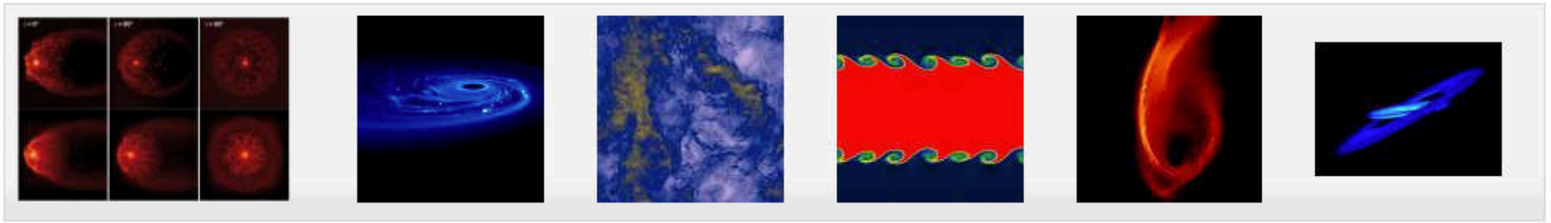
SIMULATING THE UNIVERSE

MY WISH LIST AFTER 25 YEARS OF FORTRAN DEVELOPMENT



DANIEL J. PRICE

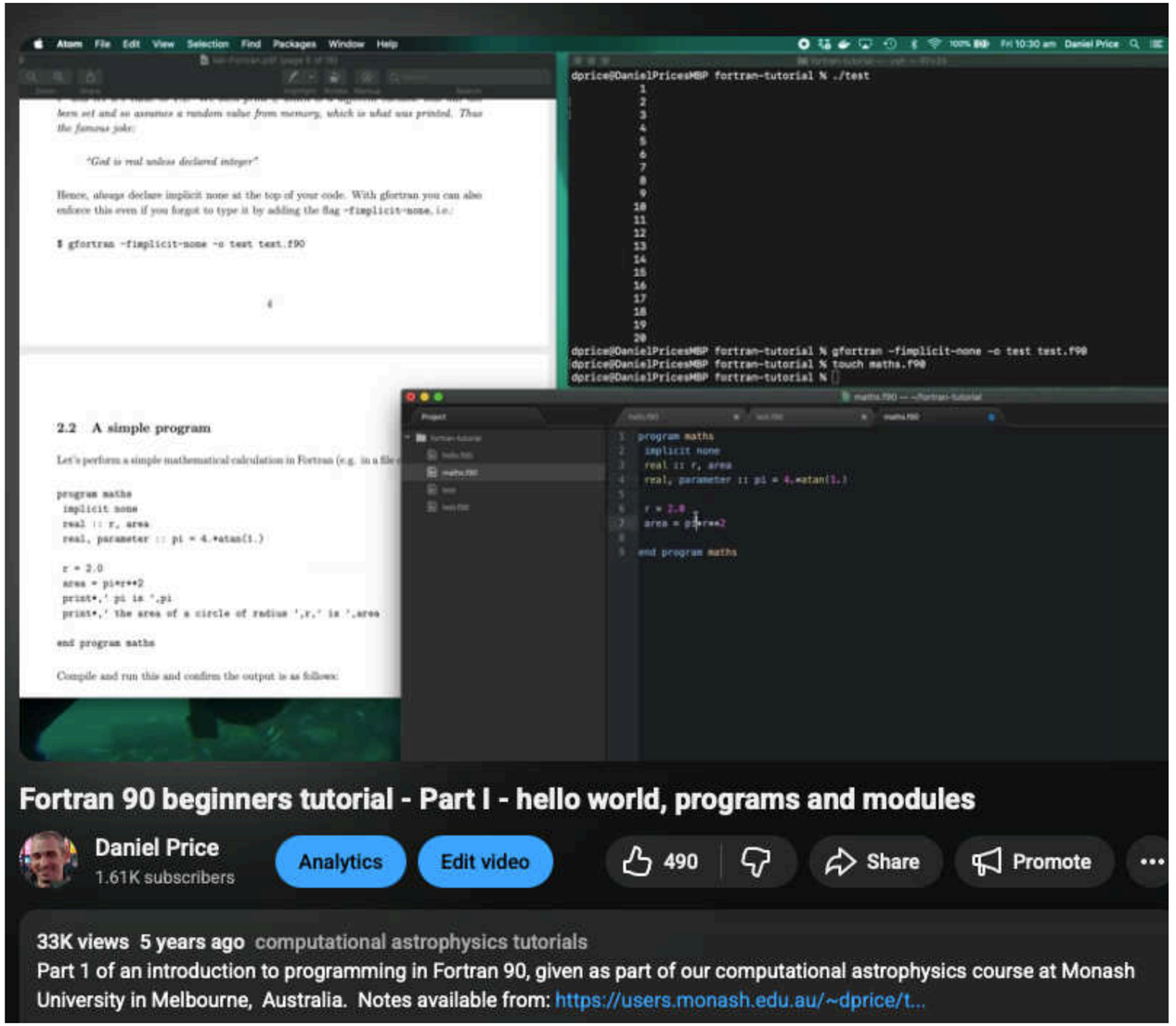
MONASH UNIVERSITY, MELBOURNE, AUSTRALIA

Australian Government
