技术细节

编程时推荐使用 PETSc 函数库，因为方便并行化编程，而且可以方便的进行防御型编程，有直接对程序进行检查的 PetscErrorCode 参数，当程序出错时会打印出错误信息。

当对 HDF5 文件进行并行读写时，要确定 HDF5 支持并行读写，确定在安装 HDF5 时，parallel HDF5 的选项是打开的，否则会无法进行并行读写。

我们根据给出的初始条件和 PDE 可以得出显式和隐式的差分格式。

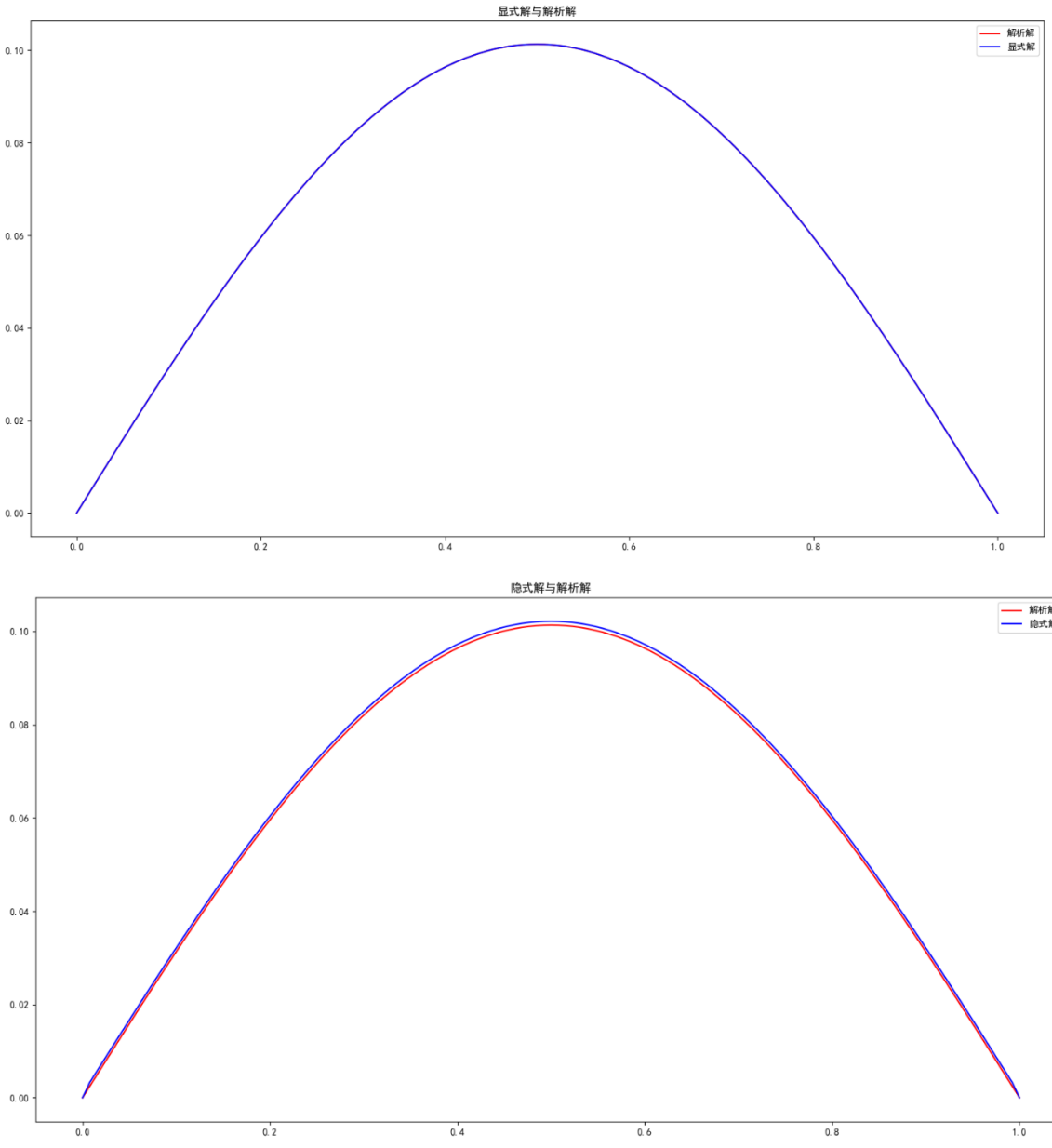
我们将显式方法构建得出可以将下一时刻的状态用这一时刻的状态乘上系数矩阵，也就是 A*Z=X，其中 A 是矩阵，Z 是该时刻状态，X 是下一时刻状态。

可以分析得出，A 是三对角矩阵，根据老师课上给出的例子和 GitHub 上的代码，构建出矩阵 A。为了将所有点都计算出，我们将向量大小设置为 n+1，矩阵大小为(n+1)*(n+1)。根据 GitHub 上的例子找到写矩阵和向量的方法，并在程序中设置。通过查找用户手册，我们获得 HDF5 读写函数：PetscViewerHDF5Open，其中的 FILE_MODE_WRITE 和 FILE_MODE_READ 分别代表对 H5 文件进行写与读。通过 VecView 和 VecLoad 进行向量的输出和载入，通过 PetscObjectSetName 对要输出的对象进行命名，这样可以方便查看数据的结果，也避免读入时发生错误。同时为了进行并行编程，要在更改矩阵和向量后进行 AssemblyBegin 和 AssemblyEnd。要注意 Petsc 中有一部分函数是以成对出现的，比如出现了 Create，后面一定会有 Destroy，出现了 Begin 一定会有 End，有了 Initialize，一定会有 Finalize。为了避免内存泄漏，要将所有的向量 Vec、矩阵 Mat 和输出 PetscView 进行销毁。最后为了防御型编程，要在所有的函数前面添加 ierr，并在此函数结束时进行检查 CHKERRQ(ierr)。

