Universidade de São Paulo – USP

Instituto de Física

Cálculo Numérico com Aplicações em Física - EP4

Aluno: Raphael Rolim
Professor: Arnaldo Gammal

Novembro/2024

Conteúdo

Parte I

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

// Define constants
const double h = 0.01;

double g(double t, double y, double z) {
    double g_func;

    g_func = z + y - (t * t * t) - 3 * (t * t) + 7 * t + 1;
    return g_func;
}

double y_analytical(double t) {
    double y_func;

    y_func = (t * t * t) - t;
    return y_func;
}

double z_analytical(double t) {
    double z_func;

    z_func = 3 * (t * t) - 1;
    return z_func;
}

void euler(double t, double& y, double& z, double h) {
    double y_prime, z_prime;

    y_prime = y;
    z_prime = z;
    y = y_prime + h * z_prime;
    z = z_prime + h * g(t, y_prime, z_prime);
}

void rk4(double t, double& y, double& z, double h) {
    double k1y, k1z, k2y, k2z, k3y, k3z, k4y, k4z;

    k1y = h * z;
    k1z = h * g(t, y, z);
    k2y = h * (z + k1z / 2);
    k2z = h * g(t + h / 2, y + k1y / 2, z + k1z / 2);
    k3y = h * (z + k2z / 2);
    k3z = h * g(t + h / 2, y + k2y / 2, z + k2z / 2);
    k4y = h * (z + k3z);
    k4z = h * g(t + h, y + k3y, z + k3z);
    y += (k1y + 2 * k2y + 2 * k3y + k4y) / 6;
    z += (k1z + 2 * k2z + 2 * k3z + k4z) / 6;
}

int main() {
    double y = 0, z = -1; // Initial conditions
    std::cout << "#------- Euler Method -------#" << std::endl;
    for (double t = 0; t <= 5; t += h) {
        euler(t, y, z, h);
    }
```

```
    // Print in double precision (default)
    std::cout << "#-------- Double Precision --------#" << std::endl;
        std::cout << " y(5): " << std::setprecision(15) << std::fixed << "Euler Method: " << y << " | " << "Analytical: " << y_analytical(5) << "\n"
                  << " dy/dt(5): " << std::setprecision(15) << std::fixed << "Euler Method: " << z << " | " << "Analytical: " << z_analytical(5) << std::endl;

        // Print in single precision (float)
        std::cout << "#-------- Single Precision --------#" << std::endl;
        std::cout << " y(5): " << std::setprecision(7) << std::fixed << "Euler Method: " << static_cast<float>(y) << " | " << "Analytical: " << y_analytical(5) << "\n"
                  << " dy/dt(5): " << std::setprecision(7) << std::fixed << "Euler Method: " << static_cast<float>(z) << " | " << "Analytical: " << z_analytical(5) << std::endl;

        y = 0, z = -1;
        std::cout << "#-------- RK4 Method --------#" << std::endl;
    for (double t = 0; t <= 5; t += h) {
        rk4(t, y, z, h);
    }
    // Print in double precision (default)
    std::cout << "#-------- Double Precision --------#" << std::endl;
        std::cout << " y(5): " << std::setprecision(15) << std::fixed << "Euler Method: " << y << " | " << "Analytical: " << y_analytical(5) << "\n"
                  << " dy/dt(5): " << std::setprecision(15) << std::fixed << "Euler Method: " << z << " | " << "Analytical: " << z_analytical(5) << std::endl;

        // Print in single precision (float)
        std::cout << "#-------- Single Precision --------#" << std::endl;
        std::cout << " y(5): " << std::setprecision(7) << std::fixed << "Euler Method: " << static_cast<float>(y) << " | " << "Analytical: " << y_analytical(5) << "\n"
                  << " dy/dt(5): " << std::setprecision(7) << std::fixed << "Euler Method: " << static_cast<float>(z) << " | " << "Analytical: " << z_analytical(5) << std::endl;
    return 0;
}
```

Para as funções analíticas, temos que

$$\begin{cases} y(5) = 120 \\ \dfrac{\mathrm{d}y(5)}{\mathrm{d}t} = 74 \end{cases}$$

Para o método de Euler, obtemos

Precisão dupla: $\begin{cases} y(5) = 85.019517400804219 \\ \dfrac{\mathrm{d}y(5)}{\mathrm{d}t} = 16.726898599884368 \end{cases}$ Precisão simples: $\begin{cases} y(5) = 85.0195160 \\ \dfrac{\mathrm{d}y(5)}{\mathrm{d}t} = 16.7268982 \end{cases}$

Agora, para o método RK4, recebemos

Precisão dupla: $\begin{cases} y(5) = 120.741497989331009 \\ \dfrac{\mathrm{d}y(5)}{\mathrm{d}t} = 74.300295127327132 \end{cases}$ Precisão simples: $\begin{cases} y(5) = 120.7415009 \\ \dfrac{\mathrm{d}y(5)}{\mathrm{d}t} = 74.3002930 \end{cases}$

Parte II

Questão 1

