



知乎 @D Flip Flop

# 10 月份科研报告

作者：Wenchong Huang

时间：Oct. 10, 2023



# 目录

<b>第 1 章 分离方法</b>	<b>1</b>
1.1 分离方法介绍 . . . . .	1
1.2 二阶时间离散: Strang Splitting . . . . .	1
1.3 四阶时间离散: Forest-Ruth splitting . . . . .	1
1.4 四阶紧致时间离散: Chin splitting . . . . .	2
<b>第 2 章 谱方法</b>	<b>2</b>
2.1 热方程的谱方法 . . . . .	2
2.2 对流扩散方程的有限体-谱分离方法 . . . . .	2
<b>第 3 章 SAV 方法</b>	<b>2</b>
3.1 Cahn-Hilliard 方程的 SAV 方法 . . . . .	2
3.2 GePUP-SAV-SDIRK . . . . .	2
<b>参考文献</b>	<b>2</b>
<b>附录 A 二维 FFT</b>	<b>3</b>
A.1 公式推导 . . . . .	3
A.2 FFTW 库的使用 . . . . .	4
<b>附录 B 本月的心路历程</b>	<b>5</b>

# 第 1 章 分离方法

## 1.1 分离方法介绍

假设我们有初值问题：

$$\frac{d\phi}{dt} = A(\phi) + B(\phi), \quad \phi(0) = \phi_0. \quad (1.1)$$

我们有求解器  $\mathcal{N}_A(\phi_0, T)$ ，它输出初值问题

$$\frac{d\phi}{dt} = A(\phi), \quad \phi(0) = \phi_0. \quad (1.2)$$

在  $T$  时刻的解。同时也有求解器  $\mathcal{N}_B(\phi_0, T)$ ，它输出初值问题

$$\frac{d\phi}{dt} = B(\phi), \quad \phi(0) = \phi_0. \quad (1.3)$$

在  $T$  时刻的解。我们可以借助这两个求解器构造一个分离格式，例如：

$$\begin{aligned} \phi^* &= \mathcal{N}_A(\phi_n, k), \\ \phi_{n+1} &= \mathcal{N}_B(\phi^*, k). \end{aligned}$$

这个格式有两个性质：

1. 只要求解器  $\mathcal{N}_A$  与  $\mathcal{N}_B$  收敛，那么分离格式也收敛；
2. 不论求解器  $\mathcal{N}_A$  与  $\mathcal{N}_B$  的精度有多高，在一些特定问题中，该分离格式也只有一阶精度。例如当  $A, B$  是不交换的线性算子时，可以证明：即使求解器都是精确的，其单步误差也将达到  $O(k^2)$ 。

## 1.2 二阶时间离散：Strang Splitting

我们沿用 (1.1)(1.2)(1.3) 式的符号。Strang Splitting 格式 [3] 如下：

$$\begin{aligned} \phi^* &= \mathcal{N}_A(\phi_n, k/2), \\ \phi^{**} &= \mathcal{N}_B(\phi^*, k), \\ \phi_{n+1} &= \mathcal{N}_A(\phi^{**}, k/2). \end{aligned}$$

为方便后文的表述，我们记由 Strang Splitting 格式算一步得到的解为：

$$\phi_{n+1} = S(\phi_n, k). \quad (1.4)$$

## 1.3 四阶时间离散：Forest-Ruth splitting

我们沿用 (1.4) 式的符号。Forest-Ruth Splitting 格式 [2] 如下：

$$\begin{aligned} \phi^* &= S(\phi_n, \omega_1 k), \\ \phi^{**} &= S(\phi^*, \omega_2 k), \\ \phi_{n+1} &= S(\phi^{**}, \omega_1 k). \end{aligned}$$

其中

$$\omega_1 = \frac{1}{2 - 2^{1/3}}, \quad \omega_2 = -\frac{2^{1/3}}{2 - 2^{1/3}}$$

众所周知，扩散方程的逆过程是不稳定的，而这个格式中居然出现了负时间步，它将会是万恶之源。

## 1.4 四阶紧致时间离散：Chin splitting

我们沿用 (1.1)(1.2)(1.3) 式的符号。同时我们设求解器  $\mathcal{N}_C(\phi_0, \tau, T)$  输出初值问题

$$\frac{d\phi}{dt} = A(\phi) + \frac{\tau^2}{48}(2ABA - AAB - BAA)(\phi), \quad \phi(0) = \phi_0. \quad (1.5)$$

