

PGF 5005 - Mecânica Clássica  
Prof. Iberê L. Caldas  
Primeiro Estudo Dirigido  
2º Semestre de 2015

Aluno: Rafael M. Miller  
NUSP.: 7581818

26 de agosto de 2015

## Introdução

### Exercício 2.1.1

A expressão para  $q^{(n+1)}$  de acordo com a equação do método de Euler (equação (2) encontrada em [1]) é:

$$q^{(n+1)} = q^n + \Delta t \frac{\partial H}{\partial p} \Big|_{(q^{(n)}, p^{(n)})} = q^n + \Delta t p \quad (1)$$

Para  $p^{(n+1)}$  temos:

$$p^{(n+1)} = p^n - \Delta t \frac{\partial H}{\partial q} \Big|_{(q^{(n)}, p^{(n)})} = p^n + \Delta t \operatorname{sen}(q) \quad (2)$$

### Exercício 2.1.2

O programa solicitado está disposto em Anexos: Programa 1 - Equações de Euler para o pêndulo simples.

### Exercício 2.1.3

Segue abaixo a Figura 1 solicitada no exercício.

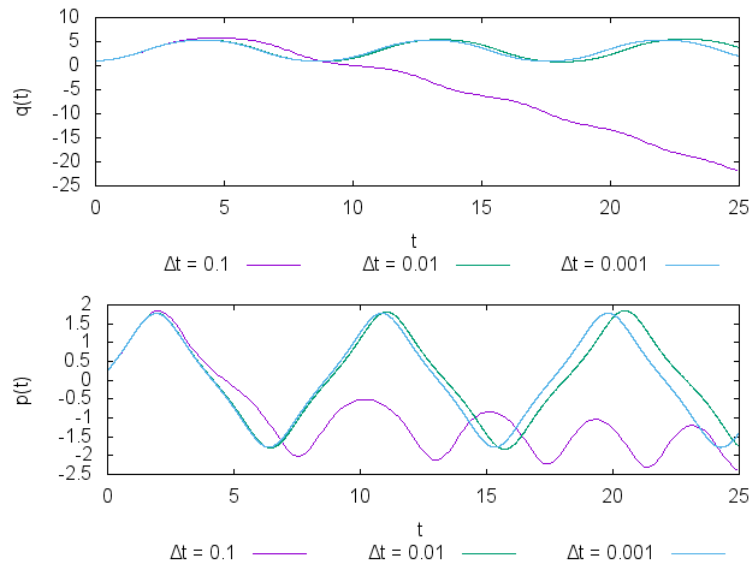


Figura 1: Libração de um pêndulo simples

Podemos observar uma grande diferença entre os valores de  $\Delta t$ . Ao utilizar um valor “alto” de  $\Delta t$  os valores de  $q(t)$  e  $p(t)$  diminuem com o passar do tempo, o que não é esperado. Ao observarmos o gráfico da Figura 5, veremos que para  $\Delta t = 0.1$  o movimento se transforma para um movimento de rotação! Escolhi manter esse gráfico pois demonstra não somente uma espiral crescente, mas uma mudança no próprio sistema físico que estamos estudando. Isso ocorre devido ao erro intrínseco do método de Euler feito dessa forma e que pode ser contornado ao diminuirmos  $\Delta t$ . Veremos que o método de Euler feito no Programa 1 não é simplético, conforme discutido no Exercício 2.2.1 e que ao alterar o método para o que é descrito na equação (3) de [1] esse erro deixará de ocorrer. Os gráficos das Figuras 1 a 6 terão esse problema pois todos foram feitos a partir dos dados gerados pelo Programa 1.

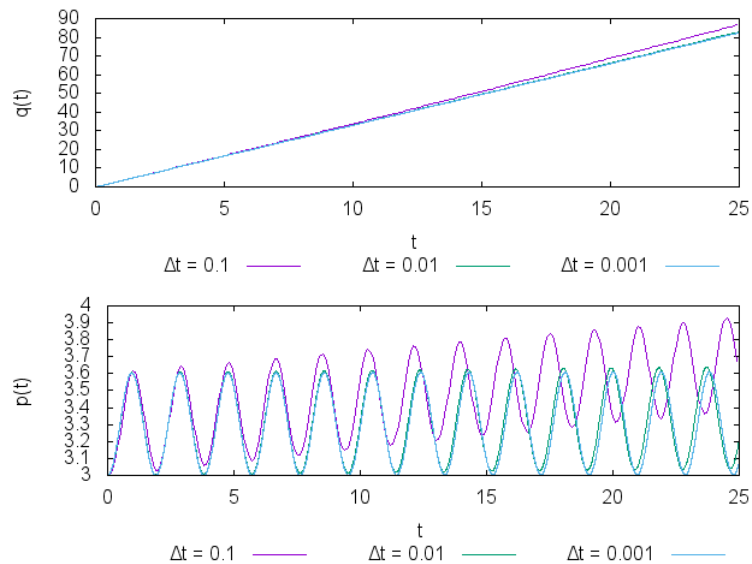


Figura 2: Rotação de um pêndulo simples

## Exercício 2.1.4

Assim como no gráfico de  $p(t)$  x  $t$  do exercício anterior, não é esperado que a energia do sistema aumente com o passar do tempo. Isso também ocorre pelo método de Euler do Programa 1 não ser simplético.

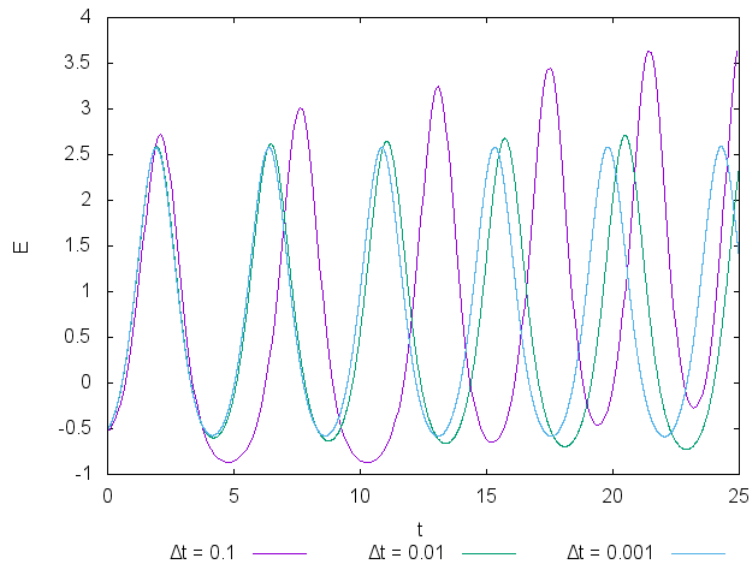


Figura 3: Energia de libração do pêndulo simples

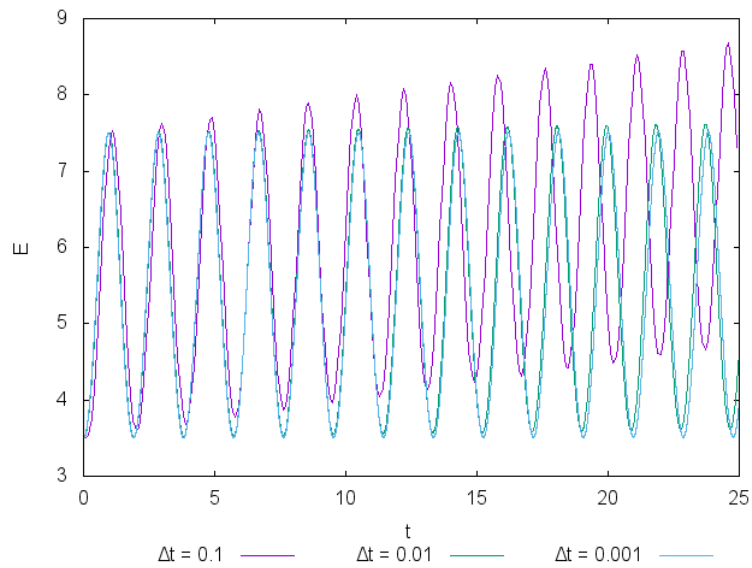


Figura 4: Energia de rotação do pêndulo simples

### Exercício 2.1.5

Observamos das figuras de trajetórias ( $p(t) \times q(t)$ ) que este método só é capaz de reproduzir as órbitas esperadas no movimento de libração para  $\Delta t = 0.01$  e  $\Delta t = 0.001$ . Para o caso de  $\Delta t = 0.1$  as linhas se abrem e o movimento de libração (com órbita fechada) se torna um movimento de rotação devido ao aumento de energia pelo fato do método não ser simplético. Na Figura 11 temos os gráficos de libração como o esperado, pois o método sofreu a modificação necessária.

