7 Appendix

7.1 Lane-Emden

```
/* Lane-Emden equation
  Run on core i5 with g++ 5.4.0
  Gianfranco Grillo 12/10/2016
*/

#include <cstdlib>
#include <cmath>

#include <iostream>
#include <fstream>
#include <iomanip>
#include "arrayt.hpp"
using namespace std;

int main() {
    int n = 2, i;
    double h, hold, x, N, pi = 4.0 * atan(1.0);
    arrayt<double> initvar(n), finalvar(n);
    void autosteprk4(arrayt<double>&, arrayt<double>&, int, double&, double, void(
       arrayt<double>&, double, arrayt<double>&, double&));
    void laneemden(arrayt<double>&, double, arrayt<double>&, double&);
    ofstream fp;
    x = 0.01;
    initvar(0) = 1-x*x/6.0+x*x*x*x/40.0-19.0*pow(x,6.0)/5040.0;
    initvar(1) = -x/3.0 + x*x*x/10.0-19.0*pow(x,5.0)/840.0;
    h = 1e-3;
    fp.open("lanemden.dat");
    while (x < 10.0) {
        hold = h;
        autosteprk4(initvar, finalvar, n, h, x, laneemden);
        fp << finalvar(0) << setw(15) << finalvar(1) << setw(15) << x << endl;
        if (finalvar(0) < 0.0) break;
        if (h > hold || h == hold) {
            for (i = 0; i < n; i++) { initvar(i) = finalvar(i); }
            x = x + hold;
        }
    }
    N = pow(4*pi,1.0/3.0)/4.0*pow(-x*x*initvar(1),-2.0/3.0);
    cout << "xi1 = " << x << setw(15) << "gamma1 = " << initvar(1) << setw(15) << "
       N = " << N << endl;
    fp.close();
    return (EXIT_SUCCESS);
}

void autosteprk4(arrayt<double>& initvar, arrayt<double>& highfinalvar, int n,
   double& h,
                                 double t0, void functions(arrayt<double>&, double,
                                     arrayt<double>&, double&)) {
```

```
    int i;
    arrayt<double> scale(n), error(n), lowfinalvar(n), k1(n), k2(n), k3(n), k4(
       n), k5(n), k6(n), k7(n), scalek(n), temp(n);
double maxerror = 1e-10, deltamax;
    double maximum(arrayt<double>&);
    double c2 = 0.2, c3 = 0.3, c4 = 0.8, c5 = 8.0/9.0, c6 = 1.0, c7 = 1.0;
    double a21 = 0.2, a31 = 3.0/40.0, a32 = 9.0/40.0, a41 = 44.0/45.0, a42 =
       -56.0/15.0, a43 = 32.0/9.0;
    double a51 = 19372.0/6561.0, a52 = -25360.0/2187.0, a53 = 64448.0/6561.0,
       a54 = -212.0/729.0;
    double a61 = 9017.0/3168.0, a62 = -355.0/33.0, a63 = 46732.0/5247.0, a64 =
       49.0/176.0, a65 = -5103.0/18656.0;
    double b1 = 35.0/384.0, b3 = 500.0/1113.0, b4 = 125.0/192.0, b5 =
       -2187.0/6784.0, b6 = 11.0/84.0;
    double d1 = 5179.0/57600.0, d3 = 7571.0/16695.0, d4 = 393.0/640.0, d5 =
       -92097.0/339200.0, d6 = 187.0/2100.0, d7 = 1.0/40.0;
    functions(initvar, t0, k1, h); // generate k1
    for (i = 0; i < n; i++) {
            temp(i) = initvar(i) + a21*k1(i);
    }
    functions(temp, t0 + c2*h, k2, h);
    for (i = 0; i < n; i++) {
            temp(i) = initvar(i) + a31*k1(i) + a32*k2(i);
    }
    functions(temp, t0 + c3*h, k3, h);
    for (i = 0; i < n; i++) {
            temp(i) = initvar(i) + a41*k1(i) + a42*k2(i) + a43*k3(i);
    }
    functions(temp, t0 + c4*h, k4, h);
    for (i = 0; i < n; i++) {
            temp(i) = initvar(i) + a51*k1(i) + a52*k2(i) + a53*k3(i) + a54*k4(i
               );
    }
