ソフトウェア概論 A/B (2018/10/12)

Ver. 1.0

栗野 俊一
kurino@math.cst.nihon-u.ac.jp
http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino/2018/soft/soft.html

2018 年 10 月 12 日

概 要

ソフトウェア概論 A/B[1] の 2018 年 10 月 12 日 の資料 [2]

目 次

[1] http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino/2018/soft/soft.html
[2] http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino/2018/soft/20181012/20181012.html

1 講義資料

1.1 当日の OHP 資料

- 当日の OHP 資料です。
  - 表示用 (HTML)[3]
  - 印刷用 (PDF)[4]

1.2 講義で利用するサンプルプログラム

Download : sample-001.c[5]

リスト 1: sample-001.c

```
/*
 * 2017/07/14 sample-001.c
 */

/*
 * 二分法による代数方程式の解法
 *
 * 利用方法
 *                  コンパイル
 *                          cc -I ~/c/include -c sample-001.c
 *                  リンク
 *                          cc -o sample-001.exe sample-001.c -lm
 *                  実行
 *                          ./sample-001.exe
 */

#include <stdio.h>

#include "s_input.h"
#include "s_print.h"

/*
 * ε : EPSILON (誤差限界:これより小さい時は同じと見做す) の定義
 */

#define EPSILON 0.000001        /* define で ε (EPSILON) を定義 */
                                                        /* 計算結果の精度の尺度になる */
                                                        /* 残念ながら「0」にはできない */
                                                        /* 今回は 0.00001 だが、もちろん、
                                                           0.01 とか 0.00000001 にしてもよ
い */

/*
 *  f(x) = x^3+2x^2-x-2
 */

double f(double x) {
```

[3] ohp/html/index.html
[4] ohp/ohp.pdf
[5] program/sample-001.c

```
          return x * x * x + 2 * x * x - x - 2;
}

/*
 * 二分法で、方程式 f(x)=0 の解 (x0) を求める
 *
 *    min < x0 < max であり f(min)<0, 0<f(max) を仮定する
 *
 *   solove_binary ( double min, double max )
 *
 */

double solve_binary ( double min, double max ) {
        /*
                f(min) < 0 < f(max) (仮定) なので、答えは [min,max] の中にある
                        max-min < EPSILON なら、十分に精度が得られたとする
                        そうでなければ、中点 (mid = (min+max)/2 ) での
                        f の値 ( f(mid) ) の符号をみて、
                                正なら、答えは [min,mid] そうでなければ [mid,max] にある
        */

        if ( (max-min) < EPSILON ) {        /* 十分に狭い範囲にできた */
                return (max+min)/2.0;                /* 中点を答えにする */
        } else if ( f((max+min)/2.0) > 0 ) {        /* まだ大きい */
                return solve_binary ( min, (max+min)/2.0 );
        } else {
                return solve_binary ( (max+min)/2.0, max );
        }
}

/*
 *        main 関数
 */

int main ( void ) {

        s_print_string ( "方程式 f(x)=x^3+2x^2-x-2=0 の解を、区間 [0,3] から探す\n" );

        s_print_string ( "f(0)=-2 < 0, f(3)=40 > 0 なので、[0,3] の中に解がある\n" );

        s_print_string ( "二分法で得られた答えは : " );
        s_print_double ( solve_binary ( 0.0, 3.0 ) );
        s_print_string ( "になりました。\n" );

        s_print_string ( "答えを代入すると " );
        s_print_double ( f( solve_binary ( 0.0, 3.0 ) ) );
        s_print_string ( " なので、ほぼ答えに近い事が分ります\n" );

        return 0;
}
```

Download : sample-002.c[6]

[6] program/sample-002.c

```
$ ./sample-001.exe
方程式 f(x)=x^3+2x^2-x-2=0 の解を、区間 [0,3] から探す
f(0)=-2 < 0, f(3)=40 > 0 なので、[0,3] の中に解がある
二分法で得られた答えは : 1.000000 になりました。
答えを代入すると 0.000001 なので、ほぼ答えに近い事が分ります
$
```

