Pythonic Geodynamics

Gabriele Morra

Department of Physics and School of Geosciences, University of Louisiana at Lafayette, United States

David A. Yuen
Department of Earth Sciences, University of Minnesota
School of Env. Studies, China Univ. of Geosciences, Wuhan

Sang-Mook Lee
School of Earth Sciences
Seoul National University, S Korea

Collaborators, Students and former Students at UL at Lafayette

Dr Raphael Gottardi
Prasanna Gunawardana (Monash Univ.)
Daniel Conlin (Now at Exxon Mobil)
Kyle Spezia (Now at Hulliburton)

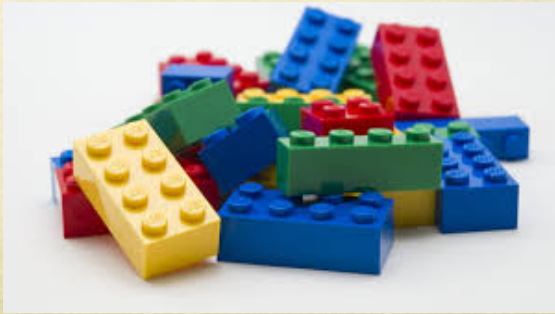
Brian Fischer
Brian Dye
Brennan Brunsvik
Saurav Gautam

Motivation

- Computational methods and data analysis play a constantly increasing role in Earth Sciences, however students and professionals need to climb a steep learning curve before reaching a sufficient level that allows them to run effective models.

- By using Python undergraduate and graduate students can learn advanced numerical technologies with a minimum dedicated effort, which in turn encourages them to develop more numerical tools and quickly progress in their computational abilities.

- Python allows combining modules and functions as pieces of LEGO, therefore enormously simplifying the life of the scientific geo-modeller.

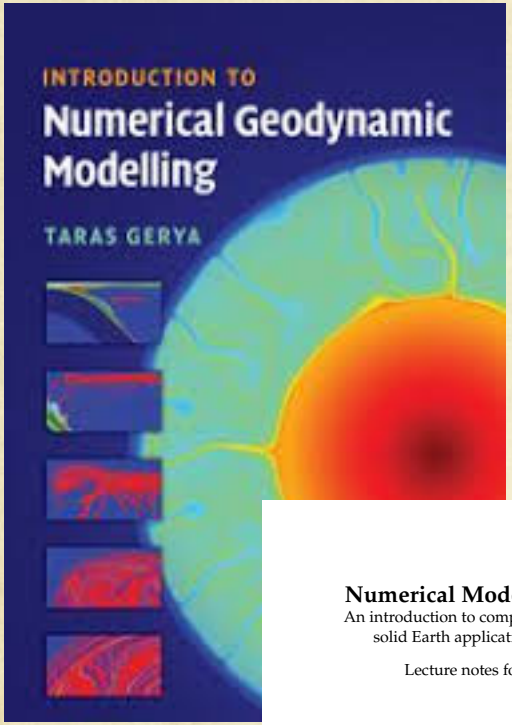


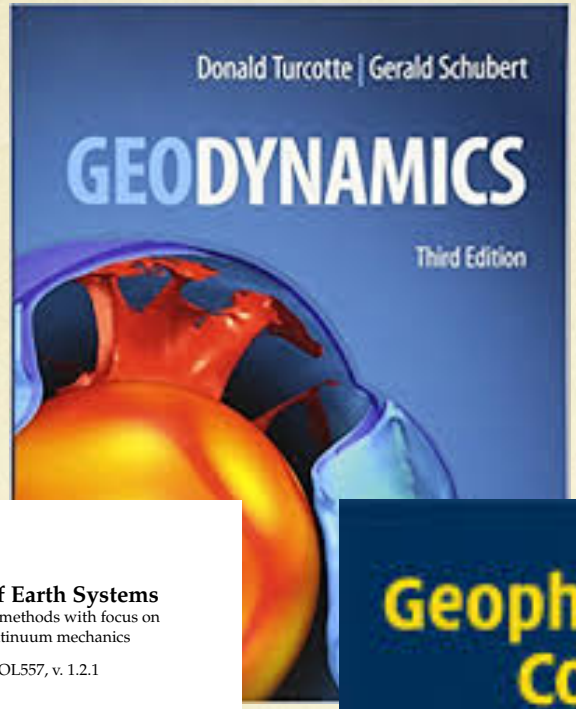
Motivation

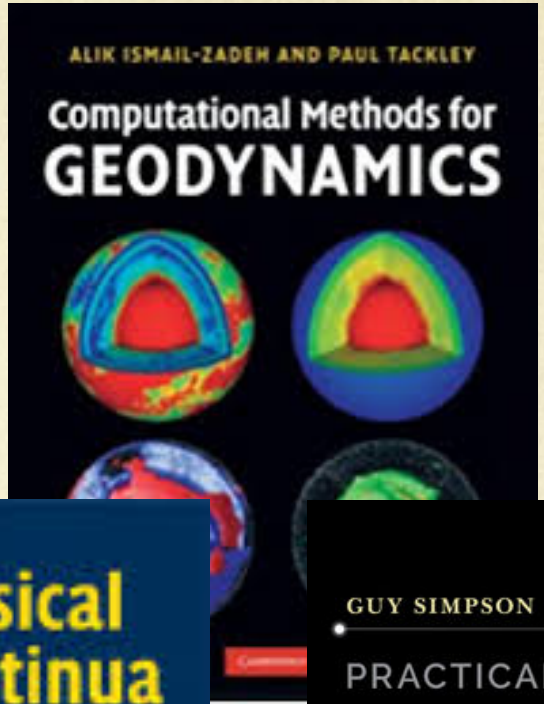
Adults, like children, learn by playing. With this insight in mind, I wrote a non-exhaustive and non-overambitius textbook that aims at introducing computational geodynamics to young students, either undergraduates or beginning graduates.

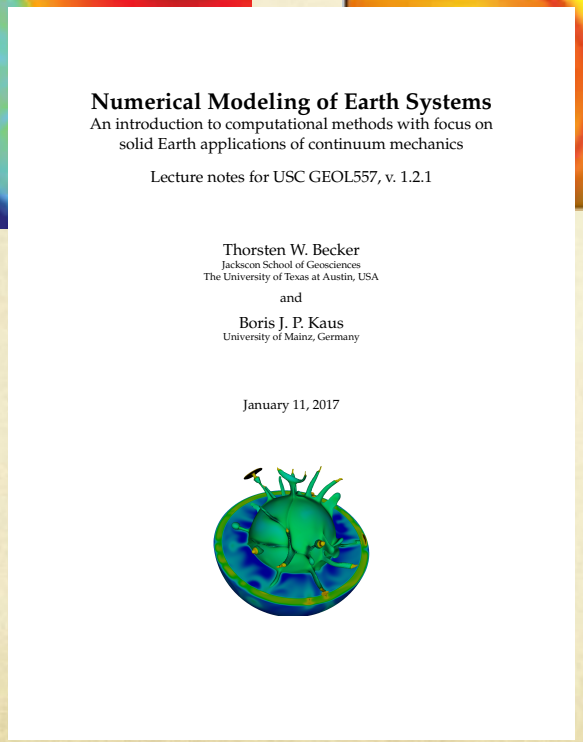
The goal, more than trying to cover a topic that is way too vast for a book to contain it, is to show some of the fundamental strategies available with a strong focus on practical, playful, easy-to-use techniques.

