Capítulo 1

Implementação

Neste capítulo do projeto de engenharia apresentamos os códigos fonte que foram desenvolvidos.

1.1 Código fonte

Apresenta-se a seguir um conjunto de classes (arquivos .h e .cpp) além do programa `main`.

Apresenta-se na listagem 1.1 o arquivo com código da classe CParticulaFluido.hpp.

Listing 1.1: Arquivo de cabeçalho da classe CParticulaFluido.hpp

```
1 #ifndef PARTICULAFLUIDO_HPP
2 #define PARTICULAFLUIDO_HPP
3
4 #include <fstream>
5 #include <iostream>
6 #include <string>
7
8 class CParticulaFluido {
9 protected:
10         double viscosidade;
11         double lambdaPonte;
12         double lambdaAdesao;
13         double parC;
14         double par_n;
15         double velocidade;
16         double C = 2.0;
17         double cb = 0.0004;
18
19         void readFile(std::string path);
20
```

```
21 public:
22         //CParticulaFluido() {}
23         // construtores por variáveis e por nome do arquivo txt
24         CParticulaFluido(double _mu, double _lambdaPonte, double
              _lambdaAdesao, double _parC, double _par_n, double
              _velocidade):
25                 viscosidade{ _mu }, lambdaPonte{ _lambdaPonte },
                      lambdaAdesao{ _lambdaAdesao }, parC{ _parC },
                      par_n{ _par_n }, velocidade{ _velocidade } {}
26
27         CParticulaFluido(std::string path) { readFile(path); }
28
29         // funções get-set
30         double getViscosidade() { return viscosidade; };
31         void setViscosidade(double novo_mu) { viscosidade = novo_mu
              ; }
32
33         double getLambdaPonte() { return lambdaPonte; }
34         double getLambdaAdesao() { return lambdaAdesao; }
35         double getParC() { return parC; }
36         double getParN() { return par_n; }
37         double getVelocidade() { return velocidade; }
38
39         // função para mostrar os parâmetros da classe
40         void printCParticulaFluido();
41 };
42
43 #endif
```

Apresenta-se na listagem 1.2 o arquivo de implementação da classe CParticulaFluido.cpp.

Listing 1.2: Arquivo de implementação da classe CParticulaFluido.cpp

```
1 #include "CParticulaFluido.hpp"
2
3 void CParticulaFluido::readFile(std::string path) {
4         std::ifstream infile;
5         infile.open(path);
6         std::string temp;
7         std::getline(infile, temp); // pular linha com texto
8         std::getline(infile, temp);             // viscosidade
9         viscosidade = atof(temp.c_str());
10
11        std::getline(infile, temp); // pular linha com texto
```

```
12          std::getline(infile, temp);              // lambdaPonte
13          lambdaPonte = atof(temp.c_str());
14
15          std::getline(infile, temp); // pular linha com texto
16          std::getline(infile, temp);              // lambdaAdesao
17          lambdaAdesao = atof(temp.c_str());
18
19          std::getline(infile, temp); // pular linha com texto
20          std::getline(infile, temp);              // parC
21          parC = atof(temp.c_str());
22
23          std::getline(infile, temp); // pular linha com texto
24          std::getline(infile, temp);              // par_n
25          par_n = atof(temp.c_str());
26
27          std::getline(infile, temp); // pular linha com texto
28          std::getline(infile, temp);              // velocidade
29          velocidade = atof(temp.c_str());
30
31 }
32
33 void CParticulaFluido::printCParticulaFluido() {
34          std::cout << "\n---------------------------" << std::endl;
35          std::cout << "Valores de ParticulaFluido" << std::endl;
36          std::cout << "---------------------------" << std::endl;
37          std::cout << "viscosidade: " << viscosidade << std::endl;
38          std::cout << "lambdaPonte: " << lambdaPonte << std::endl;
39          std::cout << "lambdaAdesao: " << lambdaAdesao << std::endl;
40          std::cout << "parC: " << parC << std::endl;
41          std::cout << "par_n: " << par_n << std::endl;
42          std::cout << "velocidade: " << velocidade << std::endl;
43 }
```

Apresenta-se na listagem 1.3 o arquivo de implementação da classe CRocha.hpp.

