1 随机配点法求解 ODE

随机配点法取高斯积分节点作为配置点，代入原方程解出确定性的 ODE/PDE。

用随机配点法求解 ODE 的代码如下：

```
template <int M>
std::pair<double, double> solve_ode_template(int N)
{
    poly::Hermite<M> hermite(poly::Method::USE_LIBRARY, poly::HermiteType::PROBABILITY);
    const auto &nodes = hermite.get_points();
    const auto &weights = hermite.get_weights();
    int num_points = nodes.size();

    std::vector<double> mean_y(N + 1, 0.0);
    std::vector<double> mean_y2(N + 1, 0.0);

    solution::ODE base_ode(0.0, 1.0, 0.0, 1.0, 1.0);
    double dt = base_ode.get_dt(N);

#pragma omp parallel
    {
        std::vector<double> local_mean_y(N + 1, 0.0);
        std::vector<double> local_mean_y2(N + 1, 0.0);

#pragma omp for
        for (int j = 0; j < num_points; ++j)
        {
            double w_j = weights[j];
            solution::ODE ode(nodes[j], 1.0, 0.0, 1.0, 1.0);
            std::vector<double> u_res = solution::runge_kutta(ode, N);

            double val = ode.get_u0();
            mean_y[0] += val * w_j;
            mean_y2[0] += (val * val) * w_j;

            for (int t = 1; t <= N; ++t)
            {
                val = u_res[t - 1];
                local_mean_y[t] += val * w_j;
                local_mean_y2[t] += (val * val) * w_j;
            }
        }

#pragma omp critical
```

```
        {
            for (int i = 0; i <= N; ++i)
            {
                mean_y[i] += local_mean_y[i];
                mean_y2[i] += local_mean_y2[i];
            }
        }
    }
    auto ode_exact = [](double t) -> std::pair<double, double>
    {
        double ex_mean = std::exp(t * t / 2.0);
        double ex_var = std::exp(2.0 * t * t) - std::exp(t * t);
        return {ex_mean, ex_var};
    };

    return process(N, dt, mean_y, mean_y2,
                   "../HW6/ode_" + std::to_string(N) + "_" + std::to_string(M) + ".csv",
                   ode_exact, 1);
}
```

对于不同的配置点个数，我们可以画出数值解与期望解 $e^{t^2/2}$ 的图像1：

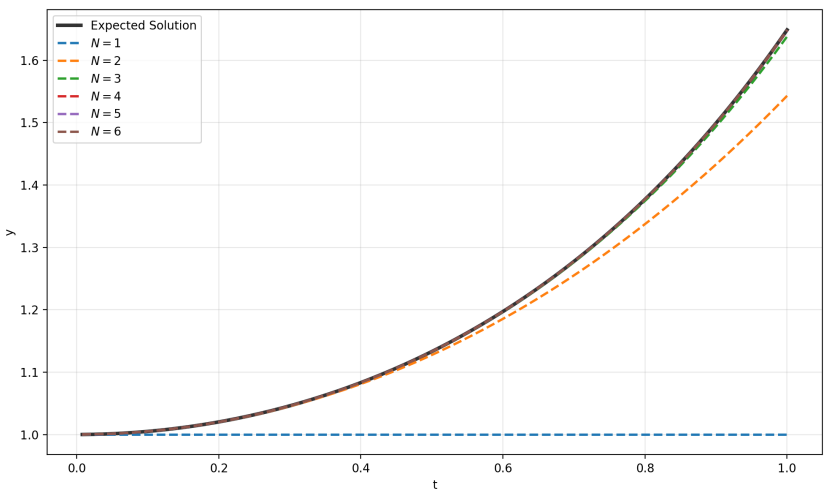

图 1: 不同配置点个数的数值解与期望解

可以看出除了配置点个数为 1 的情况，其他配置点个数下数值解与期望解的误差都较小，当配置点个数大于 3 时数值解与期望解的曲线几乎重合，说明随机配点法表现良好。

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

$$\mathbb{E}[y_e(t)] = e^{t^2/2},$$
$$\text{Var}[y_e(t)] = e^{2t^2} - e^{t^2}.$$

表 1: 不同多项式阶数下的均值与方差误差（L_∞ 范数）

配置点个数	均值误差	方差误差
1	6.487210e-1	4.670770e0
2	1.056410e-1	3.289680e0
3	1.052900e-2	1.357940e0
4	7.525960e-4	3.912670e-1
5	4.215900e-5	8.727290e-2
6	2.225990e-6	1.592670e-2

