ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

“O circuito de Chua - Segundo Exercício Programa”

Turma 01

MAP3121 – Métodos Numéricos e Aplicações para Engenharia

Larissa Kimie Takayama - 8943365

Thomas Palmeira Ferraz - 9348985

SÃO PAULO, JUNHO DE 2017

Sumário

1. Introdução

Para esse segundo exercício programa de Métodos Numéricos será analisado o circuito de Chua que nada mais é que um circuito elétrico simples formado por 2 capacitores lineares, um resistor linear, um indutor linear e um resistor não linear controlado pela tensão. Segue abaixo a representação desse circuito.

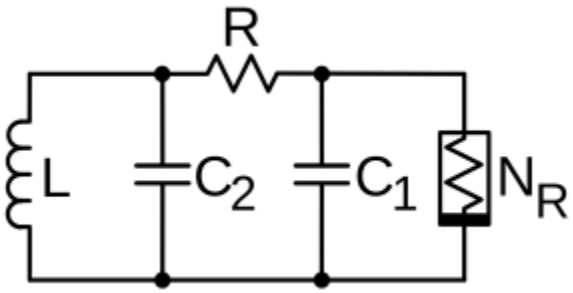

Figura 1 - Circuito de Chua.

Esse resistor não linear é conhecido como diodo de Chua e pode ser definido de forma linear por pedaços. Assim, dependendo da tensão ele fornece uma resistência diferente. A corrente dele pode ser definida como segue na figura abaixo.

$$g(V) = \begin{cases} G_b V + (G_b - G_a)E, & \text{se} \quad V \leq -E \\ G_a V, & \text{se} \quad -E < V < E \\ G_b V + (G_a - G_b)E, & \text{se} \quad V \geq E \end{cases}$$

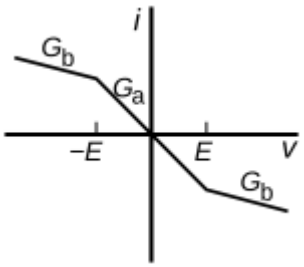

Figura 2 - Corrente do diodo de Chua.

Além disso é possível descrever o circuito com as seguintes equações diferenciais, que são deduzidas a partir das tensões de capacitores e corrente no indutor:

$$
\begin{aligned}
\dot{V}_{C_1} &= \frac{1}{RC_1}(V_{C_2} - V_{C_1}) - \frac{1}{C_1} g(V_{C_1}), \\
\dot{V}_{C_2} &= \frac{1}{RC_2}(V_{C_1} - V_{C_2}) + \frac{1}{C_2} I_L, \\
\dot{I}_L &= -\frac{1}{L} V_{C_2}.
\end{aligned}
$$

Figura 3 - Equações diferenciais do circuito de Chua.

O intuito deste exercício programa é analisar o circuito de Chua que representa um sistema dinâmico caótico, uma vez que para pequenas variações nos parâmetros e condições iniciais podem resultar em soluções bem diferentes.

Para se analisar este circuito antes foi preciso entender e aplicar o método de Runge-Kutta Fehlberg, que resolve sistemas de equações diferenciais ordinárias de primeira ordem. Construído o algoritmo para tal método, foi possível testá-lo com 3 situações propostas no enunciado do exercício programa e, por fim, resolver e analisar o circuito de Chua.

2. Resolução do problema

2.1. Justificativa do uso do C++ como linguagem

A escolha do uso do C++ se deve ao fato de ser uma linguagem muito mais simples em alguns aspectos utilizados neste trabalho. Destaca-se a maior facilidade com alocação dinâmica de espaços de memória e de operações de leitura e escrita de arquivos no formato .txt. O uso de programação defensiva também é facilitado com a existência das classes de exceção nesta linguagem.

Além disso, é uma linguagem que possui muitas bibliotecas prontas, com algoritmos e estruturas de dados já implementados. O principal motivo que levou o grupo a optar por ela foi a possibilidade de usar a classe vector da STL do C++ que permite ter vetores ou matrizes com tamanhos variáveis ao longo da execução. No caso, isso se mostrava útil pois o tamanho do vetor dependia da convergência do algoritmo. A única garantia que se tinha era quanto o h_min, que se utilizado poderia ser responsável pela alocação de um vetor onde se usa apenas 10% de seu espaço, o que seria muito ineficiente.

2.2. Algoritmo das funções F(t, x(t)) de cada teste

Para cada teste feito (os três propostos no enunciado e o do circuito de Chua) foi feito uma função diferente que calcula o F(t, x(t)) correspondente ao problema dado.

Essas funções são passadas como parâmetro na função que realiza o algoritmo do método RKF45, tornando o código mais modularizado. Assim, caso o usuário queira acrescentar algum outro tipo de teste basta escrever uma nova função que calcule a F(t, x(t)) e passá-la como parâmetro ao chamar a função do método RKF45.

É válido notar que todas as funções tem mesmos parâmetros, uma vez que elas precisam se encaixar no formato de função que é passada futuramente para a função de método RKF45. Além disso, todas a funções retornam a resposta em um vetor que é o mesmo vetor passado como parâmetro para se colocar a resposta. As funções não

precisariam retornar o vetor, uma vez que já escrevem o resultado no parâmetro fornecido, porém para facilitar o cascateamento de funções no método RKF45 achou-se interessante que ele retornasse esse mesmo valor.

2.2.1. Algoritmo do F(t, x(t)) do teste 1

Segue abaixo a equação diferencial dada para o teste 1.

$$F : R^2 \rightarrow R \text{ dada por } F(t,x) = 1 + (x-t)^2$$

Essa função é unidimensional, uma vez que o x é um valor em R, o que a torna uma função de fácil implementação.

Para implementá-la foi criada uma função chamada funcao_teste1, que segue abaixo.

```
double * funcao_teste1 (double t, double *x, double *resposta, int s){
   //int s = x->size();
   for (int i = 0; i < s; i++){
      resposta[i] = (1+pow((x[i] - t), 2));
   }
   return resposta;
}
```

Essa função faz uso de um vetor ‘x’ e um valor de ‘t’ para calcular a ‘resposta’, que, assim como ‘x’, é um vetor de tamanho ‘s’. Nota-se que para esse teste s=1 e, portanto, o “for” só irá rodar uma única vez.

A estrutura da função deve ser igual, em questão de parâmetros, para todas as funções de outros testes, por isso ‘x’ e ‘resposta’ são vetores e também é necessário passar o tamanho deles.

2.2.2. Algoritmo do F(t, x(t)) do teste 2

Para o teste 2 foi preciso implementar uma função multidimensional, uma vez que se tratava de uma equação em que $x \in R^4$. A equação proposta segue abaixo.

$F(t, X) = AX$, sendo $A = \begin{pmatrix} -2 & -1 & -1 & -2 \\ 1 & -2 & 2 & -1 \\ -1 & -2 & -2 & -1 \\ 2 & -1 & 1 & -2 \end{pmatrix}$

Para implementá-la foi criada uma função chamada funcao_teste2, que segue abaixo.

```
double* funcao_teste2 (double t, double*x, double* F, int m) {
double A[4][4];
double soma;
int i, j;

//Definindo A
A[0][0] = -2.0;
A[0][1] = -1.0;
A[0][2] = -1.0;
A[0][3] = -2.0;
A[1][0] = 1.0;
A[1][1] = -2.0;
A[1][2] = 2.0;
A[1][3] = -1.0;
A[2][0] = -1.0;
A[2][1] = -2.0;
A[2][2] = -2.0;
A[2][3] = -1.0;
A[3][0] = 2.0;
A[3][1] = -1.0;
A[3][2] = 1.0;
A[3][3] = -2.0;

//Multiplicando AX
```

```
    for (i=0; i<4; i++){
        soma = 0.0;
        for (j=0; j<4; j++){
            soma = soma + (A[i][j] * x[j]);
        }
        F[i] = soma;
    }
    return F;
}
```

Assim como na função para o teste 1, a função para o teste 2 tem como parâmetros ‘t’, o vetor ‘x’, o vetor que apresenta a resposta da função (‘F’) e o tamanho dos vetores ‘x’ e ‘F’, que neste teste é igual a 4.

Pode-se notar que, por ser uma função autônoma, ela não faz uso do ‘t’ para calcular a resposta. Mais uma vez é válido ressaltar que tal valor é passado como parâmetro apenas pelo fato de generalização e modularização desenvolvido para este programa.

2.2.3. Algoritmo do F(t, x(t)) do teste 3

Para o teste 3 foi preciso implementar uma função multidimensional, assim como no teste 2, porém agora ela deve variar o tamanho do vetor x, ou seja, $x \in R^m$. Segue abaixo a equação proposta, em que a matriz A agora é tridiagonal m x m.

$$F(t, X) = AX \text{ , } A_{i,i} = -2, i = 1, .., m, \; A_{i,i+1} = A_{i+1,i} = 1, i = 1, .., m-1 \text{ e } A_{i,j} = 0$$

Para implementar essa equação, criou-se uma função chamada funcao_teste3, que segue abaixo.

```
double* funcao_teste3(double t,double* x, double* F, int m){
    int i, j;
    double** A = new double*[m];
```

```
    for (i = 0; i< m; i++){
        A[i] = new double [m];
    }
    double soma;

    //Contruindo A
    for (i=0; i<m; i++){
        for(j=0; j<m; j++){
            if(i==j){
                A[i][j] = -2.0;
            }
            else {
                if( j==(i+1) | i==(j+1) ){
                    A[i][j] = 1.0;
                }
                else {
                    A[i][j] = 0.0;
                }
            }
        }
    }

    //Multiplicando AX
    for (i=0; i< m; i++){
        soma = 0.0;
        for (j=0; j<m; j++){
            soma = soma + (A[i][j] * x[j]);
        }
        F[i] = soma;
     }
     for (i = 0; i< m; i++){
        delete(A[i]);
     }
     delete (A);
     return F;
}
```

Essa função permite que o usuário escolha o tamanho ‘m’ do vetor ‘x’ e, consequentemente, a quantidade de dimensões do problema. Novamente, nota-se que esta é uma função autônoma, ou seja, não faz uso de ‘t’ para o cálculo da resposta ‘F’.

2.2.4. Algoritmo do F(t, x(t)) do circuito de Chua

Para resolver o circuito de Chua foi preciso resolver um sistema de equações diferenciais que segue abaixo.

$$\begin{aligned}
\dot{V}_{C_1} &= \frac{1}{RC_1}(V_{C_2} - V_{C_1}) - \frac{1}{C_1}g(V_{C_1}), \\
\dot{V}_{C_2} &= \frac{1}{RC_2}(V_{C_1} - V_{C_2}) + \frac{1}{C_2}I_L, \\
\dot{I}_L &= -\frac{1}{L}V_{C_2}.
\end{aligned}$$

Assim, foi preciso criar uma função com dimensão 3 (uma vez que o sistema consiste em três equações diferenciais de primeira ordem), que a cada elemento do vetor resposta ‘F’ fosse realizado uma das funções do sistema. Segue abaixo a função feita para o circuito de Chua.

```
double* funcao_circuito (double t,double* x, double* F, int m){
// Vc1, Vc2, IL
   F[0] = (1.0/(R*C1))*(x[1] - x[0]) + (-1.0/C1)*funcao_g(x[0]);
   F[1] = (1.0/(R*C2))*(x[0] - x[1]) + (1.0/C2)*x[2];
   F[2] = (-1.0/L)*x[1];
   return F;
}
```

Para o cálculo da resposta é preciso um vetor ‘x’, que neste caso são os parâmetros V_{c1}, V_{c2} e I_L, nesta ordem. No primeiro elemento de ‘F’ (F[0]) guarda-se a resposta calculada a partir da primeira equação do sistema dado. O segundo e o terceiro elemento são os correspondentes à segunda e terceira equação do sistema dado, respectivamente.

