Universidade de São Paulo
Escola Politécnica
Curso de Engenharia de Computação

Fabiano Shimura, fabianoshimura@hotmail.com
Rafael Ribeiro Correia, rafael.correia.poli@gmail.com

RELATÓRIO apresentado ao Professor Alexandre Roma do MAP/IME-USP como atividade da disciplina MAP3122 - Métodos Numéricos.

São Paulo - SP
20/04/2016

Resumo

Este trabalho tem sua motivação analisar o comportamento das soluções das equações de Lorenz (E. N. Lorenz, Journal of Atmospheric Sciences, 1963) através dos conhecimentos adquiridos na disciplina MAP3122 - MÉTODOS NUMÉRICOS E APLICAÇÕES, lecionada pelo prof. dr. Alexandre Roma na Escola Politécnica da USP em 2016. Vamos utilizar dois métodos de aproximação numérica (Euler explícito e Runge-Kutta Clássico), bem como a construção de um gráfico pelo método de Spline Cúbica, obtido pela construção das tabelas para a avaliação dos resultados.

PALAVRAS CHAVE: Efeito Borboleta, Atrator de Lorentz, Método de Euler Explícito, Método de Runge-Kutta Clássico, Spline Cúbica

Sumário

1 Introdução

Lorenz estava questionando a fundamentação teórica dos métodos de previsão do tempo da época, baseados em regressão linear. Na sua opinião o fenômeno do tempo é demasiado não linear para que tais métodos possam dar resultados consistentes. Para testar a sua tese, comparou numericamente diversos métodos aplicados a certos modelos simplificados. A complexidade do modelo era um aspecto crítico dos experimentos, porque os computadores da época eram lentos. Lorenz dispunha de um Royal McBee LGP-30 com 16k de memória interna, capaz de realizar 60 multiplicações por segundo. Para um sistema de doze equações diferenciais, cada passo da integração numérica tomava 1 segundo. Após diversas tentativas, Lorenz acabou adotando um modelo com 3 equações introduzido por B. Saltzmann, que veio a ser chamado sistema de Lorenz. Esse modelo é uma simplificação do modelo de Rayleigh. Tais equações serão o modelo estudado no escopo deste trabalho.

Para acelerar os cálculos, Lorenz imprimia os resultados com apenas 3 dígitos decimais, embora os cálculos fossem realizados com 6 dígitos. Em algum momento reintroduziu um resultado como novo dado inicial. Para sua surpresa, o novo cálculo divergia do anterior: as previsões para 4 dias mais tarde eram totalmente distintas. Inicialmente, Lorenz acreditou que isso se devia a falha mecânica. As consequências desta descoberta de que o modelo é sensitivo aos dados iniciais foram profundas. A idéia de sensitividade não era nova. Já J. C. Maxwell havia observado no século 19 que "as mesmas causas produzem os mesmos efeitos" não significa que "causas próximas produzem efeitos próximos". A comunidade científica havia apelidado este efeito de Efeito Borboleta. Alguns anos depois (1971), D. Ruelle e F. Takens estavam questionando a interpretação matemática do fenômeno de turbulência predominante na época. E. Hopf e, posteriormente, L. Landau e E. Lifshitz haviam sugerido que turbulência corresponde a existência de toros invariantes de grande dimensão no espaço de configurações do fluido. Ruelle e Takens demonstraram que esse modelo não tem sustentação matemática. Em troca, propuseram que turbulência deve corresponder a existência no espaço de configurações de algum "atrator estranho". Um atrator é uma região do espaço de configurações que fica invariante quando o tempo passa e que atrai muitas (ou até todas as) configurações próximas. Ruelle e Takens não definiram "estranho", nem conheciam bons exemplos. De fato, o sistema de Lorenz era um exemplo espetacular dessa noção. E atrator estranho acabou significando um atrator tal que as trajetórias que convergem para ele dependem sensitivamente do ponto inicial. Mas o trabalho de Lorenz ainda era mal conhecido, e Ruelle e Takens só tinham como exemplos os atratores hiperbólicos de Smale.

Um dos aspectos mais surpreendentes desta construção é que ela é robusta: se modificarmos ligeiramente o fluxo, continua existindo um atrator. Isto é ainda mais surpreendente porque o atrator contém um ponto estacionário, acumulada por trajetórias não estacionárias. Parecia que esse fenômeno deveria pode ser destruído

por modificações do fluxo. Acreditava-se que os atratores robustos teriam que ser hiperbólicos. Este fenômeno de Lorenz mostrou que o problema de compreender o que faz um sistema dinâmico ser robusto é especialmente sutil para sistemas com tempo contínuo (fluxos).

Continuava em aberto saber se as equações originais de Lorenz realmente têm um atrator estranho. Isso foi resolvido em 1998 por W. Tucker (Suécia), que provou Teorema [W. Tucker]: As equações de Lorenz admitem um atrator estranho para os valores dos parâmetros originalmente considerados por Lorenz.

2 EDOs e Condições Inicias

Apresentação das equações envolvidas:

$\frac{dx}{dt} = \sigma(y - x)$

$\frac{dy}{dt} = x(\rho - z) - y$

$\frac{dz}{dt} = xy - \beta z$

em que a σ se chama o número de Prandtl e a ρ se chama o número de Rayleigh.

