

# Universidade de São Paulo – USP

Instituto de Física

## Cálculo Numérico com Aplicações em Física - EP4

**Aluno:** Raphael Rolim

**Professor:** Arnaldo Gammal

Novembro/2024

# Conteúdo

<b>Parte I</b>	<b>1</b>
<b>Parte II</b>	<b>3</b>
Questão 1 . . . . .	3
Item a) . . . . .	5
Item b) . . . . .	6
Item c) . . . . .	7
Item d) . . . . .	9
Questão 2 . . . . .	10
Questão 3 . . . . .	12

# Parte I

```
1  #include <iostream>
2  #include <iomanip>
3  #include <vector>
4  #include <cmath>
5  #include <fstream>
6
7  // Define constants
8  const double h = 0.01;
9
10 double g(double t, double y, double z) {
11     double g_func;
12
13     g_func = z + y - (t * t * t) - 3 * (t * t) + 7 * t + 1;
14     return g_func;
15 }
16
17 double y_analytical(double t) {
18     double y_func;
19
20     y_func = (t * t * t) - t;
21     return y_func;
22 }
23
24 double z_analytical(double t) {
25     double z_func;
26
27     z_func = 3 * (t * t) - 1;
28     return z_func;
29 }
30
31 void euler(double t, double& y, double& z, double h) {
32     double y_prime, z_prime;
33
34     y_prime = y;
35     z_prime = z;
36     y = y_prime + h * z_prime;
37     z = z_prime + h * g(t, y_prime, z_prime);
38 }
39
40 void rk4(double t, double& y, double& z, double h) {
41     double k1y, k1z, k2y, k2z, k3y, k3z, k4y, k4z;
42
43     k1y = h * z;
44     k1z = h * g(t, y, z);
45     k2y = h * (z + k1z / 2);
46     k2z = h * g(t + h / 2, y + k1y / 2, z + k1z / 2);
47     k3y = h * (z + k2z / 2);
48     k3z = h * g(t + h / 2, y + k2y / 2, z + k2z / 2);
49     k4y = h * (z + k3z);
50     k4z = h * g(t + h, y + k3y, z + k3z);
51     y += (k1y + 2 * k2y + 2 * k3y + k4y) / 6;
52     z += (k1z + 2 * k2z + 2 * k3z + k4z) / 6;
53 }
54
55 int main() {
56     double y = 0, z = -1; // Initial conditions
57     std::cout << "##### Euler Method #####" << std::endl;
58     for (double t = 0; t <= 5; t += h) {
59         euler(t, y, z, h);
60     }
```

```

61 // Print in double precision (default)
62 std::cout << "#----- Double Precision -----#" << std::endl;
63 std::cout << " y(5): " << std::setprecision(15) << std::fixed << "Euler Method: " << y <<
64 " | " << "Analytical: " << y_analytical(5) << "\n"
65 << " dy/dt(5): " << std::setprecision(15) << std::fixed << "Euler Method: " << z
66 << " | " << "Analytical: " << z_analytical(5) << std::endl;
67
68 // Print in single precision (float)
69 std::cout << "#----- Single Precision -----#" << std::endl;
70 std::cout << " y(5): " << std::setprecision(7) << std::fixed << "Euler Method: " <<
71 static_cast<float>(y) << " | " << "Analytical: " << y_analytical(5) << "\n"
72 << " dy/dt(5): " << std::setprecision(7) << std::fixed << "Euler Method: " <<
73 static_cast<float>(z) << " | " << "Analytical: " << z_analytical(5) << std::
74 endl;
75
76 y = 0, z = -1;
77 std::cout << "#----- RK4 Method -----#" << std::endl;
78 for (double t = 0; t <= 5; t += h) {
79     rk4(t, y, z, h);
80 }
81 // Print in double precision (default)
82 std::cout << "#----- Double Precision -----#" << std::endl;
83 std::cout << " y(5): " << std::setprecision(15) << std::fixed << "Euler Method: " << y <<
84 " | " << "Analytical: " << y_analytical(5) << "\n"
85 << " dy/dt(5): " << std::setprecision(15) << std::fixed << "Euler Method: " << z
86 << " | " << "Analytical: " << z_analytical(5) << std::endl;
87
88 // Print in single precision (float)
89 std::cout << "#----- Single Precision -----#" << std::endl;
90 std::cout << " y(5): " << std::setprecision(7) << std::fixed << "Euler Method: " <<
91 static_cast<float>(y) << " | " << "Analytical: " << y_analytical(5) << "\n"
92 << " dy/dt(5): " << std::setprecision(7) << std::fixed << "Euler Method: " <<
93 static_cast<float>(z) << " | " << "Analytical: " << z_analytical(5) << std::
94 endl;
95
96 return 0;
97 }