Australian Research Council

MONASH University

UGA Université Grenoble Alpes

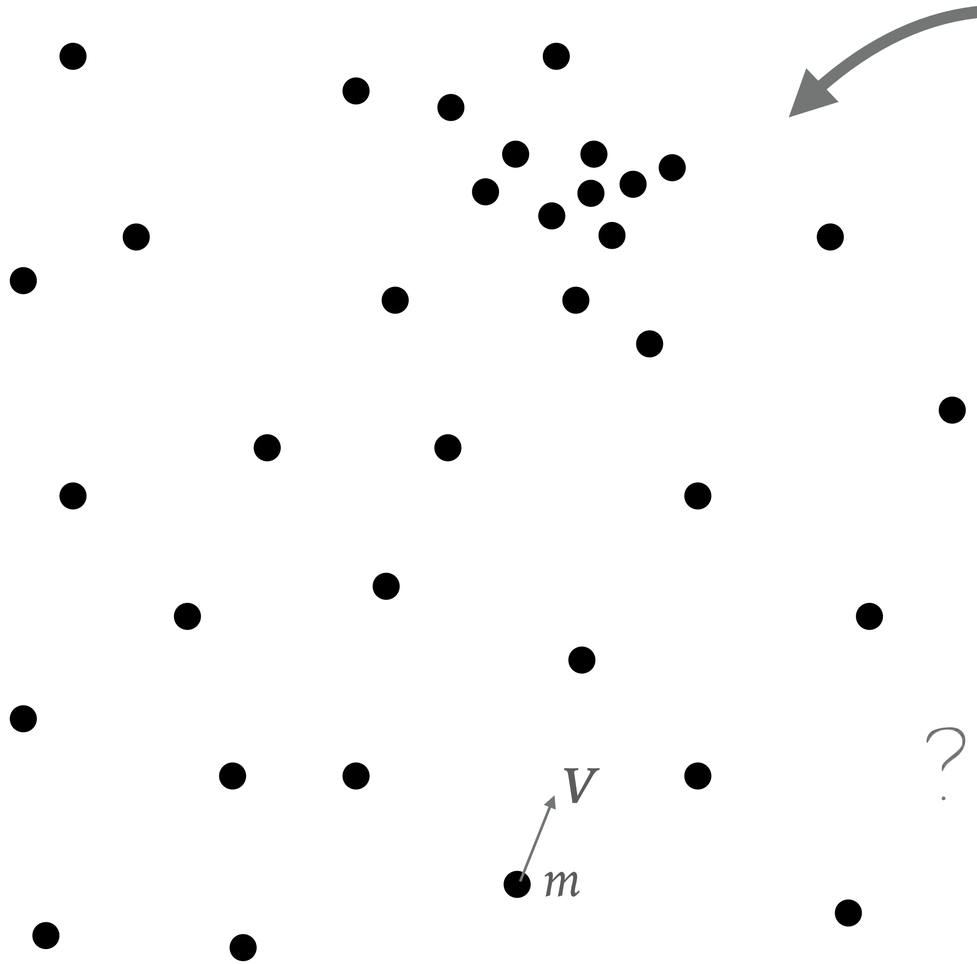
QUICK PLUG: ONLINE FORTRAN TUTORIALS



youtube.com/@danielpriceastro

https://youtu.be/d_ZNWPNzspg

HOW TO SIMULATE THE UNIVERSE ?

