PGF 5005 - Mecânica Clássica
Prof. Iberê L. Caldas
Primeiro Estudo Dirigido
2^{o} 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:

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

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

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

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.

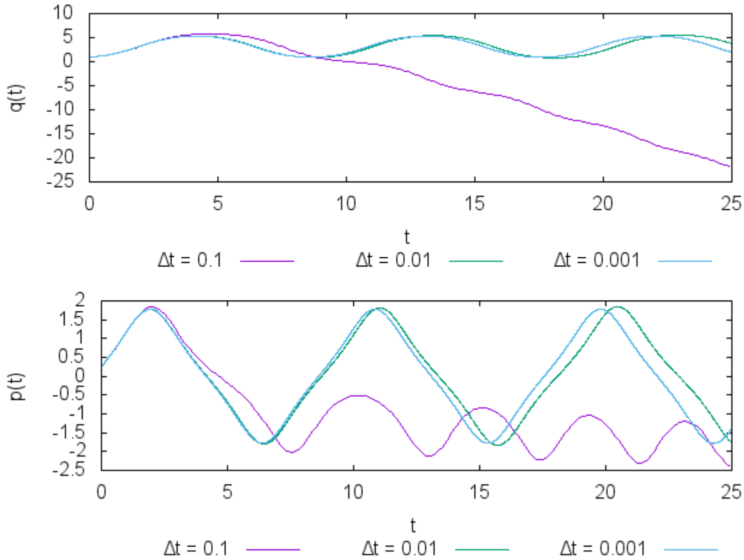

Figura 1: Libração de um pêndulo simples

Podemos observar uma grande diferença entre os valores de Δt. Ao utilizar um valor "alto" de Δt 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 Δt. 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.

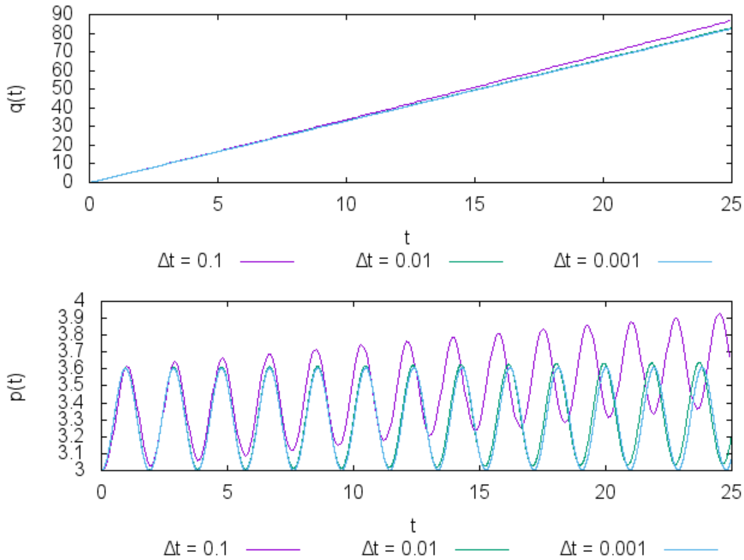

Figura 2: Rotação de um pêndulo simples

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.

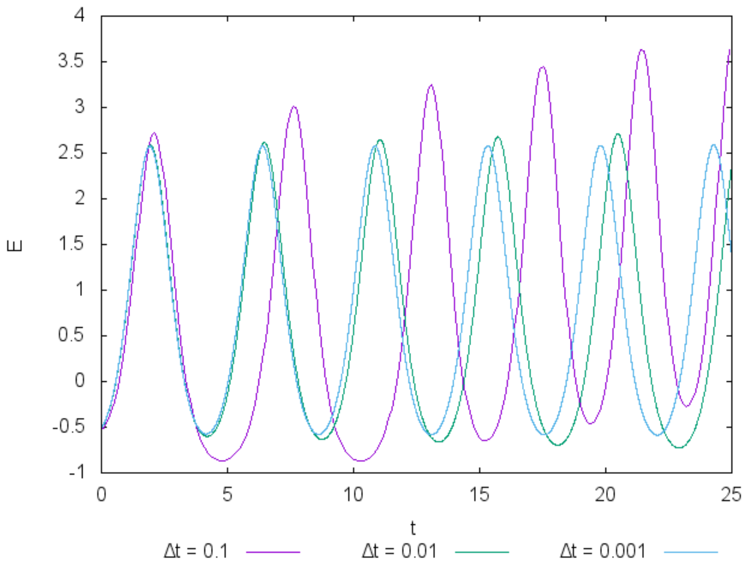

Figura 3: Energia de libração do pêndulo simples

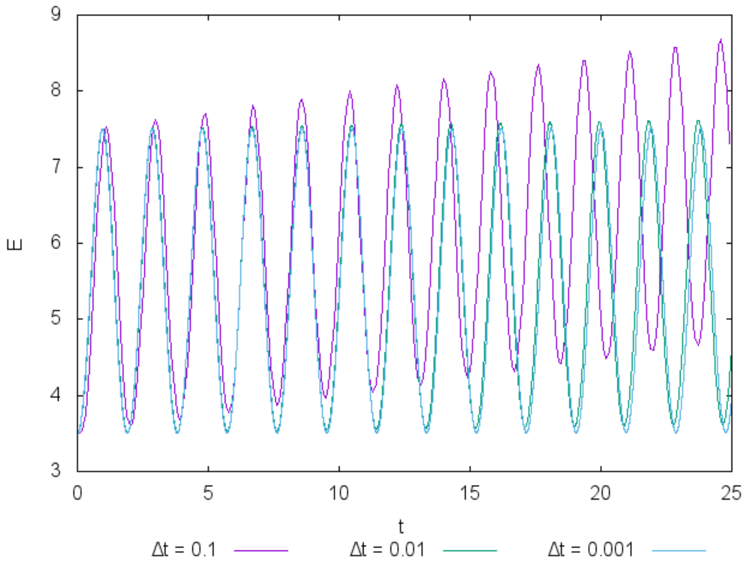

Figura 4: Energia de rotação do pêndulo simples

Exercício 2.1.5

Observamos das figuras de trajetórias (p(t) x 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.

