1 Galerkin + Runge-Kutta 方法求解 ODE

构建解的 N 阶 gPC 展开：

$$V_N(t,z) = \sum_{i=0}^{N} \hat{V}_i(t)\, H_i(z)$$

代入 ODE 后，作 Galerkin 投影：

$$\mathbb{E}\left[\frac{dV_N}{dt}\, H_k\right] = \mathbb{E}\left[\alpha_N V_N\, H_k\right], \quad k = 0, 1, \ldots, N$$

$$\Rightarrow \quad \Gamma_k \frac{d\hat{V}_k}{dt} = \sum_{i=0}^{N}\sum_{j=0}^{N} a_i \hat{V}_j\, E_{ijk}, \quad k = 0, 1, \ldots, N$$

$$\text{其中} E_{ijk} = \mathbb{E}\left[H_i H_j H_k\right]$$

整理得初值问题：

$$\begin{cases} \dfrac{d\hat{V}_k}{dt} = \dfrac{1}{\Gamma_k}\displaystyle\sum_{i=0}^{N}\sum_{j=0}^{N} a_i \hat{V}_j\, E_{ijk}, \quad k = 0, 1, \ldots, N \\ \hat{V}_k(0) = b_k \end{cases}$$

当正交多项式为 Hermite 多项式且 $\Gamma_k = k!, \quad k \geq 0$ 时：

$$E_{ijk} = \begin{cases} \dfrac{i!\, j!\, k!}{(s-i)!\,(s-j)!\,(s-k)!}, & s \geq i, j, k,\ 2s = i + j + k \\ 0, & \text{其他} \end{cases}$$

上述问题可写为矩阵形式：

令

$$\vec{V} = (\hat{V}_0, \hat{V}_1, \ldots, \hat{V}_N)^T, \quad \vec{b} = (b_0, b_1, \ldots, b_N)^T$$

定义矩阵元素：

$$A_{jk} = \frac{1}{\Gamma_k}\sum_{i=0}^{N} a_i\, E_{ijk}$$

则系统可写为：

$$\begin{cases} \dfrac{d\vec{V}}{dt} = A^T \vec{V} \\ \vec{V}(0) = \vec{b} \end{cases}$$

在本题中，用于构建矩阵 A 的代码如下：

```
// 由于alpha服从标准正态分布，a0 = 0, a1 = 1
    // 这里的A实际上是笔记中的A的转置
    Eigen::MatrixXd A_build_ode(int N)
    {
        int size = N + 1;
        Eigen::MatrixXd A = Eigen::MatrixXd::Zero(size, size);
        if (N == 0)
            return A;
        for (int j = 0; j < size; ++j)
        {
            long long jj = factorial(j); // j!
```

```
            for (int k = 0; k < size; ++k)
            {
                long long gamma = factorial(k); // _k = k!
                if (gamma == 0)
                    continue;
                double e_1jk = 0.0;
                int sum_ijk = 1 + j + k; // i=1
                if (sum_ijk % 2 == 0)
                {
                    int s = sum_ijk / 2; // s = (1+j+k)/2
                    if (s >= 1 && s >= j && s >= k)
                    {
                        e_1jk = static_cast<double>(jj * factorial(k)) / (factorial(s - 1) *
                            factorial(s - j) * factorial(s - k));
                    }
                }
                A(j, k) = (1.0 / gamma) * e_1jk;
            }
        }

        return A;
    }
```

注意上面求出的矩阵 A 还需要加一个负号。使用 Runge-Kutta 方法求解的代码如下：

```
// 这个函数用于计算每一步的 Runge-Kutta 迭代
Eigen::VectorXd rk3_step(const Eigen::VectorXd &y, const Eigen::MatrixXd &A, double dt)
{
    Eigen::VectorXd y1 = y + dt * (A * y);
    Eigen::VectorXd y2 = 0.75 * y + 0.25 * y1 + 0.25 * dt * (A * y1);
    Eigen::VectorXd y_next = (1.0 / 3.0) * y + (2.0 / 3.0) * y2 + (2.0 / 3.0) * dt * (A * y2);
    return y_next;
}
```

