Universidade de São Paulo
Escola Politécnica
Curso de Engenharia de XXX

O Efeito Borboleta

Rafael Ribeiro Correia, rafael.correia.poli@gmail.com

Fabiano Shimura, fabianoshimura@hotmail.com

.

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

São Paulo - SP

31/03/2016

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.

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.

3 Metodologias

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

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

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

4 Análise

4.1 Euler

4.1.1 Equações

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

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

4.1.2 Tabela

Tabela 1: $0 < t < 0.25$

t	x	y	z
0.015625	0.1	0.1	10.0
0.03125	0.1	0.1265625	9.583489583333334
0.046875	0.104150390625	0.15336075846354166	9.184375271267362
0.0625	0.11183951059977214	0.18158410075865686	8.80194253990571
0.078125	0.12273710281209788	0.21229530779060363	8.43551225090383
0.09375	0.1367305723399894	0.2464983271114824	8.084439706767506
0.109375	0.15388178402303518	0.2851946963341975	7.748114675089904
0.125	0.17439942657165428	0.3294322205106489	7.425962286575729
0.140625	0.1986233006246222	0.3803489104946197	7.117444891150979
0.15625	0.2270179271668093	0.4392147403493156	6.8220650960403875
0.171875	0.2601736792265759	0.5074734300871471	6.539370346516859
0.1875	0.29881426529854016	0.5867862042336864	6.268959570038273
0.203125	0.3438098807571568	0.6790793085464708	6.0104926101695755
0.21875	0.3961957288492371	0.7865969742505293	5.7637034520806765
0.234375	0.457195923443189	0.911961485867094	5.528418615976125
0.25	0.5282530425719242	1.058242015808195	5.304582607086634

Tabela 2: $0.25 < t < 0.5$

t	x	y	z
0.265625	0.6110638196400915	1.2290339107212898	5.092293012488672
0.28125	0.7076216463715287	1.4285501141629053	4.891848795655069
0.296875	0.8202667194639313	1.6617263311263097	4.703816652873215
0.3125	0.9517447837861779	1.9343412929864916	4.529122086184953
0.328125	1.105275488348727	2.2531529231797602	4.369174278984467
0.34375	1.284631337541076	2.6260501031125623	4.226037142841758
0.359375	1.4942280196616209	3.062217721873976	4.102663297146446
0.375	1.7392264106323014	3.572309165230505	4.00321354813506
0.390625	2.0256455910382707	4.168614409014198	3.9334919593655684
0.40625	2.3604844688470092	4.865201932376449	3.901536076658017
0.421875	2.751846572523484	5.677996430528672	3.9184132234511058
0.4375	3.2090574878367946	6.624728236801238	3.9992862404346674
0.453125	3.742756042362489	7.724649205031856	4.164823278643497
0.46875	4.364926849029578	8.997846118931902	4.443030811105178
0.484375	5.088820484951816	10.463886762298115	4.871575466496042
0.5	5.928674590787175	12.139394790242786	5.500606300832646

Tabela 3: $0.5 < t < 0.75$

t	**x**	**y**	**z**
0.515625	6.899099621952114	14.033960242463532	6.395953769138265
0.53125	8.013921593907023	16.143561913424637	7.642294598600282
0.546875	9.28417789383165	18.440460241631673	9.345322522938321
0.5625	10.714847010675404	20.85847405424829	11.63100460522378
0.578125	12.299788736233667	23.273049168611244	14.63849438783028
0.59375	14.014360678792663	25.47727802890932	18.501269434512825
0.609375	15.805441514748392	27.159174249323385	23.3092545113719
0.625	17.579462254525733	27.893238768569603	29.04526588881278
0.640625	19.190989834845087	27.170294221586836	35.4967361344124
0.65625	20.437756145273486	24.497792953280413	42.164968590523706
0.671875	21.07213689652457	19.591574721845067	48.23121862544789
0.6875	20.840799056730898	12.624284359645058	52.67215239976062
0.703125	19.556968635311236	4.392852253081042	54.588419595203824
0.71875	17.18757545058777	-3.800612439715472	53.656259514149426
0.734375	13.908171092727887	-10.631366873146536	50.39990526773614
0.75	10.073868285560009	-15.333091061450041	45.98955188020628