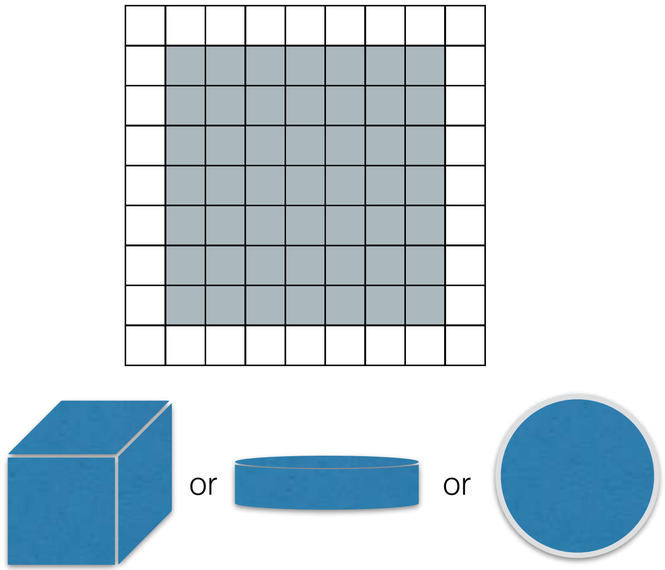


$$\frac{\partial \rho}{\partial t} + (\boldsymbol{v} \cdot \nabla)\rho = -\rho(\nabla \cdot \boldsymbol{v})$$
$$\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \nabla)\boldsymbol{v} = -\frac{\nabla P}{\rho}$$
$$\frac{\partial u}{\partial t} + (\boldsymbol{v} \cdot \nabla)u = -\frac{P}{\rho}(\nabla \cdot \boldsymbol{v})$$

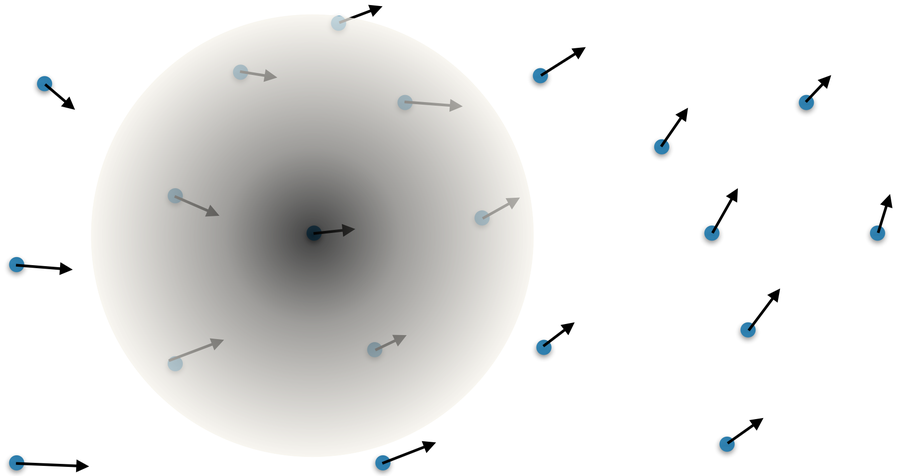
SIMULATING THE UNIVERSE WITH SMOOTHED PARTICLE HYDRODYNAMICS

e.g. Lucy (1977), Monaghan & Gingold (1977), Monaghan (1992, 2005), Price (2012)

Grid



Particles



$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{v}$$

$$\rho(\mathbf{x}) = \sum_j m_j W(|\mathbf{x}_i - \mathbf{x}_j|, h)$$

Hamiltonian hydrodynamics

$$L = \sum_j m_j \left(\frac{1}{2} v_j^2 - u_j \right)$$

$$+$$

$$\mathrm{d}u = \frac{P}{\rho^2} \mathrm{d}\rho$$

$$+$$

$$\nabla \rho_i = \frac{1}{\Omega_i} \sum_j m_j \nabla W_{ij}(h_i)$$

$$+$$

$$\frac{\mathrm{d}}{\mathrm{d}t} \left(\frac{\partial L}{\partial \mathbf{v}} \right) - \frac{\partial L}{\partial \mathbf{r}} = 0$$

$$=$$

$$\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = - \sum_j m_j \left[\frac{P_i}{\Omega_i \rho_i^2} \nabla W_{ij}(h_i) + \frac{P_j}{\Omega_j \rho_j^2} \nabla W_{ij}(h_j) \right]$$

Emmy Noether

Symmetries = conservation laws!

e.g. Gingold & Monaghan 1982; Monaghan & Price 2001; Monaghan 2002; Springel 2002; Price & Monaghan 2004b, 2007; Price 2012

THE PHANTOM SMOOTHED PARTICLE HYDRODYNAMICS CODE

Price & Federrath (2010); Lodato & Price (2010); Price et al. (2018) and many papers since

- **Open source code** for astrophysical fluid dynamics with SPH github.com/danieljprice/phantom
- **~100,000 lines** of modern Fortran, codebase started from scratch in 2007, went public in 2017
- **openMP and MPI** parallelisation
- Lots of **physics modules** (magnetic fields, multi-species dust, chemical networks, GR, radiation etc.)
- **Active user community** (7 users workshops since 2018; five in Melbourne, two in Europe, one in Canada)
- Extensive **unit testing** framework

Open

Phantom is free to use, download and redistribute under the terms of the GPLv3 license. We also welcome contributions to the code via the GitHub repo. Just get in touch!

Modular

Phantom is built in small, re-usable modules, making it easy to add new physics to the code.

Modern

All modules are written in modern Fortran and we enforce strict adherence to the very latest Fortran standards.