Todos os $\sigma, \rho, \beta > 0$, mas usualmente $\sigma = 10$, $\beta = \frac{8}{3}$, enquanto ρ varia.

O sistema exibe comportamento caótico para $\rho = 28$ mas tem órbitas periódicas para outros valores de ρ.

Condições Iniciais

$x_0 = 0$

$y_0 = 0$

$z_0 = 10$

As condições iniciais são arbitrárias e foram escolhidas para o estudo do efeito caótico.

No caso, para demonstrar o efeito caótico, basta que um dos termos deve ser diferente dos demais.

Para condicoes iniciais nulas, $x_0 = 0$, $y_0 = 0$, $z_0 = 0$, o sistema permanece em repouso.

3 Metodologias

Utilizamos o software Scilab como ferramenta para todos os métodos utilizados.

3.1 Euler

3.1.1 Algoritmo

```
for i=1:n-1
    t(i,:)=h*i
    x(i+1,:) = y(i,:)+dxdt (t(i,:),x(i,:),y(i,:),z(i,:))*h
    y(i+1,:) = y(i,:)+dydt (t(i,:),x(i,:),y(i,:),z(i,:))*h
    z(i+1,:) = z(i,:)+dzdt (t(i,:),x(i,:),y(i,:),z(i,:))*h
end
```

3.1.2 Discretização

$\Delta t = \frac{t_f - t_i}{n} = h$

Equações Euler

$\frac{dx}{dt} = -10x + 10y$

$\frac{dy}{dt} = 28x - y - xz$

$\frac{dz}{dt} = xy - \frac{8}{3}z$

Aplicando nas equações

$x_{k+1} = x_k + h(-10x_k + 10y_k)$

$y_{k+1} = x_k + h(28x_k - y_k - x_k z_k)$

$z_{k+1} = z_k + h(x_k y_k - \frac{8}{3}y_k)$

Matriz

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ z_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} \begin{bmatrix} 1-10h & 10h & 0 \\ 28h & 1-h & -hx \\ hy & -\frac{8}{3}h & 1 \end{bmatrix} \quad (1)$$

O método de Euler é de ordem de convergência 1.

3.2 Runge-Kutta

3.2.1 Algoritmo

```
for i=1:n-1
  k1 = dydt (t(i),y(i,:))
  ymid = y(i,:) + k1(h/2)
  k2 = dydt (t(i)+(h/2),ymid)
```

```
  ymid = y(i,:) + k2(h/2)
  k3 = dydt(t(i) + (h/2),ymid)
  yend = y(i,:) + k3h
  k4 = dydt(t(i)+h,yend) //como podemos perceber, o método eh de QUARTA ORDEM
  phi = (1/6)(k1+2k2+2k3+k4)
  y(i+1,:) = y(i,:)+phi*h
end
```

3.2.2 Discretização

$\frac{dx}{dt} = -10x - 10y$

RK4

$k_1 = -10x_k + 10y$

$k_2 = -10(x_k + \frac{h}{2}k_1) + 10y$

$k_3 = -10(x_k + \frac{h}{2}k_2) + 10y$

$k_4 = -10(x_k + h * k_1) + 10y$

$x_{k+1} = h\frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

$\frac{dy}{dt} = 28x - y - xz$

RK4

$k_1 = 28x - y_l - xz$

$k_2 = 28x - (y_k + \frac{h}{2}k_1) - xz$

$k_3 = 28x - (y_k + \frac{h}{2}k_2) - xz$

$k_4 = 28x - (y_k + h * k_1) - xz$

$y_{k+1} = y_k + h\frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

$\frac{dz}{dt} = xy - \frac{8}{3}y$

RK4

$k_1 = xy - \frac{8}{3}y$

$k_2 = xy - \frac{8}{3}y$

$k_3 = xy - \frac{8}{3}y$

$k_4 = xy - \frac{8}{3}y$

$z_{k+1} = z_k + h\frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

Sabemos que a discretização do RK é mais trabalhosa, porém ganhamos na convergência do método (ordem4).

4 Solução Manufaturada

4.1 Objetivos

O maior objetivo de desenvolver, criar, ou inventar uma solução manufaturada para uma determinada equação (neste caso três EDOs) é poder visualizar se o método a ser utilizado está primeiramente funcionando, em seguida convergindo, e finalmente se é possível usar o método para os mais variados formatos da equação em diferentes condições iniciais. Neste trabalho, percebemos que não temos uma EDO com solução cuja manufatura é trivial, já que as equações dependem de várias variáveis entre si. Este item (Solução Manufaturada) portanto seria impraticável nas condições em que encontramos.
Como bem sabemos, em tarefas anteriores, podemos desenvolver uma solução manufaturada para uma equação diferencial (simples) a ser resolvida. Vamos tratar sobre isto no item a seguir.

4.2 Resultados esperados (RK)

Vamos estudar rapidamente a convergência do método de Runge-Kutta de quarta ordem. Como demosntração, seja dada uma EDO arbitrária que queremos resolver.