É possível notar que para calcular o primeiro elemento de ‘F’ faz-se uso de uma função_g, que foi implementada a parte de forma a facilitar o entendimento do código. Segue abaixo a função_g desenvolvida conforme o enunciado.

```
double funcao_g (double v){
   if (v <= -1*e_max) return (G_c*v + e_max*(G_c - G_b) + E*(G_b - G_a));
   else if (v > -1*e_max and v <= -1*E) return (G_b*v + (G_b - G_a)*E);
   else if (v > -1*E and v < E) return (G_a*v);
   else if (v >= E and v < e_max) return (G_b*v + (G_a - G_b)*E);
   else return (G_c*v + e_max*(G_b - G_c) + E*(G_a - G_c));
}
```

Basicamente essa função implementa a função descrita abaixo.

$$g(V) = \begin{cases} G_c V + E_{\max}(G_c - G_b) + E(G_b - G_a) & \text{se} \quad V \leq -E_{\max} \\ G_b V + (G_b - G_a)E, & \text{se} \quad -E_{\max} < V \leq -E \\ G_a V, & \text{se} \quad -E < V < E \\ G_b V + (G_a - G_b)E, & \text{se} \quad E \leq V < E_{\max} \\ G_c V + E_{\max}(G_b - G_c) + E(G_a - G_b) & \text{se} \quad E_{\max} \geq V \end{cases}$$

2.3. Algoritmo do Método de Runge-Kutta Fehlberg

O método utilizado neste exercício programa para resolução de equações diferenciais ordinárias foi o método RKF45. Ele combina métodos de quarta e quinta ordem para o controle do passo (h).

Esse método de Runge-Kutta Fehlberg faz uso de 6 estágios que são calculados como se segue abaixo.

$$
\begin{aligned}
k_1 &= hF(t_i, x_i) \\
k_2 &= hF(t_i + h/4, x_i + \frac{1}{4}\, k_1) \\
k_3 &= hF(t_i + 3h/8, x_i + \frac{3}{32}\, k_1 + \frac{9}{32}\, k_2) \\
k_4 &= hF(t_i + 12h/13, x_i + \frac{1932}{2197}\, k_1 - \frac{7200}{2197}\, k_2 + \frac{7296}{2197}\, k_3) \\
k_5 &= hF(t_i + h, x_i + \frac{439}{216}\, k_1 - 8\, k_2 + \frac{3680}{513}\, k_3 - \frac{845}{4104}\, k_4) \\
k_6 &= hF(t_i + h/2, x_i - \frac{8}{27}\, k_1 + 2\, k_2 - \frac{3544}{2565}\, k_3 + \frac{1859}{4104}\, k_4 - \frac{11}{40}\, k_5)
\end{aligned}
$$

A partir dos k_i obtidos é possível calcular o X_{i+1} de quarta ordem e de quinta ordem, assim como mostrado nas equações abaixo.

$$x_{i+1} = x_i + \frac{25}{216}\, k_1 + \frac{1408}{2565}\, k_3 + \frac{2197}{4104}\, k_4 - \frac{1}{5}\, k_5$$

$$\tilde{x}_{i+1} = x_i + \frac{16}{135}\, k_1 + \frac{6656}{12825}\, k_3 + \frac{28561}{56430}\, k_4 - \frac{9}{50}\, k_5 + \frac{2}{55}\, k_6$$

O algoritmo deve funcionar da seguinte forma: a partir do valor inicial de x e de um h inicial deve-se calcular os k_i e as aproximações x_{i+1} e $\tilde{x}_{i+1}$. Vale notar que é preciso que os k_i sejam calculados sequencialmente, uma vez que o valor do k_i posterior depende do valor dos k_i anteriores. Além disso, apenas depois de calculados os k_i é que deverá ser calculado as aproximações de quarta e quinta ordem, pois elas dependem diretamente desses k_i.

Depois de calculadas as aproximações, é possível estimar o erro local de truncamento $\tau_{i+1}(h)$ que é dado pela equação abaixo.

$$\tau_{i+1}(h) \approx \frac{\tilde{x}_{i+1} - x_{i+1}}{h}$$

Vale notar que quando x é multidimensional é preciso pegar o valor máximo, em módulo, da diferença das duas aproximações para que tenhamos o valor máximo de erro local estimado.

Com o valor do erro local estimado, verifica-se se ele está dentro da precisão pedida (eps). No código feito pelo grupo, quando o erro está dentro da precisão necessária a flag ‘preciso’ torna-se verdadeira. Além disso o valor de x_{i+1} é aceito como nova aproximação no instante $t_{i+1} = t_i+h$ e deve-se atualizar o valor de h para αh, como mostrado na equação abaixo.

$$\alpha \leq \left(\frac{\epsilon h}{c||\tilde{x}_{i+1} - x_{i+1}||} \right)^{1/p}$$

Sendo o fator de segurança c = 2 e lembrando que p = 4 (ordem de convergência do método). Para este cálculo podemos usar o valor máximo da diferença das aproximações quando quisermos diminuir o h (ou seja, quando o erro local estimado for maior que a precisão).

Caso o valor do erro local estimado for maior que a precisão, o valor de x_{i+1} é rejeitado e o h deve ser atualizado assim como descrito anteriormente. Então deve-se refazer os cálculos de k_i e das aproximações x_{i+1} e $\tilde{x}_{i+1}$.

A integração terminará ao se atingir o instante final desejado (t_f). A fim de garantir que se chegue exatamente no instante final, sem ultrapassá-lo, em toda iteração verifica-se se h é maior que t_f - t_atual, caso positivo então h deve ser t_f - t_atual.

Além disso, limitou-se o h entre h_min e h_max de forma a este não se tornar um valor pequeno demais e nem grande demais. Tais valores podem ser modificados pelo usuário.

Segue abaixo o código feito para implementar o método KRF45 descrito acima.

```
void RKF45 (double h, double ti, double tf, double eps, double *x0, vector<double *> *x, vector<double> *t, vector<double *> *x_barra, double* funcao (double t, double *x, double *resposta, int s), unsigned int m){
    double t_atual = ti;
    double tau;
    //cout << "m: " << m <<endl;
    double *k1 = new double [m];
    double *k2 = new double [m];
    double *k3 = new double [m];
```

```
    double *k4 = new double [m];
    double *k5 = new double [m];
    double *k6 = new double [m];
    double *temporario = new double [m];
    double *temporario2 = new double [m];
    double *temporario3 = new double [m];
    double *temporario4 = new double [m];
    double *temporario5 = new double [m];
    double *temporario6 = new double [m];
    double *temporario7 = new double [m];

    double *x_atual = new double [m];
    double *x_barra_atual = new double [m];
    bool preciso = false;
    x->push_back(x0);
    x_barra->push_back(x0);
    t->push_back (ti);
    ofstream saida;
    saida.open ("Ex4.txt");
    //saida << "t x[0] x[1] x[2] x[3] err0[0] erro[1] erro[2] erro[3]" << endl;
    saida.precision(16);
          for(int i=0; t_atual < tf; i++){
       while (!preciso){
          k1 = multiplica_vetor_por_escalar(funcao(t_atual, x->at(i), temporario, m),h, k1, m);
          k2 = multiplica_vetor_por_escalar(funcao(t_atual + h/4.0, soma_vetor(x->at(i),
multiplica_vetor_por_escalar(k1, 1.0/4.0, temporario,m), temporario2, m),temporario3,m),h,k2,m);
          k3 = multiplica_vetor_por_escalar(funcao(t_atual + 3.0*(h/8.0), soma_vetor(x->at(i),
multiplica_vetor_por_escalar(k1, (3.0/32.0), temporario,m), multiplica_vetor_por_escalar(k2, (9.0/32.0),
temporario2,m), temporario3,m), temporario4,m),h,k3,m);
          k4 = multiplica_vetor_por_escalar(funcao(t_atual + (12.0/13.0)*h, soma_vetor(x->at(i),
multiplica_vetor_por_escalar(k1, (1932.0/2197.0), temporario,m), multiplica_vetor_por_escalar (k2, -
7200.0/2197.0, temporario2,m), multiplica_vetor_por_escalar (k3, (7296.0/2197.0), temporario3,m),
temporario4,m), temporario5,m),h, k4,m);
          k5 = multiplica_vetor_por_escalar(funcao(t_atual + h, soma_vetor(x-
>at(i),multiplica_vetor_por_escalar(k1, (439.0/216.0), temporario,m), multiplica_vetor_por_escalar(k2, -
8.0, temporario2,m), multiplica_vetor_por_escalar(k3, (3680.0/513.0), temporario3,m),
multiplica_vetor_por_escalar(k4, -845.0/4104.0, temporario4,m), temporario5,m), temporario6,m),h,k5,m);
```

```
k6 = multiplica_vetor_por_escalar(funcao(t_atual + h/2.0, soma_vetor(x->at(i),
multiplica_vetor_por_escalar(k1, -8.0/27.0, temporario,m), multiplica_vetor_por_escalar(k2, 2.0,
temporario2,m), multiplica_vetor_por_escalar(k3, - 3544.0/2565.0, temporario3,m),
multiplica_vetor_por_escalar(k4, (1859.0/4104.0), temporario4,m),multiplica_vetor_por_escalar(k5, -
11.0/40.0, temporario5,m), temporario6,m), temporario7,m), h,k6,m);
x_barra_atual = soma_vetor(x->at(i), multiplica_vetor_por_escalar(k1, 16.0/135.0, temporario,m),
multiplica_vetor_por_escalar(k3, 6656.0/12825.0, temporario3,m), multiplica_vetor_por_escalar(k4,
28561.0/56430.0, temporario4,m), multiplica_vetor_por_escalar(k5, -9.0/50.0,
temporario5,m),multiplica_vetor_por_escalar(k6, 2.0/55.0, temporario2,m),x_barra_atual,m);
x_atual = soma_vetor(x->at(i),multiplica_vetor_por_escalar(k1, 25.0/216.0,
temporario,m),multiplica_vetor_por_escalar(k3, 1408.0/2565.0, temporario3,m),
multiplica_vetor_por_escalar(k4, 2197.0/4104.0, temporario4,m), multiplica_vetor_por_escalar(k5, -
1.0/5.0, temporario5,m),x_atual,m);
tau = maxima_diferenca(x_barra_atual,x_atual,m)/h;
if (tau < eps){
    preciso = true;
    t_atual += h;
    //calcular novo h
    // verificar se está entre o minimo e o máximo
    //verificar se o passo não atravessa o tf.
}
/*cout << "k1: " << k1[0] << " " << k1[1] << " " << k1[2] << " " << k1[3] << endl;
cout << "k2: " << k2[0] << " " << k2[1] << " " << k2[2] << " " << k2[3] << endl;
cout << "k3: " << k3[0] << " " << k3[1] << " " << k3[2] << " " << k3[3] << endl;
cout << "k4: " << k4[0] << " " << k4[1] << " " << k4[2] << " " << k4[3] << endl;
cout << "k5: " << k5[0] << " " << k5[1] << " " << k5[2] << " " << k5[3] << endl;
cout << "k6: " << k6[0] << " " << k6[1] << " " << k6[2] << " " << k6[3] << endl;*/
cout << "tau: " << tau << endl;
h = h*pow (((h*eps)/(c*maxima_diferenca(x_barra_atual, x_atual, m))), 1.0/p);
if (h > h_max) {
    h = h_max;
}
if (h < h_min) {
    h = h_min;
}
if (h > tf - t_atual) {
    h = tf - t_atual;
```

```
        }

        //cout << "ks: " << k1[0] << " " << k2[0] << " " << k3[0] << " " << k4[0] <<" " << k5[0] <<" " << k6[0]
<< endl;
        cout << "h: " << h << endl;
        //cout << "i: " << i << " k1: " << k1 << " k2: " << k2 << " k3: " << k3 << " k4: " << k4 << " k5: " << k5
<< " k6: " << k6 << " x[i+1]: " << x_atual << "x~[i+1]: " << x_barra_atual << "tau: " << tau << "t: " << t_atual
<< "h: " << h << endl;
        //printa_vetor("x: ", x_atual, m);
        //printa_vetor (" x_barra: ", x_barra_atual, m);
        //cout << "tau " << tau << endl;
      }
      preciso = false;
      guarda_vetor (x, x_atual, m);
      t->push_back (t_atual);
      /*temporario = erro_teste2(x_atual, t_atual, temporario);
      saida << t_atual << " " << x_atual[0] << " " << x_atual[1] << " " << x_atual[2] << " " << x_atual[3] << "
" << temporario[0] << " " << temporario[1] << " " << temporario[2] << " " << temporario[3]  << endl;*/
      //saida << t_atual << " " << x_atual[0] << " " << erro_teste1(x_atual[0], t_atual) << endl;
      /*temporario = erro_teste3(x_atual, t_atual, temporario, m);
      saida << t_atual << " " << x_atual[0] << " " << x_atual[1] << " " << x_atual[2] << " " << x_atual[3] << "
"  << x_atual[4] << " "<<  x_atual[5] << " " << x_atual[6] << " " << temporario[0] << " " << temporario[1] <<
" " << temporario[2] << " " << temporario[3]  << temporario[4] << " " << temporario[5] << " " <<
temporario[6] << " " << endl;*/
      saida << t_atual << " " << x_atual[0] << " " << x_atual[1] << " " << x_atual[2] << "," << endl;
      guarda_vetor (x_barra, x_barra_atual, m);
      cout << "i =" << i << endl;
        }
        saida.close();
        delete (k1);
        delete (k2);
        delete (k3);
        delete (k4);
        delete (k5);
        delete (k6);
        delete (temporario);
        delete (temporario2);
```

```
        delete (temporario3);
        delete (temporario4);
        delete (temporario5);
        delete (x_atual);
        delete (x_barra_atual);
}
```

