Augustine Stav
Professor Onuttom Narayan
April 13, 2016

Physics 242 Homework 2

1. (a) Source:

```
package hw2prob1;
import java.io.*;
import java.lang.Math;

public class Hw2prob1 {
    public static double f(double x) {
        return Math.exp(x) + Math.sin(x);
    }

    public static double midpoint(double x, double h) {
        return (f(x + h/2.0) - f(x - h/2))/h;
    }

    public static void main(String[] args) {
        double exact = 2.0;
        double h = 2.0;
        double deltaPrev = 0.0;

        System.out.println("Midpoint method: derivative of e^x + sin(x)");
        System.out.format("%16s    %16s    %16s    %16s%n", "h",
                "derivative", "delta", "deltaPrev/delta");
        for (int i = 0; i < 25; i++) {
            double deriv = midpoint(0.0, h);
            double delta = Math.abs(deriv - exact);
            double ratio = deltaPrev/delta;
            System.out.format("%15.14f    %15.14f    %15.10e    %15.14f%n",
                    h, deriv, delta, ratio);
            h /= 2.0;
            deltaPrev = delta;
        }
    }
}
```

Output:

```
Midpoint method: derivative of e^x + sin(x)
               h          derivative               delta     deltaPrev/delta
2.00000000000000    2.01667217845170    1.6672178452e-02    0.00000000000000
1.00000000000000    2.00104168819590    1.0416881959e-03    16.00496052206596
0.50000000000000    2.00006510425076    6.5104250765e-05    16.00031002071381
0.25000000000000    2.00000406901075    4.0690107448e-06    16.00001937769255
0.12500000000000    2.00000025431315    2.5431315187e-07    16.00000123807694
0.06250000000000    2.00000001589457    1.5894572769e-08    15.99999921768908
0.03125000000000    2.00000000099341    9.9340979887e-10    16.00001609326944
0.01562500000000    2.00000000006209    6.2094329678e-11    15.99839798603959
0.00781250000000    2.00000000000385    3.8511416278e-12    16.12361623616236
0.00390625000000    2.00000000000023    2.2737367544e-13    16.93750000000000
0.00195312500000    1.99999999999994    5.6843418861e-14    4.00000000000000
0.00097656250000    2.00000000000000    0.0000000000e+00            Infinity
0.00048828125000    1.99999999999977    2.2737367544e-13    0.00000000000000
0.00024414062500    2.00000000000000    0.0000000000e+00            Infinity
0.00012207031250    2.00000000000000    0.0000000000e+00                 NaN
0.00006103515625    2.00000000000000    0.0000000000e+00                 NaN
```

...

As h decreases by a factor of 2, |δ| decreases by a factor of 16. |δ| goes as O(h^4).

$$\delta = \frac{h^2}{24}f^{(3)}(x) + \frac{h^4}{1920}f^{(5)}(x) + ...\ Here,\ f^{(3)}(x) = e^x - sin(x), f^{(3)}(0) = 0.$$

The third derivative vanishes, and the next term O(h^4) is now the leading term in |δ|.

1. (b) $|\delta|_{min} = 2.2737367544 \times 10^{-13}$**. After this, the expected O(h^4) behavior stops.**

1. (c) Let $|\delta|_2 = f_{mp}(h) - f_{mp}(h/2)$. My loop condition would be $|\delta|_2 >$ some number larger than O(10^{-11}). When this condition is not met, the loop ends.

2. (a) $Err^{tr} = h^2[\frac{1}{24} - \frac{1}{8}]hf'' + h^4[\frac{1}{1920} - \frac{1}{2^4 4!}]hf'''' + ... = -\frac{h^2}{12}hf'' - \frac{h^4}{480}hf'''' + ...$

$$Err^{mp} = \frac{h^2}{24}hf'' + \frac{h^4}{1920}hf'''' + ... \rightarrow Err^{mp} = -2Err^{tr}$$

$$remove\ O(h^2)\ error \rightarrow 3I = I^{tr} + 2I^{mp} \rightarrow I = \frac{1}{3}I^{tr} + \frac{2}{3}I^{mp}$$

$$I = \frac{1}{3}h\sum_m (f_m + f_{m+1})/2 + \frac{2}{3}h\sum_m f_{m+1/2} = \frac{h}{3}(f_0/2 + 2f_{1/2} + f_1 + 2f_{3/2} + ... + f_n/2)$$