```

Para as funções analíticas, temos que

$$\begin{cases} y(5) = 120 \\ \frac{dy(5)}{dt} = 74 \end{cases}$$

Para o método de Euler, obtemos

$$\text{Precisão dupla: } \begin{cases} y(5) = 85.019517400804219 \\ \frac{dy(5)}{dt} = 16.726898599884368 \end{cases} \quad \text{Precisão simples: } \begin{cases} y(5) = 85.0195160 \\ \frac{dy(5)}{dt} = 16.7268982 \end{cases}$$

Agora, para o método RK4, recebemos

$$\text{Precisão dupla: } \begin{cases} y(5) = 120.741497989331009 \\ \frac{dy(5)}{dt} = 74.300295127327132 \end{cases} \quad \text{Precisão simples: } \begin{cases} y(5) = 120.7415009 \\ \frac{dy(5)}{dt} = 74.3002930 \end{cases}$$

## Parte II

### Questão 1

```
1  #include <iostream>
2  #include <iomanip>
3  #include <vector>
4  #include <string>
5  #include <cmath>
6  #include <fstream>
7  #include <sstream>
8
9  // Define constants
10 const double w = 1.00; // Omega
11
12 double func(double t, double x, double v, double F, double g, int item) {
13     double result;
14     if (item == 1) {
15         result = 0.5 * x * (1 - 4 * x * x);
16     }
17     if (item == 2) {
18         result = 0.5 * x * (1 - 4 * x * x) - g * v;
19     }
20     if (item == 3) {
21         result = F * cos(w * t) + 0.5 * x * (1 - 4 * x * x) - 0.25 * v;
22     }
23     return result;
24 }
25
26 void rk4(double& t, double& y, double& z, double h, double F, double g, int item) {
27     double k1y, k1z, k2y, k2z, k3y, k3z, k4y, k4z;
28
29     k1y = h * z;
30     k1z = h * func(t, y, z, F, g, item);
31     k2y = h * (z + k1z / 2);
32     k2z = h * func(t + h / 2, y + k1y / 2, z + k1z / 2, F, g, item);
33     k3y = h * (z + k2z / 2);
34     k3z = h * func(t + h / 2, y + k2y / 2, z + k2z / 2, F, g, item);
35     k4y = h * (z + k3z);
36     k4z = h * func(t + h, y + k3y, z + k3z, F, g, item);
37     y += (k1y + 2 * k2y + 2 * k3y + k4y) / 6;
38     z += (k1z + 2 * k2z + 2 * k3z + k4z) / 6;
39     t += h;
40 }
41
42 int main() {
43     // Item a)
44     std::vector<std::ofstream> files_a(3);
45     double v_vec[3] = {0.1, 0.25, 0.5};
46     for (int i = 0; i < 3; ++i) {
47         std::ostringstream filename;
48         filename << "item_a-" << i + 1 << ".txt";
49         files_a[i].open(filename.str());
50
51         double t = 0.0, x = -0.5; // Initial conditions
52         double v = v_vec[i];
53         double h = 0.001;
54         double F = 0, g = 0;
55         for (int j = 0; j < 1000000; ++j) {
56             rk4(t, x, v, h, F, g, 1);
57             files_a[i] << x << " " << v << "\n";
```