Algumas linhas deste código estão comentadas, pois dependem do que o usuário gostaria de ver como saída. Por exemplo: se o usuário deseja saber o erro em relação à solução exata é preciso descomentar algumas linhas que chamam a função erro do determinado teste, além das linhas que as printam em uma saída.

Nota-se que foi preciso fazer uso de algumas funções auxiliares para fazer as operações com vetores. Dessa forma segue a explicação das funções feitas e utilizadas.

Para o cálculo dos k_i foi preciso fazer a função que soma vetores e que multiplica um vetor por escalar. A função que multiplica um vetor por escalar segue abaixo.

```
double * multiplica_vetor_por_escalar (double *v, double escalar, double *resposta, int s){
   //int s = v->size();
   cout << "s: " << s;
   for (int i = 0; i < s; i++){
      resposta[i] = escalar*v[i];
      cout << "resposta->at(i): " << resposta[i];
   }
   return resposta;
}
```

A função que soma vetores na verdade são 6 funções, sendo a única diferença entre elas o fato de que cada uma tem mais entradas que outra, consequentemente soma mais vetores que outra. Assim, tem-se uma função que soma 2 vetores, uma que soma 3 vetores e assim por diante. A declaração dessas funções apenas se diferencia pela quantidade de entradas, uma vez que o nome permanece o mesmo. Segue abaixo as funções de soma de vetores.

```
double * soma_vetor (double *v1, double *v2, double *resposta, int s){
```

```
    //int s = v1->size();
    cout << "soma_vetor" << endl;
    for (int i = 0; i < s; i++){
        resposta[i] = v1[i] + v2[i];
    }
    return resposta;
}

double * soma_vetor (double *v1, double *v2, double *v3, double *resposta, int s){
    //int s = v1->size();
    for (int i = 0; i < s; i++){
        resposta[i] = v1[i] + v2[i] + v3[i];
    }
    return resposta;
}

double * soma_vetor (double *v1, double *v2, double *v3, double *v4, double *resposta, int s){
    //int s = v1->size();
    for (int i = 0; i < s; i++){
        resposta[i] = v1[i]+v2[i]+v3[i]+v4[i];
    }
    return resposta;
}

double * soma_vetor (double *v1, double *v2, double *v3, double *v4, double *v5, double *resposta, int s){
    //int s = v1->size();
    for (int i = 0; i < s; i++){
        resposta[i] = v1[i]+v2[i]+v3[i]+v4[i]+v5[i];
    }
    return resposta;
}

double * soma_vetor (double *v1, double *v2, double *v3, double *v4, double *v5, double *v6, double
*resposta, int s){
    //int s = v1->size();
    for (int i = 0; i < s; i++){
        resposta[i] = v1[i]+v2[i]+v3[i]+v4[i]+v5[i]+v6[i];
```

```
    }
    return resposta;
}
```

Para o cálculo de novo h foi preciso fazer uma função que retorna a máxima diferença entre dois vetores. Assim segue abaixo a função feita

```
double maxima_diferenca(double *x1, double *x2, int s){
    //int s = x1->size();
    double dif = 0.0;
    double dif_i;
    for (int i = 0; i < s; i++){
        dif_i = abs(x1[i] - x2[i]);
        if (dif_i > dif){
            dif = dif_i;
        }
    }
    return dif;
}
```

A última função auxiliar utilizada foi aquela que guarda o vetor calculado (e aceito, ou seja, dentro da precisão solicitada) colocando seus valores ao fim de um vetor já existente. Tal função segue abaixo.

```
void guarda_vetor (vector<double *> *x,  double *y, int s){
    double *z = new double [s];
    for(int i = 0; i< s; i++){
        z[i] = y[i];
    }
    x->push_back(z);
}
```

Essa função é utilizada para guardar os valores de x_{i+1} e $\tilde{x}_{i+1}$, depois que eles foram calculados e provados dentro da precisão solicitada.

2.4. Algoritmo do Erro em relação à solução exata

A fim de verificar o erro da integração, feita pelo método KRF45 implementado, em relação a solução exata, foi feito uma função, para cada teste, que calcula tal erro. Isso só foi possível, pois para os testes 1, 2 e 3 foi fornecido a solução exata das equações diferenciais.

Para o primeiro teste a solução exata fornecida foi x(t) = t + 1/(1 − t), logo a função erro para ele faz a diferença entre essa função e a aproximação calculada naquele ponto. Vale notar que o erro foi calculado em módulo. Segue abaixo a função desenvolvida.

```
double erro_teste1 (double x_estim, double t){
    return fabs(x_estim - (t+1.0/(1.0-t)));
}
```

Para o segundo teste a solução exata fornecida foi a que segue abaixo.

$$\bar{X}(t) = \begin{pmatrix} e^{-t}\sin t + e^{-3t}\cos 3t \\ e^{-t}\cos t + e^{-3t}\sin 3t \\ -e^{-t}\sin t + e^{-3t}\cos 3t \\ -e^{-t}\cos t + e^{-3t}\sin 3t \end{pmatrix}$$

Assim a função erro desenvolvida para esse teste segue abaixo.

```
double* erro_teste2 (double* x_estim, double t, double* erro){

    erro[0] = fabs(x_estim[0] - ( (exp(-1.0*t)*sin(t)) + (exp(-3.0*t)*cos(3*t)) ));
    erro[1] = fabs(x_estim[1] - ( (exp(-1.0*t)*cos(t)) + (exp(-3.0*t)*sin(3*t)) ));
    erro[2] = fabs(x_estim[2] - ( -1.0*(exp(-1.0*t)*sin(t)) + (exp(-3.0*t)*cos(3*t)) ));
    erro[3] = fabs(x_estim[3] - ( -1.0*(exp(-1.0*t)*cos(t)) + (exp(-3.0*t)*sin(3*t)) ));

    return erro;
}
```

Já para o terceiro teste a solução exata fornecida foi: $X(t)i = e^{-\lambda 1t} \sin(\pi\ yi) + e^{-\lambda 2t} \sin(m\ \pi\ yi)$, i = 1, ..., m, com $yi = i / (m + 1)$, $\lambda 1 = 2(1 - \cos(\pi / (m + 1)))$ e $\lambda 2 = 2(1 - \cos(m\ \pi / (m + 1)))$. Assim, a função erro implementada segue abaixo.

```
double* erro_teste3 (double* x_estim, double t, double* erro, int m){
    double lamb1, lamb2, y;
    int i;

    for (i=0; i<m; i++){
        y = i/(m+1.0);
        lamb1 = 2.0*(1.0-cos(M_PI/(m+1.0)));
        lamb2 = 2.0*(1.0-cos(m*M_PI/(m+1.0)));
        erro[i] = fabs(x_estim[i] - ( (exp(-1.0*lamb1*t)*sin(M_PI*y)) + (exp(-1.0*lamb2*t)*sin(m*M_PI*y)) ));
    }
    return erro;
}
```

Essas funções podem ser utilizadas dentro do método RKF45 de modo a verificar o erro da aproximação com a solução exata da equação, ponto a ponto. Assim, a cada ‘t’ chama-se a função erro do teste determinado de modo a gerar ao final resultados de erros ponto a ponto. Para entender melhor isso basta voltar ao código do método RKF45.

3. Análise de resultados

3.1. Teste 1

O primeiro teste consiste em resolver a equação diferencial $F(t, x) = 1 + (x - t)^2$, a partir do valor inicial $x(1.05) = -18.95$ até o tempo final $t_f = 3$ pelo método RKF45 com $eps = 10^{-5}$. Para iniciar utilizou-se $h = 0.1$ e a cada passo foi impresso o valor de h empregado e o valor da solução. Além disso também foi impresso o erro em relação a solução exata desta equação.

