

TI DSP, MCU 및 Xilinx Zynq FPGA

프로그래밍 전문가 과정

2018-05-21 (58 회차)

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 - 정유경
ucong@naver.com

오늘 배운 내용

1. Expression Tree 기반 공학용 계산기 설계 방법 → 미분방정식 구현, Matlab, Labview 에 사용된다.
2. 계수가 상수항인 2 계 미분 방정식
특성방정식의 근의 유형 세가지
중근일 경우 한해를 이용하여 계수내림법으로 다른 해를 구한다.
공액복소근을 갖는 경우 '오일러공식'을 이용하여 새로운 형태의 기저 y_1', y_2' 을 구하고 이를 이용하여 일반해를 표현한다.
3. 코시-오일러 미분 방정식(계수가 상수항이 아닌 2 계 미분방정식)
4. 비동차 2 계 미분 방정식의 일반해는 동차해 + 특수해($y_g = y_h + y_p$)의 형태로 구한다
5. 연립 미분 방정식의 경우에는 라플라스 변환을 사용한다
6. 전원측의 RC 다발 회로
 - C 가 전원쪽의 RF Noise 를 제거한다
 - 갑작스러운 전원의 변화를 C 가 완화한다. (평활작용)

C 언어 기반 코드 작성할 것

1. ODE
2. 행렬식을 이용한 Gauss 소거법

1. 미분방정식, 적분을 구현하는 방법 : Parsing Tree (구문트리) 구현 계획

사람이 연산하는 방식은 중위연산이지만, 컴퓨터는 이를 전위연산으로 표현하여 연산자가 앞에 위치한다. 따라서 주어진 수식을 우선 전위연산으로 바꾸는 것부터 구현해본다. (이때 연산자의 우선순위나 괄호도 고려한다)

바꾸는 방법은 스택 2 개를 활용한다.

이진트리의 전위, 중위, 후위 운행도 고려해 본다.

2.2 계 동차/비동차 선형 미분 방정식

[illegible][illegible]

3. 선생님 코드 분석

(1) ODE

```
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define SAMPLE_TIME      0.001f

#define LEFT              0
#define RIGHT             1

void init_sample_x(float *data)
{
    int i;

    for(i = 0; i < 10001; i++)
        data[i] = -5 + SAMPLE_TIME * i;

    //printf("Check Final: %f\n", data[10000]);
}

void init_sample_data(float *data, float *sam_x)
{
    int i;

    for(i = 0; i < 10001; i++)
        data[i] = 3 * exp(-pow(sam_x[i], 2.0));

    //printf("Check Mid Val: %f\n", data[5000]);
}

void calc_derivative(float *data, float *y)
{
    int i;

    for(i = 0; i < 10000; i++)
        data[i] = (y[i + 1] - y[i]) / SAMPLE_TIME;
}

void print_arr(float *data, int num)
{
    int i;

    for(i = 0; i < num; i++)
    {
        printf("data[%d] = %17.15f", i, data[i]);

        if(!(i % 3))
```

(2) Add Gauss Elim Based Inv Matrix

```
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

void init_mat(float (*A)[3])
{
    int i, j;

    for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++)
            A[i][j] = rand() % 4;
}

void print_mat(float (*R)[3])
{
    int i, j;

    for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 3; j++)
            printf("%10.4f", R[i][j]);
        printf("\n");
    }
    printf("\n");
}

void add_mat(float (*A)[3], float (*B)[3], float (*R)[3])
{
    int i, j;

    for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++)
            R[i][j] = A[i][j] + B[i][j];
}

void sub_mat(float (*A)[3], float (*B)[3], float (*R)[3])
{
    int i, j;

    for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++)
            R[i][j] = A[i][j] - B[i][j];
}

void scale_mat(float scale_factor, float (*A)[3], float (*R)[3])
{