Listing 1.3: Arquivo de implementação da classe CRocha.hpp

```
1 #pragma once
2
3 #include <fstream>
4 #include <iostream>
5 #include <string>
6
7 class CRocha{
```

```
protected:
        double permeabilidade;
        double porosidade;
        double sigma_am;
        double N;
        double beta = 0.5;
        double porcentagemCaulinita = 0.2;

        double sigma_a0;

        void readFile(std::string path);
public:
        //CRocha() {}
        // construtores por variáveis e por nome do arquivo txt
        CRocha(double k, double fi, double _sigma_a0, double
           _sigma_am, double _N);// : permeabilidade{ k },
           porosidade{ fi }, sigma_a0{ _sigma_a0 }, sigma_am{
           _sigma_am }, N{ _N } {}
        CRocha(std::string path) {
                readFile(path); }

        double reducaoPermeabilidade(double sigma) { return 1 / (1
           + beta * sigma); }

        // função para mostrar os parâmetros da classe
        void printCRocha();
};
```

Apresenta-se na listagem 1.4 o arquivo de implementação da classe CRocha.cpp.

Listing 1.4: Arquivo de implementação da classe CRocha.cpp

```
#include "CRocha.hpp"

CRocha::CRocha(double k, double fi, double _sigma_a0, double
    _sigma_am, double _N) :
                permeabilidade{ k }, porosidade{ fi }, sigma_a0{
                   _sigma_a0 }, sigma_am{ _sigma_am }, N{ _N } {
        sigma_a0 = (1 - porosidade) * porcentagemCaulinita;
}

void CRocha::readFile(std::string path) {
        std::ifstream infile;
        infile.open(path);
```

```
11          std::string temp;
12          std::getline(infile, temp); // pular linha com texto
13          std::getline(infile, temp);             // permeabilidade
14          permeabilidade = atof(temp.c_str());
15
16          std::getline(infile, temp); // pular linha com texto
17          std::getline(infile, temp);             // porosidade
18          porosidade = atof(temp.c_str());
19
20          std::getline(infile, temp); // pular linha com texto
21          std::getline(infile, temp);             // sigma_a0
22          sigma_a0 = atof(temp.c_str());
23
24          std::getline(infile, temp); // pular linha com texto
25          std::getline(infile, temp);             // sigma_am
26          sigma_am = atof(temp.c_str());
27
28          std::getline(infile, temp); // pular linha com texto
29          std::getline(infile, temp);             // N
30          N = atof(temp.c_str());
31  }
32
33  void CRocha::printCRocha() {
34          std::cout << "\n-----------------" << std::endl;
35          std::cout << "Valores de CRocha" << std::endl;
36          std::cout << "-----------------" << std::endl;
37          std::cout << "permeabilidade: " << permeabilidade << std::
                endl;
38          std::cout << "porosidade: " << porosidade << std::endl;
39          std::cout << "sigma_a0: " << sigma_a0 << std::endl;
40          std::cout << "sigma_am: " << sigma_am << std::endl;
41          std::cout << "N: " << N << std::endl;
42  }
```

Apresenta-se na listagem 1.5 o arquivo de implementação da classe CGrid.hpp.

Listing 1.5: Arquivo de implementação da classe CGrid.cpp

```
1 #ifndef CGRID_HPP
2 #define CGRID_HPP
3
4 #include <string>
5 #include <vector>
6 #include <fstream> /// escrecrever em disco
```

```
7 #include <iomanip>      /// setw
8
9 class CGrid {
10 private:
11         int _size;
12 public:
13         std::vector<double> sigma_a, sigma_s, diff_sigma_a;
14
15 public:
16
17         CGrid(int size_x) {
18                 _size = size_x;
19                 //c_susp.resize(_size, 0.0);
20                 sigma_a.resize(_size, 0.0);
21                 sigma_s.resize(_size,0.0);
22                 diff_sigma_a.resize(_size,0.0);
23                 //concentracao.resize(_size, 0.0);
24         }
25
26         int get_size() { return _size;}
27
28         //método para salvar os concentrações no formato .txt na
               pasta "resultados malha"
29         void saveGrid(std::vector<double> malha, double time);
30         std::vector<double> get_sigma_s() { return sigma_s; }
31
32         /// metodo para criar malhas, eh static para criar as
               malhas do espaco e do tempo
33         std::vector<double> static linspace(double start, double
               end, int size);
34 };
35 #endif
```

Apresenta-se na listagem 1.6 o arquivo de implementação da classe CGrid.cpp.