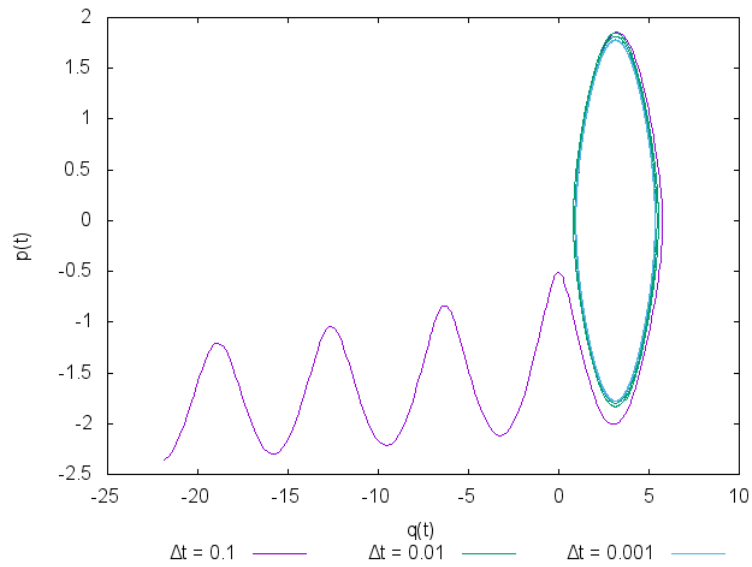


Figura 5: Trajetória de libração do pêndulo simples

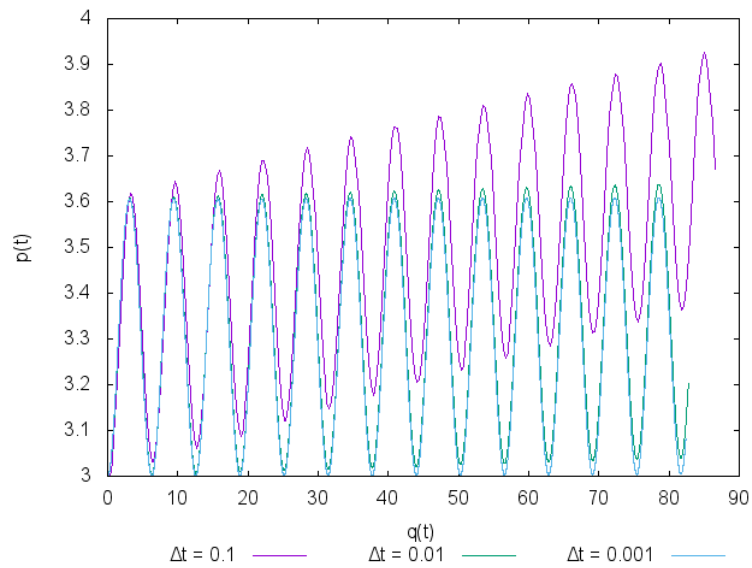


Figura 6: Trajetória de rotação do pêndulo simples

## Exercício 2.1.6

O programa solicitado está disposto em Anexos: Programa 2 - Equações modificadas de Euler para o pêndulo simples.

## Exercício 2.1.7

Seguem abaixo as Figura 7 e 8 solicitadas no exercício. Desta vez observamos o comportamento esperado para ambos os gráficos graças as modificações realizadas do Programa 1 para o Programa 2. Particularmente nessas duas Figuras solicitadas, não é possível notar uma distinção entre as curvas para os diferentes  $\Delta t$ . Ou seja, a diminuição no  $\Delta t$  que servia para “contornar” o problema do método não ser simplético não se vê mais necessária.

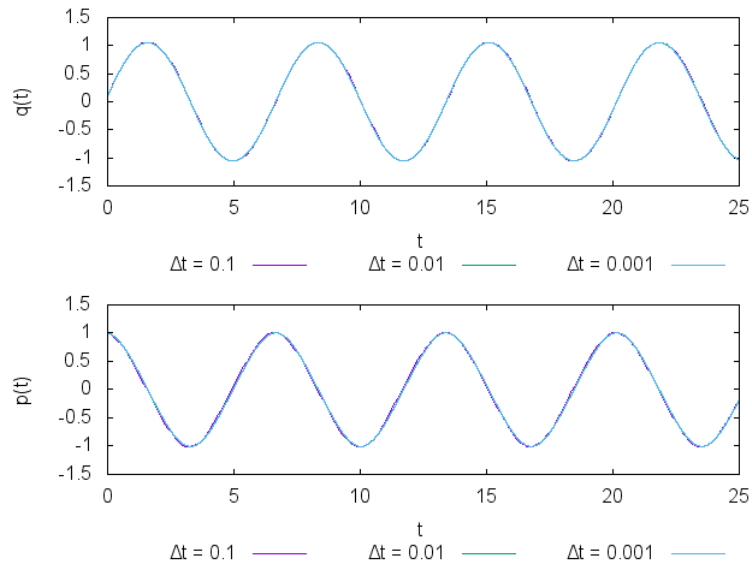


Figura 7: Libração de um pêndulo simples (versão modificada)

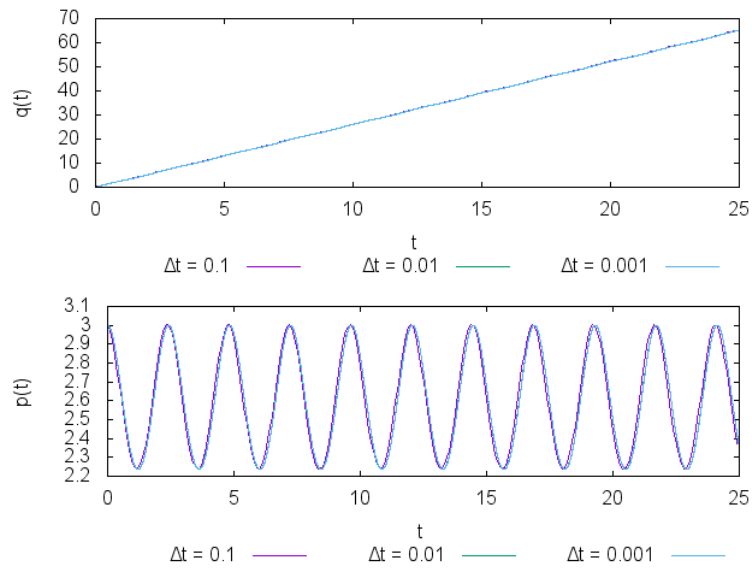


Figura 8: Rotação de um pêndulo simples (versão modificada)

## Exercício 2.1.8

Nos gráficos de energia para os dois movimentos do pêndulo, se percebe uma variação para cada  $\Delta t$ . Ao diminuir esse intervalo, observamos que a energia ( $E$ ) se torna “mais constante”. Como a energia é uma constante desse movimento, a diminuição de sua variação no tempo implica na melhor representação do sistema. Apesar de não observarmos mudanças entre  $\Delta t$  nos gráficos das Figuras 7 e 8, nos gráficos das Figuras 9 e 10 essa mudança fica nítida e a importância na diminuição de  $\Delta t$  para melhor representar o movimento se torna mais clara.

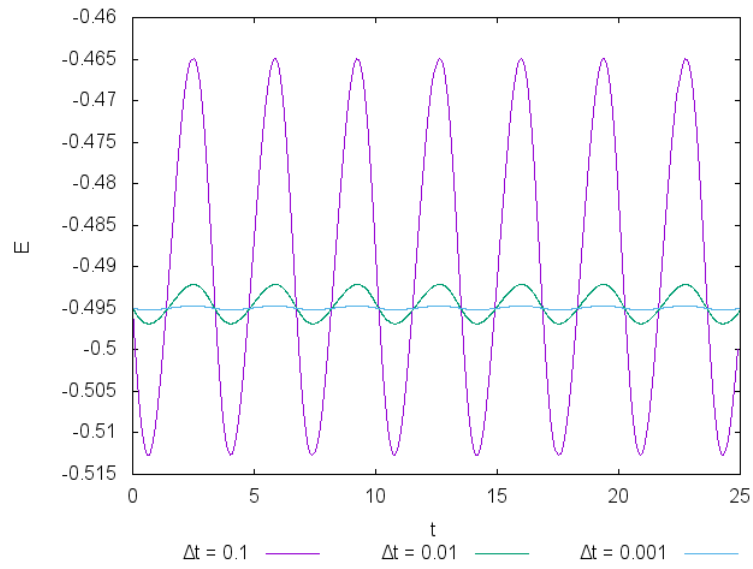


Figura 9: Energia de libração do pêndulo simples (versão modificada)

