组　　号	Z08
实验题目	数值积分
队员姓名	1. 宋昭琰
	2. 徐意
	3. 李宗琳
	4. 向飞宇

目　录

1 实验一：用不同方法求解简单的数值积分

1.1 使用 MatLab 中的内置函数求解

为了起到参照的作用, 我们首先使用 MatLab 中的内置函数 integral（包括 integral2 函数，用于进行二重积分）对积分进行数值求解，由于 integral 函数支持以正负无穷作为积分上下界，我们不需要对积分式进行太多修改. 只是需要将代码中出现幂运算操作的部分修改为点运算操作. 同时，还需要对第三题的积分区域进行变量替换，即:

$$\iint_{x^2+y^2\leqslant 2y}(1+x+y)\,\mathrm{d}x\,\mathrm{d}y=\int_0^{2\pi}\int_0^1(1+(r\cos\theta)+(r\sin\theta+1))r\,\mathrm{d}r\,\mathrm{d}\theta \tag{1}$$

然后再使用 integral 函数，可得:

$$\int_{-\infty}^{\infty}\frac{\mathrm{e}^{-x^2}}{1+x^4}\,\mathrm{d}x\approx 1.4348 \tag{2}$$

$$\int_0^1\frac{\sin x}{\sqrt{1-x^2}}\,\mathrm{d}x\approx 0.8932 \tag{3}$$

$$\iint_{x^2+y^2\leqslant 2y}(1+x+y)\,\mathrm{d}x\,\mathrm{d}y\approx 6.2832 \tag{4}$$

以下采用 Monte Carlo 方法求解上述公式中（2），（4）两式.

1.2 使用 Monte Carlo 方法进行求解

1.2.1 Monte Carlo 方法的原理

Monte Carlo 方法也称为统计模拟法、随机抽样技术、计算机随机模拟方法，是以概率和统计理论方法为基础的一种计算方法[1]，是使用随机数（或更常见的伪随机数）来解决很多计算问题的方法. 将所求解的问题同一定的概率模型相联系，用电子计算机实现统计模拟或抽样，以获得问题的近似解.

本题中，将把积分与在图形上产生的随机点相结合，求出积分，并与命令得出的积分相比较，得出一般性结论. 首先给出一维函数的随机投点计算公式.

定理 1.1 （一维函数的 Monte Carlo 方法） 设 $y=f(x),x\in[a,b]$ 为一个一元（可积）函数，且 $[a,b]$ 为有限积分区间，设:

$$M=\sup_{x\in[a,b]}f(x),\quad m=\inf_{x\in[a,b]}f(x)$$

记总共均匀投入的点数为 N，符合下述条件的点数为 n:

$$m<y<f(x^*),\quad (x^*,y^*)\in[a,b]\times[m,M]$$

则有:

$$\int_a^b f(x)\,\mathrm{d}x=(M-m)(b-a)\frac{n}{N}+m(b-a) \tag{5}$$

注：当函数为连续函数时，上下确界变为最大值和最小值.

证明： Monte Carlo 方法的原理是，在一块二维区域上均匀投入足够的点，总投入点数为 N，符合条件（位于待求区域内部）的点数为 n，二维区域的面积为 A，则待求区域的面积为 $\frac{An}{N}$. 这就是上述定理的基础. 由于定积分所求面积有可能在 x 轴上方或下方，处理起来非常不方便，我们将这个定积分平移到 x 轴上方，即作代换 $g(x) = f(x) + m$. 此时可以使用 $B = [a, b] \times [0, M - m]$ 完全将 $g(x)$ 包围，所以将它作为二维区域，由于 $g(x) \geqslant 0$，它的积分就是 $x = a, x = b, y = g(x), y = 0$ 所围成的面积，所以对于任何投入区域 B 中的点，只要满足 $0 \leqslant y \leqslant g(x)$ 即可.

然后将 $g(x)$ 重新平移为 $f(x)$，相应改变符合条件的判定规则，再增加（或减少）由于平移未计算的面积 $m(b - a)$，就证明了此定理. □

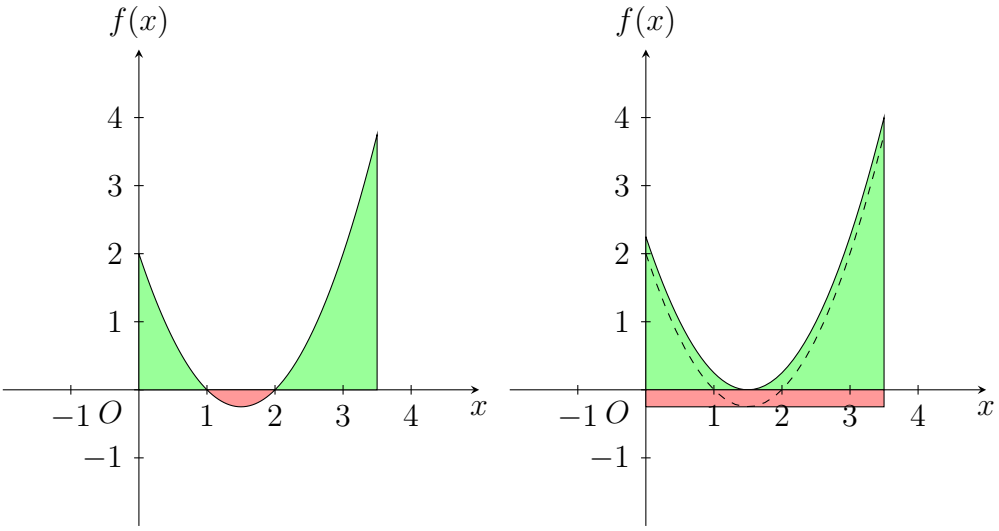

图 1 使用函数的平移证明 Monte Carlo 法

1.2.2 使用 MatLab 计算数值积分

对函数的预处理： 定理（1.1）说明，如果使用随机投点方式的 Monte Carlo 方法，处理的方程需要为一维的，否则将无法求解，所以需要再将（4）转变为一重积分，此时就需要先人工计算出一部分积分的值：

$$\begin{aligned}\iint_{x^2+y^2\leqslant 2y} (1 + x + y)\,\mathrm{d}x\,\mathrm{d}y &= \int_0^{2\pi}\int_0^1 (1 + (r\cos\theta) + (r\sin\theta + 1))r\,\mathrm{d}r\,\mathrm{d}\theta \\ &= \int_0^{2\pi} 1 + \frac{1}{3}\cos\theta + \frac{1}{3}\sin\theta\,\mathrm{d}\theta\end{aligned}$$

其次，处理的积分区间是有限的，所以也需要将（2）进行相应的变量代换. 由于 $y = \arctan(x)$ 将 $[-\infty, +\infty]$ 映射至 $[-\frac{\pi}{2}, \frac{\pi}{2}]$，可以将（2）变为：

$$\int_{-\infty}^{\infty} \frac{\mathrm{e}^{-x^2}}{1 + x^4}\,\mathrm{d}x = \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{\mathrm{e}^{(-\arctan t)^2}}{1 + (\arctan t)^4}\frac{1}{1 + t^2}\,\mathrm{d}t$$

与此同时，注意到，当 x 足够大时，

$$\frac{\mathrm{e}^{-x^2}}{1+x^4}\text{足够小.}$$

这样可以将过大的部分截断，即认为：

$$\int_{-\infty}^{\infty}\frac{\mathrm{e}^{-x^2}}{1+x^4}\,\mathrm{d}x \approx \int_{-M}^{M}\frac{\mathrm{e}^{-x^2}}{1+x^4}\,\mathrm{d}x$$

本题中，我们将同时使用这两种方法对（2）（变量代换取 $M=100$）计算，并比较它们的计算优劣.

