Classwork: Airy

Student: Gabriele Cembalo

December 12, 2025

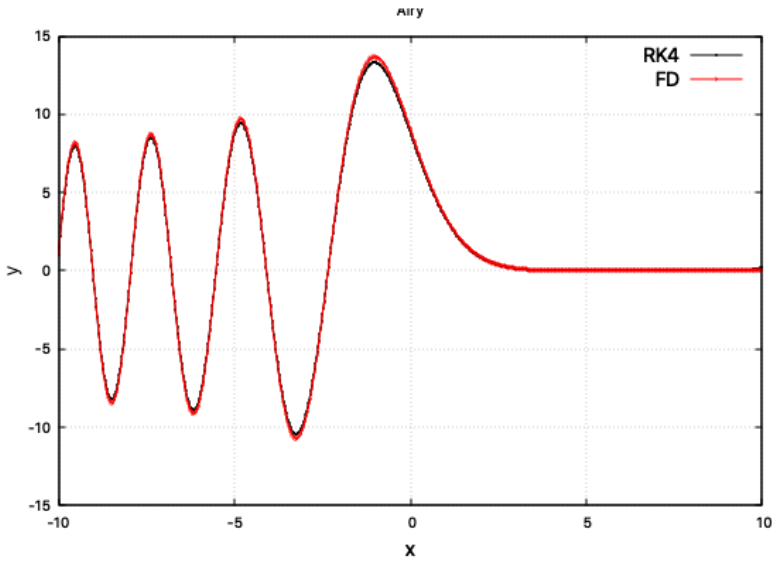

Figure 1:

```cpp
//

#include <iostream>
#include <iomanip>
#include <cmath>
#include <fstream>

#define NMAX_EQ 64 // numero massimo di eq (sicurezza)

using namespace std;

void RHSFunc(double, double *, double *);
double Residual(double);
void RK4Step(double, double *, void (*)(double, double *, double *), double, int);
int secant_method(double (*F)(double), double, double, double, double &, int &);
int bisection(double (*F)(double), double, double, double, double &, int &);
void TridiagSolver(double *, double *, double *, double *, double *, int);
void PrintVector(double *, int);

int main(){

    ofstream fdata;
    fdata.open("airy.dat"); // file per le soluzioni

    cout << setiosflags ( ios::scientific );
    cout << setprecision ( 4 );

    fdata << setiosflags ( ios::scientific );
```

```
    fdata << setprecision ( 10 );

    cout << endl;

    int neq = 2; // numero equazioni
    int nstep = 800; // numero punti
    double tol = 1.e-8;

    double x; // variabile raggio
    double x0 = -10.0;
    double xf = 10.0;
    double dx = fabs(xf - x0) / nstep; // step fisso

    double res; // variabile residuo
    double szero; // variabile per lo zero del residuo
    int l = 0; // variaible per le iterazioni di bisezione
               // (inutile per l'esercizio)

    // variabile derivata di y
    double s; // guess, derivata di y in x
    double s0 = 10.0 , sf = 40.0;
    double ds = fabs( s0 - sf )/(double)nstep;

    // plotto il residuo
    for( int i = 0 ; i < nstep ; i++ ){

        s = s0 + i*ds;
        res = Residual(s);

        fdata << s << "  " << res << endl;

    }

    fdata << endl << endl;

    // uso bisezione per trovare lo zero del residuo e quindi il valore di s
    double status;
    status = bisection(Residual, s0, sf, tol, szero, l);

    // aggiungo un controllo sullo zero del residuo
    if (status == 0){
        cout << "y'(-10) = " << szero
             << ";   iterazioni (Bisection) = " << l << endl;
    }else{
        cout << "No solution!" << endl;
    }

    // uso secante per trovare lo zero del residuo e quindi il valore di s
    status = secant_method(Residual, s0, sf, tol, szero, l);

    // aggiungo un controllo sullo zero del residuo
    if (status == 0){
        cout << "y'(-10) = " << szero
             << ";   iterazioni (Secant) = " << l << endl;
    }else{
        cout << "No solution!" << endl;
    }

    // definisco l'array delle soluzioni e imposto le condizioni iniziali
    double Y[neq];
    Y[0] = 1;
    Y[1] = szero;

    x = x0;

    // plotto la condizione iniziale
    fdata << x << "  " << Y[0] << "  " << Y[1] << endl;

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