在  $T$  时刻的解。Chin Splitting 格式 [1] 如下：

$$\begin{aligned}\phi^{(1)} &= \mathcal{N}_A(\phi_n, k/6), \\ \phi^{(2)} &= \mathcal{N}_B(\phi^{(1)}, k/2), \\ \phi^{(2)} &= \mathcal{N}_C(\phi^{(2)}, k, 2k/3), \\ \phi^{(4)} &= \mathcal{N}_B(\phi^{(3)}, k/2), \\ \phi_{n+1} &= \mathcal{N}_A(\phi^{(4)}, k/6).\end{aligned}$$

当  $2ABA - AAB - BAA = 0$  时，这是一个相当好的格式。

## 第 2 章 谱方法

### 2.1 热方程的谱方法

### 2.2 对流扩散方程的有限体-谱分离方法

## 第 3 章 SAV 方法

### 3.1 Cahn-Hilliard 方程的 SAV 方法

### 3.2 GePUP-SAV-SDIRK

## 参考文献

- [1] Siu Chin. “Symplectic integrators from composite operator factorizations”. In: *Physics Letters A* 226 (Jan. 1997), pp. 344–348. doi: [10.1016/S0375-9601\(97\)00003-0](https://doi.org/10.1016/S0375-9601(97)00003-0).
- [2] Etienne Forest and Ronald Ruth. “Fourth-order symplectic integration”. In: *Physica D: Nonlinear Phenomena* 43 (May 1990), pp. 105–117. doi: [10.1016/0167-2789\(90\)90019-L](https://doi.org/10.1016/0167-2789(90)90019-L).
- [3] Gilbert Strang. “On the Construction and Comparison of Difference Schemes”. In: *SIAM Journal on Numerical Analysis* 5.3 (1968), pp. 506–517. doi: [10.1137/0705041](https://doi.org/10.1137/0705041). eprint: <https://doi.org/10.1137/0705041>. URL: <https://doi.org/10.1137/0705041>.

## 附录 A 二维 FFT

### A.1 公式推导

#### 引理 A.1

二维 DFT 可归结为如下问题：已知二元多项式

$$f(x, y) = \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} a_{n,m} x^n y^m,$$

求  $f(\omega_N^j, \omega_N^k)$ ,  $(j, k = 0, \dots, N-1)$  的值。



引理很容易验证，关键是如何快速求解这个问题。为此，我们对多项式做奇偶项划分，即令

$$\begin{aligned} p_1(x, y) &= \sum_{n=0}^{N/2-1} \sum_{m=0}^{N/2-1} a_{2n,2m} x^n y^m, \\ p_2(x, y) &= \sum_{n=0}^{N/2-1} \sum_{m=0}^{N/2-1} a_{2n,2m+1} x^n y^m, \\ p_3(x, y) &= \sum_{n=0}^{N/2-1} \sum_{m=0}^{N/2-1} a_{2n+1,2m} x^n y^m, \\ p_4(x, y) &= \sum_{n=0}^{N/2-1} \sum_{m=0}^{N/2-1} a_{2n+1,2m+1} x^n y^m. \end{aligned}$$

于是我们有

$$f(x, y) = p_1(x^2, y^2) + yp_2(x^2, y^2) + xp_3(x^2, y^2) + xyp_4(x^2, y^2).$$

现任取整数  $j, k \in [0, N/2)$ ，注意到

$$\begin{aligned} f(\omega_N^j, \omega_N^k) &= p_1(\omega_{N/2}^j, \omega_{N/2}^k) + \omega_N^k p_2(\omega_{N/2}^j, \omega_{N/2}^k) + \omega_N^j p_3(\omega_{N/2}^j, \omega_{N/2}^k) + \omega_N^j \omega_N^k p_4(\omega_{N/2}^j, \omega_{N/2}^k) \\ f(\omega_N^j, \omega_N^{N/2+k}) &= p_1(\omega_{N/2}^j, \omega_{N/2}^k) - \omega_N^k p_2(\omega_{N/2}^j, \omega_{N/2}^k) + \omega_N^j p_3(\omega_{N/2}^j, \omega_{N/2}^k) - \omega_N^j \omega_N^k p_4(\omega_{N/2}^j, \omega_{N/2}^k) \\ f(\omega_N^{N/2+j}, \omega_N^k) &= p_1(\omega_{N/2}^j, \omega_{N/2}^k) + \omega_N^k p_2(\omega_{N/2}^j, \omega_{N/2}^k) - \omega_N^j p_3(\omega_{N/2}^j, \omega_{N/2}^k) - \omega_N^j \omega_N^k p_4(\omega_{N/2}^j, \omega_{N/2}^k) \\ f(\omega_N^{N/2+j}, \omega_N^{N/2+k}) &= p_1(\omega_{N/2}^j, \omega_{N/2}^k) - \omega_N^k p_2(\omega_{N/2}^j, \omega_{N/2}^k) - \omega_N^j p_3(\omega_{N/2}^j, \omega_{N/2}^k) + \omega_N^j \omega_N^k p_4(\omega_{N/2}^j, \omega_{N/2}^k) \end{aligned}$$