Quero resolver: y + 3y = 6, y(0)=0, I=[0,1]
ou seja, y = -3y + 6 (será nossa função)
Manufaturamos uma solução para uma solução inicial:
y = 2*exp(3t)-2
na qual y(1) = 38,171

Esta equação obedece aos requisitos, uma edo escalar de primeira ordem com solução exata conhecida que seja "bem suave", isto é, a edo deverá conter apenas uma função a ser determinada, digamos y(t), com derivadas de primeira ordem, dy(t)/dt, e a sua solução deve ser conhecida e ter uma derivada a mais que a ordem do método que você está testando (então, no caso, pelo menos 3 derivadas).

m	e	r	log_2 (r)
caso rodado	erro absoluto	razão do erro	logaritmo de "r" na base 2
0	0.2516146461130353	6.76329629962169	3.0911651831383531
1	0.1708039787881006	8.656525299080698	3.1137880480769575
2	0.01454826971578882	11.740501250312397	3.5534220992384427
3	0.001062056696227387	13.698204406098887	3.77591488849701
4	7.175414418014725E-5	14.801329015380189	3.887654816539923
5	4.662978476233093E-6	15.388049622333364	3.9437384819466432
6	2.9717938332396443E-7	15.69078724128664	3.971845832190711
7	1.8755898167910345E-8	15.844582896723741	3.9859177766492513
8	1.1779874853345973E-9	15.921984232780616	3.992948233809799
9	7.387512823697762E-11	15.945657401173415	3.9950916725566685
10	4.547473508864641E-12	16.2453125	4.021951590962514

Rapidamente, observamos que como o logaritmo tende a 4. Esta é a ordem de convergência do método. Percebemos que para o método de RK de quarta ordem, usando m=4 já temos uma ótima aproximação da EDO, visto que sua ordem de método vale 4.
Ou seja, a razão do erro tende a cair em $2^4 vezes, ouseja, 16 vezesacadaiterao.$

O objetivo do estudo era entender o processo de depuração de código empregando a estratégia de solução manufaturada, a qual no escopo deste trabalho não pudemos fazer a implementação, já que não temos um sistema de EDOs tão "fácil".

4.3 Resultados esperados (Euler)

Analogamente, se quisermos resolver uma EDO pelo método de Euler, podemos descrever o processo da forma a seguir.

Resolvendo y = t + y Podemos usar o método de Euler implícito.
Uma solução manufaturada seria y = 2*EXP(t) -t + 1, com y(0)=1.
Obtemos a tabela abaixo. Como observamos, o log da razão do erro tende a 0. Ou seja, a ordem

Método de Euler Implícito			
m	erro	r: razão do erro	log(r)
0.0	0.0	0.0	0.0
1.0	1.1579208923731618E77	1.0*	0.0*
2.0	1.6006672173030544E45	7.233988925718922*E31	73.35892848547864
3.0	1.446700651498918E32	1.1064260015675064*E13	30.03474121107009
4.0	1.0277651049706742E25	1.4076179902412599*E7	16.45999455813283
5.0	3.0489794583519016E20	33708.49554776021	10.425505177975598
6.0	2.2251921429898032E17	1370.2095200890135	7.222718941500444
7.0	1.088003609011644E15	204.5206582550949	5.320668988743168
8.0	1.809020618627824E13	60.14323981762658	4.096729047370213
9.0	6.940834879095586E11	26.063444097715596	3.260533723275714
10.0	4.84308988997084 35E10	14.331418653758192	2.662454235806603
11.0	5.265144660354014E9	9.198398529177748	2.219029395987084
12.0	8.02897689349515E8	6.557678182658224	1.8806366047356278
13.0	1.594967195374293E8	5.033944846502614	1.616203940455769
14.0	3.911837900853432E7	4.0772834554988195	1.4054309469289934
15.0	1.1381789827085428E7	3.436926845674455	1.2345777132869353
16.0	3811226.9752664147	2.9863846737414037	1.0940635165870545
17.0	1434642.3553769658	2.656569395837319	0.9770355896007186
18.0	595968.3379452531	2.40724592907611	0.8784833260270994
19.0	269216.41757741384	2.213714688384043	0.7946719589471645
20.0	130679.20053967065	2.06013211334031	0.7227701134345476
21.0	67501.29327350957	1.9359510640806468	0.6605987117170734
22.0	36807.028627616004	1.8339240028428678	0.6064579350492361
23.0	21045.42508765032	1.748932533998316	0.5590056212491947
24.0	12547.392887104463	1.6772747356368891	0.517170295086136
...	...	...	...
100.0	0.21650833096776978	1.0471342966035102	0.03968690354530867
...	...	...	...
251.0	0.011086431847107558	1.0120747323855226	0.012002414369528678
252.0	0.010954859339562972	1.0120104241840349	0.0119388713893327 7
253.0	0.010825529392788358	1.0119467549421435	0.011875955787590766
254.0	0.010698392736523843	1.0118837155632243	0.011813658693186601
255.0	0.010573401416990702	1.0118212971023959	0.011751971381589723

do método vale 1.
Mais uma vez, esta seção basicamente mostra os conhecimentos agregados ao longo da disciplina.

