Imperial College London

September 2019

Developing Fortran using Python and Literate Programming

Paul Bartholomew, Sylvain Laizet

Outline

My programming career

Seems to have gone backwards:

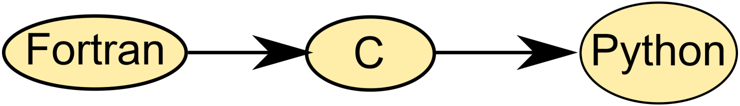


Figure 1: “Progress”

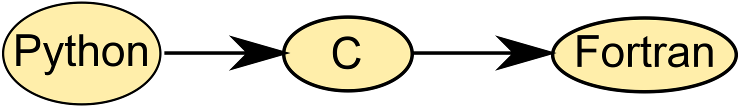


Figure 2: Experience

Literate programming: an old idea

- Introduced by Donald Knuth in 1984

  I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature. Hence, my title: "Literate Programming."

  Instead of imagining that our main task is to instruct a computer what to do, let us concentrate on explaining to human beings what we want a computer to do.

- Doesn't seem to have caught on
- Could be well suited for scientific computing

Exploring literate programming

- Fits very well for producing reports
- Can combine
  - Processing of result
  - Display of result
  - Discussion of results
- `Jupyter` does something like this
- Can we take this idea further?

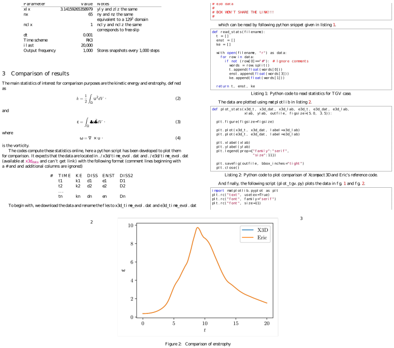


Figure 3: Sections of report on Taylor Green Vortex

The problem

- `Incompact3d` is a CFD code for simulating incompressible turbulent flows
- Want to develop a free-surface solver

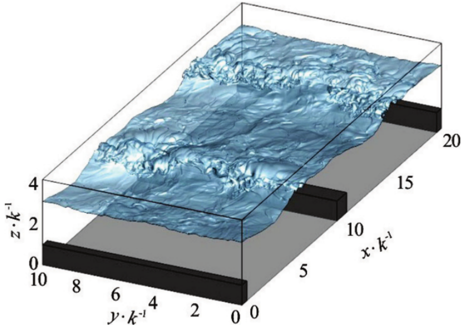


Figure 4: Water surface with submerged obstacles

The problem (cont.)

- The schemes implemented in `Incompact3d` are ill-suited to these problems

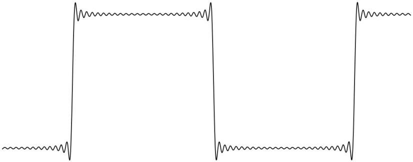
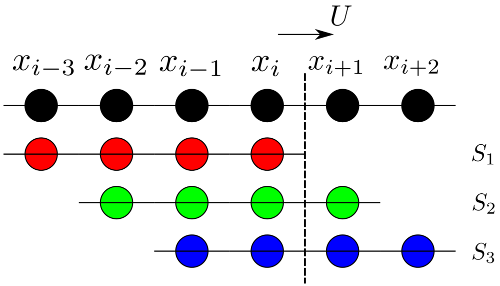


Figure 5: Illustration of Gibbs' phenomenon

- `WENO` schemes provide high-order accuracy without being susceptible to oscillations
  - Could simply code this in Fortran
  - Would like to leave behind something that is understandable
  - Testing code *inside* a complex program is difficult

WENO schemes

- Evaluate several stencils



- Check for "*smoothness*"
- Combine *smooth* stencils to obtain higher order approximation

