수치해석 HW2

2019062833 컴퓨터 소프트웨어학부 김유진

1. Chapter 1을 읽고 다음의 질문에 요약하여 답하기

1) 메모리 할당에서 포인터가 쓰이는 방법

-포인터는 배열과 밀접하게 관련되어 있다. a[i]는 *(a) 포인터를 j 만큼 증가시켜서 얻은 주소의 내용이 된다. 이러한 포인터는 zero-offset vector를 unit-offset vector, 더 나아가 arbitrary offset vector로 변환하여 사용하는데 쓰일 수 있다. 예를 들어 5개의 원소를 가진 B라는 배열이 있다면 이 배열의 인덱스 범위는 0-4이다. 직관적으로 더 잘 이해하기 위해 1부터 시작하는 인덱스를 사용하고 싶다면 A = B-1 (A는 포인터)로 설정한 후 A[1…5]를 사용하면 된다. 이런 식으로 임의의 오프셋 벡터를 malloc을 사용하여 할당하게 끔 하는 함수들을 유틸리티 파일인 nrutil.c가 포함하고 있다.

-크기가 가변적인 2차원 배열에서의 경우 a[M][N]과 같이 설정할 수 없어 추가적인 기술이 필요하게 된다. 포인터를 사용하지 않고 고정 크기로 선언 시 위의 배열은 주소에 M번의 I 더하기, N번의 J 더하기를 한 주소 값을 의미하게 된다. 반면 포인터 사용시 a[i][j]는 a의 주소에 i를 더하고 그 주소에 j를 추가하여서 주소를 만들어 낸다. 즉 배열의 기본크기는 이 계산에 들어가지 않게 되는 것이다.

2) 함수에서 포인터가 쓰이는 방법

- 1차원 배열의 경우 앞에서 언급한대로 시작 인덱스와 끝 인덱스를 함수의 인자로 받아 임의의 오프셋 벡터를 사용하는 함수가 유틸리티 파일에 존재한다.

- 2차원 배열의 경우도 앞서 언급한 이유로 함수의 인자로 배열 자체를 넘기는 것이 아닌 그 배열을 가리키는 포인터를 넘기게 되면, 그 배열의 물리적 크기는 알 수 없어도 함수 내부에서 배열 인덱싱 사용(arr[i][j])이 가능해진다. 이는 앞서 말한 포인터의 원리 때문이다.

2. 교재 문제 풀기

1) 3.6

3.6 Evaluate e^{-5} using two approaches

$$e^{-x} = 1 - x + \frac{x^2}{2} - \frac{x^3}{3!} + \cdots$$

and

$$e^{-x} = \frac{1}{e^x} = \frac{1}{1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots}$$

and compare with the true value of 6.737947×10^{-3}. Use 20 terms to evaluate each series and compute true and approximate relative errors as terms are added.

```c
#include <stdio.h>
#define TERMS 20
#define N 5
//근사값을 구하는 함수
double approx_exp(int method) {
    double ans = 1, term = 1;

    //20번의 term 에 대해
    for (int i = 1; i < TERMS; i++) {
        //1번 메소드 인경우 -1을 곱해서 부호를 음수<-> 양수로 토글
        if (method == 1) term *= -1;
        term *= (double) N/i;
        ans += term;
    }

    //각 메소드에 맞게 리턴값 설정
    if (method == 1)
        return ans;
    else
        return 1 / ans;
}

void problem_3_6() {
    //실제값
    double true_val = 6.737947e-3;

    //각각의 방법에 대해 계산
    for (int i = 1; i <= 2; i++) {
        printf("방법 #%d\n", i);
        double approx_val = approx_exp(i);
        //근사값
        printf("근사값: %e\n", approx_val);
        //상대적 오차
        printf("상대적인 오차: %e\n", (true_val - approx_val) / true_val);
        printf("\n");
    }
}

int main() {
    problem_3_6();
}
```

```
방법 #1
근사값: 6.706341e-03
상대적인 오차: 4.690738e-03

방법 #2
근사값: 6.737949e-03
상대적인 오차: -3.450780e-07
```

2) 3.7

3.7 The derivative of $f(x) = 1/(1 - 3x^2)$ is given by

$$\frac{6x}{(1 - 3x^2)^2}$$

Do you expect to have difficulties evaluating this function at $x = 0.577$? Try it using 3- and 4-digit arithmetic with chopping.

$f'(0.577) = \dfrac{6(0.577)}{(1-3\times 0.577^2)^2} = 2,352,911$

① 3 digit 방식

$6x = 6(0.577) = 3.462 \doteq 3.46$

$x = 0.577$

$x^2 = 0.332929 \to 0.332$

$3x^2 \doteq 0.996$

$1-3x^2 = 0.004$

$f'(0.577) = \dfrac{3.46}{0.004^2} = 216.250$

② 4 digit 방식

$6x = 3.462 \to 3.462$

$x = 0.577$

$x^2 \doteq 0.332929 \to 0.3329$

$3x^2 = 0.9987$

$1-3x^2 \doteq 0.0013$

$f'(0.577) = \dfrac{3.462}{0.0013^2} = 2,048,521$

3) 4.2

4.2 The Maclaurin series expansion for cos x is

$$\cos x = 1 - \frac{x^2}{2} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$

Starting with the simplest version, cos x = 1, add terms one at a time to estimate cos(π/3). After each new term is added, compute the true and approximate percent relative errors. Use your pocket calculator to determine the true value. Add terms until the absolute value of the approximate error estimate falls below an error criterion conforming to two significant figures.