Lean

We strive for a low memory, high performance code with as few options as possible. It should "just work". Phantom is not a code for testing algorithms, it is a "take the best and make it run fast" production code for astrophysical simulations.

Tested

Phantom contains a comprehensive testsuite that runs on every pull request before it is merged to master. We strive to continually increase the scope of the tests to cover every aspect of the code.

Re-useable

We aim to never repeat code.

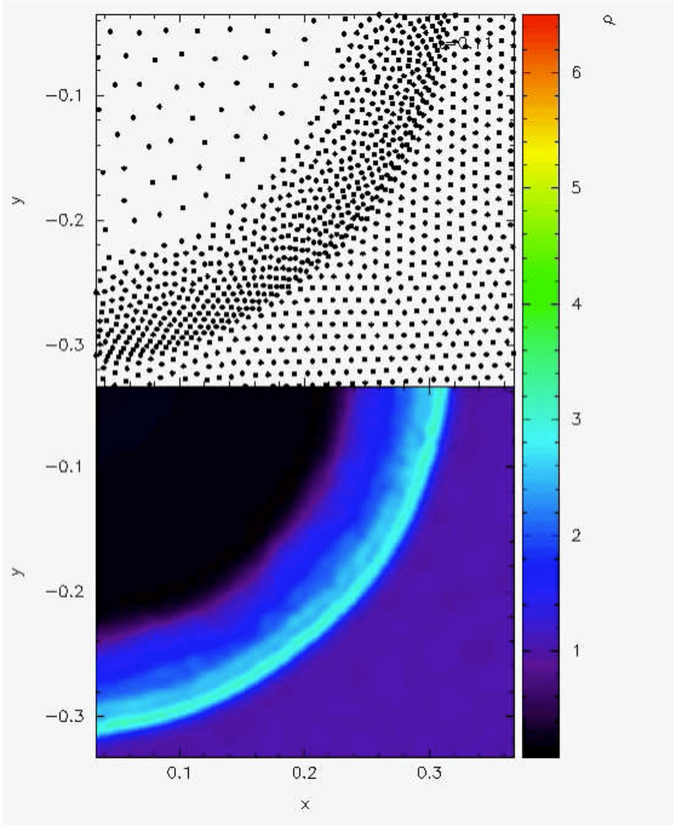
HOW TO VISUALISE RESULTS FROM SPH CODES

$$A(\boldsymbol{r}) = \sum_j \frac{m_j}{\rho_j} A_j W(|\boldsymbol{r} - \boldsymbol{r}_j|, h)$$

e.g. Gingold & Monaghan (1977)

Discrete

Continuum

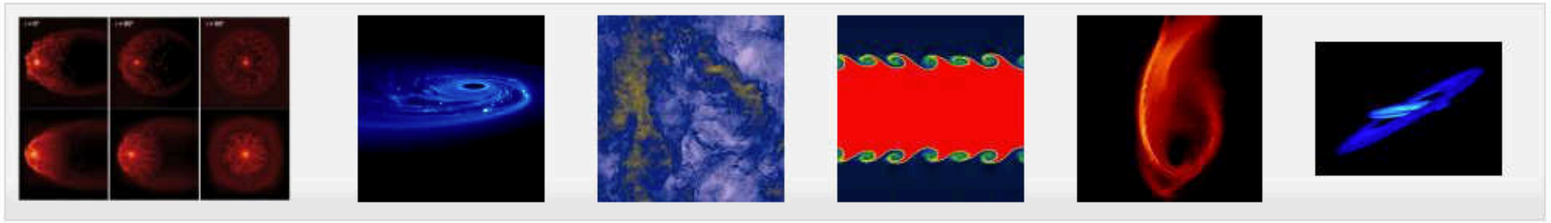


github.com/danieljprice/splash

c.f. Price (2007), PASA, 24, 159

SPLASH - A VISUALISATION TOOL FOR SMOOTHED PARTICLE HYDRODYNAMICS

Price (2007)