```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
#include <cmath>
#include <fstream>
#include <sstream>

// Define constants
const double w = 1.00; // Omega

double func(double t, double x, double v, double F, double g, int item) {
    double result;
    if (item == 1) {
        result = 0.5 * x * (1 - 4 * x * x);
    }
    if (item == 2) {
        result = 0.5 * x * (1 - 4 * x * x) - g * v;
    }
    if (item == 3) {
        result = F * cos(w * t) + 0.5 * x * (1 - 4 * x * x) - 0.25 * v;
    }
    return result;
}

void rk4(double& t, double& y, double& z, double h, double F, double g, int item) {
    double k1y, k1z, k2y, k2z, k3y, k3z, k4y, k4z;

    k1y = h * z;
    k1z = h * func(t, y, z, F, g, item);
    k2y = h * (z + k1z / 2);
    k2z = h * func(t + h / 2, y + k1y / 2, z + k1z / 2, F, g, item);
    k3y = h * (z + k2z / 2);
    k3z = h * func(t + h / 2, y + k2y / 2, z + k2z / 2, F, g, item);
    k4y = h * (z + k3z);
    k4z = h * func(t + h, y + k3y, z + k3z, F, g, item);
    y += (k1y + 2 * k2y + 2 * k3y + k4y) / 6;
    z += (k1z + 2 * k2z + 2 * k3z + k4z) / 6;
    t += h;
}

int main() {
    // Item a)
    std::vector<std::ofstream> files_a(3);
    double v_vec[3] = {0.1, 0.25, 0.5};
    for (int i = 0; i < 3; ++i) {
        std::ostringstream filename;
            filename << "item_a-" << i + 1 << ".txt";
            files_a[i].open(filename.str());

        double t = 0.0, x = -0.5; // Initial conditions
        double v = v_vec[i];
        double h = 0.001;
        double F = 0, g = 0;
        for (int j = 0; j < 1000000; ++j) {
            rk4(t, x, v, h, F, g, 1);
            files_a[i] << x << " " << v << "\n";
```

```
        }
        files_a[i].close();
    }

        // Item b)
        std::vector<std::ofstream> files_b(2);
    double g_vec[3] = {0.25, 0.8};
    for (int i = 0; i < 2; ++i) {
        std::ostringstream filename;
            filename << "item_b—" << i + 1 << ".txt";
            files_b[i].open(filename.str());

        double t = 0.0, x = —0.5, v = 0.5; // Initial conditions
        double g = g_vec[i];
        double h = 0.001;
        double F = 0;
        for (int j = 0; j < 100000; ++j) {
            rk4(t, x, v, h, F, g, 2);
            files_b[i] << x << " " << v << "\n";
        }
        files_b[i].close();
    }

    // Item c)
    std::vector<std::ofstream> files_c(4);
    double F_vec[5] = {0.11, 0.115, 0.14, 0.35};
    for (int i = 0; i < 4; ++i) {
        std::ostringstream filename;
            filename << "item_c—" << i + 1 << ".txt";
            files_c[i].open(filename.str());