Existing textbooks for geodynamics









Numerical Modeling of Earth Systems
An introduction to computational methods with focus on solid Earth applications of continuum mechanics

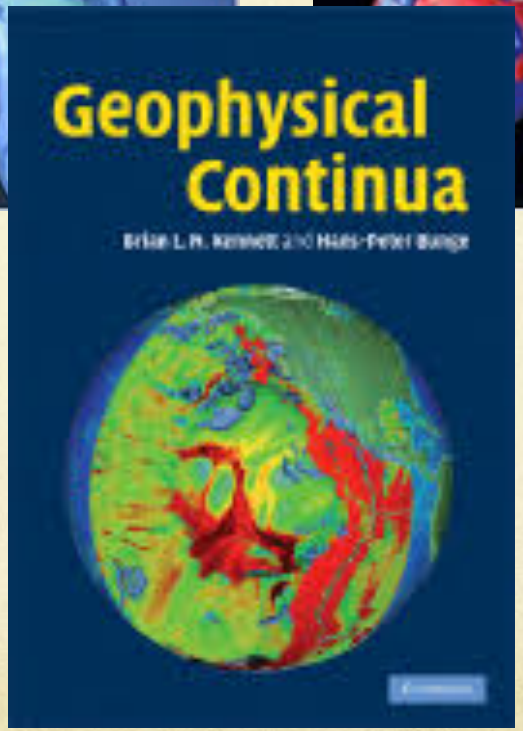
Lecture notes for USC GEOL557, v. 1.2.1

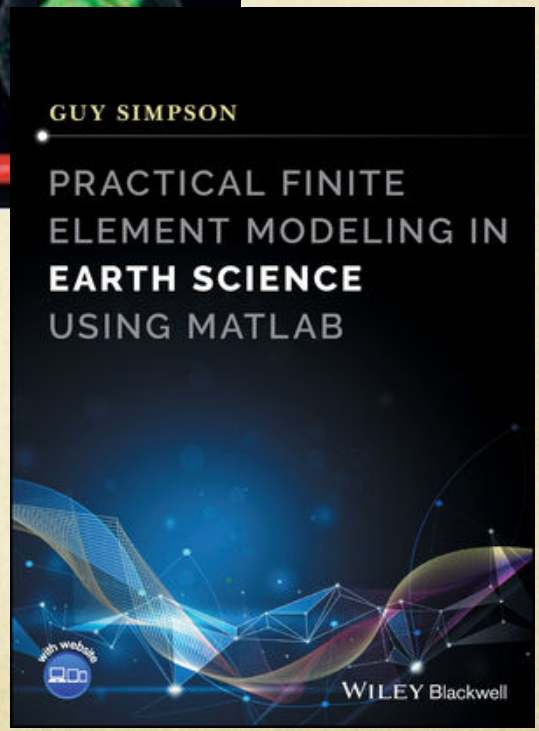
Thorsten W. Becker
Jackscon School of Geosciences
The University of Texas at Austin, USA

and

Boris J. P. Kaus
University of Mainz, Germany

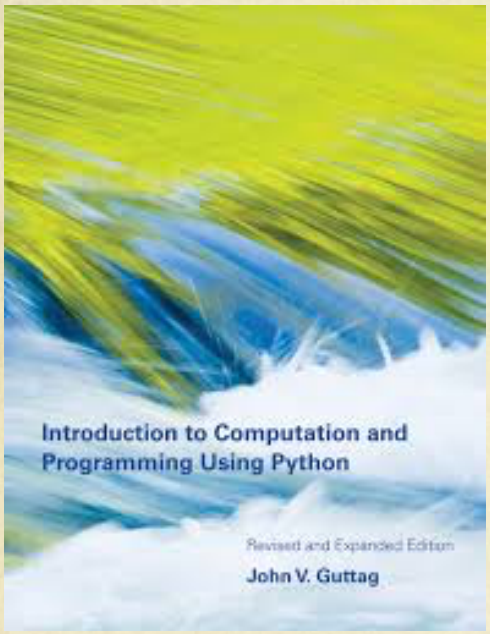
January 11, 2017





Pedagogy of High Performance Computing

A new generation of programming languages is emerging and replacing C, C++ and Fortran. Python is the most used, but other options have emerged emerge such as Julia and Ruby, or Java based Scala and Hadoop. Presently Python is the easiest, most compact and powerful, but the future is open.



Some universities offer a mandatory “Introduction to Computer Science and Programming” course at the beginning of every science program. Future geoscientists will use computing for every task. The earlier they familiarize with how computers “think”, the sharper they will use their computing tools.

Tensor Flow has been freely released by the Google Brain Team to accelerate machine learning research. All major libraries for Tensor Flow are in Python, as well as for the other machine learning tools.

TensorFlow

Students and Professionals need to Rapidly Learn Programming, Numerical Modeling and Big Data Analysis. But how?

- Most people used Matlab, a proprietary software. Python today offers the same, but in an open and fast growing environment. It reminds the time when Linux took over Unix.
- It is efficient for ODE solutions with the Numpy module.
- It offers the same visualization tools of Matlab with Matplotlib
- It goas at c-speed with just in time compilation (Cython, etc)
- Can be parallelized easily with mpi4py, PyCuda (for GPUs) and petsc4py)
- It is designed for processing large amount of data (e.g. Pandas)

Pythonic Geodynamics

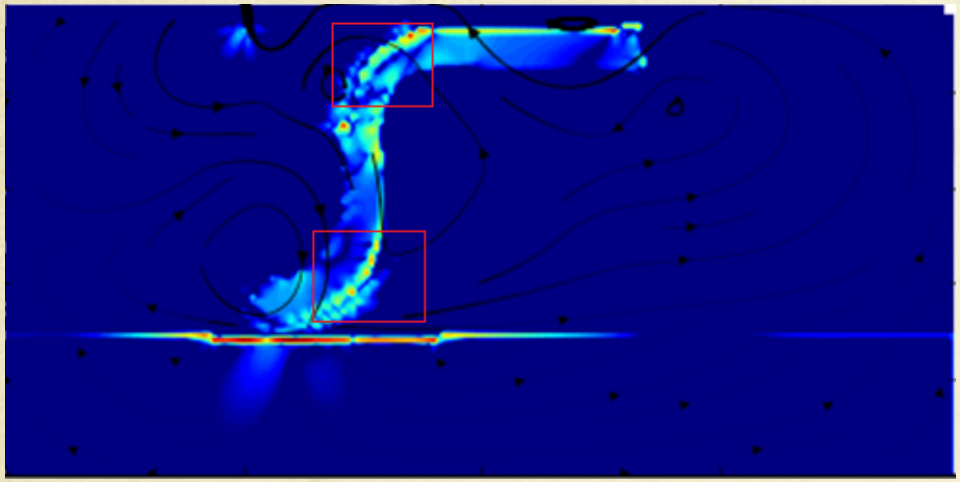
Points that I will touch today

- Student cases
- Bird's eye View: Python, Programming, Anaconda & Canopy, Libraries
- Visualization and Data formats: Matplotlib, NetCDF, ObsPy, VTK, Paraview
- The Jupyter revolution. I will show several examples
- Fast Python: NumPy, Cython, mpi4Py and more
- Monte Carlo Simulation of the Pyroclastic flow During the 1944 Mt Vesuvio Volcanic Eruption
- Advection: eulerian and lagrangian
- Operators are "abstract toys" to play with
- Non-linear flow. 3D. Overview of some more advanced techniques. Tree-codes, Surface Integrals, etc