在每个时间步迭代步进，即可求得每个时间步时公式中的 $\vec{V}(t) = (\hat{V}_0(t), \hat{V}_1(t), \ldots, \hat{V}_N(t))^T$

1.1 第一小题

当 $\alpha = 0$ 时，准确解为 $y = 1$，数值解（N 分别取 0,1,2,3,4）与准确解的图像如图1所示：

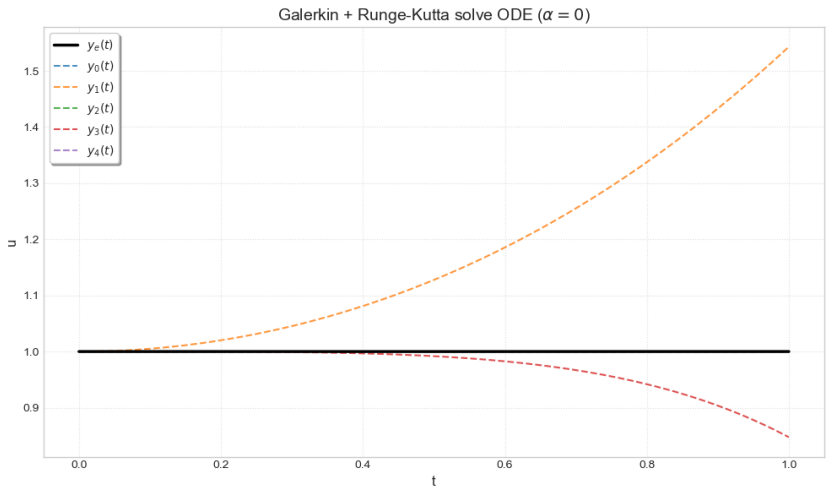

图 1: $\alpha = 0$ 时的数值解与准确解

可以看到除了 $y_1(t)$ 和 $y_3(t)$ 之外，其他阶数的数值解几乎与准确解重合。

这可以通过以下方法验证：对于 Hermite 多项式，当 $\alpha = 0$ 时，Hermite 多项式的取值是固定的：

- $H_0(0) = 1$
- $H_1(0) = 0$
- $H_2(0) = -1$
- $H_3(0) = 0$
- $H_4(0) = 3$

对照程序生成的 gpc 展开系数的 csv 文件简单求和即可验证。

当 Hermite 多项式的阶数为 4 时，gpc 展开系数如图2所示：

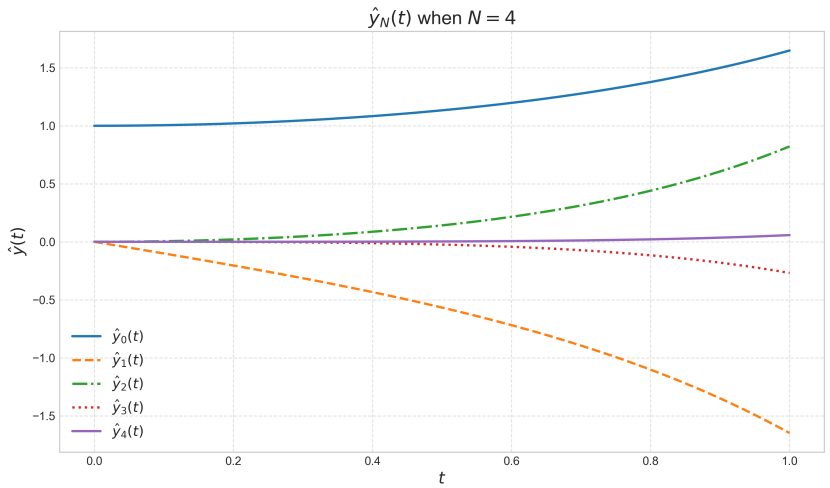

图 2: $N = 4$ 时的 gpc 展开系数

也可以画出数值解的期望值 $\hat{y}_0(t)$ 与准确解的期望 $e^{t^2/2}$ 的图像3：

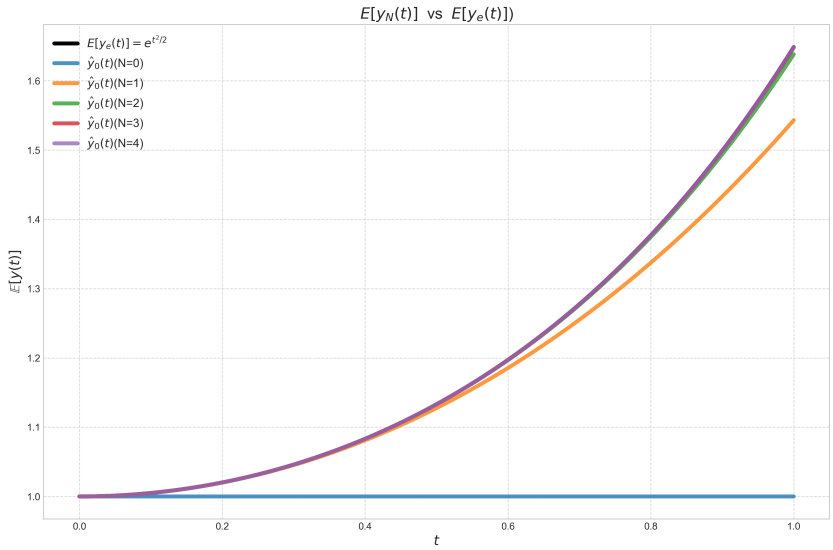

图 3: 数值解与准确解的数学期望

