

Classwork: Airy

Student: Gabriele Cembalo

December 11, 2025

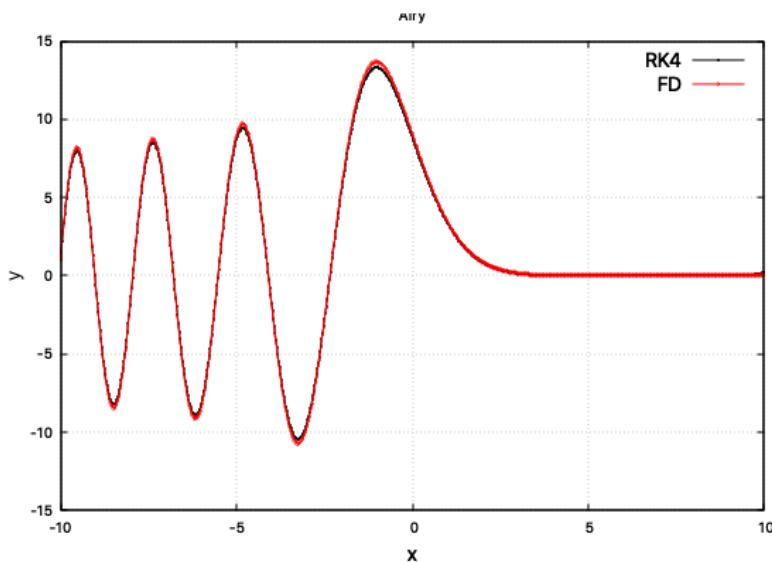


Figure 1:

```
1 //  
2  
3 #include <iostream>  
4 #include <iomanip>  
5 #include <cmath>  
6 #include <fstream>  
7  
8 #define NMAX_EQ 64 // numero massimo di eq (sicurezza)  
9  
10 using namespace std;  
11  
12 void RHSFunc(double, double *, double *);  
13 double Residual(double);  
14 void RK4Step(double, double *, void (*)(double, double *, double *), double, int);  
15 int secant_method(double (*F)(double), double, double, double &, int &);  
16 int bisection(double (*F)(double), double, double, double &, int &);  
17 void TridiagSolver(double *, double *, double *, double *, int);  
18 void PrintVector(double *, int);  
19  
20 int main(){  
21  
22     ofstream fdata;  
23     fdata.open("airy.dat"); // file per le soluzioni  
24  
25     cout << setiosflags ( ios::scientific );  
26     cout << setprecision ( 4 );  
27  
28     fdata << setiosflags ( ios::scientific );
```

```

29   fdata << setprecision ( 10 );
30
31   cout << endl;
32
33   int neq = 2; // numero equazioni
34   int nstep = 800; // numero punti
35   double tol = 1.e-8;
36
37   double x; // variabile raggio
38   double x0 = -10.0;
39   double xf = 10.0;
40   double dx = fabs(xf - x0) / nstep; // step fisso
41
42   double res; // variabile residuo
43   double szero; // variabile per lo zero del residuo
44   int l = 0; // variabile per le iterazioni di bisezione
45   // (inutile per l'esercizio)
46
47   // variabile derivata di y
48   double s; // guess, derivata di y in x
49   double s0 = 10.0 , sf = 40.0;
50   double ds = fabs( s0 - sf )/(double)nstep;
51
52   // plotto il residuo
53   for( int i = 0 ; i < nstep ; i++ ){
54
55     s = s0 + i*ds;
56     res = Residual(s);
57
58     fdata << s << " " << res << endl;
59   }
60
61   fdata << endl << endl;
62
63   // uso bisezione per trovare lo zero del residuo e quindi il valore di s
64   double status;
65   status = bisection(Residual, s0, sf, tol, szero, l);
66
67   // aggiungo un controllo sullo zero del residuo
68   if (status == 0){
69     cout << "y'(-10) = " << szero
70     << "; iterazioni (Bisection) = " << l << endl;
71   }else{
72     cout << "No solution!" << endl;
73   }
74
75
76   // uso secante per trovare lo zero del residuo e quindi il valore di s
77   status = secant_method(Residual, s0, sf, tol, szero, l);
78
79   // aggiungo un controllo sullo zero del residuo
80   if (status == 0){
81     cout << "y'(-10) = " << szero
82     << "; iterazioni (Secant) = " << l << endl;
83   }else{
84     cout << "No solution!" << endl;
85   }
86
87   // definisco l'array delle soluzioni e imposto le condizioni iniziali
88   double Y[neq];
89   Y[0] = 1;
90   Y[1] = szero;
91
92   x = x0;
93
94   // plotto la condizione iniziale
95   fdata << x << " " << Y[0] << " " << Y[1] << endl;
96
97   // risolvo le equazioni del moto usando RK a 4 step

