Augustine Stav
Professor Onuttom Narayan
April 20, 2016

Physics 242 Homework 3

1. Source:

```
package hw3prob1;
import java.io.*;
import java.lang.Math;

public class Hw3Prob1 {
    public static double f(double x) {
        return x - 4*Math.cos(x);
    }

    public static double fPrime(double x) {
        return 1 + 4*Math.sin(x);
    }

    public static void main(String[] args) {
        double x0 = -5.0;
        double x1 = -3.0;
        double x2 = 0.5*(x0 + x1);
        double xMin = 0.001;
        double fMin = 0.00001;
        double nrAcc = 0.00000001;
        boolean running = true;

        System.out.println("Bisection method: negative roots of x - 4cos(x)");
        System.out.format("%15s   %15s    %15s    %15s%n",
                "x0", "x1", "x2", "f(x2)");

        while (running) {
            x2 = 0.5*(x0 + x1);
            System.out.format("%15.7f   %15.7f    %15.7f    %15.7f%n",
                    x0, x1, x2, f(x2));
            if (  ((f(x0) < 0) && (f(x2) < 0)) ||
                    ((f(x0) > 0) && (f(x2) > 0)) ) {
                x0 = x2;
            } else {
                x1 = x2;
            }
            if ((x1 - x0) < xMin || Math.abs(f(x1)) < fMin) {
                running = false;
            }
        }

        x0 = (x1+x0)*0.5; //set x0 to the bisection root as input for Newton-
                          //Raphson
        System.out.format("bisection root: %10.5f%n", x0);
        System.out.println();
        System.out.println("Newton-Raphson:");
        System.out.format("%15s   %15s    %15s%n",
                "x0", "x1", "f(x1)");

        while (Math.abs(x1-x0) > nrAcc) {
            x0 = x1;
            x1 = x1 - f(x1)/fPrime(x1);
            System.out.format("%15.7f   %15.7f    %15.7f%n",
                    x0, x1, f(x1));
        }
        System.out.format("Newton-Raphson root: %11.8f%n", x1);
    }
```

```
    }
```

Output:

```
Bisection method: negative roots of x - 4cos(x)
             x0                 x1                  x2                  f(x2)
     -5.0000000         -3.0000000          -4.0000000          -1.3854255
     -4.0000000         -3.0000000          -3.5000000           0.2458267
     -4.0000000         -3.5000000          -3.7500000          -0.4677626
     -3.7500000         -3.5000000          -3.6250000          -0.0833347
     -3.6250000         -3.5000000          -3.5625000           0.0883743
     -3.6250000         -3.5625000          -3.5937500           0.0042765
     -3.6250000         -3.5937500          -3.6093750          -0.0390932
     -3.6093750         -3.5937500          -3.6015625          -0.0172990
     -3.6015625         -3.5937500          -3.5976563          -0.0064838
     -3.5976563         -3.5937500          -3.5957031          -0.0010968
     -3.5957031         -3.5937500          -3.5947266           0.0015916
bisection root:   -3.59521

Newton-Raphson:
             x0                 x1               f(x1)
     -3.5947266         -3.5953051          -0.0000006
     -3.5953051         -3.5953049          -0.0000000
     -3.5953049         -3.5953049           0.0000000
Newton-Raphson root: -3.59530487
```