    functions(temp, t0 + c5*h, k5, h);
    for (i = 0; i < n; i++) {
            temp(i) = initvar(i) + a61*k1(i) + a62*k2(i) + a63*k3(i) + a64*k4(i
               ) + a65*k5(i);
    }
    functions(temp, t0 + c6*h, k6, h);
    for (i = 0; i < n; i++) {
            highfinalvar(i) = initvar(i) + b1*k1(i) + b3*k3(i) + b4*k4(i) + b5*
               k5(i) + b6*k6(i);
    }
    functions(highfinalvar, t0 + h, k7, h);
    for (i = 0; i < n; i++) {
            lowfinalvar(i) = initvar(i) + d1*k1(i) + d3*k3(i) + d4*k4(i) + d5*
               k5(i) + d6*k6(i) + d7*k7(i);
    }
    functions(initvar, t0 + h, scalek, h); // generate scale
    for (i = 0; i < n; i++) {
            scale(i) = abs(initvar(i)) + abs(scalek(i)) + 0.01;
    }
```

```
        for (i = 0; i < n; i++) {
                error(i) = abs((highfinalvar(i) - lowfinalvar(i))/scale(i));
        }
        deltamax = maximum(error);
        if (5*deltamax < maxerror) {
                h = h*pow(maxerror/deltamax, 0.2); // if error is too small,
                    increase h
                return;
        }
        else if (deltamax > maxerror && abs(h) > 1e-12) { // if error is too big,
           decrease h
                h = h/3.0;
                return;
        }
        else {
                return;
        }
}

void laneemden(arrayt<double>& var, double x, arrayt<double>& k, double& h) {
    double y,z;
    y = var(0);
    z = var(1);
    k(0) = h*z;
    k(1) = h*(-2*z-x*y*y*y)/x;
        return;
}

double maximum(arrayt<double>& array) {
        double max;
        int i, n = array.n();
        max = array(0);
        for (i = 0; i < n; i++) {
                if (array(i) > max) {
                        max = array(i);
                }
        }
        return max;
}
```

7.2 Stellar Equations

```
/* Solving the Stellar Structure Equations
  Run on core i5 with g++ 5.4.0
  Gianfranco Grillo 12/03/2016
*/

#include <cstdlib>
#include <cmath>

#include <iostream>
#include <fstream>
```

```
#include <iomanip>
#include "arrayt.hpp"
using namespace std;

int main() {
    int npts = 1e5, i, j, n = 3, nsteps, minpos;
    double K = 3.841e14, pi = 4.0 * atan(1.0), M = 1.9891e33, h, hold, startP,
       stopP, stepwP, m = 0, T, L, centralT, centralP, mp = 1.672e-24, mu = 0.6, kb
        = 1.381e-16, eps0, nu, totalL = 0.0, surfaceT, sig = 5.670e-5, R, rho, G =
       6.6732e-8;
    void stellar(arrayt<double>&, double, arrayt<double>&, double&, double);
    void autosteprk4(arrayt<double>&, arrayt<double>&, int, double&, double, double
       , void(arrayt<double>&, double, arrayt<double>&, double&, double));
    int minloc(arrayt<double>&);
    ofstream fp;
    cout << "Enter␣pressure␣lower␣limit:␣" << endl;
    cin >> startP;
    cout << "Enter␣pressure␣upper␣limit:␣" << endl;
    cin >> stopP;
    cout << "Enter␣number␣of␣steps:␣" << endl;
    cin >> nsteps;
    startP = startP*2.47e17;
    stopP = stopP*2.47e17;
    arrayt<double> initvar(n), err(nsteps), finalvar(n), Plist(nsteps);
    stepwP = (stopP-startP)/nsteps;
    h = M/npts;
    for (i = 0; i < nsteps; i++) {
        m = 0;