计算步骤：

- 求出函数的上下确界（最大值与最小值）：使用 MatLab 中的 fminbnd 函数求出一个函数在某个区间上的最小值，然后根据 $\min\{f\} = -\max\{-f\}$ 定义如下的函数句柄：

```
functmax = @(x) -funct(x);
```

将求解函数的最大值转换为求解函数相反数的最小值；

- 生成点：我们用 MatLab 中的 unifrnd 函数来生成范围在闭区间内的大量均匀分布的随机点. 这里将生成两组，一从积分下限 lb 到积分上限 ub 中随机生成，另一组从积分区域内函数下确界（最小值）m 到上确界（最大值）M 中生成. 这些点分别代表横纵坐标，达到生成在矩形区域上的随机点的目的；
- 计算频率：将符合边界条件的点统计出来，可以得到一个总数，这里将其命名为 freq. 由于一共生成了大量的点，且是均匀随机生成的，故而可以认为频率等于概率；
- 计算面积：根据（5）计算积分区域的面积.

计算结果：

$$\int_{-\infty}^{\infty}\frac{\mathrm{e}^{-x^2}}{1+x^4}\,\mathrm{d}x \approx 1.3559 \quad \text{（使用变量替换）}$$

$$\int_{-\infty}^{\infty}\frac{\mathrm{e}^{-x^2}}{1+x^4}\,\mathrm{d}x \approx 1.4316 \quad \text{（使用截断区间）}$$

$$\iint_{x^2+y^2\leqslant 2y}(1+x+y)\,\mathrm{d}x\,\mathrm{d}y \approx 6.2824$$

可以从以上的计算结果发现，Monte Carlo 方法对结果求解存在一定误差，且相对误差在 5% 以下，而且使用变量替换的误差（5%）远高于使用截断区间的误差（0.8%），这说明当函数值在正负无穷趋于零时，使用**截断区间方法**能够有效减少计算误差.

1.3 实验一的代码

```
% MatLab built-in integral
func1 = @(x) exp(-x.^2) ./ (1 + x.^4);
```

```
func2 = @(x) sin(x) ./ sqrt(1 - x.^2);
func3 = @(theta, r)(1 + cos(theta) + 1 + sin(theta)) .* r;

result1 = integral(func1, -inf, +inf);
result2 = integral(func2, 0, 1);
result3 = integral2(func3, 0, 2 * pi, 0, 1);

% Monte-Carlo method
% question 1: make substitution x = arctan(t) to use finite method
value1 = monte_carlo1(@(t) func1(atan(t)) ./ (1 + t.^2), -pi / 2, pi /
↪  2, 1000000);
(value1 - result1) / result1

% question 1 method 2: using truncating function
value2 = monte_carlo1(func1, -100, 100, 1000000);
(value2 - result1) / result1

% question 3: make polar substitution
value3 = monte_carlo1(func3, 0, 2 * pi, 1000000);
(value3 - result3) / result3

function value = monte_carlo1(funct, lb, ub, total_number)
    [~, minv] = fminbnd(funct, lb, ub);
    functmax = @(x) -funct(x);
    [~, maxv] = fminbnd(functmax, lb, ub);
    maxv = -maxv;
    x = unifrnd(lb, ub, [1, total_number]);
    y = unifrnd(minv, maxv, [1, total_number]);
    freq = sum(y <= funct(x) & y >= minv); % count values which are in
    ↪  minv and maxv
    value = (ub - lb) * (maxv - minv) * freq / total_number + minv * (ub
    ↪  - lb); % calculate value
end
```

2 实验二：具有 3 次、5 次和 7 次代数精度的高斯型积分公式及数值算例

2.1 求高斯型定积分公式的原理

已知数值积分中的 Newton-Cotes 公式[2]:

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{k=0}^{n} a_k f(x_k)$$

在前面的求积公式中虽然求积系数计算方便，但使用了等距求积节点的方法，代数精度会受到限制. 在 n 阶 Newton-Cotes 求积公式中，当 n 为偶数时，其代数精度可以达到 $n+1$ 阶，为了使求积公式的代数精度更高，我们采用高斯型积分公式，按照代数精度最大的原则确定节点.

在这一部分，我们仅讨论对于定积分 $\int_{-1}^{1} f(x)\,\mathrm{d}x$ 具有 3 次、5 次和 7 次代数精度的高斯型积分公式. 注意，具有 $2n+1$ 阶代数精度是指对 $f(x)=1,x,x^2,\cdots,x^{2n+1}$，有

$$\int_{-1}^{1} f(x)\,\mathrm{d}x = \sum_{k=0}^{n} a_k f(x_k) \tag{6}$$

对于代数精度为 $2n+1$ 阶的从 -1 到 1 的高斯型积分公式，我们均有如下的非线性方程组:

$$\int_{-1}^{1} x^i\,\mathrm{d}x = \sum_{k=0}^{n} a_k x_k^i, \quad i=0,1,\cdots,n \tag{7}$$

解上述方程得到 a_i 与 x_i 的解，代入相应参数就得到了所需精度的高斯型积分公式.

2.2 构造求解 Gauss-Legendre 公式的 Matlab 程序

这一部分是一个基本的程序构造，更为普遍的构造将在实验三中具体描述.

首先，对于多项式 x^i 的积分，我们可以由多项式的性质直接得出:

$$\int_{-1}^{1} x^i\,\mathrm{d}x = \begin{cases} 0, & i \text{ 为奇数} \\ \frac{2}{i+1}, & i \text{ 为偶数} \end{cases}$$

然后我们需要对方程组（7）进行求解. 为了方便计算，我们使用符号变量的向量定义法来处理这个问题，由于该方程组是一个多元高阶的方程组，得到多个答案的情况将会发生，所以使用 solve 函数时需要设置 PrincipleValue 为 true，这样求解方程时只会输出一组解. 以下是求解结果:

当代数精度为 3 次时,

$$a_1 = 1, x_1 = \frac{\sqrt{3}}{3}$$

$$a_2 = 1, x_2 = -\frac{\sqrt{3}}{3}$$

当代数精度为 5 次时,

$$
\begin{aligned}
a_1 &= \frac{5}{9}, x_1 = \frac{\sqrt{15}}{5} \\
a_2 &= \frac{5}{9}, x_2 = -\frac{\sqrt{15}}{5} \\
a_3 &= \frac{8}{9}, x_3 = 0
\end{aligned}
$$

当代数精度为 7 次时,

$$
\begin{aligned}
a_1 &= \frac{1}{2} - \frac{\sqrt{30}}{36}, x_1 = \frac{\sqrt{7}\sqrt{\frac{2\sqrt{30}}{5}+3}}{7} \\
a_2 &= \frac{1}{2} - \frac{\sqrt{30}}{36}, x_2 = -\frac{\sqrt{7}\sqrt{\frac{2\sqrt{30}}{5}+3}}{7} \\
a_3 &= \frac{\sqrt{30}}{36} + \frac{1}{2}, x_3 = \sqrt{\frac{3}{7} - \frac{2\sqrt{30}}{35}} \\
a_4 &= \frac{\sqrt{30}}{36} + \frac{1}{2}, x_4 = -\sqrt{\frac{3}{7} - \frac{2\sqrt{30}}{35}}
\end{aligned}
$$

2.3 数值算例

在这一部分，我们以 $\int_{-1}^{1} x^8 \,\mathrm{d}x = \frac{2}{9}$ 为例，观察在不同精度下积分公式的拟合状况. 该积分使用 3 次，5 次与 7 次精度都无法逼近，适合比较计算精度的区别，以下是比较结果:

代数精度	计算结果	相对误差
3	0.0247	88.9%
5	0.1440	35.2%
7	0.2106	5.2%

表 1 对 $\int_{-1}^{1} x^8 \,\mathrm{d}x$ 计算 3 次，5 次与 7 次精度的数值积分

以上的结果比较明显的反映出 Gauss 积分存在的问题，如果待求函数的 Taylor 展开式中，高阶项具有很大的系数，Gauss 积分的误差会很大. 但是随着代数精度的增加，其误差将会不断减小. 考虑到待求的积分高阶项系数一般不会这么极端，在大多数情况下，采用 3 次或 5 次代数精度就可以符合计算要求.