```

| | |
|---|--|
| <pre> printf("\n"); else printf(" "); } } int find_prime_idx(char *expr, char *dch) { int i; for(i = 0; expr[i]; i++) if(expr[i] == '\n') { *dch = expr[i - 1]; return i; } } bool separete_non_depend(char *expr, int idx, char dch, char *nd) { bool div_flag; int i, j = 0; for(i = idx + 1; expr[i] != dch; i++) { if(expr[i] == ' ' expr[i] == '=') continue; else nd[j++] = expr[i]; if(expr[i] == '/') div_flag = true; } return div_flag; } void integral(char *lr, char *nd, int lr_choice, bool div_flag) { int i, j = 0, k = 0; int num, exponent; float coef; char digit[32] = {0}; char var; if(lr_choice == RIGHT) goto right; for(i = 0; nd[i]; i++) { </pre> | <pre> int i, j; for(i = 0; i < 3; i++) for(j = 0; j < 3; j++) R[i][j] = scale_factor * A[i][j]; } #ifdef 0 A[0][0] A[0][1] A[0][2] B[0][0] B[0][1] B[0][2] A[1][0] A[1][1] A[1][2] B[1][0] B[1][1] B[1][2] A[2][0] A[2][1] A[2][2] B[2][0] B[2][1] B[2][2] A[0][0]*B[0][0]+A[0][1]*B[1][0]+A[0][2]*B[2][0] A[0][0]*B[0][1]+A[0][1]*B[1][1]+A[0][2]*B[2][1] A[0][0]*B[0][1]+A[0][1]*B[1][1]+A[0][2]*B[2][1] A[0][0]*B[0][2]+A[0][1]*B[1][2]+A[0][2]*B[2][2] A[1][0]*B[0][0]+A[1][1]*B[1][0]+A[1][2]*B[2][0] A[1][0]*B[0][1]+A[1][1]*B[1][1]+A[1][2]*B[2][1] A[1][0]*B[0][1]+A[1][1]*B[1][1]+A[1][2]*B[2][1] A[1][0]*B[0][2]+A[1][1]*B[1][2]+A[1][2]*B[2][2] A[2][0]*B[0][0]+A[2][1]*B[1][0]+A[2][2]*B[2][0] A[2][0]*B[0][1]+A[2][1]*B[1][1]+A[2][2]*B[2][1] A[2][0]*B[0][1]+A[2][1]*B[1][1]+A[2][2]*B[2][1] A[2][0]*B[0][2]+A[2][1]*B[1][2]+A[2][2]*B[2][2] #endif void mul_mat(float (*A)[3], float (*B)[3], float (*R)[3]) { R[0][0] = A[0][0]*B[0][0]+A[0][1]*B[1][0]+A[0][2]*B[2][0]; R[0][1] = A[0][0]*B[0][1]+A[0][1]*B[1][1]+A[0][2]*B[2][1]; R[0][2] = A[0][0]*B[0][2]+A[0][1]*B[1][2]+A[0][2]*B[2][2]; R[1][0] = A[1][0]*B[0][0]+A[1][1]*B[1][0]+A[1][2]*B[2][0]; R[1][1] = A[1][0]*B[0][1]+A[1][1]*B[1][1]+A[1][2]*B[2][1]; R[1][2] = A[1][0]*B[0][2]+A[1][1]*B[1][2]+A[1][2]*B[2][2]; R[2][0] = A[2][0]*B[0][0]+A[2][1]*B[1][0]+A[2][2]*B[2][0]; R[2][1] = A[2][0]*B[0][1]+A[2][1]*B[1][1]+A[2][2]*B[2][1]; R[2][2] = A[2][0]*B[0][2]+A[2][1]*B[1][2]+A[2][2]*B[2][2]; } float det_mat(float (*A)[3]) { return A[0][0] * (A[1][1] * A[2][2] - A[1][2] * A[2][1]) + A[0][1] * (A[1][2] * A[2][0] - A[1][0] * A[2][2]) + A[0][2] * (A[1][0] * A[2][1] - A[1][1] * A[2][0]); } void trans_mat(float (*A)[3], float (*R)[3]) { R[0][0] = A[0][0]; R[1][1] = A[1][1]; R[2][2] = A[2][2]; R[0][1] = A[1][0]; R[1][0] = A[0][1]; </pre> |
|---|--|