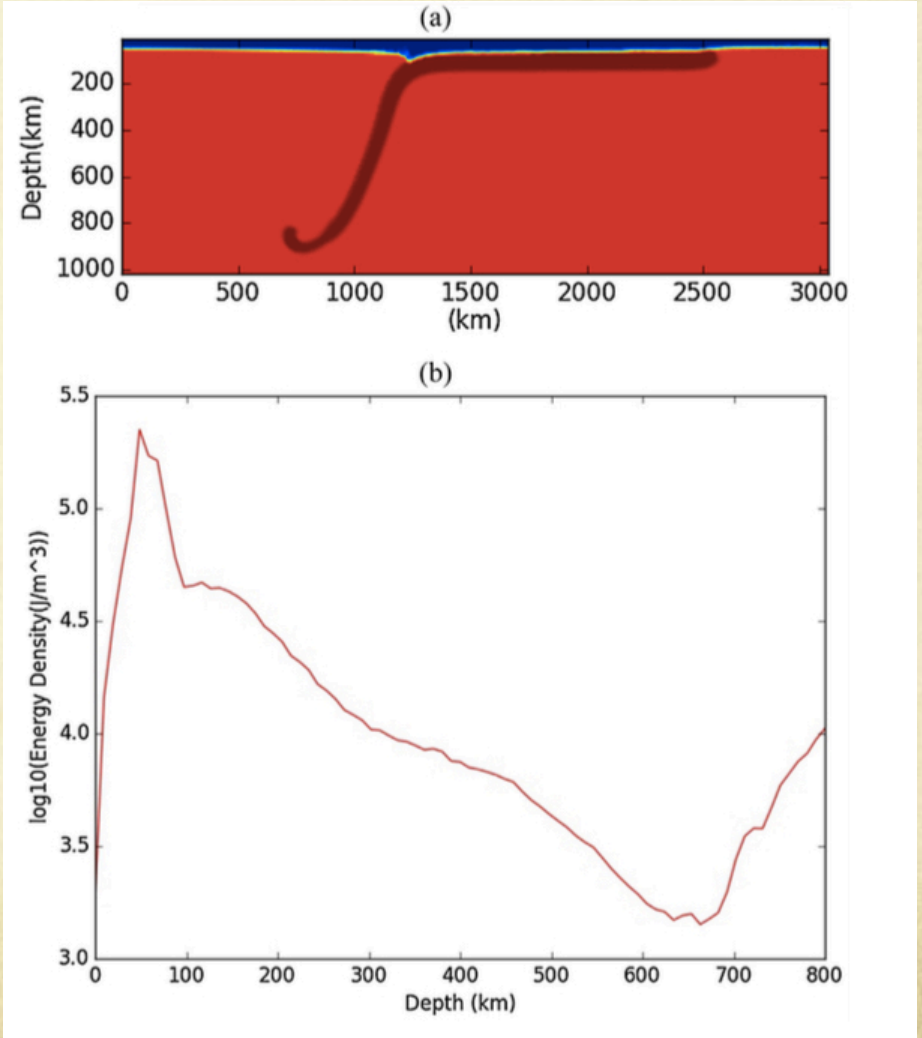
Students

Student 1

A student coming from an engineering background came to do his master at UL at Lafayette. In one year he learned to Program, the Particles in Cell Method, and geodynamic modeling of lithosphere and mantle.



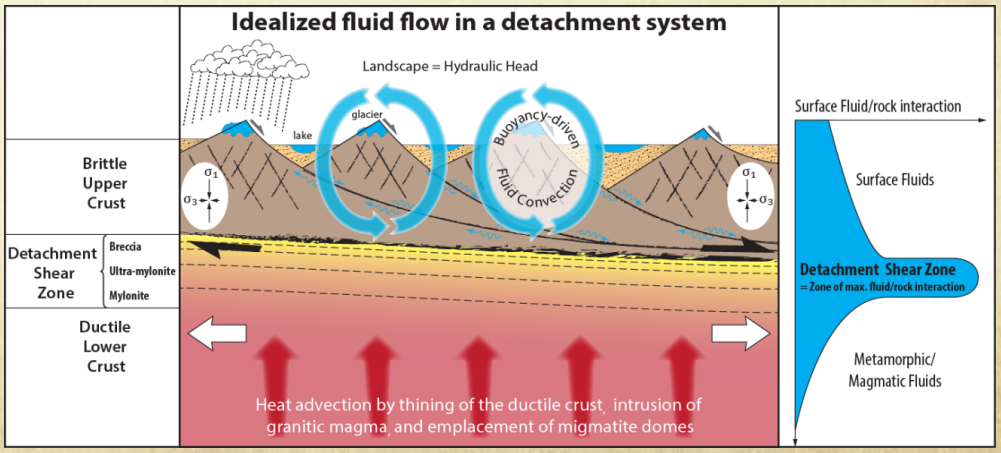
Gunawardana and Morra,
Journal of Geodynamics 106 (2017) 33–45



Student 2

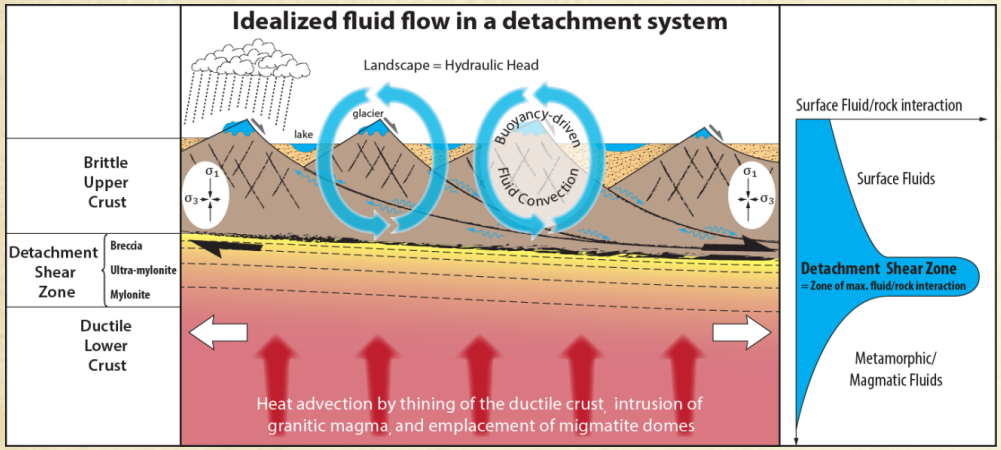
He was a student in difficulty, close to leave without completing his studies. By using Python he could learn to write a finite volume solver for Darcy Flow in 3 months, just using the numerical libraries, NumPy. He completed successfully his thesis by writing a code of only 50 lines.

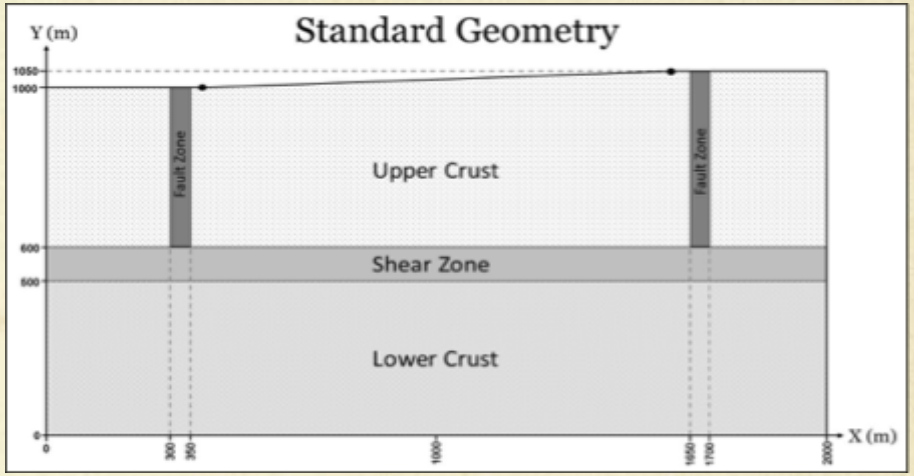
```
diffusivity1D=diffusivity.reshape(nxc*nyc)
diffSpMatrix=sparse.spdiags([diffusivity1D],[0],nxc*nyc,nxc*nyc).tocsc()

# create the Laplacian Operator
LaplacianOp=PIC.sparseVariableLaplacianOperator(nxp,nyp,dx,dy,diffSpMatrix)
LaplacianOp=PIC.addBC(LaplacianOp,nxp,nyp)*dx*dx
L=sparse.eye(nxp*nyp).tocsc()-LaplacianOp*deltaTime

# create initial pressure
pHydro1 = np.outer(np.ones(X.size),((yMax-Y)*density*gravity))
pressure2[F1toF2]=(X-0.325)*density*gravity*0.05
pressure2[postF2]=(1.675-0.325)*density*gravity*0.05
pFluid=  np.outer(pressure2,np.ones(Y.size))  #+ pHydro1
pFluid=pFluid.reshape(nxp*nyp)
for thisStep in np.arange(1000):
    pFluid = la.spsolve(L,pFluid)
pFluid=pFluid.reshape(nxp,nyp)

G=PIC.sparseGradientOperator(nxp,nyp,dx,dy)
Zout = PIC.sparseZoomInOperator(nxp,nyp,dx,dy).transpose()

vx=-diffSpMatrix*G[0].dot(pFluid.reshape(nxp*nyp))
vy=-diffSpMatrix*G[1].dot(pFluid.reshape(nxp*nyp))
vx=Zout.dot(vx); vx=vx.reshape(nxp,nyp)
vy=Zout.dot(vy); vy=vy.reshape(nxp,nyp)
```