```

```

98     for( int i = 0 ; i < nstep ; i++ ){
99
100         RK4Step(x, Y, RHSFunc, dx, neq); // risolvo la ODE
101         x += dx;
102
103         // stampo nel file i dati
104         fdata << x << " " << Y[0] << " " << Y[1] << endl;
105
106     }
107
108     fdata << endl << endl;
109
110     cout << "\nSoluzione salvata in airy.dat\n" << endl;
111
112
113     // Implemento il finite difference method
114     int n = nstep + 1; // punti griglia
115
116     // definisco i vettori degli elementi sopra e sotto la diagonale
117     double *a, *b, *c, *r, *y;
118
119     a = new double [n];
120     b = new double [n];
121     c = new double [n];
122     r = new double [n];
123     y = new double [n];
124
125     double alpha , beta;
126     alpha = 1.0;
127     beta = 0.0;
128
129     double h; // incremento
130     h = fabs( xf - x0 )/(double)( n - 1 );
131
132     // condizioni al bordo
133     y[0] = alpha;
134     y[n-1] = beta;
135
136     // riempio i vettori a, b, c ed r
137     for( int i = 0 ; i < n ; i++ ){
138
139         x = x0 + i*h;
140
141         a[i] = 1.;
142         b[i] = -2. - h*h*x;
143         c[i] = 1. ;
144
145         r[i] = 0.0;
146
147     }
148
149     // metto le condizioni iniziali nel primo e ultimo elemento di r
150     r[1] -= y[0];
151     r[n-2] -= y[n-1];
152
153     // richiamo il TridiagSolver per trovare il vettore delle soluzioni
154     // schiftiamo di 1 l'argomento, poiche' l'indice 0 lo fissiamo con la
155     // condizione al bordo e quindi il primo elemento sara' y2
156     TridiagSolver(a+1, b+1, y+1, r+1, c+1, n-2);
157     // PrintVector(y, n);
158
159     // stampo il vettore nel file
160     for ( int i = 0 ; i < n ; i++ ){
161
162         x = x0 + i*h;
163
164         fdata << x << " " << y[i] << endl;
165
166     }

```

```

167     fdata.close ();
168
169     // pulisco
170     delete[] a;
171     delete[] b;
172     delete[] c;
173     delete[] r;
174     delete[] y;
175
176     return 0;
177 }
178
179 // ----- Funzioni -----
180
181 // definisco il Right-Hand-Side-Function (e' problem dependent).
182 // Gli do in input t e il puntatore ad Y e in uscita (tramite il puntatore)
183 // mi faccio dare R
184 void RHSFunc(double x, double *Y, double *R){
185
186     R[0] = Y[1];
187     R[1] = Y[0]*x;
188
189 }
190
191 // creo la funzione residuo
192 // sara' la funzione che diamo alla funzione per la ricerca degli zeri
193 // e' problem dependent.
194 // gli do in input la guess sulla derivata
195 double Residual(double s){
196
197     // ricopio esattamente il main precedente per trovare la soluzione
198     int neq = 2; // numero equazioni
199     int nstep = 800; // numero punti
200     double tol = 1.e-8;
201
202     double x; // variabile
203     double x0 = -10.0;
204     double xf = 10.0;
205     double dx = fabs(xf - x0) / nstep; // step fisso
206
207     // definisco l'array delle soluzioni e imposto le condizioni iniziali
208     double Y[neq];
209     Y[0] = 1.0;
210     Y[1] = s;
211
212     x = x0; // setto il punto di integrazione iniziale
213
214     // risolvo le equazioni del moto usando RK a 4 step
215     for( int i = 0 ; i < nstep ; i++ ){
216
217         RK4Step(x, Y, RHSFunc, dx, neq); // risolvo la ODE
218         x += dx;
219
220     }
221
222     // calcolo e ritorno il residuo (valore atteso e' 0.0)
223     return Y[0];
224
225 }
226
227 // implemento il metodo Runge-Kutta del quarto ordine.
228 // gli do in input la variabile di integrazione, il puntatore alle soluzioni,
229 // il puntatore alla funzione del Right-Hand-Side-Function, l'incremento e
230 // l'ordine della ODE.
231 void RK4Step(double t, double *Y, void (*RHSFunc)(double t, double *Y, double *R),
232             double h, int neq){