| | |
|---|--|
| <pre> if(nd[i] > 0x61 && nd[i] < 0x7A) { var = nd[i]; if(nd[i + 1] == '^') exponent = atoi(&nd[i + 2]); else exponent = 1; } else digit[j++] = nd[i]; } num = atoi(digit); #ifdef __DEBUG__ printf("num = %d\n", num); #endif if(exponent == 1) { char sign; coef = 0.5; if(fabs(coef * num) == 1.0) ; else // expression floating sprintf, and atof ; if(coef * num > 0) ; else sign = '-'; lr[k++] = sign; lr[k++] = var; strncpy(&lr[k], "^2 + C", 6); printf("left = %s\n", lr); } else /* TODO */ ; return; right: strncpy(&lr[k], "ln ", 3); lr[3] = nd[0]; printf("right = %s\n", lr); if(div_flag) </pre> | <pre> R[0][2] = A[2][0]; R[2][0] = A[0][2]; R[2][1] = A[1][2]; R[1][2] = A[2][1]; } #ifdef 0 R[0][1] = A[1][2] * A[2][0] - A[1][0] * A[2][2]; R[0][2] = A[1][0] * A[2][1] - A[1][1] * A[2][0]; R[1][0] = A[0][2] * A[2][1] - A[0][1] * A[2][2]; R[1][2] = A[0][1] * A[2][0] - A[0][0] * A[2][1]; R[2][0] = A[0][1] * A[1][2] - A[0][2] * A[1][1]; R[2][1] = A[0][2] * A[1][0] - A[0][0] * A[1][2]; #endif void adj_mat(float (*A)[3], float (*R)[3]) { R[0][0] = A[1][1] * A[2][2] - A[1][2] * A[2][1]; R[0][1] = A[0][2] * A[2][1] - A[0][1] * A[2][2]; R[0][2] = A[0][1] * A[1][2] - A[0][2] * A[1][1]; R[1][0] = A[1][2] * A[2][0] - A[1][0] * A[2][2]; R[1][1] = A[0][0] * A[2][2] - A[0][2] * A[2][0]; R[1][2] = A[0][2] * A[1][0] - A[0][0] * A[1][2]; R[2][0] = A[1][0] * A[2][1] - A[1][1] * A[2][0]; R[2][1] = A[0][1] * A[2][0] - A[0][0] * A[2][1]; R[2][2] = A[0][0] * A[1][1] - A[0][1] * A[1][0]; } bool inv_mat(float (*A)[3], float (*R)[3]) { float det; det = det_mat(A); if(det == 0.0) return false; adj_mat(A, R); #ifdef __DEBUG__ printf("Adjoint Matrix\n"); print_mat(R); #endif scale_mat(1.0 / det, R, R); return true; } </pre> |
|---|--|

```

    {
    }
    else
        // TODO
        ;
}

void solve_ode(char *expr)
{
    int idx;
    bool div_flag;
    char depend_ch;
    char non_depend[32] = {0};

    char left[32] = {0};
    char right[32] = {0};

    idx = find_prime_idx(expr, &depend_ch);
    div_flag = seperate_non_depend(expr, idx, depend_ch, non_depend);

#ifdef __DEBUG__
    printf("nd = %s\n", non_depend);
#endif

    integral(left, non_depend, LEFT, div_flag);
    integral(right, &depend_ch, RIGHT, div_flag);
}

int main(void)
{
    float sample_x[10001] = {0};
    float orig_y[10001] = {0};
    float y_prime[10000] = {0};
    float restore_data[10001] = {0};

    init_sample_x(sample_x);
    init_sample_data(orig_y, sample_x);

    calc_derivative(y_prime, orig_y);
    //print_arr(y_prime, 10000);

    printf("Solve y' = -2xy\n");
    solve_ode("y' = -2xy");

    return 0;
}

```

```

void molding_mat(float (*A)[3], float *ans, int idx, float (*R)[3])
{
    int i, j;

    for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 3; j++)
        {
            if(j == idx)
                continue;
            R[i][j] = A[i][j];
        }

        R[i][idx] = ans[i];
    }
}

void crammer_formula(float (*A)[3], float *ans, float *xyz)
{
    float detA, detX, detY, detZ;
    float R[3][3] = {};

    detA = det_mat(A);

    molding_mat(A, ans, 0, R);
#ifdef __DEBUG__
    print_mat(R);
#endif
    detX = det_mat(R);

    molding_mat(A, ans, 1, R);
#ifdef __DEBUG__
    print_mat(R);
#endif
    detY = det_mat(R);

    molding_mat(A, ans, 2, R);
#ifdef __DEBUG__
    print_mat(R);
#endif
    detZ = det_mat(R);

    xyz[0] = detX / detA;
    xyz[1] = detY / detA;
    xyz[2] = detZ / detA;
}

void print_vec3(float *vec)
{
    int i;

```

```

        for(i = 0; i < 3; i++)
            printf("%10.4f", vec[i]);

        printf("\n");
    }

void create_3x4_mat(float (*A)[3], float *ans, float (*R)[4])
{
    int i, j;

    for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 3; j++)
            R[i][j] = A[i][j];

        R[i][3] = ans[i];
    }
}

void print_3x4_mat(float (*R)[4])
{
    int i, j;

    for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 4; j++)
            printf("%10.4f", R[i][j]);
        printf("\n");
    }
    printf("\n");
}

void adjust_3x4_mat(float (*A)[4], int idx, float (*R)[4])
{
    int i, j;
    float div_factor;

    for(i = idx + 1; i < 3; i++)
    {
        //div_factor = -A[idx][idx] / A[idx + 1][idx];
        //div_factor = -A[idx + 1][idx] / A[idx][idx];
        //div_factor = -A[i][0] / A[idx][0];
        div_factor = -A[i][idx] / A[idx][idx];
        printf("div_factor = %f\n", div_factor);

        for(j = 0; j < 4; j++)
            R[i][j] = A[idx][j] * div_factor + A[i][j];
    }
}