可以看出，当正交多项式的阶数为 0 时，数值解的期望与准确解的期望差距较大，而随着 N 的增大，数值解的期望曲线越来越靠近准确解的期望曲线。当 N=3 和 4 时，两条曲线几乎重合，说明 Galerkin + Runge-Kutta 方法表现良好。

1.2 第二小题

取不同的 N 时上述方法的期望误差 $\varepsilon_{\text{mean}}(t) = \mathbb{E}[y_N(t)] - \mathbb{E}[y_e(t)]$ 与方差误差 $\varepsilon_{\text{var}}(t) = \text{Var}[y_N(t)] - \text{Var}[y_e(t)]$ 如表1所示。

其中

$$\begin{aligned}
\mathbb{E}[y_N(t)] &= \hat{y}_0(t), \\
\mathbb{E}[y_e(t)] &= e^{t^2/2}, \\
\text{Var}[y_N(t)] &= \sum_{i=1}^{N} \hat{y}_i^2 \, \gamma_i, \\
\text{Var}[y_e(t)] &= e^{2t^2} - e^{t^2}.
\end{aligned}$$

表 1: Galerkin + Runge-Kutta 方法下不同多项式阶数 N 对应的误差

N	L^∞ 误差（均值）	L^∞ 误差（方差）
1	0.105238	3.28379
2	0.0101264	1.35206
3	0.000349889	0.385378
4	0.000360579	0.0813799
5	0.000400517	0.0100317
6	0.000402348	0.00349701
7	0.000402419	0.00555611