2. Source:

```
package hw3prob2;

public class Hw3Prob2 {
    public static double f(double x) { //fPrime in Newton-Raphson (NR)
        return (x*x*x*x)*Math.exp(-x)/24.0;
    }

    public static double midpoinInteg(int n, double a, double b) { //f in NR
        double h = (b-a)/n;
        double integral = 0;
        double x = 0;

        for (int i = 0; i < n; i++) {
            x = a + (i + 0.5)*h;
            integral += f(x);
        }
        return h*integral;
    }

    //use midpoint method to find integral to 3 decimal places given the upper
    //limit of integration
    public static double integ3Dec(double b) {
        double a = 0.0;
        int n = 2;
        double integral = 0;
        double integPrev = midpoinInteg(n, a, b);
        double diff;
        boolean running = true;
        int i = 1;
        while (running) {
               n = 2*n;
               i++;
               integral = midpoinInteg(n, a, b);
               diff = Math.abs(integral - integPrev);
               if (diff < 0.001 || i > 25) {
                   running = false;
               }
               integPrev = integral;
        }
        return integral;
```

```
    }

    public static void main(String[] args) {
        double x0 = 0.0;
        double nrAcc = 0.001;
        double a = 3.0; //here, 'a' is the *upper* limit of integration, to
                        //match the problem set
        double integral = 0;

        System.out.println("Newton-Raphson: f(a) = integral of "
                + "(1/24)(x^4)exp(-x)dx from x = 0 to a, root where "
                + "f(a) = 2/3");
        System.out.format("%15s   %15s    %15s    %15s%n",
                "a", "x1", "f'(a)", "f(a)");

        while (Math.abs(a-x0) > nrAcc) {
            integral = integ3Dec(a);
            x0 = a;
            a = a - (integral - 2.0/3.0)/f(a);
            System.out.format("%15.7f   %15.7f    %15.7f    %15.7f%n",
                    x0, a, f(a), integral);
        }
        System.out.format("Newton-Raphson root: %7.4f%n", a);
    }
}
```

Output:

```
Newton-Raphson: f(a) = integral of (1/24)(x^4)exp(-x)dx from x = 0 to a, root where f(a) = 2/3
              a                x1              f'(a)               f(a)
      3.0000000         5.8685836          0.1397103          0.1846547
      5.8685836         5.6499122          0.1493585          0.6972173
      5.6499122         5.6571498          0.1490426          0.6655857
      5.6571498         5.6571521          0.1490425          0.6666663
Newton-Raphson root:  5.6572
```

3. Source:

```
package hw3prob3;

public class Hw3Prob3 {
    public static double analytic(double t) {
        return -0.5*Math.exp(2.0*t) + t*t + 2*t - 0.5;
    }

    public static void calcK(double t, double[] y, double[] k) {
        k[0] = y[0] - y[1] + 2.0;   //u'
        k[1] = y[1] - y[0] + 4.0*t; //v'
    }

   public static void rungeKutta4(double h, double t, double[] y) {
        double[] k1 = new double[y.length];
        double[] k2 = new double[y.length];
        double[] k3 = new double[y.length];
        double[] k4 = new double[y.length];
        double[] yStep = new double[y.length];
        int i = 0;
        calcK(t, y, k1);
        for (i = 0; i < y.length; i++){
            yStep[i] = y[i] + 0.5 * h * k1[i];
        }
        calcK(t+0.5*h, yStep, k2);
        for (i = 0; i < y.length; i++){
            yStep[i] = y[i] + 0.5 * h * k2[i];
        }
        calcK(t+0.5*h, yStep, k3);
        for (i = 0; i < y.length; i++){
            yStep[i] = y[i] + h * k3[i];
        }
```

```java
            calcK(t+h, yStep, k4);
            for (i = 0; i < y.length; i++){
                y[i] = y[i] + h*(k1[i] + 2*(k2[i] + k3[i]) + k4[i])/6.0;
            }
        }

        public static void printTable(double h, double h2, double tMin, double tMax,
                double[] y, double[] y2) {
            double t = tMin;
            double t2 = tMin;
            double uAnalytic = analytic(t);
            System.out.format("%10s   %10s   %10s   %10s   %10s   %10s%n",
                    "t", "u_1", "u_2", "u_ana", "|u1-u_ana|", "|u2-u_ana|");
            while (t < tMax) {
                System.out.format("%10.7f   %10.7f   %10.7f   %10.7f   %10.7f   "
                        + "%10.7f%n", t, y[0], y2[0], uAnalytic,
                        Math.abs(y[0]-uAnalytic), Math.abs(y2[0]-uAnalytic));
                rungeKutta4(h, t, y);
                t += h;

                rungeKutta4(h2, t2, y2);
                t2 += h2;
                rungeKutta4(h2, t2, y2);
                t2 += h2;

                uAnalytic = analytic(t);
            }
        }

        public static void main(String[] args) {
            //y[0] = u and y[1] = v
            double[] y = {-1.0, 0.0}; //initialize u(0) = -1, v(0) = 0 for h = 0.1
            double[] y2 = {-1.0, 0.0}; //for h2 = 0.05
            double tMin = 0.0;
            double tMax = 1.0;

            double h = 0.1;
            y[0] = -1.0; //u(0) = -1
            y[1] = 0.0;  //v(o) = 0

            double h2 = 0.05;
            y2[0] = -1.0; //u(0) = -1
            y2[1] = 0.0;  //v(o) = 0

            System.out.println("Runge-Kutta: u' = u - v + 2, v' = v - u + 4t");
            System.out.format("              h_1: %3.2f    h_1: %3.2f%n", h, h2);
            printTable(h, h2, tMin, tMax, y, y2);
        }
    }
```