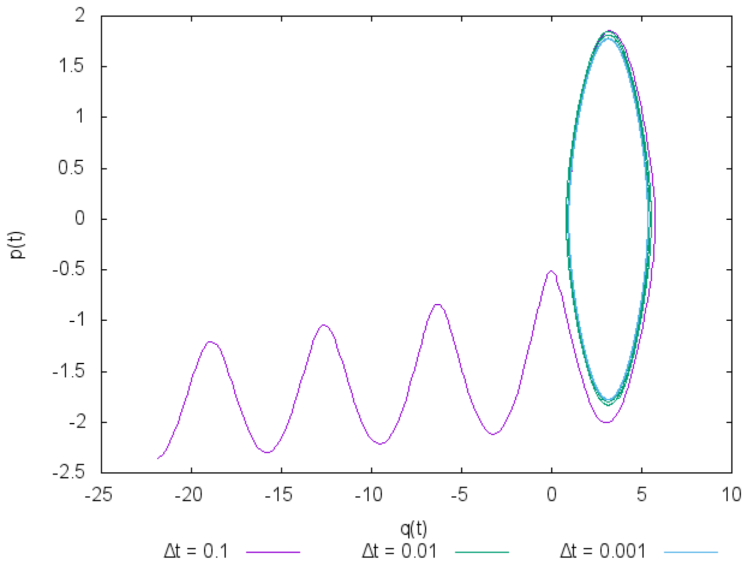

Figura 5: Trajetória de libração do pêndulo simples

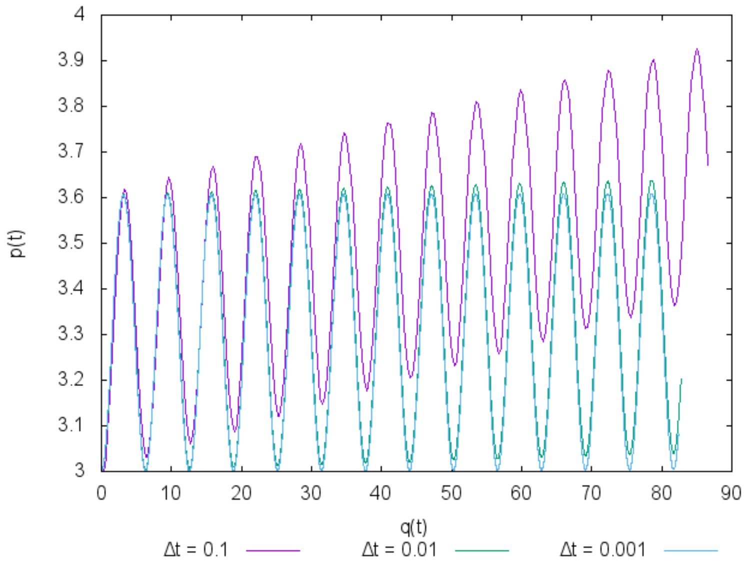

Figura 6: Trajetória de rotação do pêndulo simples

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 Δt. Ou seja, a diminuição no Δt que servia para “contornar”o problema do método não ser simplético não se vê mais necessária.

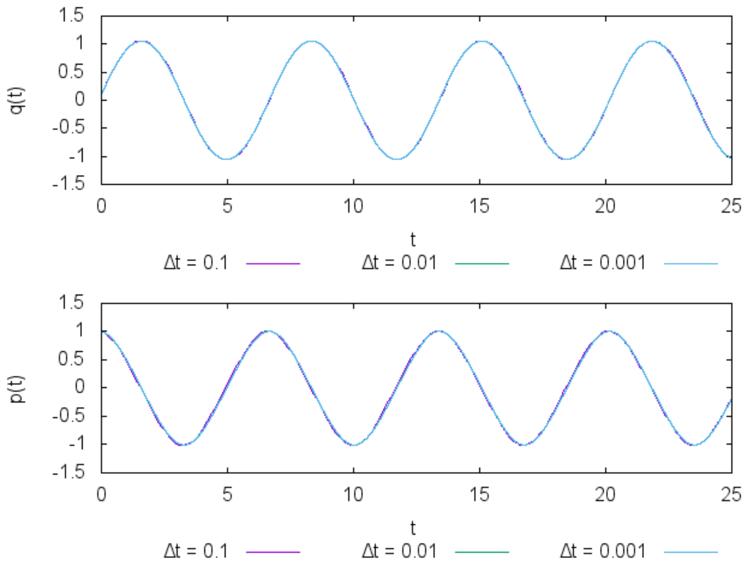

Figura 7: Libração de um pêndulo simples (versão modificada)