对误差取对数，图像如2所示：

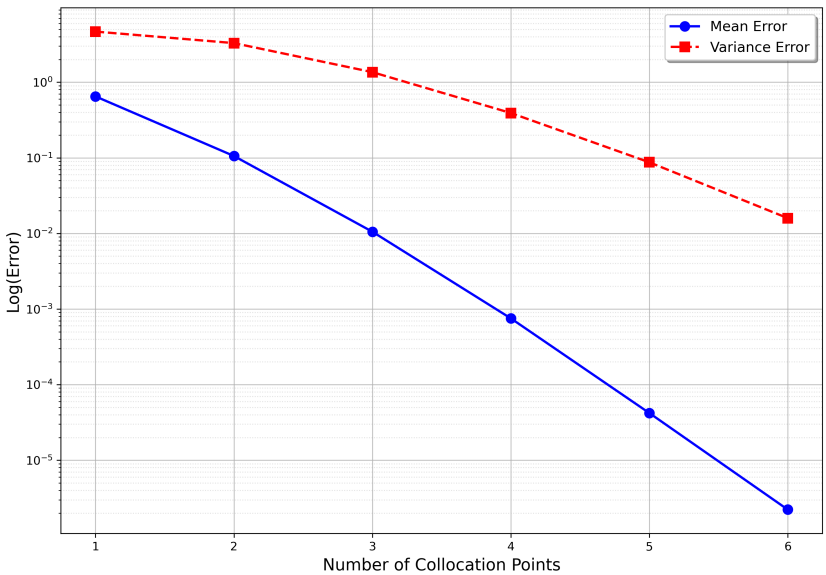

图 2: 不同配置点个数下的误差对数图像

可以看到，图像的曲线接近直线，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。

最后，我固定配置点个数为 10，取不同的时间离散度，得到 Runge-Kutta 方法的误差收敛阶接近 3，这与理论分析相符。结果如表2所示。

表 2: 不同时间步数 N 下的误差收敛阶

N	均值误差	均值误差收敛阶	方差误差	方差误差收敛阶
8	1.038820e-3	0	3.366930e-2	0
16	1.474430e-4	2.81671	4.998640e-3	2.75183
32	1.965150e-5	2.90744	6.833330e-4	2.87088
64	2.536990e-6	2.95345	9.263860e-5	2.88290

2 随机配点法求解 PDE

用随机配点法求解 ODE 的代码如下：

```
template <int M>
std::pair<double, double> solve_pde_template(int N)
{
    poly::Legendre<M> legendre(poly::Method::USE_LIBRARY);
    const auto &nodes = legendre.get_points();
    const auto &weights = legendre.get_weights();
    int num_points = nodes.size();

    std::vector<double> mean_y(N, 0.0);
    std::vector<double> mean_y2(N, 0.0);

    solution::PDE base_pde(1.0, 1.0, 5.0, 2 * poly::pi, 0.8);
    const double dx = base_pde.get_dx(N);
    const double T = base_pde.get_T();
#pragma omp parallel
    {
        std::vector<double> local_mean_y(N, 0.0);
        std::vector<double> local_mean_y2(N, 0.0);

#pragma omp for
        for (int j = 0; j < num_points; ++j)
        {
            double w_j = weights[j] / 2.0;
            solution::PDE pde(1 + 0.1 * nodes[j], 1.0, 5.0, 2 * poly::pi, 0.8);
            std::vector<double> u_res = solution::scheme3(pde, N);

            for (int x = 0; x < N; ++x)
            {
                double val = u_res[x];
                local_mean_y[x] += val * w_j;
                local_mean_y2[x] += (val * val) * w_j;
            }
        }

#pragma omp critical
        {
            for (int i = 0; i < N; ++i)
            {
                mean_y[i] += local_mean_y[i];
                mean_y2[i] += local_mean_y2[i];
            }
        }
    }
```

```
    auto pde_exact = [T](double x) -> std::pair<double, double>
    {
        double exact_mean = std::sin(x + T) * std::sin(0.1 * T) / (0.1 * T);
        double exact_var = 0.5 * (1 - std::cos(2 * (x + T)) * std::sin(0.2 * T) / (0.2 * T)) -
            exact_mean * exact_mean;
        return {exact_mean, exact_var};
    };

    return process(N, dx, mean_y, mean_y2,
                   "../HW6/pde_" + std::to_string(N) + "_" + std::to_string(M) + ".csv",
                   pde_exact, 0);
}
```