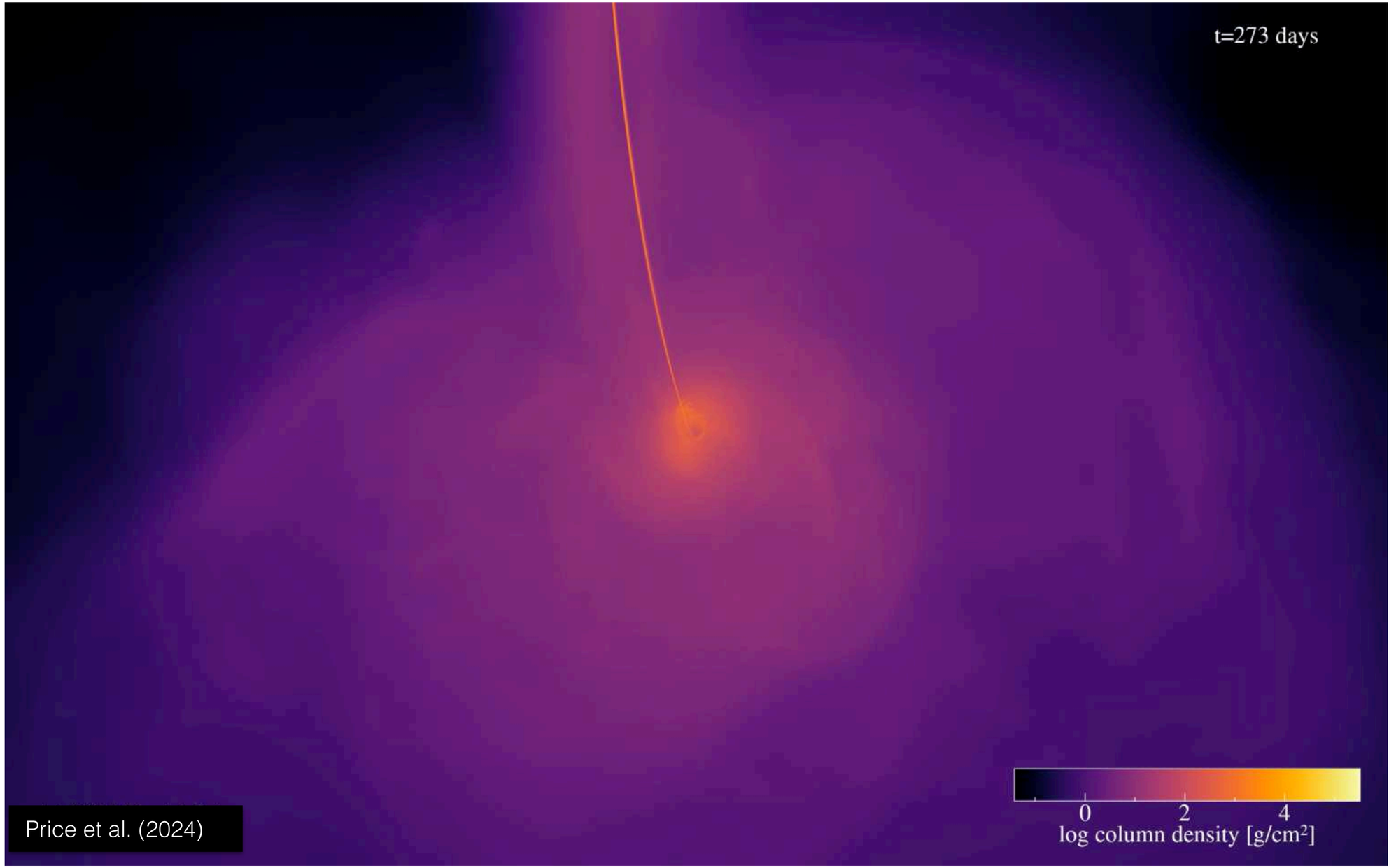
SPLASH - A visualisation tool for smoothed particle hydrodynamics

- **Open source code** for SPH interpolation/visualisation github.com/danieljprice/splash
- **~90,000 lines** of modern Fortran (mostly format readers!), original F77 codebase started in 2002
- Key requirement is **remote visualisation**, by interactive X-window sent over the network. This is because fluid simulations generate large data files that are much easier to visualise in situ.
- Originally used old FORTRAN-based PGPLOT graphics library, eventually rewrote our own graphics library in C (github.com/danieljprice/giza) based on the Cairo drawing library.
- Being pure C, **Giza is easily callable from Fortran**, has few dependencies and integrates well with system libraries
- Both codes used CVS version control in 2003, transitioned to SVN in 2009, transitioned to git/svn and eventually pure git around 2016.
- **openMP** parallelisation of key routines

```
commit 9c044239ffa1da28ab33ff8ce7031cd7c3e8f6a5
Author: dprice <dprice>
Date:   Mon Dec 15 13:12:46 2003 +0000

    freeform source -> to .f90

commit 0835dfc0e3484a19636eaa7deb5406919d6949dd
Author: dprice <dprice>
Date:   Mon Dec 15 12:47:03 2003 +0000

    lower case

commit 932be6b812538ffd7d9604d733da0d1b40331f16
Author: dprice <dprice>
Date:   Tue Dec 9 15:38:48 2003 +0000

    power spectrum of 1D data (some bugs still)

commit ac3e2b2e93b3ec26a9324ba0df41c973affb2244
Author: dprice <dprice>
Date:   Mon Dec 8 21:11:58 2003 +0000

    minor changes for spherical blast wave

commit deea7294091e525892871ed81c6958eac060c7ff
Author: dprice <dprice>
Date:   Mon Nov 24 21:29:46 2003 +0000

    update

commit 7cccc73327da84a9b4ec16e029d4c52f1bcfcd49
Author: dprice <dprice>
Date:   Mon Nov 24 21:08:52 2003 +0000

    calc_quantities added, rhoh moved

commit 31215cc7f3fdd37f293d04c171a6be2273ad3c1a
Author: dprice <dprice>
Date:   Thu Oct 30 16:07:37 2003 +0000

    fix bug when no data

commit 8002af961098d67721c1583d4e5d95b2862d6d3c
Author: dprice <dprice>
Date:   Wed Oct 22 14:41:02 2003 +0000

    Initial revision
```