实现重启功能，在运行时输入参数进行“-restart”选择，当参数为 1 或为 TRUE 时，启动重读功能。经比较，重启后的结果与一次性运行的结果相同，证明了重启功能的准确性。

程序准确性分析：

最终将显式方法和隐式方法与解析解进行对比，如下图所示：



上图是用 Python 画出的图形，从上到下分别是：显式解与解析解的对比，隐式解与解析解的对比。

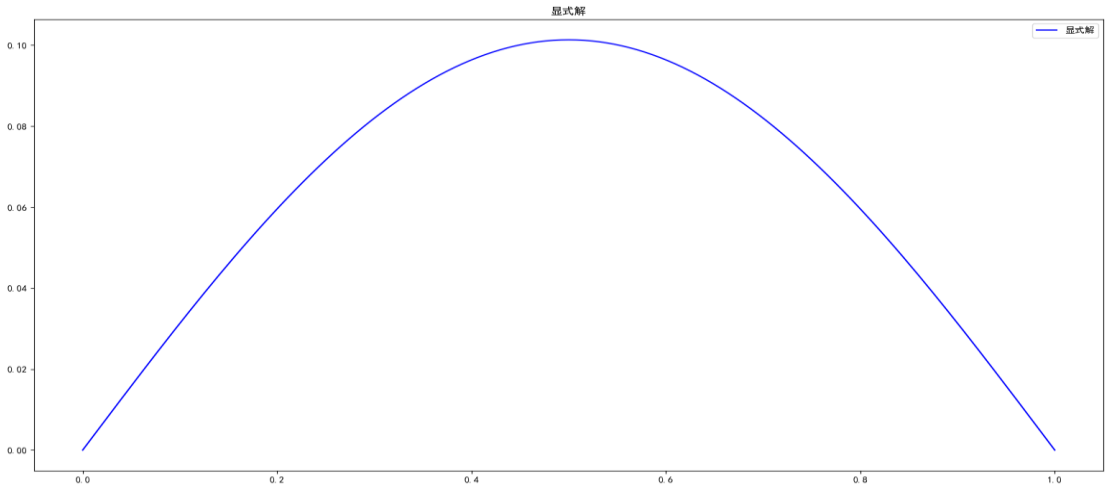
可以看出，显式解的误差要明显小于隐式解，且速度更快，但显式解会受到时间步长和空间步长的影响会发生震荡。

显式瓶颈分析：当 n 较小时，并行速度会因进程间的通信而变慢，当 n 增大时，由于稳定性的限制，时间步长会随之减小，导致程序运算量增大，程序运行时间也会随之增长。并且每十次的写入也会降低程序运行速率，隐式程序也同样。

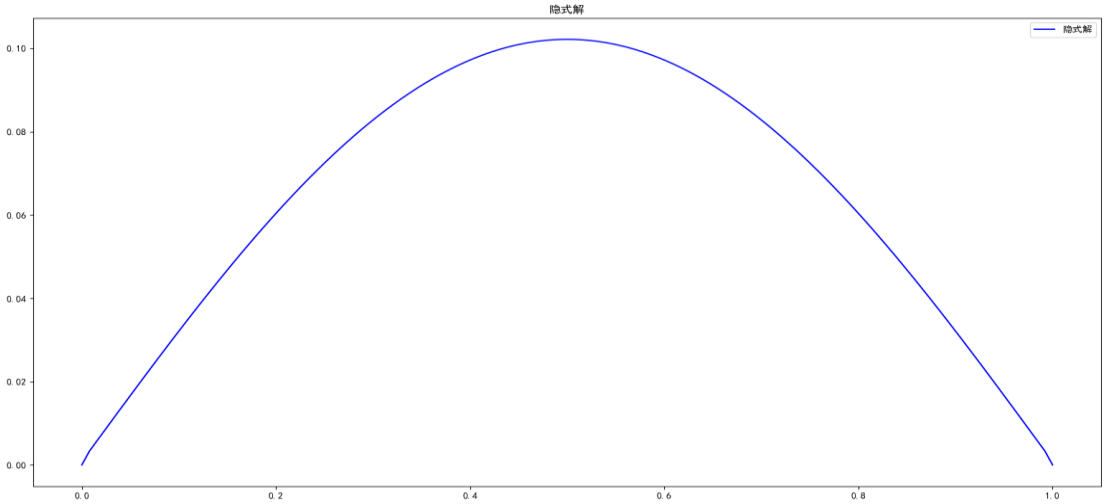
隐式瓶颈分析：当 n 较小时，并行速度会因进程间的通信而变慢，当 n 增大时，由于隐式格式是通过求解矩阵乘法得出，求解的速度也会变慢。

将结果用 Python 程序进行可视化：

显式方法：



隐式方法：



显式方法分析:

由题目我们可以知道:

$$\rho c\frac{\partial u}{\partial t}-\kappa\frac{\partial u}{\partial t}=f, f=sin(l\pi x)$$

从上面的公式中，我们可以利用差分法推出:

$$\rho c\frac{u_m^{n+1}-u_m^n}{\Delta t}-\kappa\frac{u_{m-1}^n-2u_m^n+u_{m+1}^n}{\Delta x^2}=f$$

则我们可以化简得出:

$$u_m^{n+1}=u_m^n+\frac{\kappa}{\rho c}\Delta t*f+\frac{\kappa}{\rho c}\frac{\Delta t}{\Delta x^2}(u_{m-1}^n-2u_m^n+u_{m+1}^n)$$