        double t = 0.0, x = —0.5, v = 0.5; // Initial conditions
        double F = F_vec[i];
        double h = 0.01; // Initial h
        double g = 0.0;
        for (int j = 0; j < 200000; ++j) {
            rk4(t, x, v, h, F, g, 3);
        }
        h = 0.001; // Update h
        for (int j = 0; j < 200000; ++j) {
            rk4(t, x, v, h, F, g, 3);
            files_c[i] << x << " " << v << "\n";
        }
        files_c[i].close();
    }
        return 0;
}
```

Item a)

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

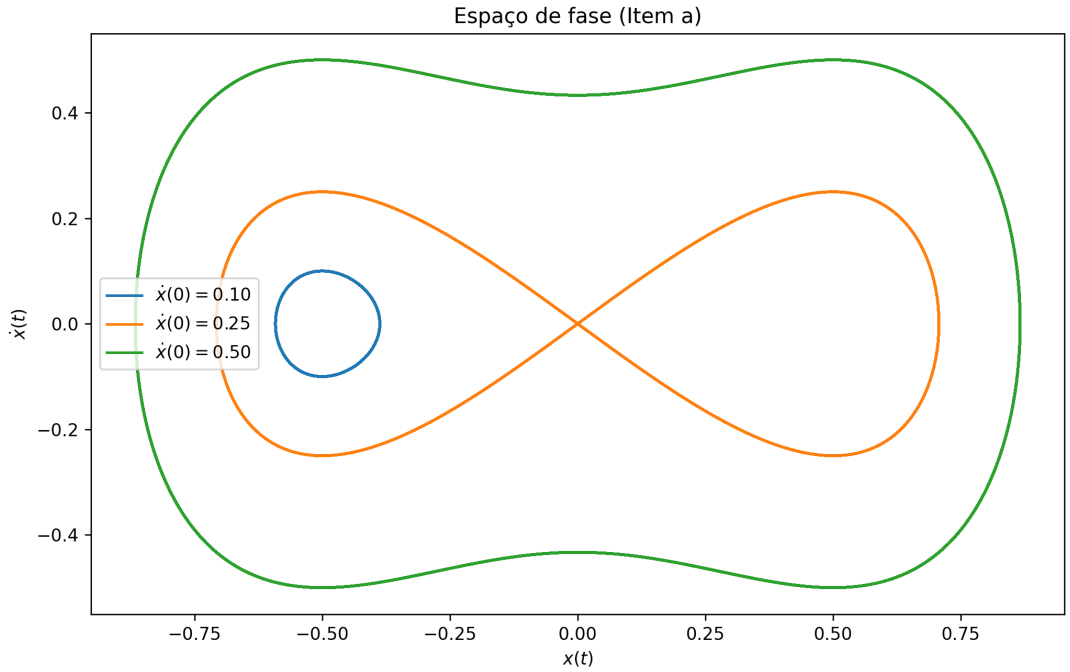

Figura 1: Diagrama do espaço de fase do oscilador de Duffing para $\dot{x}(0) = 0.10$ (azul), 0.25 (laranja), 0.50 (verde).

Item b)

Agora, incluindo o amortecimento, obtemos a Figura 2.

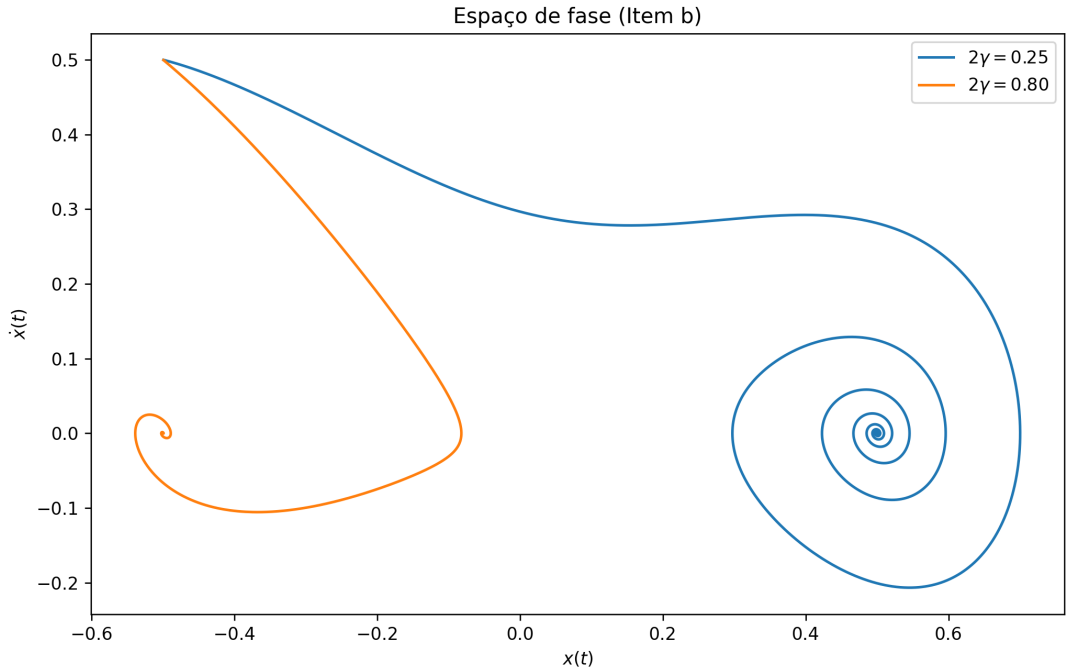

Figura 2: Diagrama do espaço de fase do oscilador de Duffing amortecido para $2\gamma = 0.25$ (azul), 0.80 (laranja).

Item c)

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

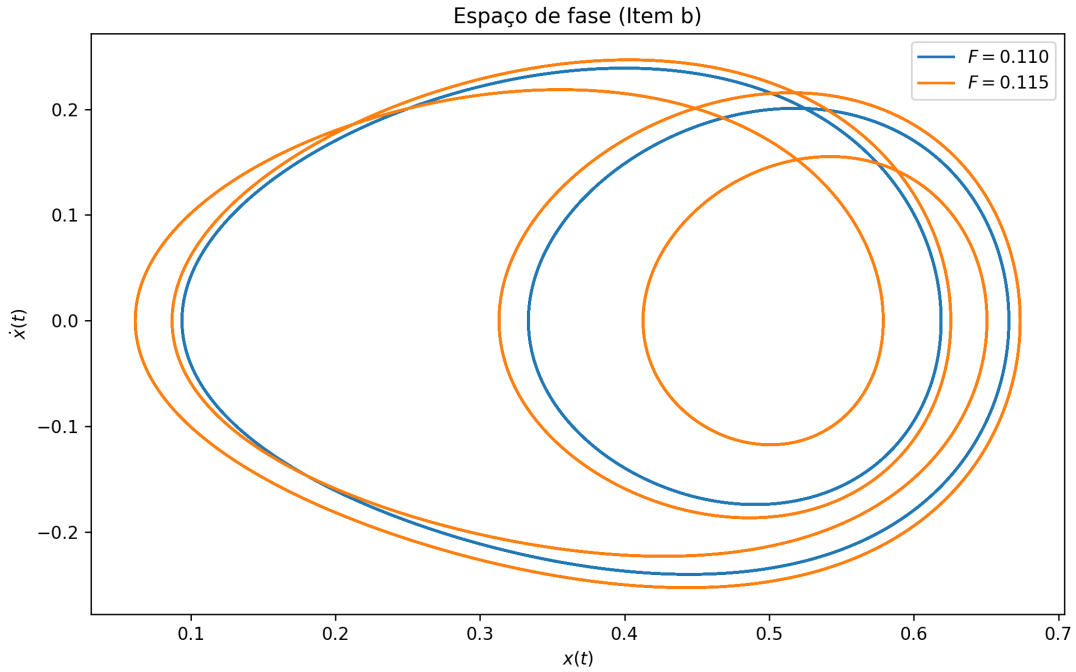

Figura 3: Diagrama do espaço de fase do oscilador de Duffing amortecido forçado para $F = 0.110$ (azul), 0.115 (laranja).