Tabela 4: $0.75 < t < 1$

t	**x**	**y**	**z**
0.765625	6.104030887589688	-17.925142391074985	41.659827743058955
0.78125	2.349472562798333	-18.947874704695476	38.21438248054246
0.796875	-0.9782379477475747	-19.026789340309023	35.92653042288409
0.8125	-3.798324102835301	-18.608338993528555	34.720415520209066
0.828125	-6.112388929506122	-17.91873500532532	34.37811439196363
0.84375	-7.957130503852872	-17.02960608648333	34.65704104454802
0.859375	-9.37470481363888	-15.935847485741814	35.33029138736928
0.875	-10.399883356154964	-14.61311052573731	36.19247507123929
0.890625	-11.058200101402205	-13.053518405974291	37.05905910376477
0.90625	-11.369968586491593	-11.284293301195628	37.77037568111729
0.921875	-11.356581823164099	-9.372212709418744	38.20132972078977
0.9375	-11.046524149141387	-7.415580703346188	38.272674844381015
0.953125	-10.479189235735888	-5.526628649469965	37.957923673564416
0.96875	-9.705351644131838	-3.8097912241611245	37.28125894945916
0.984375	-8.784170328511413	-2.3427963330844386	36.30561321491729
1.0	-7.777705641725948	-1.166222622214858	35.11443436275655

Tabela 5: $1 < t < 1.25$

t	x	y	z
1.015625	-6.74466141992734	-0.28340701435367965	33.79306006016735
1.03125	-5.735090419056455	0.33152479354454456	32.414882834053834
1.046875	-4.787181792087549	0.7219658474397697	31.034554518044242
1.0625	-3.926377473411405	0.9376689456966233	29.6874452398022
1.078125	-3.166370220425776	1.0265420399384677	28.392942695030747
1.09375	-2.511227679743863	1.0299429775625417	27.15911572630119
1.109375	-1.9579197645397373	0.980855558169787	25.987073071343648
1.125	-1.498736120366374	0.9039492304141569	24.874271539982985
1.140625	-1.123316534306916	0.8166274901657378	23.81667510035975
1.15625	-0.8202002804830639	0.7304427164421525	22.809980338437374
1.171875	-0.5779123122134988	0.6525161805270208	21.85020340786338
1.1875	-0.38565786022279264	0.5867886907030928	20.93388612314024
1.203125	-0.233713086640623	0.535040393069602	20.058104935652217
1.21875	-0.11359535543590034	0.4976784367831294	19.22039672049366
1.234375	-0.01808382540167694	0.47431905253461615	18.418663514031646
1.25	0.05885412427586886	0.4642005170867789	17.651085177463703

Tabela 6: $1.25 < t < 1.5$

t	x	y	z
1.265625	0.12218949815257354	0.46646420149301565	16.91605017186503
1.28125	0.17598242054951763	0.4803372962673141	16.212105326579298
1.296875	0.22353786988042332	0.5052454984789395	15.537921733598036
1.3125	0.26755468684894146	0.5408783255332469	14.892273039007963
1.328125	0.31026150539336417	0.5872245045460032	14.274022826929684
1.34375	0.353536974010964	0.6445904082360929	13.682118643913618
1.359375	0.3990140731086404	0.7136108776211348	13.115591094308952
1.375	0.44816982381371767	0.7952589673969657	12.573557206778801
1.390625	0.5024025024986002	0.8908590796049871	12.055227912818818
1.40625	0.5630988426714731	1.002106490835685	11.559920028393563
1.421875	0.6316937876971312	1.131095295751347	11.087073647083777
1.4375	0.7097252733306024	1.280356150.265625	0.6110638196400915
1.453125	0.7988863491316097	1.452904866582673	10.20729667112974
1.46875	0.9010767424833384	1.652302412473303	9.80012867229881
1.484375	1.0184557534192704	1.8827270743031093	9.415053278798268
1.5	1.1534981473073702	2.1490586186921	9.052719926883896