2.4 实验二的代码

```
%% Exercise 2
gauss_legendre_coeff(2)
gauss_legendre_coeff(3)
gauss_legendre_coeff(4)

function gauss_legendre_coeff(total_vars)
    % constructing variables and symbols
    argument = sym('a', [1 total_vars]);
    variable = sym('x', [1 total_vars]);

    % constucting equations
    for i = 1: 2 * total_vars
        if (mod(i, 2) == 1)
            key = sym(2 / i);
        else
            key = sym(0);
        end
        equations(i) = sum(argument .* variable .^ (i - 1)) == key;
    end

    % solving equations
    sol = solve(equations, 'PrincipalValue', true);
    sol = struct2cell(sol);

    % displaying results
    disp('-- Gauss-Legendre coeffcient where order is ', total_vars, '
    ↪ --');
    for i = 1: total_vars
        disp('a', int2str(i), ': ', double(sol{i}), ', x', int2str(i),
        ↪ ': ', double(sol{i + total_vars}));
    end
end
```

3 实验三：具有 3 次、5 次和 7 次代数精度的带权函数积分公式及数值算例

3.1 含权函数高斯求积公式的相关概念

作为对实验二的推广，我们讨论含权函数的高斯求积公式，即考察 $n+1$ 个节点的求积公式，满足：

$$\int_a^b \rho(x)f(x)\,\mathrm{d}x \approx \sum_{i=0}^{n} A_i f(x_i) \tag{8}$$

其中 $\rho(x)$ 为 $[a,b]$ 上的权函数. 当 $\rho(x)=1$ 时，问题转换成了实验二的内容，实验三则是 $\rho(x)$ 取到特定的函数时具有 3 次，5 次与 7 次等代数精度的积分公式.

对于高斯求积公式，$n+1$ 个节点可以使求积公式具有 $2n+1$ 次代数精度. 构造方式是：设节点为 $x_0, x_1, x_2, \cdots, x_n$, 对应的系数为 $a_0, a_1, \cdots, a_n$. 分别令 $f(x) = 1, x, x^2, \cdots, x^{2n+1}$, 求出 $2n+2$ 个 $\int_a^b \rho(x)f(x)\,\mathrm{d}x$ 的解, 第 i 个解与 $a_1x_1^{i-1}+\cdots+a_nx_n^{i-1}$ 相等，构成了 $2n+2$ 个方程组，即为：

$$\int_a^b \rho(x)x^i\,\mathrm{d}x = \sum_{k=0}^{n} a_k x_k^i, \quad i=0,1,\cdots,n$$

可以求出 $2n+2$ 个未知数 $x_0, x_1, \cdots, x_n, a_0, a_1, \cdots, a_n$.

3.2 构造通用求解积分公式的 Matlab 程序

从上述的构造方法，不难设计出一个输入代数精度、权函数、积分区间的函数，使之输出相应代数精度的节点和节点对应的系数. 该函数代码将在下一节展示，此节重点关注函数编写的步骤.

函数缺省和变量定义:

为了和上一题的 Gauss-Legendre 求积公式产生联系，我们将权函数 $\rho(x)$ 的缺省值设为 1，积分下限和上限的缺省值设为 -1 和 1.

我们采取**符号变量的向量定义法**来规定节点和系数, 这样在多未知数的方程组中能将大量的参数输入放置与一个符号变量的数组之中. 由于符号变量只能从下标 1 开始, 我们将原来下标依次加 1，转换成讨论 n 个节点的情况.

方程组的构造和计算:

首先我们把权函数也用符号对象表示. 接下来我们依次求出 $2n$ 个 $\int_a^b \rho(x)f(x)\,\mathrm{d}x$ 的解，第 i 个解与 $a_1x_1^{i-1}+\cdots+a_nx_n^{i-1}$ 相等，构成了 $2n$ 个方程的方程组, 并将由结构体构成的方程组转换成元胞. 然后使用循环和 disp 函数将每个系数和对应节点打印出来. 这样便完成了实验三中对系数的求解工作.

求解结果:

权函数为 $\rho(x)=\sqrt{x}$，积分区间为 $[0,1]$：由于所给权函数在代数精度为 7 次时，并不能像 Gauss-Legendre 计算公式一样得到可以写出公式，我们此处将所有的系数都使用近似小数表示.

当代数精度为 3 次时,

$$a_1=0.38911, x_1=0.82116$$
$$a_2=0.27756, x_2=0.28995$$

当代数精度为 5 次时,

$$a_1=0.23328, x_1=0.90081$$
$$a_2=0.3076, x_2=0.54987$$
$$a_3=0.12578, x_3=0.16471$$

当代数精度为 7 次时,

$$a_1=0.15236, x_1=0.93733$$
$$a_2=0.25253, x_2=0.69895$$
$$a_3=0.1961, x_3=0.37622$$
$$a_4=0.065681, x_4=0.10514$$

权函数为 $\rho(x)=\frac{1}{\sqrt{1-x^2}}$，积分区间为 $[-1,1]$：

当代数精度为 3 次时,

$$a_{1,2}=\frac{\pi}{2}, x_1=\frac{\sqrt{2}}{2}, x_2=-\frac{\sqrt{2}}{2}$$

当代数精度为 5 次时,

$$a_{1,2,3}=\frac{\pi}{3}, x_1=\frac{\sqrt{3}}{2}, x_2=0, x_3=-\frac{\sqrt{3}}{2}$$

当代数精度为 7 次时,

$$a_{1,2,3,4}=\frac{\pi}{4}, x_1=\frac{\sqrt{2+\sqrt{2}}}{2}, x_2=\frac{\sqrt{2-\sqrt{2}}}{2}, x_3=-\frac{\sqrt{2-\sqrt{2}}}{2}, x_4=\frac{\sqrt{2+\sqrt{2}}}{2}$$

权函数为 $\rho(x)=\mathrm{e}^{-x}$，积分区间为 $[0,\infty)$：由于所给权函数在代数精度为 7 次时，并不能像 Gauss-Legendre 计算公式一样得到可以写出公式，我们此处将所有的系数都使用近似小数表示.

当代数精度为 3 次时,

$$a_1=0.14645, x_1=3.1412$$
$$a_2=0.85355, x_2=0.58579$$

当代数精度为 5 次时,

$$a_1 = 0.010389, x_1 = 6.2899$$
$$a_2 = 0.27852, x_2 = 2.2943$$
$$a_3 = 0.71109, x_3 = 0.41577$$

当代数精度为 7 次时,

$$a_1 = 5.3929 \times 10^{-4}, x_1 = 9.3951$$
$$a_2 = 0.038888, x_2 = 4.5366$$
$$a_3 = 0.35742, x_3 = 1.7458$$
$$a_4 = 0.60315, x_4 = 0.32255$$

权函数为 $\rho(x) = \mathrm{e}^{-x^2}$，积分区间为 $(-\infty, +\infty)$:

当代数精度为 3 次时,

$$a_1 = \frac{\sqrt{\pi}}{2}, x_1 = \frac{\sqrt{2}}{2}$$
$$a_2 = \frac{\sqrt{\pi}}{2}, x_2 = -\frac{\sqrt{2}}{2}$$

当代数精度为 5 次时,

$$a_1 = \frac{\sqrt{\pi}}{6}, x_1 = \frac{\sqrt{6}}{2}$$
$$a_2 = \frac{2\sqrt{\pi}}{3}, x_2 = 0$$
$$a_3 = \frac{\sqrt{\pi}}{6}, x_3 = -\frac{\sqrt{6}}{2}$$