5 Análise

5.1 Tabela de RK

Voltando aos métodos que vamos utilizar para resolver as equações, usando o SciLab podemos obter os gráficos (item a seguir) que nos mostra, as três coordenadas simultâneas, ao longo do tempo. Ou seja, aumentamos o tempo, para um determinado h (no qual n=64), e obtemos em sequência os gráficos. Vale ressaltar que ao utilizar o método de Runge-Kutta, obtemos uma aproximação muito melhor que o método de Euler, já que ela tem uma ordem de convergência 4. Então, basta observar os gráficos obtidos pelo método de RK.

t	x	y	z
0	0	0	10
0,015625	0,1	0,1	9,583489583
0,03125	0,104150391	0,1265625	9,184375271
0,046875	0,111839511	0,153360758	8,80194254
0,0625	0,122737103	0,181584101	8,435512251
0,078125	0,136730572	0,212295308	8,084439707
0,09375	0,153881784	0,246498327	7,748114675
0,109375	0,174399427	0,285194696	7,425962287
0,125	0,198623301	0,329432221	7,117444891
0,140625	0,227017927	0,38034891	6,822065096
0,15625	0,260173679	0,43921474	6,539370347
0,171875	0,298814265	0,50747343	6,26895957
0,1875	0,343809881	0,586786204	6,01049261
0,203125	0,396195729	0,679079309	5,763703452
0,21875	0,457195923	0,786596974	5,528418616
0,234375	0,528253043	0,911961486	5,304582607

t	x	y	z
0,25	0,61106382	1,058242016	5,092293012
0,265625	0,707621646	1,229033911	4,891848796
0,28125	0,820266719	1,428550114	4,703816653
0,296875	0,951744784	1,661726331	4,529122086
0,3125	1,105275488	1,934341293	4,369174279
0,328125	1,284631338	2,253152923	4,226037143
0,34375	1,49422802	2,626050103	4,102663297
0,359375	1,739226411	3,062217722	4,003213548
0,375	2,025645591	3,572309165	3,933491959
0,390625	2,360484469	4,168614409	3,901536077
0,40625	2,751846573	4,865201932	3,918413223
0,421875	3,209057488	5,677996431	3,99928624
0,4375	3,742756042	6,624728237	4,164823279
0,453125	4,364926849	7,724649205	4,443030811
0,46875	5,088820485	8,997846119	4,871575466
0,484375	5,928674591	10,46388676	5,500606301

t	x	y	z
0,5	6,899099622	12,13939479	6,395953769
0,515625	8,013921594	14,03396024	7,642294599
0,53125	9,284177894	16,14356191	9,345322523
0,546875	10,71484701	18,44046024	11,63100461
0,5625	12,29978874	20,85847405	14,63849439
0,578125	14,01436068	23,27304917	18,50126943
0,59375	15,80544151	25,47727803	23,30925451
0,609375	17,57946225	27,15917425	29,04526589
0,625	19,19098983	27,89323877	35,49673613
0,640625	20,43775615	27,17029422	42,16496859
0,65625	21,0721369	24,49779295	48,23121863
0,671875	20,84079906	19,59157472	52,6721524
0,6875	19,55696864	12,62428436	54,5884196
0,703125	17,18757545	4,392852253	53,65625951
0,71875	13,90817109	-3,80061244	50,39990527
0,734375	10,07386829	-10,6313669	45,98955188

t	x	y	z
0,75	6,104030888	-15,3330911	41,65982774
0,765625	2,349472563	-17,9251424	38,21438248
0,78125	-0,97823795	-18,9478747	35,92653042
0,796875	-3,7983241	-19,0267893	34,72041552
0,8125	-6,11238893	-18,608339	34,37811439
0,828125	-7,9571305	-17,918735	34,65704104
0,84375	-9,37470481	-17,0296061	35,33029139
0,859375	-10,3998834	-15,9358475	36,19247507
0,875	-11,0582001	-14,6131105	37,0590591
0,890625	-11,3699686	-13,0535184	37,77037568
0,90625	-11,3565818	-11,2842933	38,20132972
0,921875	-11,0465241	-9,37221271	38,27267484
0,9375	-10,4791892	-7,4155807	37,95792367
0,953125	-9,70535164	-5,52662865	37,28125895
0,96875	-8,78417033	-3,80979122	36,30561321
0,984375	-7,77770564	-2,34279633	35,11443436
1	-6,74466142	-1,16622262	33,79306006

5.2 Gráficos de RK

A seguir apresentaremos os gráficos obtidos a partir da resolução das equações diferenciais pelas metodologias apresentadas no item anterior.

É importante notar que apresentaremos apenas um gráfico para cada intervalo de tempo (apesar de termos duas metodologias) pois os resultados foram extremamente parecidos.

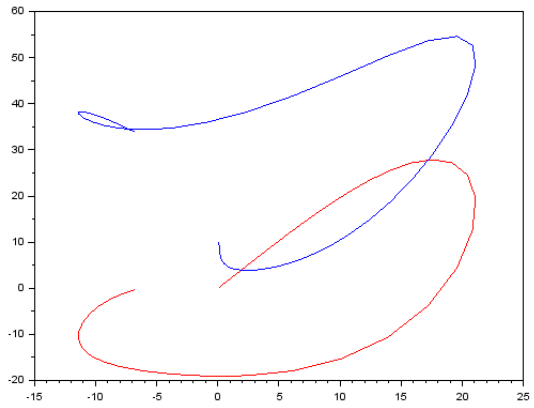
$\Delta t = \frac{t_f - t_i}{n} = h$, n=64.