设$\frac{\kappa}{\rho c}=\alpha$，$\frac{\Delta t}{\Delta x^2}=\beta$，为了方便计算，我们可以设$\frac{\kappa}{\rho c}=\alpha=1$，则上面的公式可以化为:

$$\begin{aligned}u_m^{n+1}&=u_m^n+f+\beta*(u_{m-1}^n-2u_m^n+u_{m+1}^n)\\&=\Delta t*f+\beta*u_{m-1}^n+(1-2\beta)*u_m^n+\beta*u_{m+1}^n\end{aligned}$$

利用冯诺依曼稳定性分析:

$$\delta u_m^{n+1}=\beta*\delta u_{m-1}^n+(1-2\beta)*\delta u_m^n+\beta*\delta u_{m+1}^n$$

$$\delta u_m^{n+1}\sim e^{\delta n\Delta t}*e^{i(k*m\Delta x)}$$

$$e^{\delta\Delta t}=\beta*e^{ik\Delta x}+(1-2*\beta)+\beta*e^{-ik\Delta x}=\beta*\left(e^{ik\Delta x}+e^{-ik\Delta x}\right)+(1-2*\beta)$$

$$=1-2\beta+2\beta\cos(k\Delta x)$$

故若要$\left|e^{\delta\Delta t}\right|\le 1$,则

$$|1-2\beta+2\beta\cos(k\Delta x)|\le 1\Rightarrow -1\le 1-4\beta\le 1\Rightarrow 0\le\beta\le\frac{1}{2}$$

我们以 dx=0.01 为固定条件，更改 dt 的大小，以 0-2s 为时间长度，来观察当结果的变化:

0.	0.	0.	0.
0.00318284	0.00318284	0.00318284	inf.
0.00636253	0.00636253	0.00636253	-inf.
0.00953595	0.00953595	0.00953595	inf.
0.0127	0.0127	0.0127	-inf.
0.0158514	0.0158514	0.0158514	inf.
0.0189873	0.0189873	0.0189873	-inf.
0.0221044	0.0221044	0.0221044	inf.
0.0251996	0.0251996	0.0251996	-inf.
0.02827	0.02827	0.02827	inf.
0.0313125	0.0313125	0.0313125	-inf.
0.0343242	0.0343242	0.0343242	inf.
0.0373019	0.0373019	0.0373019	-inf.
0.0402428	0.0402428	0.0402428	inf.
0.043144	0.043144	0.043144	-inf.
0.0460026	0.0460026	0.0460026	inf.
0.0488159	0.0488159	0.0488159	-inf.
0.0515809	0.0515809	0.0515809	inf.
0.0542951	0.0542951	0.0542951	-inf.
0.0569556	0.0569556	0.0569556	inf.
0.05956	0.05956	0.05956	-inf.
0.0621056	0.0621056	0.0621056	inf.
0.0645899	0.0645899	0.0645899	-inf.
0.0670104	0.0670104	0.0670104	inf.
0.0693648	0.0693648	0.0693648	-inf.
0.0716508	0.0716508	0.0716508	inf.
0.073866	0.073866	0.073866	-inf.
0.0760084	0.0760084	0.0760084	inf.
0.0780757	0.0780757	0.0780757	-inf.
0.080066	0.080066	0.080066	inf.
0.0819773	0.0819773	0.0819773	-inf.
0.0838077	0.0838077	0.0838077	inf.
0.0855553	0.0855553	0.0855553	-inf.

上图从左到右分别是 dt=0.00001、0.00003、0.00005、0.000051

可以看到当 dt=0.000051 时，$\beta = 0.51 \geq 0.5$ ，此时不满足稳态条件，结果不收敛。

因此我们可以知道最大的时间步长是 0.00005，与理论分析相符。

隐式方法分析

由题目我们可以知道：

$$\rho c \frac{\partial u}{\partial t} - \kappa \frac{\partial u}{\partial t} = f, f = sin(l\pi x)$$

从上面的公式中，我们可以利用差分法推出：

$$\rho c \frac{u_m^{n+1} - u_m^n}{\Delta t} - \kappa \frac{u_{m-1}^{n+1} - 2u_m^{n+1} + u_{m+1}^{n+1}}{\Delta x^2} = f$$

则我们可以化简得出：

$$u_m^{n+1} = u_m^n + \frac{\kappa}{\rho c}\Delta t * f + \frac{\kappa}{\rho c}\frac{\Delta t}{\Delta x^2}(u_{m-1}^{n+1} - 2u_m^{n+1} + u_{m+1}^{n+1})$$

设$\frac{\kappa}{\rho c} = \alpha$，$\frac{\Delta t}{\Delta x^2} = \beta$，为了方便计算，我们可以设$\frac{\kappa}{\rho c} = \alpha = 1$，则上面的公式可以化为：

$$\begin{aligned} u_m^{n+1} &= u_m^n + f + \beta * (u_{m-1}^{n+1} - 2u_m^{n+1} + u_{m+1}^{n+1}) \\ &= \Delta t * f + \beta * u_{m-1}^{n+1} + (1 - 2\beta) * u_m^{n+1} + \beta * u_{m+1}^{n+1} \end{aligned}$$