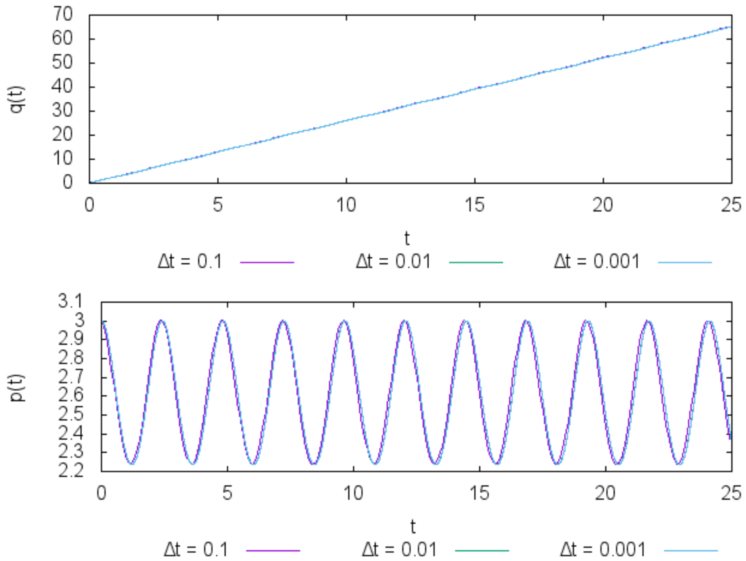

Figura 8: Rotação de um pêndulo simples (versão modificada)

Exercício 2.1.8

Nos gráficos de energia para os dois movimentos do pêndulo, se percebe uma variação para cada Δt. 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 Δt nos gráficos das Figuras 7 e 8, nos gáficos das Figuras 9 e 10 essa mudança fica nítida e a importância na diminuição de Δt para melhor representar o movimento se torna mais clara.

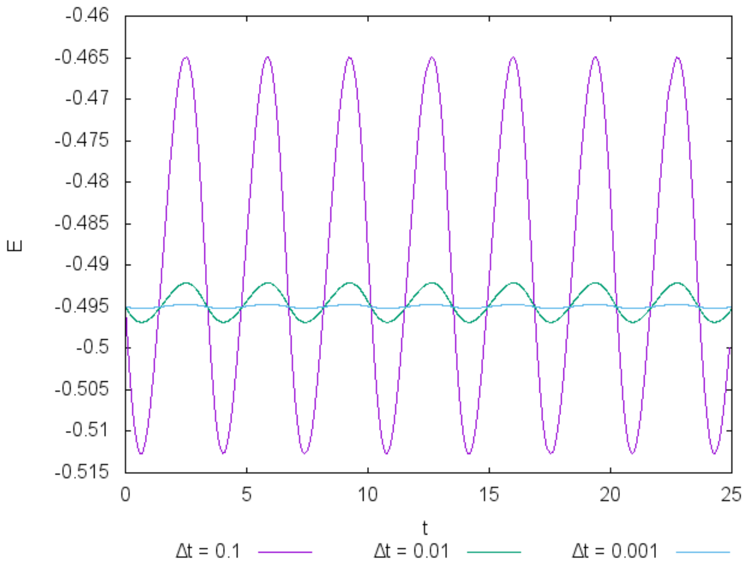

Figura 9: Energia de libração do pêndulo simples (versão modificada)

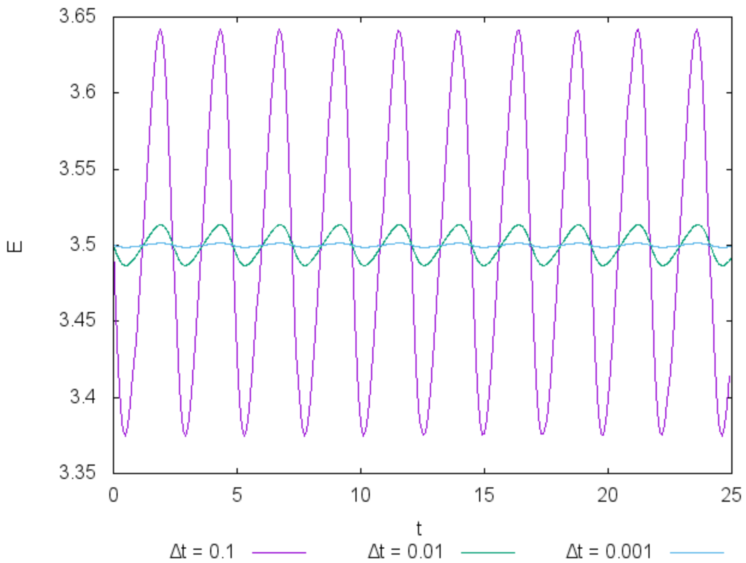

Figura 10: Energia de rotação do pêndulo simples (versão modificada)

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.

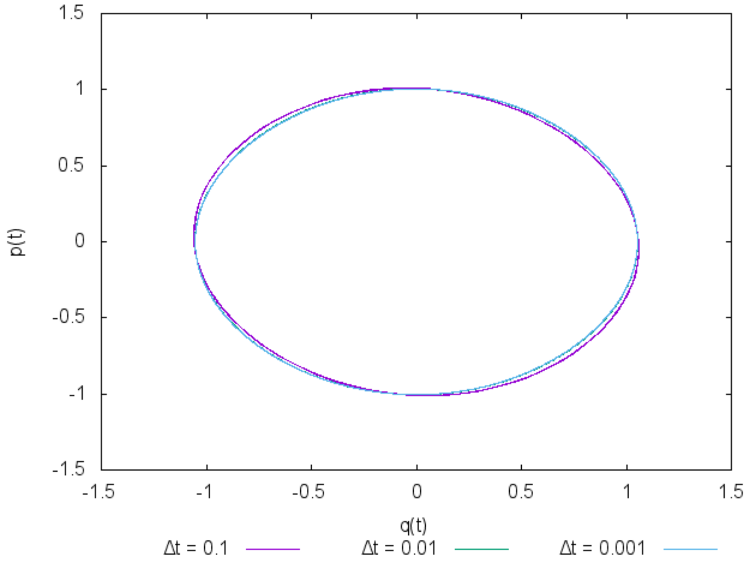

Figura 11: Trajetória de libração do simples (versão modificada)