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3所示，其中

$$\mathbb{E}[y_e(t)] = \frac{\sin(x+t)\sin(0.1t)}{0.1t},$$
$$\mathrm{Var}[y_e(t)] = \frac{1}{2}\left(1 - \cos\big(2(x+t)\big) \cdot \frac{\sin(0.2t)}{0.2t}\right) - \mathbb{E}^2[y_e(t)].$$

表 3: 不同配置点个数的数值解误差

配置点个数	均值误差	方差误差
1	4.038200e-2	7.926450e-2
2	9.552850e-4	1.657410e-3
3	7.263250e-4	1.412400e-4
4	7.268150e-4	1.274910e-4
5	7.268150e-4	1.275510e-4
6	7.268150e-4	1.275510e-4
7	7.268150e-4	1.275510e-4

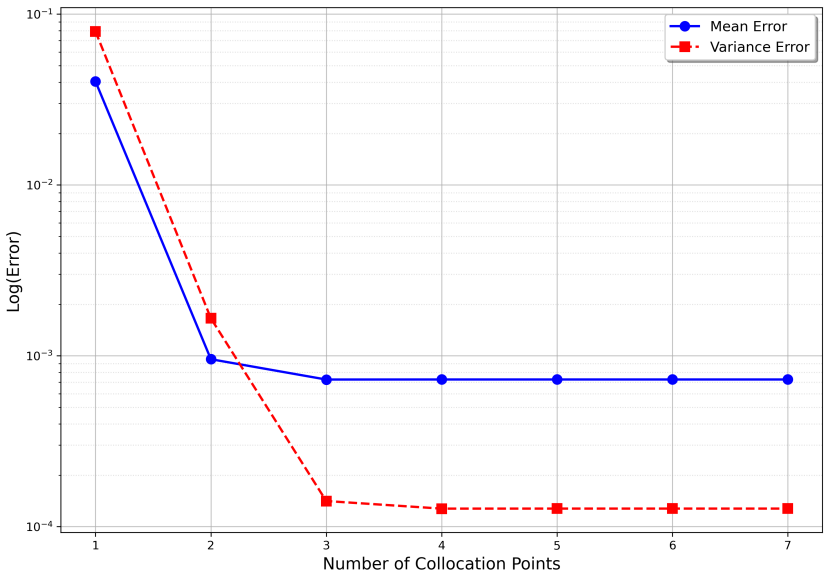

图 3: 不同配置点个数下的误差对数图像

误差图如3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。

我们同样也可以画出数值解与期望解的图像，两者非常接近，几乎重合，如图4所示：

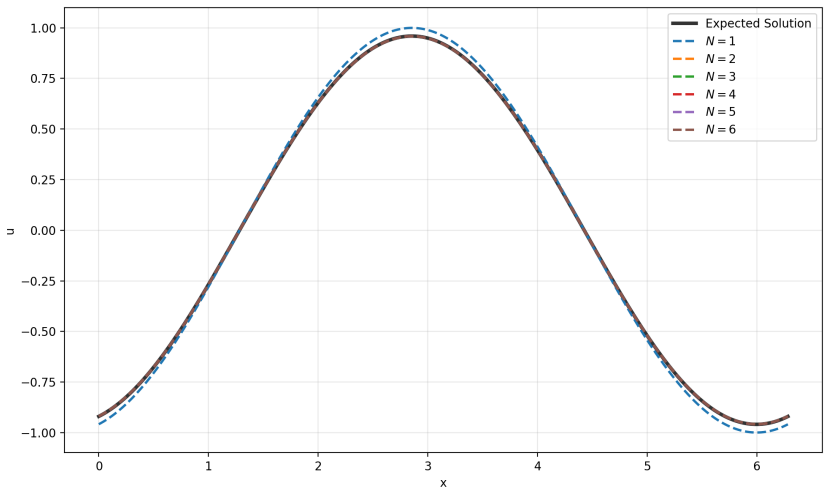

图 4: 数值解与期望解

最后，我固定配置点个数 $N = 10$, 改变空间离散度，计算不同空间离散度下的误差，如表4所示：

表 4: 不同空间网格数 N 下的误差收敛阶

N	均值误差	均值误差阶	方差误差	方差误差阶
32	9.026540e-2	0	1.464410e-2	0
64	4.604240e-2	0.97121	7.779680e-3	0.912533
128	2.313810e-2	0.99269	3.988040e-3	0.964032

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`