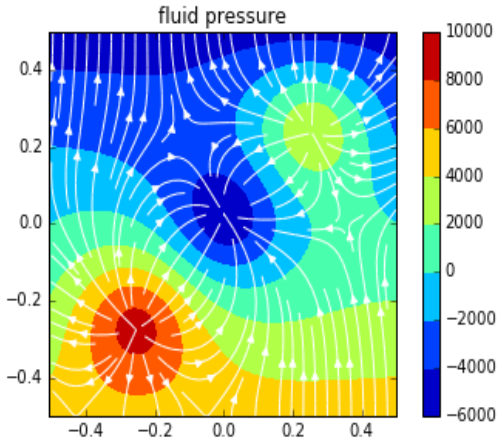


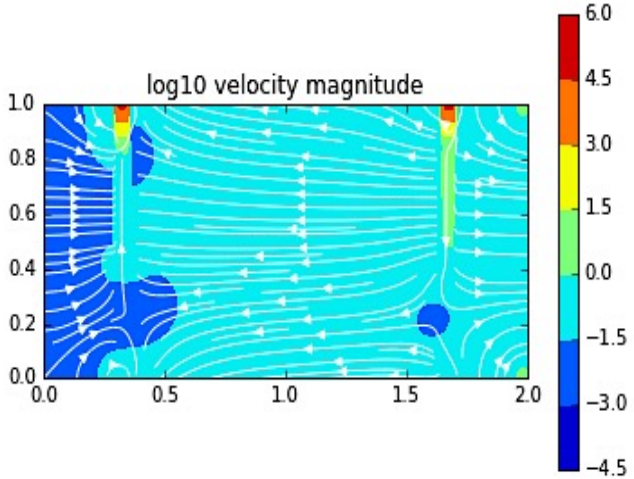
Student 2

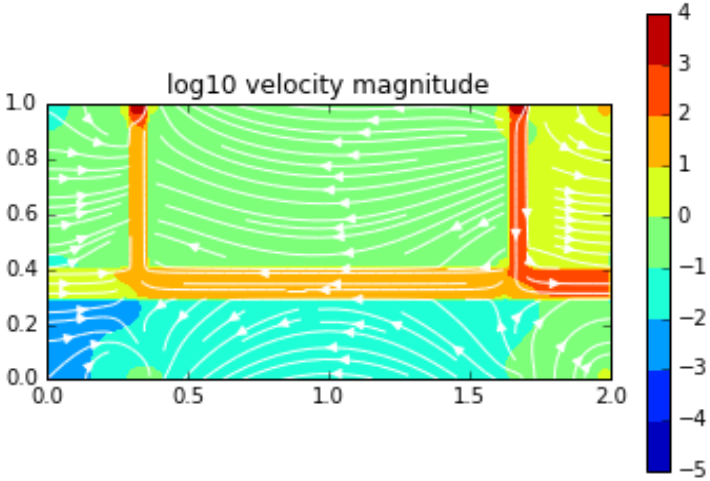
He was a student in difficulty, close to leave without completing his studies. By using Python he could learn to write a finite volume solver for Darcy Flow in 3 months, just using the numerical libraries, NumPy. He completed successfully his thesis by writing a code of only 50 lines.











Student case 3:

Three students of the Southern Methodist University, completing a Master of Science in Data Science, decided to do their capstone work on geodynamics. They contacted me and we started to work on volcano seismicity.

Python was central in all we did.

- Machine Learning was from SciKits Learn of Python.
- Seismic data were downloaded and processed using ObsPy.
- Data preparation and processing was all done with Python.

The Students in only three months were able to reproduce the results of professional volcanologists analyzing the strombolian activity of Villarica in Chile.

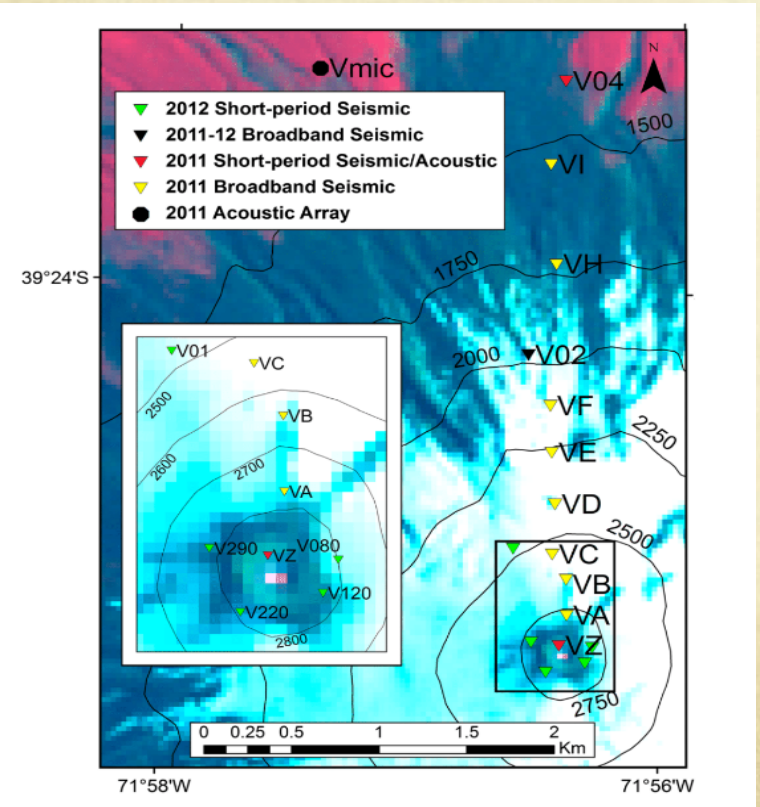
Student case 3

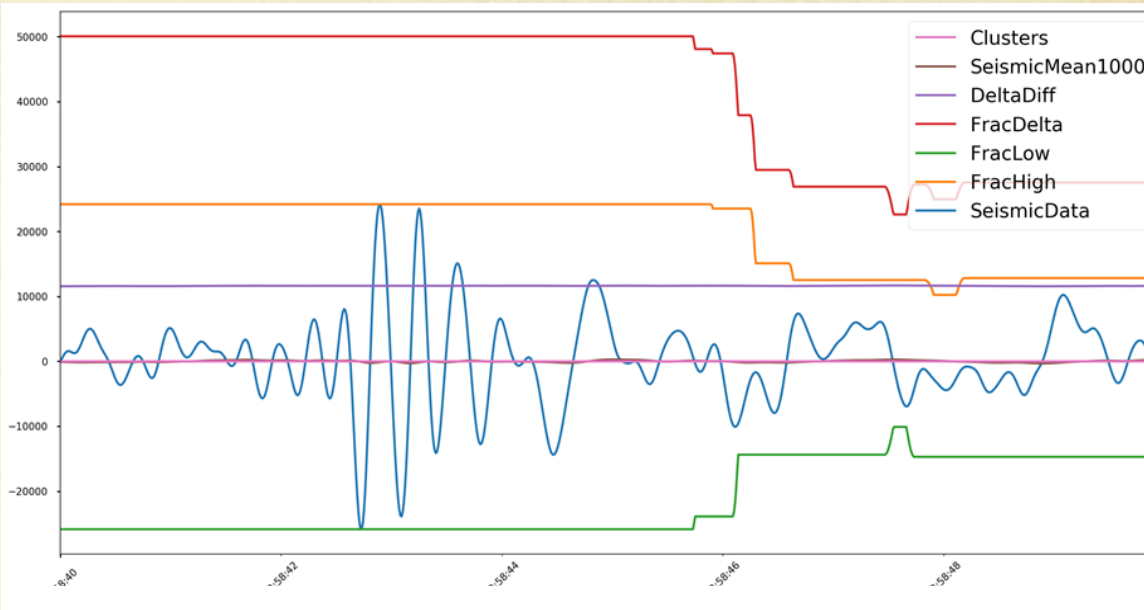
The data was gathered from publically available IRIS website with Obs Py