当代数精度为 7 次时,

$$a_1 = \frac{\sqrt{\pi}}{4(3-\sqrt{6})}, x_1 = -\sqrt{\frac{3}{2} - \sqrt{\frac{3}{2}}}$$
$$a_2 = \frac{\sqrt{\pi}}{4(3-\sqrt{6})}, x_2 = \sqrt{\frac{3}{2} - \sqrt{\frac{3}{2}}}$$
$$a_3 = \frac{\sqrt{\pi}}{4(3+\sqrt{6})}, x_3 = -\sqrt{\frac{3}{2} + \sqrt{\frac{3}{2}}}$$
$$a_4 = \frac{\sqrt{\pi}}{4(3+\sqrt{6})}, x_4 = \sqrt{\frac{3}{2} + \sqrt{\frac{3}{2}}}$$

3.3 数值算例

与实验二类似，我们以 $f(x)=\frac{x^8}{,}\rho(x)=\sqrt{x},\frac{1}{\sqrt{1-x^2}},e^{-x},e^{-x^2}$ 为例, 计算积分，得到

$$\int_0^1 \sqrt{x}x^8\,\mathrm{d}x = 0.1053$$
$$\int_{-1}^1 \sqrt{1-x^2}x^8\,\mathrm{d}x = 0.8590$$
$$\int_0^\infty \mathrm{e}^{-x}x^8\,\mathrm{d}x = 40320$$
$$\int_{-\infty}^\infty \sqrt{x}x^8\,\mathrm{d}x = 11.631$$

将系数和节点代入函数，所得结果如下表所示:

权函数	代数精度	计算结果	相对误差
$\rho(x)=\sqrt{x}$	3	0.0805	23.6%
$\rho(x)=\sqrt{x}$	5	0.1037	1.5%
$\rho(x)=\sqrt{x}$	7	0.1038	1.4%
$\rho(x)=\frac{1}{\sqrt{1-x^2}}$	3	0.1963	77.2%
$\rho(x)=\frac{1}{\sqrt{1-x^2}}$	5	0.6627	22.9%
$\rho(x)=\frac{1}{\sqrt{1-x^2}}$	7	0.8345	2.9%
$\rho(x)=\mathrm{e}^{-x}$	3	1388.2	96.6%
$\rho(x)=\mathrm{e}^{-x}$	5	25666	36.3%
$\rho(x)=\mathrm{e}^{-x}$	7	39744	1.4%
$\rho(x)=\mathrm{e}^{-x^2}$	3	0.1108	99.1%
$\rho(x)=\mathrm{e}^{-x^2}$	5	2.9901	74.3%
$\rho(x)=\mathrm{e}^{-x^2}$	7	8.9739	22.9%

表 2 对 $\int_a^b \rho(x)x^8\,\mathrm{d}x$ 计算 3 次，5 次与 7 次精度的数值积分

又上表可以看出，权函数为 $\sqrt{x}$ 时，计算的相对误差较为稳定，其余权函数在七次代数精度时的计算误差较小，但在其它代数精度时计算误差较大，这说明带权函数的 Gauss 积分的误差也与 $f(x)$ 的 Taylor 展开式中的高阶系数密切相关.

3.4 补充说明：关于差值型求积公式的理论分析

对于一般的插值型求积公式，我们都能通过数值解法来求解. 但从以上程序的运行速度来看，精度越高的求解速度越慢. 对于特殊的权函数，我们自然可以从数值以外的角度去探索高效率的求解过程，而本实验的 Gauss-Chebyshev、Gauss-Laguerre、Gauss-Hermite 都是著名的多项式的名称，从这个角度切入我们可以对代数精度有更深刻的理解.

定理 3.1 对于插值型求积公式（8），节点 $x_1, \cdots, x_n$ 是 Gauss 点的充分必要条件为：$W_n(x) = (x - x_1) \cdots (x - x_n)$ 与任意次数不超过 n 的多项式 $q(x)$ 正交. 即

$$\int_a^b \rho(x) w_{n+1}(x) q(x) \, \mathrm{d}x = 0 \tag{9}$$

本定理及之后对 Gauss 型积分参数的证明见[3].

由定理(3.2)可知,要使插值型求积公式(8)成为 Gauss 型积分,只要取 $x_k (0 \leq k \leq n)$ 为区间 $[a, b]$ 上带权 $\rho(x)$ 的 $n+1$ 次正交多项式的零点即可.

3.4.1 Gauss-Chebyshev 公式

根据 Chebyshev 多项式的正交性，$\rho(x) = \frac{1}{\sqrt{1-x^2}}$ 与 Chebyshev 多项式族正交，即 $\int_{-1}^{1} T_m(x) T_n(x) \frac{\mathrm{d}x}{\sqrt{1-x^2}} = 0$, 其中 $T_m(x)$ 和 $T_n(x) (m \neq n)$ 均为 Chebyshev 多项式，满足 $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$，且具有如下表达形式：

$$T_n(x) = \cos(n \arccos(x))$$

要求 n 个节点，等价于求 n 次 Chebyshev 多项式的零点，即为 $x_i = \cos((2i-1)/2n)\pi$，通过计算得到

$$a_i = \frac{\pi}{n+1}, \quad i = 1, \cdots, n$$

3.4.2 Gauss-Laguerre 公式

根据 Laguerre 多项式的正交性，$\rho(x) = e^{-x}$ 与 Laguerre 多项式族正交，满足 $\int_0^{+\infty} L_m(x) L_n(x) e^{-x} \, \mathrm{d}x = 0$, 其中 $L_m(x)$ 和 $L_n(x) (m \neq n)$ 均为 Laguerre 多项式, 并且

$$L_n(x) = \frac{e^x}{n!} \frac{\mathrm{d}^n}{\mathrm{d}x^n} (e^{-x} \, \mathrm{d}x)$$

要求 n 个节点，等价于求 n 次 Laguerre 多项式的零点. 对应的

$$a_i = \frac{(n!)^2}{x_i (L_n'(x_i))^2}, \quad i = 1, \cdots, n$$

3.4.3 Gauss-Hermite 公式

根据 Hermite 多项式的正交性，$\rho(x) = e^{-x^2}$ 与 Hermite 多项式族正交，满足 $\int_{-\infty}^{+\infty} H_m(x)H_n(x)e^{-x^2}\,\mathrm{d}x = 0$, 其中 $H_m(x)$ 和 $H_n(x)(m \neq n)$ 均为 Hermite 多项式, 并且

$$H_n(x) = (-1)^n e^{x^2}\frac{\mathrm{d}^n}{\mathrm{d}x^n}(e^{-x^2})$$

要求 n 个节点，等价于求 n 次 Hermite 多项式的零点. 由于 Hermite 多项式满足以下的递归性质:

$$H_0(x) = 1, \frac{\partial}{\partial x}(H_n(x)) = 2nH_{n-1}(x)$$

通过 Hermite 多项式的根，对应的

$$a_i = \frac{2^{n+1}n!\sqrt{\pi}}{(H_n'(x_i))^2}, \quad i = 1, \cdots, n$$

以上的结果说明，只要多项式存在正交性，对应的系数与节点能够很方便的求出, 而且可以通过先求解节点值将求解系数的方程转换为一个线性方程组，这对理论分析与计算误差估计很有帮助. 实际上，Gauss 型求积公式的误差估计为[2]:

$$R(\rho, f) = \frac{f^{(2n)}(\eta)}{(2n)!}\int_a^b \rho(x)(g_n^*(x))^2\,\mathrm{d}x$$

