Universidade Federal de Minas Gerais - Departamento de Engenharia Eletrônica

ELT062 - OFICINA DE SIMULAÇÃO ANALÓGICA E DIGITAL EM CONTROLE

TUTORIAL 3

Instrutores: Alexandre R. Mesquita e Bruno O. S. Teixeira

Parte 1

Nosso primeiro objetivo é comparar diversos métodos de simulação disponíveis no Matlab. Os mesmos métodos não estão diretamente disponíveis em Octave, embora seja possível encontrar implementações não-oficiais.

Queremos resolver a EDO

$$\dot{x} = f(t,x) = -x^3,\ x(0) = 1,$$

cuja solução analítica é conhecida e dada por

$$x(t) = \frac{1}{\sqrt{1+2t}}\ .$$

Crie um arquivo chamado tut3a.m. Começamos nosso programa declarando a função principal e definindo os tempos de simulação e a condição inicial.

```
function edo1

close all

T=20;
x0=1;
```

Definimos $f(t,x)$ no fim do arquivo através da função

```
function f=fx(t,x)
f=-x.^3;
```

Definiremos também uma função xsol para calcular a solução analítica da EDO

```
function x=xsol(t,x0)
x=1./sqrt(2*t+1/x0^2);
```

De volta à função principal, definimos a tolerância de nossa solução numérica criando o objeto options

```
tol=1e-6;
options = odeset('RelTol',tol,'AbsTol',tol);
```

Experimente usar as linhas acima na linha de comando e observe o valor de options. Você verá que há vários outros parâmetros que podem ser configurados, como por exemplo o máximo passo de integração.

Em seguida, usamos o comando ode45 para obter a solução da EDO através do método de Dormand-Prince.

```
tic
[t,x45]=ode45(@fx,[0 T],x0,options);
toc
```

Note o uso de tic e toc para cronometrar o tempo de solução. O comando ode45 tem como entradas a função $f(t,x)$ definida através da função fx, o intervalo de integração $[0,T]$, a condição inicial x_0 e as opções de solução definidas em options. Como saída, obtemos a solução x45 nos instantes de tempo t. Note que os tempos em t não são necessariamente espaçados de maneira uniforme.

Em seguida, usando a solução exata, calculamos e exibimos o erro máximo cometido pelo método numérico.

```
erro45=max(abs(x45-xsol(t,x0)))
```

Enfim, plotamos a solução obtida e apagamos o vetor x45.

```
plot(t,x45,'r')
hold on

clear t x45
```

Como queremos comparar o tempo de solução de diversos métodos de integração, é interessante apagar x45 para que a memória utilizada por essa variável não prejudique os demais métodos testados. Ainda, para tornar a comparação mais justa, é recomendável fechar todos os demais aplicativos.

Simulamos em seguida usando os métodos ode23 e ode113.

```
tic
[t,x23]=ode23(@fx,[0 T],x0,options);
toc
erro23=max(abs(x23-xsol(t,x0)))

plot(t,x23,'k')
clear t x23

tic
[t,x113]=ode113(@fx,[0 T],x0,options);
toc
erro113=max(abs(x113-xsol(t,x0)))

plot(t,x113,'m')
clear x113
```

Queremos também comparar com o método de Euler, que não está implementado em Matlab. Por isso, temos que escrever nossa própria implementação. O primeiro passo é definir o passo de integração. Para isso, analisamos o vetor de tempo do último método utilizado e selecionamos o menor intervalo entre seus tempos para ser nosso tempo de integração. Tente entender como isso é realizado pelo código abaixo.

```
h=conv([1,-1],t);
h(1)=[];
h(end)=[];
h=min(h);
```

Agora podemos prosseguir com nossa simulação usando o método de Euler.

```
xn=x0;
xe=x0;
tic
```

```
for k=1:ceil(T/h)
    xn=xn+h*fx(0,xn);
    xe=[xe xn];
end
t=0:h:T;
xe=xe(1:length(t));
toc

plot(t,xe,'k')
erroe=max(abs((xe-xsol(t,x0))))
```

Por fim, plotamos a solução exata e salvamos a figura obtida.

```
plot(t,xsol(t,x0))
print -dpng comparacaoodesolvers
```

Execute o arquivo e observe os tempos de solução de cada método bem como os erros obtidos. Os tempos de solução podem variar muito de sistema para sistema e podem também ser afetados por outros processos sendo executados pelo sistema operacional. Experimente executar seu arquivo várias vezes e provavelmente notará uma variação significativa nesses tempos. Uma forma de diminuir o efeito dessas oscilações e tornar maiores as diferenças entre os métodos numéricos é aumentar o tempo total de simulação.

Parte 2

Nosso próximo objetivo é simular um sistema do tipo *stiff*. O sistema simulado será o oscilador de Van der Pol introduzido em PC2.

Crie o arquivo tut3b.m e inicie declarando a função principal e os parâmetros da simulação.

```
function edo1

close all
T=600;
mu=400;
x0=[1;1];
```

Em seguida, crie uma função no final do arquivo para definir a EDO a ser simulada.

```
function f=fx(t,x,mu)
f =
```

Complete a lacuna acima com o valor de $f(t,x)$ calculado em PC2 para $L = C = 1$, onde mu é o parâmetro μ.

Voltando à função principal, simulamos usando Dormand-Prince (ode45) e o método implícito ode15s para sistemas *stiff*.

```
tic
[t,x45]=ode45(@fx,[0 T],x0,[],mu);
toc

subplot(2,1,1)
plot(t,x45(:,1),'r.')
hold on
```

```
subplot(2,1,2)
plot(t,x45(:,2),'r.')
hold on


tic
[t,x15]=ode15s(@fx,[0 T],x0,[],mu);
toc

subplot(2,1,1)
plot(t,x15(:,1),'m.')
ylabel('I_L')
subplot(2,1,2)
ylabel('V_C')
xlabel('tempo(s)')
plot(t,x15(:,2),'m.')
```

Note como [] indica que nenhum objeto de opções é fornecido (portanto teremos opções padrão) e como o parâmetro μ é passado para a função fx.

Execute o arquivo e observe o resultado. O método explícito pode ser até 800 vezes mais lento que o método implícito. Note no gráfico como o método ode45 calcula um número excessivo de pontos (em vermelho) na parte lenta da curva, ao passo que ode15s calcula apenas uma dezena de pontos no trecho lento.

Experimente fazer $\mu = 4$ e verá que o método explícito se tornará mais vantojoso (o sistema não será mais *stiff*).