利用冯诺依曼稳定性分析：

$$-\beta * \delta u_{m-1}^{n+1} + (1 + 2\beta) * \delta u_m^{n+1} - \beta * \delta u_{m+1}^{n+1} = \delta u_m^n$$

$$\delta u_m^{n+1} \sim e^{\delta n \Delta t} * e^{i(k*m\Delta x)}$$

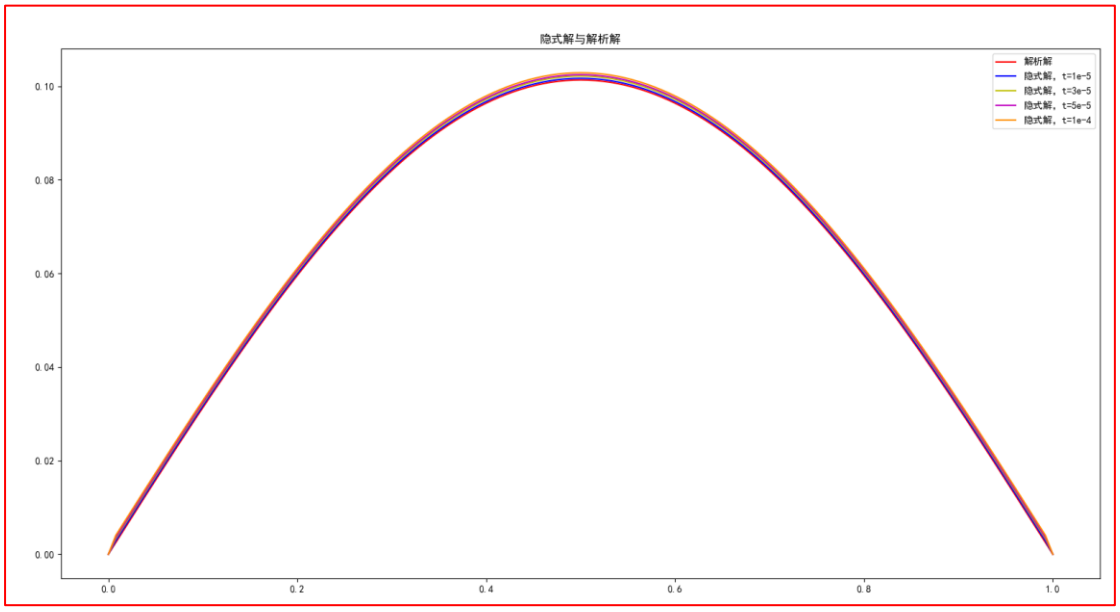
$$-\beta * e^{ik\Delta x} + (1 + 2 * \beta) - \beta * e^{-ik\Delta x} = 1 \Rightarrow \beta * e^{ik\Delta x} - (1 + 2 * \beta) + \beta * e^{-ik\Delta x}$$

$$= e^{\delta \Delta t}(2\beta \cos(k\Delta x) - (1 + 2\beta)) = -1$$

所以$e^{\delta \Delta t} = \left|\frac{-1}{2\beta \cos(k\Delta x) - (1+2\beta)}\right| \leq 1$