Listing 1.6: Arquivo de implementação da classe CGrid.cpp

```
1 #include "CGrid.hpp"
2
3
4 void CGrid::saveGrid(std::vector<double> malha, double time) {
5         std::string path = "resultados_malha//malha_" + std::
              to_string(time) + ".txt";
6
```

```
7         std::ofstream file(path);
8         file << "␣␣␣␣␣malha␣␣-␣␣sigma_a␣␣-␣␣sigma_s␣-␣diff_sigma_a␣
              \n";
9         for (unsigned int i = 0; i < _size; i++) {
10                file << std::setw(9) << malha[i]
11                        << std::setw(12) << sigma_a[i]
12                        << std::setw(12) << sigma_s[i]
13                        << std::setw(15) << diff_sigma_a[i] << "\n"
                              ;
14        }
15        file.close();
16}
17
18
19std::vector<double> CGrid::linspace(double start, double end, int
      size) {
20        double delta = (end - start) / (1.0 * size - 1);
21        std::vector<double> grid;
22
23        for (int i = 0; i < size; i++)
24                grid.push_back(start + delta * i);
25
26        return grid;
27}
```

Apresenta-se na listagem 1.7 o arquivo de implementação da classe CSimuladorParticulas.hpp.

Listing 1.7: Arquivo de implementação da classe CSimuladorParticulas.cpp

```
1#ifndef CSIMULADORPARTICULAS_HPP
2#define CSIMULADORPARTICULAS_HPP
3
4#include "CGrid.cpp"
5#include "CRocha.cpp"
6#include "funcao.hpp"
7#include "CGnuplot.cpp"
8#include "metodosimpson.cpp"
9#include "CParticulaFluido.cpp"
10
11#include<vector>
12#include<string>
13#include<iostream>
14
```

```
15 class CSimuladorParticulas : public CParticulaFluido, CRocha {
16 private:
17         size_t indiceTempoAtual = 0;
18         size_t size_tempo = 101;
19         size_t size_malha = 1001;
20         size_t numPontosIntegral = 1001;
21         double start_x = 0.0, end_x = 0.10, start_t = 0, end_t =
               200.0;
22
23         std::vector<double> tempo;
24         std::vector<double> malha;
25
26         std::vector<CGrid*> resultados_ao_longo_do_tempo;
27
28 public:
29         /// CONSTRUTORES
30         CSimuladorParticulas(std::string pathParticulaFluido, std::
               string pathRocha);
31
32         /// 'MAIN' metodo, ele que esta executando o objeto
33         void run();
34
35         /// CALCULOS
36 private:
37         double CalculoSigma_a(double x, double t);
38         double CalculoDiffSigma_a(double x, double t);
39         double CalculoLinhaZona(double x);
40         double CalculoTb(double x);
41         //double CalculoConcentracoes(double x, double t);
42         double CalculoConcentracoes_N_igual_1(double x, double t);
43         double CalculoConcentracoes_N_diferente_1(double x, double
               t);
44         double CalculoDeltaPressao(double t);
45
46         double CalculoSigma(double x, double t);
47         double CalculoSigma_N_igual_1(double x, double t);
48         double CalculoSigma_N_diferente_1(double x, double t);
49         double funcao_sigma_n1(double x, double t);
50         double funcao_sigma_n_dif_1(double x, double t);
51
52 public:
53         /// metodos para SALVAR e APRESENTAR os resultados
```

```
54          void printCSimuladorParticulas();
55          void print_vector(std::vector<double> vetor);
56
57          void saveInFile(std::vector<double> vector1, std::string
               name_vector1);
58          void saveInFile(std::vector<double> vector1, std::vector<
               double> vector2, std::string name_vector1, std::string
               name_vector2);
59
60          void plot(std::vector<double> vector1, std::vector<double>
               vector2, std::string name_vector1, std::string
               name_vector2);
61 };
62 #endif
```

Apresenta-se na listagem 1.8 o arquivo de implementação da classe CSimuladorParticulas.cpp.

