第六次作业

任俊屹，PB16070892，github

1 给出函数表

x	1.05	1.10	1.15	1.20
f(x)	2.12	2.20	2.17	2.32

构造拉格朗日插值函数，计算 f(1.075) 和 f(1.175) 的近似值。

直接套公式计算即可。其中乘法用 product 函数实现。已部分并行化。代码如下：
1.f90

```
function lagrange(x)
    !works only on the given question.
    implicit none
    Real :: lagrange, x, ret
    Integer :: length, i
    Real, parameter :: xs(*) = (/1.05, 1.1, 1.15, 1.2/)
    Real, parameter :: ys(*) = (/2.12, 2.2, 2.17, 2.32/)

    ret = 0.
    length = size(xs)
    do i = 1, length
        !$omp workshare
        ret = ret + &
                    ys(i)*product(x-xs(1:i-1))*product(x-xs(i+1:length))/ &
                    product(xs(i)-xs(1:i-1))/product(xs(i)-xs(i+1:length))
        !$omp end workshare
    end do
    lagrange = ret
end function lagrange

program main
    implicit none
    Real :: lagrange
    write(*, *) 'f(1.075) = ', lagrange(1.075)
    write(*, *) 'f(1.175) = ', lagrange(1.175)
end program
```

输出文件为output.txt。

2 IDL 编写拉格朗日插值，自己写/调包各一个，并比较。

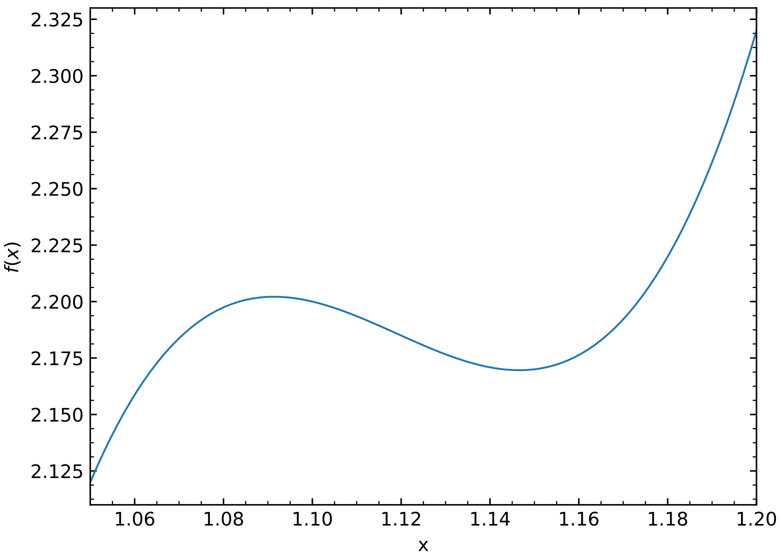
买不起 IDL，支持正版。用 python 完成。
同上，直接套公式。为复用性，定义为类。代码如下：
lagrange.py

```python
#!/usr/bin/python3
import numpy as np
import matplotlib.pyplot as plt

class Lagrange:
    '''
    This class works as a function once initialed.
    '''

    def __init__(self, xs, ys):
        '''
        __init__(self, xs, ys)

        args:
            xs, ys: points for interpolate.
        '''
        self.xs = xs.reshape(xs.size, 1)
        self.ys = ys.reshape(ys.size, 1)

    def __call__(self, x):
        '''
        __call__(self, x)

        args:
            x: a number or a ndarray.

        returns:
            pridiction of x, using lagrange interpolate.
        '''
        ret = 0
        if isinstance(x, np.ndarray):
            x = x.reshape(1, x.size)
        else:
            x = np.array(x).reshape(1, 1)
```

```python
        for i in range(len(xs)):
            ret += self.ys[i][0]*np.product(x-self.xs[:i], axis=0)* \
                      np.product(x-self.xs[i+1:], axis=0)/ \
                      np.product(self.xs[i][0]-self.xs[:i,0])/ \
                      np.product(self.xs[i][0]-self.xs[i+1:,0])
        return ret

if __name__ == '__main__':
    #Solve the question
    import sys
    xs = np.array([1.05, 1.1, 1.15, 1.2])
    ys = np.array([2.12, 2.2, 2.17, 2.32])

    f = Lagrange(xs, ys)
    x = np.linspace(1.05, 1.2, 100)
    y = f(x)
    plt.plot(x, y, lw=1)
    plt.minorticks_on()
    plt.tick_params(which='both',
                    direction='in',
                    top=True,
                    right=True)
    plt.xlim(x.min(), x.max())
    plt.xlabel(r'x')
    plt.ylabel(r'$f(x)$')
    if len(sys.argv) == 1:
        plt.show()
    else:
        plt.savefig(sys.argv[1], format='eps')
```