Legenda:

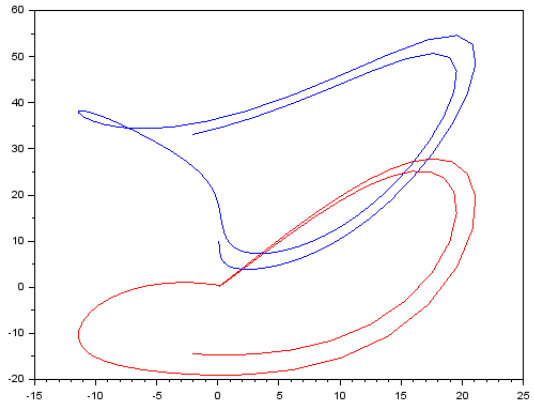
Vermelho: (x, y)

Azul: (x, z)

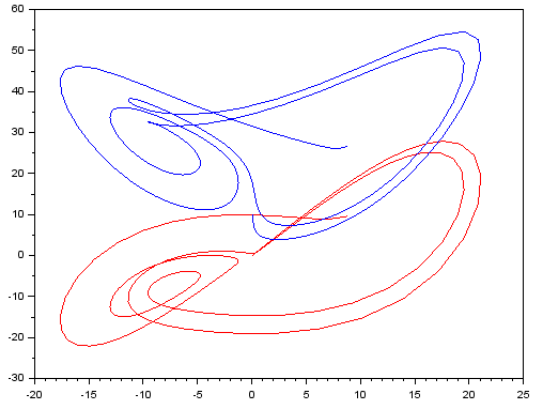
1) $0 < t < 1$



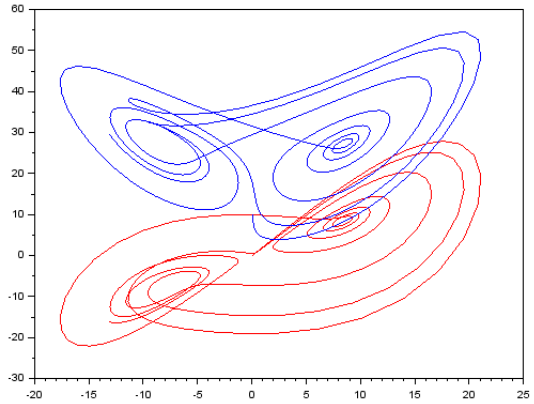
2) $0 < t < 2$



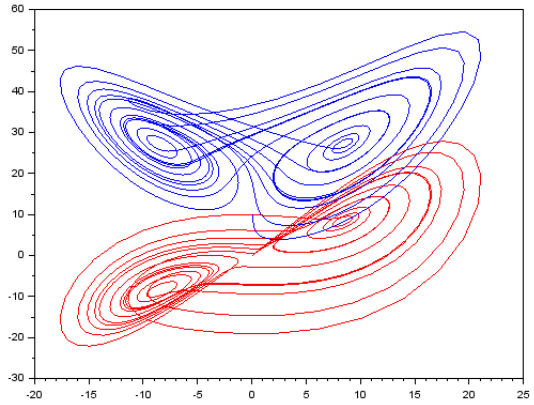
3) $0 < t < 4$



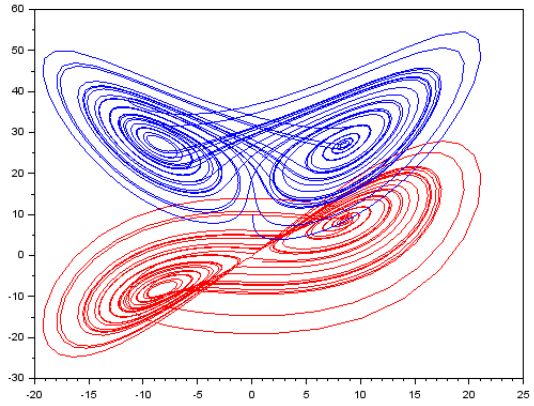
4) $0 < t < 8$



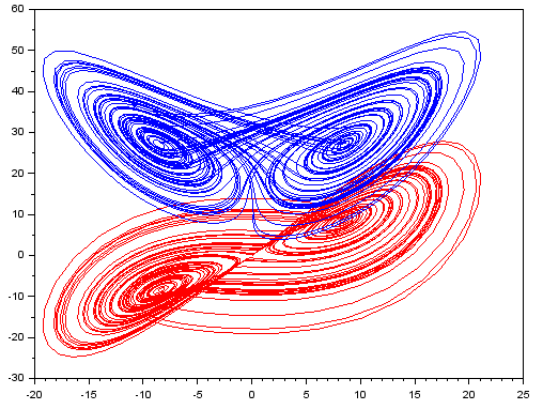
5) $0 < t < 16$



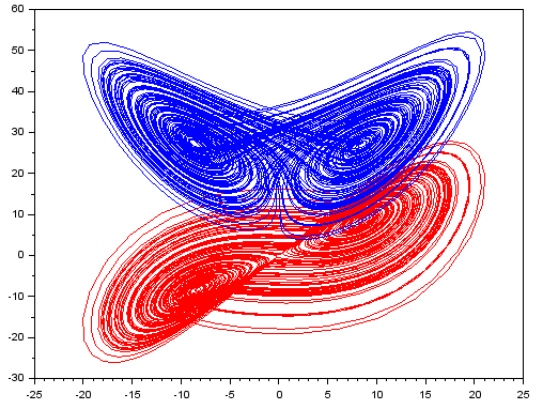
6) $0 < t < 32$



7) $0 < t < 64$



8) $0 < t < 128$



5.3 Comentários

Como previsto, o formato dos gráficos simultâneamente plotados assemelham-se a uma borboleta.