Listing 1.8: Arquivo de implementação da classe CSimuladorParticulas.cpp

```
1 #include "CSimuladorParticulas.hpp"
2
3 CSimuladorParticulas::CSimuladorParticulas(std::string
    pathParticulaFluido, std::string pathRocha):CParticulaFluido(
    pathParticulaFluido), CRocha(pathRocha) {
4           tempo = CGrid::linspace(start_t, end_t, size_tempo); ///
               nao eh objeto CGrid, eh vetor - metodo static
5           malha = CGrid::linspace(start_x, end_x, size_malha);
6           /// inicio as malhas
7           resultados_ao_longo_do_tempo.resize(size_tempo);
8           for (unsigned int i = 0; i < size_tempo; i++)
9                   resultados_ao_longo_do_tempo[i] = new CGrid(
                       size_malha);
10 }
11
12 void CSimuladorParticulas::run() {
13          int nx = resultados_ao_longo_do_tempo[0]->get_size();
14          std::vector<double>linhazona(nx);
15          std::vector<double>tb(nx);
16          std::vector<double>delta_p(size_tempo);
17
18          for (unsigned int j = 0; j < size_tempo; j++) { /// loop
               dos tempos
19                  for (unsigned int i = 0; i < nx; i++) { /// loop ao
```

```
                    longo do testemunho
20                      resultados_ao_longo_do_tempo[j]->sigma_a[i]
                            = CalculoSigma_a(malha[i], tempo[j]);
21                      resultados_ao_longo_do_tempo[j]->
                           diff_sigma_a[i] = CalculoDiffSigma_a(
                           malha[i], tempo[j]);
22                      //resultados_ao_longo_do_tempo[j]->
                           concentracao[i] = CalculoConcentracoes(
                           malha[i], tempo[j]);
23                      resultados_ao_longo_do_tempo[j]->sigma_s[i]
                            = CalculoSigma(malha[i], tempo[j]);
24                      linhazona[i] = CalculoLinhaZona(malha[i]);
25                      tb[i] = CalculoTb(malha[i]);
26                  }
27                  delta_p[j] = CalculoDeltaPressao(tempo[j]);
28                  resultados_ao_longo_do_tempo[j]->saveGrid(malha,
                       tempo[j]); /// os resultados são salvos em
                       arquivo .txt
29          }
30          std::cout << "Vetor de tb: " << std::endl;
31          print_vector(tb);
32          saveInFile(tempo, "tempo");
33          saveInFile(malha, "grid");
34          plot(malha, resultados_ao_longo_do_tempo[size_tempo - 1]->
               get_sigma_s(), "grid", "sigma s");
35          plot(tempo, delta_p, "tempo", "dp");
36  }
37
38  /// abaixo estao os calculos do simulador
39
40  double CSimuladorParticulas::CalculoSigma_a(double x, double t){
41          if (N == 1) {
42                  if (t < (x * porosidade / velocidade))
43                          return sigma_a0;
44                  else {
45                          return sigma_am + (sigma_a0 - sigma_am) *
                               exp(-parC * (t - x * porosidade /
                               velocidade));
46                  }
47          }
48          else {
49                  if (t < (x * porosidade / velocidade))
```

```
50                          return sigma_a0;
51                  else {
52                          return pow(sigma_am+pow(sigma_a0-sigma_am,
                               1- N) - parC *(1- N)*(t-x* porosidade /
                               velocidade), N / (1- N));
53                  }
54          }
55}
56
57double CSimuladorParticulas::CalculoDiffSigma_a(double x, double t)
        {
58          if (N == 1) {
59                  if (t < (x * porosidade / velocidade))
60                          return 0.0;
61                  else {
62                          return -parC *(sigma_a0 - sigma_am) * exp(-
                               parC * (t - x * porosidade / velocidade)
                               );
63                  }
64          }
65          else {
66                  if (t < (x * porosidade / velocidade))
67                          return 0.0;
68                  else {
69                          return -parC *pow( pow(sigma_a0 - sigma_am,
                                1 - N) - parC * (1 - N) * (t - x *
                               porosidade / velocidade), N / (1 - N));
70                  }
71          }
72}
73
74double CSimuladorParticulas::CalculoLinhaZona(double x) {
75          return porosidade * x / velocidade;
76}
77
78double CSimuladorParticulas::CalculoTb(double x) {
79          double cb = 0.0004; /// rocha
80          double linhaZona = CalculoLinhaZona(x);
81
82          double tb;
83          if (N == 1)
84                  tb = linhaZona + (1/ parC) * log((parC *(sigma_a0 -
```

```
                    sigma_am)*(1-exp(-lambdaPonte *x)))/(cb*
                    velocidade * lambdaPonte));
85          else
86                  tb = 2 * linhaZona + pow(sigma_a0 - sigma_am, 1 - N
                    ) / (parC * (1 - N)) - (1 / (parC * (1 - N))) *
                    pow(lambdaPonte * velocidade * cb / (parC * (1 -
                     exp(-x * lambdaPonte))), (1 - N) / N);
87
88          return tb;
89  }
90
91
92  double CSimuladorParticulas::CalculoDeltaPressao(double t) {
93          double integral = 0.0;
94          double const1 = viscosidade * velocidade * end_x /
               permeabilidade;
95          for (int i = 0; i < size_malha; i++) {
96                  integral += CalculoSigma(malha[i], t)*(malha[1]-
                    malha[0]);
97          }
98          return 1*(const1 + const1 * beta * integral);
99  }
100
101 double CSimuladorParticulas::CalculoSigma(double x, double t) {
102         if (N == 1)