所以隐式方法是无条件稳定的。

当时间步长增大时：



当时间步长为 1e-5 时，与解析解误差最大为：0.0003555030568204 72

当时间步长为 3e-5 时，与解析解误差最大为：0.0008248621607031825

当时间步长为 5e-5 时，与解析解误差最大为：0.0011253324038537 35

当时间步长为 1e-4 时，与解析解误差最大为：0.0015497055507966 29

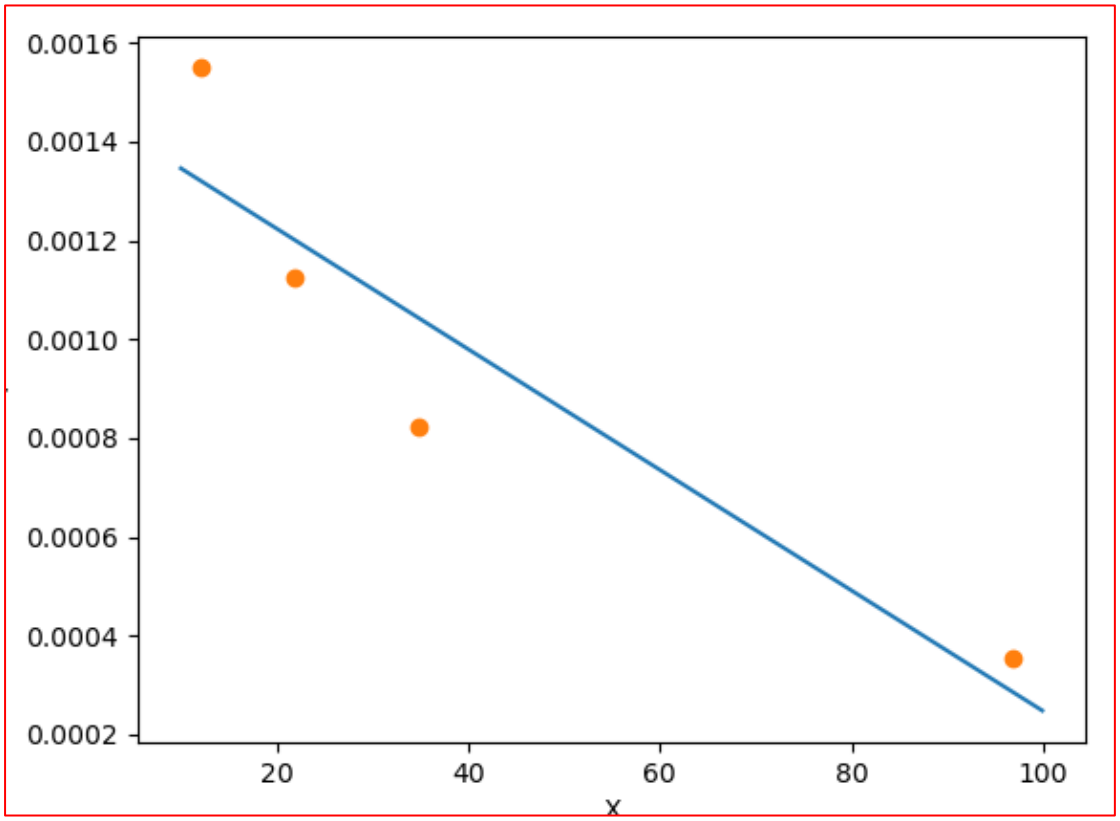
当时间步长为 1e-5 时，运行时间为：96.74s

当时间步长为 3e-5 时，运行时间为：34.78s

当时间步长为 5e-5 时，运行时间为：21.85s

当时间步长为 1e-4 时，运行时间为：12.06s

可以看出随着时间步长的增大，运行时间逐渐缩小，但同时与解析解的误差也在逐渐增加。将运行时间和误差进行拟合后得到如下的图像：



当坐标点在直线下方时，代表真实误差要小于估计误差，根据上图我们发现。只有时间步长为 3e-5 和 5e-5 时真实误差小于估计误差，因此当时间步长为 5e-5 时，速度和误差都是可以接受的。

误差函数拟合

显式误差函数拟合：

当 dx 不变，dt 变化时，我们获得的误差均为：

8.816357662227992e-06

则显式方法 dt 对 e 无影响。

当 dt 不变，dx 变化时，我们获得的误差为：

8.816357662227992e-06, 2.4833571247440123e-06, 1.3718624483904929e-06, 9.413415984482754e-07, 8.163576622199908e-07, 6.99499012199456e-07, 5.712932274470894e-07

空间节点数为：

100, 200, 300, 400, 500, 600, 700

则可以求出 log(e)关于 log(dx)的函数为：log(e)= 1.375log(dx) -2.386

综上所述，$e = C_1(\Delta x)^{1.375}$

隐式误差函数拟合：

当 dx 不变，dt 变化时，我们获得的误差为：

0.000355, 0.000824, 0.001125, 0.001549

时间步长为：

0.00001, 0.00003, 0.00005, 0.0001

则可以求出 log(e)关于 log(dt)的函数为：log(e)=0.648log(dt)-0.186

当 dt 不变，dx 变化时，我们获得的误差为：

4.024826116419411e-05,6.376415385268841e-05,8.90377675831916e-05, 0.00011076415385269378,0.00012812593769832847

空间节点数为：

100, 200, 300, 400, 500

则可以求出 log(e)关于 log(dx)的函数为：log(e)= -0.730log(dx) -5.861

综上所述，$e = C_1(\Delta x)^{-0.730} + C_2(\Delta t)^{0.648}$

代码内存泄露分析

显式格式：

```
==442488== Memcheck, a memory error detector
==442488== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==442488== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==442488== Command:
/share/intel/2018u4/compilers_and_libraries_2018.5.274/linux/mpi/intel64/bin/mpi
run ./explicit.out
==442488==
==442491==
==442491== HEAP SUMMARY:
==442491==     in use at exit: 53,689 bytes in 1,163 blocks
==442491==   total heap usage: 1,712 allocs, 549 frees, 90,399 bytes allocated
==442491==
==442491== LEAK SUMMARY:
==442491==    definitely lost: 0 bytes in 0 blocks
==442491==    indirectly lost: 0 bytes in 0 blocks
==442491==      possibly lost: 0 bytes in 0 blocks
==442491==    still reachable: 53,689 bytes in 1,163 blocks
==442491==         suppressed: 0 bytes in 0 blocks
==442491== Rerun with --leak-check=full to see details of leaked memory
==442491==
==442491== For counts of detected and suppressed errors, rerun with: -v
==442491== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==442507==
==442507== HEAP SUMMARY:
==442507==     in use at exit: 56,009 bytes in 1,251 blocks
==442507==   total heap usage: 3,444 allocs, 2,193 frees, 142,782 bytes
allocated
==442507==
==442507== LEAK SUMMARY:
==442507==    definitely lost: 10 bytes in 1 blocks
==442507==    indirectly lost: 0 bytes in 0 blocks
==442507==      possibly lost: 0 bytes in 0 blocks
==442507==    still reachable: 55,999 bytes in 1,250 blocks
==442507==         suppressed: 0 bytes in 0 blocks
==442507== Rerun with --leak-check=full to see details of leaked memory

==442507==
==442507== For counts of detected and suppressed errors, rerun with: -v
==442507== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==442488==
==442488== HEAP SUMMARY:
==442488==     in use at exit: 59,460 bytes in 1,236 blocks
==442488==   total heap usage: 3,727 allocs, 2,491 frees, 149,042 bytes
allocated
==442488==
==442488== LEAK SUMMARY:
==442488==    definitely lost: 0 bytes in 0 blocks
==442488==    indirectly lost: 0 bytes in 0 blocks
==442488==      possibly lost: 0 bytes in 0 blocks
==442488==    still reachable: 59,460 bytes in 1,236 blocks
==442488==         suppressed: 0 bytes in 0 blocks
==442488== Rerun with --leak-check=full to see details of leaked memory
==442488==
==442488== For counts of detected and suppressed errors, rerun with: -v
==442488== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

经检查，红框内的泄漏是 PETSc 的函数库造成的，程序无泄漏。

隐式格式：