h: 0.00102247 x: -17.3432
h: 0.00160625 x: -18.5482 erro: 8.27827e-010
h: 0.0016614 x: -17.9484 erro: 7.62517e-009
h: 0.00174051 x: -17.3653 erro: 1.38945e-008
h: 0.00182386 x: -16.7913 erro: 2.00831e-008
h: 0.00191256 x: -16.2269 erro: 2.61874e-008
h: 0.00200702 x: -15.6719 erro: 3.22197e-008
h: 0.00210771 x: -15.1262 erro: 3.81926e-008
h: 0.00221514 x: -14.59 erro: 4.41184e-008
h: 0.00232988 x: -14.0632 erro: 5.00105e-008
h: 0.00245255 x: -13.5457 erro: 5.58822e-008
h: 0.00258385 x: -13.0376 erro: 6.17476e-008
h: 0.00272454 x: -12.5388 erro: 6.7621e-008
h: 0.00287545 x: -12.0493 erro: 7.35178e-008
h: 0.00303752 x: -11.569 erro: 7.94538e-008
h: 0.00321179 x: -11.0981 erro: 8.54454e-008
h: 0.00339941 x: -10.6363 erro: 9.15101e-008
h: 0.00360167 x: -10.1837 erro: 9.76663e-008
h: 0.00382 x: -9.74033 erro: 1.03933e-007
h: 0.00405601 x: -9.30606 erro: 1.10331e-007
h: 0.0043115 x: -8.88089 erro: 1.16882e-007
h: 0.00458848 x: -8.46478 erro: 1.2361e-007
h: 0.00488923 x: -8.05769 erro: 1.30538e-007
h: 0.00521631 x: -7.65958 erro: 1.37693e-007
h: 0.00557263 x: -7.27041 erro: 1.45105e-007
h: 0.00596146 x: -6.89013 erro: 1.52802e-007
h: 0.00638656 x: -6.51867 erro: 1.60819e-007
h: 0.00685216 x: -6.15599 erro: 1.69192e-007
h: 0.00736312 x: -5.80202 erro: 1.77957e-007
h: 0.00792502 x: -5.4567 erro: 1.87159e-007
h: 0.00854423 x: -5.11994 erro: 1.96843e-007
h: 0.00922813 x: -4.79167 erro: 2.07058e-007
h: 0.00998522 x: -4.47178 erro: 2.1786e-007
h: 0.01 x: -4.16019 erro: 2.29309e-007
h: 0.01 x: -3.87878 erro: 2.3155e-007
h: 0.01 x: -3.62359 erro: 2.28102e-007
h: 0.01 x: -3.39099 erro: 2.21317e-007
h: 0.01 x: -3.17801 erro: 2.12674e-007
h: 0.01 x: -2.98215 erro: 2.03107e-007
h: 0.01 x: -2.80135 erro: 1.93202e-007
h: 0.01 x: -2.63383 erro: 1.83326e-007
h: 0.01 x: -2.47811 erro: 1.73701e-007
h: 0.01 x: -2.33292 erro: 1.6446e-007
h: 0.01 x: -2.19714 erro: 1.55672e-007
h: 0.01 x: -2.06984 erro: 1.47372e-007
h: 0.01 x: -1.95017 erro: 1.39565e-007
h: 0.01 x: -1.83742 erro: 1.32244e-007
h: 0.01 x: -1.73096 erro: 1.25393e-007
h: 0.01 x: -1.63023 erro: 1.18988e-007
h: 0.01 x: -1.53472 erro: 1.13003e-007
h: 0.01 x: -1.444 erro: 1.07412e-007
h: 0.01 x: -1.35768 erro: 1.02189e-007
h: 0.01 x: -1.27541 erro: 9.73063e-008
h: 0.01 x: -1.19687 erro: 9.27406e-008
h: 0.01 x: -1.12179 erro: 8.84681e-008
h: 0.01 x: -1.04991 erro: 8.44669e-008
h: 0.01 x: -0.980989 erro: 8.07168e-008
h: 0.01 x: -0.914832 erro: 7.7199e-008
h: 0.01 x: -0.851244 erro: 7.38962e-008
h: 0.01 x: -0.790055 erro: 7.07922e-008

h: 0.01 x: -0.731105 erro: 6.78726e-008
h: 0.01 x: -0.674251 erro: 6.51237e-008
h: 0.01 x: -0.619362 erro: 6.25332e-008
h: 0.01 x: -0.566317 erro: 6.00897e-008
h: 0.01 x: -0.515004 erro: 5.77827e-008
h: 0.01 x: -0.46532 erro: 5.56027e-008
h: 0.01 x: -0.417171 erro: 5.35408e-008
h: 0.01 x: -0.37047 erro: 5.15889e-008
h: 0.01 x: -0.325136 erro: 4.97397e-008
h: 0.01 x: -0.281094 erro: 4.79862e-008
h: 0.01 x: -0.238275 erro: 4.6322e-008
h: 0.01 x: -0.196612 erro: 4.47415e-008
h: 0.01 x: -0.156048 erro: 4.32392e-008
h: 0.01 x: -0.116524 erro: 4.18101e-008
h: 0.01 x: -0.07799 erro: 4.04497e-008
h: 0.01 x: -0.0403958 erro: 3.91536e-008
h: 0.01 x: -0.00369608 erro: 3.79181e-008
h: 0.01 x: 0.032152 erro: 3.67394e-008
h: 0.01 x: 0.0671885 erro: 3.56142e-008
h: 0.01 x: 0.101451 erro: 3.45393e-008
h: 0.01 x: 0.134975 erro: 3.35119e-008
h: 0.01 x: 0.167794 erro: 3.25291e-008
h: 0.01 x: 0.199939 erro: 3.15885e-008
h: 0.01 x: 0.231438 erro: 3.06878e-008
h: 0.01 x: 0.262321 erro: 2.98247e-008
h: 0.01 x: 0.292614 erro: 2.89972e-008
h: 0.01 x: 0.32234 erro: 2.82034e-008
h: 0.01 x: 0.351524 erro: 2.74415e-008
h: 0.01 x: 0.380187 erro: 2.67098e-008
h: 0.01 x: 0.40835 erro: 2.60068e-008
h: 0.01 x: 0.436033 erro: 2.5331e-008
h: 0.01 x: 0.463256 erro: 2.46811e-008
h: 0.01 x: 0.490035 erro: 2.40556e-008
h: 0.01 x: 0.516388 erro: 2.34535e-008
h: 0.01 x: 0.54233 erro: 2.28736e-008
h: 0.01 x: 0.567877 erro: 2.23149e-008
h: 0.01 x: 0.593044 erro: 2.17762e-008
h: 0.01 x: 0.617844 erro: 2.12567e-008
h: 0.01 x: 0.64229 erro: 2.07555e-008
h: 0.01 x: 0.666395 erro: 2.02717e-008
h: 0.01 x: 0.690171 erro: 1.98046e-008
h: 0.01 x: 0.713629 erro: 1.93534e-008
h: 0.01 x: 0.736781 erro: 1.89173e-008
h: 0.01 x: 0.759635 erro: 1.84958e-008
h: 0.01 x: 0.782203 erro: 1.80881e-008
h: 0.01 x: 0.804494 erro: 1.76937e-008
h: 0.01 x: 0.826517 erro: 1.7312e-008
h: 0.01 x: 0.848281 erro: 1.69425e-008
h: 0.01 x: 0.869793 erro: 1.65847e-008
h: 0.01 x: 0.891063 erro: 1.6238e-008
h: 0.01 x: 0.912096 erro: 1.59021e-008
h: 0.01 x: 0.932902 erro: 1.55764e-008
h: 0.01 x: 0.953486 erro: 1.52607e-008
h: 0.01 x: 0.973856 erro: 1.49544e-008
h: 0.01 x: 0.994018 erro: 1.46572e-008
h: 0.01 x: 1.01398 erro: 1.43687e-008
h: 0.01 x: 1.03374 erro: 1.40887e-008
h: 0.01 x: 1.05332 erro: 1.38167e-008
h: 0.01 x: 1.07271 erro: 1.35525e-008
h: 0.01 x: 1.09192 erro: 1.32958e-008
h: 0.01 x: 1.11095 erro: 1.30463e-008
h: 0.01 x: 1.12982 erro: 1.28038e-008
h: 0.01 x: 1.14852 erro: 1.25679e-008
h: 0.01 x: 1.16706 erro: 1.23385e-008
h: 0.01 x: 1.18545 erro: 1.21153e-008
h: 0.01 x: 1.20368 erro: 1.18981e-008
h: 0.01 x: 1.22177 erro: 1.16867e-008
h: 0.01 x: 1.23971 erro: 1.14808e-008
h: 0.01 x: 1.25752 erro: 1.12803e-008
h: 0.01 x: 1.27518 erro: 1.1085e-008
h: 0.01 x: 1.29272 erro: 1.08948e-008
h: 0.01 x: 1.31013 erro: 1.07094e-008
h: 0.01 x: 1.32741 erro: 1.05286e-008
h: 0.01 x: 1.34456 erro: 1.03524e-008
h: 0.01 x: 1.3616 erro: 1.01806e-008
h: 0.01 x: 1.37852 erro: 1.0013e-008
h: 0.01 x: 1.39533 erro: 9.84951e-009
h: 0.01 x: 1.41203 erro: 9.68999e-009
h: 0.01 x: 1.42862 erro: 9.5343e-009
h: 0.01 x: 1.4451 erro: 9.38233e-009
h: 0.01 x: 1.46148 erro: 9.23396e-009
h: 0.01 x: 1.47776 erro: 9.08908e-009
h: 0.01 x: 1.49394 erro: 8.94757e-009
h: 0.01 x: 1.51002 erro: 8.80934e-009

h: 0.01 x: 1.52601 erro: 8.67429e-009
h: 0.01 x: 1.54191 erro: 8.54231e-009
h: 0.01 x: 1.55772 erro: 8.41332e-009
h: 0.01 x: 1.57344 erro: 8.28722e-009
h: 0.01 x: 1.58908 erro: 8.16394e-009
h: 0.01 x: 1.60463 erro: 8.04338e-009
h: 0.01 x: 1.6201 erro: 7.92547e-009
h: 0.01 x: 1.63549 erro: 7.81013e-009
h: 0.01 x: 1.6508 erro: 7.69729e-009
h: 0.01 x: 1.66604 erro: 7.58687e-009
h: 0.01 x: 1.6812 erro: 7.47881e-009
h: 0.01 x: 1.69629 erro: 7.37304e-009
h: 0.01 x: 1.7113 erro: 7.2695e-009
h: 0.01 x: 1.72625 erro: 7.16812e-009
h: 0.01 x: 1.74113 erro: 7.06884e-009
h: 0.01 x: 1.75594 erro: 6.97161e-009
h: 0.01 x: 1.77068 erro: 6.87638e-009
h: 0.01 x: 1.78536 erro: 6.78307e-009
h: 0.01 x: 1.79997 erro: 6.69165e-009
h: 0.01 x: 1.81452 erro: 6.60207e-009
h: 0.01 x: 1.82902 erro: 6.51427e-009
h: 0.01 x: 1.84345 erro: 6.42821e-009
h: 0.01 x: 1.85782 erro: 6.34385e-009
h: 0.01 x: 1.87214 erro: 6.26113e-009
h: 0.01 x: 1.8864 erro: 6.18002e-009
h: 0.01 x: 1.90061 erro: 6.10047e-009
h: 0.01 x: 1.91476 erro: 6.02245e-009
h: 0.01 x: 1.92886 erro: 5.94591e-009
h: 0.01 x: 1.9429 erro: 5.87082e-009
h: 0.01 x: 1.9569 erro: 5.79715e-009
h: 0.01 x: 1.97084 erro: 5.72485e-009
h: 0.01 x: 1.98474 erro: 5.65389e-009
h: 0.01 x: 1.99859 erro: 5.58425e-009
h: 0.01 x: 2.01239 erro: 5.51588e-009
h: 0.01 x: 2.02615 erro: 5.44876e-009
h: 0.01 x: 2.03985 erro: 5.38286e-009
h: 0.01 x: 2.05352 erro: 5.31815e-009
h: 0.01 x: 2.06714 erro: 5.25459e-009
h: 0.01 x: 2.08072 erro: 5.19217e-009
h: 0.01 x: 2.09425 erro: 5.13085e-009
h: 0.01 x: 2.10774 erro: 5.07061e-009
h: 0.01 x: 2.1212 erro: 5.01143e-009
h: 0.01 x: 2.13461 erro: 4.95327e-009
h: 0.01 x: 2.14798 erro: 4.89612e-009
h: 0.01 x: 2.16131 erro: 4.83995e-009
h: 0.01 x: 2.17461 erro: 4.78475e-009
h: 0.01 x: 2.18787 erro: 4.73048e-009
h: 0.01 x: 2.20109 erro: 4.67713e-009
h: 0.01 x: 2.21427 erro: 4.62468e-009
h: 0.01 x: 2.22742 erro: 4.5731e-009
h: 0.01 x: 2.24053 erro: 4.52238e-009
h: 0.01 x: 2.25361 erro: 4.4725e-009
h: 0.01 x: 2.26666 erro: 4.42345e-009
h: 0.01 x: 2.27967 erro: 4.37519e-009
h: 0.01 x: 2.29265 erro: 4.32772e-009
h: 0.01 x: 2.3056 erro: 4.28101e-009
h: 0.01 x: 2.31851 erro: 4.23506e-009
h: 0.01 x: 2.3314 erro: 4.18984e-009
h: 0.01 x: 2.34425 erro: 4.14535e-009
h: 0.01 x: 2.35708 erro: 4.10156e-009
h: 0.01 x: 2.36987 erro: 4.05846e-009
h: 0.01 x: 2.38263 erro: 4.01603e-009
h: 0.01 x: 2.39537 erro: 3.97426e-009
h: 0.01 x: 2.40808 erro: 3.93315e-009
h: 0.01 x: 2.42076 erro: 3.89267e-009
h: 0.01 x: 2.43341 erro: 3.8528e-009
h: 0.01 x: 2.44603 erro: 3.81355e-009
h: 0.01 x: 2.45863 erro: 3.7749e-009
h: 0.01 x: 2.4712 erro: 3.73683e-009
h: 0.01 x: 2.48375 erro: 3.69933e-009
h: 0.00298637 x: 2.49627 erro: 3.66239e-009
h: 0 x: 2.5 erro: 3.65146e-009