Output:

```
Runge-Kutta: u' = u - v + 2, v' = v - u + 4t
              h_1: 0.10    h_1: 0.05
         t          u_1          u_2        u_ana   |u1-u_ana|   |u2-u_ana|
 0.0000000   -1.0000000   -1.0000000   -1.0000000    0.0000000    0.0000000
 0.1000000   -0.9007000   -0.9007013   -0.9007014    0.0000014    0.0000001
 0.2000000   -0.8059090   -0.8059121   -0.8059123    0.0000034    0.0000002
 0.3000000   -0.7210532   -0.7210590   -0.7210594    0.0000062    0.0000004
 0.4000000   -0.6527604   -0.6527698   -0.6527705    0.0000101    0.0000007
 0.5000000   -0.6091256   -0.6091399   -0.6091409    0.0000153    0.0000010
 0.6000000   -0.6000360   -0.6000569   -0.6000585    0.0000225    0.0000015
 0.7000000   -0.6375679   -0.6375978   -0.6376000    0.0000321    0.0000022
 0.8000000   -0.7364715   -0.7365132   -0.7365162    0.0000447    0.0000030
 0.9000000   -0.9147623   -0.9148196   -0.9148237    0.0000615    0.0000042
 1.0000000   -1.1944446   -1.1945224   -1.1945280    0.0000834    0.0000057
```

4. (a)(i) Source:

```
package hw3prob4;

public class Hw3Prob4 {
    public static void calcK(double t, double[] y, double[] k) {
        k[0] = y[1]; // dx/dt = v
        k[1] = -y[0]*y[0]*y[0]; // dp/dt = dv/dt (unit mass) = f(x) = -dV/dx
    }

    public static void rungeKutta2(double h, double t, double[] y) {
        double[] k1 = new double[y.length];
        double[] k2 = new double[y.length];
        double[] yStep = new double[y.length];
        int i = 0;
        calcK(t, y, k1);
        for (i = 0; i < y.length; i++){
            yStep[i] = y[i] + 0.5 * h * k1[i];
        }
        calcK(t+0.5*h, yStep, k2);
        for (i = 0; i < y.length; i++){
            y[i] = y[i] + h * k2[i];
        }
    }

    public static double period(double h, double tMin, double[] y) {
        double t = tMin;
        int count = 0;
        double y1 = 0.0; //y1 will be the velocity at the step before last
        System.out.format("%10s   %10s   %10s%n",
                "half T", "x", "v");
        do {
            y1 = y[1];
            if (count % 250 == 0) {
                System.out.format("%10.7f   %10.7f   %10.7f%n", t, y[0], y[1]);
            }
            rungeKutta2(h, t, y);
            t += h;
            count++;
        } while (y[1] < 0); //stop the loop when the velocity crosses 0 again

        //Let the time of the half period be t0. At t0, v0 = 0. vNow is y[1]
        //at this stage of the program (overshot v0 = 0). vPrev = is y[1] at
        //the time step immediately before this.
        //linear interp: v0-vNow = -vNow = (vNow - vPrev)(t0 - tNow)/h
        //so, t0 = tNow - h*vNow/(vNow - vPrev)
        double halfPeriod = t - h*y[1]/(y[1] - y1);
        return 2*(halfPeriod);
    }

    public static void main(String[] args) {
        double tMin = 0.0;
        //error is O(h^2), so pick h ~ sqrt(0.001)
        double h = 0.05;
        double amp = 0.1;
        double v0 = 0.0;
        //y[0] = x and y[1] = v
        double[] y = {amp, v0}; //initialize x(0) = amp, v(0) = v0
        y[0] = amp;
        y[1] = v0;

        System.out.println("Runge-Kutta Second Order: Period of V(x) = (x^4)/4");
        double period1 = period(h, tMin, y);
        System.out.printf("Period with h = %6.4f: %7.4f%n",h, period1);
        System.out.println();

        //reset values to find the period for a h = h/2
```

```
            h = h/2.0;
            y[0] = amp;
            y[1] = v0;
            double period2 = period(h, tMin, y);
             System.out.printf("Period with h = %6.4f: %7.4f%n",h, period2);
            if (Math.abs(period2-period1) < 0.00075) {
                //Err(h)~O(h^2), so abs(Err(h/2) - Err(h))~3/4 of target 0.001 error
                System.out.println("The period is correct to three sig figs.");
            } else {
                System.out.println("Error is too large. Choose a smaller h.");
            }
        }
    }
```