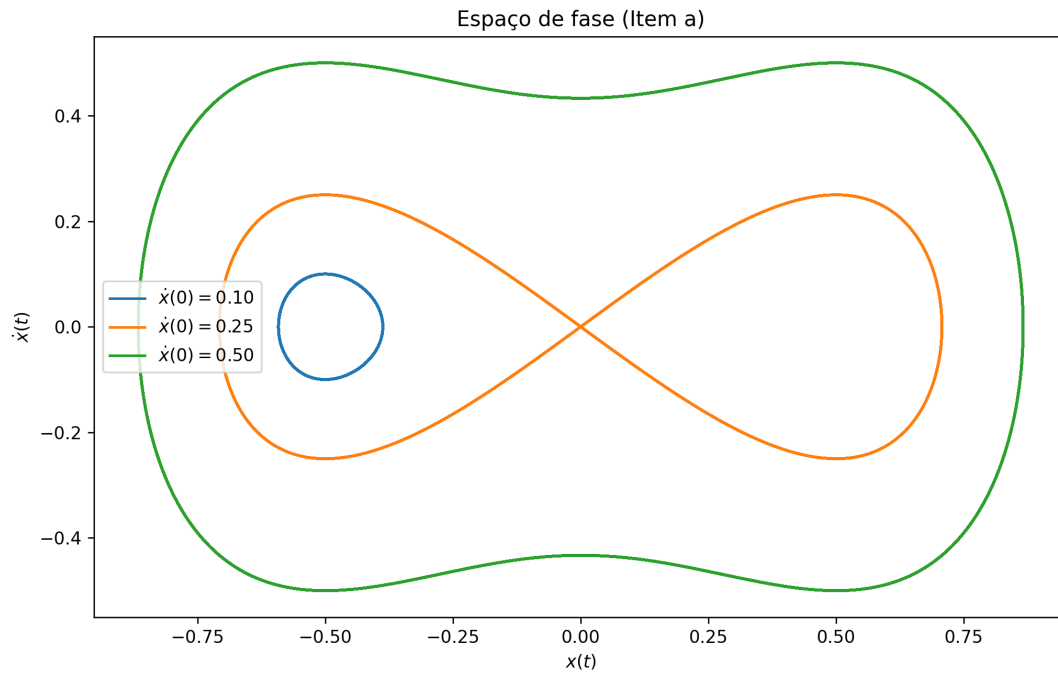
```

58     }
59     files_a[i].close();
60 }
61
62 // Item b)
63 std::vector<std::ofstream> files_b(2);
64 double g_vec[3] = {0.25, 0.8};
65 for (int i = 0; i < 2; ++i) {
66     std::ostringstream filename;
67     filename << "item_b-" << i + 1 << ".txt";
68     files_b[i].open(filename.str());
69
70     double t = 0.0, x = -0.5, v = 0.5; // Initial conditions
71     double g = g_vec[i];
72     double h = 0.001;
73     double F = 0;
74     for (int j = 0; j < 100000; ++j) {
75         rk4(t, x, v, h, F, g, 2);
76         files_b[i] << x << " " << v << "\n";
77     }
78     files_b[i].close();
79 }
80
81 // Item c)
82 std::vector<std::ofstream> files_c(4);
83 double F_vec[5] = {0.11, 0.115, 0.14, 0.35};
84 for (int i = 0; i < 4; ++i) {
85     std::ostringstream filename;
86     filename << "item_c-" << i + 1 << ".txt";
87     files_c[i].open(filename.str());
88
89     double t = 0.0, x = -0.5, v = 0.5; // Initial conditions
90     double F = F_vec[i];
91     double h = 0.01; // Initial h
92     double g = 0.0;
93     for (int j = 0; j < 200000; ++j) {
94         rk4(t, x, v, h, F, g, 3);
95     }
96     h = 0.001; // Update h
97     for (int j = 0; j < 200000; ++j) {
98         rk4(t, x, v, h, F, g, 3);
99         files_c[i] << x << " " << v << "\n";
100     }
101     files_c[i].close();
102 }
103 return 0;
104 }