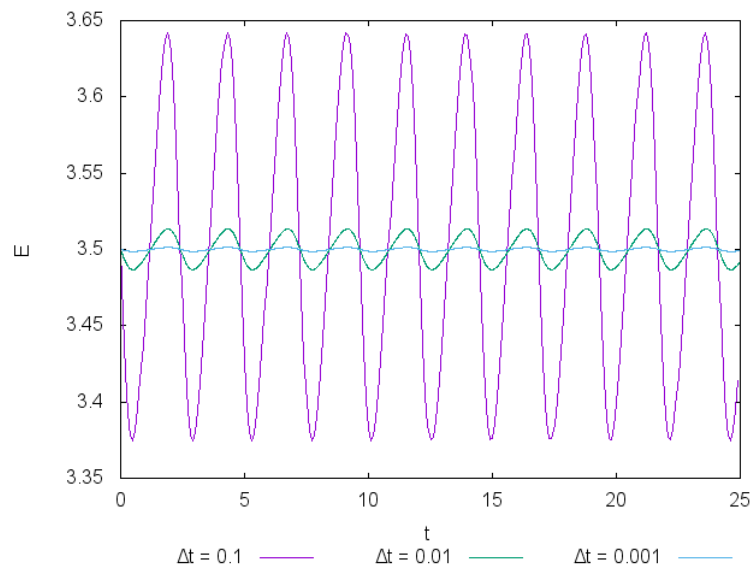


Figura 10: Energia de rotação do pêndulo simples (versão modificada)

## Exercício 2.1.9

Seguem abaixo as Figura 11 e 12 solicitadas no exercício. Nestes gráficos observamos o comportamento do pêndulo para os dois tipos de movimento. Como era esperado, a libração possui uma órbita fechada nesse gráfico de trajetória, assim como as trajetórias abertas no caso do movimento de rotação.

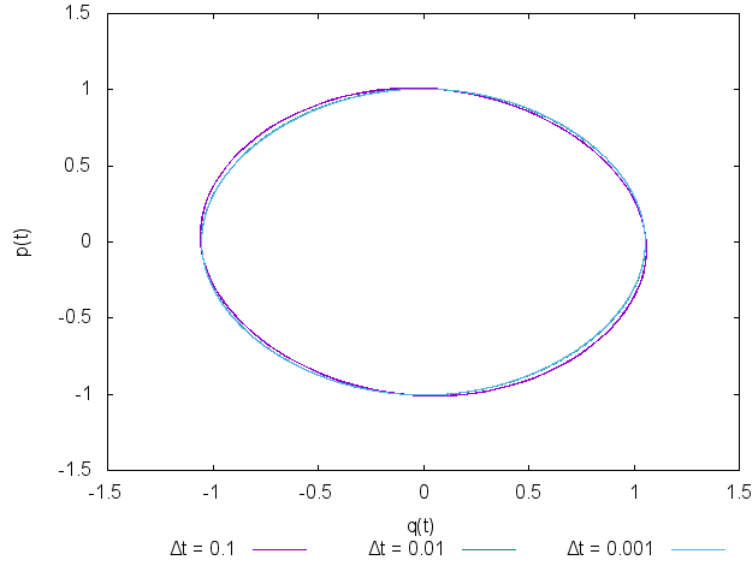


Figura 11: Trajetória de libração do simples (versão modificada)

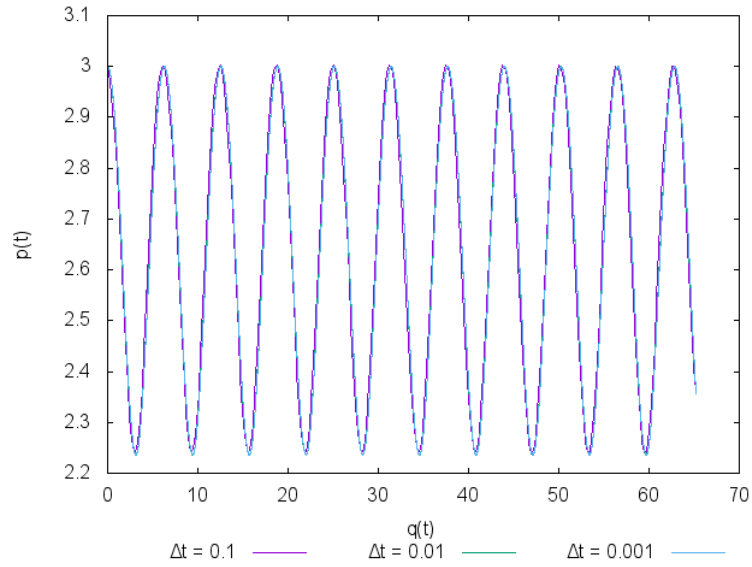


Figura 12: Trajetória de rotação do pêndulo simples (versão modificada)

## Exercício 2.2.1

Para verificar que um método é simplético, precisamos calcular o Jacobiano:

$$|J| = \begin{vmatrix} \frac{\partial p^{(n+1)}}{\partial p^{(n)}} & \frac{\partial q^{(n+1)}}{\partial p^{(n)}} \\ \frac{\partial p^{(n+1)}}{\partial q^{(n)}} & \frac{\partial q^{(n+1)}}{\partial q^{(n)}} \end{vmatrix} \quad (3)$$

Para o método de Euler, utilizando as equações (1) e (2), obtemos o seguinte Jacobiano:

$$|J| = \begin{vmatrix} 1 & \Delta t \\ \Delta t \cos(q^{(n)}) & 1 \end{vmatrix} = 1 - (\Delta t)^2 \cos(q^{(n)}) \quad (4)$$

Observamos que esse Jacobiano é diferente de 1, o que mostra que esse método não é simplético e, por isso, carrega o erro que vemos nos gráficos das Figuras 1 a 6.

Utilizando as equações para o método de Euler modificado, temos o seguinte Jacobiano:

$$|J| = \begin{vmatrix} 1 & \Delta t \\ 0 & 1 \end{vmatrix} = 1 \quad (5)$$

Isso implica que o método de integração modificado é simplético. Por isso temos uma melhor coerência nos gráficos das Figuras 7 a 12.

### Exercício 2.3.1

A expressão para  $q^{(n+1)}$  de acordo com a equação do método de Euler (equação (2) encontrada em [1]) é:

$$q_1^{(n+1)} = q_1^n + \Delta t \frac{\partial H}{\partial p_1} \Big|_{(q_1^{(n)}, p_1^{(n)}, q_2^{(n)}, p_2^{(n)})} = q_1^n + \Delta t p_1^n \quad (6)$$

$$q_2^{(n+1)} = q_2^n + \Delta t \frac{\partial H}{\partial p_2} \Big|_{(q_1^{(n)}, p_1^{(n)}, q_2^{(n)}, p_2^{(n)})} = q_2^n + \Delta t p_2^n \quad (7)$$

Para  $p^{(n+1)}$  temos:

$$p_1^{(n+1)} = p_1^n - \Delta t \frac{\partial H}{\partial q_1} \Big|_{(q_1^{(n+1)}, p_1^{(n)}, q_2^{(n+1)}, p_2^{(n)})} = p_1^n + \Delta t (q_1^{(n+1)} + 2q_1^{(n+1)} q_2^{(n+1)}) \quad (8)$$

$$p_2^{(n+1)} = p_2^n - \Delta t \frac{\partial H}{\partial q_2} \Big|_{(q_1^{(n+1)}, p_1^{(n)}, q_2^{(n+1)}, p_2^{(n)})} = p_2^n + \Delta t (q_2^{(n+1)} + (q_1^{(n+1)})^2 - (q_2^{(n+1)})^2) \quad (9)$$

### Exercício 2.3.2

O programa solicitado está disposto em Anexos: Programa 3 - Método de Euler simplético para a hamiltoniana de Hénon-Heiles.

### Exercício 2.3.3

Segue abaixo o mapa de Poincaré para a Hamiltoniana de Hénon-Heiles com  $E = 0.08333$ . Nessa figura podemos observar quatro regiões distintas que possuem órbitas fechadas, a separatriz dessas regiões (representada em azul claro) e em roxo a “curva limite” para o movimento desse sistema. Na legenda do gráfico estão dispostas as condições iniciais de  $q_2(t)$  e  $p_2(t)$  utilizadas para obter as respectivas curvas.



