

```

1  /*
2   Projeto de Cálculo Numérico
3   Prof. Alexandre Roma (IME-USP)
4
5       IAG-USP
6       2015
7
8   Fábio Oliveira - 7978417
9   Jessé Stenico - 9051932
10
11  Resolução de E.D.O. pelos métodos:
12  Euler Explícito,
13  Euler Implícito e
14  Runge-Kutta Clássico.
15
16  Os resultados obtidos serão comparados com a solução analítica conhecida previamente.
17
18  Caso de calibragem: Decaimento Radioativo
19
20  y'=-Lambda*y
21  y=exp(-Lambda*t)
22
23  onde "Lambda" depende do material radioativo em questão.
24  */
25  #ifndef FUNCOES_H_INCLUDED
26  #define FUNCOES_H_INCLUDED
27
28  #define max_indice pow(2, TETO) + 1
29  static double ultimo_y_analitico;
30  static double ytf[3][TETO];
31  /*
32  Nesse caso,
33  ytf[0][j] - euler explicito
34  ytf[1][j] - euler implicito
35  ytf[2][j] - runge kutta
36  */
37
38  void verifica_info( ){
39      printf ( "Intervalo: [%f, %f]\n"
40              "Condicao de contorno: %f\n", t_MIN, t_MAX, y_contorno);
41  }
42
43  void inicializa_vetores(double n[ ], double h[ ]){
44
45      int k;
46
47      for(k=0; k<TETO; k++){
48          n[k]=pow(2, k+1);
49          h[k]=(t_MAX-t_MIN)/n[k];
50      }
51      for (k=0; k<TETO; k++){
52          printf("Passos: %4.f DeltaT: %4f\n", n[k], h[k]);
53      }
54  }
55
56  void resolve_analitico (double (*analitico_f)(double), double h[ ]){ /*SOLUÇÃO ANALÍTICA*/
57
58      FILE *saida;
59      double k;
60      double max_passo=(t_MAX-t_MIN)/h[TETO-1];
61      double y=y_contorno;
62      double t=t_MIN;
63
64      saida=fopen("y_analitico.txt", "w");
65
66      fprintf (saida, "%.10f %.10f\n", t, y);