EXAMPLE: TIDAL DISRUPTION OF A STAR BY A SUPERMASSIVE BLACK HOLE

Price et al. (2024)
"Eddington envelopes: the fate of stars on parabolic orbits tidally disrupted by supermassive black holes", Astrophysical Journal Letters 971 , L46



Simulation with github.com/danieljprice/phantom; visualisation with github.com/danieljprice/splash

WHAT IS THE PROBLEM WITH COMPUTATIONAL WORK?

- Humans can type fast, but not usually accurately
- Physics modules typically developed “quickly” for one project tend to be reused by others for years
- Codes maintained by individuals or small teams
- 99% of development time is spent debugging
- Bugs can really waste peoples lives
- This is why I started a fresh codebase and keep to modern standards, to allow the compiler to debug as much as possible!

Stable smoothed particle magnetohydrodynamics in very steep density gradients

Monthly Notices of the ROYAL ASTRONOMICAL SOCIETY
MNRAS 464, 2499–2501 (2017) doi:10.1093/mnras/stw2474
Advance Access publication 2016 September 28

Erratum and Addendum: Smoothed particle magnetohydrodynamic simulations of protostellar outflows with misaligned magnetic field and rotation axes

by Benjamin T. Lewis,[1,2]★
[1] School of Physics and Astronomy, University o
[2] Monash Centre for Astrophysics, School of Ma

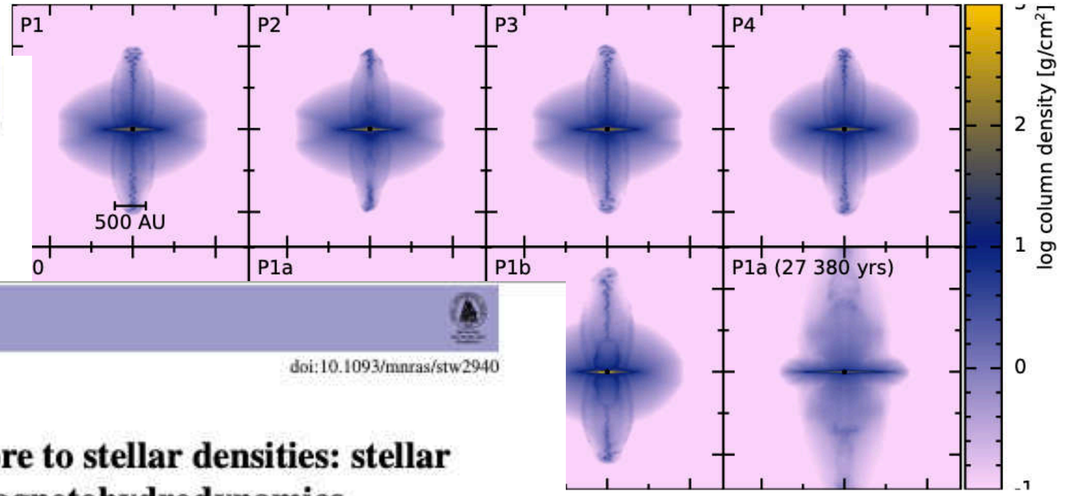
Key words: accretion, accretion discs

The paper ‘Smoothed particle magnet
tions of protostellar outflows with misalig
rotation axes’ was published in MNRAS 45
‘the Original Paper’).
The calculations presented in that work

Monthly Notices of the ROYAL ASTRONOMICAL SOCIETY
MNRAS 465, 2714–2716 (2017) doi:10.1093/mnras/stw2940
Advance Access publication 2016 November 14

Erratum: Collapse of a molecular cloud core to stellar densities: stellar core and outflow formation in radiation magnetohydrodynamics simulations

by Matthew R. Bate,[1]★ Terrence S. Tricco[2] and Daniel J. Price[2]
[1] School of Physics and Astronomy, University of Exeter, Stocker Road, Exeter EX4 4QL, UK



collapse of a magnetised molecular cloud core of total mass 1 $M_\odot$ with a
h models described in § IV, the stability and essentially identical evolution
tly explodes. Model P1a is the same as model P1, but with the h-averaging
is timestep, it becomes unstable shortly after (as seen in the far-right plot on
omentum equation Eqn. (12) and, whilst significantly improved over P0, still

TEST-DRIVEN DEVELOPMENT

Test-driven development

Article Talk

From Wikipedia, the free encyclopedia

Test-driven development (**TDD**) is a way of writing code that involves writing an automated unit-level test case that fails, then writing just enough code to make the test pass, then refactoring both the test code and the production code, then repeating with another new test case.