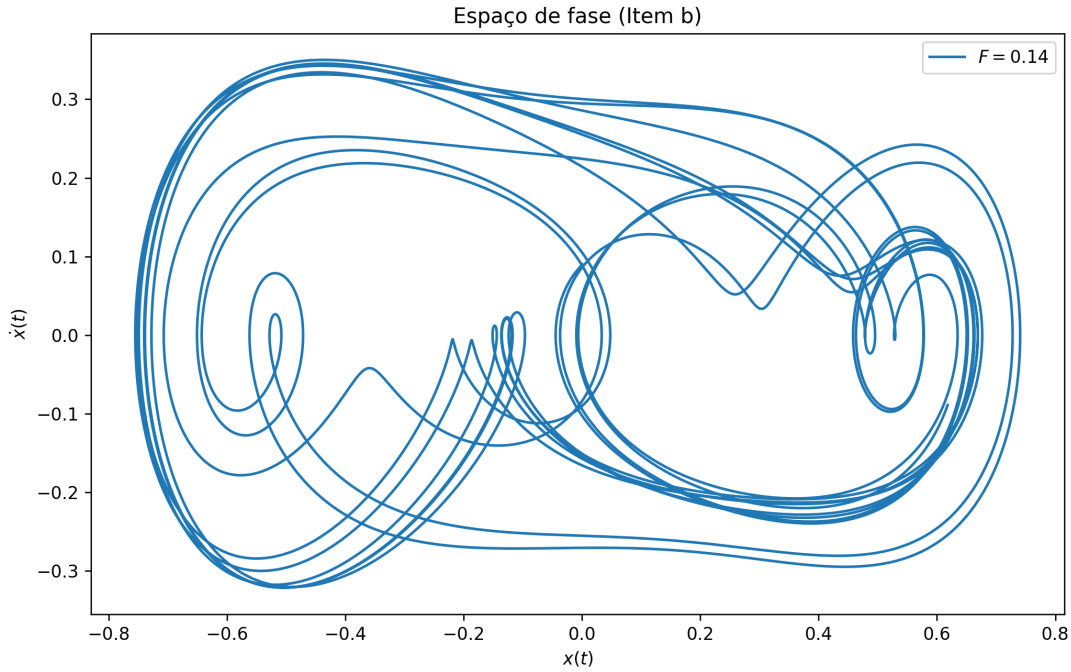

Figura 4: Diagrama do espaço de fase do oscilador de Duffing amortecido forçado para $F = 0.140$.

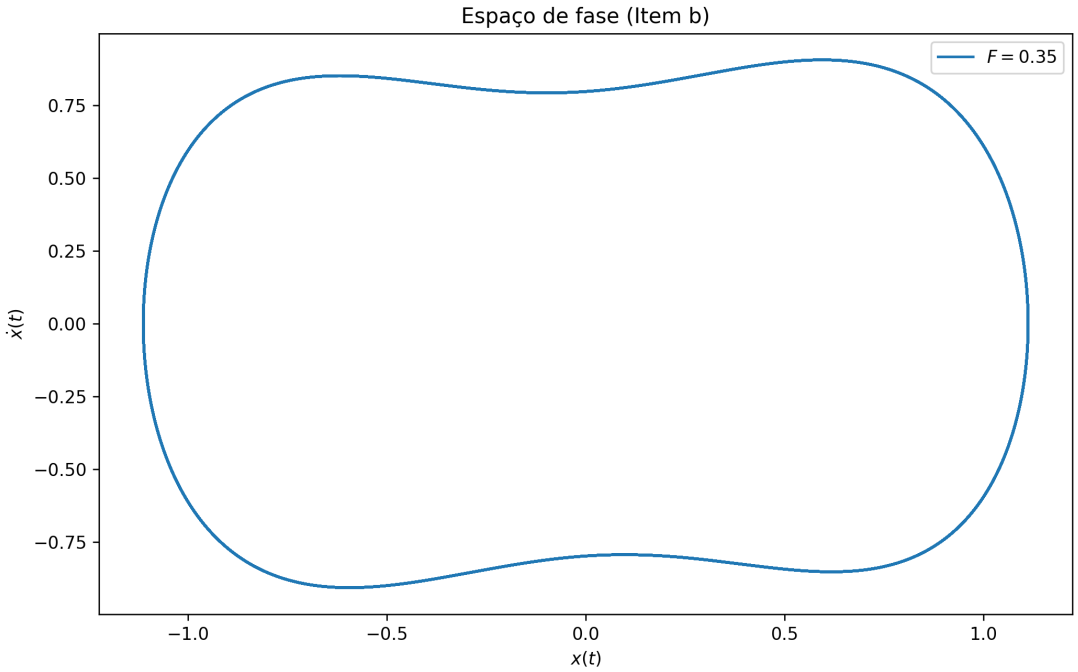

Figura 5: Diagrama do espaço de fase do oscilador de Duffing amortecido forçado para $F = 0.35$.

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

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

// Define constants
const double d = 0.50; // Delta
const double w = 1.00; // Omega
const double g = 0.25; // Gamma
const double F_start = 0.0;
const double F_end = 0.35;
const double F_step = 0.00025;
const int transient_steps = 200000;

double forced(double t, double x, double v, double F) {
    double result;

    result = - g * v + d * x * (1 - 4 * x * x) + F * cos(w * t);
    return result;
}

void rk4_forced(double& t, double& y, double& z, double h, double F) {
    double k1y, k1z, k2y, k2z, k3y, k3z, k4y, k4z;

    k1y = h * z;
    k1z = h * forced(t, y, z, F);
    k2y = h * (z + k1z / 2);
    k2z = h * forced(t + h / 2, y + k1y / 2, z + k1z / 2, F);
    k3y = h * (z + k2z / 2);
    k3z = h * forced(t + h / 2, y + k2y / 2, z + k2z / 2, F);
    k4y = h * (z + k3z);
    k4z = h * forced(t + h, y + k3y, z + k3z, F);
    y += (k1y + 2 * k2y + 2 * k3y + k4y) / 6;
    z += (k1z + 2 * k2z + 2 * k3z + k4z) / 6;
    t += h;
}

int main() {
    std::ofstream bifurcation_file("bifurcation_data.txt");
    std::cout << "Starting main function.\n" << std::endl;
    for (double F = F_start; F <= F_end; F += F_step) {
        double t = 0;
        double x = -0.5, v = 0.5; // Initial conditions

        double h = 0.01 * (2 * M_PI / w); // Initial h
        for (int i = 0; i < transient_steps; ++i) {
            rk4_forced(t, x, v, h, F);
        }

        h = 0.001 * (2 * M_PI / w); // Update h
        for (int i = 0; i < 100; ++i) {
                    // Advance one period
                    for (int j = 0; j < 1000; ++j) {
                        rk4_forced(t, x, v, h, F);
                    }
                    // Save the point on the Poincaré section
                    bifurcation_file << F << " " << x << "\n";
            }
```