Tabela 7: $1.5 < t < 1.75$

t	x	y	z
1.515625	1.3090544709612342	2.4569741474758686	8.714256624749925
1.53125	1.4884169204166458	2.8130540606587786	8.401417489231013
1.546875	1.695391473579479	3.224894806979896	8.116780259397398
1.5625	1.9343763694232943	3.701222094613411	7.864010131281014
1.578125	2.21044601398425	4.251993358400652	7.648211113693382
1.59375	2.5294377865493125	4.888470503131664	7.476391928287891
1.609375	2.898036648515305	5.623231741722061	7.358080004334014
1.625	3.3238483818288604	6.470072505807043	7.306123519839968
1.640625	3.8154459012004516	7.443716808096719	7.3377257694623985
1.65625	4.382363230402993	8.559217944257464	7.47575436425276
1.671875	5.034996779442754	9.830866092094968	7.750352130324302
1.6875	5.7843513595446625	11.270335556416406	8.200832965198776
1.703125	6.641536390305872	12.883694981403718	8.87775004162198
1.71875	7.61687367016491	14.666779731117606	9.84483643145951
1.734375	8.718421492188769	16.598323600361912	11.18018191910794
1.75	9.949656196590823	18.630260160530547	12.975453212244577

Tabela 8: $1.75 < t < 2.0$

t	x	y	z
1.765625	11.306000565956404	20.674929143143498	15.331132507312333
1.78125	12.769895656141887	22.589918110318653	18.344690953780916
1.796875	14.304274164607007	24.163470781220244	22.087686598668572
1.8125	15.844773635952825	25.10734392275938	26.568005555334125
1.828125	17.29205024326635	25.06956710871115	31.676945652864028
1.84375	18.5072872534921	23.684387482464345	37.13057626140257
1.859375	19.31620916426901	20.673972089618132	42.43243395719278
1.875	19.528359621354813	15.995005134051686	46.90414664344501
1.890625	18.9762729827137	9.97684921026929	49.83037301876779
1.90625	17.57011301826926	3.3481622270056253	50.71228582196665
1.921875	15.347933207134318	-2.93942513271616	49.51845498665537
1.9375	12.490533466532682	-8.053868644016111	46.75027820693588
1.953125	9.280470636759432	-11.587417219117906	43.230520013420715
1.96875	6.019863159278598	-13.614901227680562	39.748987639040095
1.984375	2.951931223816229	-14.507282424858634	36.812157284532205

4.2 Runge-Kutta

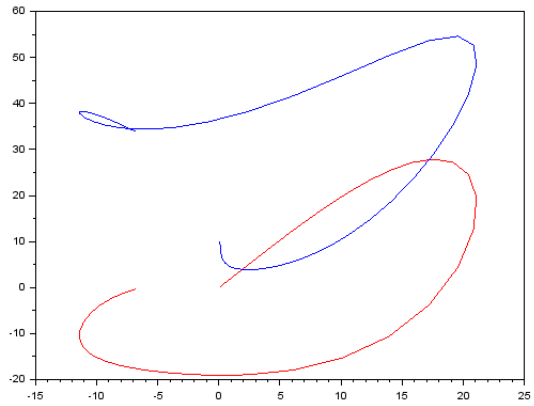
4.2.1 Equações

4.2.2 Tabela

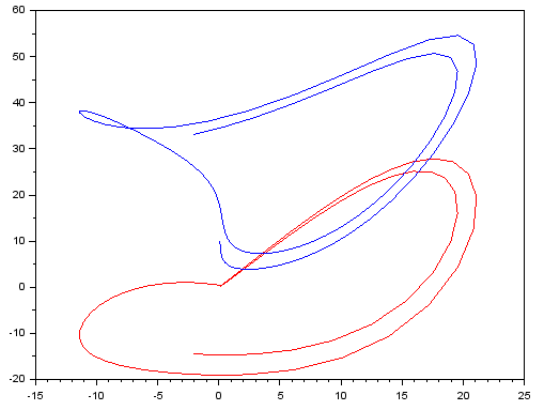
4.3 Gráficos

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

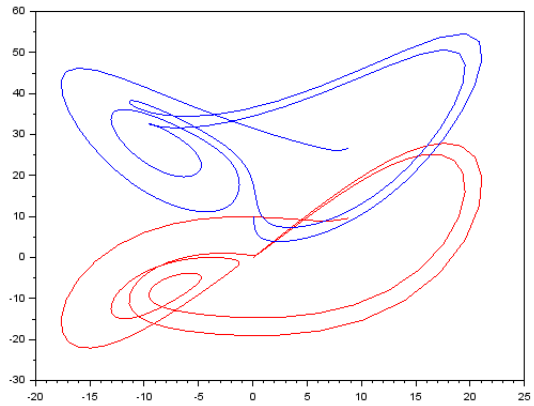
É importante notar que apresentaremos apenas um gráfico para cada intervalo de tempo (apesar de termos duas metodologias) pois os resultados foram extremamente parecidos. $0 < t < 1$