6 Splines

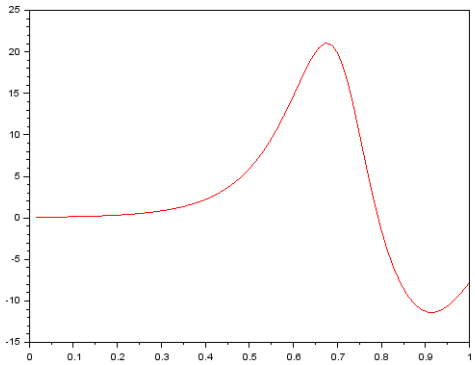
6.1 Pós-processamento

Nesta seção, vamos utilizar o spline como etapa de pós processamento, no intuito de observar a forma da curva gerada pelas tabelas geradas pelo método de RK4 (neste caso, $0 < t < 1$).

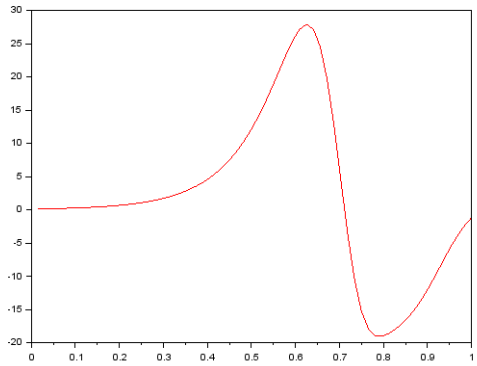
Para tanto, foi utilizada a função "splin"do SciLab, para cada variável x, y e z em relação ao tempo. Conseguimos obter, não para o Scilab, mas para a programação em c, um algoritmo desenvolvido por um estudante de engenharia, na internet. Tal implementação e seu código estão mais bem explicados no Apêndice A - Splines.

Vale ressaltar que o código gerou uma curva suave (Spline cúbica) com os pontos iniciais e finais para o período em questão. O Spline é um método de boa precisão na interpolação de pontos, para uma boa visualização de gráficos gerados por tabelas.

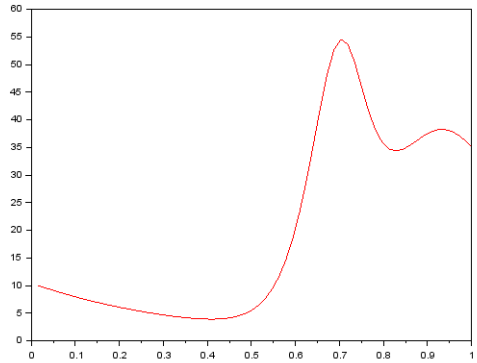
(t, x)



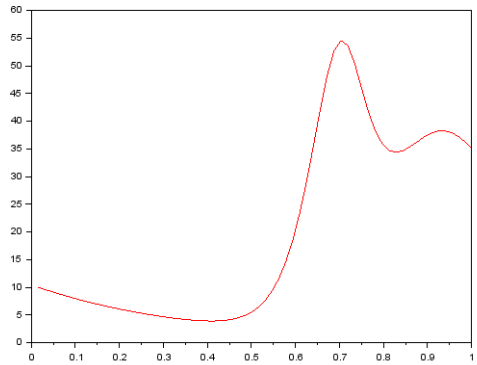
(t, y)



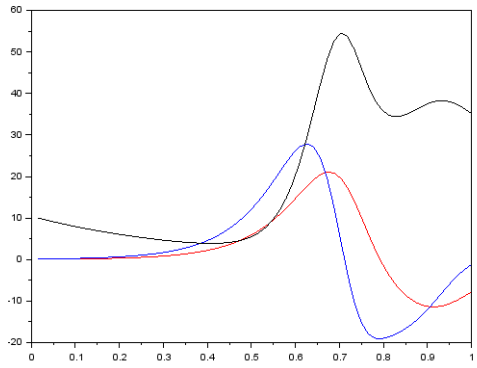
(t, z)



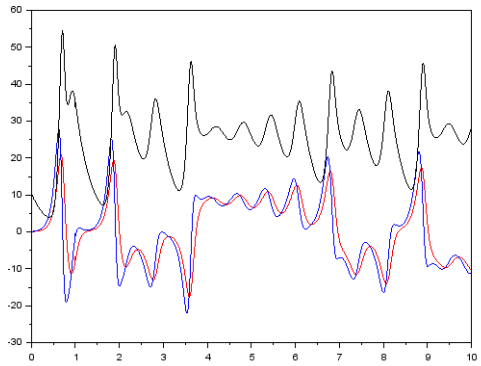
(t, z)



Colocando todos os três gráficos ao mesmo tempo, temos o seguinte: **(t, x, y, z)**



Para um melhor efeito visual (efeito caótico), aumentamos o intervalo de integração (neste caso, $0 < t < 10$). **(t, x, y, z)**



6.2 Comentários

Sabe-se o Spline gera a curva exatamente igual o previsto pelo RK. No método, temos que "amarrar as pontas"(pontos da extremidade, dados pela tabela), e aproximamos a curva através de uma função do terceiro grau, na qual devemos encontrar os coeficientes. O resultado que obtemos é bastante satisfatório. As análises encontram-se no Apêndice.

