Jan 09, 12 9:01

quadratic_equation_1a.cpp

```
//  file: quadratic_equation_1a.c
//
//  Program to calculate roots of a quadratic equation:
//      a*x^2 + b*x + c = 0
//   as an illustration of subtractive cancellation errors
//   [THIS VERSION IS NOT DEBUGGED OR FORMATTED!!!!]
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//      01/04/04  original version, converted quadratic_equation_1.c
//
//  Notes:
//   * Based on discussion in section 3.4 of Landau/Paez, "Computational
//      Physics"
//   * First pass: no subroutine, calculate all roots, read in a,b,c
//   * Use single precision
//
//  To do:
//   * pick out the best roots
//   * make it into a subroutine
//   * add double precision
//*************************************************************************


// include files
#include <iostream>             // note that .h is omitted
#include <cmath>
using namespace std;    // we need this when .h is omitted

//**********************************************************************//

main () {
float a, b, c;                // coefficients of quadratic equation

cout << endl
  << "Calculation of quadratic equation roots in single precision"
  << endl << endl;

cout << "Enter a, b, c: [with spaces between, followed by <return>] ";
  cin >> a >> b >> c;

out << "a = " << a << ", b = " << b << ", c = " << c;

disc = pow (b * b - 4. * a * c, 0.5); // definition of discriminant

float x1 = (-b + disc) / (2. * a);      // first root, standard formula
float x1p = -2. * c / (b + disc);    // first root, new formula
float x2 = (-b - disc) / (2. * a);        // second root, standard formula
float x2p = (-2. * c) / (b - disc);   // second root, new formula

cout << "    first root          second root  " << endl;
cout << fixed << setprecision (16) << x1 << "  " << x2;
cout << fixed << setprecision (16) << x1 << "  " << x2 << endl;

return (0);
}
```

Jan 06, 09 23:26

quadratic_equation_1.cpp

```
//  file: quadratic_equation_1.c
//
//  Program to calculate roots of a quadratic equation:
//      a*x^2 + b*x + c = 0
//   as an illustration of subtractive cancellation errors
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//      01/04/04  original version, converted quadratic_equation_1.c
//
//  Notes:
//   * Based on discussion in section 3.4 of Landau/Paez, "Computational
//      Physics"
//   * First pass: no subroutine, calculate all roots, read in a,b,c
//   * Use single precision to highlight the subtractive cancellations
//
//  To do:
//   * pick out the best roots
//   * make it into a subroutine
//   * add double precision
//
//*************************************************************************


// include files
#include <iostream>             // note that .h is omitted
#include <iomanip>              // note that .h is omitted
#include <cmath>
using namespace std;            // we need this when .h is omitted

//**********************************************************************//

int
main ()
{
  float a, b, c;                // coefficients of quadratic equation

  cout << endl
    << "Calculation of quadratic equation roots in single precision"
    << endl << endl;

  cout << "Enter a, b, c: [with spaces between, followed by <return>] ";
  cin >> a >> b >> c;

  cout << "a = " << a << ", b = " << b << ", c = " << c << endl;

  float disc = pow (b * b - 4. * a * c, 0.5);   // definition of discriminant

  float x1 = (-b + disc) / (2. * a);    // first root, standard formula
  float x1p = -2. * c / (b + disc);     // first root, new formula
  float x2 = (-b - disc) / (2. * a);    // second root, standard formula
  float x2p = (-2. * c) / (b - disc);   // second root, new formula

  cout << "    first root          second root  " << endl;
  cout << fixed << setprecision (16) << x1 << "  " << x2 << endl;
  cout << fixed << setprecision (16) << x1p << "  " << x2p << endl;

  return (0);
}
```

```
//  file: quadratic_equation_2.cpp
//
//  Program to calculate roots of a quadratic equation:
//      a*x^2 + b*x + c = 0
//   as an illustration of subtractive cancellation errors
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//      01/04/04  original version, based on quadratic_equation_1.cpp
//
//  Notes:
//   * Based on discussion in section 3.4 of Landau/Paez, "Computational
//      Physics"
//   * Second pass: no subroutine, calculate all roots, read in a,b,c,
//      but now pick the best roots and estimate error, output relative
//      error and 1/(a*c) to a plot file
//   * For a,b of order unity, we expect the error to go like
//        [1/(a*c)]*(machine precision)
//   * Use single precision to highlight the subtractive cancellations
//
//
//  To do:
//   * make it into a subroutine
//   * add double precision
//
//************************************************************************

// include files
#include <iostream>             // note that .h is omitted
#include <iomanip>              // note that .h is omitted
#include <fstream>              // note that .h is omitted
#include <cmath>
using namespace std;            // we need this when .h is omitted

//*********************************************************************//

int
main ()
{
  float a, b, c;                // coefficients of quadratic equation

  // open the plot file stream
  ofstream fplot ("quadratic_eq.dat");

  // print out title to screen
  cout << endl
    <<  "Calculation of quadratic equation roots in single precision"
    << endl << endl;

  // print titles to the plot file, with "#" as a comment character
  fplot << endl
    <<  "# Calculation of quadratic equation roots in single precision"
    << endl << endl;
  fplot  << "#    1/c    |relative error 1|  |relative error 2|"  << endl;

  // get the coefficients
  cout <<  "Enter a, b, c: [with spaces between, followed by <return>] ";
  cin >> a >> b >> c;

  int i_max = 8;                // maximum number of times to loop
  for (int i = 0; i < i_max; i++)
  {
    float disc = pow (b * b - 4. * a * c, 0.5); // define discriminant

    float x1 = (-b + disc) / (2. * a);  // first root, standard formula
    float x1p = -2. * c / (b + disc);           // first root, new formula
    float x2 = (-b - disc) / (2. * a);  // second root, std formula
    float x2p = (-2. * c) / (b - disc); // second root, new formula
```

```

    // print the results to the terminal
    cout << "   first root         second root " << endl;
    cout << fixed << setprecision (16) << x1 << "  " << x2 << endl;
    cout << fixed << setprecision (16) << x1p << "  " << x2p << endl;


    // best:  the roots without subtractive cancellation
    // worst: the root with subtractive cancellation
    // look at formulas to decide, considering b>=0 or b<0 separately
    float x1_best = 0., x1_worst= 0., x2_best= 0., x2_worst= 0.;

    if (b >= 0)
    {
      x1_best = x1p;
      x1_worst = x1;
      x2_best = x2;
      x2_worst = x2p;
    }
    else if (b < 0)
    {
      x1_best = x1;
      x1_worst = x1p;
      x2_best = x2p;
      x2_worst = x2;
    }

    // find the magnitude of the relative errors, assuming the "best"
    //   root is much more accurate
    float rel_error1 = fabs ((x1_worst - x1_best) / x1_best);
    float rel_error2 = fabs ((x2_worst - x2_best) / x2_best);

    cout << " (x1c-x1)/x1 = " << scientific << rel_error1
         << " (x2c-x2)/x2 = " << scientific << rel_error2
         << endl << endl;

    // print the relative errors and 1/(a*c) to the plot file
    fplot << " " << scientific << setprecision(6)
      << 1. / (a * c) << " " << rel_error1 << "    " << rel_error2
      << endl;

    c /= 10.;                   // decrease c by 10 every pass through loop
  }

  // close the plot file
  fplot.close ();

  return (0);
}
```