        Plist(i) = startP + i*stepwP;
        initvar(0) = 1e-5;
        initvar(1) = Plist(i);
        initvar(2) = 0;
        while (m < 2*M) {
            hold = h;
            autosteprk4(initvar, finalvar, n, h, m, K, stellar);
            if (finalvar(1) < 0.0) break;
            if (h > hold || h == hold) { // means error is low enough, result is
               correct
                for (j = 0; j < n; j++) { initvar(j) = finalvar(j); }
                m = m + h;
            }
        }
        err(i) = abs(m/M)+abs(finalvar(1));
    }
    minpos = minloc(err);
    m = 0;
    h = M/npts;
    centralP = Plist(minpos);
    initvar(0) = 1e-5;
    initvar(1) = centralP;
    initvar(2) = 0;
    fp.open("solution.dat");
```

```
    while (m < 2*M) {
        hold = h;
        autosteprk4(initvar, finalvar, n, h, m, K, stellar);
        if (finalvar(1) < 0.0) break;
        if (h > hold || h == hold) {
            for (i = 0; i < n; i++) { initvar(i) = finalvar(i); }
            rho = pow(initvar(1)/K, 0.75);
            fp << m << setw(15) << initvar(0) << setw(15) << initvar(1) << setw(15)
                << rho << setw(15) << endl;
            m = m + h;
        }
    }
    fp.close();
    R = initvar(0);
    centralT = mu*mp*pow(centralP, 0.25)*pow(K, 0.75)/kb;
    totalL = initvar(2);
    surfaceT = pow(totalL/(4*pi*sig*R*R), 0.25);
    cout << "Mass = " << m/M << "      " << "Radius = " << R/6.9598e10 << "       "
        << "Central pressure = " << centralP/2.47e17 << "       " << "Central
       Temperature = " << centralT/1.57e7 << "      " << "Luminosity = " << totalL
       /3.83e33 << "    " << "Surface temperature = " << surfaceT/5772 << "     "
       << "Central density = " << pow(centralP/K, 0.75)/160 << endl;
    return (EXIT_SUCCESS);
}

void autosteprk4(arrayt<double>& initvar, arrayt<double>& highfinalvar, int n,
    double& h,
                                 double t0, double K, void functions(arrayt<double
                                    >&, double, arrayt<double>&, double&, double))
                                    {
        int i;
        arrayt<double> scale(n), error(n), lowfinalvar(n), k1(n), k2(n), k3(n), k4(
            n), k5(n), k6(n), k7(n), scalek(n), temp(n);
    double maxerror = 1e-10, deltamax;
        double maximum(arrayt<double>&);
        double c2 = 0.2, c3 = 0.3, c4 = 0.8, c5 = 8.0/9.0, c6 = 1.0, c7 = 1.0;
        double a21 = 0.2, a31 = 3.0/40.0, a32 = 9.0/40.0, a41 = 44.0/45.0, a42 =
            -56.0/15.0, a43 = 32.0/9.0;
        double a51 = 19372.0/6561.0, a52 = -25360.0/2187.0, a53 = 64448.0/6561.0,
            a54 = -212.0/729.0;
        double a61 = 9017.0/3168.0, a62 = -355.0/33.0, a63 = 46732.0/5247.0, a64 =
            49.0/176.0, a65 = -5103.0/18656.0;
        double b1 = 35.0/384.0, b3 = 500.0/1113.0, b4 = 125.0/192.0, b5 =
            -2187.0/6784.0, b6 = 11.0/84.0;
        double d1 = 5179.0/57600.0, d3 = 7571.0/16695.0, d4 = 393.0/640.0, d5 =
            -92097.0/339200.0, d6 = 187.0/2100.0, d7 = 1.0/40.0;
        functions(initvar, t0, k1, h, K); // generate k1