```

**Item a)**

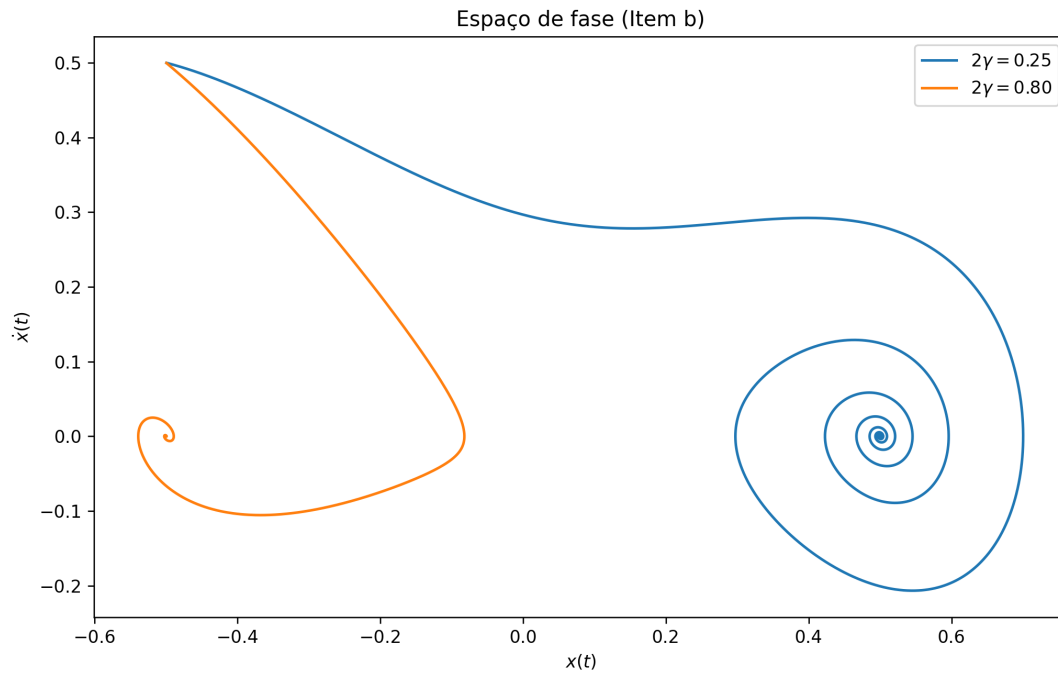
Utilizando o método RK4 para a equação do potencial poço duplo, obtemos a Figura 1.



**Figura 1:** Diagrama do espaço de fase do oscilador de Duffing para  $\dot{x}(0) = 0.10$  (azul),  $0.25$  (laranja),  $0.50$  (verde).

**Item b)**

Agora, incluindo o amortecimento, obtemos a Figura 2.

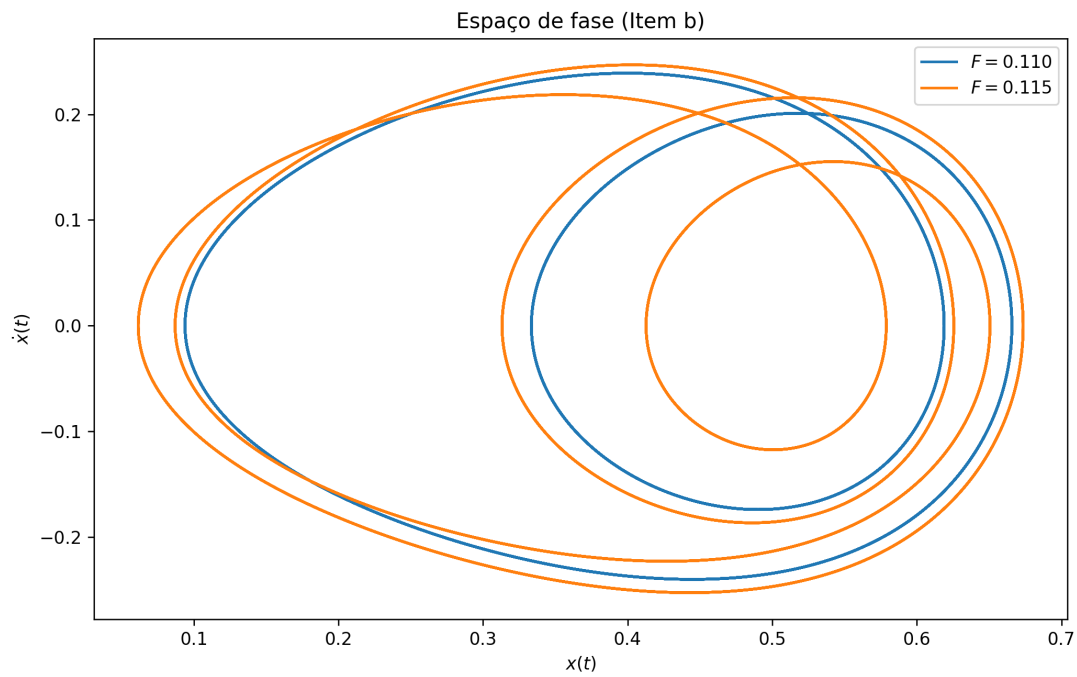


**Figura 2:** Diagrama do espaço de fase do oscilador de Duffing amortecido para  $2\gamma = 0.25$  (azul),  $0.80$  (laranja).