其中

$$g_n^*(x) = \prod_{k=1}^{n}(x - x_k), \quad x_k \text{ 为正交多项式的零点}$$

3.5 实验三的代码

```
generic_integral_coeff(2, @(x) sqrt(x), 0, 1)
generic_integral_coeff(3, @(x) sqrt(x), 0, 1)
generic_integral_coeff(4, @(x) sqrt(x), 0, 1)

generic_integral_coeff(2, @(x) 1 ./ sqrt(1 - x .^ 2), -1, 1)
generic_integral_coeff(3, @(x) 1 ./ sqrt(1 - x .^ 2), -1, 1)
generic_integral_coeff(4, @(x) 1 ./ sqrt(1 - x .^ 2), -1, 1)

generic_integral_coeff(2, @(x) exp(-x), 0, +inf)
generic_integral_coeff(3, @(x) exp(-x), 0, +inf)
generic_integral_coeff(4, @(x) exp(-x), 0, +inf)

generic_integral_coeff(2, @(x) exp(-x .^ 2), -inf, +inf)
```

```
generic_integral_coeff(3, @(x) exp(-x .^ 2), -inf, +inf)
generic_integral_coeff(4, @(x) exp(-x .^ 2), -inf, +inf)

function generic_integral_coeff(total_vars, funct, left, right)
    if (nargin == 1)
        funct = @(x) 1;
    elseif (nargin == 2)
        left = -1;
        right = 1;
    end
    % constructing variables and symbols
    argument = sym('a', [1 total_vars]);
    variable = sym('x', [1 total_vars]);

    % constucting equations
    sym_funct = sym(funct);
    for i = 1: 2 * total_vars
        key = int(sym_funct .* sym(@(x) x .^ (i - 1)), left, right);
        equations(i) = sum(argument .* variable .^ (i - 1)) == key;
    end

    % solving equations
    sol = solve(equations, 'PrincipalValue', true);
    %sol = vpasolve(equations);
    sol = struct2cell(sol);

    % displaying results
    disp("-- Generic integral coeffcient where order is " + total_vars +
    ↪  " --");
    for i = 1: total_vars
        disp("a" + int2str(i) + ": " + double(sol{i}) + ", x" +
        ↪  int2str(i) + ": " + double(sol{i + total_vars}));
    end
end
```

4 实验四：求解两个多边形相交区域的面积

4.1 数据点的输入

首先我们需要能够在 MatLab 的屏幕上确定不同的输入数据点，这里我们使用了 ginput 函数，这个函数的功能是：获得用户鼠标点击时刻所对应的坐标点值，并将其按顺序存放在一个矩阵中，当用户输入回车键时，结束输入进程. 通过连续使用两次 ginput 函数，就能够得到用户输入的所有数据点. 根据这些数据点的坐标，我们可以利用 plot 函数得出相应的多边形，只是使用 plot 时只能得到从第一个点到最后一个点的折线图形，还需要在待画图的点中再次加入第一个点的坐标，以保证图像的闭合性. 除此之外，在两次绘制多边形之间，还需要使用 hold on 函数以防第一次的多边形图像消失，为了提前执行绘制（由于后面的步骤耗时较大，第二个多边形需要提前画出），还需要 drawnow 函数立即进行图像刷新. 以下是数据点输入的示意图：

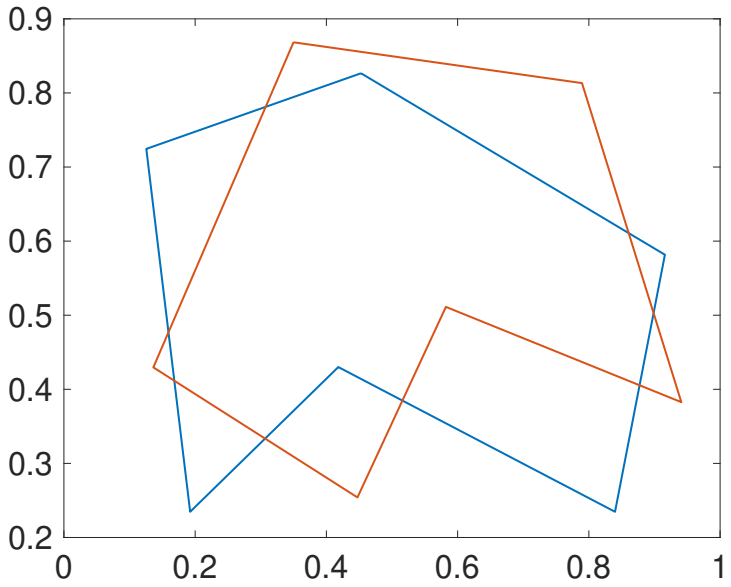

图 2 数据点输入示意图

4.2 点在多边形内的判定

如何判定一个点在一个多边形内是这个问题的关键，一旦能够判断出一个点是否在一个多边形内，就可以通过 Monte Carlo 方法，向规定的区域内投入大量的点，将符合条件的点统计出来，就可以按比例得出两个多边形相交区域的面积了. 我们首先考虑了点在凸多边形内的判定问题.

定理 4.1 （点在凸多边形内的判定） 如果 K 是一个凸多边形，其顶点分别为 $K_i(i =$

$1, 2, \cdots, n)$，P 为平面上任一点，记 $K_{n+1} = K_1$，若 P 满足：

$$\sum_{i=1}^{n} \angle K_i P K_{i+1} = 2\pi \tag{10}$$

则 P 在凸多边形内，否则 P 在凸多边形外.

证明： 首先，含有 n 个顶点的凸多边形的内角和为 $2(n-2)\pi$. 如果一个点在一个凸多边形内部，则它与凸多边形的任何一个顶点相连必能将这个凸多边形分解为 n 个三角形，这些三角形的内角和为 $2n\pi$. 故由平面几何知识可得：

$$\sum_{i=1}^{n} \angle K_i P K_{i+1} = n\pi - (n-2)\pi = 2\pi$$

而如果一个点在凸多边形外部，必然有部分凸多边形的内角不能被完全计算，此时 P 点就不满足上式，这样就证明了这个定理.

注：此处可以单独考虑一下点在凸多边形边界上的情况. 若这个点在边界上，与其相邻的两个顶点构成一个平角（即角度为 π），此时能够构成的三角形个数为 $n-1$，再与之前的平角结合到一起，可知公式（10）仍然成立. 这是一种特殊情况. □

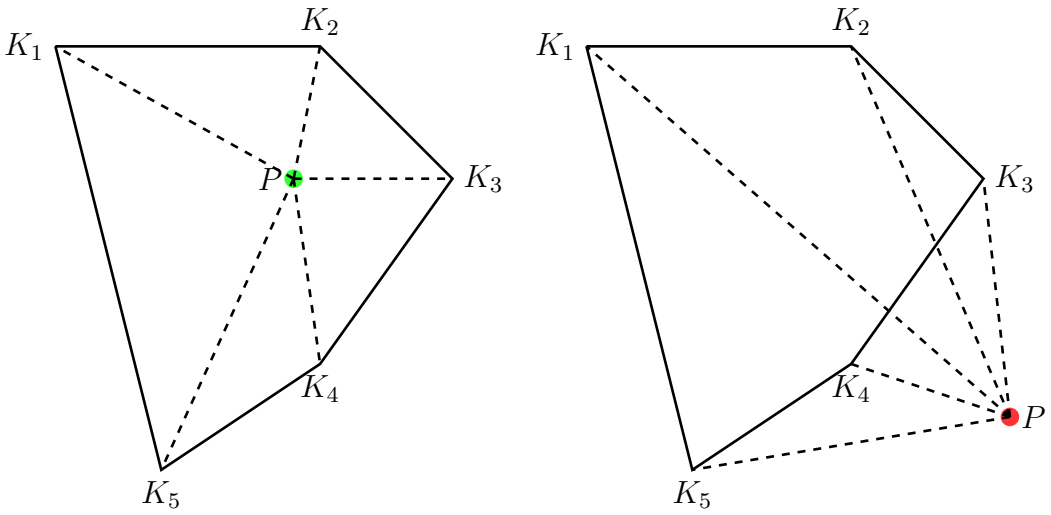

图 3 点在凸多边形内的判定

但是，对于非凸多边形，以上的方法就会失效，因为此时如果继续使用（10）式，就会发现，即使点在多边形的内部，仍然有可能由于不满足（10）式而被舍弃，从而影响求解结果. 实际上，上面的方法仅限于判定一个一般多边形中的一个凸多边形区域，所以我们需要加强一下定理，首先需要在本题中对角度重新定义，此处的角度需要考虑方向，且定义域为 $[-\pi, \pi]$：

定义 4.2 （重新定义的角度） 此处使用旋转定义角度，设 A, O, B 为三个平面中的点，若 A 点能够绕 O 点顺时针旋转 θ 角度（普遍定义的角度，且 $\theta \in [0, \pi]$），使得旋转后

的点 A' 与 O, B 共线，则定义 $\angle\overrightarrow{AOB} = \theta$. 若 A 点能够绕 O 点逆时针旋转 θ 角度（普遍定义的角度，且 $\theta \in [0, \pi]$），使得旋转后的点 A' 与 O, B 共线，则定义 $\angle\overrightarrow{AOB} = -\theta$.