```



```

void finalize(float (*R)[4], float *xyz)
{
    xyz[2] = R[2][3] / R[2][2];
    xyz[1] = (R[1][3] - R[1][2] * xyz[2]) / R[1][1];
    xyz[0] = (R[0][3] - R[0][2] * xyz[2] - R[0][1] * xyz[1]) / R[0][0];
}

void gauss_elimination(float (*A)[3], float *ans, float *xyz)
{
    float R[3][4] = {};

    create_3x4_mat(A, ans, R);
#ifdef __DEBUG__
    print_3x4_mat(R);
#endif

    adjust_3x4_mat(R, 0, R);
#ifdef __DEBUG__
    print_3x4_mat(R);
#endif

    adjust_3x4_mat(R, 1, R);
#ifdef __DEBUG__
    print_3x4_mat(R);
#endif

    finalize(R, xyz);
}

void create_3x6_mat(float (*A)[3], float (*R)[6])
{
    int i, j;

    for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++)
        {
            R[i][j] = A[i][j];

            if(i == j)
                R[i][j + 3] = 1;
            else
                R[i][j + 3] = 0;
        }
}

void print_3x6_mat(float (*R)[6])
{
    int i, j;

    for(i = 0; i < 3; i++)
    {

```

```

        for(j = 0; j < 6; j++)
            printf("%10.4f", R[i][j]);
        printf("\n");
    }
    printf("\n");
}

void adjust_3x6_mat(float (*A)[6], int idx, float (*R)[6])
{
    int i, j;
    float div_factor, scale;

    scale = A[idx][idx];

    for(i = idx + 1; i < 3; i++)
    {
        //div_factor = -A[idx][idx] / A[idx + 1][idx];
        //div_factor = -A[idx + 1][idx] / A[idx][idx];
        //div_factor = -A[i][0] / A[idx][0];
        div_factor = -A[i][idx] / A[idx][idx];
        printf("div_factor = %f\n", div_factor);

        if(div_factor == 0.0)
            continue;

        for(j = 0; j < 6; j++)
            R[i][j] = A[idx][j] * div_factor + A[i][j];
    }

    for(j = 0; j < 6; j++)
        R[idx][j] = A[idx][j] / scale;
}

void gauss_elim_mat(float (*A)[3], float (*R)[3])
{
    float mid[3][6] = {};

    create_3x6_mat(A, mid);
#ifdef __DEBUG__
    print_3x6_mat(mid);
#endif

    adjust_3x6_mat(mid, 0, mid);
#ifdef __DEBUG__
    print_3x6_mat(mid);
#endif

    adjust_3x6_mat(mid, 1, mid);
#ifdef __DEBUG__
    print_3x6_mat(mid);
#endif
}

```

```
}

int main(void)
{
    bool inv_flag;

    float test[3][3] = {{2.0, 0.0, 4.0}, {0.0, 3.0, 9.0}, {0.0, 0.0, 1.0}};
    float stimul[3][3] = {{2.0, 4.0, 4.0}, {6.0, 2.0, 2.0}, {4.0, 2.0, 4.0}};
    float ans[3] = {12.0, 16.0, 20.0};
    float xyz[3] = {};

    float A[3][3] = {};
    float B[3][3] = {};
    float R[3][3] = {};

    srand(time(NULL));

    printf("Init A Matrix\n");
    init_mat(A);
    print_mat(A);

    printf("Init B Matrix\n");
    init_mat(B);
    print_mat(B);

    printf("A + B Matrix\n");
    add_mat(A, B, R);
    print_mat(R);

    printf("A - B Matrix\n");
    sub_mat(A, B, R);
    print_mat(R);

    printf("Matrix Scale(A)\n");
    scale_mat(0.5, A, R);
    print_mat(R);

    printf("AB Matrix\n");
    mul_mat(A, B, R);
    print_mat(R);

    printf("det(A) = %f\n", det_mat(A));
    printf("det(B) = %f\n", det_mat(B));

    printf("\nA^T(Transpose) Matrix\n");
    trans_mat(A, R);
    print_mat(R);

    printf("B^T(Transpose) Matrix\n");
    trans_mat(B, R);
    print_mat(R);
}
```

| | |
|--|---|
| | <pre>printf("A Inverse Matrix\n"); inv_flag = inv_mat(A, R); if(inv_flag) print_mat(R); else printf("역행렬 없다!\n"); printf("test Inverse Matrix\n"); inv_flag = inv_mat(test, R); if(inv_flag) print_mat(R); else printf("역행렬 없다!\n"); printf("크래머 공식 기반 연립 방정식 풀기!\n2x + 4y + 4z = 12\n6x + 2y + 2z = 16\n4x + 2y + 4z = 20\n"); crammer_formula(stimul, ans, xyz); print_vec3(xyz); printf("가우스 소거법 기반 연립 방정식 풀기!(문제 위의 것과 동일함)\n"); gauss_elimination(stimul, ans, xyz); print_vec3(xyz); printf("가우스 소거법으로 역행렬 구하기!\n"); gauss_elim_mat(test, R); print_mat(R); return 0; }</pre> |
|--|---|