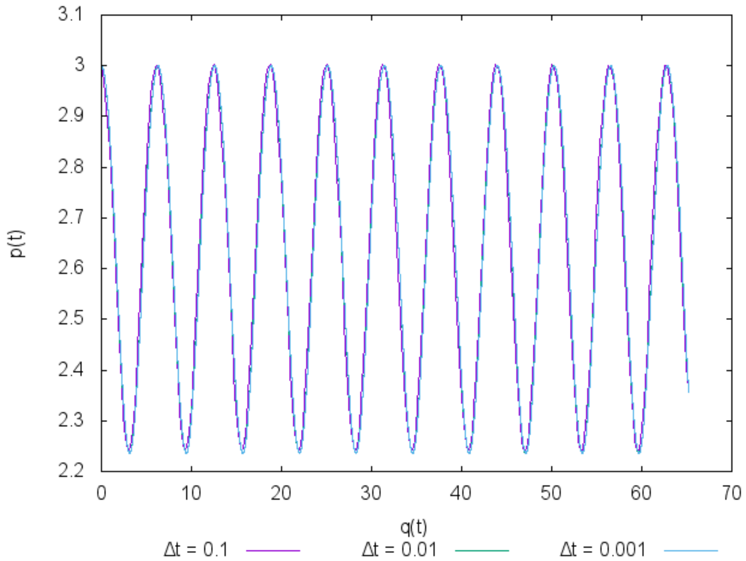

Figura 12: Trajetória de rotação do pêndulo simples (versão modificada)

Exercício 2.2.1

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

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

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

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

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:

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

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:

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

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

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

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

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

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-Helies 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.

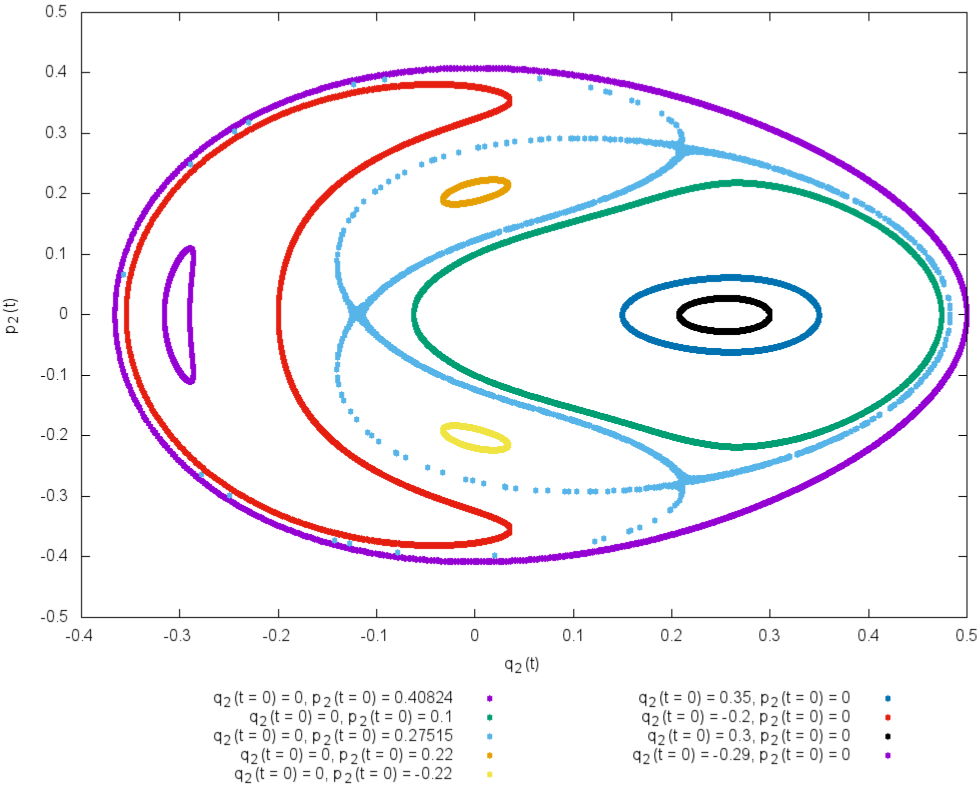

Figura 13: Seção de Poincaré para a Hamiltoniana de Hénon-Helies (E = 0.08333)

Exercício 2.3.4

Segue abaixo o mapa de Poincaré para a Hamiltoniana de Hénon-Helies com $E = 0.125$. Diferentemente da Figura 13, agora podemos observar o comportamento caótico desse sistema (ateriormente 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 unica órbita periódica. Também observamos um movimento quasi-periódico nos pontos em verde, que surgem muito próximos a uma separatriz desse sistema.