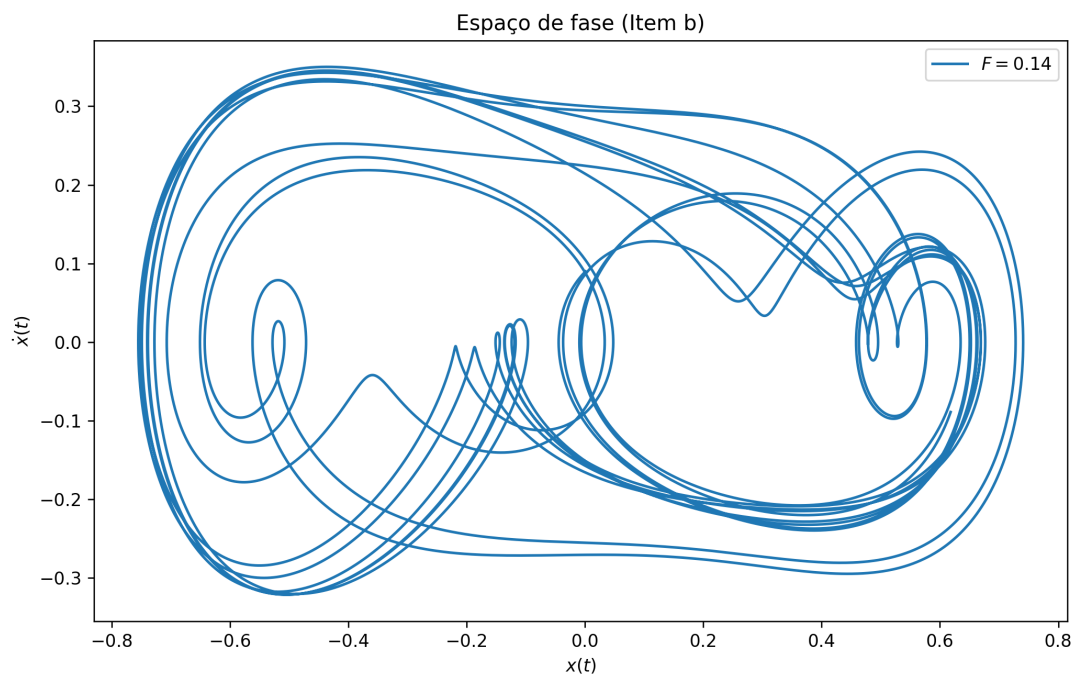


**Item c)**

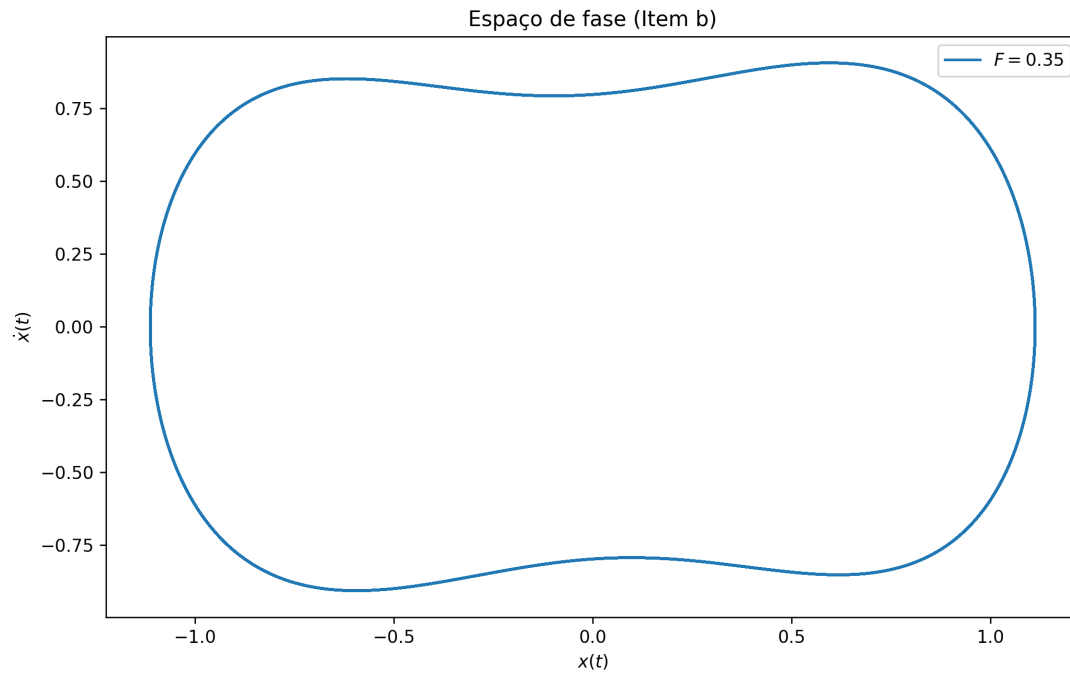
Agora, forçando o sistema ao variar  $F$ , obtemos as Figuras 3, 4 e 5.