对误差取对数，图像如4所示：

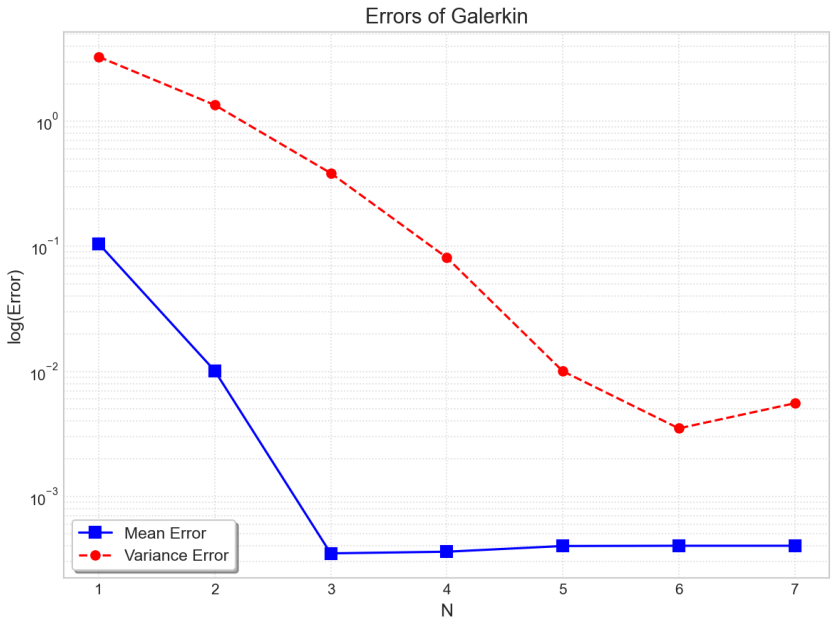

图 4: 不同阶数 N 下的误差对数图像

可以看到，前半段图像曲线接近直线，这是因为 ODE 的解是 $\beta e^{-\alpha t}$ 是完全光滑的，符合 $log(error) = -kN$ 的理论分析。后半段图像曲线几乎水平，说明误差不再下降，可能是因为误差被 Runge-Kutta 方法所主导，因此不再取更多的 N 值。从表中可以看出此时误差已经非常小了。

2 Galerkin + scheme3 方法求解 PDE

与 ODE 类似，构建解的 N 阶 gPC 展开：

$$V_N(x,t,z) = \sum_{i=0}^{N} \hat{V}_i(x,t)\,\phi_i(z)$$

代入原方程并作 Galerkin 投影：

$$\mathbb{E}\left[\frac{\partial V_N}{\partial t}\,\phi_k\right] = \mathbb{E}\left[c(z)\frac{\partial V_N}{\partial x}\,\phi_k\right], \quad k = 0,1,\ldots,N$$

利用基函数的正交关系，我们得到关于展开系数 $\hat{V}_i$ 耦合的方程组：

$$\frac{\partial \hat{V}_k}{\partial t}(x,t) = \sum_{i=0}^{N} a_{ik}\frac{\partial \hat{V}_i}{\partial x}, \quad k = 0,1,\ldots,N$$

其中

$$a_{ik} = \mathbb{E}\left[c(z)\,\phi_i(z)\,\phi_k(z)\right]$$

令 $A = (a_{ij})$ 为一个 $(N+1)\times(N+1)$ 的矩阵，$\vec{V} = (\hat{V}_0, \hat{V}_1, \ldots, \hat{V}_N)^T$，
则方程组可写成矩阵形式：

$$\frac{\partial \vec{V}}{\partial t} = A\frac{\partial \vec{V}}{\partial x}$$

本题构建矩阵 A 的代码如下：

```
Eigen::MatrixXd A_build_pde(int N)
{
```

```
    int size = N + 1;
    Eigen::MatrixXd A = Eigen::MatrixXd::Zero(size, size);

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

        A(i, i) = 1.0; // 注意归一化
        if (i - 1 >= 0)
        {
            A(i, i - 1) += 0.1 * (double)i / (2.0 * i - 1.0);
        }
        if (i + 1 < size)
        {
            A(i, i + 1) += 0.1 * (double)(i + 1) / (2.0 * i + 3.0);
        }
    }
    return A;
}
```

用 scheme3 进行迭代步进的代码如下：

```
std::vector<Eigen::VectorXd> scheme3_step(const std::vector<Eigen::VectorXd> &U_old, const Eigen::
    MatrixXd &A,
double ratio)
{
int size = U_old.size(); // 空间网格数
std::vector<Eigen::VectorXd> U_new(size);
#pragma omp parallel for schedule(static)
for (int j = 0; j < size; ++j)
{
int j_next = (j + 1) % size;
Eigen::VectorXd diff = U_old[j_next] - U_old[j];
// U{j+1} = U{j} + (dt/dx) * A * (U{j+1} - U{j})
U_new[j] = U_old[j] + ratio * A * diff;
}
return U_new;
}
```