103                 return CalculoSigma_N_igual_1(x, t);
104         else
105                 return CalculoSigma_N_diferente_1(x, t);
106 }
107
108 double CSimuladorParticulas::CalculoSigma_N_igual_1(double x,
        double t) {
109         double sigma = 0.0;
110         double tb = CalculoTb(x);
111         double C0 = x * porosidade / velocidade;
112         if (t < C0)
113                 sigma = 0.0;
114         else if (t >= C0 && t < tb) {
115                 sigma = funcao_sigma_n1(x, t);
116         }
117         else {
118                 double maxT = tb > C0 ? tb : C0;
```

```
119                 sigma = funcao_sigma_n1(x, maxT) + (sigma_a0 -
                        sigma_am) * (1 - exp(-lambdaAdesao * x)) * exp(C
                         * porosidade * x / velocidade) * (exp(-C * maxT
                        ) - exp(-C * t));
120         }
121         return sigma;
122 }
123
124 double CSimuladorParticulas::CalculoSigma_N_diferente_1(double x,
        double t) {
125         double sigma = 0.0;
126         double tb = CalculoTb(x);
127         double C0 = x * porosidade / velocidade;
128         if (t < C0)
129                 sigma = 0.0;
130         else if (t >= C0 && t < tb) {
131                 sigma = funcao_sigma_n1(x, t);
132         }
133         else {
134                 double maxT = tb > C0 ? tb : C0;
135                 sigma = funcao_sigma_n1(x, maxT) +(1-exp(-x*
                        lambdaAdesao))*(N-1)/(1-N)*pow(pow(sigma_a0-
                        sigma_am, 1-N) - C*(1-N)*(t-2*porosidade*x/
                        velocidade), 1/(1-N));
136         }
137         return sigma;
138 }
139
140 double CSimuladorParticulas::funcao_sigma_n1(double x, double t) {
141         Funcao_Sigma_n_1 funcao(velocidade, lambdaAdesao,
                lambdaPonte, cb, C, sigma_a0, sigma_am, porosidade, x);
142         MetodoIntegracaoNumerica1D* metodo = new MetodoSimpson(
                funcao);
143         double integral = metodo->Integrar(porosidade * x /
                velocidade, t, numPontosIntegral);
144         //return integral; - para validação do método de integração
                .
145         return C * (sigma_a0 - sigma_am) * ((1 - exp(-C * (t -
                porosidade * x / velocidade)) / C) + exp(-(lambdaAdesao
                + lambdaPonte - C * (x * porosidade / velocidade) * x))
                * integral);
146 }
```

```
147
148 double CSimuladorParticulas::funcao_sigma_n_dif_1(double x, double
    t) {
149         /// na linha abaixo, é criado a funcao relacionado a funcao
                sigma
150         Funcao_Sigma_n_diferente_1 funcao(velocidade, lambdaAdesao,
                lambdaPonte,  cb, C, sigma_a0, sigma_am, porosidade, x,
                N);
151
152         /// na linha abaixo, é criado o método de simpson, e é
               enviada a função criada acima
153         MetodoIntegracaoNumerica1D* metodo = new MetodoSimpson(
               funcao);
154         // na linha abaixo, é executado o método para integrar
155         double integral = metodo->Integrar(porosidade * x /
               velocidade, t, numPontosIntegral);
156         return C * integral;
157 }
158
159 void CSimuladorParticulas::printCSimuladorParticulas() {
160         printCParticulaFluido();
161         printCRocha();
162         std::cout << "\n---------------------" << std::endl;
163         std::cout << "Classe␣da␣simulacao:␣" << std::endl;
164         std::cout << "---------------------" << std::endl;
165
166         std::cout << "Grid␣dos␣tempos:" << std::endl;
167         print_vector(tempo);
168 }
169
170 void CSimuladorParticulas::saveInFile(std::vector<double> vector1,
    std::string name_vector1) {
171         std::ofstream outdata; //save data
172         outdata.open((name_vector1 + ".dat").c_str());
173         outdata << "#␣"<< name_vector1 << std::endl;
174         for (unsigned int i = 0; i < vector1.size(); i++)
175                 outdata << vector1[i] << std::endl;
176         outdata.close();
177 }
178
179 void CSimuladorParticulas::saveInFile(std::vector<double> vector1,
    std::vector<double> vector2, std::string name_vector1, std::
```

```
    string name_vector2){
180         if (vector1.size() != vector2.size()) {
181                 std::cout << "Nao␣foi␣possivel␣salvar␣os␣vetores,␣
                        por␣terem␣tamanhos␣distintos!"<<std::endl;
182                 return;
183         }
184
185         std::ofstream outdata; //save data
186         outdata.open((name_vector1+"_"+ name_vector2 + ".dat").
                c_str());
187         outdata << "#␣" << name_vector1 << "␣" << name_vector2 <<
                std::endl;
188         for (unsigned int i = 0; i < vector1.size(); i++)
189                 outdata << vector1[i] << "␣" << vector2[i] << std::
                        endl;
190         outdata.close();
191 }
192
193 void CSimuladorParticulas::print_vector(std::vector<double> vetor)
    {
194         std::cout << vetor[0]; /// este primeiro nao fica dentro do
                loop por causa do ' - '
195         for (unsigned int i = 1; i < vetor.size(); i++)
196                 std::cout << "␣-␣" << vetor[i];
197         std::cout << std::endl;
198 }
199
200 void CSimuladorParticulas::plot(std::vector<double> vector1, std::
    vector<double> vector2, std::string name_vector1, std::string
    name_vector2) {
201         saveInFile(vector1, vector2, name_vector1, name_vector2);
202         CGnuplot::plot((name_vector1 + "_" + name_vector2 + ".dat")
                .c_str(), name_vector1, name_vector2, (name_vector1 + "_
                " + name_vector2 + ".png").c_str());
203 }
```