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

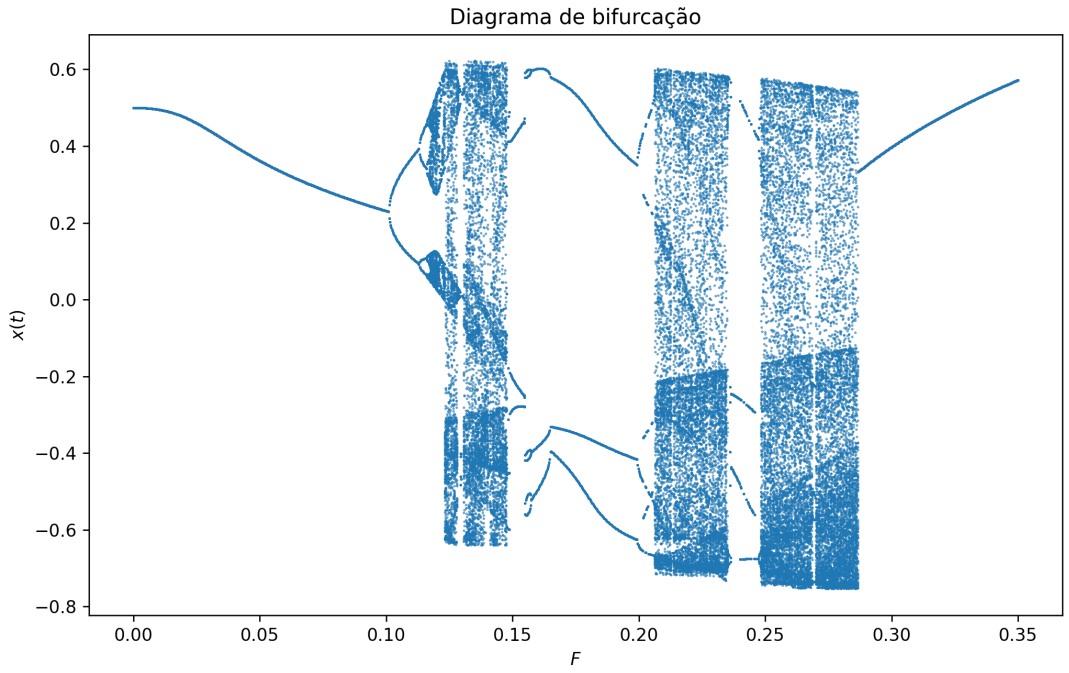

Figura 6: Diagrama de bifurcação para o oscilador de Duffing amortecido forçado.

