**信号与系统上机第一次实验报告**

班级：162332

学号：16231275

姓名： 刘瀚骋

实验一、连续时间系统卷积的数值计算

# 实验目的

使用C语言编写程序，计算给定两个信号的卷积数值解，并为计算结果绘图。

# 实验原理

卷积是分析数学中一种重要的运算。设{\displaystyle f(x)}{\displaystyle g(x)}是{\displaystyle \mathbb {R} }R上的两个可积函数，作积分

如果{\displaystyle \int \_{-\infty }^{\infty }f(\tau )g(x-\tau )\,\mathrm {d} \tau }{\displaystyle x\in (-\infty ,\infty )}上述积分是存在的。这样随着x{\displaystyle x}的不同取值，这个积分就定义了一个新函数,称为的卷积.{\displaystyle h(x)}hssss {\displaystyle g} 本实验利用积分的定义， 通过求和模拟积分， 计算两个给定信号的数值解。

实现上述卷积计算的程序原理不再赘述。请参考源代码。

# 实验内容

1. 分析问题
2. 编写代码
3. 进行计算
4. 绘制图像

# 实验代码

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include "./convolution.h"

#include <string.h>

void stupidPlot(FILE \*buffer,int sr);

/\*!This is MAGIC!!

\* 由于在C语言中很难动态构造纯函数用于计算, 故定义该Marco,以利用C99标准中的State Expression快速的实现一个闭包.

\* 已在 gcc 5.4.0 中测试通过

\* 部分版本较低的编译器可能无法识别这种语法

\*/

#define BLACK\_MAGIC(c\_) ({ c\_ \_; })

/\*! 计算简单的一重积分

\param inf 积分下界

\param sup 积分上界

\param func 被积函数

\param step 积分步长

\return double 积分结果

采用最朴素的分割-求和-取极限的思路计算积分.

\*/

double integrate(double inf, double sup, double (\*func)(double), double step)

{

double i = 0;

double result = 0;

for (i = inf; i < sup; i += step)

{

if (i + step > sup)

step = sup - i;

result += (\*func)(i)\*step;

}

return result;

}

/\*!计算两个函数的卷积的数值解

\param range1 paor\_double\_double 类型，指明函数1的范围

\param range2 paor\_double\_double 类型，指明函数1的范围

\param func1 要卷积的第一个函数

\param func2 要卷积的第二个函数

\param step 要卷积的部长

\return 返回一个pair类型的数组表示数值结果。数组的长度由卷积区间和卷积部长确定

\*/

pair \*convolution(pair range1, pair range2, double (\*func1)(double), double (\*func2)(double), double step)

{

//确定卷积区间

double inf = range1.first - (range2.second - range2.first);

double sup = range1.second + (range2.second - range2.first);

int outputRange = (sup - inf) / step;

int gov = 1; //gov用于防止数组越界

pair \*pairList = (pair \*)malloc(sizeof(pair) \* (outputRange + 1));

pairList[0].first = (double)outputRange;

for (double t = inf; t <= sup; t += step)

{

if (gov > outputRange)

break;

double (\*func)(double) = BLACK\_MAGIC(double \_(double x) { return (\*func1)(x) \* (\*func2)(t - x); });

pairList[gov].first = t;

pairList[gov].second = integrate(inf, sup, func, 0.01);

gov++;

}

return pairList;

}

/\*!计算两个函数的卷积的数值解,用动画表示!

\param range1 paor\_double\_double 类型，指明函数1的范围

\param range2 paor\_double\_double 类型，指明函数1的范围

\param func1 要卷积的第一个函数

\param func2 要卷积的第二个函数

\param step 要卷积的部长

\return 返回一个pair类型的数组表示数值结果。数组的长度由卷积区间和卷积部长确定

由于C语言不支持默认参数和函数重载,事实上这个函数内的核心计算逻辑同 convolution 函数,仅仅添加了实现动画的一些逻辑.

\*/

pair \*animationConvolution(pair range1, pair range2, double (\*func1)(double), double (\*func2)(double), double step, FILE \*pipe)

{

FILE \*temp = fopen(".convolution", "w");

FILE \*func1buf = fopen(".Signal1", "w");

FILE \*func2buf = fopen(".Signal2", "w");

;

func1buf = func2buf = NULL;

fprintf(pipe, "set terminal dumb 160 60\n");

fprintf(pipe, "set xzeroaxis \n");

fprintf(pipe, "set xrange [-6:9]\n");

fprintf(pipe, "set yrange [-1:6]\n");

fprintf(pipe, "set style data lines \n");

fprintf(pipe, "plot '.convolution' using 1:2 title \"卷积结果\", \\\n");

fprintf(pipe, "'.Signal1' using 1:2 title \"信号1\", \\\n");

fprintf(pipe, "'.Signal2' using 1:2 title \"信号2\" \n");

//确定卷积区间

double inf = range1.first - (range2.second - range2.first);

double sup = range1.second + (range2.second - range2.first);

int outputRange = (sup - inf) / step;

int gov = 1; //gov用于防止数组越界

pair \*pairList = (pair \*)malloc(sizeof(pair) \* (outputRange + 1));

pairList[0].first = (double)outputRange;

for (double t = inf; t <= sup; t += step)