Alternative approaches to writing automated tests is to write all of the production code before starting on the test code or to write all of the test code before starting on the production code. With TsDD, both are written together, therefore shortening debugging time necessities.[1]

TDD is related to the test-first programming concepts of extreme programming, begun in 1999,[2] but more recently has created more general interest in its own right.[3]

Programmers also apply the concept to improving and debugging legacy code developed with older techniques.[4]

- Write tests BEFORE writing the code
- Helps to frame the problem you are trying to solve, and define what a routine or module is supposed to achieve
- Choice of testing “framework” is unimportant, just has to return pass/fail in some way and count them (e.g. can easily write a script to do this yourself)
- Even easier in GitHub actions workflows

UNIT TEST: SIMPLE EXAMPLE

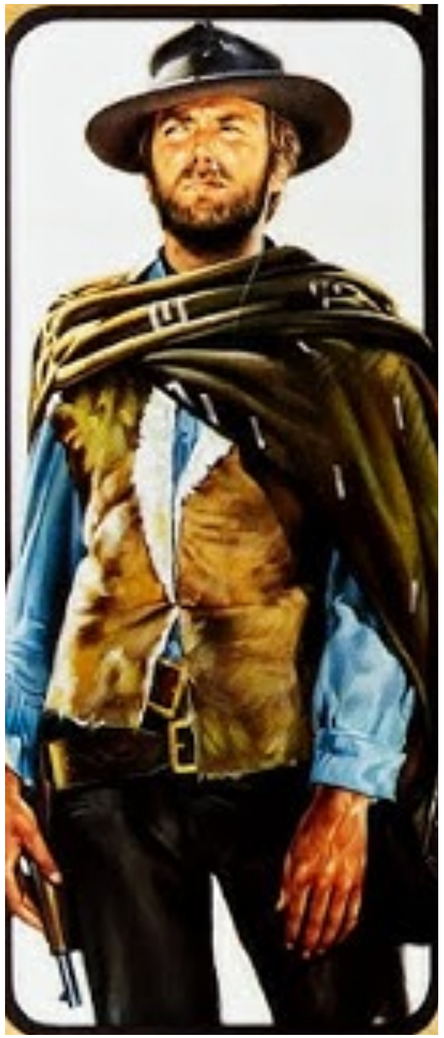
```
!-----------------------------------------
!+
!  Unit tests of the quartic solver
!+
!-----------------------------------------
subroutine test_quartic(ntests,npass)
 use quartic, only:quarticsolve,quarticf
 integer, intent(inout) :: ntests,npass
 real :: a(0:3),xold,x
 integer :: ierr,nfail(2)
 logical :: moresweep
 real, parameter :: tol = 1.e-6

 if (id==master) write(*,"(/,a)") '--> checking x^4 = 16 gives x=2'

 a = [-16.,0.,0.,0.] ! a0 + a1*x + a2*x^2 + a3*x^3 + x^4 = 0
 xold = -2e6
 !print*,'a = ',a
 call quarticsolve(a,xold,x,moresweep,ierr)
 call checkval(x,2.,tol,nfail(1),'x=2')
 call checkval(ierr,0,0,nfail(2),'ierr=0')
 call checkval(quarticf(a,x),0.,tol,nfail(1),'f(x)=0 for solution found')
 call update_test_scores(ntests,nfail,npass)

 if (id==master) write(*,"(/,a)") '--> checking arbitrary quartic'
 !a = [-6.,-2.,3.,0.]  ! this one fails: solution is real and negative
 !a = [-6.,2.,3.,0.]
 !a = [-6.,2.,3.,2.]
 a = [-6.,2.,3.,1.]
 !a = [0.,1.,-2.,0.]   ! x = 0 is a solution: this also fails
 xold = 1.
 call quarticsolve(a,xold,x,moresweep,ierr)
 call checkval(quarticf(a,x),0.,tol,nfail(1),'f(x)=0 for solution found')
 call checkval(ierr,0,0,nfail(2),'ierr=0')
 call update_test_scores(ntests,nfail,npass)

end subroutine test_quartic
```

- You perform these kinds of tests naturally when debugging, so the idea is just to automate them
- It is shocking how often tests like this fail

