Okeefe Niemann
6/9/2014
1281465
PHYS 115

Final Exam

Question 1:

- Input:

```
#include <stdio.h>
#include <math.h>


int  i, j, k, N;
double xdata, ydata, yerrordata, x[10], y[10], yerror[10], U[2][2], v[2],
       U_inv[2][2], Delta, a[2], sigma[2], f[10], chi_squared;

FILE* fout;

int main()
{
        fout = fopen("questiononedata.txt", "r");

        //Imports data
        while(fscanf(fout, "%lf %lf %lf", &xdata, &ydata, &yerrordata) != EOF)
        {
                x[i] = xdata;
                y[i] = ydata;
                yerror[i++] = yerrordata;
        }

        N = 10;

        //Calulates symmetric 2x2 Matrix from x values
        for(i = 0; i < 2; i++)
        {
                for(j = 0; j < 2; j++)
                {
                        for(k = 0; k < N; k++)
                                U[i][j] += pow(x[k], i+j) / (yerror[k]*yerror[k]);
                }

        }

        //Calculates value needed to find parameters
        for(i = 0; i < 2; i++)
        {
                for(j = 0; j < N; j++)
                {
                        v[i] += y[j] * pow(x[j], i) / (yerror[j]*yerror[j]);
                }
        }

        //Determinant of above 2x2 matrix
        Delta = U[0][0]*U[1][1] - U[0][1] * U[0][1];
```

```
        //Creates inverse of the above 2x2 matrix
        U_inv[0][0] = U[1][1] / Delta;
        U_inv[0][1] = -U[0][1] / Delta;
        U_inv[1][0] = -U[0][1] / Delta;
        U_inv[1][1] = U[0][0] / Delta;

        //Calculates the two parameters from above inverted matrix and constant
        for(i = 0;  i < 2; i++)
        {
                for(j = 0; j < 2; j++)
                {
                        a[i] += U_inv[i][j] * v[j];
                }
        }

        //Calculates the error of the parameters
        for(i = 0; i < 2; i++)
        {
                sigma[i] = sqrt(U_inv[i][i]);
        }

        printf("a = %f p/m %f\n", a[0], sigma[0]);
        printf("b = %f p/m %f\n", a[1], sigma[1]);

        //Calculates the chi-squared value per degree of freedom
        for(i = 0; i < N; i++)
        {
                f[i] = a[0] + a[1]*x[i];
                chi_squared += pow(((y[i] - f[i]) / yerror[i]),2);
        }

        printf("Chi^2 = %f\n", chi_squared);
        printf("d.o.f. = %d\n", (N-2));
        printf("Chi^2/d.o.f. = %f\n", chi_squared/(N-2));

        fclose(fout);

return 0;
}
```

- Output:

```
y = a + bx

a = 2.864746 p/m 0.052811
b = 2.027720 p/m 0.010809
Chi^2 = 5.648292
d.o.f. = 8
Chi^2/d.o.f. = 0.706037
```

Question 2:

- Input:

```
#include<stdio.h>
#include<math.h>
```

```
//declares desired function
double f(double x)
{
        return 1 / (1 + x*x*x);
}

//Simpson's rule
double simp(double f(double), double a, double b, int n)
{
        double sum, h;
        int i;
        h = (b - a) / n;
        sum = f(a) + f(b);

        for (i = 0; i < n; i++){
               if (i % 2 == 0)
               {
                        sum += 2 * f(a + i * h);
               }
               else
               {
                       sum += 4 * f(a + i * h);
               }
        }
        return h / 3 * sum;
}

int main() {
        double x, a, b, h, exact, ans, prevans, EPS, sum;
        int i, n;

        a = 0;          //lower bound
        b = 2.0; //upper bound
        EPS = 1.e-6; //desired precision
        ans = 1.e50; //starting point to keep below loop running

        printf ("       h         ans         (ans - prevans)/h^4\n");

        n = 1;

        /*iterates Simpson's rule with a greater number of measuring points
        until desired precision is achieved.*/
        while (fabs(ans - prevans) > EPS)
        {
                h = (b - a) / n;
                prevans = ans;

                ans = simp(f, a, b, n);
                n *= 2;
        }
        printf("integral = %.6f \n", ans);

return 0;
}
```

- Output:

```
integral = 1.090002
```

Comment: The accuracy 10^-6 was achieved by comparing the difference between two successive iterations, increasing the number of points evaluated by a factor of two until the accuracy was met.