2. (b) Replacing h with 2h and therefore m with 2m produces Simpson's Rule. The error is $Err^{simp} = (\frac{-1}{3 \times 480} - \frac{2}{3 \times 1920})(2h)^4 f'''' = -\frac{h^4}{180}f''''$ **where h is replaced with 2h.**

2. (c) Source:

```
package hw2prob2;
import java.io.*;
import java.lang.Math;

public class Hw2prob2 {
    public static double f(double x) {
        return 1.0/(1.0 + x*x);
    }

    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 = 0.0;
            double b = 2.0;
            int n = 2;
            double h = (b-a)/n;
            double integral;
            double integPrev = simpsons(n, a, b);
            double diff;
            boolean running = true;
            int i = 1;

            System.out.println("Simpsons method: integral of 1/(1+x^2) "
                    + "from 0 to 2");
            System.out.format("%10s    %15s    %15s    %15s    %15s%n", "trial",
                    "h", "integral", "|diff|", "|diff|/(h^4)");
            System.out.format("%10d    %15.13f    %15.13f    %15s    %15s%n",
                i, h, integPrev, "no previous", "no previous");

            while (running) {
                n = 2*n;
                i++;
                h = (b-a)/n;
                integral = simpsons(n, a, b);
                diff = Math.abs(integral - integPrev);
                System.out.format("%10d    %15.13f    %15.13f    %15.13f    "
                        + "%15.9e%n", i, h, integral, diff, diff/(h*h*h*h));

                if (diff < 0.00000001 || i > 25) {
                    running = false;
                }
                integPrev = integral;
            }
        }
    }
```

Output:

```
Simpsons method: integral of 1/(1+x^2) from 0 to 2
     trial                  h           integral             |diff|       |diff|/(h^4)
         1    1.0000000000000    1.0666666666667        no previous        no previous
         2    0.5000000000000    1.1051282051282    0.0384615384615    6.153846154e-01
         3    0.2500000000000    1.1071401243424    0.0020119192142    5.150513188e-01
         4    0.1250000000000    1.1071484061511    0.0000082818087    3.392228838e-02
         5    0.0625000000000    1.1071486982762    0.0000002921251    1.914471218e-02
         6    0.0312500000000    1.1071487165736    0.0000000182974    1.918617752e-02
         7    0.0156250000000    1.1071487177178    0.0000000011442    1.919655874e-02
```

As h decreased by a factor of two, |present – previous|/h^4 ~ constant. As expected, the error scaled as $O(h^4)$.
As in 1. (c), the accuracy condition was met when |diff| = |pres integral – prev integral| < 10^{-8}.