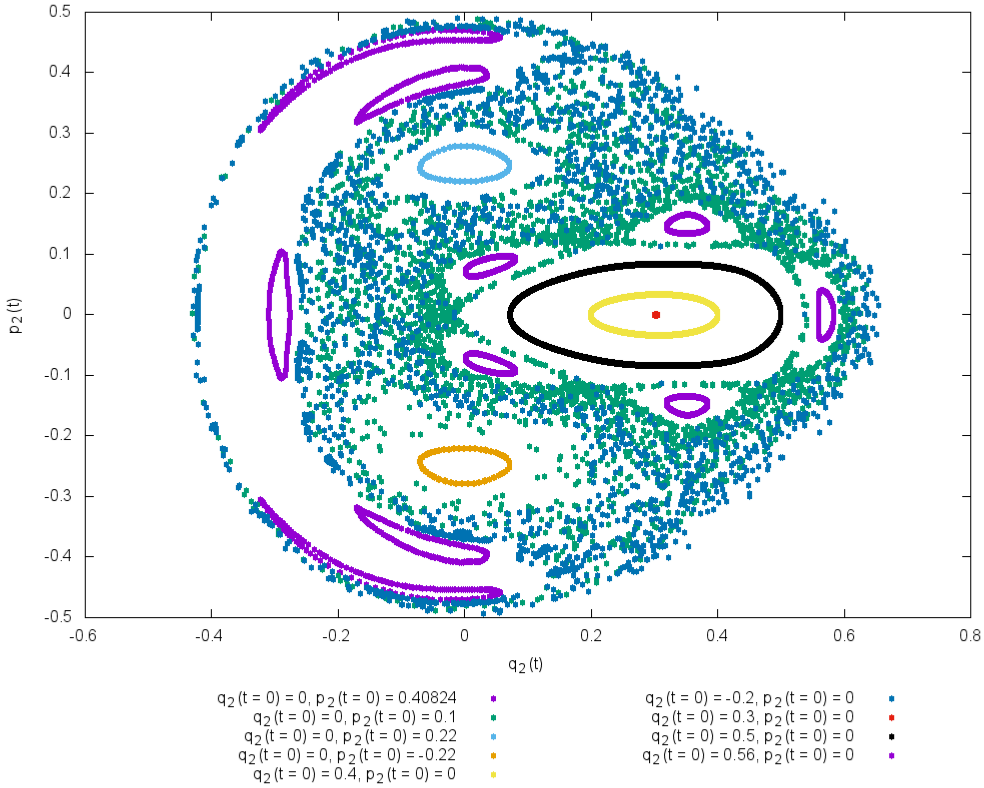

Figura 14: Seção de Poincaré para a Hamiltoniana de Hénon-Helies (E = 0.125)

Exercício 2.3.5

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

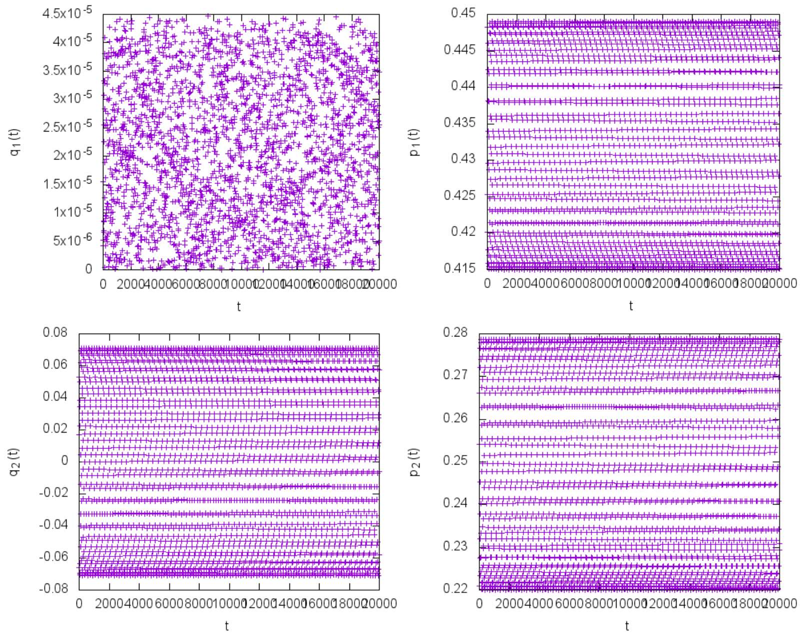

Figura 15: Trajetória regular para a Hamiltoniana de Hénon-Helies

Exercício 2.3.6

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

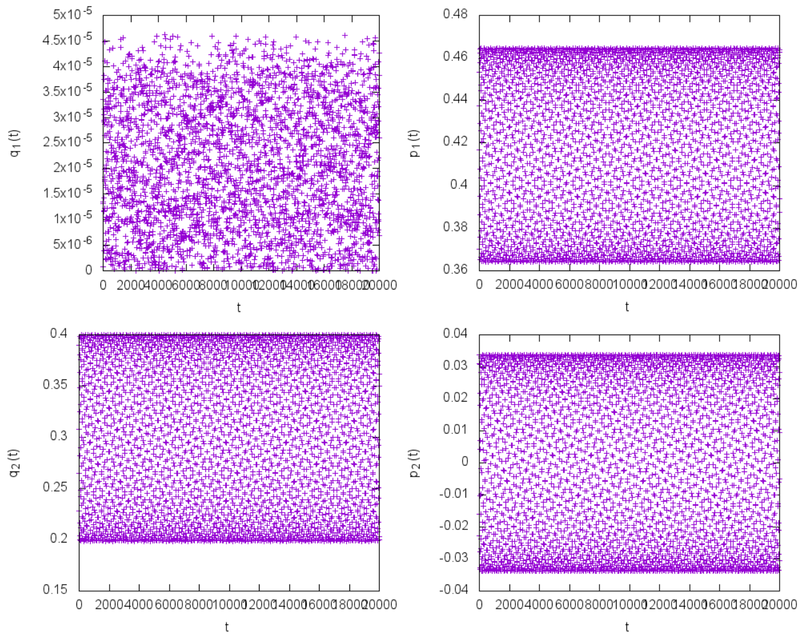

Figura 16: Trajetória caótica para a Hamiltoniana de Hénon-Helies

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.