**Figura 3:** Diagrama do espaço de fase do oscilador de Duffing amortecido forçado para  $F = 0.110$  (azul),  $0.115$  (laranja).



**Figura 4:** Diagrama do espaço de fase do oscilador de Duffing amortecido forçado para  $F = 0.140$ .



**Figura 5:** Diagrama do espaço de fase do oscilador de Duffing amortecido forçado para  $F = 0.35$ .

#### Item d)

Na Figura 1, todos os casos do espaço de fase têm um comportamento periódico, então temos que o atrator desse caso é o de ciclos limite.

Na Figura 2, cada caso do espaço de fase tende a um ponto específico, sendo assim ele tem um atrator pontual.

Na Figura 3, os casos têm um atrator de ciclos limite.

Na Figura 4, o diagrama não parece apresentar uma órbita fechada, então o atrator desse caso deve ser caótico.

E por último, na Figura 5, o diagrama apresenta órbita fechada, logo tem um atrator de ciclos limite.

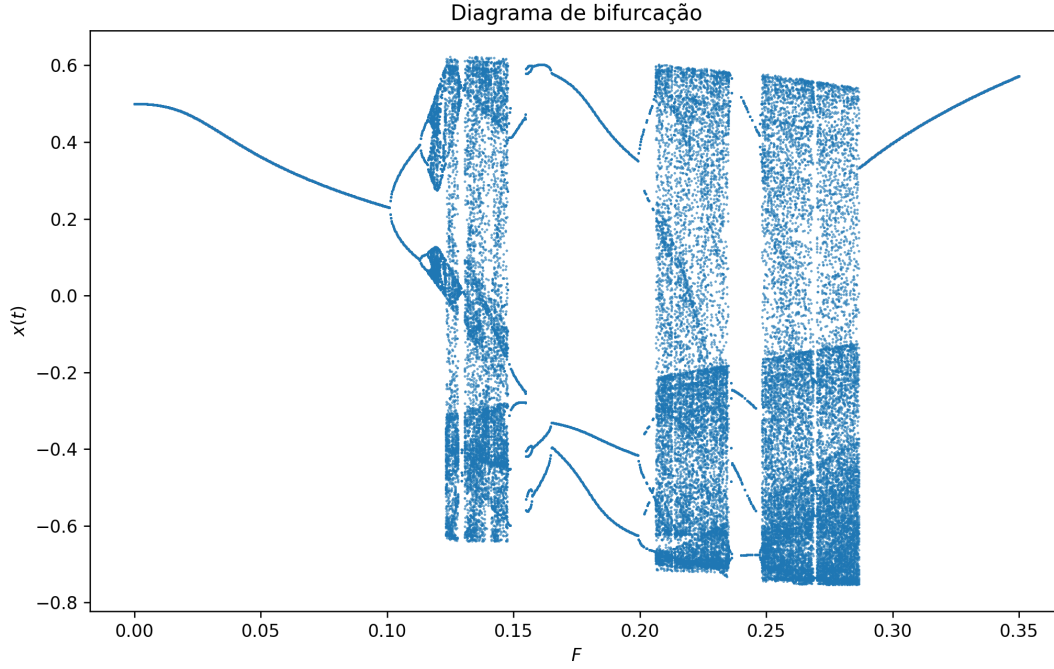
## Questão 2

```
1  #include <iostream>
2  #include <iomanip>
3  #include <vector>
4  #include <cmath>
5  #include <fstream>
6
7  // Define constants
8  const double d = 0.50; // Delta
9  const double w = 1.00; // Omega
10 const double g = 0.25; // Gamma
11 const double F_start = 0.0;
12 const double F_end = 0.35;
13 const double F_step = 0.00025;
14 const int transient_steps = 200000;
15
16 double forced(double t, double x, double v, double F) {
17     double result;
18
19     result = -g * v + d * x * (1 - 4 * x * x) + F * cos(w * t);
20     return result;
21 }
22
23 void rk4_forced(double& t, double& y, double& z, double h, double F) {
24     double k1y, k1z, k2y, k2z, k3y, k3z, k4y, k4z;
25