{

if (gov > outputRange)

break;

double (\*func)(double) = BLACK\_MAGIC(double \_(double x) { return (\*func1)(x) \* (\*func2)(t - x); });

double (\*func2Transfer)(double) = BLACK\_MAGIC(double \_(double x) { return (\*func2)(t - x); });

pairList[gov].first = t;

pairList[gov].second = integrate(inf, sup, func, 0.01);

fprintf(temp, "%.3lf %.3lf \n", pairList[gov].first, pairList[gov].second);

func1buf = bufFunc(range1.first, range1.second, step, func1, "Signal1", func1buf);

func2buf = bufFunc(-5, 5, step, func2Transfer, "Signal2", func2buf);

fflush(temp);

printf("%.3lf %.3lf \n", pairList[gov].first, pairList[gov].second);

system("clear");

fprintf(pipe, "replot \n");

fflush(pipe);

usleep(9 \* 1000);

gov++;

}

func1buf = bufFunc(range1.first, range1.second, step, func1, "Signal1", func1buf);

func2buf = bufFunc(range2.first, range2.second, step, func2, "Signal2", func2buf);

fprintf(pipe, "replot \n");

return pairList;

}

/\*!将函数运行结果缓冲入一个文件.

\param inf 函数下界

\param sup 函数上界

\param step 缓冲步长

\param func 要缓冲的函数

\param step 文件描述符,如果为空则会新建一个文件

\return 如果传入的文件描述符非空,则返回该描述符,否则返回一个新的文件描述符

\*/

FILE \*bufFunc(double inf, double sup, double step, double (\*func)(double), char \*funcname, FILE \*buffer)

{

char nameBuf[256];

sprintf(nameBuf, ".%s", funcname);

if (buffer != NULL)

fclose(buffer);

buffer = fopen(nameBuf, "w");

for (double i = inf; i <= sup; i += step)

fprintf(buffer, "%.3lf %.3lf \n", i, (\*func)(i));

fflush(buffer);

return buffer;

}

/\*!heviside函数的极简实现

\*/

double heviside(double x)

{

return x >= 0 ? 1 : 0;

}

int main()

{

FILE \*gnuplotPipe = popen("gnuplot -persistent", "w");

FILE \*result= fopen("./result.csv","w");

int i = 0;

double (\*foo)(double) = BLACK\_MAGIC(double \_(double x) { return heviside(x + 2) - heviside(x - 2); });

double (\*bar)(double) = BLACK\_MAGIC(double \_(double x) { return (x \* (heviside(x) - heviside(x - 2))) + ((4 - x) \* (heviside(x - 2) - heviside(x - 4))); });

pair a = {.first = -3, .second = 3};

pair b = {.first = -0.1, .second = 4.1};

// pair \*Res = animationConvolution(a, b, foo, bar, 0.01, gnuplotPipe);

pair \*Res = convolution(a, b, foo, bar, 0.01);

for (i = 1; i <= (int)Res[0].first; i++)

{

fprintf(result, "%.3lf %.3lf \n", Res[i].first, Res[i].second);

}

fclose(result);

FILE \*fresult = fopen("result.csv", "r");

stupidPlot(fresult,740);

return 0;

}

#define xRange 12 //-4-8

#define yRange 5

#define xzoomCof 10

#define yzoomCof 10

#define yHeight ((yzoomCof\*yRange)+10)

#define xWidth ((xRange\*xzoomCof)+10)

char Canvas[xWidth][yHeight];

#define xOFFSET 4

#define xToCanvasx(x) ((x + xOFFSET) \* xzoomCof)

#define yToCanvasy(y) ((y)\*yzoomCof)

#define CanvasxToX(x) ()

#define CanvasyToy(y) ((double)((double)y/yzoomCof))

void stupidPlot(FILE \*buffer,int stupidRequirements)

{

double a, b;

int i, j;

int stupidCounter=0;

for (i = 0; i < yHeight; i++)

for (j = 0; j < xWidth; j++)

Canvas[j][i] = ' ';

while (fscanf(buffer, "%lf %lf\n", &a, &b) == 2)

{

if (a < -4 || a > 8)

continue;

else if (b > 5)

continue;

else

{

int ca, cb;

ca = (int)xToCanvasx(a);

cb = (int)yToCanvasy(b);

Canvas[ca][cb] = '#';

if(stupidCounter==stupidRequirements){

char buf[10];

sprintf(buf,"(%.1lf,%.1lf)",a,b);

for(i=0;i<9;i++){

Canvas[ca+i+2][cb]=buf[8-i];

}

}

stupidCounter++;

}

}

for (i = yHeight - 1; i >= 0; i--)

{

printf("%.2lf",CanvasyToy(i));

for (j =xWidth-1; j >= 0; j--)

{

printf("%c", Canvas[j][i]);

}

printf("\n");

}

double curindex=-4.0;

for(int i=0;i<xWidth;i++){

if (i==xToCanvasx(curindex)){

printf("%.0lf", curindex);

curindex+=2;

}

else

printf("$");

}

}

# 实验分析

由于积分的数学意义，本次实验编写了签名如下的函数进行积分运算。

double integrate(double inf, double sup, double (\*func)(double), double step)

并采取如下的循环结构， 在给定的精度下用矩形求和替代积分。

for (i = inf; i < sup; i += step)

{

if (i + step > sup)

step = sup - i;

result += (\*func)(i)\*step;

}

此外，我们可以证明两个区间为的信号的卷积结果区间为,故在实现卷积的函数中，要求输入信号的区间，由此确定了卷积积分时的上下限。