Através dos resultados foi possível fazer o gráfico solução numérica x tempo apresentado abaixo.

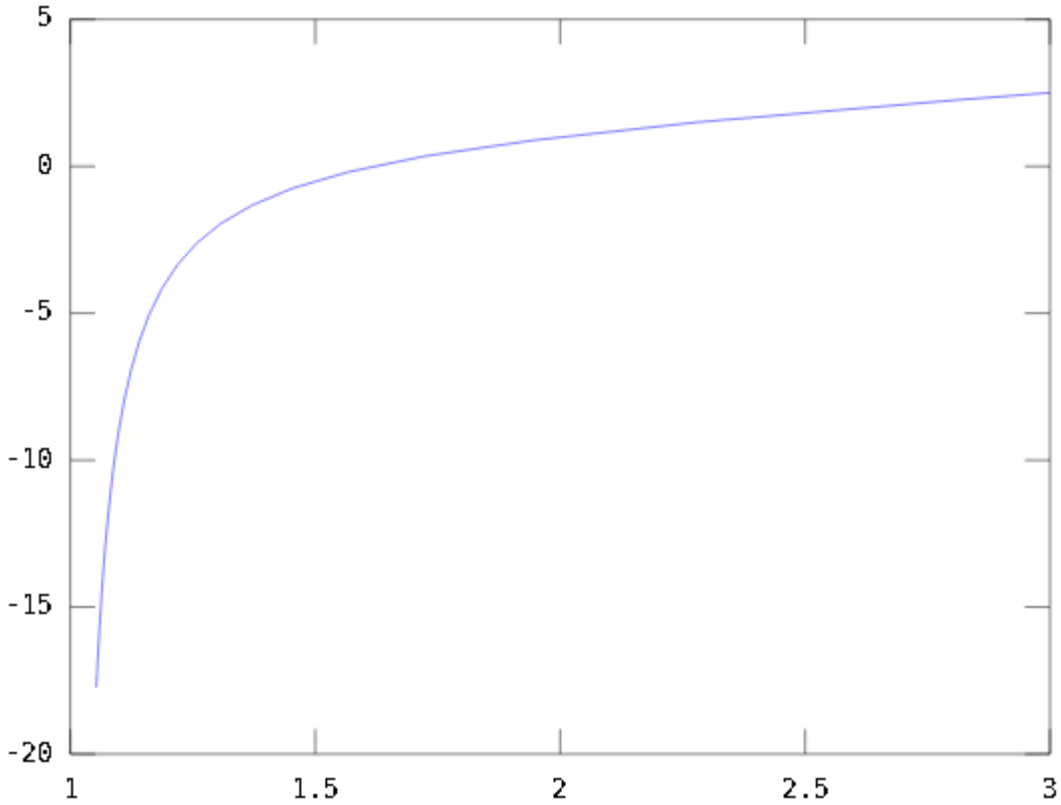

Gráfico 1 - Solução numérica do teste 1 x tempo.

3.2. Teste 2

O segundo teste consiste em resolver a equação diferencial autônoma F(t, X) = AX, onde:

$$A = \begin{pmatrix} -2 & -1 & -1 & -2 \\ 1 & -2 & 2 & -1 \\ -1 & -2 & -2 & -1 \\ 2 & -1 & 1 & -2 \end{pmatrix}$$

A equação foi integrada a partir de X(0) = (1, 1, 1, −1) até t_f = 2, usando h = 0.1 inicialmente e eps = 10−5. A cada passo foi impresso o valor de h, ti e ||Xi − $\bar{X}$(ti)||.

Através dos resultados foi possível fazer os gráficos referentes a x1(t), x2(t), x3(t) e x4(t) variando no tempo. Segue abaixo os gráficos.

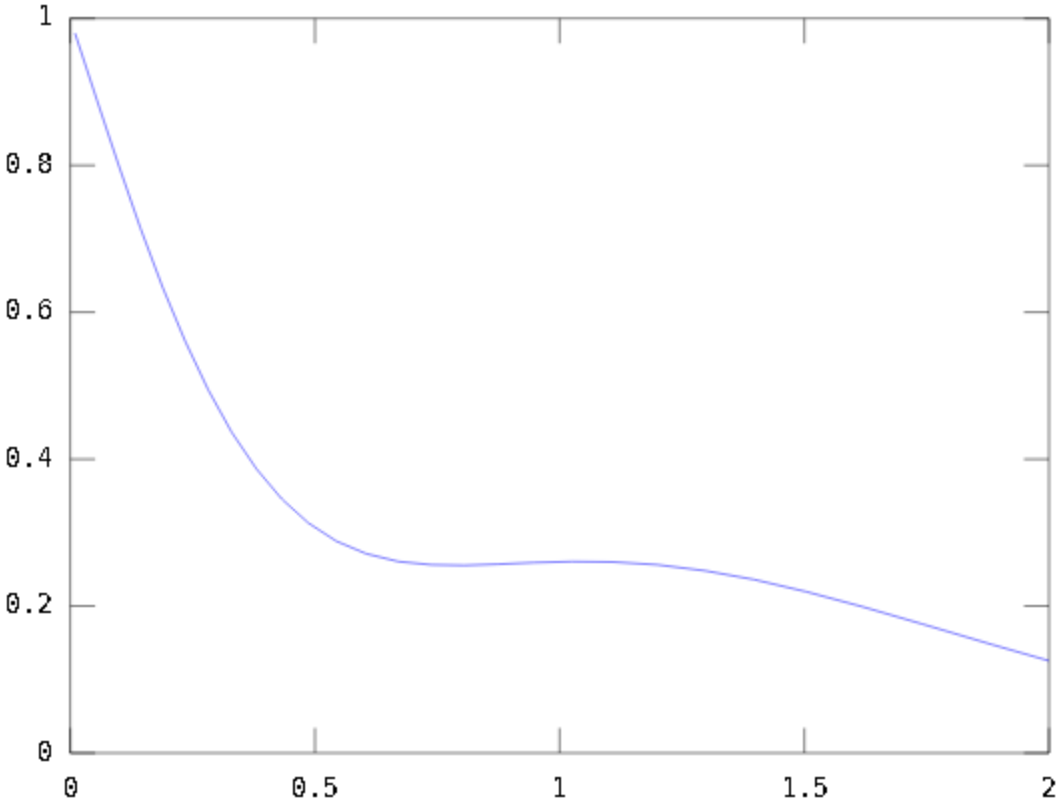

Gráfico 2 - Solução para x1(t) x tempo.

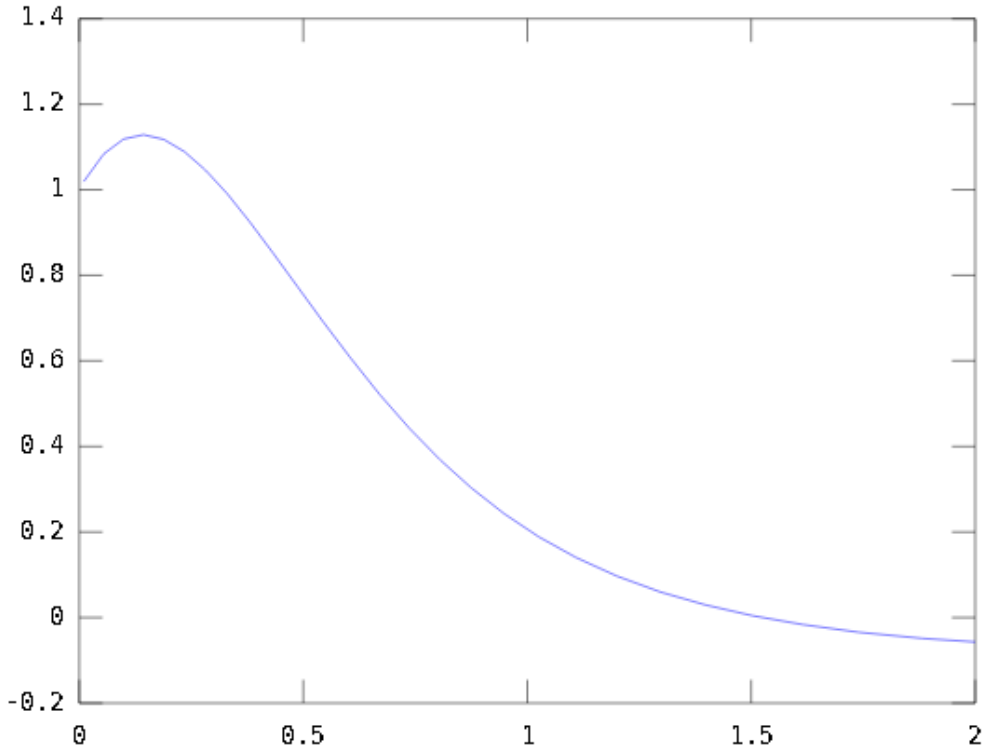

Gráfico 3 - Solução para x2(t) x tempo.