```
236
237     // definisco i vettori per gli step intermedi
238     double Y1[NMAX_EQ], k1[NMAX_EQ], k2[NMAX_EQ], k3[NMAX_EQ], k4[NMAX_EQ];
239
240     RHSFunc(t,Y,k1); // calcolo k1 con il RSH con t_n e Y_n
241
242     // scrivo il ciclo per determinare Y_n + k1*h/2
243     for( int i = 0 ; i < neq ; i++ ){
244
245         Y1[i] = Y[i] + 0.5*h*k1[i];
246
247     }
248
249     RHSFunc(t+0.5*h,Y1,k2); // calcolo k2 con il RSH con t_n+h/2 e Y_n+k1*h/2
250
251     // scrivo il ciclo per calcolare Y_{n+1} = Y_n + k2*h/2
252     for (int i = 0 ; i < neq ; i++){
253
254         Y1[i] = Y[i] + h*k2[i]*0.5;
255
256     }
257
258     RHSFunc(t+0.5*h,Y1,k3); // calcolo k3 con il RSH con t_n+h/2 e Y_n+k2*h/2
259
260     // scrivo il ciclo per calcolare Y_{n+1} = Y_n + k3*h
261     for (int i = 0 ; i < neq ; i++){
262
263         Y1[i] = Y[i] + h*k3[i];
264
265     }
266
267     RHSFunc(t+h,Y1,k4); // calcolo k4 con il RSH con t_n+h e Y_n+k3*h
268
269     // scrivo il ciclo per calcolare
270     // Y_{n+1} = Y_n + h/6 * ( k1 + 2*k2 + 2*k3 + k4 )
271     for (int i = 0 ; i < neq ; i++){
272
273         Y[i] += h * ( k1[i] + 2.0*k2[i] + 2.0*k3[i] + k4[i] ) / 6.0;
274
275     }
276
277 }
278
279 // metodo della secante
280 // gli do in input la funzione, gli estremi a e b, la tolleranza su x,
281 // uno zero per riferimento e il numero di iterazioni
282 int secant_method(double (*F)(double), double a, double b, double tol,
283                   double &zero, int &l){
284
285     // definisco le variabili che mi servono per tenere traccia delle
286     // varie iterazioni di x
287     double xk1 = a, xk = b, xk2 = xk + 1; // dove uso xk come x_k, xk1 come
288     // x_{k-1} e xk2 come x_{k+1} ; inizializzo gli zeri sugli estremi
289     // dell'intervallo in cui ricaviamo la retta
290     int n = 0; // variabile per contare
291     // una variabile di controllo per vedere di quanto miglioriamo la guess
292     double xp = 0;
293
294     // definisco le variabili della funzione valutata
295     double fa = F(a);
296     double fb = F(b);
297     double fxk;
298     double fxk1;
299
300     // metto i controlli di non avere gia' uno zero
301     if( fa == 0.0 ){
302
303         zero = a;
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             fxk = F(xk);
344             fxk1 = F(xk1);
345 
346             // calcolo lo zero x_{k+1}
347             xk2 = xk - fxk*( xk - xk1 )/( fxk - fxk1 );
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 = 0; // 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     }
397     else if( fb == 0.0 ){
398
399         zero = b;
400
401         // creo l'output delle iterazioni
402         cout << "(Bisection) # = " << n
403              << "(l'estremo " << b << " e' gia' lo zero)"<< endl;
404
405         l = n;
406         return 0;
407
408     }
409     else{
410
411         while( fabs(a-b) > tol ){
412
413             n++;
414
415             // metto il controllo sul numero di iterazioni
416             if( n == 100 ){
417
418                 cout << "(Bisection) Troppe iterazioni." << endl;
419
420                 l = n;
421                 return 0;
422
423             }
424
425             // calcolo la prima stima dello zero
426             x = ( a+b ) * 0.5;
427
428             // definisco le variabili delle funzioni valutate
429             fa = F(a);
430             fb = F(b);
431             fx = F(x);
432
433             // controllo se e' uno zero
434             if( fx == 0 ){
435
436                 zero = x;
437
438                 // creo l'output delle iterazioni
439                 //cout << "(Bisection) # = " << n << endl;
440
441                 l = n;
442                 return 0;
```

```
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             //       << ";   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
514 // implemento la funzione per stampare un vettore (dinamico) di n dimensioni
515 void PrintVector(double *v, int n){
516
517     cout << fixed << setprecision(4);
518
519     for(int j = 0 ; j < n ; j++ ){
520
521         cout << setw(10) << right << v[j] << endl;
522
523     }
524
525 }
```