输出图片：

lagrange.eps



调用 scipy.interpolate.interp1d 完成，代码如下：

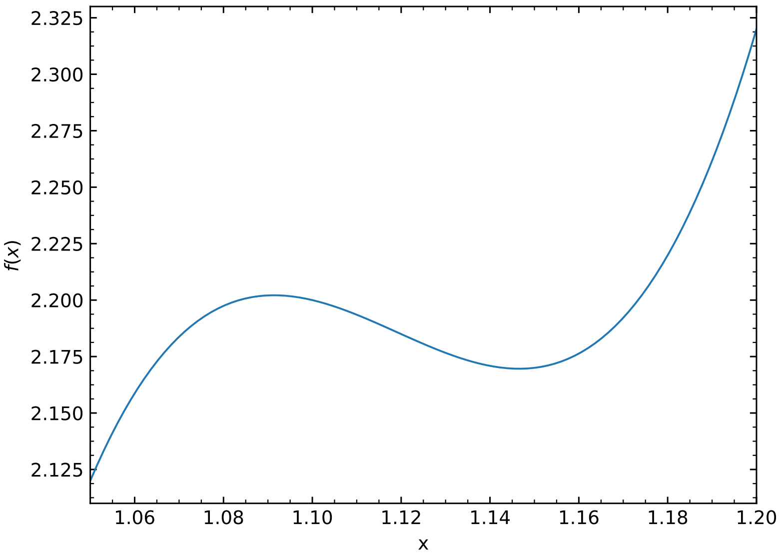
scipy-interpolate.py

```python
import sys
import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

xs = np.array([1.05, 1.1, 1.15, 1.2])
ys = np.array([2.12, 2.2, 2.17, 2.32])

#call function
f = interp1d(xs, ys, kind=xs.size-1)

x = np.linspace(1.05, 1.2, 100)
y = f(x)
plt.plot(x, y, lw=1)
plt.minorticks_on()
plt.tick_params(which='both',
                direction='in',
                top=True,
                right=True)
plt.xlim(x.min(), x.max())
plt.xlabel(r'x')
plt.ylabel(r'$f(x)$')
if len(sys.argv) == 1:
    plt.show()
```

```
else:
    plt.savefig(sys.argv[1], format='eps')
```

输出图片：

scipy-interpolate.eps



结果一样（为什么不呢？）。

3 求高次多项式的全部实根。

考虑用矩阵实现。由于 n 阶方阵有 n 个特征值，可构造方阵，使其特征值为 n 次方程的 n 个根。考虑如下矩阵：

$$M = \begin{pmatrix} 0 & 0 & \dots & -p_0 \\ 1 & 0 & \dots & -p_1 \\ \vdots & \ddots & & \vdots \\ 0 & \dots & 1 & -p_{n-1} \end{pmatrix}$$

式中，p_i 为多项式中 x^i 项系数。容易算出，M 的特征多项式为：

$$p_0 + p_1\lambda + p_2\lambda^2 + \cdots + p_{n-1}\lambda^{n-1} + \lambda^n$$

所以 M 的特征值即为多项式方程的解。

例程中使用 QR 方法求解特征值，算法参考自维基百科。经过足够的迭代后，矩阵 $R\times Q$ 对角线上的元素即为矩阵的实特征值，如果该元素下方即左方元素为零的话。而其所有复特征值将在结果中表现为对角线附近 2×2 的方阵的特征值。参考http://people.inf.ethz.ch/arbenz/ewp/Lnotes/chapter4.pdf