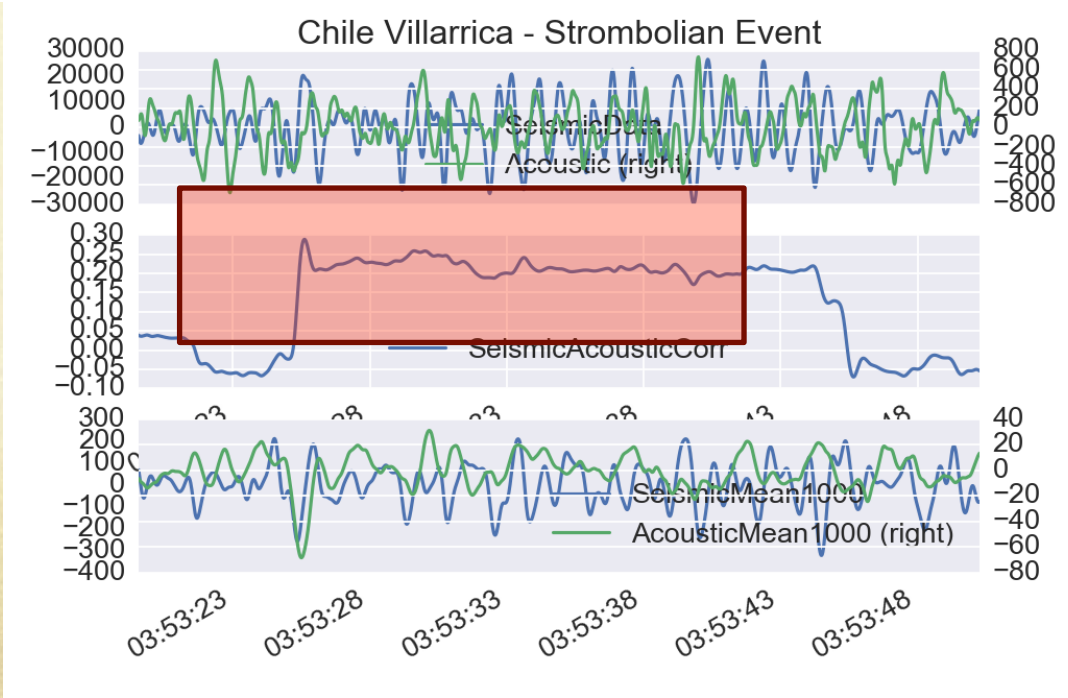
http://ds.iris.edu/

Chile's Villarrica Volcano

Richardson, Joshua P., Waite, Gregory P., Palma, Jose Luis,
"Varying seismic-acoustic properties of the fluctuating lava lake at Villarrica volcano, Chile,"







Bird's eye

- *Readability counts.*
  Simple is better than complex.
  Complex is better than complicated.
  If the implementation is hard to explain, it's a bad idea.
  If the implementation is easy to explain, it may be a good idea.

- – from The Zen of Python of T. Peters

History of Python

- Dutch programmer Guido van Rossum created the first implementation in December 1989, and released it for the first time in 1991. Benevolent Dictator for Life.
- Versions have been released on January 1994 (1.0), on October 2000 (2.0). In December 2008 the backward incompatible 3.0.
- Python 2.x will continue to be supported only until 2020. After that Python 3.x will become the only supported version.
- Some new features are essential in allowing Python programmers to use **threads** that are essential in the connected present world.

Literature on Python

- Programming in Python: “There should be one – and preferably only one – obvious way to do it. Although that way may not be obvious at first unless you're Dutch.” from The Zen of Python.

- Many books can introduce you to the Python Language. **Think Python** from Allen B. Downey, O'Reilly, is freely available.

- Complete manuals and tutorials are freely available on the Python Website (www.python.org) as well as on Code Academy.

- Hybrid Projects involve Cython (www.cython.org), the principle way for creating compiled extension for Numerical Python.

