```
/*

Problem number 1


*/
/*

numerov.cc: mumerov method to find eigenvalues and eigenfunctions of
            a particle in a potential well.

*/
#include <iostream>
#include <cmath>
#include <fstream>

using namespace std;


const double V0 = -83.0;             // depth of potential well in MeV
const double E_min = -83.0;          // energy band to find
const double E_max = 0.0;            // energy eigenvalue
const double eps = 1E-6;             // matching tolerance
double h = 0.004;                    // step size
                                     // well is assumed 4 fm wide
                                     // universe is 8 fm wide
                                     // 2000 total steps


double k2(int i, double E);          /* returns potential at x */
double diff(double E);               /* difference of derivatives */
void write_data(double E);           /* saved data for final plot */

int main()
{
  double E_mid, E_lo, E_hi;
  int i=0;                           // counter for iterations

  E_lo = E_min;
  E_hi = E_max;

  do
    {
      ++i;
      E_mid =(E_lo + E_hi)/2.0;              // first guess for energy eigenvale
      if (diff(E_mid)*diff(E_hi) < 0.0) E_lo = E_mid;   // the bisection algorithm
      else E_hi = E_mid;
    }while(fabs(diff(E_mid))>eps);

  cout << "Eigenvalue E = " << E_mid << endl;
  cout << "after " << i << " iterations." << endl;
  write_data(E_mid);
  return 0;
}


/*----------------------- end of main program ------------------------*/

/* function returns difference between left and right wavefunction */
double diff(double E)
{
  double psi_now, psi_last, psi_next, psi_left, psi_right;
  int i;
  psi_last = 0.0;
  psi_now = 0.00001;
  for (i=1; i<=1500; ++i)               /* left side first */
    {
      psi_next = (2*psi_now*(1 - ((5./12.)*h*h*k2(i,E)))-
                  (1. + ((1./12.)*h*h)*k2(i-1,E))*psi_last)/(1.+ ((h*h)/12.)*k2(i+1,E))
;
      psi_last = psi_now;
      psi_now = psi_next;
    }
  psi_left = psi_now;                           /* value at matching point */

  psi_last = 0.0;                               /* reset starting conditions */
  psi_now = 0.00001;
  for (i=1; i<500; ++i)         /* now the right side */
    {
      psi_next = (2*psi_now*(1 - ((5./12.)*h*h*k2(i,E)))-
                  (1. + ((1./12.)*h*h)*k2(i-1,E))*psi_last)/(1.+ ((h*h)/12.)*k2(i+1,E))
;

      psi_last = psi_now;
      psi_now = psi_next;
    }
  psi_right = psi_now;                          /* value at matching point */
  return((psi_left - psi_right));
}

/*-----------------------------------------------------------------------*/

// function returns k^2 depending on position i
double k2(int i, double E)
{
  const double units_convert = 0.4829;          // MeV^1 fm^-2
  if (i<500) return(units_convert*E);           // outside the well
  if (500<=i && i < 1000) return (units_convert*(E-((V0/2)*0.004*(i-500))));    // insi
de the well ( chage here for different shapes of wells.)
  if (i >= 1000) return (units_convert*(E-(V0*(1-(0.5*0.004*(i-1000))))));
}
/*-----------------------------------------------------------------------*/

/* write data for eigenfuntion into files left.dat, right.dat */
void write_data(double E)
{
  double psi_last, psi_now, psi_next;
  int i;

  ofstream out_file_left;
  ofstream out_file_right;

  out_file_left.open("left.dat");    // save data in files
  out_file_right.open("right.dat");

  psi_last = 0.0;
  psi_now = 0.00001;
  for (i=1; i<=1500; i++)            // left side first
    {
      psi_next = (2*psi_now*(1- ((5./12.)*h*h*k2(i,E)))-
                  (1 + ((1./12.)*h*h)*k2(i-1,E))*psi_last)/(1 + ((h*h)/12.)*k2(i+1,E));

      out_file_left << (i-1000)*h << "\t" <<  psi_next/3.0 << endl;

     psi_last = psi_now;
     psi_now = psi_next;
   }

  psi_last = 0.0;                               /* reset starting conditions */
  psi_now = 0.00001;
  for (i=1; i<500; i++)         /* now the right side */
   {
     psi_next=(2*psi_now*(1- ((5./12.)*h*h*k2(i,E)))-
               (1 + ((1./12.)*h*h)*k2(i-1,E))*psi_last)/(1 + ((h*h)/12.)*k2(i+1,E));

      out_file_right << (1000 -i)*h << "\t" <<  psi_next/3.0 << endl;

     psi_last = psi_now;
     psi_now = psi_next;
   }
  out_file_left.close();
  out_file_right.close();
  cout << "data saved in left.dat and right.dat" << endl;
}

/*

use gnuplot to plot left.dat and right.dat. modify the energy scale to find different b
ound states.

*/
```