在实验中，存在部分使得 QR 算法不收敛的情况，此时该算法将给出错误的结果。这种情况很少见。算法思路参考自numpy源码。

求解代码如下：

util.f90

```fortran
module util
    implicit none
    contains
        subroutine QR(a, q, r, n)
            !q and r will be QR decomposition of a.
            !a, q, and r should be n*n array

            Real*8 :: a(:, :), q(:, :), r(:, :)
            Integer :: n, i
            Real*8, allocatable :: e(:, :), u(:), v(:, :), h(:, :)
            Real*8 :: alpha

            allocate(e(n, n))
            allocate(u(n))
            allocate(v(n, 1))
            allocate(h(n, n))

            e = 0
            forall (i = 1:n)
                e(i, i) = 1
            end forall
            r(:, :) = a
            q(:, :) = e


            do i = 1, n-1
                u = 0
                v = 0
                h(:, :) = e
                alpha = sqrt(sum(r(i:, i)**2))
                u(i:) = r(i:, i) - e(i:, i)*alpha
                v(i:, 1) = u(i:) / sqrt(sum(u(i:)**2))
                h(i:, i:) = e(i:, i:) - 2*matmul(v(i:, :), transpose(v(i:, :)))
                r(:, :) = matmul(h, r)
                q(:, :) = matmul(q, transpose(h))
            end do
            r(:, :) = matmul(transpose(q), a)

            deallocate(e)
            deallocate(u)
            deallocate(v)
```

```
        deallocate(h)

    end subroutine QR

    subroutine eigen(a, x, n, e, max_step)
        !eigen(a, x, n[, e=1e-5, max_step=2000])
        !x will be a's eigenvalues.
        !a should be shaped n*n, x's size should be n

        Real*8 :: a(:, :), eo
        Real*8 :: x(:)
        Real*8, optional :: e
        Integer, optional :: max_step
        Integer :: n, i, max_stepo, j, num
        Real*8 :: error
        Real*8, allocatable :: q(:, :), r(:, :), a_t(:, :)
        Logical, allocatable :: mask(:, :)
        Logical :: flag

        eo = 1e-5
        max_stepo = 2000
        if (present(e)) eo = e
        if (present(max_step)) max_stepo = max_step

        allocate(q(n, n))
        allocate(r(n, n))
        allocate(a_t(n, n))
        allocate(mask(n, n))

        mask = .false.
        forall (i = 2:n)
            forall (j = 1:i-1)
                mask(i, j) = .true.
            end forall
        end forall

        a_t(:, :) = a

        do i = 1, max_stepo
            error = maxval(abs(a_t), mask=mask)
            if (error < eo) then
```

```fortran
                goto 100
            end if
            call QR(a_t, q, r, n)
            a_t(:, :) = matmul(r, q)
        end do
    100 continue
        num = 0
        do i = 1, n
            flag = .true.
            do j = 1, i-1
                if (a_t(i, j) > eo) flag = .false.
            end do
            do j = i+1, n
                if (a_t(j, i) > eo) flag = .false.
            end do
            if (flag) then
                num = num + 1
                x(num) = a_t(i, i)
            end if
        end do
        n = num

        deallocate(q)
        deallocate(r)
        deallocate(a_t)
        deallocate(mask)

    end subroutine eigen

    subroutine solve(p, x, n)
        !p(1) + p(2) * x + ... + p(n+1) * x**n

        Real*8 :: p(:)
        Real*8 :: x(:)
        Integer :: n, i
        Real*8, allocatable :: m(:, :)

        allocate(m(n, n))

        m = 0
        forall (i = 1:n-1)
```

```
                m(i+1, i) = 1
            end forall
            m(:, n) = -p(:n)/p(n+1)

            call eigen(m, x, n)

            deallocate(m)

        end subroutine solve

end module util
```