図 1: sample-001.c の実行結果

リスト 2: sample-002.c

```
/*
 * 2017/07/14 sample-002.c
 */

/*
 * リーマン積分
 *
 * 利用方法
 *                  コンパイル
 *                          cc -I ~/c/include -c sample-002.c
 *                  リンク
 *                          cc -o sample-002.exe sample-002.c
 *                  実行
 *                          ./sample-002.exe
 */

/*
 *
 *         定積分
 *      \int_0^1 x^2 dx
 *   を「数値的」に解く
 *
 */

#include <stdio.h>

#include "s_input.h"
#include "s_print.h"

/*
 *
 */

#define FRACTIONAL 1000 /* 区間の等分数 */

/*
 * f(x)=x^2
 */

double f(double x) {

            /*
                 * 引数 x に対して、x の 二乗を値として返す関数
                 */
```

```
          return x * x;
}

/*
 reman_sum ( int i, int n, double min, double max )
        S_i ～ S_{n^1} の和を計算する
*/

double reman_sum ( int i, int n, double min, double max ) {

        if ( i < n )        {        /* まだ計算が必要 */
                /* 注目している短冊の面積と、残りの部分の面積の和を計算する */
                return
                        reman_sum ( i + 1, n, min, max )
                                +
                        f(min+i*(max-min)/n)*(max-min)/n;
        } else {                        /* もう、全て計算した */
                return 0.0;                /* 残る結果は 0 になる */
        }
}

/*
 * リーマン積分
 *
 *   関数の積分値が、小さな幅の短冊の面積の和で近似できる事を利用
 *
 *   solove_reaman ( double min, double max )
 *
 */

double solve_reman ( double min, double max ) {
        /*
       min から max までを積分
       基本は reman_sum に任せる
        */

    return reman_sum ( 0, FRACTIONAL, min, max );

}

/*
 *        main 関数
 */

int main ( void ) {

        s_print_string ( "関数 f(x)=x^2 を区間 [0,1] で数値定積分する。\n" );

        s_print_string ( "リーマンの定義に従って計算した答えは : " );
        s_print_double ( solve_reman ( 0.0, 1.0 ) );
        s_print_string ( "になりました。\n" );

        s_print_string ( "解析的な計算の結果は 1/3 なので、誤差は " );
        s_print_double ( solve_reman ( 0.0, 1.0 ) - 1.0/3.0 );
        s_print_string ( " になり、ほぼ答えに近い事がわかります\n" );

        return 0;
```

```
}
```

```
$ ./sample-002.exe
関数 f(x)=x^2 を区間 [0,1] で数値定積分する。
リーマンの定義に従って計算した答えは : 0.332833 になりました。
解析的な計算の結果は 1/3 なので、誤差は -0.000500 になり、ほぼ答えに近い事がわかります
$
```

図 2: sample-002.c の実行結果

1.3 講議中に作成したプログラム

- 講議中に作成したプログラム [7]

1.4 本日の課題

課題プログラム内の「/*名前:ここ*/」の部分を書き換え「/*この部分を完成させなさい*/」の部分にプログラムを追加して、プログラムを完成させます。

1.4.1 課題 20181012-01 : 一つ浮動小数点数値をキーボードから入力し、その立方根を出力する

Download : 20181012-01.c[8]

リスト 3: 20181012-01.c

```
/*
 * 20181012-01-QQQQ.c
 *
 *        一つ浮動小数点数値をキーボードから入力し、その立方根を出力する
 *        手段としては、「二分法」を使う
 *
 *        コンパイル :
 *                cc -I ~/c/include -c 20181012-01-QQQQ.c
 *                cc -o 20181012-01-QQQQ.exe 20181012-01-QQQQ.o
 *        実行 :
 *                ./20181012-01-QQQQ.exe
 *
 */

#include <stdio.h>

#include "s_input.h"
```