根据此处定义的角度，我们可以给出一个更为普遍的结论：

定理 4.3 （点在一般多边形内的判定） 如果 K 是一个多边形，其顶点分别为 $K_i (i = 1, 2, \cdots, n)$，$P$ 为平面上任一点，记 $K_{n+1} = K_1$，若 P 满足：

$$\left| \sum_{i=1}^{n} \angle\overrightarrow{K_i P K_{i+1}} \right| = 2\pi \tag{11}$$

则 P 在多边形内，否则 P 在多边形外.

证明： 如果 K 是凸多边形，由（10）可知，如果考虑点的旋转方向，公式（10）的结果只会因为 n 个点的顺序为顺时针或逆时针而相应变为 2π 或 -2π，这已经符合公式（11）.

如果 K 是一般的多边形，则必然存在一个有限的分割，将这个多边形划分为多个三角形.（此处的分割指通过一条连接多边形的两个顶点的线段，将这个多边形分为两块区域，由于可以连接的顶点数目有限，这样的分割操作会在有限步之后终止.）

此时，P 点必然位于任何一个三角形之中. 不妨假设多边形 K 的顶点根据顺时针旋转定义，此时需证：

$$\sum_{i=1}^{n} \angle\overrightarrow{K_i P K_{i+1}} = 2\pi$$

考察三角形 ABC（这三个点顺时针旋转），与形外一点 O，如下图所示：

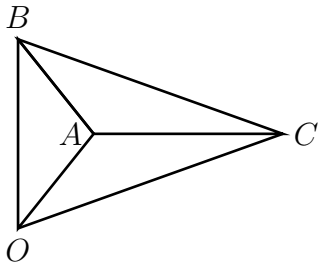

图 4 三角形与其形外一点

由平面几何关系，有：

$$\angle AOC = \angle BOC - \angle AOB$$

转化为新定义的角度关系式，可得：

$$\angle\overrightarrow{AOC} = \angle\overrightarrow{BOC} + \angle\overrightarrow{AOB}$$

上面的结果说明，可以将一个三角形的两条边转换为其另一条边，且此过程可逆.

这样，对于任意的点 P，首先在其处于的三角形 ABC 中得出 $\angle APB + \angle BPC + \angle CPA = 2\pi$，然后将其所有邻接三角形的公共边替换为那些三角形的其它边，经过有限步操作，必然还原成整个多边形，公式（11）成立.

逆时针旋转的定义顶点所得结果与顺时针旋转定义的结果相反，即:

$$\sum_{i=1}^{n} \angle \overrightarrow{K_i P K_{i+1}} = -2\pi$$

综上所述，定理（4.4）得证.

注 1：由定理（4.3）证明的注解可以看出，P 点落在凸多边形的边界上可以看作一种特殊情况，此处不赘述这种特殊情况.

注 2：以上的方法不仅对一般多边形有效，对边相交的多边形（如下图）也可以进行处理，只需要将相交点作为一个新的点，将原多边形看成多个多边形构成的图形即可.

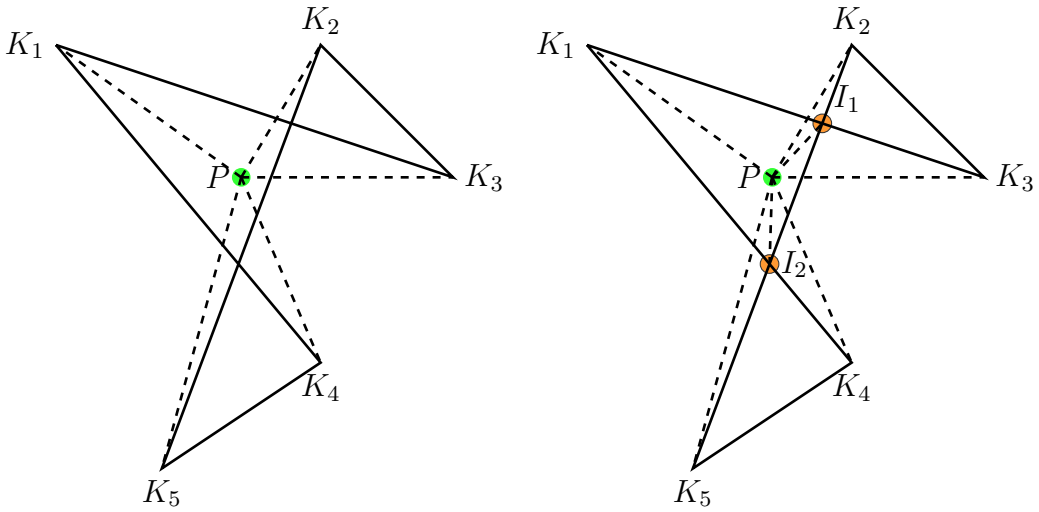

图 5 更普遍的多边形的处理方式

□

4.3 对实际问题的求解

在知道如何判断一个点在一个多边形之内后，我们采用了以下方法求解这个问题:

- Step 1: 使用 ginput 获取用户输入的点, 将两组坐标分别存储在 x_1, y_1, x_2, y_2 中;
- Step 2：计算

$$lx = \min\{\min x_1, \min x_2\}$$

$$rx = \max\{\max x_1, \max x_2\}$$

$$ly = \min\{\min y_1, \min y_2\}$$

$$ry = \max\{\max y_1, \max y_2\}$$

得到随机投点的边界;

- Step 3：随机投入 N 个点，对于每一个投入的点，判断它是否同时满足位于两个多边形之中的条件，此时取计算精度为 $\varepsilon = 10^{-6}$；
- Step 4：记第三步中满足条件的点的总数为 n，则两个多边形的面积重叠区域可以表示为：

$$S = \frac{n}{N}(rx - lx)(ry - ly)$$

以下为一个求解结果：

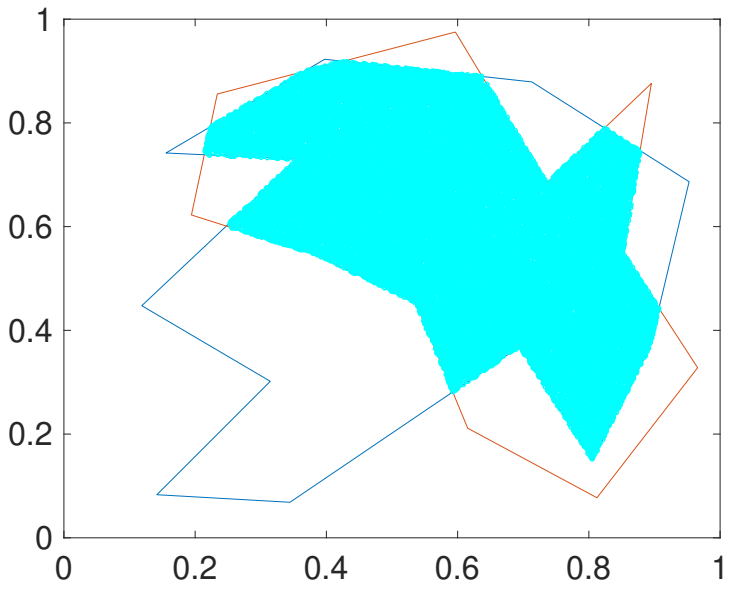

图 6 两个多边形的重叠面积表示

此图中，两个多边形的重叠区域面积为 0.2730.