```

```

236 // definisco i vettori per gli step intermedi
237 double Y1[NMAX_EQ], k1[NMAX_EQ], k2[NMAX_EQ], k3[NMAX_EQ], k4[NMAX_EQ];
238
239 RHSFunc(t,Y,k1); // calcolo k1 con il RSH con t_n e Y_n
240
241 // scrivo il ciclo per determinare Y_n + k1*h/2
242 for( int i = 0 ; i < neq ; i++ ){
243
244     Y1[i] = Y[i] + 0.5*h*k1[i];
245
246 }
247
248 RHSFunc(t+0.5*h,Y1,k2); // calcolo k2 con il RSH con t_n+h/2 e Y_n+k1*h/2
249
250 // scrivo il ciclo per calcolare Y_{n+1} = Y_n + k2*h/2
251 for (int i = 0 ; i < neq ; i++) {
252
253     Y1[i] = Y[i] + h*k2[i]*0.5;
254
255 }
256
257 RHSFunc(t+0.5*h,Y1,k3); // calcolo k3 con il RSH con t_n+h/2 e Y_n+k2*h/2
258
259 // scrivo il ciclo per calcolare Y_{n+1} = Y_n + k3*h
260 for (int i = 0 ; i < neq ; i++) {
261
262     Y1[i] = Y[i] + h*k3[i];
263
264 }
265
266 RHSFunc(t+h,Y1,k4); // calcolo k4 con il RSH con t_n+h e Y_n+k3*h
267
268 // scrivo il ciclo per calcolare
269 // Y_{n+1} = Y_n + h/6 * ( k1 + 2*k2 + 2*k3 + k4 )
270 for (int i = 0 ; i < neq ; i++) {
271
272     Y[i] += h * ( k1[i] + 2.0*k2[i] + 2.0*k3[i] + k4[i] ) / 6.0;
273
274 }
275
276 }
277
278 // metodo della secante
279 // gli do in input la funzione, gli estremi a e b, la tolleranza su x,
280 // uno zero per riferimento e il numero di iterazioni
281 // int secant_method(double (*F)(double), double a, double b, double tol,
282 //                     double &zero, int &l){
283
284 // definisco le variabili che mi servono per tenere traccia delle
285 // varie iterazioni di x
286 double xk1 = a, xk = b, xk2 = xk + 1; // dove uso xk come x_k, xk1 come
287 // x_{k-1} e xk2 come x_{k+1} ; inizializzo gli zeri sugli estremi
288 // dell'intervallo in cui ricaviamo la retta
289 int n = 0; // variabile per contare
290 // una variabile di controllo per vedere di quanto miglioriamo la guess
291 double xp = 0;
292
293 // definisco le variabili della funzione valutata
294 double fa = F(a);
295 double fb = F(b);
296 double fzk;
297 double fxk1;
298
299 // metto i controlli di non avere gia' uno zero
300 if( fa == 0.0 ){
301
302     zero = a;
303
304 }
```

```

305     // creo l'output delle iterazioni
306     cout << "(Secant) # = " << n
307         << "(l'estremo " << a << " e' gia' lo zero)" << endl;
308
309     l = n; // sono le iterazioni
310     return 0;
311
312 }
313 else if( fb == 0.0 ){
314
315     zero = b;
316
317     // creo l'output delle iterazioni
318     cout << "(Secant) # = " << n
319         << "(l'estremo " << b << " e' gia' lo zero)" << endl;
320
321     l = n; // sono le iterazioni
322     return 0;
323
324 }
325 else{
326
327     // metto nel ciclo la condizione sia sulla tolleranza che sul numero
328     // di cicli (messo dopo)
329     while( fabs( xk2 - xp ) > tol ){
330
331         n++;
332
333         if( n == 100 ){
334
335             cout << "(Secant) Troppe iterazioni." << endl;
336
337             l = n; // sono le iterazioni
338             return 0;
339
340         }
341
342         xp = xk2;
343         ffxk = F(xk);
344         ffxk1 = F(xk1);
345
346         // calcolo lo zero x_{k+1}
347         xk2 = xk - ffxk*( xk - xk1 )/( ffxk - ffxk1 );
348
349         xk1 = xk;
350         xk = xk2;
351
352         // creo l'output voluto (esercizio froot.cpp)
353         //cout << "n = " << n << "; [a,b] = [" << xk1 << ", " << xk
354         //           << "]; x0 = " << xk2 << "; Deltax = " << fabs(xk2-xk1)
355         //           << "; f(x0) = " << F(xk2) << endl;
356
357     }
358
359     // creo l'output delle iterazioni
360     //cout << "(Secant) # = " << n << endl;
361
362     l = n; // sono le iterazioni
363     zero = xk2;
364     return 0;
365
366 }
367
368 }
369
370 // metodo della bisezione
371 // gli do in input la funzione, gli estremi a e b, la tolleranza su x,
372 // uno zero per riferimento e il numero di iterazioni
373 int bisection(double (*F)(double), double a, double b, double tol,