```
==373547== Memcheck, a memory error detector
==373547== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==373547== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==373547== Command:
/share/intel/2018u4/compilers_and_libraries_2018.5.274/linux/mpi/intel64/bin/mpi
run ./implicit.out
==373547==
==373722==
==373722== HEAP SUMMARY:
==373722==     in use at exit: 53,731 bytes in 1,163 blocks
==373722==   total heap usage: 1,712 allocs, 549 frees, 90,455 bytes allocated
==373722==
==373722== LEAK SUMMARY:
==373722==    definitely lost: 0 bytes in 0 blocks
==373722==    indirectly lost: 0 bytes in 0 blocks
==373722==      possibly lost: 0 bytes in 0 blocks
==373722==    still reachable: 53,731 bytes in 1,163 blocks
==373722==         suppressed: 0 bytes in 0 blocks
==373722== Rerun with --leak-check=full to see details of leaked memory
==373722==
==373722== For counts of detected and suppressed errors, rerun with: -v
==373722== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==375051==
==375051== HEAP SUMMARY:
==375051==     in use at exit: 56,051 bytes in 1,251 blocks
==375051==   total heap usage: 3,444 allocs, 2,193 frees, 142,852 bytes
allocated
==375051==
==375051== LEAK SUMMARY:
==375051==    definitely lost: 10 bytes in 1 blocks
==375051==    indirectly lost: 0 bytes in 0 blocks
==375051==      possibly lost: 0 bytes in 0 blocks
==375051==    still reachable: 56,041 bytes in 1,250 blocks
==375051==         suppressed: 0 bytes in 0 blocks
==375051== Rerun with --leak-check=full to see details of leaked memory
```

```
==375051==
==375051== For counts of detected and suppressed errors, rerun with: -v
==375051== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==373547==
==373547== HEAP SUMMARY:
==373547==     in use at exit: 59,502 bytes in 1,236 blocks
==373547==   total heap usage: 3,727 allocs, 2,491 frees, 149,112 bytes
allocated
==373547==
==373547== LEAK SUMMARY:
==373547==    definitely lost: 0 bytes in 0 blocks
==373547==    indirectly lost: 0 bytes in 0 blocks
==373547==      possibly lost: 0 bytes in 0 blocks
==373547==    still reachable: 59,502 bytes in 1,236 blocks
==373547==         suppressed: 0 bytes in 0 blocks
==373547== Rerun with --leak-check=full to see details of leaked memory
==373547==
==373547== For counts of detected and suppressed errors, rerun with: -v
==373547== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