WENO gradient computation

Weighted combination of stenciles

$$\left.\frac{\partial\phi}{\partial x}\right|_i = \begin{cases} \left.\frac{\partial\phi}{\partial x}\right|_i^- & u > 0 \\ \left.\frac{\partial\phi}{\partial x}\right|_i^+ & u < 0 \end{cases}$$

$$\left.\frac{\partial\phi}{\partial x}\right|_i^{\pm} = \left[\omega_1\left(2q_1^{\pm} + 7q_2^{\pm} + 11q_3^{\pm}\right) + \omega_2\left(-q_2^{\pm} + 5q_3^{\pm} + 2q_4^{\pm}\right) + \omega_3\left(2q_3^{\pm} + 5q_4^{\pm} - q_5^{\pm}\right)\right]/6$$

Listing 1: Evaluation of $\partial\phi/\partial x$ using fifth-order WENO scheme.

```
gradphi(i,j,k)=&
     w1*(2.0*q1-7.0*q2+11.0*q3)&
     +w2*(-q2+5.0*q3+2.0*q4)&
     +w3*(2.0*q3+5.0*q4-q5)
gradphi(i,j,k)=gradphi(i,j,k)/6.0
```

Stencil computation

Stencil definition

$$q_1^{\pm} = \frac{\phi_{i-2} - \phi_{i-3}}{\Delta x},\ q_2^{\pm} = \frac{\phi_{i-1} - \phi_{i-2}}{\Delta x},$$
$$q_3^{\pm} = \frac{\phi_{i} - \phi_{i-1}}{\Delta x},\ q_4^{\pm} = \frac{\phi_{i+1} - \phi_{i}}{\Delta x},$$
$$q_5^{\pm} = \frac{\phi_{i+2} - \phi_{i+1}}{\Delta x},$$

- They are *symmetric* about the gradient evaluation point x_i

Listing 2: Stencil evaluation for fifth-order WENO scheme.

```
q1=dsign*(phi(im2,jm2,km2)-phi(
      im3,jm3,km3))/deltax
q2=dsign*(phi(im1,jm1,km1)-phi(
      im2,jm2,km2))/deltax
q3=dsign*(phi(i,j,k)-phi(im1,jm1
      ,km1))/deltax
q4=dsign*(phi(ip1,jp1,kp1)-phi(i
      ,j,k))/deltax
q5=dsign*(phi(ip2,jp2,kp2)-phi(
      ip1,jp1,kp1))/deltax
```

The stencil weights

The key to `weno`

- Smooth regions have $\approx$ weights $\rightarrow$ high-order
- If stencil k contains discontinuity $\omega_k \rightarrow 0$

Definition

$$\omega_k = \frac{\alpha_k}{\sum_l \alpha_l}$$

Listing 3: Weight calculation for fifth-order `WENO` scheme.

```
w1 = a1 / (a1 + a2 + a3)
w2 = a2 / (a1 + a2 + a3)
w3 = a3 / (a1 + a2 + a3)
```

The weight coefficients

Definition

$$\alpha_k = \frac{C_k}{(IS_k + \varepsilon)^2}$$

Listing 4: Calculating the weighting coefficients

```
<<src:calc-indicators.f90>>
<<src:calc-a1.f90>>
<<src:calc-a2.f90>>
<<src:calc-a3.f90>>
```

Listing 5: Calculating coefficient α_1

```
a1=1.0/(e+is1)**2/10.0
```

Listing 6: Calculating the smoothness indicators

```
<<src:calc-is1.f90>>
<<src:calc-is2.f90>>
<<src:calc-is3.f90>>
```

Listing 7: Calculating IS_1

```
is1=(13.0/12.0) &
     *(phi(im2,jm2,km2)-2.0*phi(im1,jm1,
          km1)+phi(i,j,k))**2 &
     +(phi(im2,jm2,km2)-4.0*phi(im1,jm1,
          km1)+three*phi(i,j,k))**2 &
     /4.0
```

The weno module

Listing 8: The weno module.

```
module weno
  implicit none
  private
  public :: weno5
contains
  <<src:weno5.f90>>
endmodule weno
```