[7] p/
[8] ex/01/q/01.c

```
#include "s_print.h"

/*
 *
 */

#define        EPSILON        0.00000001                /* 誤差幅 */

/*
 * double regula_falsi_cubic_root ( double a, double min, double mid, double max )
 *                double a        立方根の元になる数(正を仮定している)
 *                double min, max 根の入る区間の範囲
 *                double mid        min と mid の中点
 *                return                a 立方根
 *                        二分法により、a の立方根を求める
 *                        0 < min < a の立方根 < max
 */

double regula_falsi_cubic_root ( double a, double min, double mid, double max ) {

        if ( max - min < EPSILON ) {        /* 十分に精度が上った */
                return mid;                                        /* 中点の値を答として返
す */
        } else {                                                /* まだ、狭める必要がある */
                /* min が解のどちら側にあるかを調べ.. それに併せて区間を調整 */
                /* f(x)=x^3-a */
                if ( mid * mid * mid - a < 0.0 ) {        /* f(mid) の符号を確認 */
                                        /* 解の左にあった */

        /*
        **          この部分を完成させなさい
        */

                } else {        /* 解の右にあった */
                        return regula_falsi_cubic_root ( a, min, (min+mid)/2.0, mid );
                }
        }
}


/*
 * double cubic_root ( double a )
 *                double a        立方根の元になる数
 *                return                a 立方根
 *                        a の立方根を求めて結果として返すが、
 *                        計算の基本は、regula_falsi_cubic_root にまかせる
 *                        ここでは、計算の正規化を行う
 */

double cubic_root ( double a ) {

        if ( a < 0.0 ) {                                        /* a が負の数ならば.. */
                /* -a の立方根を計算し、負数を返す */

        /*
        **          この部分を完成させなさい
        */
```

```
        } else if ( a < 1.0 ) {                                 /* a が 1.0 以下なら */

        /*
        **          この部分を完成させなさい
        */

                                                                                /* 立方根
は a と 1.0 の間にある */
        } else {                                                        /* そうでなければ.. */
                return regula_falsi_cubic_root ( a, 1.0, (1.0+a)/2.0, a );
                                                                                /* 立方根
は 1.0 と a の間にある */
        }

}

/*
 * void print_cubic_root ( double a )
 *                 double a         立方根を計算する数
 *                         元の数と、立方根を出力する
 */

void print_cubic_root ( double a ) {

         s_print_double ( a );
         s_print_string ( " の立方根は " );

        /*
        **          この部分を完成させなさい
        */

         s_print_string ( " です。\n" );

}

/*
 *        main
 */

int main( double argc, char *argv[] )
{

        s_print_string ( "実数値を一つ入力してください : " );
        print_cubic_root ( s_input_double() );

        return 0;
}
```

```
12.34
```

図 3: 入力例

```
$ ./20181012-01-QQQQ.exe
実数値を一つ入力してください : 12.340000
12.340000 の立方根は 2.310850 です。
$
```

図 4: 20181012-01.c の実行結果

1.4.2 課題 20181012-02 : CSV

Download : 20181012-02.c[9]

リスト 4: 20181012-02.c

```
/*
 * 20181012-02-QQQQ.c
 *
 *        CSV ファイル内の総計を求める
 *
 *        コンパイル :
 *                cc -I ~/c/include -c 20181012-02-QQQQ.c
 *                cc -o 20181012-02-QQQQ.exe 20181012-02-QQQQ.o
 *        実行 :
 *                ./20181012-02-QQQQ.exe
 *
 */
/*
 *        第一引数で与えられた csv ファイル内の、
 *        A1:J10 (10x10) の行の和を J1:J10 に入れるた結果を
 *  第二引数で与えられた csv ファイルに保存する。
 *
 *        この課題を解く時には、
 *                PNAME/s_csv
 *  に、ファイル 20181012-02-QQQQ.c, 20181012-02.csv を保存し、
 *                make TEST=20181012-02-QQQQ
 *  で、コンパイル、リンク、実行を行う
 *                20181012-DIRout.csv
 *  が作られ、その内容が表示されれば OK
 */

#include <stdio.h>

#include "s_cellio.h"        /* CSV ファイルの操作に必要 */
#include "s_csvio.h"

/*
 * void sum_row ( int row, int sum, int col, int col_max )
 *                row                処理対象になる行番号
 *                sum                ここまでのセルの値の和
 *                col                処理対象になる列番号
 *                col_max        最大の列番号 - 1 であると同時に、総和の保存先の列番号
 */

void        sum_row ( int row, int sum, int col, int col_max )        {
```