测试代码如下：

test.f90

```
program main
    use util
    !There are still bugs in this program.
    !Some times QR-algorithm is not convergent, and this program gives wrong answ
    !i.e. p = (/-1, 1, -1, 1/), x = 1 is a root, but can't be given here.
    Real*8, allocatable :: p(:)
    Real*8, allocatable :: x(:)
    Integer :: n, i

    write(0, *) 'Input order of the polynomial'
    read(*, *) n
    allocate(p(n+1))
    allocate(x(n))

    write(0, *) 'Input coefficients p(i)'
    write(0, *) 'format: p(1) + p(2) * x + ... + p(n+1) * x^n'
    read(*, *) p

    call solve(p, x, n)
    write(*, *) 'real roots are:'
    do i = 1, n
        write(*, *) x(i)
    end do

end program main
```

测试输入文件为input.txt，输出文件为output.txt，输入提示保存为err.txt。

4 用四阶龙格-库塔格式求解初值问题

$$\frac{dy}{dx} = y^2 \cos x$$

$$y(0) = 1$$

$$0 \le x \le 0.8$$

直接套用公式求解即可。代码如下：

runge-kutta.f90

```
module runge_kutta
    implicit none
    contains
        subroutine RK4_step(f, y0, n, x, dx, yout)
            !Run one step of 4-order RK.
            Real :: y0(n), yout(n), x, dx
            Real, allocatable :: k1(:), k2(:), k3(:), k4(:)
            Integer :: n, i, j
            Real, external :: f

            allocate(k1(n))
            allocate(k2(n))
            allocate(k3(n))
            allocate(k4(n))

            k1 = f(x, y0)
            k2 = f(x + dx/2., y0 + k1*dx/2.)
            k3 = f(x + dx/2., y0 + k2*dx/2.)
            k4 = f(x + dx, y0 + dx*k3)

            yout = y0 + dx/6*(k1 + 2*k2 + 2*k3 + k4)

            deallocate(k1)
            deallocate(k2)
            deallocate(k3)
            deallocate(k4)
        end subroutine RK4_step

        subroutine RK4(f, y0, n, x, dx, steps, yout)
            !yout here is array of outputs in each step.
            Real :: y0(n), yout(steps, n), x, dx
            Integer :: n, steps, i
```

```fortran
            Real, external :: f

            call RK4_step(f, y0, n, x, dx, yout(1, :))
            do i = 2, steps
                call RK4_step(f, yout(i-1, :), n, x+(i-1)*dx, dx, yout(i, :))
            end do
        end subroutine RK4
end module runge_kutta
```

测试代码如下：

test.f90

```fortran
function f(t, y)
    !dy/dt = f(t, y)
    Real :: f
    f = y**2 * cos(t)
end function f

program main
    use runge_kutta
    implicit none
    Real :: y0(1) = (/1/), xr(2) = (/0., 0.8/)
    Real, allocatable :: x(:), yout(:, :)
    Integer :: n = 100, i  !steps to run
    Real, external :: f

    allocate(x(n))
    allocate(yout(n, 1))
    forall (i = 1:n)
        x(i) = xr(1) + (xr(2) - xr(1))/(n-1)*i
    end forall

    call RK4(f, y0, 1, xr(1), (xr(2)-xr(1))/(n-1), n, yout)

    open(10, file='res.dat', form='unformatted')
    write(10) x
    write(10) yout
    close(10)
end program
```

输出文件为res.dat。

画图代码如下：

plot.py

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

f = FortranFile('res.dat', 'r')

x = f.read_reals(dtype=np.float32)
y = f.read_reals(dtype=np.float32)

plt.plot(x, y)
#plot more beautiful
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(x.min(), x.max())
plt.minorticks_on()
plt.tick_params(which='both',
                top=True,
                right=True,
                direction='in')
plt.savefig('res.eps', format='eps')
f.close()
```

图像：

res.eps