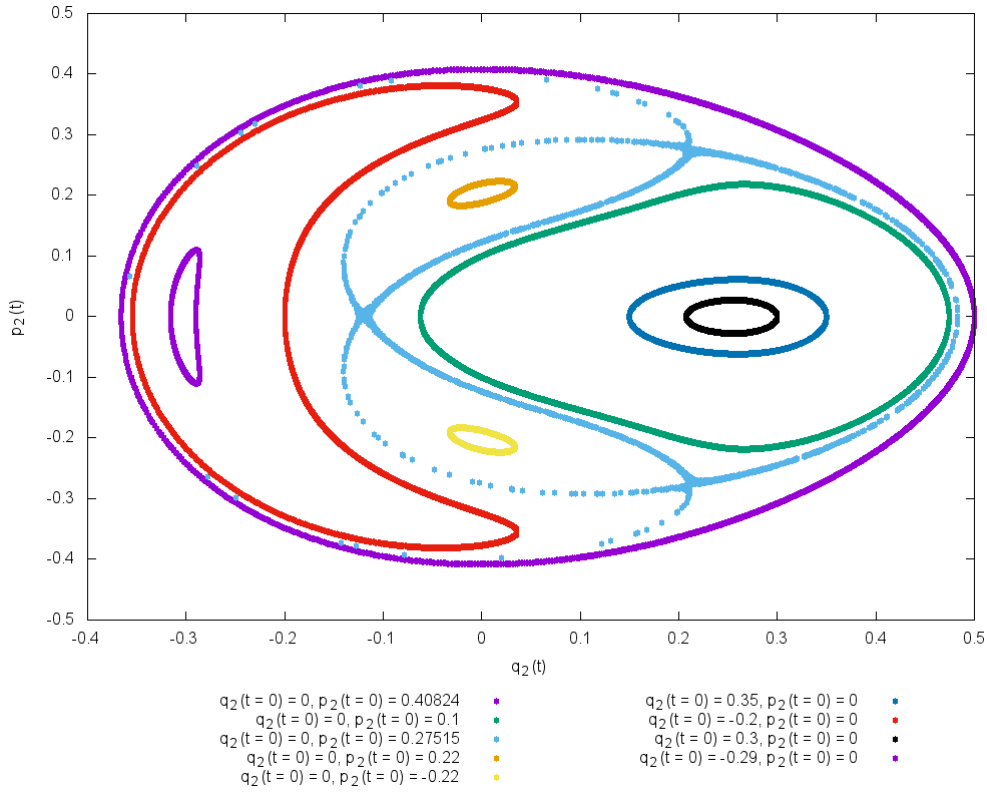


Figura 13: Seção de Poincaré para a Hamiltoniana de Hénon-Héles ( $E = 0.08333$ )

### Exercício 2.3.4

Segue abaixo o mapa de Poincaré para a Hamiltoniana de Hénon-Héles com  $E = 0.125$ . Diferentemente da Figura 13, agora podemos observar o comportamento caótico desse sistema (anteriormente a energia utilizada não apresentava esse tipo de comportamento). Nas mesmas regiões em que observamos as órbitas para  $E = 0.08333$  ocorrem as órbitas para a Figura 14, porém, temos uma região com cinco “ilhas” que pertencem a uma única órbita periódica. Também observamos um movimento quasi-periódico nos pontos em verde, que surgem muito próximos a uma separatriz desse sistema.

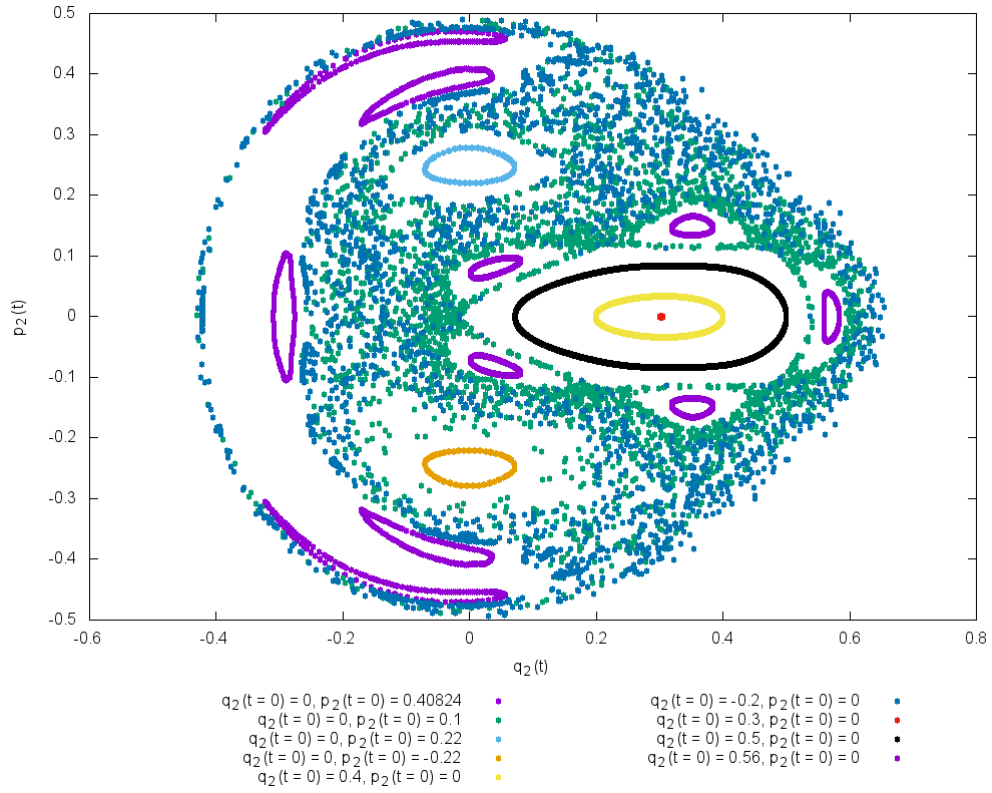


Figura 14: Seção de Poincaré para a Hamiltoniana de Hénon-Héles ( $E = 0.125$ )

### Exercício 2.3.5

Está disposta abaixo a Figura com os gráficos solicitados neste exercício.

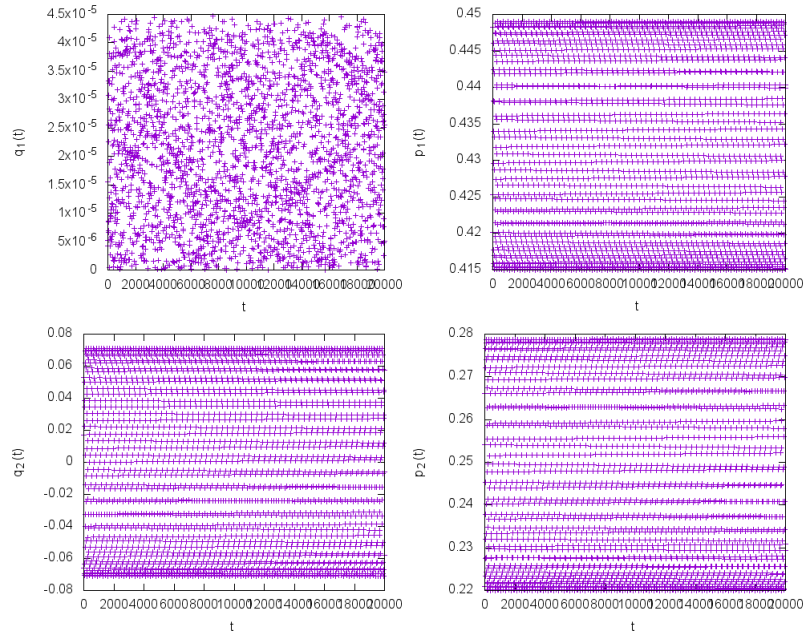


Figura 15: Trajetória regular para a Hamiltoniana de Hénon-Héles

### Exercício 2.3.6

Está disposta abaixo a Figura com os gráficos solicitados neste exercício.

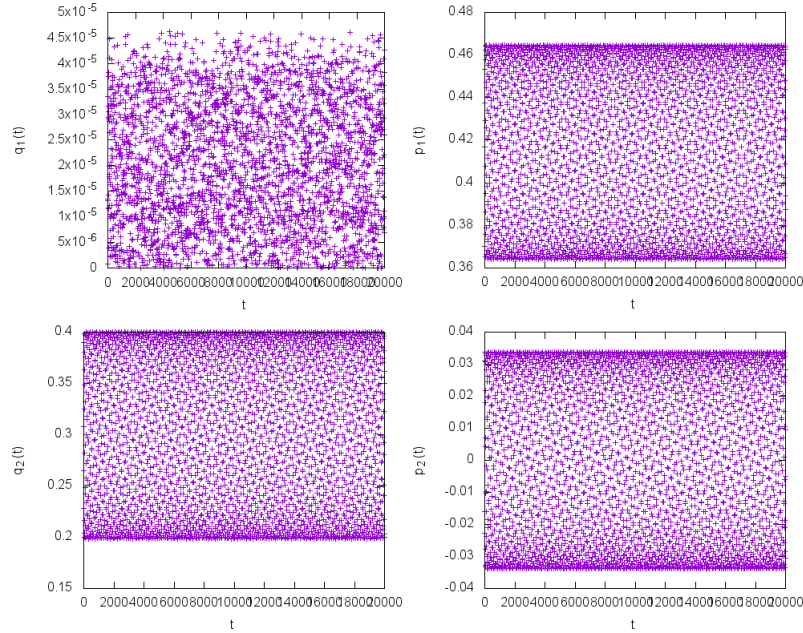


Figura 16: Trajetória caótica para a Hamiltoniana de Hénon-Helies

### Exercício 2.3.7

Segue abaixo o mapa de Poincaré para a Hamiltoniana de Hénon-Helies para o caso em que  $E = 0.16667$ . Nessa figura podemos observar um comportamento predominantemente caótico. Um movimento periódico ocorre em para os pontos em azul, onde surgem duas “ilhas”. Também podemos notar que ocorre um movimento periódico para os pontos em verde. Pontos de um movimento quasi-periódico estão em amarelo que, pela aproximação com as “ilhas”, podemos afirmar que estão próximos a uma separatriz do sistema.