(i) Output:

```
Runge-Kutta Second Order: Period of V(x) = (x^4)/4
    half T            x            v
 0.0000000    0.1000000    0.0000000
12.5000000    0.0425721   -0.0069540
25.0000000   -0.0454763   -0.0069182
Period with h = 0.0500: 74.1626

    half T            x            v
 0.0000000    0.1000000    0.0000000
 6.2500000    0.0821754   -0.0052154
12.5000000    0.0425726   -0.0069540
18.7500000   -0.0014798   -0.0070711
25.0000000   -0.0454759   -0.0069182
31.2500000   -0.0843084   -0.0049738
Period with h = 0.0250: 74.1629
The period is correct to three sig figs.
```

(ii) Output with double amp = 1.0 and h = 0.005 in initialization:

```
Runge-Kutta Second Order: Period of V(x) = (x^4)/4
    half T            x            v
 0.0000000    1.0000000    0.0000000
 1.2500000    0.4257208   -0.6953955
 2.5000000   -0.4547633   -0.6918190
Period with h = 0.0050:  7.4163

    half T            x            v
 0.0000000    1.0000000    0.0000000
 0.6250000    0.8217536   -0.5215353
 1.2500000    0.4257257   -0.6953957
 1.8750000   -0.0147980   -0.7071064
 2.5000000   -0.4547591   -0.6918203
 3.1250000   -0.8430835   -0.4973820
Period with h = 0.0025:  7.4163
The period is correct to three sig figs.
```

(iii) Output with double amp = 10.0 and h = 0.0005 in initialization:

```
Runge-Kutta Second Order: Period of V(x) = (x^4)/4
    half T            x            v
 0.0000000   10.0000000    0.0000000
 0.1250000    4.2572078   -69.5395495
 0.2500000   -4.5476333   -69.1818972
Period with h = 0.0005:  0.7416

    half T            x            v
 0.0000000   10.0000000    0.0000000
 0.0625000    8.2175364   -52.1535269
 0.1250000    4.2572568   -69.5395737
 0.1875000   -0.1479797   -70.7106401
```

```
 0.2500000   -4.5475912   -69.1820291
 0.3125000   -8.4308355   -49.7381977
Period with h = 0.0003:  0.7416
The period is correct to three sig figs.
```

4. (b): The time period is ~ 7.416/amplitude.