26     k1y = h * z;
27     k1z = h * forced(t, y, z, F);
28     k2y = h * (z + k1z / 2);
29     k2z = h * forced(t + h / 2, y + k1y / 2, z + k1z / 2, F);
30     k3y = h * (z + k2z / 2);
31     k3z = h * forced(t + h / 2, y + k2y / 2, z + k2z / 2, F);
32     k4y = h * (z + k3z);
33     k4z = h * forced(t + h, y + k3y, z + k3z, F);
34     y += (k1y + 2 * k2y + 2 * k3y + k4y) / 6;
35     z += (k1z + 2 * k2z + 2 * k3z + k4z) / 6;
36     t += h;
37 }
38
39 int main() {
40     std::ofstream bifurcation_file("bifurcation_data.txt");
41     std::cout << "Starting main function.\n" << std::endl;
42     for (double F = F_start; F <= F_end; F += F_step) {
43         double t = 0;
44         double x = -0.5, v = 0.5; // Initial conditions
45
46         double h = 0.01 * (2 * M_PI / w); // Initial h
47         for (int i = 0; i < transient_steps; ++i) {
48             rk4_forced(t, x, v, h, F);
49         }
50
51         h = 0.001 * (2 * M_PI / w); // Update h
52         for (int i = 0; i < 100; ++i) {
53             // Advance one period
54             for (int j = 0; j < 1000; ++j) {
55                 rk4_forced(t, x, v, h, F);
56             }
57             // Save the point on the Poincaré section
58             bifurcation_file << F << " " << x << "\n";
59         }
60     }
```

```

60         std::cout << "\rProgress of F steps: " << std::setw(3) << 100 * (F / F_end) << "%
           complete          " << std::flush;
61     }
62     std::cout << "\nFinished!" << std::endl;
63     bifurcation_file.close();
64     std::cout << "Bifurcation data saved to bifurcation_data.txt\n";
65     return 0;
66 }