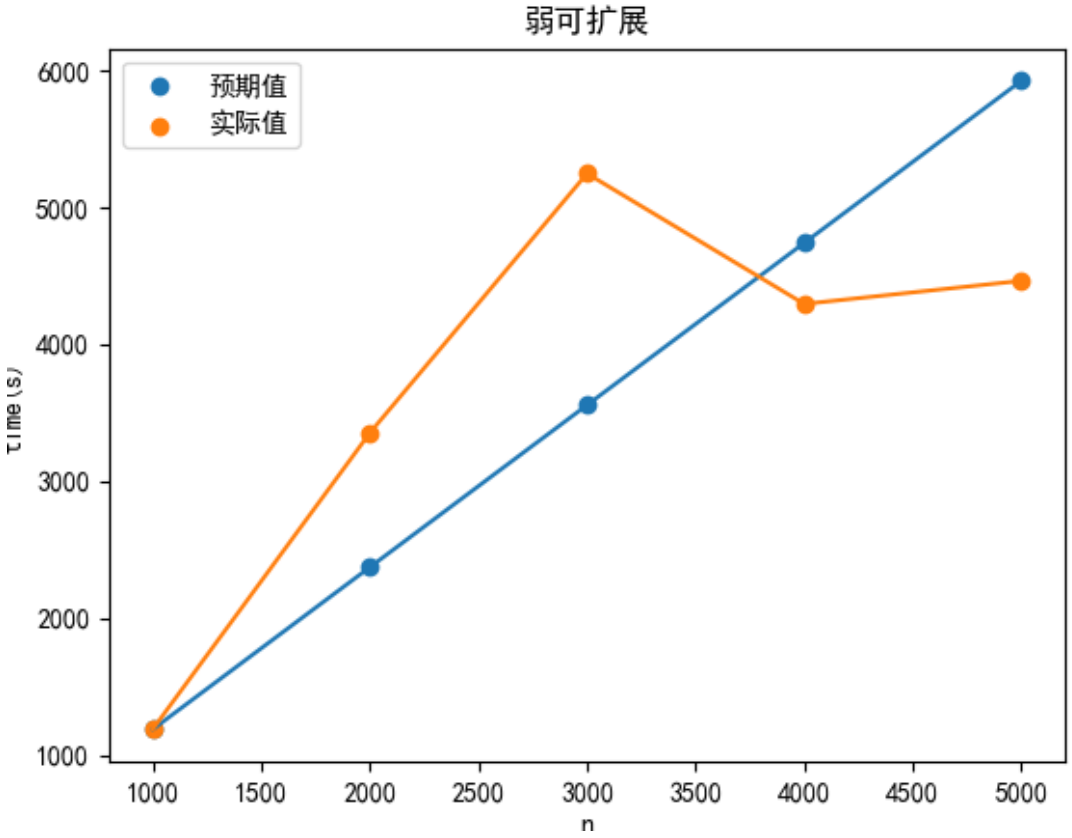
经检查，红框内的泄漏是 PETSc 的函数库造成的，程序无泄漏。

综上所述，显式和隐式程序均无内存泄漏。

并行：

1）显式格式

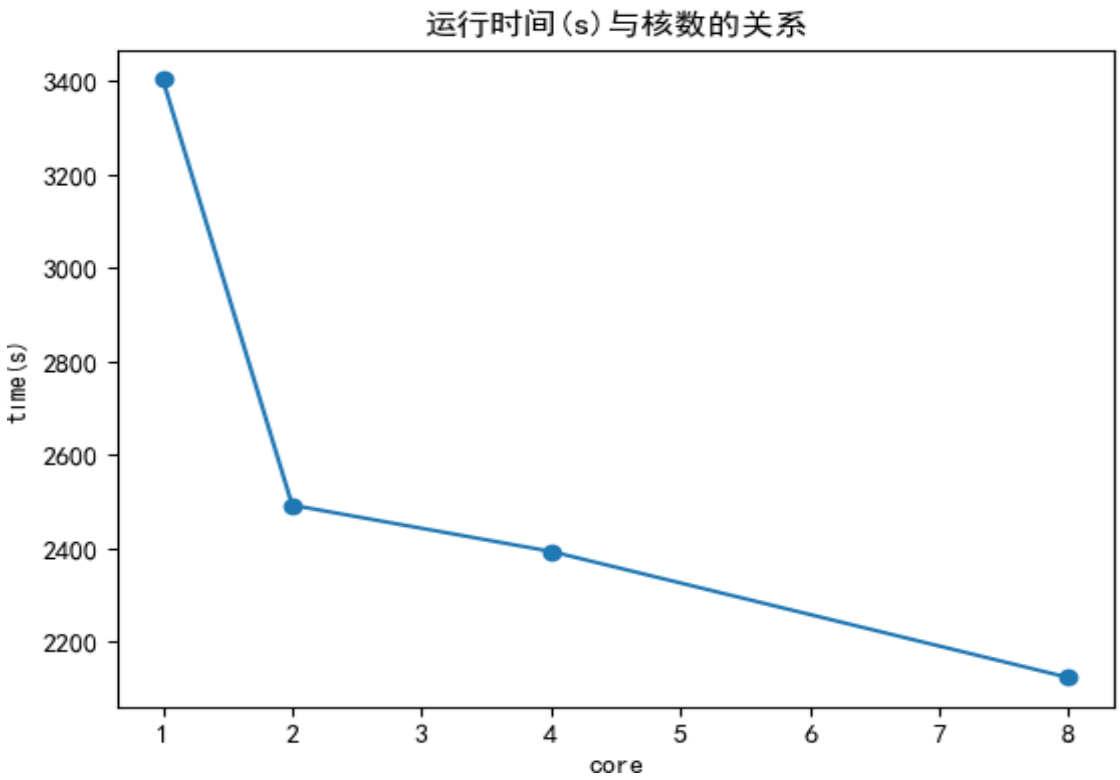
当每个核运行的规模一样时，测试程序的弱可扩展性，则显式格式的弱可扩展性如下



由上图可以看出，当 dt=1e-8 时，当 n<4000 时，程序可扩展性并不好，但当 n>=4000 时，程序并行性较好，弱可扩展性良好。

当每个程序运行的规模一样时，使用不同数量的核，测试程序的强可扩展性，则隐式格式的强可扩展性如下：

则当 n=5000,dt=2e-8 核数从 1 到 8 时，显式格式的强可扩展性如下：

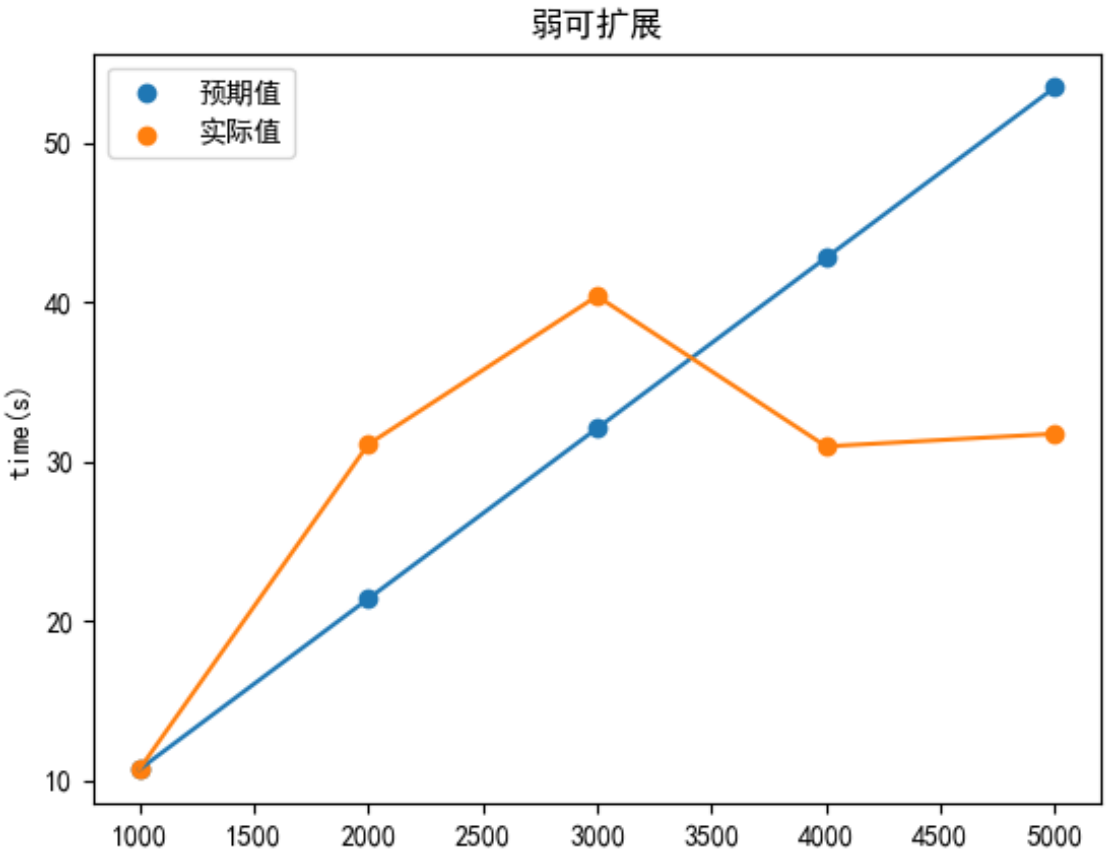


由上图可以看出，当 dt=2e-8，n=5000 时，程序强可扩展性良好

2）隐式格式：

当每个核运行的规模一样时，测试程序的弱可扩展性，则隐式格式的弱可扩展性如下：

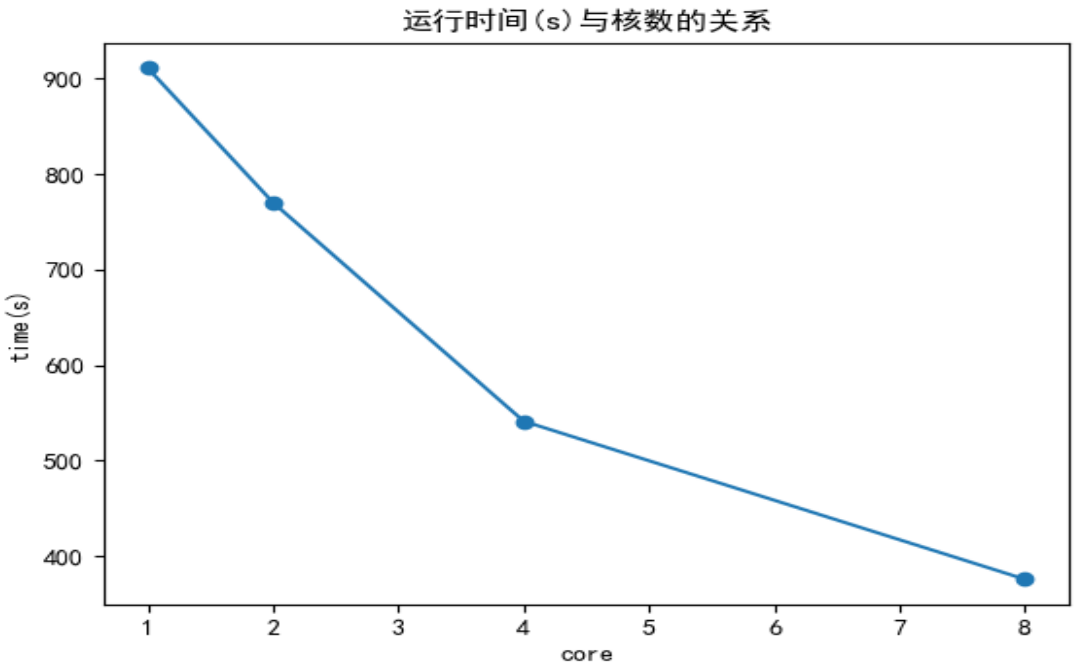
则当 dt=1e-5 核数从 1 到 5，n 从 1000 到 5000 时，隐式格式的强可扩展性如下：



由上图可以看出，当 dt=1e-5 时，当 n<4000 时，程序可扩展性并不好，但当 n>=4000 时，程序并行性较好，弱可扩展性良好。

当每个程序运行的规模一样时，使用不同数量的核，测试程序的强可扩展性，则隐式格式的强可扩展性如下：

则当 n=10000,dt=1e-5 核数从 1 到 8 时，隐式格式的强可扩展性如下：



由上图可以看出，当 dt=1e-5 时，当 n=5000 时，程序强可扩展性良好。

线性求解器性能比较：

选择不同的 PC 参数为程序参数，运行程序，比较性能，结果如下：

	Max	Max/Min	Avg	Total
Time (sec):	4.314e+02	1.000	4.314e+02	
Objects:	4.000e+01	1.000	4.000e+01	