Distributions

Anaconda Spyder

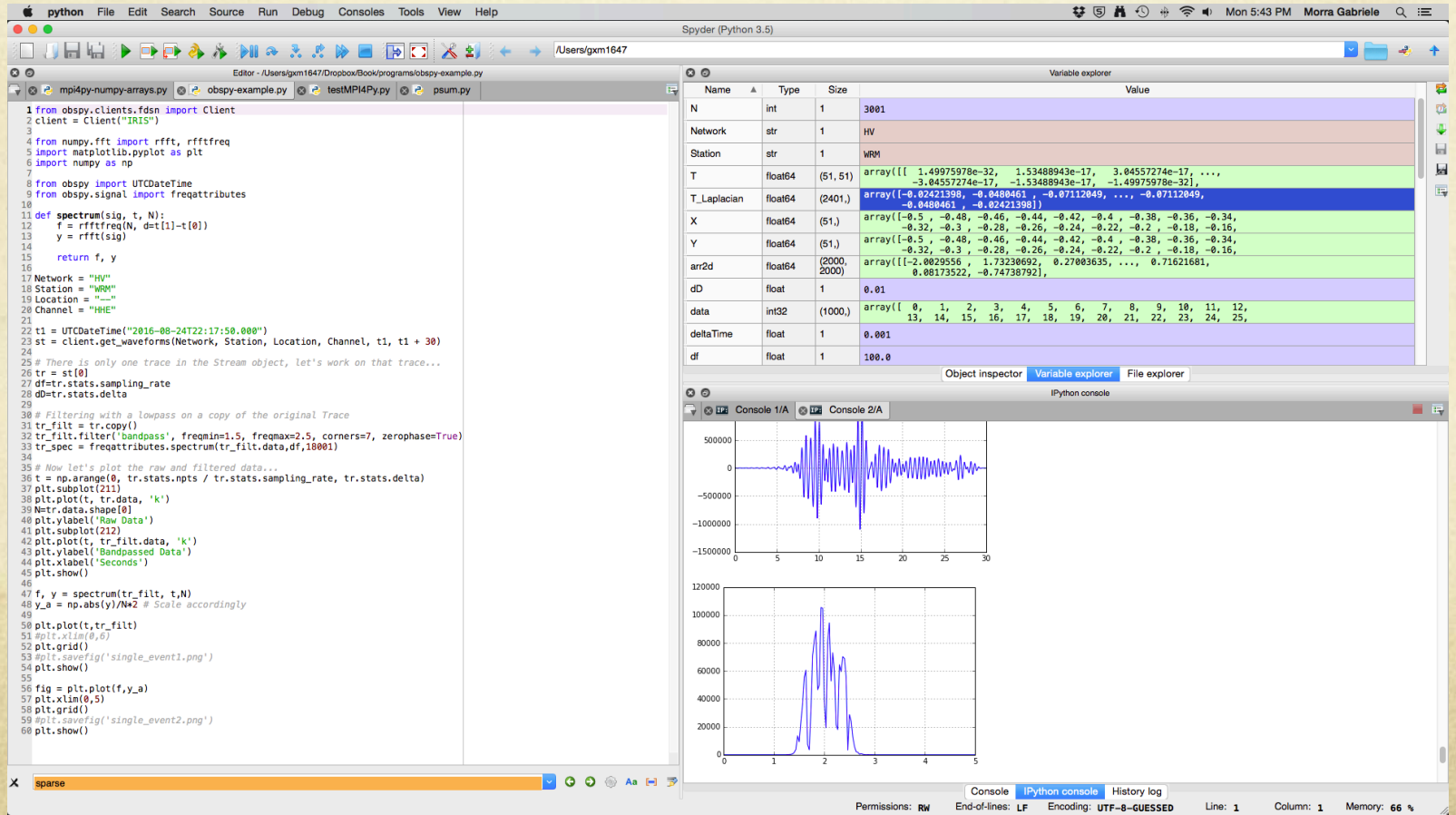
https://www.continuum.io/

Enthought Canopy

https://www.enthought.com/products/canopy/

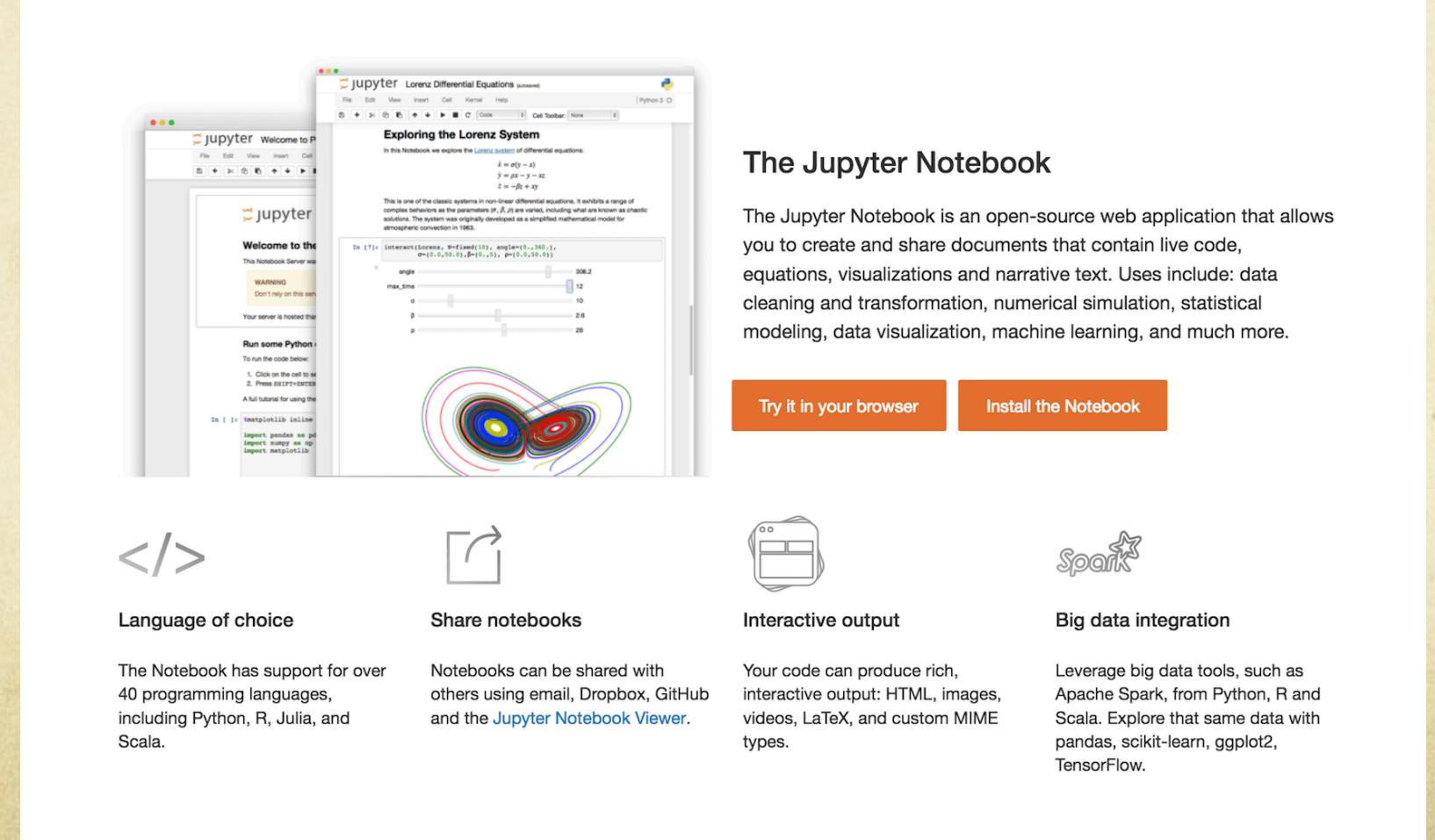
Python Interface

- *Canopy* and *Spyder* both have a similar graphical style to MatLab.
- Each of these interfaces are based on a "three windows" system in which one has a (i) editor, a (ii) object/variable explorer and a (iii) standard or iPython console.



Jupyter

- *Jupyter* combines the interactivty of *iPython* with the easy to use interaction style of *Spyder*.
- Plus, it can be set up on a separate server, to give to the user the possibility to work immediately.



Visualization

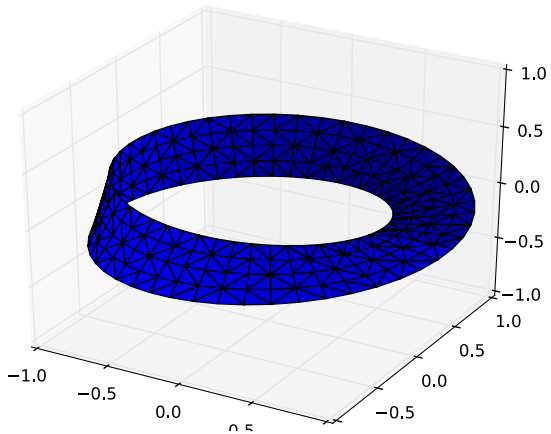
Create your own visual style.

Let it be unique for yourself and yet identifiable for others.