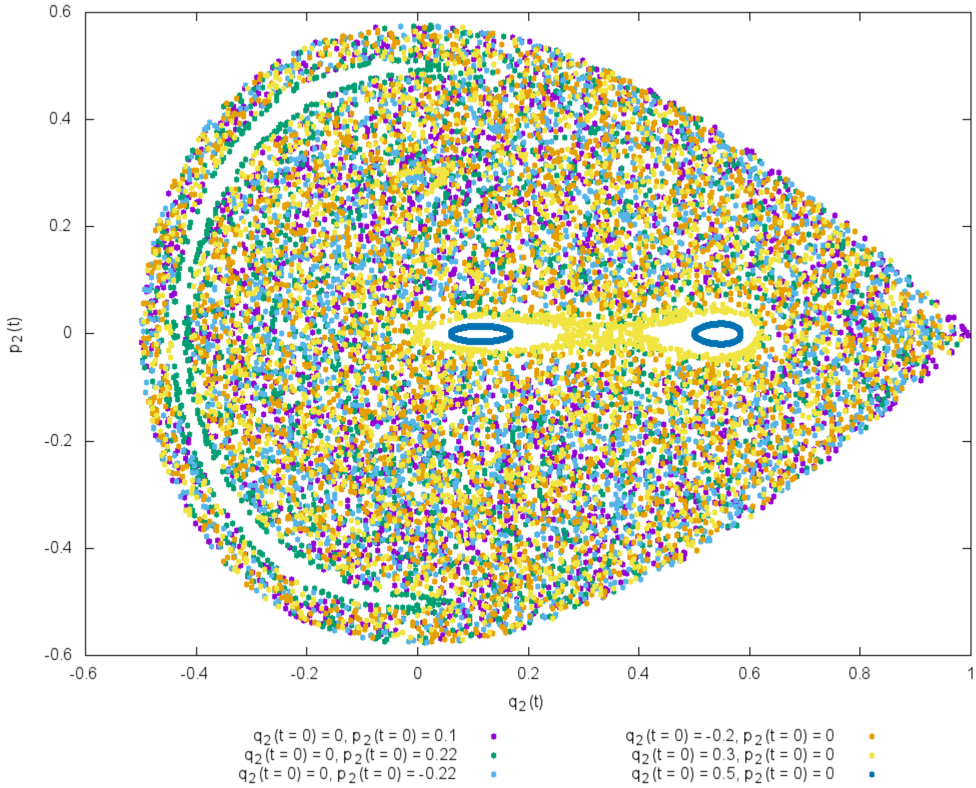

Figura 17: Seção de Poincaré para a Hamiltoniana de Hénon-Helies (E = 0.16667)

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

```
/*****************************************************
*             PGF 5005 - Mecânica Clássica            *
*               Primeiro Estudo Dirigido              *
*                2º Semestre de 2015                  *
*                                                     *
*                http://goo.gl/7PvBxi                 *
* Programa 1 - Equações de Euler para pêndulo simples *
*                                                     *
* Aluno: Rafael Mendonça Miller    NUSP.:7581818      *
*         e-mail: rafael.miller@usp.br                *
*****************************************************/

#include <iostream>
#include <cmath>
#include <fstream>
#include <vector>

using namespace std;

double hamiltoniana(double p, double q) {
```

```
  return (p*p)/2 - cos(q);
}

double dHamiltoniana_dp(double p) {
  return p;
}

double dHamiltoniana_dq(double q) {
  return -sin(q);
}

double q_nMaisUm(double q_n, double deltaT, double p_n) {
  return q_n + deltaT*dHamiltoniana_dp(p_n);
}

double p_nMaisUm(double p_n, double deltaT, double q_n) {
  return p_n - deltaT*dHamiltoniana_dq(q_n);
}

int main() {

  ofstream dadosLibracao_q_01, dadosLibracao_p_01, dadosLibracao_e_01;
  ofstream dadosLibracao_q_001, dadosLibracao_p_001, dadosLibracao_e_001;
  ofstream dadosLibracao_q_0001, dadosLibracao_p_0001, dadosLibracao_e_0001;
  ofstream dadosLibracao_pvq_01, dadosLibracao_pvq_001, dadosLibracao_pvq_0001;

  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");
  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")
    ;

  double T = 0.0;
  double deltaT = 0.1;

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

  // Condições Iniciais para Libração:
  P_n[0] = 0.25;
  Q_n[0] = 1.0;
  E_n[0] = hamiltoniana(P_n[0], Q_n[0]);

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

    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]);
    E_n[i+1] = hamiltoniana(P_n[i], Q_n[i]);

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

    T+=deltaT;
  }

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

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

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

```
    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]);
    E_n[i+1] = hamiltoniana(P_n[i], Q_n[i]);

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

    T+=deltaT;
  }

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

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

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

    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]);
    E_n[i+1] = hamiltoniana(P_n[i], Q_n[i]);

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

    T+=deltaT;
  }

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

  ofstream dadosRotacao_q_01, dadosRotacao_p_01, dadosRotacao_e_01;
  ofstream dadosRotacao_q_001, dadosRotacao_p_001, dadosRotacao_e_001;
  ofstream dadosRotacao_q_0001, dadosRotacao_p_0001, dadosRotacao_e_0001;
  ofstream dadosRotacao_pvq_01, dadosRotacao_pvq_001, dadosRotacao_pvq_0001;

  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");
  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");
  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");


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

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

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

    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]);
    E_n[i+1] = hamiltoniana(P_n[i], Q_n[i]);

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

```

```
    T+=deltaT;
  }

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

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

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

    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]);
    E_n[i+1] = hamiltoniana(P_n[i], Q_n[i]);

    dadosRotacao_q_001 << T << "\t" << Q_n[i] << endl;
    dadosRotacao_p_001 << T << "\t" << P_n[i] << endl;
    dadosRotacao_e_001 << T << "\t" << E_n[i] << endl;
    dadosRotacao_pvq_001 << Q_n[i] << "\t" << P_n[i] << endl;

