第五次作业

任俊屹，PB16070892，github

1 用 interface 定义出一个新的虚拟函数 Area，当调用 area 只输入一个浮点数时，把他当成是圆的半径值，计算并返回圆的面积。当输入两个浮点数时，把他们当成是矩形的两个边长，并返回矩形的面积。

直接计算。测试程序输出 area(5.) 和 area(3., 4.) 的值。代码如下：
area.f90

```
module area_module
    implicit none
    interface area
    module procedure area_circle, area_rectangle
    end interface

    contains
        function area_circle(r)
        !Calculate area of a circle, if one arg recieved.
            Real :: area_circle, r
            area_circle = 3.14159265358979 * r**2
        end function area_circle

        function area_rectangle(a, b)
        !Calculate area of a rectangle, if two args recieved.
            Real :: area_rectangle, a, b
            area_rectangle = a * b
        end function area_rectangle
end module area_module
```

测试代码如下：
test.f90

```
program main
    !Just a test program, print results calling functing "area".
    use area_module
    write(*, *) "Value of 'area(5.)': ", area(5.)
    write(*, *) "Value of 'area(3., 4.)': ", area(3., 4.)
end program
```

无输入文件，输出文件为output.txt

2 用 Lax-Wendroff 格式编写程序求解 Burgers 方程，并分析 CFL（柯朗数）对计算的影响。

对于非线性方程

$$\frac{\partial u(x,t)}{\partial t}+\frac{\partial f(u(x,t))}{\partial x}=0$$

Lax-Wendroff 格式的表达式为：

$$u_i^{n+1}=u_i^n-\frac{\Delta t}{2\Delta x}[f(u_{i+1}^n)-f(u_{i-1}^n)]+\frac{\Delta t^2}{2\Delta x^2}[A_{i+1/2}(f(u_{i+1}^n)-f(u_i^n))-A_{i-1/2}(f(u_i^n)-f(u_{i-1}^n))]$$

式中，$A_{i\pm1/2}$ 的值为 $\frac{1}{2}(u_i^n+u_{i\pm1}^n$。本例中，

$$f(u)=\frac{1}{2}u^2$$

全局定义代码：

global.f90

```
module Global
    implicit none
    !Scale and output info
    Integer, parameter :: nx = 101, ntend = 4
    !Grid
    Real, parameter :: x_range(2) = (/-2., 14./), &
        dx = (x_range(2) - x_range(1))/(nx-1)
    !CFL
    Real, parameter :: CFL = 0.1
    !Times to output
    Real, parameter :: tend(ntend) = (/1., 2., 3., 4./)

    !Global variables
    Real*4, save :: t, u(nx), x(nx), dt, s=1.
    !$omp threadprivate(t, u, x, dt, s)
end module Global
```

计算代码：

routines.f90

```
module Routines
    !This module defines init operation and step operation.
    use Global
    implicit none
    contains

        subroutine init
            implicit none
```

```
            Integer i
            forall (i = 1:nx)
                x(i) = (i - 1) * dx + x_range(1)
            end forall
            u = 1.5 + s * tanh(x)
            t = 0
        end subroutine init

        subroutine next_LW
            !Lax-Wendroff method.
            !Calling this subroutine will make the module step forward.
            implicit none
            Real, save :: u_t(0:nx+1), a_t(1:nx+1)
            !$omp threadprivate(u_t, a_t)
            u_t(1:nx) = u
            u_t(0) = 1.5 + s * tanh(x_range(1))
            u_t(nx+1) = 1.5 + s * tanh(x_range(2))

            a_t = 1/2. * (u_t(0:nx) + u_t(1:nx+1))

            dt = CFL * dx / maxval(abs(a_t))

            u = u - dt/(4*dx) * (u_t(2:nx+1)**2 - u_t(0:nx-1)**2) + &
                1/4.*(dt/dx)**2 * (a_t(2:nx+1)*(u_t(2:nx+1)**2-u_t(1:nx)**2) - &
                                  a_t(1:nx)*(u_t(1:nx)**2-u_t(0:nx-1)**2))
            t = t + dt
        end subroutine next_LW

end module Routines
```