4.4 实验四的代码

```
% ginput test
[x1, y1] = ginput(); % first polynomial
plot([x1; x1(1)], [y1; y1(1)])
hold on;
[x2, y2] = ginput(); % second polynomial
plot([x2; x2(1)], [y2; y2(1)])
drawnow;
hold on;

square = calculate_square(x1, y1, x2, y2, 100000);
square
```

```
function final_square = calculate_square(x1, y1, x2, y2, total_points,
↪  epsilon)
    if (nargin == 5)
        epsilon = 1e-6;
    end

    x_min = min(min(x1), min(x2));
    x_max = max(max(x1), max(x2));
    y_min = min(min(y1), min(y2));
    y_max = max(max(y1), max(y2));

    total_square = (x_max - x_min) * (y_max - y_min);

    random_points = rand(total_points, 2);
    x_points = x_min * ones(total_points, 1) + (x_max - x_min) *
    ↪  random_points(:, 1);
    y_points = y_min * ones(total_points, 1) + (y_max - y_min) *
    ↪  random_points(:, 2);
    in_points = 0;

    valid = [];
    for i = 1: total_points
        if (interior([x_points(i) y_points(i)], x1, y1, epsilon) &&
        ↪  (interior([x_points(i) y_points(i)], x2, y2, epsilon)))
            in_points = in_points + 1;
            valid = [valid; x_points(i), y_points(i)];
        end
    end

    scatter(valid(:, 1), valid(:, 2), 5, 'cyan');
    hold off;

    final_square = in_points / total_points * total_square;
end
```

```
function result = interior(point, xpoints, ypoints, epsilon)
    % judging a point is an interior of a polynomial
    if (nargin == 3)
        epsilon = 1e-6;
    end

    if (size(xpoints, 1) ~= size(ypoints, 1))
        error("Incompatible points");
    end

    number = size(xpoints, 1);
    angle_sum = 0;
    for i = 1: number
        angle_sum = angle_sum + get_angle(point, [xpoints(i)
        ↪  ypoints(i)], [xpoints(mod(i, number) + 1) ypoints(mod(i,
        ↪  number) + 1)]);
    end
    if (abs(abs(angle_sum) - 2 * pi) < epsilon)
        result = true;
    else
        result = false;
    end
end

function angle = get_angle(point1, point2, point3, epsilon)
    % calculate the angle through point 1 between point 2 and point 3
    if (nargin == 3)
        epsilon = 1e-6;
    end
    angle = acos((point2 - point1) * (point3 - point1)' / (norm(point2 -
    ↪  point1) * norm(point3 - point1)));
    validate_point3 = point1 + (point2 - point1) * [cos(angle)
    ↪  -sin(angle); sin(angle) cos(angle)] / norm(point2 - point1) *
    ↪  norm(point3 - point1);
```

```
    if (norm(validate_point3 - point3) >= epsilon)
        angle = -angle; % reverse rotation
    end
end
```

参考文献

[1] 董秀芳. 蒙特卡罗方法及其基本特点 [J]. 原子能科学技术, 1978, 12(3):277-277.

[2] 郑慧娆, 陈绍林, 莫忠息等. 数值计算方法 [M]. 武汉大学出版社, 2012, 1(2).

[3] 谢彦红. 对高斯求积公式中的系数 Ak 的探讨 [J]. 沈阳化工学院学报, 1999(1):51-54.

附录 A 对实验一的延伸：使用均值估计法扩展积分维度

A.1 均值估计法的介绍

实验一中使用随机投点法，需要使用平移的方法将函数平移到 x 轴上方，但是对于多重积分，这个方法就不太适用了. 所以我们需要引入如下的均值估计法：

定理 A.1 （均值估计法） 多元函数 $y = f(x_1, x_2, \cdots, x_n)$，其定义域为 $A = [0,1]^n$，在 A 内投入 N 个点，这些点的坐标分别为 $(x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)}), i = 1, 2, \cdots, N$，则：

$$\int_0^1 \int_0^1 \cdots \int_0^1 f(x_1, x_2, \cdots, x_n)\,\mathrm{d}x_1\,\mathrm{d}x_2 \cdots \mathrm{d}x_n = \frac{1}{N}\sum_{i=1}^{N} f(x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)})$$

证明： 这个方法具有很强的概率意义. 我们仅在一维情形下证明，即为：

设随机变量 X 的概率密度为 $p(x), 0 \leqslant x \leqslant 1$, $Y = f(X)$ 的期望为: $E(f(X)) = \int_0^1 f(x)p(x)\,\mathrm{d}x$. 若 X 在 $[0,1]$ 均匀分布, 则 $E(f(X)) = \int_0^1 f(x)\,\mathrm{d}x$. □

对于有限的积分区间 $[a,b]$，可以作线性变换，将这个区间转变到 $[0,1]^n$ 上面，即有：

$$\begin{cases} 0 = ma + n \\ 1 = mb + n \end{cases}$$

解得：

$$\begin{cases} m = \frac{1}{b-a} \\ n = \frac{-a}{b-a} \end{cases}$$

所以可以通过换元法，将均值估计法推广至一般有界区间：

定理 A.2 （有界区间的均值估计法） 多元函数 $y = f(x_1, x_2, \cdots, x_n)$，其定义域为 $A = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_n, b_n]$，在 A 内投入 N 个点，这些点的坐标分别为 $(x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)})$，则：

$$\begin{aligned}
&\int_{a_1}^{b_1} \int_{a_2}^{b_2} \cdots \int_{a_n}^{b_n} f(x_1, x_2, \cdots, x_n)\,\mathrm{d}x_1\,\mathrm{d}x_2 \cdots \mathrm{d}x_n \\
&= \prod_{k=1}^{n}(b_k - a_k) \int_0^1 \int_0^1 \cdots \int_0^1 f(a_1 + (b_1 - a_1)x_1, \cdots, a_n + (b_n - a_n)x_n)\,\mathrm{d}x_1\,\mathrm{d}x_2 \cdots \mathrm{d}x_n \\
&= \prod_{k=1}^{n}(b_k - a_k) \cdot \frac{1}{N}\sum_{i=1}^{N} f(a_1 + (b_1 - a_1)x_1^{(i)}, a_2 + (b_2 - a_2)x_2^{(i)}, \cdots, a_n + (b_n - a_n)x_n^{(i)})
\end{aligned}$$

对于无界区间的情况，仍然需要使用变量代换，或者截断函数的方法，将待求积分的区间置于有限区域. 对于积分区域不能按坐标表达的情况，则需要通过极坐标代换等方法，将积分区域变成一个方体.

A.2 均值估计法的计算结果

此时输入的匿名函数需要做一些更改以符合用向量输入一个参数:

```
func3 = @(x) x(2) .* (2 + x(2) * cos(x(1)) + x(2) * sin(x(1)));
```

我们使用了这种方法，对实验一的结果进行了重新计算，得到:

$$\int_{-\infty}^{\infty} \frac{\mathrm{e}^{-x^2}}{1+x^4}\,\mathrm{d}x \approx 1.3571 \quad \text{（使用变量替换）}$$
$$\int_{-\infty}^{\infty} \frac{\mathrm{e}^{-x^2}}{1+x^4}\,\mathrm{d}x \approx 1.4410 \quad \text{（使用截断区间）}$$
$$\iint_{x^2+y^2\leqslant 2y} (1+x+y)\,\mathrm{d}x\,\mathrm{d}y \approx 6.2828$$

从以上的结果可以看出，使用均值估计法投入 1000000 个点时，计算误差比随机投点法更小. 而且它能够适应更高维度，所以在利用概率解决积分问题时，使用均值估计法更好.