3. Source:

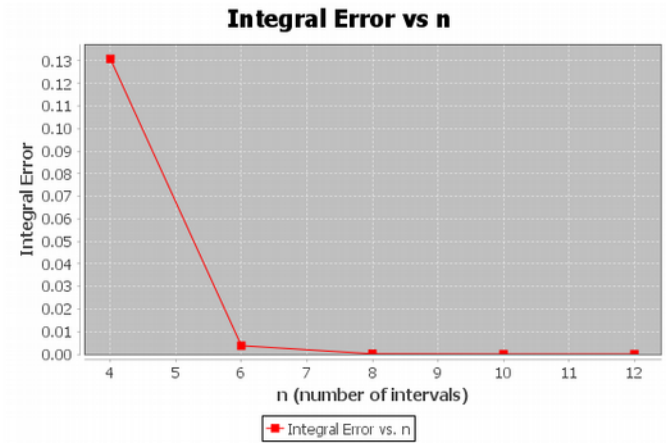
```
/*
 * JFreeChart implementation:
 *      http://www.tutorialspoint.com/jfreechart/jfreechart_xy_chart.htm
```

```
 */

package hw2prob3;
import java.lang.Math;
import java.awt.Color;
import java.awt.BasicStroke;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.data.xy.XYSeries;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.XYSeriesCollection;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;


public class Hw2prob3 extends ApplicationFrame {
    public Hw2prob3( String applicationTitle, String chartTitle,
            XYSeriesCollection dataset) {
        super(applicationTitle);
        JFreeChart xylineChart = ChartFactory.createScatterPlot(
                chartTitle , "n (number of intervals)" , "Integral Error" ,
                dataset, PlotOrientation.VERTICAL, true ,true ,false);
        ChartPanel chartPanel = new ChartPanel( xylineChart );
        chartPanel.setPreferredSize( new java.awt.Dimension( 560 , 367 ) );
        final XYPlot plot = xylineChart.getXYPlot( );
        XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer( );
        renderer.setSeriesPaint( 0 , Color.RED );
        plot.setRenderer( renderer );
        setContentPane( chartPanel );
    }

    public static double f(double x) {
        return 1/(1 + Math.cos(x)*Math.cos(x));
    }

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

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

        return h*(0.5*(fA + fB) + sumNoEndpoints);
    }

    public static void main(String[] args) {
        XYSeries dataPoints = new XYSeries("Integral Error vs. n");

        double a = 0.0;
        double b = Math.PI;
        int n = 2;
```

```
        double h = (b-a)/n;
        double integral;
        double integPrev = trapInteg(n, a, b);
        double diff;
        double diffPrev = 0;
        boolean running = true;
        int i = 1;

        System.out.println("Trapezium method: integral of 1/(1 + cos^2(x))"
                + " dx from x = 0 to pi");
        System.out.format("%10s    %10s    %10s    %10s   %13s%n", "n",
                "h", "integral", "|I_n-I_n+2|", "diffPrev/diff");
        System.out.format("%10d    %10.8f    %10.8f    %10s    %10s%n",
            n, h, integPrev, "no prev", "no prev");

        while (running) {
            n += 2;
            i++;
            h = (b-a)/n;
            integral = trapInteg(n, a, b);
            diff = Math.abs(integral - integPrev);

            dataPoints.add((double)n, diff);
            System.out.format("%10d    %10.8f    %10.8f    %10.8f    "
                    + "%10.6f%n", n, h, integral, diff, diffPrev/diff);

            if (diff < 0.0000001 || i > 25) {
                running = false;
            }
            integPrev = integral;
            diffPrev = diff;
        }

        XYSeriesCollection dataset = new XYSeriesCollection( );
        dataset.addSeries( dataPoints );
        Hw2prob3 chart = new Hw2prob3("Integral Error vs n",
                "Integral Error vs n", dataset);
        chart.pack( );
        RefineryUtilities.centerFrameOnScreen( chart );
        chart.setVisible( true );
    }
}
```

Output:

```
Trapezium method: integral of 1/(1 + cos^2(x)) dx from x = 0 to pi
         n             h      integral    |I_n-I_n+2|   diffPrev/diff
         2    1.57079633    2.35619449       no prev       no prev
         4    0.78539816    2.22529480    0.13089969      0.000000
         6    0.52359878    2.22155481    0.00373999     35.000000
         8    0.39269908    2.22144481    0.00011000     34.000000
        10    0.31415927    2.22144157    0.00000324     33.971429
        12    0.26179939    2.22144147    0.00000010     33.970588
```



As n increases by 2, Err(n) = $|I_n - I_{n+2}|$ decreases by a factor of ~34. This can be seen in the fifth column, diffPrev/diff = $|I_{n-2} - I_n| / |I_n - I_{n+2}|$ ~ 34. It follows that

$$Err(n) = Err(4) \times 34^{-(n-4)/2} \rightarrow h = \frac{\pi}{n} \rightarrow Err(h)\ goes\ as\ O(34^2(34^{1/2})^{-\pi/h}).$$

$$Err^{tr} = -\frac{h^2}{12}[f'(b) - f'(a)] + O(h^4) + ...$$

In the case of a periodic function with a-b = 1 period, f'(b) = f'(a). Similarly, higher derivatives will be equal at a and b, so that higher order errors vanish. If the leading order error term vanishes, one might expect Err^{tr} ~$O(h^q)$ where q > 2. Here, we have exponential dependence.

4. (a)

$$I = \int_0^\infty exp[-x^2]dx,\ y = \frac{1}{1+x} \rightarrow x = \frac{1}{y} - 1 \rightarrow dx = -\frac{dy}{y^2}\ and\ y(x \rightarrow \infty) = 1,\ y(x = 0) = 1$$

$$so\ I = \int_1^0 exp\Big[-(\frac{1}{y} - 1)^2\Big]\frac{-1}{y^2}dy$$

4. (b) Source:

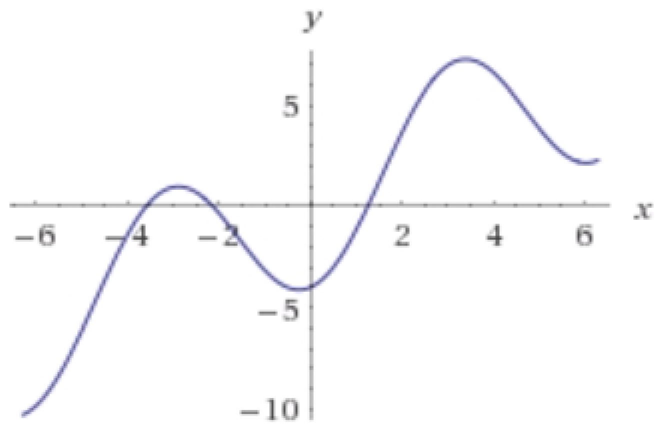
```
package hw2prob4;
import java.io.*;
import java.lang.Math;

public class Hw2prob4 {
    public static double f(double y) {
        return -(Math.exp(-(1/y - 1)*(1/y - 1)))/(y*y);
    }

    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 void main(String[] args) {
        double a = 1.0;
        double b = 0.0;
        int n = 2;
        double h = (b-a)/n;
        double integral;
        double integPrev = midpoinInteg(n, a, b);
        double diff;
        boolean running = true;
        int i = 1;

        System.out.println("Midpoint method: integral of -exp[-(1/y - 1)^2]/"
                + "(y^2) dy from y = 1 to 0");
        System.out.println("or integral of exp[-x^2] dx from x = 0 "
                + "to infinity with a change of variables");
        System.out.format("%10s    %15s    %15s    %15s%n", "trial",
                "h", "integral", "|diff|");
        System.out.format("%10d    %15.13f    %15.13f    %15s%n",
            i, h, integPrev, "no previous");

        while (running) {
            n = 2*n;
            i++;
            h = (b-a)/n;
            integral = midpoinInteg(n, a, b);
            diff = Math.abs(integral - integPrev);
            System.out.format("%10d    %15.13f    %15.13f    %15.13f%n"
                    , i, h, integral, diff);

            if (diff < 0.0000001 || i > 25) {
                running = false;
            }
            integPrev = integral;
        }
    }
}
```

Output:

```
Midpoint method: integral of -exp[-(1/y - 1)^2]/(y^2) dy from y = 1 to 0
or integral of exp[-x^2] dx from x = 0 to infinity with a change of variables
     trial                  h           integral             |diff|
         1    -0.5000000000000    0.7964000044899        no previous
         2    -0.2500000000000    0.8769831522416    0.0805831477517
         3    -0.1250000000000    0.8871222236666    0.0101390714249
         4    -0.0625000000000    0.8865515694213    0.0005706542453
         5    -0.0312500000000    0.8863083056139    0.0002432638074
         6    -0.0156250000000    0.8862472705042    0.0000610351097
         7    -0.0078125000000    0.8862320117158    0.0000152587885
         8    -0.0039062500000    0.8862281970185    0.0000038146973
         9    -0.0019531250000    0.8862272433442    0.0000009536743
        10    -0.0009765625000    0.8862270049256    0.0000002384186
        11    -0.0004882812500    0.8862269453210    0.0000000596046
```

5. function: x - 4cos(x)



Source:

```
package hw2prob5;
import java.io.*;
import java.lang.Math;

public class Hw2prob5 {
    public static double f(double x) {
        return x - 4*Math.cos(x);
    }

    public static void main(String[] args) {
        double x0 = -3.0;
        double x1 = -1.0;
        double x2;
        double xMin = 0.001;
        double fMin = 0.00001;
        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;
            }
        }

        System.out.format("root: %10.5f%n", (x1+x0)*0.5);
    }
}
```

Output:

```
Bisection method: negative roots of x - 4cos(x)
             x0                   x1                    x2                 f(x2)
     -3.0000000           -1.0000000            -2.0000000            -0.3354127
     -3.0000000           -2.0000000            -2.5000000             0.7045745
     -2.5000000           -2.0000000            -2.2500000             0.2626945
     -2.2500000           -2.0000000            -2.1250000            -0.0199347
```

```
     -2.2500000       -2.1250000       -2.1875000        0.1258968
     -2.1875000       -2.1250000       -2.1562500        0.0540602
     -2.1562500       -2.1250000       -2.1406250        0.0173262
     -2.1406250       -2.1250000       -2.1328125       -0.0012392
     -2.1406250       -2.1328125       -2.1367188        0.0080599
     -2.1367188       -2.1328125       -2.1347656        0.0034144
     -2.1347656       -2.1328125       -2.1337891        0.0010886
root:   -2.13330
```

With x0 = -5.0 and x1 = -3.0 initially, 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
root:   -3.59521
```