1 第一题

设 $F_X(x) = P(X \le x)$ 为随机变量 X 的分布函数，$F_X^{-1}(u) = \inf\{x : F_X(x) \ge u\}$ 为其广义逆分布函数。证明：

(a) $\boldsymbol{u \le F_X(x) \iff F_X^{-1}(u) \le x}$.

证明. 设 $A = \{x : F_X(x) \ge u\}$，则 $F_X^{-1}(u) = \inf A$.

($\Rightarrow$) 若 $u \le F_X(x)$，则 $x \in A$，因此 $\inf A \le x$，即 $F_X^{-1}(u) \le x$.

($\Leftarrow$) 若 $F_X^{-1}(u) \le x$，因为 $F_X^{-1}(u) = \inf A$ 和 F_X 是右连续的，所以 $F_X(F_X^{-1}(u)) = F_X(F_X^{-1}(u)^+) = \lim_{\varepsilon \to 0^+} F_X(F_X^{-1}(u) + \varepsilon) \ge u$，因此 $F_X(F_X^{-1}(u)) \ge u$，由于分布函数 F_X 单调不减，$F_X(F_X^{-1}(u)) \le F_X(x)$，所以

$$F_X(x) \ge F_X(F_X^{-1}(u)) \ge u,$$

即 $u \le F_X(x)$.

综上，$u \le F_X(x) \iff F_X^{-1}(u) \le x$. □

(b) **若 U 在 $[0,1]$ 上均匀分布，则 $F_X^{-1}(U)$ 的分布函数为 F_X.**

证明. 设 $Y = F_X^{-1}(U)$，需证：$P(Y \le y) = F_X(y)$.

由 (a) 的结论知：

$$F_X^{-1}(U) \le y \iff U \le F_X(y).$$

因此，

$$P(Y \le y) = P(F_X^{-1}(U) \le y) = P(U \le F_X(y)).$$

又因 $U \sim \text{Uniform}(0,1)$，故对任意 $u \in [0,1]$，有 $P(U \le u) = u$. 于是

$$P(U \le F_X(y)) = F_X(y).$$

即 $P(Y \le y) = F_X(y)$ 得证. □

(c) **若 F_X 连续，则 $F_X(X)$ 在 $[0,1]$ 上均匀分布**.

证明. 设 $U = F_X(X)$，需证：对任意 $u \in [0,1]$，有 $P(U \le u) = u$.

分布函数：

$$P(U \le u) = P(F_X(X) \le u) = P\left(X \le F_X^{-1}(u)\right) = F_X\left(F_X^{-1}(u)\right).$$

由于 F_X 是连续的分布函数，对任意的 $\varepsilon > 0$，

$$F(F^{-1}(y) - \varepsilon) < y \le F(F^{-1}(y) + \varepsilon).$$

令 $\varepsilon \searrow 0$，由 $F(\cdot)$ 的连续性可得 $F(F^{-1}(y)) = y$。

因此

$$F_X\left(F_X^{-1}(u)\right) = u.$$

故 $P(U \le u) = u$ 得证, 即 $U = F_X(X)$ 在 $[0,1]$ 上服从均匀分布. □

2 第二题：蒙特卡洛法求解 ODE

对于

$$\frac{du}{dt}(t,\omega) = -\alpha(\omega)u, \quad u(0,\omega) = \beta(\omega), \quad t \in [0,2].$$

其中 $\beta = 1$，$\alpha \sim N(0,1)$，它的期望解为：$E[Y] = e^{\frac{t^2}{2}}$.

根据 $\alpha \sim N(0,1)$ 的分布生成 M 个 ODE，对于每个 ODE 分别用显式欧拉方法和 Rumge-Kutta 方法求解，取这 M 个解的平均值作为最终的解。代码如下：

```
/// @brief 用蒙特卡洛法求解ODE
/// @param M M个独立同分布的随机数
/// @param N 迭代次数
/// @param func 求解ODE的函数
std::vector<double> ODE_monte_carlo(int M, int N, std::function<std::vector<double>(const solution::
    ODE &, int)> func,
                                    std::mt19937 &generator)
{
    using namespace solution;
    std::normal_distribution<double> normal(0.0, 1.0); // alpha服从标准正态分布
    const double beta = 1.0;

    std::vector<double> res(N, 0); // 存储解的和
    for (int i = 0; i < M; ++i)
    {
        double alpha = normal(generator); // 随机生成alpha
        ODE ode(alpha, beta, 0.0, 2.0, 1.0); // 生成ODE
        std::vector<double> _res = func(ode, N);// 用显式欧拉法/Runge-Kutta方法求解ODE
        res += _res;
    }
    // res /= M; // 在调用处再取平均
    return res;
}
```