A.3 均值估计法的代码

```
% MatLab built-in integral
func1 = @(x) exp(-x.^2) ./ (1 + x.^4);
func2 = @(x) sin(x) ./ sqrt(1 - x.^2);
func3 = @(theta, r) r .* (1 + r.*cos(theta) + 1 + r.*sin(theta));

result1 = integral(func1, -inf, +inf);
result2 = integral(func2, 0, 1);
result3 = integral2(func3, 0, 2 * pi, 0, 1);

result1, result2, result3

func3 = @(x) x(2) .* (2 + x(2) * cos(x(1)) + x(2) * sin(x(1)));

% Monte-Carlo method
% question 1: make substitution x = arctan(t) to use finite method
value1 = monte_carlo(@(t) func1(atan(t)) ./ (1 + t.^2), 1, 1000000, -pi
↪  / 2, pi / 2);
(value1 - result1) / result1
```

```
% question 1 method 2: using truncating function
value2 = monte_carlo(func1, 1, 1000000, -100, 100);
(value2 - result1) / result1

% question 3: make polar substitution
value3 = monte_carlo(func3, 2, 1000000, [0, 0], [2 * pi, 1]);
(value3 - result3) / result3

function value = monte_carlo(funct, dimension, iteration, lb, ub)
    % monte-carlo main method
    points = zeros(dimension, iteration);
    total_value = 0;
    for i = 1: dimension
        points(i, :) = generate_random_points(iteration, lb(i), ub(i));
    end
    for j = 1: iteration
        total_value = total_value + funct(points(:, j)');
    end
    value = total_value * prod(ub - lb) / iteration;
end

function point = generate_random_points(numbers, lb, ub)
    % random point generator by uniform distribution
    point = lb + (ub - lb) .* rand(1, numbers);
end
```

附录 B 对实验二的延伸：使用细分区间方法提高高斯型积分的计算精度

B.1 细分区间的介绍

在 Gauss 积分中，我们发现，对于大多数高阶积分，使用三次代数精度的 Gauss-Legendre 公式并不能求出很好的实验结果，此时就需要对区间进行一定的分割：

$$\int_a^b f(x)\,\mathrm{d}x = \sum_{i=1}^{N} \int_{a+\frac{i-1}{N}\cdot(b-a)}^{a+\frac{1}{N}\cdot(b-a)} f(x)\,\mathrm{d}x$$

通过减小区间长度，在这个区间内的函数更容易被低次的曲线拟合，这样就会降低计算误差.

为了使用 Gauss-Legendre 公式，我们还需要将积分区间变换至 Gauss-Legendre 公式的适用积分区间 $[-1,1]$ 上. 与上一部分的方法相同，对于任意积分区间 $[a,b]$，有：

$$\int_a^b f(x)\,\mathrm{d}x = \frac{b-a}{2}\int_{-1}^{1} f(\frac{a+b}{2}+\frac{b-a}{2}x)\,\mathrm{d}x$$

通过上式的结果，使用 Gauss-Legendre 公式估计积分结果，可得：

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=0}^{n} A_i f(\frac{a+b}{2}+\frac{b-a}{2}x_i), x_i, A_i\text{为原来的系数与节点.}$$

这样就可以得出近似的求解结果.

对于含权函数的情况，由于 Gauss-Laguerre 公式与 Gauss-Hermite 公式的积分区间中均含有无穷，不太适合作区间分割，所以此处我们省略对带权函数的积分的讨论.

B.2 两个数值算例

我们使用三次代数精度的 Gauss-Legendre 公式,取划分区间数为 $10, 100, 1000, 10000$，仍然对 $\int_{-1}^{1} x^8\,\mathrm{d}x$ 进行计算，所得计算结果如下表所示：

分割区间数	计算结果	相对误差
10	0.111103382637346	-6.9556×10^{-5}
100	0.111111110333383	-6.9995×10^{-9}
1000	0.111111111111033	-6.9894×10^{-13}
10000	0.111111111111099	-1.0966×10^{-13}

表 3 对 $\int_{-1}^{1} x^8\,\mathrm{d}x$ 计算分割区间为 $10, 100, 1000, 10000$ 的三次代数精度的数值积分

以上的结果说明，对区间的分割对计算的精确程度有了很大的提升，选用 1000 等分的精度与选用 10000 等分的精度区别已经不大，所以一般选用 1000 等分就能够达到很高的精度要求.

对于有限区间的积分的讨论与上面的数值算例类似，下面考虑一种特殊情况：对无限区间的积分.

我们使用三次代数精度的 Gauss-Legendre 公式，对 $\int_{-\infty}^{\infty} \frac{\mathrm{e}^{-x^2}}{1+x^4}\,\mathrm{d}x$ 采用不同的截断区间进行计算，并根据截断区间的长度采用不同的分割区间数，所得计算结果如下表所示：

近似积分区间	分割区间数	计算结果	相对误差
$[-10, 10]$	10	1.644363655214255	$\approx 14.5\%$
$[-10, 10]$	100	1.434846557526389	$\approx 10^{-9}$
$[-10, 10]$	1000	1.434846557529338	$\approx 10^{-13}$
$[-100, 100]$	100	1.644363655214255	$\approx 14.5\%$
$[-100, 100]$	1000	1.434846557526390	$\approx 10^{-11}$
$[-100, 100]$	10000	1.434846557529337	$\approx 10^{-13}$
$[-1000, 1000]$	1000	1.644363655214255	$\approx 14.5\%$
$[-1000, 1000]$	10000	1.434846557526433	$\approx 10^{-11}$
$[-1000, 1000]$	100000	1.434846557529334	$\approx 10^{-13}$

表 4 对 $\int_{-\infty}^{\infty} \frac{\mathrm{e}^{-x^2}}{1+x^4}\,\mathrm{d}x$ 计算三次代数精度的数值积分

由以上的结果可以看出，近似积分区间越大，得到的结果越精细，但是精细程度由于计算精度的限制也是有限的. 更有意义的发现是，当分割区间数大于近似积分区间长度的 5 倍时，计算结果的误差已经很小，但是在这个比例之下，计算误差会比较大，所以需要选取恰当的比例，以平衡计算速度与精度.

B.3 使用 Gauss-Legendre 公式求解数值积分的代码

```
function result = generic_gauss_integral(funct, lb, ub, density, coeff,
↪  points)
    % generic_gauss_integral calculates generic integrals using
    % Gauss-Legendre integral method.
    % funct: the function to be integrated
    % lb: lower bound of the integral, default = -1
    % ub: upper bound of the integral, default = 1
    % interval: intervals of the integral segmentations, default = 1000
    % coeff: coefficients of the approximation in the integral, default
    ↪  =
    % Gauss-Legendre coefficients of 3-order precision
```

```
    % points: nodes to use in the integral, default = Gauss-Legendre
    ↪  nodes
    % of 3-order precision

    % Parameter completing
    if (nargin == 1)
        lb = -1;
        ub = 1;
        density = 50;
        coeff = [1, 1];
        points = [sqrt(3) / 3, -sqrt(3) / 3];
    elseif (nargin == 3)
        density = 50;
        coeff = [1, 1];
        points = [sqrt(3) / 3, -sqrt(3) / 3];
    elseif (nargin == 4)
        coeff = [1, 1];
        points = [sqrt(3) / 3, -sqrt(3) / 3];
    end

    % Parameter checking
    if (lb > ub) || (density <= 0) || (size(coeff, 2) ~= size(points, 2))
        error('Incompatible parameters');
    end

    interval = (ub - lb) * density;
    result = 0;
    for i = 1: interval
        lb_now = lb + (i - 1) / interval * (ub - lb);
        ub_now = lb_now + (ub - lb) / interval;
        [coeff_a, coeff_b] = interval_transform(lb_now, ub_now, -1, 1);
        result = result + interval_sum(@(x)funct(x), coeff, (points -
        ↪  coeff_b * ones(size(points))) / coeff_a) / coeff_a;
    end
end
```

```
function [coeff_a, coeff_b] = interval_transform(lb, ub, nlb, nub)
    coeff_a = (nub - nlb) / (ub - lb);
    coeff_b = -(nub * lb - nlb * ub) / (ub - lb);
end

function interval = interval_sum(funct, coeff, points)
    interval = 0;
    for i = 1: size(coeff, 2)
        interval = interval + coeff(i) * funct(points(i));
    end
end
```