Flop:	1.460e+12	1.000	1.460e+12	1.460e+12
Flop/sec:	3.384e+09	1.000	3.384e+09	3.384e+09
MPI Messages:	0.000e+00	0.000	0.000e+00	0.000e+00
MPI Message Lengths:	0.000e+00	0.000	0.000e+00	0.000e+00
MPI Reductions:	0.000e+00	0.000		

	Max	Max/Min	Avg	Total
Time (sec):	9.857e+01	1.000	9.857e+01	
Objects:	4.300e+01	1.000	4.300e+01	
Flop:	6.203e+10	1.000	6.203e+10	6.203e+10
Flop/sec:	6.293e+08	1.000	6.293e+08	6.293e+08
MPI Messages:	0.000e+00	0.000	0.000e+00	0.000e+00
MPI Message Lengths:	0.000e+00	0.000	0.000e+00	0.000e+00
MPI Reductions:	0.000e+00	0.000		

	Max	Max/Min	Avg	Total
Time (sec):	6.146e+01	1.000	6.146e+01	
Objects:	2.400e+01	1.000	2.400e+01	
Flop:	7.774e+10	1.000	7.774e+10	7.774e+10
Flop/sec:	1.265e+09	1.000	1.265e+09	1.265e+09
MPI Messages:	0.000e+00	0.000	0.000e+00	0.000e+00
MPI Message Lengths:	0.000e+00	0.000	0.000e+00	0.000e+00
MPI Reductions:	0.000e+00	0.000		

从上到下，分别是 Jacobi, Additive Schwarz, LU 三种 PC 参数，可以看出，程序运行速度逐渐加快。