根据中心极限定理，蒙特卡洛法的收敛速率应该满足 $\bar{y}(t) - E[y(t)] \sim \frac{1}{\sqrt{M}}$，即误差阶应该在 0.5 附近。

表 1: 显式欧拉法（单次实验）

N	M	L^2-error	L^2-order	L_∞-error	L_∞-order
1000	1000	0.1336	0.0000	0.4878	0.0000
1000	2000	0.1906	-0.5123	0.7602	-0.6402
1000	4000	0.0263	2.8567	0.1304	2.5440
1000	8000	0.0176	0.5766	0.0333	1.9697

表 2: 三阶 Runge-Kutta 法（单次实验）

N	M	L^2-error	L^2-order	L_∞-error	L_∞-order
1000	1000	0.0548	0.0000	0.1085	0.0000
1000	2000	0.1799	-1.7147	0.6337	-2.5466
1000	4000	0.0691	1.3815	0.2772	1.1928
1000	8000	0.0328	1.0745	0.1002	1.4681

如表 1和表 2所示，如果只进行单次实验，即使 M 取了很大的数，误差阶也是混乱的，甚至出现了负数。这可能是因为正态分布的尾巴是无限长的，有一定概率取到非常大的数，导致误差突然剧增，使误差阶无法收敛。

因此，我进行了多组重复独立实验，每组用不同的随机数生成器，并对多组的误差取平均值计算误差阶。

表 3: 显式欧拉法 (N=20)

N	M	L^2-error	L^2-order	L_∞-error	L_∞-order
20	1000	0.7744	0.0000	2.3068	0.0000
20	2000	0.7795	-0.0095	2.3185	-0.0073
20	4000	0.7720	0.0140	2.2970	0.0135
20	8000	0.7725	-0.0010	2.3003	-0.0020

表 4: 三阶 Runge-Kutta 法 (N=20)

N	M	L^2-error	L^2-order	L_∞-error	L_∞-order
20	1000	0.3392	0.0000	1.0425	0.0000
20	2000	0.2516	0.4312	0.7791	0.4201
20	4000	0.1939	0.3753	0.6119	0.3486
20	8000	0.1416	0.4541	0.4484	0.4484

表 3可以看出，对于显式欧拉法来说，当 N 较小时，尽管 M 不断增大，误差几乎没有减小。这说明当 M 足够大时，总误差的瓶颈在于数值方法的精度 N。实际的应用中存在两种误差，一种是第一次作业中讨论的数值方法的误差，另一种是蒙特卡洛法的随机误差，为了更好地研究蒙特卡洛法的误差，需要取一个较大的 N 使得总误差不被数值方法的误差所主导。

此外从表 4我们也可以看出，当取相同的 N 和 M 时，三阶 Runge-Kutta 法的误差远小于显式欧拉法，这是我们在第一次作业中讨论过的。

表 5: 显式欧拉法 (N=1000)

N	M	L^2-error	L^2-order	L_∞-error	L_∞-order
1000	1000	0.349712	0	1.28364	0
1000	2000	0.247885	0.496498	0.9168	0.485559
1000	4000	0.169183	0.551081	0.630646	0.539775
1000	8000	0.116726	0.535464	0.435045	0.535665

表 6: 3 阶 Runge-Kutta 法 (N=1000)

N	M	L^2-error	L^2-order	L_∞-error	L_∞-order
1000	1000	0.3268	0.0000	1.1762	0.0000
1000	2000	0.2381	0.4566	0.8658	0.4420
1000	4000	0.1764	0.4330	0.6517	0.4098
1000	8000	0.1226	0.5244	0.4567	0.5129

最终的结果（如表 5和表 6）表明，误差阶在 0.5 附近，符合我们的理论预期。

3 第三题：蒙特卡洛法求解 PDE

考虑以下偏微分方程:

$$u_t(x,t,Z) = a(Z)u_x(x,t,Z), \quad x \in [0,2\pi],\ t > 0.$$

初值条件为 $u(x,0) = \sin(x)$, 边界条件为周期的。其中 $a(Z) = 1+0.1Z, Z \sim U[-1,1]$。给定 $T = 5, \Delta t/\Delta x = 0.9$, $\Delta x = 2\pi/N$.