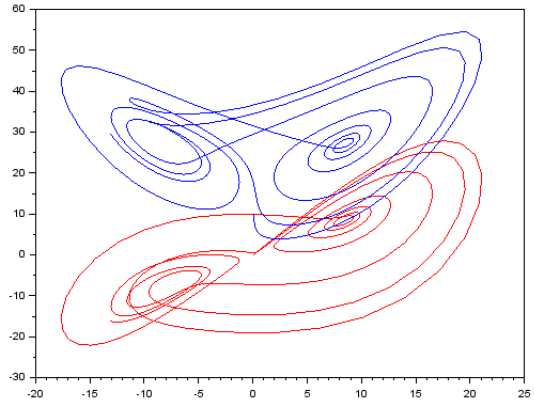
$0 < t < 2$



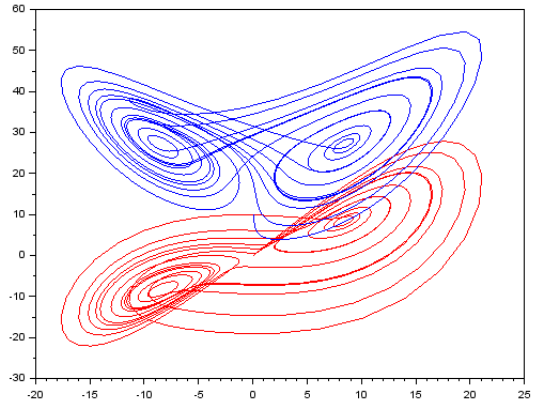
$0 < t < 4$


60
50
40
30
20
10
0
-10
-20
-30
-20
-15
-10
-5
0
5
10
15
20
25

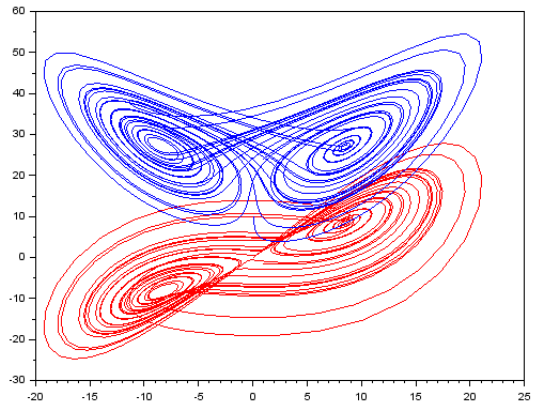
$0 < t < 8$


60
50
40
30
20
10
0
-10
-20
-30
-20
-15
-10
-5
0
5
10
15
20
25

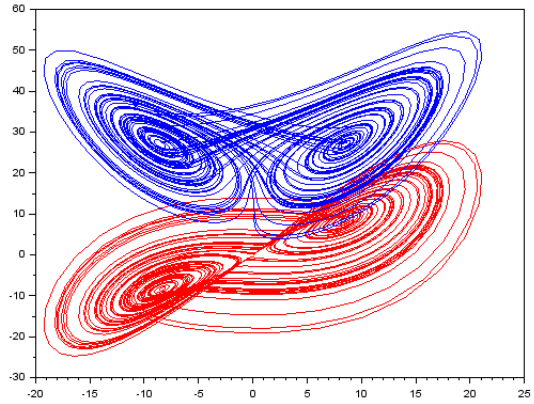
$0 < t < 16$


60
50
40
30
20
10
0
-10
-20
-30
-20
-15
-10
-5
0
5
10
15
20
25

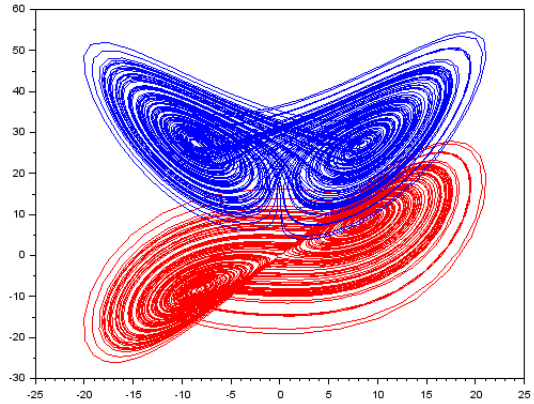
$0 < t < 32$



$0 < t < 64$



$0 < t < 128$



5 Spline

6 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 pela sua facilidade do algoritmo e pela obtenção de uma boa aproximação inicial, na qual podemos comparar com o gráfico da solução manufaturada.

Em seguida, implementamos o método de Runge-Kutta Clássico, e já observamos uma aproximação muito mais consistente e próxima do esperado, já que este possui convergência de quarta ordem.

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.

Agradecimentos Agradecimentos ao Prof. Dr. Alexandre Roma

Referências

[1] Richard L. Burden. Numerical analysis. NINTH EDITION.

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

[3] E. N. Lorenz. Deterministic nonperiodic flow.

[4] J. Sotomayor. Lições de equações diferenciais ordinárias. pages 189–252. IMPA, 1979.

[5] Marcelo Viana. Atratores estranhos de lorenz.

[5], [1], [2], [3], [4].

7 Apêndice

7.1 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) encontra-se abaixo com comentários em cada etapa importante.

7.2 Algoritmo Spline Cúbico

```
#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 Spl
    spl.leituraPontos(); //Le do arquivo pontos.txt, os pontos que serÃ£o interpolados.
```

```
    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© a resoluÃ§Ã£o do sistema linear.
    spl.passo3e4();
    spl.passo5e6();*/
    spl.preparandoThomas(); // prepara o sistema e utiliza o algoritmo de thomas pra resolv