Apresenta-se na listagem 1.9 o arquivo de implementação da classe Função1x1, da classe Funcao_Sigma_n_1 e da classe Funcao_Sigma_n_diferente_1.

Listing 1.9: Arquivo de implementação da classe funcao.cpp

```
1 #ifndef FUNCAO_HPP_
2 #define FUNCAO_HPP_
3
```

```
4 #include <math.h>
5 class Funcao1x1 {
6 public:
7     Funcao1x1() {}
8     Funcao1x1(double, double, double, double, double, double,
          double, double, double) {}
9     Funcao1x1(double, double, double, double, double, double,
          double, double, double, double) {}
10    virtual double operator()(double) = 0; // Funcao virtual pura
11 };
12
13 //classe herdeira da Funcao1x1
14 //passa os parametros da função e tornar a função que será
      integrada
15 class Funcao_Sigma_n_1 : public Funcao1x1 {
16 public:
17     Funcao_Sigma_n_1(double _U, double _lambda_s, double _lambda_b,
          double _cb, double _C, double _sigma_a0, double _sigma_am,
          double _phi, double _x):
18         U{ _U }, lambda_s{ _lambda_s }, lambda_b{ _lambda_b }, cb{
              _cb }, C{ _C }, sigma_a0{ _sigma_a0 }, sigma_am{
              _sigma_am }, phi{ _phi }, x{ _x }{}
19
20     virtual double operator()(double t) {
21         //return t*sin(t);
22         return (U * (lambda_s + lambda_b) * cb / (C * (sigma_a0 -
              sigma_am) * exp(C * phi * x / U)) - exp(-C * t)) / (1 -
              U * lambda_s * cb / (C * (sigma_a0 * sigma_am) * exp(C *
               phi * x / U)));
23     }
24 private:
25     double U, lambda_s, lambda_b, cb, C, sigma_a0, sigma_am, phi, x
          ;
26 };
27 //classe herdeira da Funcao1x1
28 //passa os parametros da função e tornar a função que será
      integrada
29 class Funcao_Sigma_n_diferente_1 : public Funcao1x1 {
30 public:
31     Funcao_Sigma_n_diferente_1(double _U, double _lambda_s, double
          _lambda_b, double _cb, double _C, double _sigma_a0, double
          _sigma_am, double _phi, double _x, double _n) :
```

```
32          U{ _U }, lambda_s{ _lambda_s }, lambda_b{ _lambda_b }, cb{
                _cb }, C{ _C }, sigma_a0{ _sigma_a0 }, sigma_am{
                _sigma_am }, phi{ _phi }, x{ _x }, n{ _n }{}
33
34      virtual double operator()(double t) {
35          double esquerda = pow(pow(sigma_a0 - sigma_am, 1 - n) - C *
                (1 - n) * (t - 2 * phi * x / U), n / (1 - n));
36          double dir_sup = exp(x*(lambda_b+lambda_s)) * (1 - U*(
                lambda_s+lambda_b)*cb/ esquerda);
37          double dir_inf = pow(1-lambda_s*U*cb/ esquerda, 1+lambda_b/
                lambda_s);
38          return esquerda * (1 - dir_sup / dir_inf);
39      }
40  private:
41      double U, lambda_s, lambda_b, cb, C, sigma_a0, sigma_am, phi, x
            , n;
42  };
43  #endif
```