```

```

67     for (k=1; k<=max_passo; k++){
68
69         t=t_MIN+k*h[TETO-1];
70         y=analitico_f(t);
71         fprintf (saida, "%.10f %.10f\n", t, y);
72
73     }
74     ultimo_y_analitico=y;
75     fclose(saida);
76 }
77
78 void teste_convergencia (FILE *saida, double ytf[][TETO], int metodo){
79
80     int k;
81     double a, b;
82
83     a=ytf[metodo][0];
84     for (k=1; k<TETO; k++){
85
86         b=ytf[metodo][k];
87         fprintf(saida, "%d %.10f\n", k, fabs( (a-ultimo_y_analitico) / (b-ultimo_y_analitico) ));
88         a=b;
89
90     }
91 }
92
93 void euler_explicito (double (*fxy)(double, double), double n[ ], double h[ ]){
94
95     FILE *saida;
96     FILE *convergencia;
97
98     char arq[15];
99     int k;
100    double y, t;
101    double passo;
102
103
104    for (k=0; k<TETO; k++){
105
106        sprintf(arq, "exp%d.txt", (int)pow(2, k+1)); /*UM ARQUIVO DIFERENTE PARA CADA QUANTIDADE DE n
PASSOS*/
107        saida=fopen(arq, "w");
108
109        y=y_contorno;
110        t=t_MIN;
111        fprintf (saida, "%.10f %.10f\n", t, y);
112        for(passo=1; passo<=n[k]; passo++){
113
114            t=t_MIN+passo*h[k];
115            y=y+h[k]*fxy(t, y);
116            fprintf (saida, "%.10f %.10f\n", t, y);
117
118        }
119        fclose(saida);
120
121        ytf[0][k]=y; /*ARMAZENA VALORES DE y(tf) EM UM VETOR SEPARADO*/
122    } /*ÚTIL PARA TESTES DE CONVERGENCIA*/
123
124
125    convergencia=fopen("conv_exp.txt", "w");
126    teste_convergencia(convergencia, ytf, 0);
127    fclose(convergencia);
128 }
129
130 void euler_implicito (double (*fxy)(double, double), double (*df)(double, double), double n[ ], double h[
])
131 }
```

```

131
132     FILE *saida;
133     FILE *convergenca;
134
135     char arq[15];
136
137     double erro;
138     int passo, k, iteracao;
139
140     double y_implicito;
141     double t;
142     double chute_inicial=(t_MAX-t_MIN)/2.0;
143     double y1, y2;
144
145     for (k=0; k<TETO; k++){ /*PERCORRE TODO O VETOR DE INCREMENTOS*/
146         sprintf(arq, "imp%d.txt", (int)pow(2,k+1));
147         saida=fopen(arq, "w");
148
149         y1=chute_inicial;
150         y_implicito=y_contorno;
151         t=t_MIN;
152         fprintf (saida, "%.10f %.10f\n", t, y_implicito);
153
154         for(passo=1; passo<=n[k]; passo++){
155
156             t=t_MIN+passo*h[k];
157             erro=1/EPSILON; /*VALOR SEGURAMENTE MAIOR QUE EPSILON QUALQUER QUE SEJA ELE*/
158             iteracao=1;
159             while (erro>EPSILON){ /*METODO DE NEWTON*/
160
161                 if(iteracao>MAXIMO_ITERACAO){
162                     printf ("Newton-Raphson: Atingiu maximo de iteracoes determinado!\n");
163                     break;
164                 }
165
166                 y2=y1-(y1-y_implicito-h[k]*fxy(t, y1))/(1-h[k]*df(t, y1));
167                 erro=fabs(y2-y1);
168                 y1=y2;
169                 iteracao++;
170             }
171             y_implicito=y1;
172
173             fprintf (saida, "%.10f %.10f\n", t, y_implicito);
174         }
175         fclose(saida);
176         ytf[1][k]=y_implicito;
177     }
178
179     convergenca=fopen("conv_imp.txt", "w");
180     teste_convergenca(convergenca, ytf, 1);
181     fclose(convergenca);
182
183 }
184
185 void runge_kutta (double (*fxy)(double, double), double n[ ], double h[ ]){
186
187     FILE *saida;
188     FILE *convergenca;
189
190     char arq[15];
191
192     int k;
193
194     double passo;
195     double K_1, K_2, K_3, K_4;
196     double y_rk;

```

```

197     double t;
198
199     for (k=0; k<TETO; k++){
200         sprintf(arq, "rk%d.txt", (int)pow(2,k+1));
201         saida=fopen(arq, "w");
202
203         y_rk=y_contorno;
204         t=t_MIN;
205         fprintf (saida, "%.10f %.10f\n", t, y_rk);
206
207         for(passo=1; passo<=n[k]; passo++){
208
209             K_1=fxy(t, y_rk);
210
211             K_2=fxy(t+0.5*h[k], y_rk+(0.5*h[k]*K_1));
212
213             K_3=fxy(t+0.5*h[k], y_rk+(0.5*h[k]*K_2));
214
215             K_4=fxy(t+h[k], y_rk+(h[k]*K_3));
216
217             y_rk=y_rk+(1.0/6.0)*h[k]*(K_1+K_2+K_2+K_3+K_3+K_4);
218
219             t=t_MIN+passo*h[k];
220
221             fprintf (saida, "%.10f %.10f\n", t, y_rk);
222
223         }
224         fclose(saida);
225         ytf[2][k]=y_rk;
226
227     }
228
229     convergencia=fopen("conv_rk.txt", "w");
230     teste_convergencia(convergencia, ytf, 2);
231     fclose(convergencia);
232 }
233
234 #endif

```