由方程的精确解 $\sin(x+at)$ 可以计算出方程的期望解为 $\frac{sin(x+t)sin(0.1t)}{0.1t}$.

与蒙特卡洛法求解 ODE 时相同，根据 a 服从的均匀分布生成 M 个随机数，用 scheme3 求解每个 PDE，最后对这 M 个解取平均值作为最终的解。

```
/// @brief 用蒙特卡洛法求解PDE
/// @param M M个独立同分布的随机数
/// @param N 迭代次数
/// @param func 求解PDE的函数
std::vector<double> PDE_monte_carlo(int M, int N, std::function<std::vector<double>(const solution::
    PDE &, int, const std::string &)> func,
                                    std::mt19937 &generator)
{
    using namespace solution;
    std::uniform_real_distribution<double> uniform(-1.0, 1.0);
    const double T = 5.0;
    const double L = 2 * M_PI;
    const double cfl = 0.9;
    const double k = 1.0;
    std::vector<double> res(N, 0);
    for (int i = 0; i < M; ++i)
    {
        double a = 1 + 0.1 * uniform(generator); // a=1+0.1*Z,Z~U(0,1)
        PDE pde(a, k, T, L, cfl);
        std::vector<double> _res = func(pde, N, "");
        res += _res;
    }
```

```
    // res /= M; // 在调用处再取平均
    return res;
}
```

一开始我和求解 ODE 时一样，希望取一个较大的 N 来使数值方法的误差变小，从而体现蒙特卡洛法的随机误差，但这样计算开销较大，例如进行 100 次重复实验，M 取 1000、2000、4000、8000，N 取 5120 时，使用 32 个线程并行，程序依然运行了一小时左右。于是我选择减少 M 的大小，取 M 为 100、200、400、800，从而放大蒙特卡洛法的随机误差，使其占主导。

表 7: Scheme3

N	M	L^2-error	L^2-order	L_∞-error	L_∞-order
640	100	0.0166	0.0000	0.0235	0.0000
640	200	0.0114	0.5426	0.0161	0.5426
640	400	0.0084	0.4419	0.0119	0.4419
640	800	0.0062	0.4465	0.0087	0.4465

同 ODE 一样，误差阶在 0.5 附近（如表 7），符合理论预期。值得注意的是，L2 误差阶和 L 无穷误差阶几乎完全相同，可能是因为这里计算的是最后一个时间步空间维度上的 L2 误差和 L 无穷误差，在空间维度上，每个空间点统计误差的产生机制是相同的，可以看作是同一随机过程在不同位置的实现，因而具有相似的统计特性。而在解 ODE 的过程中，误差会随时间步积累和传播，因此误差阶不像 PDE 那样几乎完全相同。

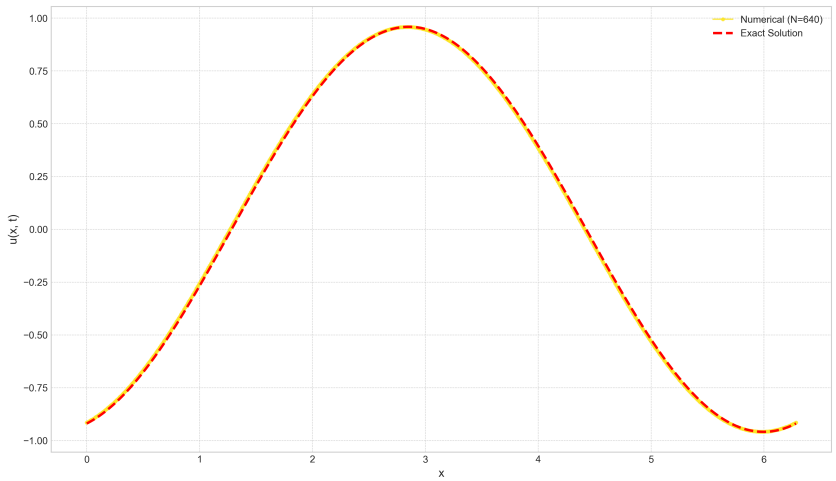

图 1: M=800,N=640 时的数值解与期望解

从图 1可以看出，数值解和期望解相当接近。