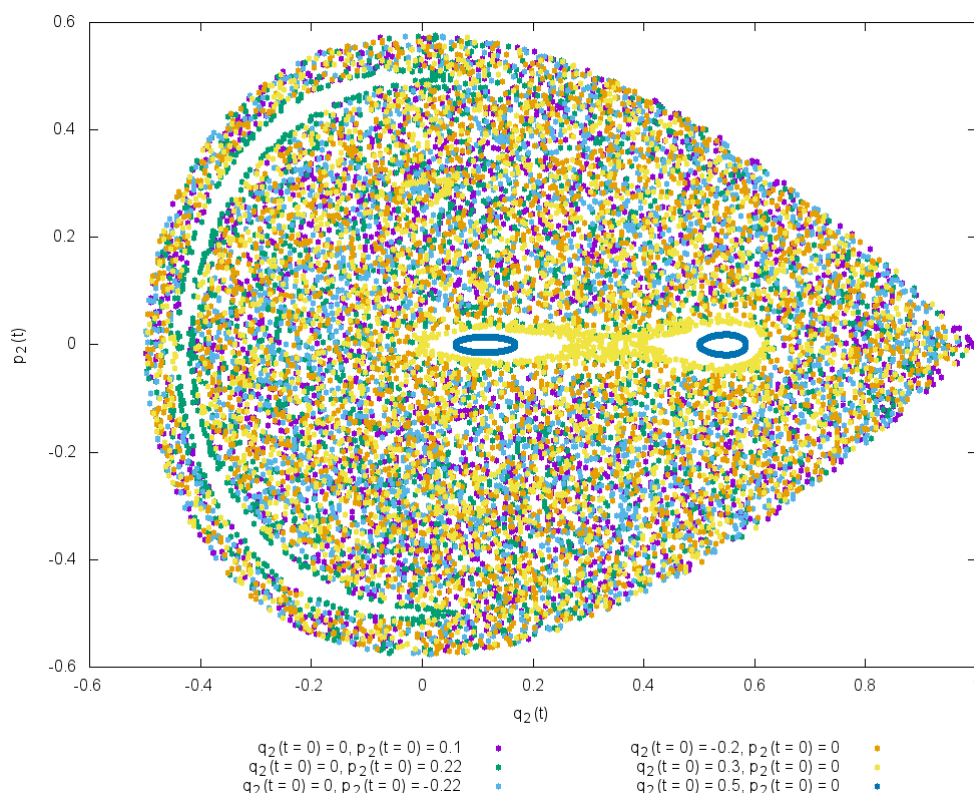


Figura 17: Seção de Poincaré para a Hamiltoniana de Hénon-Héles ( $E = 0.16667$ )

## Referências

- [1] Prof. Iberê L. Caldas, Primeiro Estudo Dirigido (2º Semestre de 2015)  
<http://web.if.usp.br/controle/sites/web.if.usp.br/controle/files/EstudoDirigido1.pdf>
- [2] Henon, M., Heiles, C., *The applicability of the third integral of motion: Some numerical experiments*, Astronomical Journal, Vol. 69, p. 73 (1964)

## Anexos

### Programa 1 - Equações de Euler para o pêndulo simples

```

1  /*****
2  *          PGF 5005 – Mecânica Clássica          *
3  *          Primeiro Estudo Dirigido              *
4  *          2º Semestre de 2015                  *
5  *
6  *          http://goo.gl/7PvBxi                  *
7  * Programa 1 – Equações de Euler para pêndulo simples *
8  *
9  * Aluno: Rafael Mendonça Miller   NUSP.:7581818   *
10 * e-mail: rafael.miller@usp.br                 *
11 *****/
12
13 #include <iostream>
14 #include <cmath>
15 #include <fstream>
16 #include <vector>
17
18 using namespace std;
19
20 double hamiltoniana(double p, double q) {

```

```

21 |     return (p*p)/2 - cos(q);
22 | }
23 |
24 | double dHamiltoniana_dp(double p) {
25 |     return p;
26 | }
27 |
28 | double dHamiltoniana_dq(double q) {
29 |     return -sin(q);
30 | }
31 |
32 | double q_nMaisUm(double q_n, double deltaT, double p_n) {
33 |     return q_n + deltaT*dHamiltoniana_dp(p_n);
34 | }
35 |
36 | double p_nMaisUm(double p_n, double deltaT, double q_n) {
37 |     return p_n - deltaT*dHamiltoniana_dq(q_n);
38 | }
39 |
40 | int main() {
41 |
42 |     ofstream dadosLibracao_q_01, dadosLibracao_p_01, dadosLibracao_e_01;
43 |     ofstream dadosLibracao_q_001, dadosLibracao_p_001, dadosLibracao_e_001;
44 |     ofstream dadosLibracao_q_0001, dadosLibracao_p_0001, dadosLibracao_e_0001;
45 |     ofstream dadosLibracao_pvq_01, dadosLibracao_pvq_001, dadosLibracao_pvq_0001;
46 |
47 |     dadosLibracao_q_01.open("out/dadosLibracao_q_01.dat"); dadosLibracao_q_001.open("out/
48 |         dadosLibracao_q_001.dat"); dadosLibracao_q_0001.open("out/dadosLibracao_q_0001.dat");
49 |     dadosLibracao_p_01.open("out/dadosLibracao_p_01.dat"); dadosLibracao_p_001.open("out/
50 |         dadosLibracao_p_001.dat"); dadosLibracao_p_0001.open("out/dadosLibracao_p_0001.dat");
51 |     dadosLibracao_e_01.open("out/dadosLibracao_e_01.dat"); dadosLibracao_e_001.open("out/
52 |         dadosLibracao_e_001.dat"); dadosLibracao_e_0001.open("out/dadosLibracao_e_0001.dat");
53 |     dadosLibracao_pvq_01.open("out/dadosLibracao_pvq_01.dat"); dadosLibracao_pvq_001.open("out/
54 |         dadosLibracao_pvq_001.dat"); dadosLibracao_pvq_0001.open("out/dadosLibracao_pvq_0001.dat")
55 |         ;
56 |
57 |     double T = 0.0;
58 |     double deltaT = 0.1;
59 |
60 |     vector<double> Q_n (99999);
61 |     vector<double> P_n (99999);
62 |     vector<double> E_n (99999);
63 |
64 |     // Condições Iniciais para Libração:
65 |     P_n[0] = 0.25;
66 |     Q_n[0] = 1.0;
67 |     E_n[0] = hamiltoniana(P_n[0], Q_n[0]);
68 |
69 |     for(int i = 0; T < 25; i++) {
70 |
71 |         Q_n[i+1] = q_nMaisUm(Q_n[i], deltaT, P_n[i]);
72 |         P_n[i+1] = p_nMaisUm(P_n[i], deltaT, Q_n[i]);
73 |         E_n[i+1] = hamiltoniana(P_n[i], Q_n[i]);
74 |
75 |         // Imprime os dados nos arquivos
76 |         dadosLibracao_q_01 << T << "\t" << Q_n[i] << endl;
77 |         dadosLibracao_p_01 << T << "\t" << P_n[i] << endl;
78 |         dadosLibracao_e_01 << T << "\t" << E_n[i] << endl;
79 |         dadosLibracao_pvq_01 << Q_n[i] << "\t" << P_n[i] << endl;
80 |
81 |         T+=deltaT;
82 |     }
83 |
84 |     // Diminuindo deltaT e reajustando as Condições Iniciais:
85 |     T = 0.0;
86 |     deltaT = 0.01;
87 |
88 |     P_n[0] = 0.25;
89 |     Q_n[0] = 1.0;
90 |     E_n[0] = hamiltoniana(P_n[0], Q_n[0]);
91 |
92 |     for(int i = 0; T < 25; i++) {