对于 PDE，我们只在最后一个时间步上计算它的空间误差。同样计算期望误差与方差误差如表2所示，其中

$$\begin{aligned}
\mathbb{E}[y_N(t)] &= \hat{y}_0(t), \\
\mathbb{E}[y_e(t)] &= \frac{\sin(x+t)\sin(0.1t)}{0.1t}, \\
\mathrm{Var}[y_N(t)] &= \sum_{i=1}^{N} \hat{y}_i^2 \gamma_i, \\
\mathrm{Var}[y_e(t)] &= \frac{1}{2}\left(1 - \cos\big(2(x+t)\big) \cdot \frac{\sin(0.2t)}{0.2t}\right) - \mathbb{E}^2[y_e(t)].
\end{aligned}$$

表 2: 不同 N 下的误差

N	均值误差	方差误差
0	0.0407655	7.92645e-2
1	0.000592188	1.71837e-3
2	0.000362937	7.77646e-5
3	0.000363428	6.37375e-5
4	0.000363427	6.37984e-5

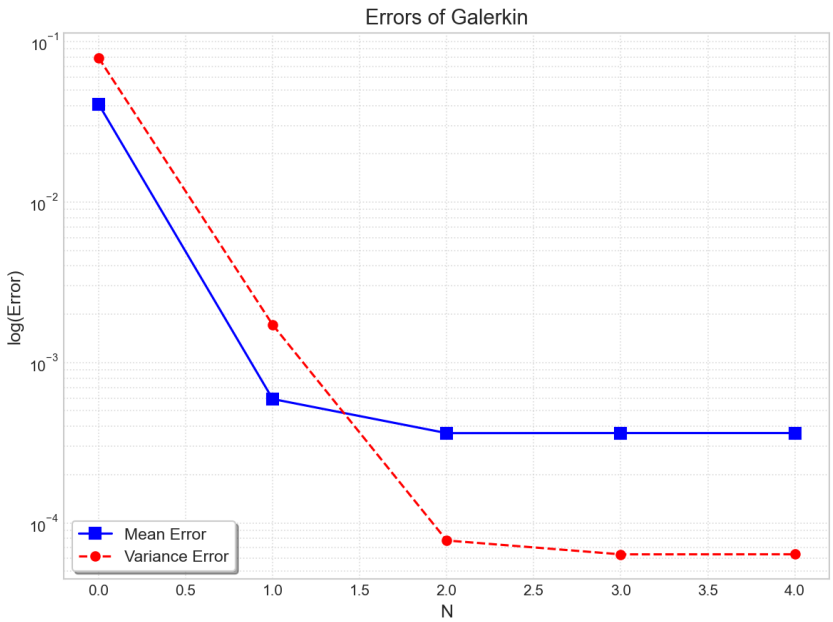

图 5: 不同阶数 N 下的误差对数图像

误差图如5所示，可以看出前半段图像曲线接近直线，这是因为 PDE 的解是 $sin(x+(1+0.1z)t)$ 是完全光滑的，符合 $log(error) = -kN$ 的理论分析。后半段图像曲线几乎水平，说明误差不再下降，可能是因为误差被 scheme3 方法所主导，因此不再取更多的 N 值。从表中同样可以看出此时误差已经非常小了。

最后，我固定正交多项式阶数 $N = 4$, 改变空间离散度，计算不同空间离散度下的误差，如表3所示：

表 3: $N = 4$ 时不同空间离散度的误差

grids	mean_error	mean_order	var_error	var_order
32	0.0902654	0	0.0146441	0
64	0.0460424	0.97121	0.00777968	0.912533
128	0.0231381	0.99269	0.00398804	0.964032
256	0.0116018	0.995919	0.00201817	0.982628

可以看出，期望误差和方差误差的误差阶都接近 1。这是因为 scheme3 的精度是 $O(\Delta t) + O(\Delta x)$，当 $a > 0$ 时，稳定性条件是 $0 \leq \alpha \frac{\Delta t}{\Delta x} = 0.8\alpha \leq 1$，在最后一个时间步，空间上的误差阶是 1 阶的。

3 使用的第三方库和头文件

本次作业使用了 Boost 库 (version 1.89.0), GSL 库 (version 2.8) 和 Eigen 库 (version 4.0), 并启用了 OpenMP 并行计算。

参考网页链接:

- **Boost 库**: `https://www.boost.org/library/latest/math/`
- **GSL 库**: `https://www.gnu.org/software/gsl/doc/html/index.html`