Listing 9: Calculate $\partial\phi/\partial x$ using weno

```
<<src:calcq.f90>>
<<src:calc-weight-coeffs.f90>>
<<src:calcweights.f90>>
<<src:calcgrad.f90>>
```

Listing 10: WENO subroutine definition.

```
subroutine weno5(gradphi, phi, advvel, &
     axis, bc0, bcn, &
     isize, jsize, ksize, &
     dx, dy, dz)
  implicit none
  <<src:weno5-declarations.f90>>
  <<src:weno5-setup.f90>>
  do k = kstart, kend
     do j = jstart, jend
        !! Note, if axis==2 and y is
           stretched, need to set
           deltax here
        do i = istart, iend
           <<src:sign.f90>>
           <<src:wenograd.f90>>
        enddo
        <<src:bcx.f90>>
     enddo
     <<src:bcy.f90>>
  enddo
  <<src:bcz.f90>>
endsubroutine weno5
```

Imperial College London

Testing

Approaches to testing

- Add module directly to `Xcompact3d`
- Test module independently before adding to `Xcompact3d`

Using `f2py`

Can easily build `weno.f90` as a standalone module and call from Python to test

- Easy to setup test cases
- Rapid feedback

Testing on a smooth function

Consider

$$f(x) = \sin(x)$$
$$\Rightarrow f'(x) = \cos(x)$$

Listing 11: Testing the x-derivative

```
for i in range(N):
  for j in range(1):
    for k in range(1):
      u[i][j][k] = 1.0
      phi[i][j][k] = f[i]
      gradphi[i][j][k] = 0.0
weno5(gradphi, phi, u, 1, 2, 2, dx, dx,
     dx)
plt.plot(x, gradphi[:,0,0], marker="o")
plt.plot(x, fp)
plt.title("Test x-derivative (smooth)")
plt.savefig("weno-smoothx.eps",
     bbox_inches="tight")
plt.close()
```

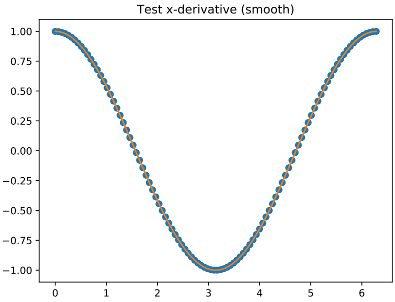


Figure 6: Comparison of numerical and analytical derivative of $f(x) = \sin(x)$

Application to a pure advection equation

Motivating implementation

$$\frac{\partial \phi}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla} \phi = 0$$

- In periodic domain, ϕ simply moves with velocity u
- Simple to implement in Python using `weno5` + `scipy`'s ode solvers
- Domain $x \in [-1, 1]$ discretised with 200 points

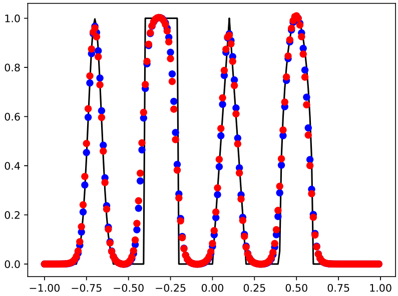


Figure 7: Comparison of analytical solution and numerical solutions at $t = 8, 10$

Conclusion

- A `weno` scheme was implemented and has been incorporated into `Xcompact3d`
+ Using literate programming we can write programs in a way that makes sense to us
+ Explanatory document automatically generated
– Tooling isn't as strong as traditional tooling

- Using `f2py` simplifies testing
+ Quicker feedback on tests
+ Can explore results using `Python`
+ Using literate programming can embed testing + results into same source document

Imperial College London

Code availability

- This talk is available on github at[1]
  - It is *runnable*
  - "Compiling" the talk's source with `emacs` produces this pdf + `weno.f90` + `Python` testing code
- Xcompact3d is also available on github at[2]
  - Current release preview is on the `release` branch

Acknowledgement

Work undertaken as part of an `eCSE` project funded by `EPCC`

archer

[1] https://github.com/pbartholomew08/presentations

[2] https://github.com/xcompact3d/Incompact3d