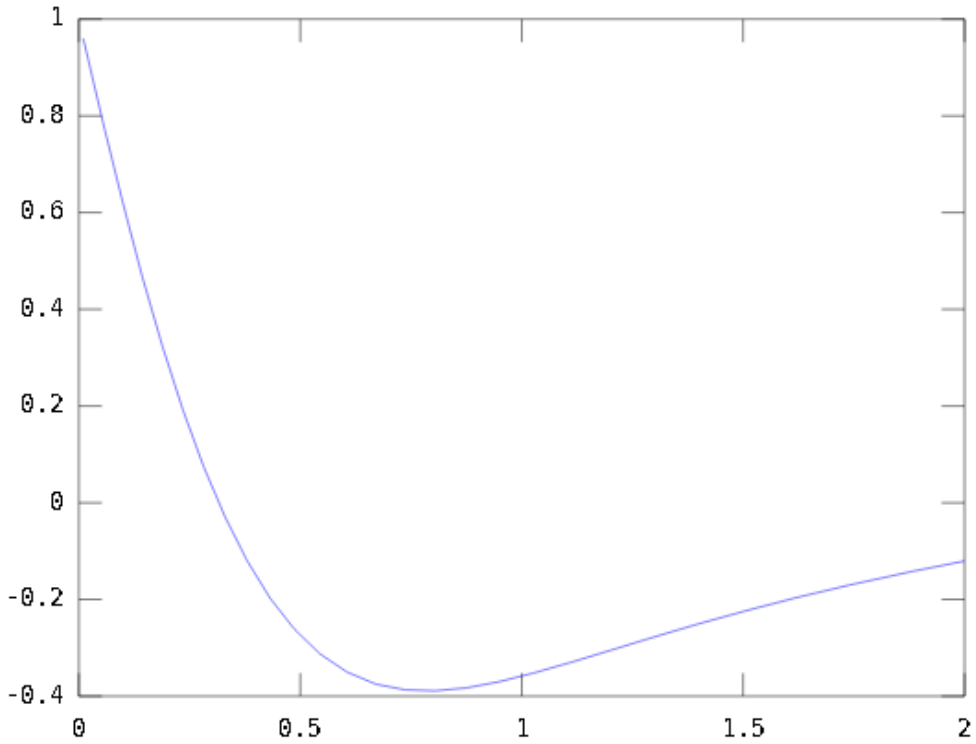

Gráfico 4 - Solução para x3(t) x tempo.

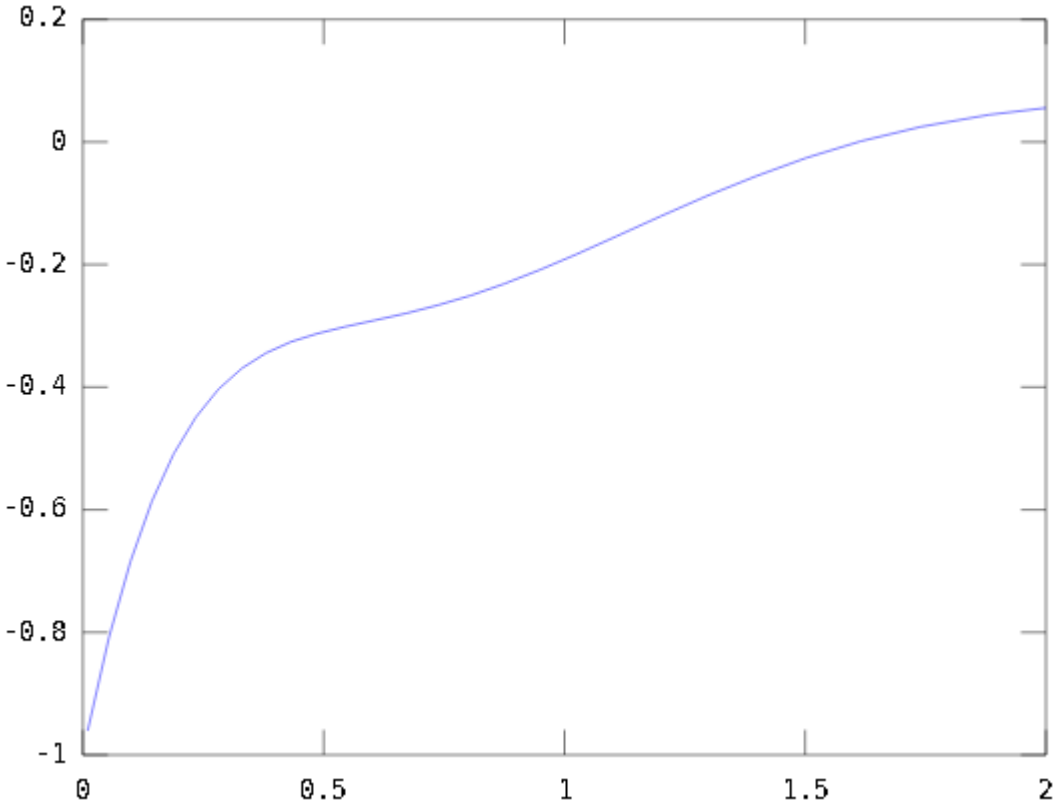

Gráfico 5 - Solução para x4(t) x tempo.

Além disso também foi impresso o h, a solução e erro em relação a solução exata desta equação. Porém será omitido neste relatório por questões de tamanho. Cabe ao examinador testar o código e gerar tais valores a partir das instruções dadas neste relatório.

3.3. Teste 3

O terceiro teste consiste em resolver o sistema diferencial autônomo F(t, X) = AX, com $X \in R^m$, onde A é uma matriz tridiagonal m × m, tal que Ai,i = −2, i = 1, .., m, Ai,i+1 = Ai+1,i = 1, i = 1, .., m − 1 e Ai,j = 0 nas posições restantes.

Para tal fez-se uso da condição inicial X(0)i = sin(πyi) + sin(mπyi), onde yi = i / (m + 1), para i = 1, ..., m. O teste foi feito com m = 7, usando h = 0.1 inicialmente e eps = 10−5. A cada passo foi impresso o valor de h, ti e ||Xi − X(ti)||.

Foi impresso o h, a solução e erro em relação a solução exata desta equação. Porém será omitido neste relatório por questões de tamanho. Cabe ao examinador testar o código e gerar tais valores a partir das instruções dadas neste relatório.

Apenas para efeito de curiosidade, segue abaixo as primeiras iterações feitas para esse teste. Vale notar que ‘x’ e ‘erro’ são vetores e seus elementos são impressos intercalados por um espaço.

h: 0.01
x: 0.0631286 0.640497 0.21528 1.5387 0.304198 1.53556 0.152046
h: 0.01
x: 0.00750267 0.750377 0.0256154 1.81148 0.0362254 1.81144 0.0181126
erro: 0.00750267 3.75122e-005 1.25123e-007 1.0404e-010 1.24931e-007 3.75124e-005 0.00750267
h: 0.01
x: 0.0147115 0.736006 0.0502257 1.77652 0.0710275 1.77637 0.0355132
erro: 0.0147115 0.000147098 9.80744e-007 2.00226e-010 9.80374e-007 0.000147098 0.0147115
h: 0.01
x: 0.0216382 0.722228 0.0738695 1.74283 0.104459 1.7425 0.0522281
erro: 0.0216382 0.000324489 3.2446e-006 2.89002e-010 3.24407e-006 0.00032449 0.0216382
h: 0.01
x: 0.0282942 0.709019 0.0965842 1.71036 0.136572 1.70979 0.0682825

erro: 0.0282942 0.000565626 7.53981e-006 3.70791e-010 7.53912e-006 0.000565627 0.0282942
h: 0.01
x: 0.0346903 0.696357 0.118405 1.67906 0.167416 1.6782 0.0837006
erro: 0.0346903 0.000866642 1.44379e-005 4.45993e-010 1.44371e-005 0.000866643 0.0346903
h: 0.01
x: 0.0408371 0.684219 0.139368 1.6489 0.197036 1.64767 0.098506
erro: 0.0408371 0.00122386 2.44617e-005 5.1499e-010 2.44608e-005 0.00122386 0.0408371
h: 0.01
x: 0.0467445 0.672585 0.159504 1.61982 0.225481 1.61819 0.112721
erro: 0.0467445 0.00163378 3.80884e-005 5.78143e-010 3.80873e-005 0.00163378 0.0467445
h: 0.01
x: 0.0524222 0.661434 0.178846 1.59179 0.252792 1.5897 0.126368
erro: 0.0524222 0.00209308 5.57516e-005 6.35795e-010 5.57504e-005 0.00209308 0.0524222
h: 0.01
x: 0.0578793 0.650747 0.197424 1.56477 0.279012 1.56217 0.139467
erro: 0.0578793 0.00259859 7.78445e-005 6.8827e-010 7.78432e-005 0.00259859 0.0578793

3.4. Circuito de Chua

Para fazer o teste com o circuito de Chua foram considerados os seguintes parâmetros:

- C_1 = 10 nF;
- C_2 = 100 nF;
- L = 18 mH;
- E = 1.17391304 V;
- G_a = -50/66 mS;
- G_b = -9/22 mS;
- E_{max} = 8.1818 V;
- G_c = 4.591 mS.

Além disso, foram consideradas as seguintes condições iniciais: $V_1 = -0.5V$, $V_2 = -0.2$ V e $I_L = 0$. O valor de R é variado nos experimentos a seguir.

3.4.1. Para R < 1500 Ohms

Primeiro verificou-se o comportamento do sistema quando R < 1500 Ohms. Para tal usou-se R=1450 Ohms, h_min = 10^{-20}, h_max = 0.01 e eps = 10^{-1}. Segue abaixo um gráfico 3D da solução encontrada.

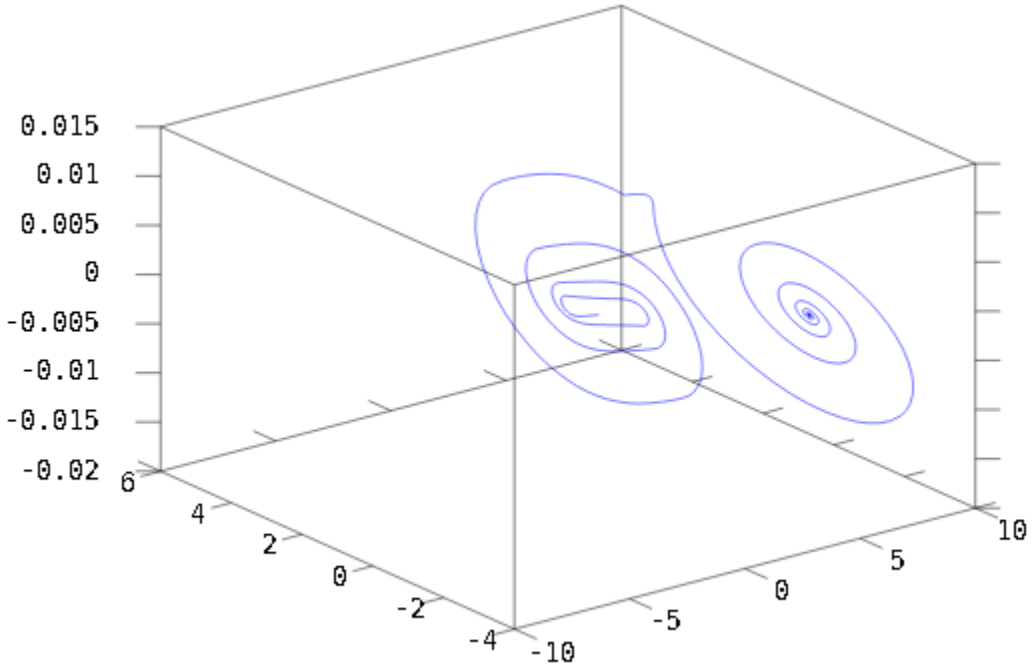

Gráfico 6 - Solução para R = 1450 Ohms.

Pode-se perceber que o comportamento do sistema apresenta um “padrão” que pode ser notado mais à frente neste relatório. Basta por agora perceber a imagem que se forma neste gráfico e sua repetição (um pouco distorcida) em outros valores de R.

3.4.2. Para R = 1600 Ohms

Depois verificou-se o comportamento do sistema quando R = 1600 Ohms. Para tal utilizou-se R=1600 Ohms, h_min = 10^{-20}, h_max = 0.01 e eps = 10^{-1}. Segue abaixo um gráfico 3D da solução encontrada.