Apresenta-se na listagem 1.10 o arquivo de implementação da classe MetodoSimpson.hpp.

Listing 1.10: Arquivo de implementação da classe MetodoSimpson.hpp

```
1  #ifndef METODOSIMPSON_HPP_
2  #define METODOSIMPSON_HPP_
3
4  #include "metodointegracaonumerica.hpp"
5  #include<iostream>
6
7  //herdeira da classe MetodoIntegracaoNumerica1D  - acessa o metodo
       de integrar
8  class MetodoSimpson : public MetodoIntegracaoNumerica1D {
9  public:
10     // construtor
11     MetodoSimpson(Funcao1x1& f_x_in) : MetodoIntegracaoNumerica1D(
           f_x_in) {}
12
13     // a função abaixo resolve a integral pelo método de SIMPSON. A
           função é usada aqui dentro
14     virtual double Integrar(double a, double b, size_t numPontos);
15 };
16 #endif
```

Apresenta-se na listagem 1.11 o arquivo de implementação da classe MetodoSimpson.cpp.

Listing 1.11: Arquivo de implementação da classe MetodoSimpson.cpp

```
#include "metodosimpson.hpp"

double MetodoSimpson::Integrar(double a, double b, size_t numPontos
    ) {
    // se numPontos for par, somo 1
    if (numPontos % 2 == 0) {
        std::cout << "\nWarning: numPontos passado eh par, somando
           1 para integrar pelo metodo de Simpson!"<<std::endl;
        numPontos += 1;
    }

    double h = (b-a)/(numPontos-1);

    // método de Simpson
    double dx = (b - a) / (numPontos - 1);

    double resultado = (funcao(a) + funcao(b)) * h / 3;

    for (size_t i = 1; i < numPontos - 1; i++) {
        if (i % 2 == 0) // se i for par, multiplico por 2
            resultado += funcao(a + i * dx) * 2 * h / 3;
        else // se if for impar, multiplico por 4
            resultado += funcao(a + i * dx) * 4 * h / 3;
    }

    return resultado;
}
```

Apresenta-se na listagem 1.12 o arquivo de implementação da classe MetodoIntegracao-Numerica1D.hpp.

Listing 1.12: Arquivo de implementação da classe metodointegracaonumerica.hpp

```
#ifndef METODOINTEGRACAONUMERICA_HPP_
#define METODOINTEGRACAONUMERICA_HPP_

#include <cstddef>

#include "funcao.hpp"

class MetodoIntegracaoNumerica1D {
```

```
9 public:
10     MetodoIntegracaoNumerica1D(Funcao1x1& fx_in) : funcao(fx_in) {}
11     virtual double Integrar(double a, double b, size_t numPontos) =
           0; // Virtual pura
12
13     Funcao1x1& funcao; // Funcao 1D a ser integrada
14 };
15 #endif
```

Apresenta-se na listagem 1.13 o arquivo de implementação da classe CGnuplot.hpp.

Listing 1.13: Arquivo de implementação da classe CGnuplot.cpp

```
1 #ifndef CGNUPLOT_HPP
2 #define CGNUPLOT_HPP
3
4 #include <vector>
5 #include <string>
6 #include <iostream>
7 #include <stdio.h>
8 #include <stdlib.h>
9
10 #ifdef _WIN32
11 #define GNUPLOT_NAME "C:\\Program\" \"Files\\gnuplot\\bin\\gnuplot␣
      -p"
12 #else
13 #define GNUPLOT_NAME "gnuplot"
14 #endif
15
16 class CGnuplot {
17 public:
18         CGnuplot() {}
19
20         static void plot(std::string name, std::string xlabel, std
              ::string ylabel, std::string saveName);
21         static void semilogx(std::string name, std::string xlabel,
              std::string ylabel, std::string saveName);
22         static void semilogy(std::string name, std::string xlabel,
              std::string ylabel, std::string saveName);
23 };
24 #endif
```

Apresenta-se na listagem 1.14 o arquivo de implementação da classe CGnuplot.cpp.