    T+=deltaT;
  }

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

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

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

    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]);
    E_n[i+1] = hamiltoniana(P_n[i], Q_n[i]);

    dadosRotacao_q_0001 << T << "\t" << Q_n[i] << endl;
    dadosRotacao_p_0001 << T << "\t" << P_n[i] << endl;
    dadosRotacao_e_0001 << T << "\t" << E_n[i] << endl;
    dadosRotacao_pvq_0001 << Q_n[i] << "\t" << P_n[i] << endl;

    T+=deltaT;
  }



  return 0;
}
```

../Programa1/programa1.cpp

Programa 2 - Equações modificadas de Euler para o pêndulo simples

```
/*****************************************************************
*                    PGF 5005 - Mecânica Clássica                *
*                      Primeiro Estudo Dirigido                  *
*                        2º Semestre de 2015                     *
*                                                                *
*                        http://goo.gl/7PvBxi                    *
*  Programa 2 - Equações modificadas de Euler para pêndulo simples *
*                                                                *
*          Aluno: Rafael Mendonça Miller    NUSP.:7581818        *
*                 e-mail: rafael.miller@usp.br                   *
*****************************************************************/

#include <iostream>
```

```cpp
#include <cmath>
#include <fstream>
#include <vector>

using namespace std;

double hamiltoniana(double p, double q) {
  return (p*p)/2 - cos(q);
}

double dHamiltoniana_dp(double p) {
  return p;
}

double dHamiltoniana_dq(double q) {
  return sin(q);
}

double q_nMaisUm(double q_n, double deltaT, double p_n) {
  return q_n + deltaT*dHamiltoniana_dp(p_n);
}

double p_nMaisUm(double p_n, double deltaT, double q_n) {
  return p_n - deltaT*dHamiltoniana_dq(q_n);
}



int main() {

  ofstream dadosLibracao_q_01, dadosLibracao_p_01, dadosLibracao_e_01;
  ofstream dadosLibracao_q_001, dadosLibracao_p_001, dadosLibracao_e_001;
  ofstream dadosLibracao_q_0001, dadosLibracao_p_0001, dadosLibracao_e_0001;
  ofstream dadosLibracao_pvq_01, dadosLibracao_pvq_001, dadosLibracao_pvq_0001;

  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");
  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");

  double T = 0.0;
  double deltaT = 0.1;

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

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

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

    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]);
    E_n[i+1] = hamiltoniana(P_n[i+1], Q_n[i+1]);

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

    T+=deltaT;
  }
```

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

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

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

  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]);
  E_n[i+1] = hamiltoniana(P_n[i+1], Q_n[i+1]);

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

  T+=deltaT;
}

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

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

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

  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]);
  E_n[i+1] = hamiltoniana(P_n[i+1], Q_n[i+1]);

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

  T+=deltaT;
}

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

ofstream dadosRotacao_q_01, dadosRotacao_p_01, dadosRotacao_e_01;
ofstream dadosRotacao_q_001, dadosRotacao_p_001, dadosRotacao_e_001;
ofstream dadosRotacao_q_0001, dadosRotacao_p_0001, dadosRotacao_e_0001;
ofstream dadosRotacao_pvq_01, dadosRotacao_pvq_001, dadosRotacao_pvq_0001;

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");
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");
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");


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

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

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

  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]);
    E_n[i+1] = hamiltoniana(P_n[i+1], Q_n[i+1]);

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

    T+=deltaT;
  }

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

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

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

    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]);
    E_n[i+1] = hamiltoniana(P_n[i+1], Q_n[i+1]);

    dadosRotacao_q_001 << T << "\t" << Q_n[i] << endl;
    dadosRotacao_p_001 << T << "\t" << P_n[i] << endl;
    dadosRotacao_e_001 << T << "\t" << E_n[i] << endl;
    dadosRotacao_pvq_001 << Q_n[i] << "\t" << P_n[i] << endl;

    T+=deltaT;
  }

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

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

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

    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]);
    E_n[i+1] = hamiltoniana(P_n[i+1], Q_n[i+1]);

    dadosRotacao_q_0001 << T << "\t" << Q_n[i] << endl;
    dadosRotacao_p_0001 << T << "\t" << P_n[i] << endl;
    dadosRotacao_e_0001 << T << "\t" << E_n[i] << endl;
    dadosRotacao_pvq_0001 << Q_n[i] << "\t" << P_n[i] << endl;

    T+=deltaT;
  }

  return 0;
}
```