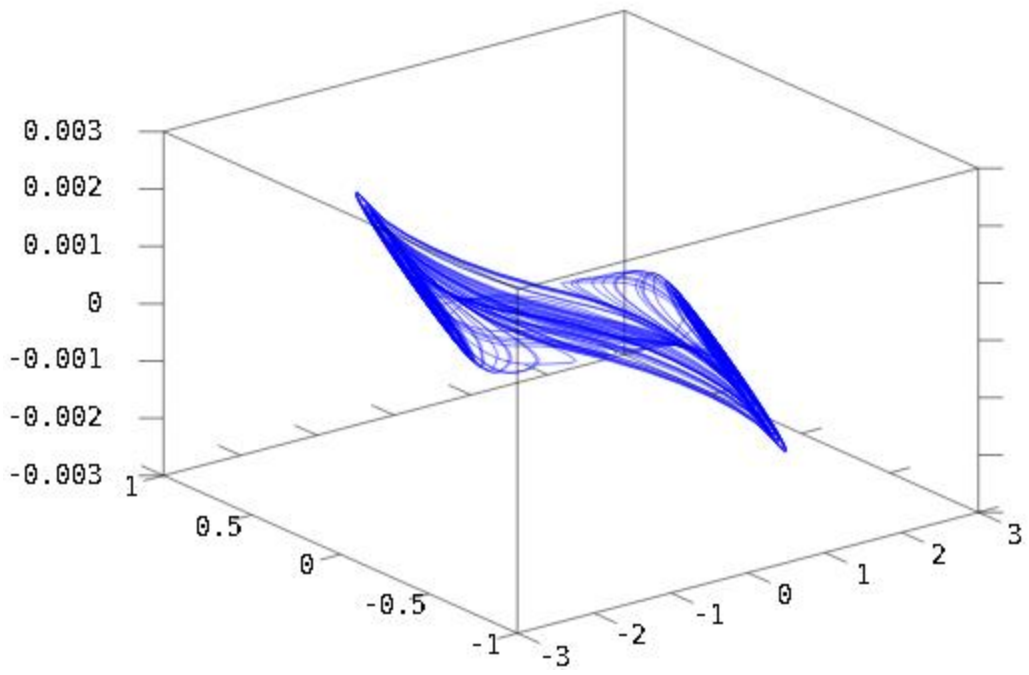

Gráfico 7 - Solução para R=1600 Ohms.

Neste segundo exercício vê-se que a imagem que se forma é bem diferente da que foi obtida no primeiro exercício. Esse é o segundo “padrão” que será observado durante as demais mudanças que forem feitas de R.

Nota-se que segundo o enunciado há três partes, do gráfico v x i do diodo de Chua, que apresentam linearidade. A primeira ocorre quando a tensão que cai sobre o diodo é menor que -E, a segunda quando a tensão está entre -E e E e a terceira quando a tensão é maior que E.

Ao variarmos o R a tensão que cai no diodo irá mudar, deslocando o comportamento do sistema para uma dessas parte de linearidade do diodo de Chua. Assim, no primeiro exercício pode-se dizer que a tensão que cai sobre o diodo está em uma dessas faixas de linearidade, enquanto que neste segundo exercício essa tensão encontra-se em outra dessas faixas de linearidade.

3.4.3. Valor de R para uma bifurcação do sistema

O terceiro exercício consistiu em encontrar o valor de R entre 1500 e 1600 Ohms para o qual o sistema passa de um comportamento para outro, ou seja, há uma bifurcação do sistema.

Para tal variou-se o R de forma a encontrar uma mudança de comportamento da solução. Foi utilizado h_min = 10^{-20}, h_max = 0.01 e eps = 10^{-1}. Segue abaixo o gráfico de R = 1538,9 Ohms e de R = 1539 Ohms.

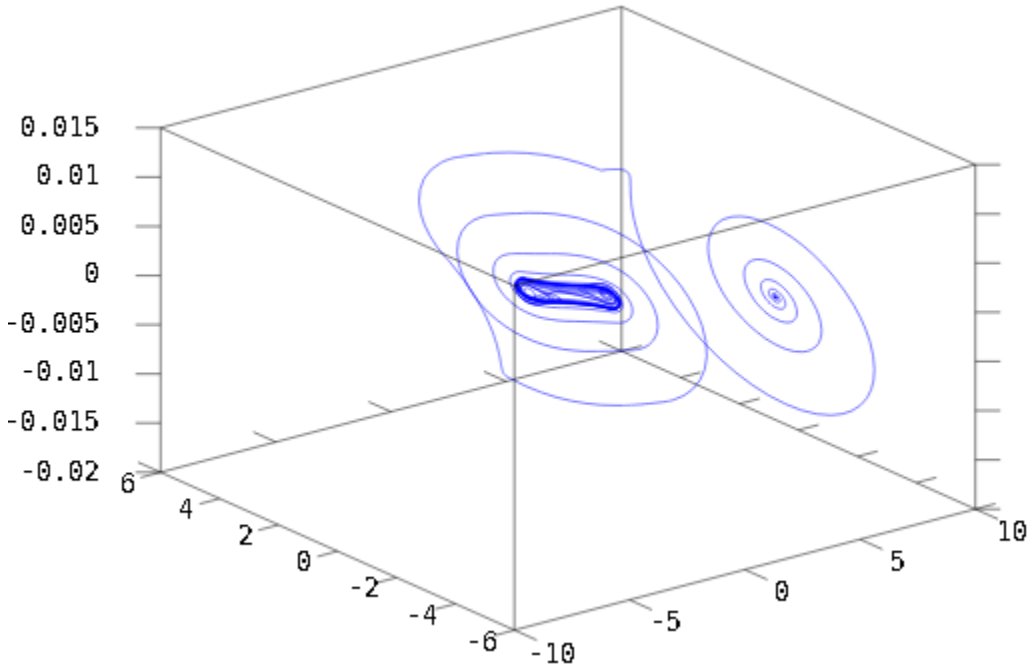

Gráfico 8 - Solução para R=1538,9 Ohms.

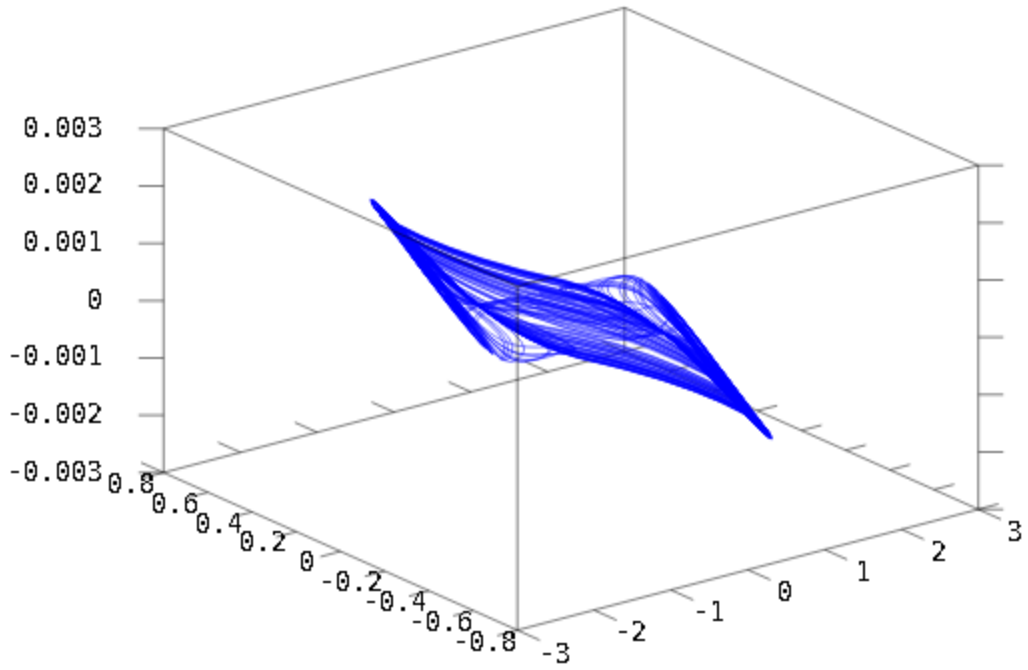

Gráfico 9 - Solução para R=1539 Ohms.

Percebe-se nitidamente a mudança de comportamento da solução na transição de R = 1538,9 Ohms e R = 1539 Ohms. Assim, podemos concluir que 1539 Ohms é o R para o qual o sistema sofre uma bifurcação.

Aqui podemos onde ocorre essa transição em que a tensão que cai no diodo de Chua está em uma faixa de linearidade e sua passagem para outra faixa de linearidade.

3.4.4. Para R > 2000 Ohms

Neste quarto exercício verificou-se o comportamento do sistema quando R > 2000 Ohms. Para tal utilizou-se h_min = 10^{-20}, h_max = 0.01, eps = 10^{-1} e três valores de R: 2000 Ohms, 2500 Ohms e 5000 Ohms. Segue abaixo um gráfico 3D das respectivas soluções encontradas.

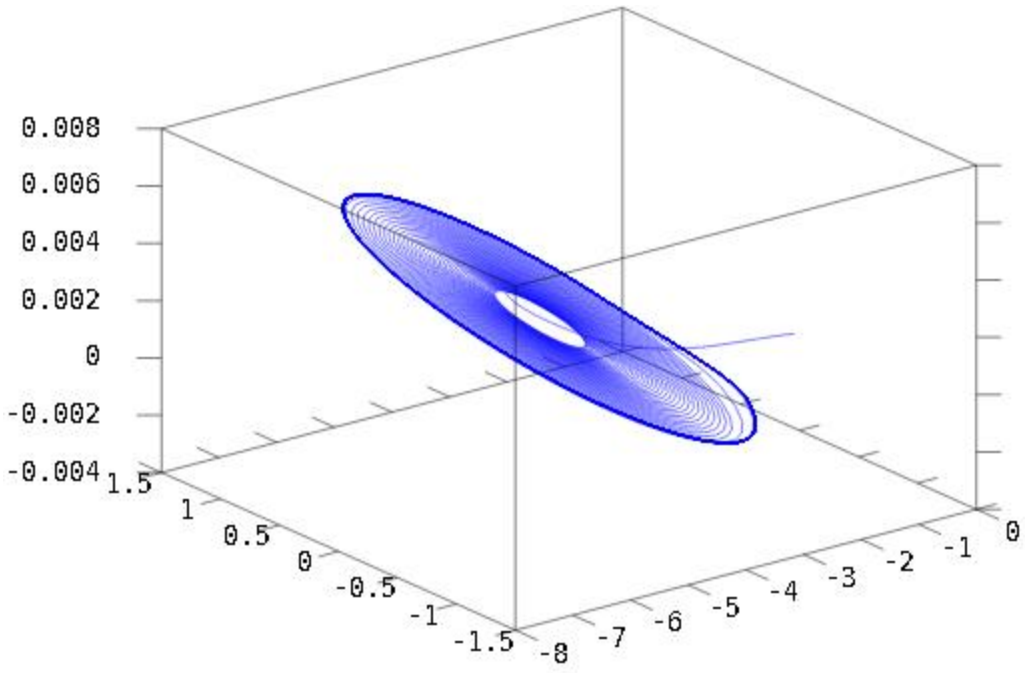

Gráfico 10 - Solução para R=2000 Ohms.