    spl.saidaSpline(); // saida na tela e em arquivo .txt, dos coeficientes das do spline C
    spl.splineCubicoGNU(); // gera um arquivo .gnu para desenhar o grÃ¡fico 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
```

```
        //* Gera os h's, espaÃ§os 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
   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).
```

```
   h = new double[n+1];  // vetor h, que armazena os espaÃ§os entres os x's.
   a = new double[n+1];  // equivale aos f(x), aj = f(xj), sao termos independentes dos pol
   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 = new double[n+1];  // sao os valores que multiplicam o termo cubico, dj*(x-xj)^3
   //Nota: S(x) = Sj(x) = aj + bj(x - xj) + cj(x - xj)^2 + dj(x - xj)^3

   // Sao variaveis auxiliares usadas no algoritmo 1, para resolver o sistema linear
   alfa = new double[n+1];
   beta = new double[n+1];
   gama = new double[n+1];
   delta = new double[n+1];
}

void Spline::leituraPontos(void){
     // classe ifstream, de leitura
     ifstream arq("pontos.txt");  //Leitura dos pontos do problema, no arquivo pontos.txt
     for (int i=0; i <= n; i++){
         if (! arq.eof()){
            arq >> x[i];
            arq >> y[i];
            a[i] = y[i]; // ai = f(xi)
         }
     }
}

void Spline::exibePontos(void){   // Exibe na tela os pontos q foram lidos.
     cout << "j        xj      f(xj) \n";
     cout << "----------------------\n";
     for (int i=0; i <= n; i++){
         printf("%d\t%0.2f\t%0.2f\n",i,x[i],y[i]);
     }
     cout << "\nPrecione Enter pra executar o metodo Spline Cubica.";
     getchar();
     cout << "\n\n";
}

//Nota: S(x) = Sj(x) = aj + bj(x - xj) + cj(x - xj)^2 + dj(x - xj)^3
void Spline::passo1(void){ // Calcura os h's, distancia entres os x's
     for (int i=0; i < n; i++){
         h[i] = x[i+1] - x[i];
```

```
     }
}

/*** 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]);
     }
}
```