```

```

89     Q_n[i+1] = q_nMaisUm(Q_n[i], deltaT, P_n[i]);
90     P_n[i+1] = p_nMaisUm(P_n[i], deltaT, Q_n[i]);
91     E_n[i+1] = hamiltoniana(P_n[i], Q_n[i]);

93     dadosLibracao_q_001 << T << "\t" << Q_n[i] << endl;
94     dadosLibracao_p_001 << T << "\t" << P_n[i] << endl;
95     dadosLibracao_e_001 << T << "\t" << E_n[i] << endl;
96     dadosLibracao_pvq_001 << Q_n[i] << "\t" << P_n[i] << endl;

97     T+=deltaT;
98 }

101 // Diminuindo deltaT e reajustando as Condições Iniciais:
102 T = 0.0;
103 deltaT = 0.001;

105 P_n[0] = 0.25;
106 Q_n[0] = 1.0;
107 E_n[0] = hamiltoniana(P_n[0], Q_n[0]);

109 for(int i = 0; T < 25; i++) {

111     Q_n[i+1] = q_nMaisUm(Q_n[i], deltaT, P_n[i]);
112     P_n[i+1] = p_nMaisUm(P_n[i], deltaT, Q_n[i]);
113     E_n[i+1] = hamiltoniana(P_n[i], Q_n[i]);

115     dadosLibracao_q_0001 << T << "\t" << Q_n[i] << endl;
116     dadosLibracao_p_0001 << T << "\t" << P_n[i] << endl;
117     dadosLibracao_e_0001 << T << "\t" << E_n[i] << endl;
118     dadosLibracao_pvq_0001 << Q_n[i] << "\t" << P_n[i] << endl;

119     T+=deltaT;
120 }

123 // Agora redefinimos os arquivos e ajustamos parâmetros para a Rotação

125 ofstream dadosRotacao_q_01, dadosRotacao_p_01, dadosRotacao_e_01;
126 ofstream dadosRotacao_q_001, dadosRotacao_p_001, dadosRotacao_e_001;
127 ofstream dadosRotacao_q_0001, dadosRotacao_p_0001, dadosRotacao_e_0001;
128 ofstream dadosRotacao_pvq_01, dadosRotacao_pvq_001, dadosRotacao_pvq_0001;

129 dadosRotacao_q_01.open("out/dadosRotacao_q_01.dat"); dadosRotacao_q_001.open("out/
130     dadosRotacao_q_001.dat"); dadosRotacao_q_0001.open("out/dadosRotacao_q_0001.dat");
131 dadosRotacao_p_01.open("out/dadosRotacao_p_01.dat"); dadosRotacao_p_001.open("out/
132     dadosRotacao_p_001.dat"); dadosRotacao_p_0001.open("out/dadosRotacao_p_0001.dat");
133 dadosRotacao_e_01.open("out/dadosRotacao_e_01.dat"); dadosRotacao_e_001.open("out/
134     dadosRotacao_e_001.dat"); dadosRotacao_e_0001.open("out/dadosRotacao_e_0001.dat");
135 dadosRotacao_pvq_01.open("out/dadosRotacao_pvq_01.dat"); dadosRotacao_pvq_001.open("out/
136     dadosRotacao_pvq_001.dat"); dadosRotacao_pvq_0001.open("out/dadosRotacao_pvq_0001.dat");

137 // Condições Iniciais para a Rotação:
138 T = 0.0;
139 deltaT = 0.1;

141 P_n[0] = 3.0;
142 Q_n[0] = 0.0;
143 E_n[0] = hamiltoniana(P_n[0], Q_n[0]);

145 for(int i = 0; T < 25; i++) {

147     Q_n[i+1] = q_nMaisUm(Q_n[i], deltaT, P_n[i]);
148     P_n[i+1] = p_nMaisUm(P_n[i], deltaT, Q_n[i]);
149     E_n[i+1] = hamiltoniana(P_n[i], Q_n[i]);

151     // Imprime os dados nos arquivos
152     dadosRotacao_q_01 << T << "\t" << Q_n[i] << endl;
153     dadosRotacao_p_01 << T << "\t" << P_n[i] << endl;
154     dadosRotacao_e_01 << T << "\t" << E_n[i] << endl;
155     dadosRotacao_pvq_01 << Q_n[i] << "\t" << P_n[i] << endl;

```

```

157     T+=deltaT;
159 }
161 // Diminuindo deltaT e reajustando as Condições Iniciais:
163 T = 0.0;
165 deltaT = 0.01;
167 P_n[0] = 3.0;
169 Q_n[0] = 0.0;
171 E_n[0] = hamiltoniana(P_n[0], Q_n[0]);
173 for(int i = 0; T < 25; i++) {
175     Q_n[i+1] = q_nMaisUm(Q_n[i], deltaT, P_n[i]);
177     P_n[i+1] = p_nMaisUm(P_n[i], deltaT, Q_n[i]);
179     E_n[i+1] = hamiltoniana(P_n[i], Q_n[i]);
181     dadosRotacao-q_001 << T << "\t" << Q_n[i] << endl;
183     dadosRotacao-p_001 << T << "\t" << P_n[i] << endl;
185     dadosRotacao-e_001 << T << "\t" << E_n[i] << endl;
187     dadosRotacao-pvq_001 << Q_n[i] << "\t" << P_n[i] << endl;
189     T+=deltaT;
191 }
193 // Diminuindo deltaT e reajustando as Condições Iniciais:
195 T = 0.0;
197 deltaT = 0.001;
199 P_n[0] = 3.0;
201 Q_n[0] = 0.0;
203 E_n[0] = hamiltoniana(P_n[0], Q_n[0]);
205 for(int i = 0; T < 25; i++) {
207     Q_n[i+1] = q_nMaisUm(Q_n[i], deltaT, P_n[i]);
209     P_n[i+1] = p_nMaisUm(P_n[i], deltaT, Q_n[i]);
211     E_n[i+1] = hamiltoniana(P_n[i], Q_n[i]);
213     dadosRotacao-q_0001 << T << "\t" << Q_n[i] << endl;
215     dadosRotacao-p_0001 << T << "\t" << P_n[i] << endl;
217     dadosRotacao-e_0001 << T << "\t" << E_n[i] << endl;
219     dadosRotacao-pvq_0001 << Q_n[i] << "\t" << P_n[i] << endl;
221     T+=deltaT;
223 }
225 return 0;
227 }

```

../Programa1/programa1.cpp

## Programa 2 - Equações modificadas de Euler para o pêndulo simples

```

1  /*****
2  *                               *
3  * PGF 5005 – Mecânica Clássica *
4  * Primeiro Estudo Dirigido    *
5  * 2º Semestre de 2015        *
6  *                               *
7  * http://goo.gl/7PvBxi       *
8  * Programa 2 – Equações modificadas de Euler para pêndulo simples *
9  *                               *
10 * Aluno: Rafael Mendonça Miller NUSP.:7581818 *
11 * e-mail: rafael.miller@usp.br *
12 *****/
13 #include <iostream>

```

```

#include <cmath>
15 #include <fstream>
#include <vector>
17
using namespace std;
19
double hamiltoniana(double p, double q) {
21     return (p*p)/2 - cos(q);
}
23
double dHamiltoniana_dp(double p) {
25     return p;
}
27
double dHamiltoniana_dq(double q) {
29     return sin(q);
}
31
double q_nMaisUm(double q_n, double deltaT, double p_n) {
33     return q_n + deltaT*dHamiltoniana_dp(p_n);
}
35
double p_nMaisUm(double p_n, double deltaT, double q_n) {
37     return p_n - deltaT*dHamiltoniana_dq(q_n);
}
39
41 int main() {
43
    ofstream dadosLibracao_q_01, dadosLibracao_p_01, dadosLibracao_e_01;
45     ofstream dadosLibracao_q_001, dadosLibracao_p_001, dadosLibracao_e_001;
    ofstream dadosLibracao_q_0001, dadosLibracao_p_0001, dadosLibracao_e_0001;
47     ofstream dadosLibracao_pvq_01, dadosLibracao_pvq_001, dadosLibracao_pvq_0001;

49     dadosLibracao_q_01.open("out/dadosLibracao_q_01.dat"); dadosLibracao_q_001.open("out/
        dadosLibracao_q_001.dat"); dadosLibracao_q_0001.open("out/dadosLibracao_q_0001.dat");
    dadosLibracao_p_01.open("out/dadosLibracao_p_01.dat"); dadosLibracao_p_001.open("out/
        dadosLibracao_p_001.dat"); dadosLibracao_p_0001.open("out/dadosLibracao_p_0001.dat");
51     dadosLibracao_e_01.open("out/dadosLibracao_e_01.dat"); dadosLibracao_e_001.open("out/
        dadosLibracao_e_001.dat"); dadosLibracao_e_0001.open("out/dadosLibracao_e_0001.dat");
    dadosLibracao_pvq_01.open("out/dadosLibracao_pvq_01.dat"); dadosLibracao_pvq_001.open("out/
        dadosLibracao_pvq_001.dat"); dadosLibracao_pvq_0001.open("out/dadosLibracao_pvq_0001.dat")
        ;

53     double T = 0.0;
55     double deltaT = 0.1;

57     vector<double> Q_n (99999);
    vector<double> P_n (99999);
59     vector<double> E_n (99999);

61     // Condições Iniciais para Libração:
    P_n[0] = 1.0;
63     Q_n[0] = 0.1;
    E_n[0] = hamiltoniana(P_n[0], Q_n[0]);
65

    for(int i = 0; T < 25; i++) {
67
        Q_n[i+1] = q_nMaisUm(Q_n[i], deltaT, P_n[i]);
69         P_n[i+1] = p_nMaisUm(P_n[i], deltaT, Q_n[i+1]);
        E_n[i+1] = hamiltoniana(P_n[i+1], Q_n[i+1]);
71

        // Imprime os dados nos arquivos
73         dadosLibracao_q_01 << T << "\t" << Q_n[i] << endl;
        dadosLibracao_p_01 << T << "\t" << P_n[i] << endl;
75         dadosLibracao_e_01 << T << "\t" << E_n[i] << endl;
        dadosLibracao_pvq_01 << Q_n[i] << "\t" << P_n[i] << endl;
77

        T+=deltaT;
79     }
}