Listing 1.14: Arquivo de implementação da classe CGnuplot.cpp

```
1 #include "CGnuplot.hpp"
2
3 using namespace std;
4
5 void CGnuplot::plot(string name, string xlabel, string ylabel,
      string saveName) {
6 #ifdef _WIN32
7         FILE* pipe = _popen(GNUPLOT_NAME, "w");
8 #else
9         FILE* pipe = popen(GNUPLOT_NAME, "w");
10 #endif
11         fprintf(pipe, ("set␣xlabel␣'" + xlabel + "'\n").c_str());
12         fprintf(pipe, ("set␣ylabel␣'" + ylabel + "'\n").c_str());
13         fprintf(pipe, "unset␣key\n");
14         fprintf(pipe, ("plot␣'" + name + "'␣with␣linespoints␣
              linestyle␣1\n").c_str());
15         fprintf(pipe, "set␣term␣pngcairo\n");
16         fprintf(pipe, ("set␣output␣'" + saveName + "'\n").c_str());
17         fprintf(pipe, "replot\n");
18         fprintf(pipe, "set␣term␣win\n");
19         fflush(pipe);
20 }
21
22 void CGnuplot::semilogy(string name, string xlabel, string ylabel,
      string saveName) {
23 #ifdef _WIN32
24         FILE* pipe = _popen(GNUPLOT_NAME, "w");
25 #else
26         FILE* pipe = popen(GNUPLOT_NAME, "w");
27 #endif
28         fprintf(pipe, ("set␣xlabel␣'" + xlabel + "'\n").c_str());
29         fprintf(pipe, ("set␣ylabel␣'" + ylabel + "'\n").c_str());
30         fprintf(pipe, ("set␣logscale␣y\n"));
31         fprintf(pipe, "unset␣key\n");
32         fprintf(pipe, ("plot␣'" + name + "'␣with␣linespoints␣
              linestyle␣1\n").c_str());
33         fprintf(pipe, "set␣term␣pngcairo\n");
34         fprintf(pipe, ("set␣output␣'" + saveName + "'\n").c_str());
35         fprintf(pipe, "replot\n");
36         fprintf(pipe, "set␣term␣win\n");
37         fflush(pipe);
38 }
```

```
39
40 void CGnuplot::semilogx(string name, string xlabel, string ylabel,
       string saveName) {
41 #ifdef _WIN32
42         FILE* pipe = _popen(GNUPLOT_NAME, "w");
43 #else
44         FILE* pipe = popen(GNUPLOT_NAME, "w");
45 #endif
46         fprintf(pipe, ("set␣xlabel␣'" + xlabel + "'\n").c_str());
47         fprintf(pipe, ("set␣ylabel␣'" + ylabel + "'\n").c_str());
48         fprintf(pipe, ("set␣logscale␣x\n"));
49         fprintf(pipe, "unset␣key\n");
50         fprintf(pipe, ("plot␣'" + name + "'␣with␣linespoints␣
              linestyle␣1\n").c_str());
51         fprintf(pipe, "set␣term␣pngcairo\n");
52         fprintf(pipe, ("set␣output␣'" + saveName + "'\n").c_str());
53         fprintf(pipe, "replot\n");
54         fprintf(pipe, "set␣term␣win\n");
55         fflush(pipe);
56 }
```

Apresenta-se na listagem 1.15 o arquivo de implementação da classe main.cpp.

Listing 1.15: Arquivo de implementação da função `main()`.

```
1 #include<iostream>
2 #include<string>
3
4
5 #include "CSimuladorParticulas.cpp"
6
7 using namespace std;
8
9 int main() {
10
11         ////////////////////////////////////////////////////////
12         CSimuladorParticulas simulacao("particulaFluido.txt", "
              rocha.txt");
13         simulacao.printCSimuladorParticulas();
14         simulacao.run();
15         ////////////////////////////////////////////////////////
16         /*delete x_par_ptr;
17         delete y_par_ptr;
18         delete objSeno2Dptr;
```

```
19          delete metodo;*/
20 }
```