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

void problem_4_2() {
    double sum = 1, term = 1;

    //합이 실제 값인 0.5이 될때까지 반복
    for (int i = 1; sum != 0.5; i++) {
        term *= -M_PI * M_PI / ((2 * i - 1) * (2 * i)) / 9;
        sum += term;
        printf("[%d]: estimated %e, relative err %f%%\n", i, sum, fabs(0.5 - sum) * 100 / sum);
    }
    printf("\n");
}

int main() {
    problem_4_2();
}
```

```
[1]: estimated 4.516886e-01, relative err 10.695721%
[2]: estimated 5.017962e-01, relative err 0.357954%
[3]: estimated 4.999646e-01, relative err 0.007087%
[4]: estimated 5.000004e-01, relative err 0.000087%
[5]: estimated 5.000000e-01, relative err 0.000001%
[6]: estimated 5.000000e-01, relative err 0.000000%
[7]: estimated 5.000000e-01, relative err 0.000000%
[8]: estimated 5.000000e-01, relative err 0.000000%
[9]: estimated 5.000000e-01, relative err 0.000000%
```

4) 4.5

error criterion conforming to two significant figures.

4.5 Use zero- through third-order Taylor series expansions to predict $f(3)$ for

$$f(x) = 25x^3 - 6x^2 + 7x - 88$$

using a base point at $x = 1$. Compute the true percent relative error ε_t for each approximation.

```
true val: 554.000000
[1]: estimated -62.000000, relative err 111.191336%
[2]: estimated 78.000000, relative err 85.920578%
[3]: estimated 354.000000, relative err 36.101083%
[4]: estimated 554.000000, relative err 0.000000%
```

```c
#include <stdio.h>
#include <math.h>
const int MAX_LEN = 5;
typedef struct Polynomial {
    int c[MAX_LEN];
} Polynomial;

//미분 함수
void differentiate(Polynomial *f) {
    for (int i = 0; i < MAX_LEN - 1; i++) {
        f->c[i] = f->c[i + 1] * (i + 1);
    }
}

//실제값
int get_true(Polynomial *f, int x) {
    int ans = 0;
    for (int i = 0; i < MAX_LEN; i++) {
        int term = 1;
        for (int j = 0; j < i; j++)
            term *= x;
        //계수와 x의 거듭제곱 값을 곱해서 각 항의 값을 구해낸다.
        ans += term * f->c[i];
    }
    return ans;
}
```

```c
void problem_4_5() {
    const int x = 3;
    const int a = 1;
    const int steps = 3;
    //각 항의계수들을 넣어서 다항식을 생성
    Polynomial f = { {-88, 7, -6, 25} };

    //다항식의 x에 실제값을 넣은 결과를 반환
    double true_val = get_true(&f, x);
    printf("true val: %f₩n", true_val);

    double sum = 0;
    int fac = 1;
    for (int i = 0; i <= steps; i++) {
        //현재 f값에 a 대입
        double term = get_true(&f, a);

        //(x-a)^(?) 만큼 곱해 준다.
        for (int j = 0; j < i; j++)
            term *= (x - a);

        //팩토리얼 분모를 적용 후 항 더하기
        term /= fac;
        sum += term;

        printf("[%d]: estimated %f, relative err %f%%₩n",i + 1,sum, fabs(true_val - sum) * 100 /

        //미분후 나눠주는 수 1 증가
        differentiate(&f);
        fac *= i + 1;
    }
    printf("₩n");
}
int main() {
    problem_4_5();
}
```

5) 4.12

4.12 Repeat Prob. 4.11 with $g = 9.81$, $t = 6$, $c = 12.5 \pm 1.5$, and $m = 50 \pm 2$.

```
Estimated v: 30.484373 ± 2.776149
```

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

void problem_4_12() {
    double g = 9.81, t = 6,c = 12.5,dc = 1.5, m = 50, dm = 2;
    double v_partial_c, v_partial_m, delta_v, v;
    //estimated error : Δv = |∂v/∂c| · Δc + |∂v/∂m| · Δm
    v_partial_c = -g * m / (c*c) * (1 - exp(-t * c / m)) - g * t / c * -exp(-t * c / m);
    v_partial_m = (1 - (c*t / m + 1)*exp(-c * t / m)) * g / c;
    delta_v = fabs(v_partial_c) * dc + fabs(v_partial_m) * dm;
    v = g * m*(1 - exp(-c * t / m)) / c;

    printf("Estimated v: %f ± %f₩n", v, delta_v);
}

int main() {
    problem_4_12();
}
```