[9] ex/02/q/02.c

```
        if ( col < col_max ) {        /* 列番号がまだ、最大値になっていない */
                sum_row ( row, sum + s_get_cell_with_position ( col, row ), col + 1, col_max );
                                                        /* 注目している row, col の値を、*/
                                                        /* sum に加えて、次の列(col+1)へ */
        } else {                                    /* 列が最大を越えたので.. */

        /*
        **          この部分を完成させなさい
        */

                                                        /* その場所に、和を記録する */
        }

}

/*
 * void sum_all_row ( int row, int row_max )
 *                row                処理対象になる行番号
 *                row_max        最大の行番号 - 1
 */

void sum_all_row ( int row, int row_max ) {

         if ( row < row_max ) {        /* 行番号がまだ、最大値になっていない */
                sum_row ( row, 0, 0, 10 );                          /* その行の総和を計算する */

                                                                                        /* 次
の行 (row+1) を計算 */

        /*
        **          この部分を完成させなさい
        */

        } else {                                    /* 行が最大を越えたので.. */
                /* やる事は何もない */
        }
}

/*
 * update_csv
 */

void update_csv ( char *in_file, char *out_file ) {

        s_load_csv ( in_file );                /* 第一引数のファイルからデータを入手 */

        sum_all_row ( 0, 10 );                /* 0 列目から 10 列分の総和を計算 */

        /* 計算結果を第二引数のファイルに保存 */

        /*
        **          この部分を完成させなさい
        */

}

/*
 * main
```

```
 */

int main ( int argc, char *argv[] ) {

        if ( argc == 3 ) {        /* 引数の個数が 2 ( = 3-1 ) の時 */
                update_csv ( argv[1], argv[2] );
                                                /* 第一引数のファイルを変更して第二引数のファ
イルに */
        } else {
                printf ( "ファイル名を二つ指定してください\n" );
        }

  return 0;
}
```

```
$ ./20181012-02-QQQQ.exe
$
```

図 5: 20181012-02.c の実行結果

1.4.3 課題 20181012-03 : 関数

Download : 20181012-03.c[10]

リスト 5: 20181012-03.c

```
/*
 * 20181012-03-QQQQ.c
 *
 *        関数 sin(x) の区間 [0, π/4] の定積
 *
 *        コンパイル :
 *                cc -I ~/c/include -c 20181012-03-QQQQ.c
 *                cc -o 20181012-03-QQQQ.exe 20181012-03-QQQQ.o
 *        実行 :
 *                ./20181012-03-QQQQ.exe
 *
 */

#include <stdio.h>
#include <math.h>                /* 数学的関数 sin を利用するので.. */

#include "s_input.h"
#include "s_print.h"

/*
 * リーマン積分を利用する
 */
```

[10] ex/03/q/03.c

```
#define FRACTIONAL 1000 /* 区間の等分数 */

/*
 * f(x)=sin(x)
 */

double f(double x) {

           /*
                * 引数 x に対して、x の 正弦値 sin(x) を値として返す関数
                */


        /*
        **          この部分を完成させなさい
        */

}

/*
 reman_sum ( int i, int n, double min, double max )
        S_i 〜 S_{n^1} の和を計算する
*/

double reman_sum ( int i, int n, double min, double max ) {

        if ( i < n )        {        /* まだ計算が必要 */
                /* 注目している短冊の面積と、残りの部分の面積の和を計算する */

        /*
        **          この部分を完成させなさい
        */

        } else {                        /* もう、全て計算した */
                return 0.0;                /* 残る結果は 0 になる */
        }
}

/*
 * リーマン積分
 *
 *   関数の積分値が、小さな幅の短冊の面積の和で近似できる事を利用
 *
 *   solove_reaman ( double min, double max )
 *
 */

double solve_reman ( double min, double max ) {
        /*
       min から max までを積分
       基本は reman_sum に任せる
        */

    return reman_sum ( 0, FRACTIONAL, min, max );
}

/*
 *        main 関数
```

```
 */

int main ( void ) {

        s_print_string ( "関数 f(x)=sin(x) を区間 [0, π/4] で数値定積分する。\n" );

        s_print_string ( "リーマンの定義に従って計算した答えは : " );

        /*
        **          この部分を完成させなさい
        */

        s_print_string ( "になりました。\n" );

        s_print_string ( "解析的な計算の結果は 1-√ 2/2 なので、誤差は " );
        s_print_double ( solve_reman ( 0.0, M_PI/4.0 ) - (1.0-sqrt(2.0)/2.0) );
        s_print_string ( " になり、ほぼ答えに近い事がわかります\n" );

        return 0;
}
```

```
$ ./20181012-03-QQQQ.exe
関数 f(x)=sin(x) を区間 [0, π/4] で数値定積分する。
リーマンの定義に従って計算した答えは : 0.292616 になりました。
解析的な計算の結果は 1-√ 2/2 なので、誤差は -0.000278 になり、ほぼ答えに近い事がわかります
$
```

図 6: 20181012-03.c の実行結果