MY FORTRAN WISH LIST

THE GOOD

- Modules are an excellent way to group related functionality
- Fortran 90 was overall implemented very well
- Code mirrors the maths (*For*mula *Trans*lation lives up to its name)
- Easy to write code that runs fast
- Fortran does package management perfectly: There is a good and intuitive standard library of functions. Can program Fortran without having to constantly be on Stack Overflow
- It's the right tool for a particular job: solving mathematical equations on the computer. Does this job well.
- Backwards compatibility means that code still works 20 years later (with means to enforce obsolescence if desired, but old code does not break by default)
- Open source Fortran 90 compilers (gfortran) changed the game due to widespread availability, easy installation via package managers and avoidance of headaches managing licences. Also these compilers have continued to improve via community contributions.
- Easy parallelisation with openMP (and MPI though not "easy")
- Compilers pick up most (but not all) of the errors, especially with bounds-checking
- Interoperability with C in later standards is extremely useful



THE BAD

- Precision-handling is a bit of a mess, e.g. typing 0.0_dp everywhere is ugly and error-prone. My solution is to declare everything as default real and use -fdefault-real-8 -fdefault-double-8 (equivalent to -r8 which was always the standard flag for this but not in gfortran??)
- End up constantly re-writing low-level modules for simple things (integrating a function, inverting a matrix, cross product of two vectors) that should really be in the language
- No obvious way to share modules between two projects except by copying the module over to the other project. These two versions then diverge with time…
- Dependencies are hard to manage properly in Makefiles (but make has the great advantage of being simple, unlike cmake)
- Everyone invents their own file format, for both parameter files and binary code outputs. Fortran namelists were a good idea, but no way to format the output nicely (e.g. as toml). Depending on a giant library like hdf5 is also a mess, but would be simple if this was supported natively.
- Co-arrays are a great idea, but have not seen this feature used “in the wild” because performance matters and easier for most people to write their own low-level MPI code. Shows need to be led by user adoption of working libraries, not just putting things in the standards…
- No native plotting/visualisation (see giza…)

```
# input file for binary setup routines

# units
          mass_unit =      solarm    ! mass unit (e.g. solarm,jup
          dist_unit =      solarr    ! distance unit (e.g. au,pc,

# options for body 1
           iprofile1 =           5    ! 0=Sink,1=Unif,2=Poly,3=Den
          isoftcore1 =           2    ! 0=no core softening, 1=cub
     input_profile1 =  profile9donor.data   ! Path to input profi
    outputfilename1 =  mysoftenedstar.dat   ! Output path for sof
            rcore1 =        10.0    ! Radius of core softening
            mcore1 =         9.5    ! Initial guess for mass of
            lcore1 =     0.*lsun    ! Luminosity of point mass s
              np1 =       10000    ! number of particles

# options for body 2
           iprofile2 =           5    ! 0=Sink,1=Unif,2=Poly,3=Den
     input_profile2 =  profile9accretor.data   ! Path to input pr
        isoftcore2 =           0    ! 0=no core softening, 1=cub
        isinkcore2 =           F    ! Add a sink particle stella
            lcore2 =     0.*lsun    ! Luminosity of sink core pa
```

THE UGLY

- Libraries in Fortran are a mess because .mod format is compiler-dependent. My solution is to write C code to interface with system libraries and call C from Fortran (which is done well)
- No sensible way to share modules with the world or for the community to contribute libraries/modules to the language. Even within attempts at this (fpm) module dependencies get tangled quite quickly. Hard to pull a module from one code and use it in another.
- It's easier to use Fortran libraries from Python than from Fortran!
- Numpy shows all the additional commands that should be part of the core Fortran language (e.g. linspace, logspace, roll, etc). But Python package management is also a nightmare that has taken many years to find good solutions for (conda, virtual envs, etc)

A (PARTIAL) WISH LIST

- Greatly expanded standard library (NOT package management) that is required to be available in every Fortran compiler. The library itself should not have to be written by vendors. https://stdlib.fortran-lang.org/ is a great effort in this direction (*but I have no idea how to use it!*). Versions of this should be pinned to the language standard.

- Header files (e.g. from submodules) that are not compiler dependent. In practice I share the raw .f90 module file containing the definitions to be compiled by the calling code.

- Package management like the fpm effort, but made available natively when installing gfortran or other compilers. No need for separate installation. Package management should not be required for standard libraries, and popular packages should be integrated into the language over time, unlike what happens in Python (why do I have to import numpy as np just to take a sqrt?).

- Toml format for namelist input/output

- Native hdf5 support (or similar) for high level input/output of code snapshots

- Build on success of openMP, openACC integration in Fortran, need further support for hybrid CPU/GPU parallelisation, similar to SYCL effort for C++

- It's a positive thing to have implementations ahead of the standard, some of the problems in Fortran have arisen from trying to be too "top-down"

Huge thanks to everyone who has contributed to Fortran over the years

Comments and feedback welcome!