非并行测试代码：

main.f90

```
program main
    !One thread test, use s = 1 as default
    use Global
    use Routines
    implicit none
    Integer :: i

    open(10, file='LW.dat', form='unformatted', status='replace')

```

```fortran
    call init
    write(10) t, u

    do i = 1, ntend
        do while(t < tend(i))
            call next_LW
        end do
        write(10) t, u
    end do

    close(10)
end program
```

并行测试代码，包括 $s = \pm 1$：

parallel.f90

```fortran
program main
    !Parallel test. Use s = 1 and s = -1.
    !Output to parallel_0.dat and parallel_1.dat
    use Global
    use Routines
    implicit none
    Integer :: i, j
    Character :: filename

    !$omp parallel do private(j, filename)
    do i = 0, 1
        write(filename, '(i1)') i
        select case(i)
            case(0)
                s = 1
                call init
            case(1)
                s = -1
                call init
        end select
        call init
        open(10+i, file='parallel'//filename//'.dat', &
            status='replace', form='unformatted')
        write(10+i) t, u

        do j = 1, ntend
```

```
            do while(t < tend(j))
                call next_LW
            end do
            write(10+i) t, u
        end do
    end do
    !$omp end parallel do

end program
```

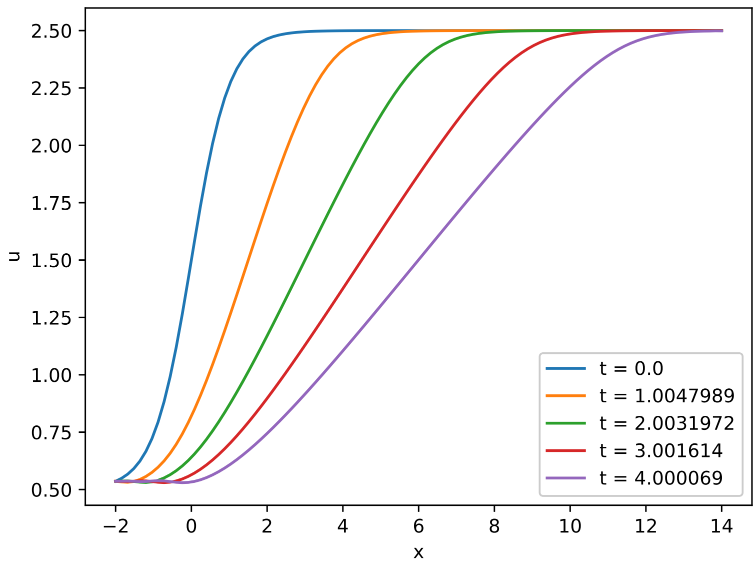
绘图代码：

draw.py

```
import numpy as np
from scipy.io import FortranFile
import matplotlib.pyplot as plt
import sys

f = FortranFile(sys.argv[1], 'r')
x = np.linspace(-2, 14, 101)

lines = []
labels = []

while True:
    try:
        data = f.read_reals(dtype=np.float32)
    except (TypeError):
        break
    labels.append('t = ' + str(data[0]))
    l, = plt.plot(x, data[1:])
    lines.append(l)

plt.xlabel('x')
plt.ylabel('u')
plt.legend(handles=lines, labels=labels)
plt.savefig(sys.argv[1][:-4]+'.eps', format='eps')
```

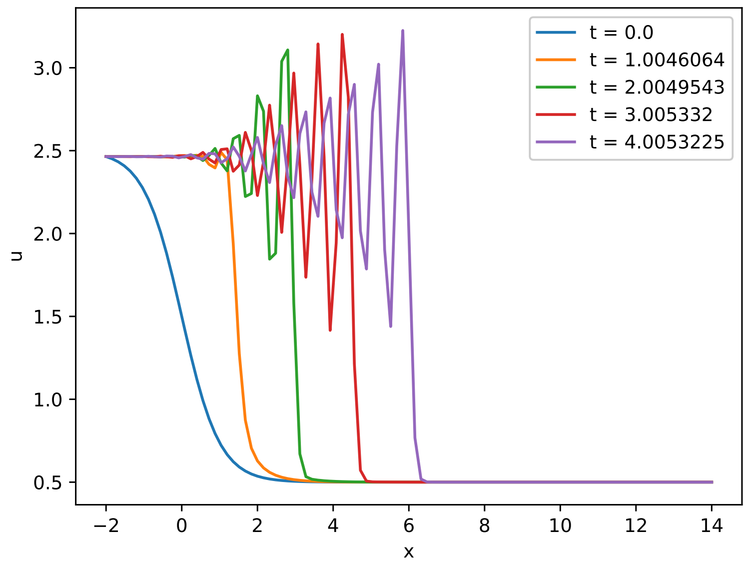
其中，并行测试部分的第一个输出与非并行部分相同，以下插图不再重复。$s = 1$：