Question 3

- Input:

```
#include<stdio.h>
#include<math.h>

/*defines function*/
double f(double x)
{
        return (x - sin(3*x));
}

main()
{
        double f(), x, xn, xp, EPS, n;

        x = 0.5;  /*x_(n-1)*/
        xn = 1.5;  /*x_n*/
        EPS = 1.e-3;   /*desired precision*/
        n = 0;

        /*Using the secant method, loops the given function for its root until
        desired precision is achieved*/
        while (fabs(xn - x) >= EPS)
        {
                n++;
                xp = x;
                x = xn;
                xn = x - f(x) * (x - xp) / (f(x)-f(xp));
        }

        printf("positive root = %.3f\n", x);
}
```

- Output:

```
positive root = 0.760
```

Comment: The secant method was iterated until the difference between the x_n and x_{n-1} was smaller than the desired precision.

Question 4:

- Input:

```
#include<stdio.h>
#include<math.h>

//Establishes the value f(y) for given differential equation dy/dx
```

```
double f(double y, double x)
{
        return sqrt(y + x * x);
}

int main()
{
        double y, y2, y1, y0, h, k1, k2, f(), b, a, exact,x, EPS;
        int i, n;

        //limits of integration
        a = 0;
        b = 2;
        //condition for y=0
        y0 = 1;
        //analytical value of integral
        exact = 1;
        EPS = 10.e-6;
        n = 1;
        y2 = 1000;
        y1 = 1;
        printf(" Value of Integral    Precision        n\n");

        while(fabs(y2 - y1) > EPS)
        {
                y1 = y2;
                h = (b - a) / n;
                x = 0;
                y = y0;

                for (i = 1; i < n; i++)
                {
                        //Runge-Kutta Method 2
                        k1 = f(y, x);    //defines k1
                        x += h;
                        k2 = f(y + h * k1, x);    //defines k2
                        y += h/2 * (k1 + k2); //evaluates higher precision value for y'
                }

                y2 = y;
                n *= 2;
        }

        printf("y(2) = %.4f\n", y2);

return 0;
}
```

- Output:

```
y(2) = 4.7624
```

Comment: By comparing the the most recent iteration of y with the one before it, I was able to find the accuracy when the difference of the two was less than 10^{-4}.

Question 5:

- Input:

```c
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>

int main()
{
        double x, a, b, exact, favg, f2avg, error, nsigma, denominator, term, ans;
        int i, j, NPOINTS;

        NPOINTS = 100000;
        b = 2.0; //upper bound
        a = 0; //lower bound
        srand(time(NULL)); //initializes random number generator

        favg = 0;
        f2avg = 0;

        for(i = 0; i < NPOINTS; i++)
        {
                denominator = 0;

                for(j = 0; j < 8; j++)
                {
                        x = a + (b - a) * rand() / (RAND_MAX + 1.); //computes random values for x
                        denominator += x;                               //computes denominator
                }

                term = 1 / (1 + denominator); //condenses terms
                favg += term;
                f2avg += term * term;
        }

        favg /= NPOINTS;                     //computes <f>
        f2avg /= NPOINTS;         //computes <f>^2

        error = pow(2,8) * sqrt((f2avg - favg*favg)/ (NPOINTS - 1)); //error bar

        ans = favg * pow(2,8);                //multiplies <f> by the range raised to the
                                              //number of dimensions to take all dimensions
                                              //into account

        printf("number of points = %d\n", NPOINTS);
        printf("computational answer = %f\n", ans);
        printf("error bar = %f\n", error);

return 0;
}
```

- Output:

```
number of points = 100000
computational answer = 29.495722
error bar = 0.018874
```