因此，只要我们先算出

$$p_i(\omega_{N/2}^j, \omega_{N/2}^k), (j, k = 0, \dots, N/2-1), \quad i = 1, 2, 3, 4. \quad (\text{A.1})$$

我们就可以再用  $O(N)$  的时间完成问题求解。不难看出 (A.1) 式的形式与原问题完全一致，只是  $N$  换成了  $N/2$ 。我们设求解原问题的时间复杂度为  $T(N)$ ，有：

$$T(N) = 4T\left(\frac{N}{2}\right) + \Theta(N) = \Theta(N^2 \log N).$$

因此我们可以借助分治实现高效求解。但如果使用递归式写法，我们需要在每一层存储奇偶项划分后  $p_1, p_2, p_3, p_4$  的各项系数，从而空间复杂度也达到  $\Theta(N^2 \log N)$ 。我们可以用蝴蝶变换来避免递归，即自底向上模拟分治过程，空间复杂度可以下降到  $\Theta(N^2)$ 。详见代码 `fft2D.cpp`。

## A.2 FFTW 库的使用

FFTW(FFT in the West) 库曾经是世界上最快的 FFT 库，不过现在 Intel 用 mkl 优化后的 FFTW 要更快。以 2 维 FFT 为例，在输入输出均为复数数组的情况下，速度大约是我手写 FFT 的 1.2 倍。

注意到一个问题：FFT 正变换时，输入数组为实数、输出数组为复数；FFT 逆变换时，输入数组为复数、输出数组为实数。FFTW 能利用这个性质，进一步优化 FFT。下面是一个使用 FFTW 的例程。

```
#include <bits/stdc++.h>
#include <fftw3.h>
using namespace std;

const int N = 1024;

int main(){
    auto out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N*(N/2+1));
    auto in = (double*) fftw_malloc(sizeof(double) * N*N);
    auto p = fftw_plan_dft_r2c_2d(N, N, in, out, FFTW_MEASURE);
    auto pinv = fftw_plan_dft_c2r_2d(N, N, out, in, FFTW_MEASURE);

    for(int i = 0; i < N*N; i++)
        in[i] = (double)rand()/RAND_MAX;

    // Run FFT and save to out[]
    fftw_execute(p);

    // Run IFFT and save to in[]
    fftw_execute(pinv);

    return 0;
}
```

实际测试表明，针对输入输出数组的类型优化后的 FFTW，其运行速度是我手写 FFT 的 2.4 倍。

## 附录 B 本月的心路历程

本月搞懂了三维不规则有限体程序的脉络，并着手处理了一些程序细节。但是两位学长几乎没有给我安排代码任务，导致我在做完手头工作后有了空闲时间，于是开始读 GePUP-SAV 的文章。阅读文章的过程中，我看不懂 SAV，然后就去找了 SAV 的原始论文来看。看完之后感觉很顺畅，于是想把文章中的数值测试简单复现一下。然后就开始了“递归式”的学习。

我发现文章中是用谱方法离散的，然后去看了一下谱方法。我发现谱方法要用到 FFT，但我只学过 1 维 FFT，网上也找不到 2 维 FFT 的资料，索性自己按 1 维的思路推了一遍。然后我用谱方法写了个二维规则周期区域热方程的求解器。有了这个求解器之后我又想把它用在对流扩散方程里，但非线性对流项在做 Fourier 变换之后反而比原来更复杂了。于是我就想用有限体处理对流项、用谱方法处理扩散项。为了把他们耦合起来，我又去看了些分离方法的文章。

看了一些文章，发现分离方法似乎现在已经没什么人研究了，于是回头把 SAV 原始文章里的数值测试复现了一下，接着回头读 GePUP 文章的 SAV 部分，然后这个月就这么过去了。