parallel0.eps



$s = -1$：

parallel1.eps



非并行程序的输出文件为LW.dat,相应图像为LW.eps。并行程序的输出文件为parallel0.dat和parallel1.dat。

3 用蛙跳格式编写程序求解扩散方程，并分析 CFL（柯朗数）对计算的影响。

套入格式直接计算即可。由于该隐式格式简单，可以直接转换为显示格式计算，即：

$$u_i^{n+1} = (u_i^{n-1} + \frac{2b\Delta t}{\Delta x^2}(u_{i+1}^n - u_i^{n-1} + u_{i-1}^n))/(1 + \frac{2b\Delta t}{\Delta x^2})$$

全局定义代码：

global.f90

```
module Global
    implicit none
    Integer, parameter :: nx = 101, ntend = 4
    Real, parameter :: x_range(2) = (/-2, 2/), tend(ntend) = (/1., 2., 3., 4./)
    Real, parameter :: b = 0.01, dx = (x_range(2) - x_range(1))/(nx-1), CFL=0.2
    !u_last storing u's value in last step for calculating u_next
    Real, save :: x(nx), t, dt, u(nx), u_last(nx), u_next(nx)
    !$omp threadprivate(x, t, dt, u, u_last, u_next)
end module Global
```

计算代码：

routines.f90

```
module Routines
    use Global
    implicit none
    contains

        !Initialize everything to default value.
        subroutine init
            implicit none
            Integer :: i
            forall (i = 1:nx)
                x(i) = (i-1) * dx + x_range(1)
            end forall
            u = exp(-(x/0.1)**2)
            u_last = u
            u_next = u
            t = 0
        end subroutine init

        !Use when need different u.
        subroutine init_without_u
            implicit none
            Integer :: i
            forall (i = 1:nx)
                x(i) = (i-1) * dx + x_range(1)
            end forall
            t = 0
        end subroutine init_without_u

        !Let u_last and u_next equals to u. Call when u is envalued.
```

```
        subroutine init_u
            implicit none
            u_last = u
            u_next = u
        end subroutine init_u

        subroutine next
            implicit none
            Real, save :: u_t(0:nx+1), alpha
            !$omp threadprivate(u_t, alpha)

            dt = dx*CFL/maxval(abs(u))
            u_t(1:nx) = u
            u_t(0) = 0
            u_t(nx+1) = 0

            alpha = 2*b*dt/dx**2

            u_next = (u + alpha*(u_t(2:nx+1) - u_last + u_t(0:nx-1)))/(1+alpha)
            u_last = u
            u = u_next
            t = t + dt
        end subroutine next
end module Routines
```

非并行测试代码：

main.f90

```
program main
    !One thread case.
    use Global
    use Routines
    implicit none
    Integer i

    open(10, file='data.dat', form='unformatted', status='replace')

    call init

    write(10) t, u

    do i = 1, ntend
```

```
        do while(t < tend(i))
            call next
        end do
        write(10) t, u
    end do

    close(10)
end program
```

并行测试代码，包括 $s = \pm 1$：

parallel.f90

```
program main
    !Two threads case.
    !Data will be saved to parallel_0.dat and parallel_1.dat
    use Global
    use Routines
    implicit none
    Integer :: i, j
    Real, parameter :: pi=3.14159265358979
    Character :: filename

    !$omp parallel do private(j)
    do i = 0, 1
        call init_without_u
        select case(i)
            case(0)
                u = exp(-(x/0.1)**2)
            case(1)
                u = exp(-((x-1)/0.1)**2) + exp(-((x+1)/0.1)**2)
        end select
        call init_u
        write(filename, '(i1)') i
        open(10+i, file='parallel'//filename//'.dat', &
            form='unformatted', status='replace')
        write(10+i) t, u

        do j = 1, ntend
            do while(t < tend(j))
                call next
            end do
            write(10+i) t, u
```

```
        end do

    end do
    !$omp end parallel do
end program
```