```



```

81 // Diminuindo deltaT e reajustando as Condições Iniciais:
T = 0.0;
83 deltaT = 0.01;

85 P_n[0] = 1.0;
Q_n[0] = 0.1;
87 E_n[0] = hamiltoniana(P_n[0], Q_n[0]);

89 for(int i = 0; T < 25; i++) {

91     Q_n[i+1] = q_nMaisUm(Q_n[i], deltaT, P_n[i]);
P_n[i+1] = p_nMaisUm(P_n[i], deltaT, Q_n[i+1]);
93     E_n[i+1] = hamiltoniana(P_n[i+1], Q_n[i+1]);

95     dadosLibracao-q-001 << T << "\t" << Q_n[i] << endl;
dadosLibracao-p-001 << T << "\t" << P_n[i] << endl;
97     dadosLibracao-e-001 << T << "\t" << E_n[i] << endl;
dadosLibracao-pvq-001 << Q_n[i] << "\t" << P_n[i] << endl;

99     T+=deltaT;
101 }

103 // Diminuindo deltaT e reajustando as Condições Iniciais:
T = 0.0;
105 deltaT = 0.001;

107 P_n[0] = 1.0;
Q_n[0] = 0.1;
109 E_n[0] = hamiltoniana(P_n[0], Q_n[0]);

111 for(int i = 0; T < 25; i++) {

113     Q_n[i+1] = q_nMaisUm(Q_n[i], deltaT, P_n[i]);
P_n[i+1] = p_nMaisUm(P_n[i], deltaT, Q_n[i+1]);
115     E_n[i+1] = hamiltoniana(P_n[i+1], Q_n[i+1]);

117     dadosLibracao-q-0001 << T << "\t" << Q_n[i] << endl;
dadosLibracao-p-0001 << T << "\t" << P_n[i] << endl;
119     dadosLibracao-e-0001 << T << "\t" << E_n[i] << endl;
dadosLibracao-pvq-0001 << Q_n[i] << "\t" << P_n[i] << endl;

121     T+=deltaT;
123 }

125 // Agora redefinimos os arquivos e ajustamos parâmetros para a Rotação

127 ofstream dadosRotacao-q-01, dadosRotacao-p-01, dadosRotacao-e-01;
ofstream dadosRotacao-q-001, dadosRotacao-p-001, dadosRotacao-e-001;
129 ofstream dadosRotacao-q-0001, dadosRotacao-p-0001, dadosRotacao-e-0001;
ofstream dadosRotacao-pvq-01, dadosRotacao-pvq-001, dadosRotacao-pvq-0001;

131 dadosRotacao-q-01.open("out/dadosRotacao-q-01.dat"); dadosRotacao-q-001.open("out/
dadosRotacao-q-001.dat"); dadosRotacao-q-0001.open("out/dadosRotacao-q-0001.dat");
133 dadosRotacao-p-01.open("out/dadosRotacao-p-01.dat"); dadosRotacao-p-001.open("out/
dadosRotacao-p-001.dat"); dadosRotacao-p-0001.open("out/dadosRotacao-p-0001.dat");
dadosRotacao-e-01.open("out/dadosRotacao-e-01.dat"); dadosRotacao-e-001.open("out/
dadosRotacao-e-001.dat"); dadosRotacao-e-0001.open("out/dadosRotacao-e-0001.dat");
135 dadosRotacao-pvq-01.open("out/dadosRotacao-pvq-01.dat"); dadosRotacao-pvq-001.open("out/
dadosRotacao-pvq-001.dat"); dadosRotacao-pvq-0001.open("out/dadosRotacao-pvq-0001.dat");

137 // Condições Iniciais para Rotação:
139 T = 0.0;
deltaT = 0.1;

141 P_n[0] = 3;
Q_n[0] = 0.0;
143 E_n[0] = hamiltoniana(P_n[0], Q_n[0]);

145 for(int i = 0; T < 25; i++) {

147     Q_n[i+1] = q_nMaisUm(Q_n[i], deltaT, P_n[i]);

```

```

149     P_n[i+1] = p_nMaisUm(P_n[i], deltaT, Q_n[i+1]);
150     E_n[i+1] = hamiltoniana(P_n[i+1], Q_n[i+1]);
151
152     // Imprime os dados nos arquivos
153     dadosRotacao_q_01 << T << "\t" << Q_n[i] << endl;
154     dadosRotacao_p_01 << T << "\t" << P_n[i] << endl;
155     dadosRotacao_e_01 << T << "\t" << E_n[i] << endl;
156     dadosRotacao_pvq_01 << Q_n[i] << "\t" << P_n[i] << endl;
157
158     T+=deltaT;
159 }
160
161 // Diminuindo deltaT e reajustando as Condições Iniciais:
162 T = 0.0;
163 deltaT = 0.01;
164
165 P_n[0] = 3;
166 Q_n[0] = 0.0;
167 E_n[0] = hamiltoniana(P_n[0], Q_n[0]);
168
169 for(int i = 0; T < 25; i++) {
170
171     Q_n[i+1] = q_nMaisUm(Q_n[i], deltaT, P_n[i]);
172     P_n[i+1] = p_nMaisUm(P_n[i], deltaT, Q_n[i+1]);
173     E_n[i+1] = hamiltoniana(P_n[i+1], Q_n[i+1]);
174
175     dadosRotacao_q_001 << T << "\t" << Q_n[i] << endl;
176     dadosRotacao_p_001 << T << "\t" << P_n[i] << endl;
177     dadosRotacao_e_001 << T << "\t" << E_n[i] << endl;
178     dadosRotacao_pvq_001 << Q_n[i] << "\t" << P_n[i] << endl;
179
180     T+=deltaT;
181 }
182
183 // Diminuindo deltaT e reajustando as Condições Iniciais:
184 T = 0.0;
185 deltaT = 0.001;
186
187 P_n[0] = 3;
188 Q_n[0] = 0.0;
189 E_n[0] = hamiltoniana(P_n[0], Q_n[0]);
190
191 for(int i = 0; T < 25; i++) {
192
193     Q_n[i+1] = q_nMaisUm(Q_n[i], deltaT, P_n[i]);
194     P_n[i+1] = p_nMaisUm(P_n[i], deltaT, Q_n[i+1]);
195     E_n[i+1] = hamiltoniana(P_n[i+1], Q_n[i+1]);
196
197     dadosRotacao_q_0001 << T << "\t" << Q_n[i] << endl;
198     dadosRotacao_p_0001 << T << "\t" << P_n[i] << endl;
199     dadosRotacao_e_0001 << T << "\t" << E_n[i] << endl;
200     dadosRotacao_pvq_0001 << Q_n[i] << "\t" << P_n[i] << endl;
201
202     T+=deltaT;
203 }
204
205 return 0;
206 }

```

../Programa2/programa2.cpp

### Programa 3 - Método de Euler simplético para a hamiltoniana de Hénon-Heiles

```

/*****
2  *                               PGF 5005 – Mecânica Clássica                               *
3  *                               Primeiro Estudo Dirigido                               *
4  *                               2º Semestre de 2015                               *
5  *                               *                               *
6  *                               http://goo.gl/7PvBxi                               *
7  * Programa 3 – Método de Euler simplético para a Hamiltoniana de Hénon-Helies *
8  */