5. (a) Source:

```
package hw3prob5;

public class Hw3Prob5 {
    public static double f(double x) {
        //dt = dx/v(x)
        return -1.0/Math.sqrt(2*(Math.cosh(4) - Math.cosh(x)));
    }

    public static double midpoinInteg(int n, double a, double b) {
        double h = (b-a)/n;
        double integral = 0;
        double x = 0;

        for (int i = 0; i < n; i++) {
            x = a + (i + 0.5)*h;
            integral += f(x);
        }

        return h*integral;
    }

    public static double simpsons(int n, double a, double b) {
        double h = (b-a)/n;
        double oddSum = 0;
        double evenSum = 0;
        double fA = f(a);
        double fB = f(b);
        double x = 0;

        for (int i = 1; i < n; i+=2) {
            x = a + i*h;
            oddSum += f(x);
        }

        for (int j = 2; j < n; j+=2) {
            x = a + j*h;
            evenSum += f(x);
        }

        return h*(fA + 4.0*oddSum + 2.0*evenSum + fB)/3.0;
    }

    public static void main(String[] args) {
        double a = 3.99; //integrate over half period, not at x=4.0 because
                         //that would divide by zero (see f(double x))
        double b = -3.99;
        int n = 2;
        double h = (b-a)/n;
        double integral = 0;
        double integPrev = simpsons(n, a, b);
        double diff;
        boolean running = true;
        int i = 1;

        System.out.println("Simpsons method: integral of dt = "
                + "-1/sqrt[2cosh4 - 2coshx] from 3.9 to -3.9");
```

```
        System.out.format("%10s    %15s    %15s%n", "trial",
                "h", "Half Period");
        System.out.format("%10d    %15.11f    %15.11f%n",
            i, h, integPrev);

        while (running) {
            n = 2*n;
            i++;
            h = (b-a)/n;
            integral = simpsons(n, a, b);
            diff = Math.abs(integral - integPrev);
            System.out.format("%10d    %15.11f    %15.11f%n", i, h, integral);

            if (diff < 0.000001 || i > 25) {
                running = false;
            }
            integPrev = integral;
        }
        double simpIntegral = integral;

        double c = 4.0;
        double d = 3.99;
        n = 2;
        integPrev = midpoinInteg(n, c, d);
        running = true;
        i = 1;
        while (running) {
            n = 2*n;
            i++;
            integral = midpoinInteg(n, c, d);
            diff = Math.abs(integral - integPrev);
            if (diff < 0.000001 || i > 25) {
                running = false;
            }
            integPrev = integral;
        }
        double midIntegral1 = integral;
        System.out.println("\nMidpoint Integration from 4.0 to 3.9: "
                + midIntegral1);

        c = -3.99;
        d = -4.0;
        n = 2;
        integPrev = midpoinInteg(n, c, d);
        running = true;
        i = 1;
        while (running) {
            n = 2*n;
            i++;
            integral = midpoinInteg(n, c, d);
            diff = Math.abs(integral - integPrev);
            if (diff < 0.000001 || i > 25) {
                running = false;
            }
            integPrev = integral;
        }
        double midIntegral2 = integral;
        System.out.println("\nMidpoint Integration from -3.9 to 4.0: "
                + midIntegral2);

//integrated over a half period: add up all integral pieces and mult*2
        System.out.format("\nPeriod: %8.5f%n",
                (simpIntegral + midIntegral1 + midIntegral2)*2);
    }
}
```

Output:

```
Simpsons method: integral of dt = -1/sqrt[2cosh4 - 2coshx] from 3.9 to -3.9
     trial                 h        Half Period
         1     -3.99000000000      4.34295034158
         2     -1.99500000000      2.76306617307
         3     -0.99750000000      2.01034079149
         4     -0.49875000000      1.66086951853
         5     -0.24937500000      1.50489720450
         6     -0.12468750000      1.43944834454
         7     -0.06234375000      1.41453441861
         8     -0.03117187500      1.40639454471
         9     -0.01558593750      1.40428083231
        10     -0.00779296875      1.40388023281
        11     -0.00389648438      1.40382781061
        12     -0.00194824219      1.40382295667
        13     -0.00097412109      1.40382260109

Midpoint Integration from 4.0 to 3.9: 0.027092179314715668

Midpoint Integration from -3.9 to 4.0: 0.02709217931471277

Period:  2.91601
```