A constante de Feigenbaum δ é definida como

$$\delta = \frac{\Delta F_{n-1}}{\Delta F_n}, \tag{1}$$

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 $F_2 \approx 0.225$, 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

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

// Define constants
const double d = 0.50; // Delta
const double w = 1.00; // Omega
const double g = 0.25; // Gamma
const double F= 0.26;

double forced(double t, double x, double v, double F) {
    double result;

    result = - g * v + d * x * (1 - 4 * x * x) + F * cos(w * t);
    return result;
}

void rk4_forced(double& t, double& y, double& z, double h, double F) {
    double k1y, k1z, k2y, k2z, k3y, k3z, k4y, k4z;

    k1y = h * z;
    k1z = h * forced(t, y, z, F);
    k2y = h * (z + k1z / 2);
    k2z = h * forced(t + h / 2, y + k1y / 2, z + k1z / 2, F);
    k3y = h * (z + k2z / 2);
    k3z = h * forced(t + h / 2, y + k2y / 2, z + k2z / 2, F);
    k4y = h * (z + k3z);
    k4z = h * forced(t + h, y + k3y, z + k3z, F);
    y += (k1y + 2 * k2y + 2 * k3y + k4y) / 6;
    z += (k1z + 2 * k2z + 2 * k3z + k4z) / 6;
    t += h;
}

int main() {
    std::ofstream vXx_file("vXx.txt");
    std::cout << "Starting main function.\n" << std::endl;
    double t = 0;
    double x = -0.5, v = 0.5; // Initial conditions
    double h = 0.01 * (2 * M_PI / w); // Initial h
    for (int i = 0; i < 200000; ++i) {
        rk4_forced(t, x, v, h, F);
    }

    h = 0.001 * (2 * M_PI / w); // Update h
    for (int i = 0; i < 20000; ++i) {
            // Advance one period
                for (int j = 0; j < 1000; ++j) {
                    rk4_forced(t, x, v, h, F);
                }
                // Save the point on the Poincaré section
                vXx_file << v << " " << x << "\n";
        }
    std::cout << "\nFinished!" << std::endl;
        vXx_file.close();
        std::cout << "Bifurcation data saved to vXx.txt\n";
        return 0;
}
```

Podemos ver o Mapa de Poincaré gerado na Figura 7.

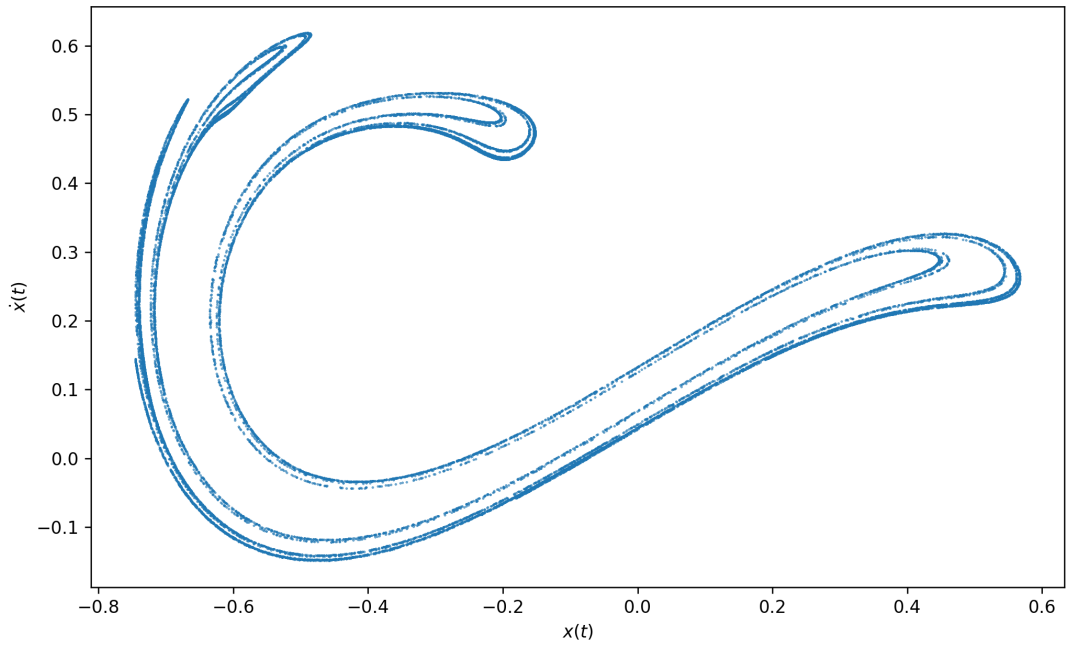

Figura 7: Mapa de Poincaré para o oscilador de Duffing amortecido forçado com $F = 0.26$.