        for (i = 0; i < n; i++) {
                temp(i) = initvar(i) + a21*k1(i);
        }
        functions(temp, t0 + c2*h, k2, h, K);
        for (i = 0; i < n; i++) {
```

```
                temp(i) = initvar(i) + a31*k1(i) + a32*k2(i);
        }
        functions(temp, t0 + c3*h, k3, h, K);
        for (i = 0; i < n; i++) {
                temp(i) = initvar(i) + a41*k1(i) + a42*k2(i) + a43*k3(i);
        }
        functions(temp, t0 + c4*h, k4, h, K);
        for (i = 0; i < n; i++) {
                temp(i) = initvar(i) + a51*k1(i) + a52*k2(i) + a53*k3(i) + a54*k4(i
                    );
        }
        functions(temp, t0 + c5*h, k5, h, K);
        for (i = 0; i < n; i++) {
                temp(i) = initvar(i) + a61*k1(i) + a62*k2(i) + a63*k3(i) + a64*k4(i
                    ) + a65*k5(i);
        }
        functions(temp, t0 + c6*h, k6, h, K);
        for (i = 0; i < n; i++) {
                highfinalvar(i) = initvar(i) + b1*k1(i) + b3*k3(i) + b4*k4(i) + b5*
                    k5(i) + b6*k6(i);
        }
        functions(highfinalvar, t0 + h, k7, h, K);
        for (i = 0; i < n; i++) {
                lowfinalvar(i) = initvar(i) + d1*k1(i) + d3*k3(i) + d4*k4(i) + d5*
                    k5(i) + d6*k6(i) + d7*k7(i);
        }
        functions(initvar, t0 + h, scalek, h, K); // generate scale
        for (i = 0; i < n; i++) {
                scale(i) = abs(initvar(i)) + abs(scalek(i)) + 0.01;
        }
        for (i = 0; i < n; i++) {
                error(i) = abs((highfinalvar(i) - lowfinalvar(i))/scale(i));
        }
        deltamax = maximum(error);
        if (5*deltamax < maxerror) {
                h = h*pow(maxerror/deltamax, 0.2); // if error is too small,
                    increase h
                return;
        }
        else if (deltamax > maxerror && abs(h) > 1e-12) { // if error is too big,
            decrease h
                h = h/3.0;
                return;
        }
        else {
                return;
        }
}

void stellar(arrayt<double>& y, double m, arrayt<double>& k, double& h, double K) {
    double pi = 4.0 * atan(1.0), r, P, G = 6.6732e-8, T, rho, mp = 1.672e-24, mu =
        0.6, kb = 1.381e-16, eps0, nu;
```

```
    r = y(0);
    P = y(1);
    rho = pow(P/K, 0.75);
    T = mu*mp*P/(kb*rho);
    eps0 = -3.90476190e-23*T*T*T+1.29666667e-14*T*T-2.68666667e-07*T+1.49904762;
    nu = -1.209*log(T)+23.98;
    k(0) = h/(4*pi*r*r*rho);
    k(1) = -h*G*m/(4*pi*r*r*r*r);
    k(2) = h*eps0*rho*pow(T/1e7,nu);
        return;
}

double maximum(arrayt<double>& array) {
        double max;
        int i, n = array.n();
        max = array(0);
        for (i = 0; i < n; i++) {
                if (array(i) > max) {
                        max = array(i);
                }
        }
        return max;
}

int minloc(arrayt<double>& array) {
    double m;
        int i, n = array.n(), minpos;
        m = array(0);
    minpos = 0;
        for (i = 0; i < n; i++) {
                if (array(i) < m) {
                        m = array(i);
            minpos = i;
                }
        }
    return minpos;
}
```