绘图代码：

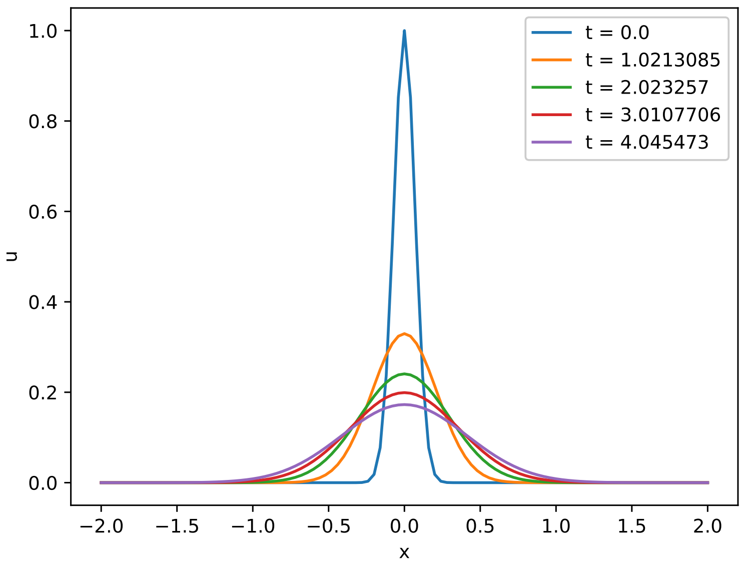
draw.py

```
import numpy as np
from scipy.io import FortranFile
import matplotlib.pyplot as plt
import sys

f = FortranFile(sys.argv[1], 'r')
x = np.linspace(-2, 2, 101)

lines = []
labels = []

while True:
    try:
        data = f.read_reals(dtype=np.float32)
    except (TypeError):
        break
    labels.append('t = ' + str(data[0]))
    l, = plt.plot(x, data[1:])
    lines.append(l)

plt.xlabel('x')
plt.ylabel('u')
plt.legend(handles=lines, labels=labels)
plt.savefig(sys.argv[1][:-4]+'.eps', format='eps')
```

其中，并行测试部分的第一个输出与非并行部分相同，以下插图不再重复。

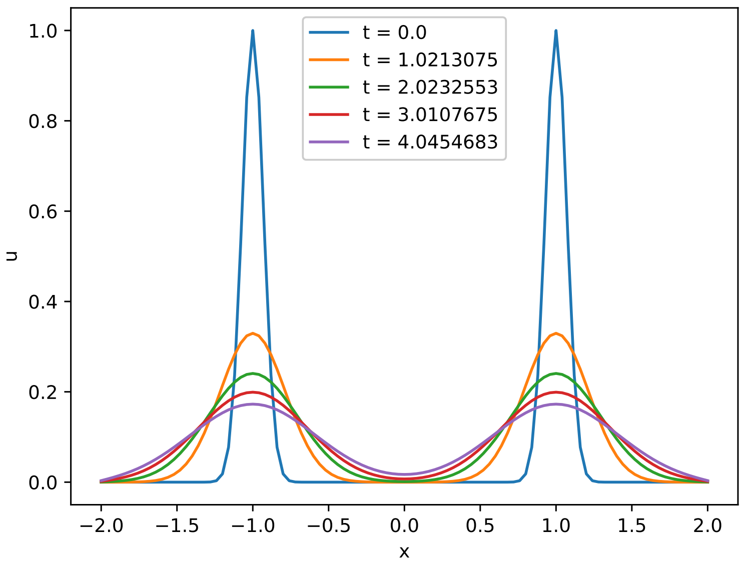
单峰：

parallel0.eps



双峰：

parallel1.eps



非并行程序的输出文件为data.dat，相应图像为data.eps。并行程序的输出文件为parallel0.dat和parallel1.dat。