```



**Figura 6:** Diagrama de bifurcação para o oscilador de Duffing amortecido forçado.

A constante de Feigenbaum  $\delta$  é definida como

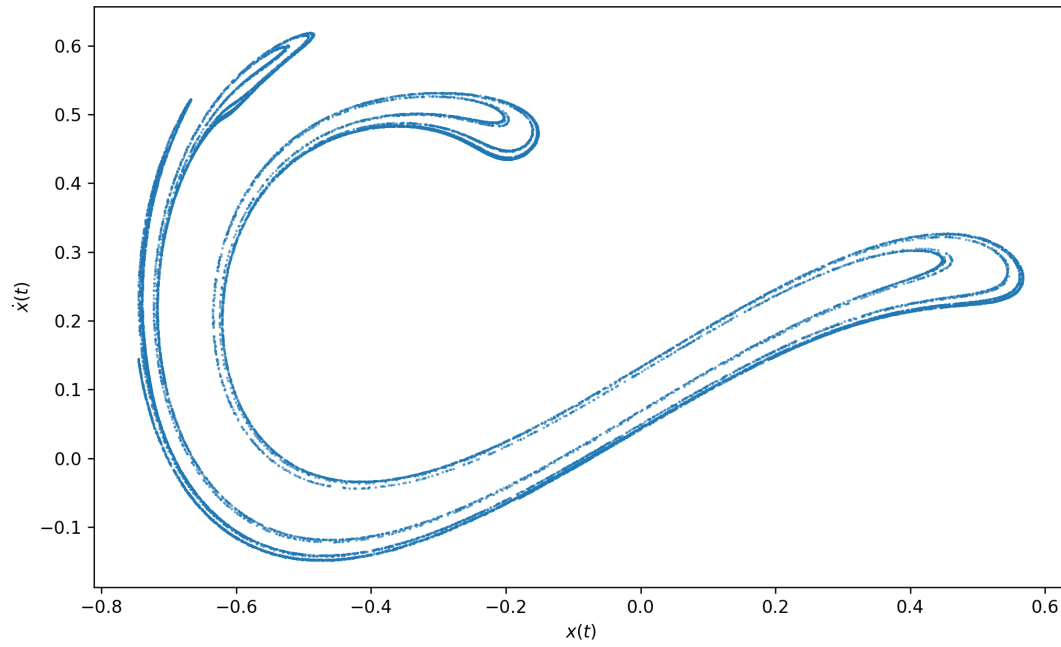
$$\delta = \frac{\Delta F_{n-1}}{\Delta F_n}, \quad (1)$$

onde  $n$  é o número da bifurcação. Olhando para a Figura 6, temos bifurcações nos pontos  $F_1 \approx 0.11$ ,  $F_2 \approx 0.20$  e  $F_3 \approx 0.225$ , e obtemos que  $\delta \approx 4$  (onde a constante de Feigenbaum real é  $\delta_{\text{real}} \approx 4.6692$ ). Esse valor é apenas uma estimativa, já que a precisão para o cálculo depende muito de como escolhemos os pontos a serem considerados.

### Questão 3

```
1  #include <iostream>
2  #include <iomanip>
3  #include <vector>
4  #include <cmath>
5  #include <fstream>
6
7  // Define constants
8  const double d = 0.50; // Delta
9  const double w = 1.00; // Omega
10 const double g = 0.25; // Gamma
11 const double F = 0.26;
12
13 double forced(double t, double x, double v, double F) {
14     double result;
15
16     result = -g * v + d * x * (1 - 4 * x * x) + F * cos(w * t);
17     return result;
18 }
19
20 void rk4_forced(double& t, double& y, double& z, double h, double F) {
21     double k1y, k1z, k2y, k2z, k3y, k3z, k4y, k4z;
22
23     k1y = h * z;
24     k1z = h * forced(t, y, z, F);
25     k2y = h * (z + k1z / 2);
26     k2z = h * forced(t + h / 2, y + k1y / 2, z + k1z / 2, F);
27     k3y = h * (z + k2z / 2);
28     k3z = h * forced(t + h / 2, y + k2y / 2, z + k2z / 2, F);
29     k4y = h * (z + k3z);
30     k4z = h * forced(t + h, y + k3y, z + k3z, F);
31     y += (k1y + 2 * k2y + 2 * k3y + k4y) / 6;
32     z += (k1z + 2 * k2z + 2 * k3z + k4z) / 6;
33     t += h;
34 }
35
36 int main() {
37     std::ofstream vXx_file("vXx.txt");
38     std::cout << "Starting main function.\n" << std::endl;
39     double t = 0;
40     double x = -0.5, v = 0.5; // Initial conditions
41     double h = 0.01 * (2 * M_PI / w); // Initial h
42     for (int i = 0; i < 200000; ++i) {
43         rk4_forced(t, x, v, h, F);
44     }
45
46     h = 0.001 * (2 * M_PI / w); // Update h
47     for (int i = 0; i < 20000; ++i) {
48         // Advance one period
49         for (int j = 0; j < 1000; ++j) {
50             rk4_forced(t, x, v, h, F);
51         }
52         // Save the point on the Poincaré section
53         vXx_file << v << " " << x << "\n";
54     }
55     std::cout << "\nFinished!" << std::endl;
56     vXx_file.close();
57     std::cout << "Bifurcation data saved to vXx.txt\n";
58     return 0;
59 }
```

Podemos ver o Mapa de Poincaré gerado na Figura 7.



**Figura 7:** Mapa de Poincaré para o oscilador de Duffing amortecido forçado com  $F = 0.26$ .