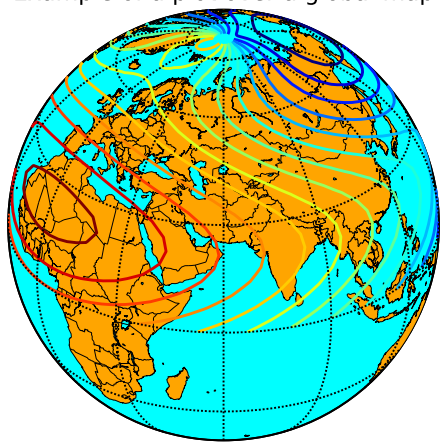
—Orson Welles

Matplotlib

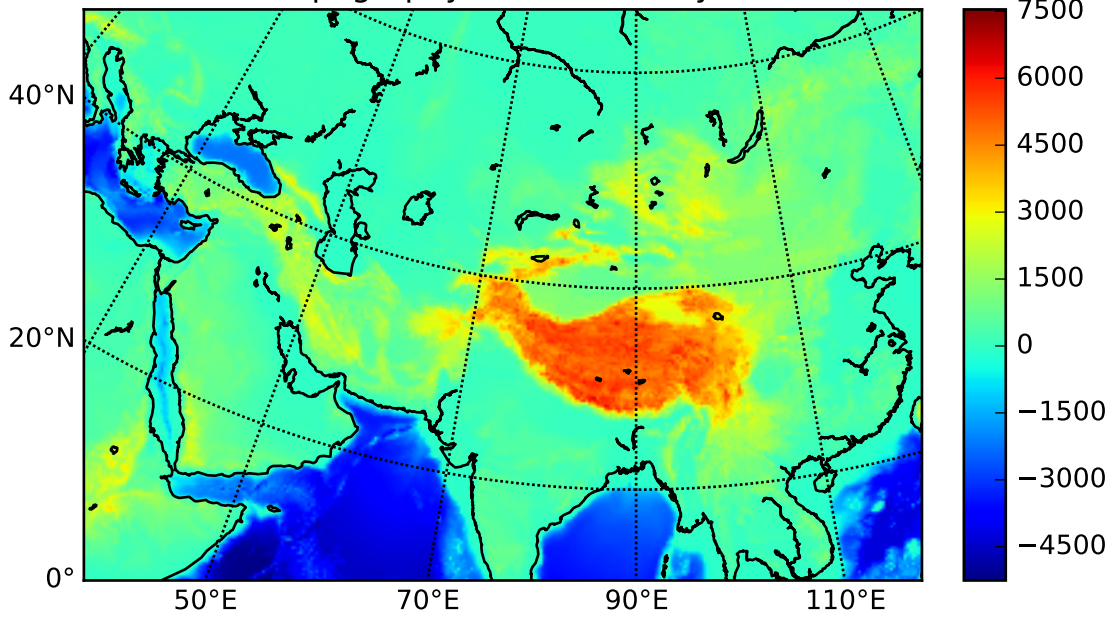
```
#Plotting a seismic event near Kilauea
import matplotlib.pyplot as plt
import numpy as np
from obspy.clients.fdsn import Client
from obspy import UTCDateTime
from obspy.signal import freqattributes

# Load the data from the web
Network = "HV"; Station = "WRM"; Location
t1 = UTCDateTime("2016-08-24T22:17:50.000 ,
client = Client("IRIS")
st = client.get_waveforms(Network, Station, Location, Chann

# There is only one trace in the Stream object, let's work
tr = st[0]; df=tr.stats.sampling_rate; dD=tr.stats.delta

# Filtering a copy of the original Trace
fMin=1.5; fMax=2.5
tr_filt = tr.copy()
tr_filt.filter('bandpass', freqmin=fMin, freqmax=fMax, corn
  ↪ zerophase=True)
tr_spec = freqattributes.spectrum(tr_filt.data,df,18001)

# Plottnig raw and filtered data
t = np.arange(0, tr.stats.npts / tr.stats.sampling_rate, tr
plt.subplot(211); plt.plot(t, tr.data, 'k'); plt.ylabel('Ra
plt.subplot(212); plt.plot(t, tr_filt.data, 'k')
plt.ylabel('Bandpassed Data'); plt.xlabel('Seconds')

plt.show()
```



Example of a plot over a global map

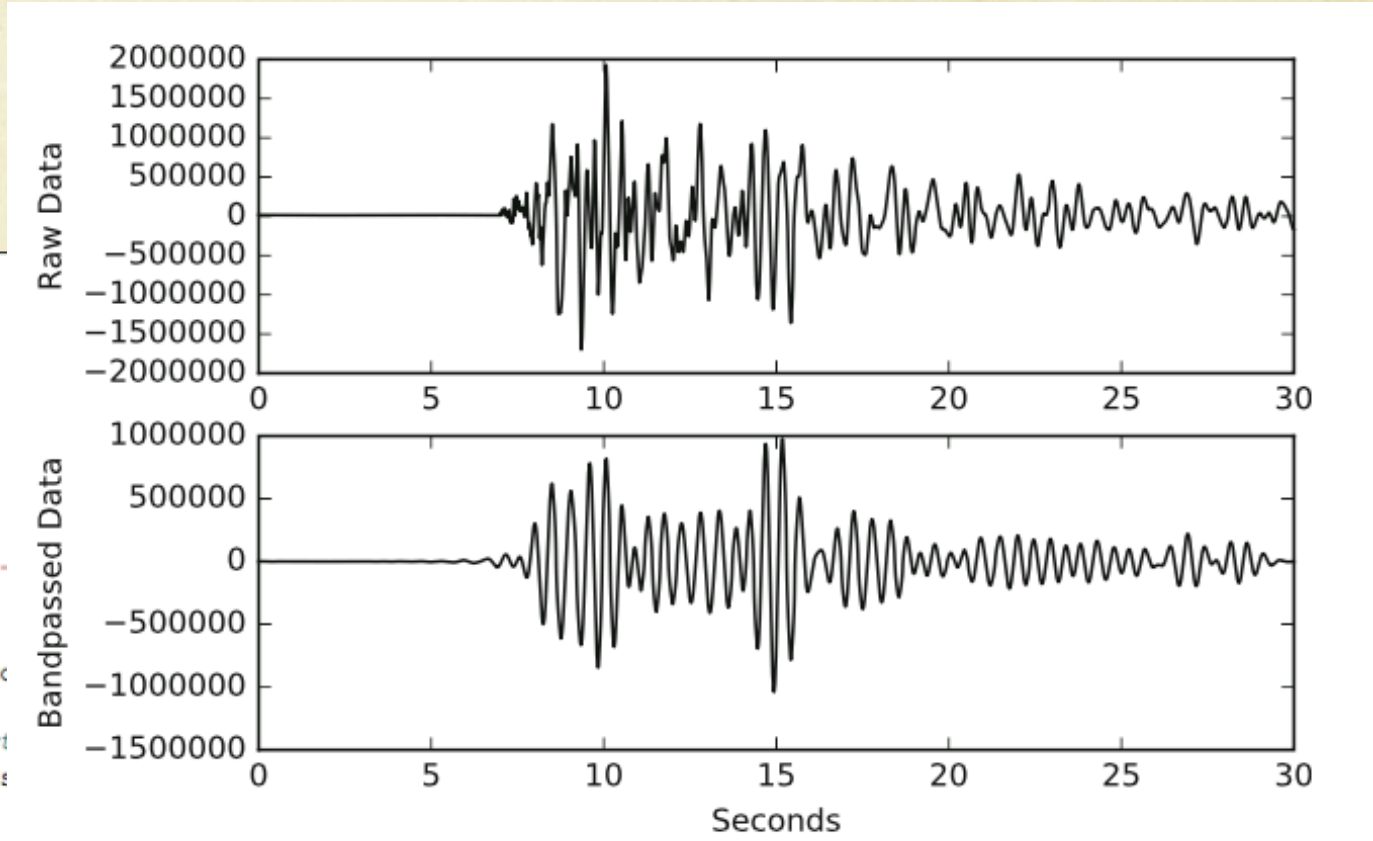


Topography above Himalaya

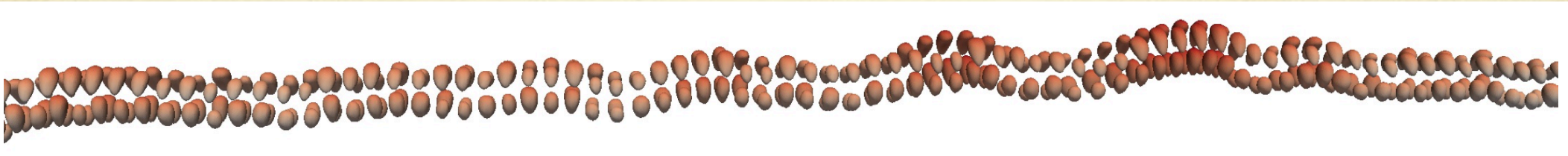


Matplotlib + ObsPy

```
#Plotting a seismic event near Kilauea
import matplotlib.pyplot as plt
import numpy as np
from obspy.clients.fdsn import Client
from obspy import UTCDateTime
from obspy.signal import freqattributes

# Load the data from the web
Network = "HV"; Station = "WRM"; Location = "-
t1 = UTCDateTime("2016-08-24T22:17:50.000")
client = Client("IRIS")
st = client.get_waveforms(Network, Station, Lo

# There is only one trace in the Stream object
tr = st[0]; df=tr.stats.sampling_rate; dD=tr.s

# Filtering a copy of the original Trace
fMin=1.5; fMax=2.5
tr_filt = tr.copy()
tr_filt.filter('bandpass', freqmin=fMin, freqmax=fMax, corners=3,
  ↪ zerophase=True)
tr_spec = freqattributes.spectrum(tr_filt.data,df,18001)

# Plottnig raw and filtered data
t = np.arange(0, tr.stats.npts / tr.stats.sampling_rate, tr.stats.delta)
plt.subplot(211); plt.plot(t, tr.data, 'k'); plt.ylabel('Raw Data')
plt.subplot(212); plt.plot(t, tr_filt.data, 'k')
plt.ylabel('Bandpassed Data'); plt.xlabel('Seconds')

plt.show()
```



Fast Python



"Give me 6 hours to chop a tree, I will spend the first 4 sharpening my axe."