```



```

80 double E_n;

82 double Q2s;
double P2s;

84

86 /***** E = 0.08333: *****/

88 vector <double> CondicaoInicial_Q2 (30, 0.0);
vector <double> CondicaoInicial_P2 (30, 0.0);

90 CondicaoInicial_P2[0] = 0.40824; // Separatriz
92 CondicaoInicial_P2[1] = 0.1;
CondicaoInicial_P2[2] = 0.27515; // Separatriz
94 CondicaoInicial_P2[3] = 0.22;
CondicaoInicial_P2[4] = -0.22;

96 CondicaoInicial_Q2[5] = 0.35;
98 CondicaoInicial_Q2[6] = -0.2;
CondicaoInicial_Q2[7] = 0.3;
100 CondicaoInicial_Q2[8] = -0.29;

102
104 for(int j = 0; j < 9; j++) {
    ofstream dadosMapaDePoincare;
106     string file_name = "out/dadosMapaDePoincare";
    file_name += to_string(j);
108     file_name += ".dat";
    dadosMapaDePoincare.open(file_name);

110
    Q1_n = 0.0;

112
    P2_n = CondicaoInicial_P2[j];
114    Q2_n = CondicaoInicial_Q2[j];

116    cout << P2_n << "\t" << Q2_n << "\t" << file_name << endl;
    E_n = 0.08333;
118    P1_n = sqrt(2*(E_n - P2_n*P2_n*(1./2) - Q1_n*Q1_n*(1./2) - Q2_n*Q2_n*(1./2) - Q1_n*Q1_n*
    Q2_n + (1./3)*Q2_n*Q2_n*Q2_n));

120    for(double T = 0.0; T < 20000; T+=deltaT) {

122        Q1_nMaisUm = q1_nMaisUm(Q1_n, deltaT, P1_n);
        Q2_nMaisUm = q2_nMaisUm(Q2_n, deltaT, P2_n);

124
        P1_nMaisUm = p1_nMaisUm(P1_n, deltaT, Q1_nMaisUm, Q2_nMaisUm);
        P2_nMaisUm = p2_nMaisUm(P2_n, deltaT, Q1_nMaisUm, Q2_nMaisUm);

128        if (Q1_nMaisUm*Q1_n < 0 && P1_n > 0) {

130            Q2s = q2s(P1_n, P2_n, Q2_n, -Q1_n);
            P2s = p2s(P1_n, P2_n, Q1_n, Q2_n, -Q1_n);

132            dadosMapaDePoincare << Q2s << "\t" << P2s << endl;

134        }

136        Q1_n = Q1_nMaisUm;
        Q2_n = Q2_nMaisUm;
138        P1_n = P1_nMaisUm;
        P2_n = P2_nMaisUm;

140    }

142 }

144 /***** E = 0.125: *****/

146 ofstream q1PeriodicaE0125,
        p1PeriodicaE0125,
148        q2PeriodicaE0125,
        p2PeriodicaE0125;
150

```

```

152 q1PeriodicaE0125.open("out/q1PeriodicaE0125.dat");
153 p1PeriodicaE0125.open("out/p1PeriodicaE0125.dat");
154 q2PeriodicaE0125.open("out/q2PeriodicaE0125.dat");
155 p2PeriodicaE0125.open("out/p2PeriodicaE0125.dat");

156 ofstream q1CaoticaE0125,
157          p1CaoticaE0125,
158          q2CaoticaE0125,
159          p2CaoticaE0125;

160
161 q1CaoticaE0125.open("out/q1CaoticaE0125.dat");
162 p1CaoticaE0125.open("out/p1CaoticaE0125.dat");
163 q2CaoticaE0125.open("out/q2CaoticaE0125.dat");
164 p2CaoticaE0125.open("out/p2CaoticaE0125.dat");

165
166 CondicaoInicial_P2[9] = 0.40824;
167 CondicaoInicial_P2[10] = 0.1; // Quasi-Periódica ou Separatriz?
168 CondicaoInicial_P2[11] = 0.22;
169 CondicaoInicial_P2[12] = -0.22;

170
171 CondicaoInicial_Q2[13] = 0.4; // Caótico
172 CondicaoInicial_Q2[14] = -0.2;
173 CondicaoInicial_Q2[15] = 0.3; // Centro da ilha
174 CondicaoInicial_Q2[16] = 0.5;
175 CondicaoInicial_Q2[17] = 0.56;

176
177 for(int j = 9; j < 18; j++) {
178
179     ofstream dadosMapaDePoincare;
180     string file_name = "out/dadosMapaDePoincare";
181     file_name += to_string(j);
182     file_name += ".dat";
183     dadosMapaDePoincare.open(file_name);

184
185     Q1_n = 0.0;

186
187     P2_n = CondicaoInicial_P2[j];
188     Q2_n = CondicaoInicial_Q2[j];

189
190     cout << P2_n << "\t" << Q2_n << "\t" << file_name << endl;
191     E_n = 0.125;
192     P1_n = sqrt(2*(E_n - P2_n*P2_n*(1./2) - Q1_n*Q1_n*(1./2) - Q2_n*Q2_n*(1./2) - Q1_n*Q1_n*
193     Q2_n + (1./3)*Q2_n*Q2_n*Q2_n));

194     for(double T = 0.0; T < 20000; T+=deltaT) {

195
196         Q1_nMaisUm = q1_nMaisUm(Q1_n, deltaT, P1_n);
197         Q2_nMaisUm = q2_nMaisUm(Q2_n, deltaT, P2_n);

198
199         P1_nMaisUm = p1_nMaisUm(P1_n, deltaT, Q1_nMaisUm, Q2_nMaisUm);
200         P2_nMaisUm = p2_nMaisUm(P2_n, deltaT, Q1_nMaisUm, Q2_nMaisUm);

201
202         if(Q1_nMaisUm*Q1_n < 0 && P1_n > 0) {

203
204             Q2s = q2s(P1_n, P2_n, Q2_n, -Q1_n);
205             P2s = p2s(P1_n, P2_n, Q1_n, Q2_n, -Q1_n);

206
207             dadosMapaDePoincare << Q2s << "\t" << P2s << endl;

208
209             // Para construir os gráficos de q1 x t, q2 x t, p1 x t, p2 x t:
210             if(j == 11) {
211                 q1PeriodicaE0125 << T << "\t" << Q1_nMaisUm << endl;
212                 p1PeriodicaE0125 << T << "\t" << P1_nMaisUm << endl;
213                 q2PeriodicaE0125 << T << "\t" << Q2_nMaisUm << endl;
214                 p2PeriodicaE0125 << T << "\t" << P2_nMaisUm << endl;
215             }

216
217             if(j == 13) {
218                 q1CaoticaE0125 << T << "\t" << Q1_nMaisUm << endl;
219                 p1CaoticaE0125 << T << "\t" << P1_nMaisUm << endl;
220                 q2CaoticaE0125 << T << "\t" << Q2_nMaisUm << endl;
221                 p2CaoticaE0125 << T << "\t" << P2_nMaisUm << endl;

```

```

222     }
223 }
224
225     Q1_n = Q1_nMaisUm;
226     Q2_n = Q2_nMaisUm;
227     P1_n = P1_nMaisUm;
228     P2_n = P2_nMaisUm;
229 }
230
231 }
232
233 /***** E = 0.16667: *****/
234
235 CondicaoInicial_P2[18] = 0.1;
236 CondicaoInicial_P2[19] = 0.22;
237 CondicaoInicial_P2[20] = -0.22;
238
239 CondicaoInicial_Q2[21] = -0.2;
240 CondicaoInicial_Q2[22] = 0.3;
241 CondicaoInicial_Q2[23] = 0.5;
242
243 for(int j = 18; j < 24; j++) {
244
245     ofstream dadosMapaDePoincare;
246     string file_name = "out/dadosMapaDePoincare";
247     file_name += to_string(j);
248     file_name += ".dat";
249     dadosMapaDePoincare.open(file_name);
250
251     Q1_n = 0.0;
252
253     P2_n = CondicaoInicial_P2[j];
254     Q2_n = CondicaoInicial_Q2[j];
255
256     cout << P2_n << "\t" << Q2_n << "\t" << file_name << endl;
257     E_n = 0.16667;
258     P1_n = sqrt(2*(E_n - P2_n*P2_n*(1./2) - Q1_n*Q1_n*(1./2) - Q2_n*Q2_n*(1./2) - Q1_n*Q1_n*
259     Q2_n + (1./3)*Q2_n*Q2_n*Q2_n));
260
261     for(double T = 0.0; T < 20000; T+=deltaT) {
262
263         Q1_nMaisUm = q1_nMaisUm(Q1_n, deltaT, P1_n);
264         Q2_nMaisUm = q2_nMaisUm(Q2_n, deltaT, P2_n);
265
266         P1_nMaisUm = p1_nMaisUm(P1_n, deltaT, Q1_nMaisUm, Q2_nMaisUm);
267         P2_nMaisUm = p2_nMaisUm(P2_n, deltaT, Q1_nMaisUm, Q2_nMaisUm);
268
269         if(Q1_nMaisUm*Q1_n < 0 && P1_n > 0) {
270
271             Q2s = q2s(P1_n, P2_n, Q2_n, -Q1_n);
272             P2s = p2s(P1_n, P2_n, Q1_n, Q2_n, -Q1_n);
273
274             dadosMapaDePoincare << Q2s << "\t" << P2s << endl;
275         }
276
277         Q1_n = Q1_nMaisUm;
278         Q2_n = Q2_nMaisUm;
279         P1_n = P1_nMaisUm;
280         P2_n = P2_nMaisUm;
281     }
282 }
283
284 return 0;
285 }

```

../Programa3/programa3.cpp