5. (b) Source:

```
package hw3prob5b;

public class Hw3Prob5b {
    public static double f(double x) {
        //E = T + V
        //E = p^2/2 + cosh(x)
        //V = cosh(x), F = -dV/dx = -sinh(x)
        return -Math.sinh(x);
    }

    public static double energy(double[] y) {
        return (y[1]*y[1])/2.0 + Math.cosh(y[0]);
    }

    public static void velocityVerlet(double h, double[] y)
    {
        y[1] += 0.5 * h * f(y[0]); //v_n+1/2
        y[0] += h * y[1]; //x_n+1
        y[1] += 0.5* h * f(y[0]); //v_n+1
    }

    public static void main(String[] args) {
        double[] y = {4.0, 0.0}; //y[0] = x, y[1] = v
        //initialize x(0)= 4 and v(0) = 0
        double period = 2.91601;
        int[] periodNum = {1, 10, 100};
        double energyExact = Math.cosh(4.0);
        double h = period * 0.02; //50 time steps per period
        double energyDev = 0.0;
        double energ;

        System.out.println("Energy deviation using Velocity Verlet:");
        for (int periods: periodNum) {
            y[0] = 4.0;
            y[1] = 0.0;
            for (int i = 0; i <= periods*50.0; i++) {
                //if(periods == 1) {
                //    System.out.format("t: %6.3f, x: %6.3f, v: %6.3f, E: %6.3f%n",
                //    h*i, x[0], v[0], energy(x, v));
                //}
                velocityVerlet(h, y);
                energ = energy(y);
```

```
                energyDev = Math.max(energyDev,
                        Math.abs(energ - energyExact));
            }
            System.out.format("Periods: %2d, Max Energy Deviation: %7.5f%n",
                    periods, energyDev);
        }
    }
}
```

Output:

```
Energy deviation using Velocity Verlet:
Periods:  1, Max Energy Deviation: 0.19054
Periods: 10, Max Energy Deviation: 0.19081
Periods: 100, Max Energy Deviation: 0.19093
```

5. (c) Source:

```
package hw3prob5c;

public class Hw3Prob5c {
    public static double energy(double x, double v) {
        return (v*v)/2.0 + Math.cosh(x);
    }

    public static void calcK(double[] y, double[] k) {
        k[0] = y[1]; // dx/dt = v
        k[1] = -Math.sinh(y[0]); // dp/dt = dv/dt (unit mass) = f(x) = -dV/dx
    }

    public static void rungeKutta2(double h, double[] y) {
        double[] k1 = new double[y.length];
        double[] k2 = new double[y.length];
        double[] yStep = new double[y.length];
        int i = 0;
        calcK(y, k1);
        for (i = 0; i < y.length; i++){
            yStep[i] = y[i] + 0.5 * h * k1[i];
        }
        calcK(yStep, k2);
        for (i = 0; i < y.length; i++){
            y[i] = y[i] + h * k2[i];
        }
    }

    public static void main(String[] args) {
        double[] y = {4.0, 0.0}; //y[0] = x, y[1] = v
        //initialize x(0)= 4 and v(0) = 0
        double period = 2.91601;
        int[] periodNum = {1, 10, 100};
        double energyExact = Math.cosh(4.0);
        double h = period * 0.02; //50 time steps per period
        double energyDev = 0.0;
        double energ;

        System.out.println("Energy deviation using Runga Kutta 2:");
        for (int periods: periodNum) {
            y[0] = 4.0;
            y[1] = 0.0;
            if (periods == 100) {
                System.out.println("100 Periods Table:");
                System.out.format("%9s   %9s   %9s   %9s%n",
                    "t", "x", "v", "Energy");
            }
            int count = 0;
            for (int i = 0; i <= periods*50.0; i++) {
                if ((periods == 100) && (h*i < 260) && (count%1043 == 0)) {
                    System.out.format("%9.3f   %9.3f   %9.3f   %9.3g%n",
```

```
                h*i, y[0], y[1], energy(y[0], y[1]));
            }
            rungeKutta2(h, y);
            energ = energy(y[0], y[1]);
            energyDev = Math.max(energyDev,
                    Math.abs(energ - energyExact));
            count++;
        }
        System.out.format("Periods: %2d, Max Energy Deviation: %7.5f%n",
                periods, energyDev);
    }
}
}
```

Output:

```
Energy deviation using Runga Kutta 2:
Periods:  1, Max Energy Deviation: 0.40715
Periods: 10, Max Energy Deviation: 4.43467
100 Periods Table:
        t           x           v      Energy
    0.000       4.000       0.000        27.3
   60.828       2.206      -8.426        40.1
  121.656   9908556926580870000.000   -Infinity    Infinity
  182.484         NaN         NaN         NaN
  243.312         NaN         NaN         NaN
Periods: 100, Max Energy Deviation:     NaN
```

The energy diverged between 10 and 100 periods using 2nd order Runga Kutta.