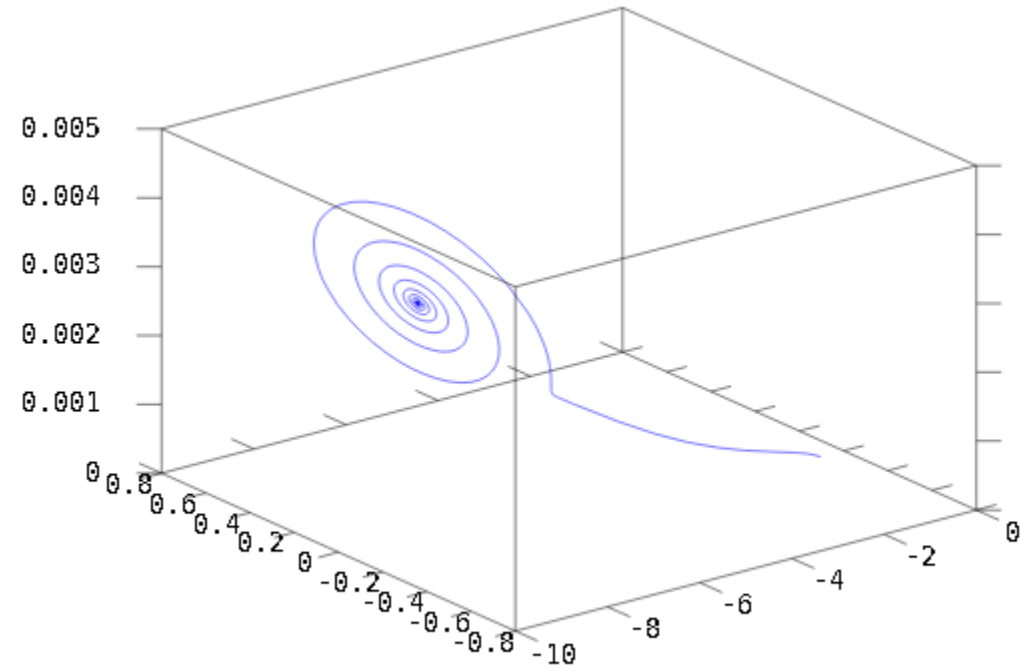

Gráfico 11 - Solução para R=2500 Ohms.

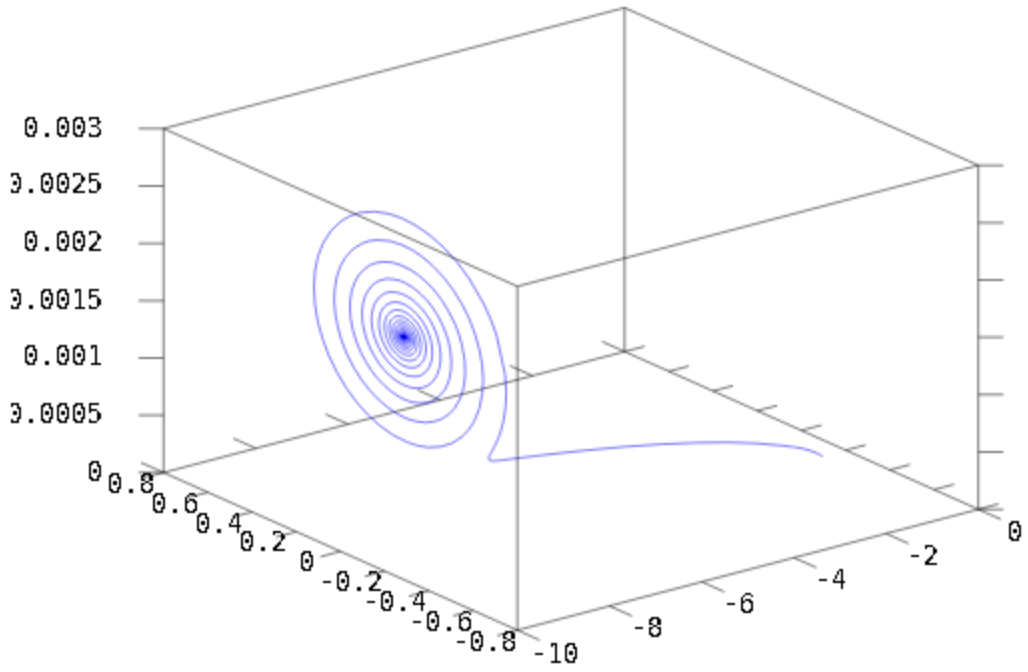

Gráfico 12 - Solução para R=5000 Ohms.

Neste quarto exercício pode-se notar que, novamente, a tensão que cai sobre o diodo de Chua passa de uma faixa de linearidade para outra faixa de linearidade. Isso ocasiona na mudança de “padrão” de imagem obtida na solução do sistema, que passa a lembrar o padrão obtido quando R < 1500 Ohms.

3.4.5. Tempo de execução do código

Os exercícios anteriores foram realizados com h_min = 10^{-20}, e =10^{-1}, ti = 0.0, tf = 0.5.

R (Ohms)	Tempo de execução (s)
0.5	> 600
1500	15.3
1800	9.6

10000	11.6
100000	11.6

Tabela 1 - Tempos de execução.

O tempo de execução é influenciado pelo valor de R, pois depende da região do sistema que está, se o comportamento é instável ou estável em tal região. Além disso, quanto menor o valor de R, maiores são os valores das tensões, sendo elas estando em regiões mais críticas do sistema.

Regiões mais críticas possuem uma variação mais brusca das funções exigindo valores de h menores possíveis. E quanto menor o passo (h), maior vai ser o número de iterações e consequentemente maior será o tempo de execução.

3.5. Discussões extras

3.5.1. Variação dos parâmetros do circuito

O aumento do valor do parâmetro C1 do circuito faz com que a convergência do sistema seja mais rápida. Isso, porque a equação de V1 mostra que a dependência dos valores de estado da função não-linear g(v1) é proporcional ao inverso de C1, sendo esta função a responsável pelo comportamento não-linear que dificulta a convergência.

O aumento dos outros valores de parâmetro (R, C2, L) também podem ter esse efeito por conta das razões apontadas no item anterior. A figura a seguir mostra o retrato de fase para R = 1500 Ohms, C1 = 1e-4 F, e os outros parâmetros com mesmo valor e mesmas condições iniciais.

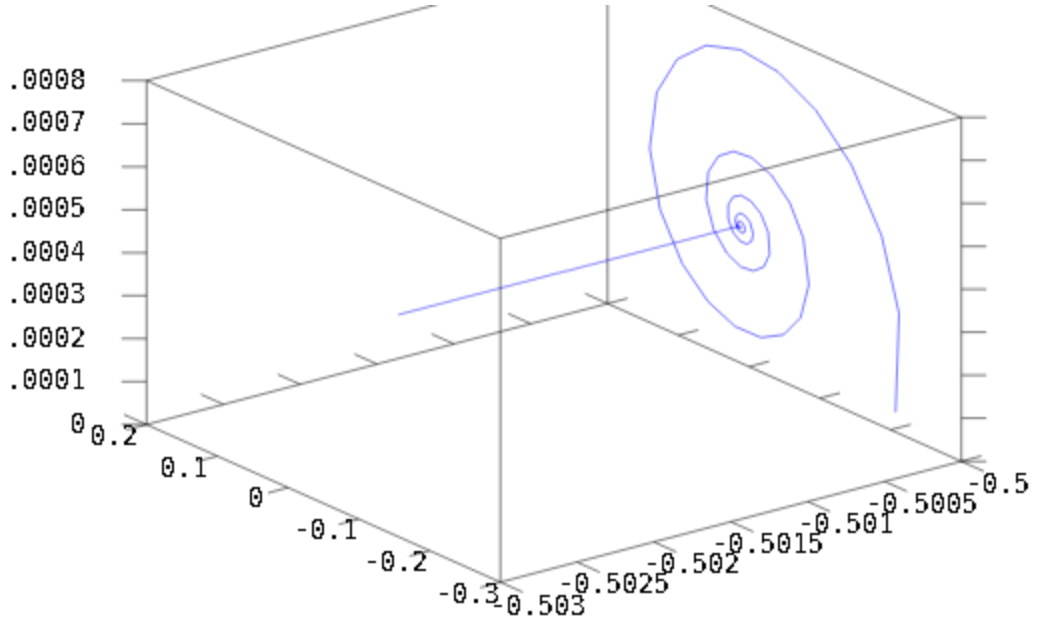

Gráfico 13 - Solução para R=1500 Ohms e C1 = 10^{-4} F.

3.5.2. Condições iniciais

Para condições iniciais nulas, o sistema se manteve no estado zero.

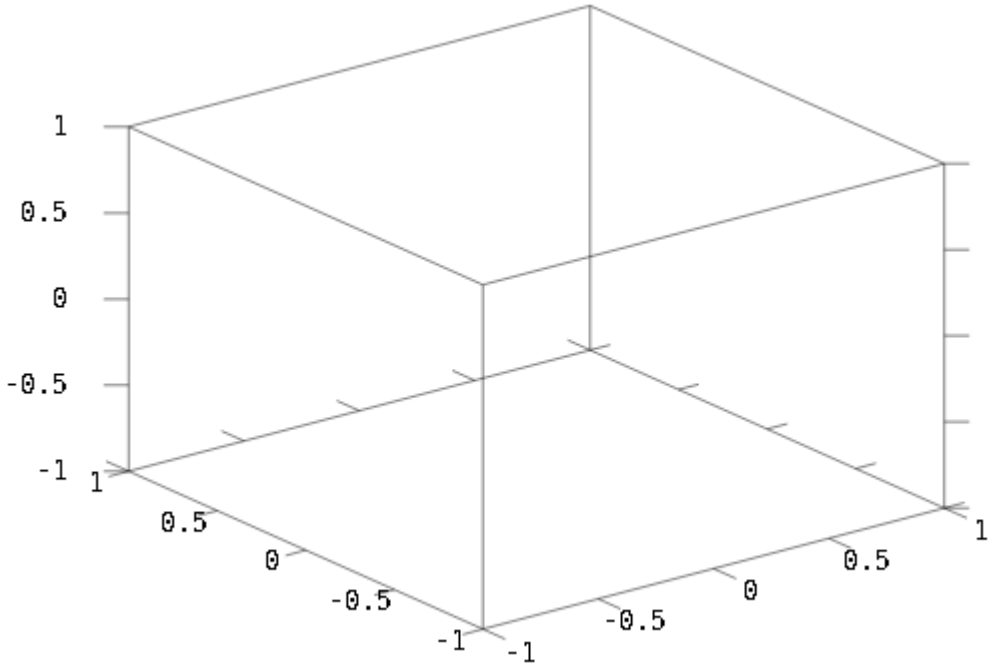

Gráfico 14 - Solução para condições iniciais nulas.

Ao se inserir um valor inicial não nulo de corrente no indutor (no caso 0.5), o sistema teve fortes dificuldades para convergir, mostrando alta sensibilidade do sistema também às condições iniciais

Quanto aos parâmetros do algoritmo, vimos que em vários casos o sistema divergia para h > 1e-20, principalmente para R < 1600. Novamente, isso se deve ao caráter do sistema de equações que possui uma forte sensibilidade à variações pequenas. Para acompanhar tais variações é necessário um passo tão pequeno quanto o possível. O valor h = 1e-20 se mostrou suficiente.

4. Conclusão

Ao final dessa atividade pode-se desenvolver um programa em C++ que resolve o circuito de Chua, através da resolução numérica de equações diferenciais ordinárias.

Para tal foi preciso compreender e desenvolver um programa que faz o método RKF45 e fazer alguns testes preliminares com 1 dimensão, 4 dimensões e m dimensões.