```

```

374         double &zero, int &l){
375
376     double x; // la mia guess dello zero che aggiorno ad ogni iterazione
377     int n; // la variabile che mi permette di contare le iterazioni
378
379     // definisco le variabili della funzione valutata
380     double fa = F(a);
381     double fb = F(b);
382     double fx;
383
384     // metto i controlli di non avere gia' uno zero
385     if( fa == 0.0 ){
386
387         zero = a;
388
389         // creo l'output delle iterazioni
390         cout << "(Bisection) # = " << n
391             << " (l'estremo " << a << " e' gia' lo zero)"<< endl;
392
393         l = n;
394         return 0;
395     }
396     else if( fb == 0.0 ){
397
398         zero = b;
399
400         // creo l'output delle iterazioni
401         cout << "(Bisection) # = " << n
402             << "(l'estremo " << b << " e' gia' lo zero)"<< endl;
403
404         l = n;
405         return 0;
406     }
407     else{
408
409         while( fabs(a-b) > tol ){
410
411             n++;
412
413             // metto il controllo sul numero di iterazioni
414             if( n == 100 ){
415
416                 cout << "(Bisection) Troppe iterazioni." << endl;
417
418                 l = n;
419                 return 0;
420             }
421
422             // calcolo la prima stima dello zero
423             x = ( a+b ) * 0.5;
424
425             // definisco le variabili delle funzioni valutate
426             fa = F(a);
427             fb = F(b);
428             fx = F(x);
429
430             // controllo se e' uno zero
431             if( fx == 0 ){
432
433                 zero = x;
434
435                 // creo l'output delle iterazioni
436                 //cout << "(Bisection) # = " << n << endl;
437
438                 l = n;
439                 return 0;
440             }
441         }
442     }
443 }

```

```

443
444     }
445     // controllo dove si trova x rispetto gli estremi a e b
446     else if( fa*fx < 0 ){
447
448         b = x;
449
450     }
451     else if ( fa*fx > 0 ){
452
453         a = x;
454
455     }
456
457     // creo l'output voluto (esercizio froot.cpp)
458     //cout << "n = " << n << ";      [a,b] = [" << a << ", " << b
459     //           << "];      xm = " << x << ";      Deltax = " << fabs(a-b)
460     //           << ";\n      f(xm) = " << F(x) << endl;
461
462 }
463
464     // creo l'output delle iterazioni
465     //cout << "(Bisection) # = " << n << endl;
466
467     l = n; // sono le iterazioni
468     zero = x;
469     return 0;
470
471 }
472
473 }
474
475 // Implemento la funzione di TridiagSolver
476 // gli do in input i vettori contenenti rispettivamente: gli elementi sotto
477 // la diagonale, i termini noti, le soluzioni, r, gli elementi sopra la diagonale
478 // e la dimensione
479 void TridiagSolver(double *a, double *b, double **x, double *r, double *c, int n){
480
481     // definisco i vettori h e p
482     double *h = new double [n];
483     double *p = new double [n];
484
485     // calcolo gli elementi h[n] e p[n] per il metodo Tridiag
486     // separo i termini patologici
487     h[0] = c[0]/b[0];
488     p[0] = r[0]/b[0];
489
490     for( int i = 1 ; i < n ; i++ ){
491
492         h[i] = c[i] / ( b[i] - a[i]*h[i-1] );
493
494         p[i] = ( r[i] - a[i]*p[i-1] ) / ( b[i] - a[i]*h[i-1] );
495
496     }
497
498     // applico il metodo di risoluzione
499     x[n-1] = p[n-1]; // termine patologico non avendo definito x[n+1]
500
501     // applichiamo back-substitution
502     for( int i = n-1 ; i >= 0 ; i-- ){
503
504         x[i] = p[i] - h[i]*x[i+1];
505
506     }
507
508     // pulisco
509     delete[] h;
510     delete[] p;
511

```

```
512 }
513 // implemento la funzione per stampare un vettore (dinamico) di n dimensioni
514 void PrintVector(double *v, int n){
515
516     cout << fixed << setprecision(4);
517
518     for(int j = 0 ; j < n ; j++ ){
519         cout << setw(10) << right << v[j] << endl;
520     }
521
522 }
523
524 }
```