7 Conclusões

O escopo deste trabalho consistiu na resolução das três equações diferenciais de Lorentz através de métodos numéricos aprendidos na disciplina.

Para tanto, utilizamos o método de Euler explícito e o método de RK. Esta convergirá muito mais rapidamente para a solução que queremos encontrar, já que sua ordem é 4, enquanto que a do método de Euler vale 1.

O estudo destas equações, além de ter despertado o interesse na aplicação de métodos numéricos para sua resolução, tem forte aplicação em diversas áreas no estudo da teoria do caos como mencionado anteriormente, apesar do modelo ser derivado de uma simplificação de equações mais complexas.

Apesar da análise por solução manufaturada não ser possível de modo trivial, agregamos um conhecimento ao longo do aprendizado na disciplina e na entrega de tarefas passadas.

Agradecimentos Prof. Dr. Alexandre Roma, monitores Rodolfo José Ferrari da Silva e Leonardo Andrés Poveda Cuevas

8 Referências

[1] Robert C. Hilborn. Chaos and nonlinear dynamics: an introduction for scientists and enginers.

[2] Robert C. Hilborn. Chaos and nonlinear dynamics: an introduction for scientists and enginers.

[3] E. N. Lorenz. Deterministic nonperiodic

[4] J. Sotomayor. Lições de equações diferenciais ordinárias. pages 189252. IMPA, 1979.

[5] Marcelo Viana. Atratores estranhos de lorenz.

9 Apêndice

9.1 Definição Spline

Dada uma função f definida em [a, b] e um conjunto de nós $a = x0 < x1 < ... < xn = b$, um spline cúbico interpolador S para f é uma função que satisfaz as seguintes condições:

$S(x)$ é um polinômio cúbico, indicado por $S_j(x)$, no subintervalo $[xj, x_{j+1}]$ para cada $j = 0, 1, ..., n-1$;

$S(x_j) = f(x_j)$ para cada $j = 0, 1, ..., n-2$;

$S_{j+1}(x_{j+1}) = S_j j(x_{j+1})$ para cada $j = 0, 1, ..., n-2$;

$S'_{j+1}(x_{j+1}) = S' j(x_{j+1})$ para cada $j = 0, 1, ..., n-2$;

$S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$ para cada $j = 0, 1, ..., n-2$;

Um dos seguintes conjuntos de condições de contorno é satisfeito:

$S''(x_0) = S''(x_n) = 0$ (contorno livre ou natural);

$S'(x_0) = f'(x_0)$ e $S'(x_n) = f'(x_n)$ (contorno restrito).

Este trabalho utiliza a condição livre, ou seja, o método do Spline Cúbico Natural. O polinômio gerado tem a forma

$S(x) = S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)2 + d_j(x - x_j)3$

Para a montagem do gráfico na parte de análise de resultados por Spline, utilizamos a função "splin"do SciLab, cujo algoritmo é fechado ao uso do software.

A fim de propor uma melhor explicação da utilização do método, um algoritmo alternativo (adaptado do arquivo fonte de Matlab, de um estudante) encontra-se abaixo com comentários em cada etapa importante.

9.2 Algoritmo Spline

include <fstream>
include "Parametros.h"
include "Spline.h"
using namespace std;
int main()
Spline spl(nPontos); //declaracao e passagem de parametro para construtor da Classe Spline.
spl.leituraPontos(); //Le do arquivo pontos.txt, os pontos que serÃ¿o interpolados.
20
spl.exibePontos(); //Exibe na tela os pontos

spl.passo1(); // gera os h's, pela equacao hi = xi+1 - xi
/* spl.passo2(); // passo de 2 a 6 Ã c
a resoluÃÃ¿o do sistema linear.
spl.passo3e4();
spl.passo5e6();*/
spl.preparandoThomas(); // prepara o sistema e utiliza o algoritmo de thomas pra resolver-lo.
spl.saidaSpline(); // saida na tela e em arquivo .txt, dos coeficientes das do spline Cubico.
spl.splineCubicoGNU(); // gera um arquivo .gnu para desenhar o grÃfico da solucao.
system("PAUSE");
return EXIT$_SUCCESS$;