6. Source:

```
package hw3prob6;

public class Hw3Prob6 {
    //lowest energy solution: psi(x) = psi(-x) (even)
    //above condition goes to psi(h) - psi(-h) = 0
    //first excited energy solution: psi(0) = 0 (odd)
    //d^2/dx^2 (psi) + k^2 psi = 0, k = 2(E-V)
    public static double kSquared(double x, double energy) {
        //return (2.0*energy - x*x); //simple oscillator
        return (2.0*energy - x*x*x*x/2.0);
    }

    public static void numerov(double h, double[] psi, double[] x,
            double energy) {
        double kSq0 = kSquared(x[0], energy);
        double kSq1 = kSquared(x[1], energy);
        double kSq2 = kSquared(x[1]+h, energy);
        double factor0 = 1.0 + (1.0/12)*h*h*kSq0;
        double factor1 = 2.0 - (10.0/12)*h*h*kSq1;
        double factor2 = 1.0 + (1.0/12)*h*h*kSq2;
        double psi0 = psi[0];
        double psi1 = psi[1];
        psi[0] = psi1;
        psi[1] = (factor1*psi1 - factor0*psi0)/factor2;
        x[0] = x[0] + h;
        x[1] = x[1] + h;
    }

    //a "function" of energy to find the root psi(E, h) - psi(E, -h) = 0
    public static double f(double[] psi, double[] x,
            double energy, boolean groundState) {
        double h = 0.01;
        psi[0] = 0.0;
        psi[1] = h;
        x[0] = -1000*h;
```

```
        x[1] = x[0]+h;
        double psiTwoStepsPrev = psi[0];
        if (groundState) {
            while (x[1] < h) { //stop when x[1] is iterated to h (even case)
                psiTwoStepsPrev = psi[0];
                numerov(h, psi, x, energy);
            }
            return psi[1] - psiTwoStepsPrev; //root condition for even function
        } else {
            while (x[1] < 0) { //stop when x[1] is iterated to 0 (odd case)
                numerov(h, psi, x, energy);
            }
            return psi[1]; //root condition for an odd function
        }
    }

    public static void main(String[] args) {
        double[] psi = new double[2];
        double[] x = new double[2];
        double ener0 = 0.0;
        double ener1 = 4.0;
        double ener2;
        double enerDiffMin = 0.0001;
        boolean running = true;

        System.out.println("Lowest Energy States, Shooting Method:");
        while (running) {
            ener2 = 0.5*(ener0 + ener1);
            if ( ((f(psi, x, ener0, true) < 0) && (f(psi, x, ener2, true) < 0))
                    || ((f(psi, x, ener0, true) > 0) &&
                    (f(psi, x, ener2, true) > 0)) ) {
                ener0 = ener2;
            } else {
                ener1 = ener2;
            }
            if ((ener1 - ener0) < enerDiffMin) {
                running = false;
            }
        }
        System.out.format("Ground State Energy: %8.4f%n", (ener1+ener0)*0.5);
        ener0 = 0.0;
        ener1 = 3.0;
        running = true;
        while (running) {
            ener2 = 0.5*(ener0 + ener1);
            if ( ((f(psi, x, ener0, false) < 0) && (f(psi, x, ener2, false) < 0))
                    || ((f(psi, x, ener0, false) > 0) &&
                    (f(psi, x, ener2, false) > 0)) ) {
                ener0 = ener2;
            } else {
                ener1 = ener2;
            }
            if ((ener1 - ener0) < enerDiffMin) {
                running = false;
            }
        }
        System.out.format("1st Excited State Energy: %8.4f%n",
                (ener1+ener0)*0.5);
    }
}
```

Output:

```
Lowest Energy States, Shooting Method:
Ground State Energy:   0.4161
1st Excited State Energy:   1.4935
```