AMS 326 | FALL 2012 | STONY BROOK UNIVERSITY

Homework 1

Dinia Gepte | 107092681

9/18/2012

Problem 1

Please write a small program to use Taylor series formula to estimate the following function

$$f(x) = (1.7541226)^{-x^4} - x^3 \sin(x^4) - 1.6360913$$

at the points $x = 0.01$ and $x = -0.01$ assuming you can easily compute the values of following function and their derivatives $f(0), f'(0), f''(0)$ … Your answer only needs to consider the first four terms of the Taylor series.

- The source code of the program is in a file called `taylor.c` included in the zip file. A copy of the code is provided below:

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

// CONSTANTS
#define DELTA 0.005
#define DELTA_INV 200.0
#define A 0.0

// PROTOTYPES
double function(double x);
double first_derivative(double x);
double second_derivative(double x);
double third_derivative(double x);

int main(void)
{
        // DECLARATIONS
        double x_value, function_value;
        double first_term, second_term, third_term, fourth_term;

        // ASKS USER FOR A NUMBER
        printf("Enter a number x to estimate the function f(x): ");
        scanf("%lf", &x_value);

        // ACTUAL CALCULATION USING TAYLOR SERIES WHERE a=0
        first_term = function(A);
        second_term = first_derivative(A)*(x_value);
        third_term = second_derivative(A)*pow(x_value-A, 2)*0.5;
        fourth_term = third_derivative(A)*pow(x_value-A, 3)/6;
        function_value = first_term + second_term + third_term + fourth_term;

        // OUTPUT
        printf("\nThe value of the function at f(%f) = %f where:", x_value, function_value);
        printf("\n  the 1st term = %f", first_term);
        printf("\n  the 2nd term = %f", second_term);
        printf("\n  the 3rd term = %f", third_term);
        printf("\n  the 4th term = %f", fourth_term);

        return 0;
}

double function(double x)
{
        return pow(1.7541226, -(pow(x, 4))) - (pow(x, 3) * sin(pow(x, 4))) - 1.6360913;
}

/*** Computes the derivatives using the Central Difference Method. ***/

double first_derivative(double x)
{
```

```
        return (function(x+DELTA) - function(x-DELTA))*(0.5*DELTA_INV);
}

double second_derivative(double x)
{
        return (function(x+DELTA) + function(x-DELTA) - 2*function(x)) * pow(DELTA_INV, 2);
}

double third_derivative(double x)
{
        return ((function(x+DELTA) - function(x-DELTA) - 2*first_derivative(x)*DELTA)*6) * (0.5) *
pow(DELTA_INV, 3);
}
```

- When run, the program will provide the following results:
  a) for x = 0.01

```
Enter a number x to estimate the function f(x): 0.01

The value of the function at f(0.010000) = -0.636091 where:
  the 1st term = -0.636091
  the 2nd term = -0.000000
  the 3rd term = -0.000000
  the 4th term = 0.000000
```

  b) for x = -0.01

```
Enter a number x to estimate the function f(x): -0.01

The value of the function at f(-0.010000) = -0.636091 where:
  the 1st term = -0.636091
  the 2nd term = 0.000000
  the 3rd term = -0.000000
  the 4th term = -0.000000
```

- This program solves the old problem before the function was changed. All the derivatives at a=0 gives zero for an answer, hence the second term onwards are zeroes. In most floating point cases, a zero can be represented by both positive and negative, so -0.00 == (+)0.00. By using Taylor series, we were able to calculate an approximation of the value of the function. However, looking at the source code closely, a faster way to do this is by using the function called **`function`** with prototype **`double function(double x)`**`;` without calculating the derivatives, i.e. get the input from the user and call the function. The derivative functions are formulas derived by expanding the Taylor series. `DELTA = 0.005` is arbitrarily chosen for this program and its inverse, `DELTA_INV = 200`, is computed before runtime to avoid division operators which take many cycles.

Problem 2

Most digital computers are incapable of computing the square root or the inverse of a number directly, as computers can only perform additions, subtractions and multiplications of a pair of real numbers. Please write a program to find the square root of the following number

$$a = 1.7541226$$

Hint: The simplest way is to write a program to find the root of the following equation:

$$x^2 - 1.7541226 = 0$$

- The source code of the program is in a file called `sqrt.c` included in the zip file. A copy of the code is provided below:

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

void check(double n1, double n2, int *pass);

int main(void)
{
	double n1, n2, number;
	int pass = 1;	// USED FOR TOLERANCE

	// GET USER INPUT
	printf("Square root of: ");
	scanf("%lf", &number);

	// SET AN ARBITRARY VALUE OF n1
	n1 = number / 2;

	// THIS LOOP WILL DETERMINE THE ROOT BY USING NEWTON'S METHOD.
	// IT WILL BE DONE WHEN n1 AND n2 ARE ACCURATE UP TO 6 DECIMALS
	while (pass != 0)
	{
		n2 = 0.5 * (n1 + number/n1);
		check(n1, n2, &pass);
		n1 = n2;
	}

	// OUTPUT
	printf("= %f", n2);

	return 0;
}

/** Performs the checking whether n1 and n2 are correct up to 6 decimals. **/
void check(double n1, double n2, int *pass)
{
	if (fabs(n2-n1) < 0.000001)
		*pass = 0;
}
```

- When run, the program will produce the following result:

```
Square root of: 1.7541226
= 1.324433
```

- This is a simple program that asks the user for a positive real number that computes for its square root using Newton's Method. The method basically computes for n_1 and n_2 continuously until the numbers fall up to 6 correct digits. The program runs fairly fast and the only bottleneck would be the division operations necessary to do the calculations.