Spline.h

```
// Classe Spline
include "Parametros.h"
ifndef SPLINE
define SPLINE
class Spline
private: //Atributos Privados
int n;
double* x;
double* y;
double* h;
double* a;
double* b;
double* c;
double* d;
double* alfa;
double* beta;
double* gama;
double* delta;
public: //Metodos Publicos
Spline(int ns); // Construtor da Classe Spline
void leituraPontos(void); // Le os pontos do arquivo "ponto.txt"
void exibePontos(void); // Exibe na tela esses pontos
21
//* Gera os h's, espaÃos entre os x's
void passo1(void);
//* Metodos usados para resolver o Sistema Linear
void passo2(void);
void passo3e4(void);
void passo5e6(void);
```

```
//*************************
    //* Metodos usados para resolver o Sistema Linear, usando algorimto de Thomas
    void algoritmoThomas(double *A,double *B, double *C, double *D, double *X, long
    int N);
    void preparandoThomas(void);
    //*************************
    //* Exibe os coeficientes dos polinomios
    void saidaSpline(void);
    //* Gera um arquivo .gnu para ser plotado
    void splineCubicoGNU(void);
    ;
    endif
    Spline.cpp
    include "Spline.h"
    include <fstream>
    include <iostream>
    using std::cout;
    using std::cin;
    using std::endl;
    using namespace std;
    Spline::Spline(int ns) // Construtor
    n = ns - 1; // sao ns pontos(qntdade de pontos), mas a referencia sera de ns-1 para os
    metodos
    x = new double[n+1]; // vetor x, armazena os valores do eixo x
    y = new double[n+1]; // vetor y, armazena os valores do eixo y, os f(x).
    22
    h = new double[n+1]; // vetor h, que armazena os espaÃos entres os x's.
    a = new double[n+1]; // equivale aos f(x), aj = f(xj), sao termos independentes dos
    polinomios b =
    new double[n+1]; // sao os valores que multiplicam o termo linear, bj*(x-xj)
    c = new double[n+1]; // sao os valores que multiplicam o termo quadratico, cj*(x-xj)^2
    d = newdouble[n + 1]; //saoosvaloresquemultiplicamotermocubico, dj * (x - xj)^3
    //Nota : S(x) = Sj(x) = aj + bj(x - xj) + cj(x - xj)^2 + dj(x - xj)^3
    //Saovariaveisauxiliaresusadasnoalgoritmo1, pararesolverosistemalinear
    alfa = newdouble[n + 1];
    beta = newdouble[n + 1];
    gama = newdouble[n + 1];
    delta = newdouble[n + 1];

    voidSpline :: leituraPontos(void)//classeifstream, deleituraifstreamarq("pontos.txt"); //Leituradospo
    voidSpline :: exibePontos(void)//Exibenatelaospontosqforamlidos.cout << "jxjf(xj)";
```

```
cout « -————————————";
        for (int i=0; i <= n; i++)
        printf("

        cout « "Enter pra executar o metodo Spline Cubica.";
        getchar();
        cout « "";

        //Nota: S(x) = Sj(x) = aj + bj(x - xj) + cj(x - xj)^2 + dj(x − xj)^3
        voidSpline :: passo1(void)//Calcuraosh's, distanciaentresosx's
for (int i=0; i < n; i++)
h[i] = x[i+1] - x[i];
23
```

```
/*** ALGORITMO 1 - Resolucao do Sistema Linear ***/
// Consiste nos Passos de 2 a 6
void Spline::passo2(void)
for (int i=1; i < n; i++)
alfa[i] = (3.0/h[i])*(a[i+1] - a[i]) - (3.0/h[i-1])*(a[i] - a[i-1]);


void Spline::passo3e4(void)
beta[0] = 1.0;
gama[0] = 0.0;
delta[0] = 0.0;
for (int i=1; i < n; i++)
beta[i] = 2.0*(x[i+1] - x[i-1]) - h[i-1]*gama[i-1];
gama[i] = h[i]/beta[i];
delta[i] = (alfa[i] - h[i-1]*delta[i-1])/beta[i];


void Spline::passo5e6(void)
beta[n] = 1.0;
gama[n] = 0.0;
delta[n] = 0.0;
for (int j=n-1; j >=0 ; j–)
c[j] = delta[j] - gama[j]*c[j+1];
b[j] = (a[j+1] - a[j])/h[j] - h[j]*(c[j+1] + 2.0*c[j])/3.0;
d[j] = (c[j+1] - c[j])/(3*h[j]);
```

9.3 O Atrator de Lorentz

O Atractor de Lorenz foi introduzido por Edward Lorenz em 1963, que o derivou a partir das equações simplicadas de rolos de convecção que ocorrem nas equações da atmosfera. É um mapa caótico que mostra como o estado de um sistema dinâmico evolui no tempo num padrão complexo, não-repetitivo e cuja forma é conhecida por se assemelhar a uma borboleta. Trata-se de um sistema não-linear, tridimensional e determinístico que exibe comportamento caótico e demonstra aquilo a que hoje se chama um atractor estranho. As equações do modelo de Lorenz introduzidas são desenvolvidas a partir da equação de escoamento Navier-Stokes e a equação que descreve a difusão da energia térmica. A dedução do modelo não está no escopo do trabalho, uma vez que ela é derivadade várias simplicações e conjecturas modeladas.

$$\rho\frac{\partial v_z}{\partial t}+\rho\vec{v}\cdot \boldsymbol{grad}\, v_z = -\rho g - \frac{\partial p}{\partial z}+\mu\nabla^2 v_z$$

$$\rho\frac{\partial v_x}{\partial t}+\rho\vec{v}\cdot \boldsymbol{grad}\, v_x = -\frac{\partial p}{\partial x}+\mu\nabla^2 v_x$$

Eq. Navier Stokes