../Programa2/programa2.cpp

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

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

```cpp
*                                                                              *
*                  Aluno: Rafael Mendonça Miller   NUSP.:7581818               *
*                          e-mail: rafael.miller@usp.br                        *
********************************************************************************/

#include <iostream>
#include <cmath>
#include <fstream>
#include <vector>

using namespace std;

double hamiltoniana(double p1, double p2, double q1, double q2) {
  return (1./2)*(p1*p1 + p2*p2 + q1*q1 + q2*q2) + q1*q1*q2 - (1./3)*q2*q2*q2;
}

double dHamiltoniana_dp1(double p1) {
  return p1;
}

double dHamiltoniana_dp2(double p2) {
  return p2;
}

double dHamiltoniana_dq1(double q1, double q2) {
  return q1 + 2*q1*q2;
}

double dHamiltoniana_dq2(double q1, double q2) {
  return q2 + q1*q1 - q2*q2;
}

double q1_nMaisUm(double q1_n, double deltaT, double p1_n) {
  return q1_n + deltaT*dHamiltoniana_dp1(p1_n);
}

double q2_nMaisUm(double q2_n, double deltaT, double p2_n) {
  return q2_n + deltaT*dHamiltoniana_dp2(p2_n);
}

double p1_nMaisUm(double p1_n, double deltaT, double q1_n, double q2_n) {
  return p1_n - deltaT*dHamiltoniana_dq1(q1_n, q2_n);
}

double p2_nMaisUm(double p2_n, double deltaT, double q1_n, double q2_n) {
  return p2_n - deltaT*dHamiltoniana_dq2(q1_n, q2_n);
}

double p2s(double p1_n, double p2_n, double q1_n, double q2_n, double DeltaQ1) {
    return p2_n - DeltaQ1*(dHamiltoniana_dq2(q1_n, q2_n)/dHamiltoniana_dp1(p1_n));
}

double q2s(double p1_n, double p2_n, double q2_n, double DeltaQ1) {
    return q2_n + DeltaQ1*(dHamiltoniana_dp2(p2_n)/dHamiltoniana_dp1(p1_n));
}

int main() {

  double deltaT = 0.0001;

  double Q1_n;
  double Q1_nMaisUm;

  double Q2_n;
  double Q2_nMaisUm;

  double P1_n;
  double P1_nMaisUm;

  double P2_n;
  double P2_nMaisUm;
```

```
double E_n;

double Q2s;
double P2s;


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

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

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

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


for(int j = 0; j < 9; j ++) {

  ofstream dadosMapaDePoincare;
  string file_name = "out/dadosMapaDePoincare";
  file_name += to_string(j);
  file_name += ".dat";
  dadosMapaDePoincare.open(file_name);

  Q1_n = 0.0;

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

  cout << P2_n << "\t" << Q2_n << "\t" << file_name << endl;
  E_n = 0.08333;
  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));

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

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

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

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

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

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

    Q1_n = Q1_nMaisUm;
    Q2_n = Q2_nMaisUm;
    P1_n = P1_nMaisUm;
    P2_n = P2_nMaisUm;
  }

}

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

ofstream q1PeriodicaE0125,
         p1PeriodicaE0125,
         q2PeriodicaE0125,
         p2PeriodicaE0125;

```

```
    q1PeriodicaE0125.open("out/q1PeriodicaE0125.dat");
    p1PeriodicaE0125.open("out/p1PeriodicaE0125.dat");
    q2PeriodicaE0125.open("out/q2PeriodicaE0125.dat");
    p2PeriodicaE0125.open("out/p2PeriodicaE0125.dat");

    ofstream q1CaoticaE0125,
             p1CaoticaE0125,
             q2CaoticaE0125,
             p2CaoticaE0125;

    q1CaoticaE0125.open("out/q1CaoticaE0125.dat");
    p1CaoticaE0125.open("out/p1CaoticaE0125.dat");
    q2CaoticaE0125.open("out/q2CaoticaE0125.dat");
    p2CaoticaE0125.open("out/p2CaoticaE0125.dat");

    CondicaoInicial_P2[9] = 0.40824;
    CondicaoInicial_P2[10] = 0.1; // Quasi-Periódica ou Separatriz?
    CondicaoInicial_P2[11] = 0.22;
    CondicaoInicial_P2[12] = -0.22;

    CondicaoInicial_Q2[13] = 0.4; // Caótico
    CondicaoInicial_Q2[14] = -0.2;
    CondicaoInicial_Q2[15] = 0.3; // Centro da ilha
    CondicaoInicial_Q2[16] = 0.5;
    CondicaoInicial_Q2[17] = 0.56;

    for(int j = 9; j < 18; j++) {

      ofstream dadosMapaDePoincare;
      string file_name = "out/dadosMapaDePoincare";
      file_name += to_string(j);
      file_name += ".dat";
      dadosMapaDePoincare.open(file_name);

      Q1_n = 0.0;

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

      cout << P2_n << "\t" << Q2_n << "\t" << file_name << endl;
      E_n = 0.125;
      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));

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

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

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

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

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

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

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

          if(j == 13) {
            q1CaoticaE0125 << T << "\t" << Q1_nMaisUm << endl;
            p1CaoticaE0125 << T << "\t" << P1_nMaisUm << endl;
            q2CaoticaE0125 << T << "\t" << Q2_nMaisUm << endl;
            p2CaoticaE0125 << T << "\t" << P2_nMaisUm << endl;
```

```
        }
      }

      Q1_n = Q1_nMaisUm;
      Q2_n = Q2_nMaisUm;
      P1_n = P1_nMaisUm;
      P2_n = P2_nMaisUm;
    }

  }

  /************************** E = 0.16667: **********************************/

  CondicaoInicial_P2[18] = 0.1;
  CondicaoInicial_P2[19] = 0.22;
  CondicaoInicial_P2[20] = -0.22;

  CondicaoInicial_Q2[21] = -0.2;
  CondicaoInicial_Q2[22] = 0.3;
  CondicaoInicial_Q2[23] = 0.5;

  for(int j = 18; j < 24; j ++) {

    ofstream dadosMapaDePoincare;
    string file_name = "out/dadosMapaDePoincare";
    file_name += to_string(j);
    file_name += ".dat";
    dadosMapaDePoincare.open(file_name);

    Q1_n = 0.0;

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

    cout << P2_n << "\t" << Q2_n << "\t" << file_name << endl;
    E_n = 0.16667;
    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));

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

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

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

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

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

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

      Q1_n = Q1_nMaisUm;
      Q2_n = Q2_nMaisUm;
      P1_n = P1_nMaisUm;
      P2_n = P2_nMaisUm;
    }

  }

  return 0;
}
```

../Programa3/programa3.cpp