
   x(i+1)=x(i)+dx;
   T=talor(x(i),y(i));
   y(i+1)=y(i)+T(1)*dx+(T(2)+T(3)*T(1))*(dx^2)*0.5;
end
disp(x)
disp(y)
```

```
     0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000    0.8000    0.9000    1.0000

3.1416    3.1573    3.2035    3.2766    3.3696    3.4713    3.5667    3.6399    3.6792    3.6800    3.6457
```

مراجع :

1- Solving ODE in MATLAB , P . Howard , Fall 2007

2- Solving ODEs with MATLAB,L.F.SHAMPINE,I.GLADWELL,S.THOMPSON,CAMBRIDGE UNIVERSITY PRESS ,2003

3-Help of MATLAB software