– Abe Lincoln

How to make Numerical Python fast?

1. Vectorization of most operations. Highly optimized if NO LOOPS.

```
In [27]: %timeit c=addArray(a,b) #standard python
1 loops, best of 3: 639 ms per loop
In [28]: %timeit c=a+b #NumPy arrays broadcasting
100 loops, best of 3: 3.74 ms per loop
```

2. Cython (=C in Python) implementation of difficult routines.

```
cimport numpy as np
def setNegativeValuesToZero(int n, int m, np.ndarray[double, ndim=2] a):
        cdef int i, j
        for i in range(n):
                for j in range(m):
                        if a[i,j]<0:
                                a[i,j]=0.
```

3. Going parallel with the extension libraries (mpi4py, pyCuda, petsc4py).

```
from mpi4py import MPI
if rank == 0:
        data = np.arange(10000, dtype=np.float64)
        comm.Send(data, dest=1, tag=13)
elif rank == 1:
        data = np.empty(10000, dtype=np.float64)
        comm.Recv(data, source=0, tag=13)
```

Fun exercise with NumPy!

1. Ask to your students to create a random series of 100 integers (0 to 9) with the random generator, and one "manually" by guessing 100 numbers "randomly" chosen in the best possible way. There are several ways to find out which series has been generated by the computer, and the students have always fun in finding out how to distinguish one from the other.

```
import numpy as np
import matplotlib.pyplot as plt
a=np.array([7,3,2,5,4,9,5,2,0,3,1,2,3,4,8,7,6,8,5,9,4,6,7,3,1,3,8,0,7,3,2,5,6,8,9,7,3,
2,7,9,0,8,7,5,1,2,4,3,6,7,5,9,8,6,3,2,6,0,9,5,3,7,9,5,3,2,6,8,5,3,8,7,0,5,3,2,6,9,0,6,
4,1,1,2,3,6,9,5,6,3,5,8,9,0,7,4,3,3,8,7])
b=np.random.randint(0,10,100)
```

Mean? Deviatoric? Else? It takes some steps to find it out...

Tutorials are available online

If you feel just very weak in programming, in general, and you have never programmed before, these are very simple and introductory pages:

1. https://docs.python.org/3/tutorial/introduction.html
2. https://docs.python.org/3/tutorial/controlflow.html
3. https://docs.python.org/3/tutorial/datastructures.html

If you don't feel confident in using Matplotlib, use the following simple tutorial:

https://www.python-course.eu/matplotlib.php

and do not forget to look always at the gallery page, and look for interesting examples from which to learn:

https://matplotlib.org/gallery.html

It is extremely important that you become proficient with Numerical Python. If you do not feel confident in using arrays, there are some great tutorials online. For example the following pages could be a good starting point:

1. https://www.python-course.eu/numpy_create_arrays.php
2. https://www.python-course.eu/numpy_numerical_operations_on_numpy_arrays.php
3. https://www.python-course.eu/python_numpy_probability.php
4. https://docs.scipy.org/doc/numpy-dev/user/quickstart.html
5. https://docs.scipy.org/doc/numpy-dev/user/basics.creation.html

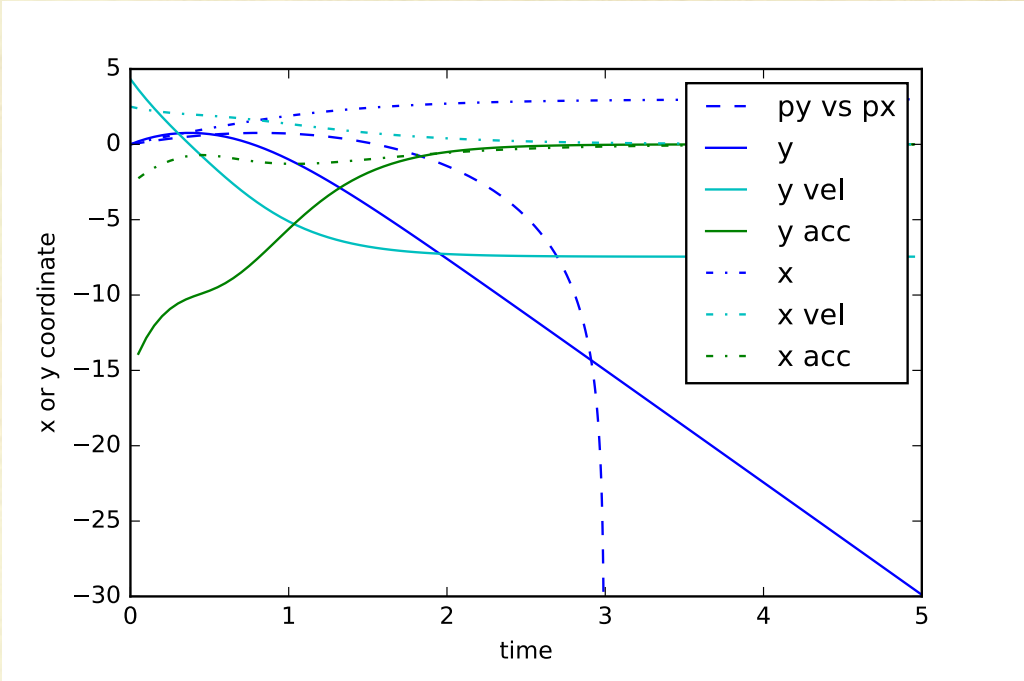
“Physics I” style examples

“Acceleration is finite, I think according to some laws of physics.”

— Terry Riley

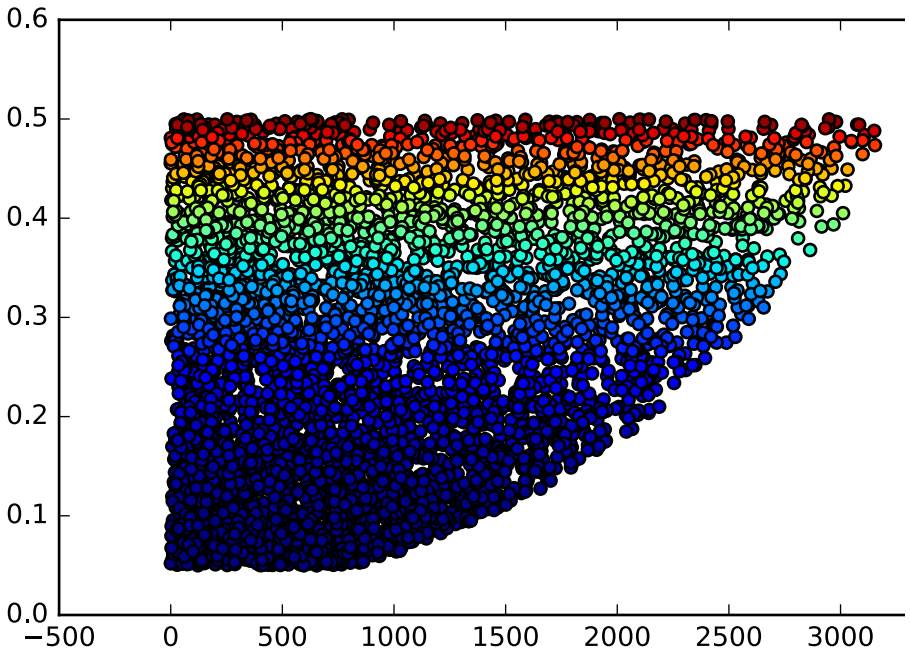
Learning ballistic trajectories first

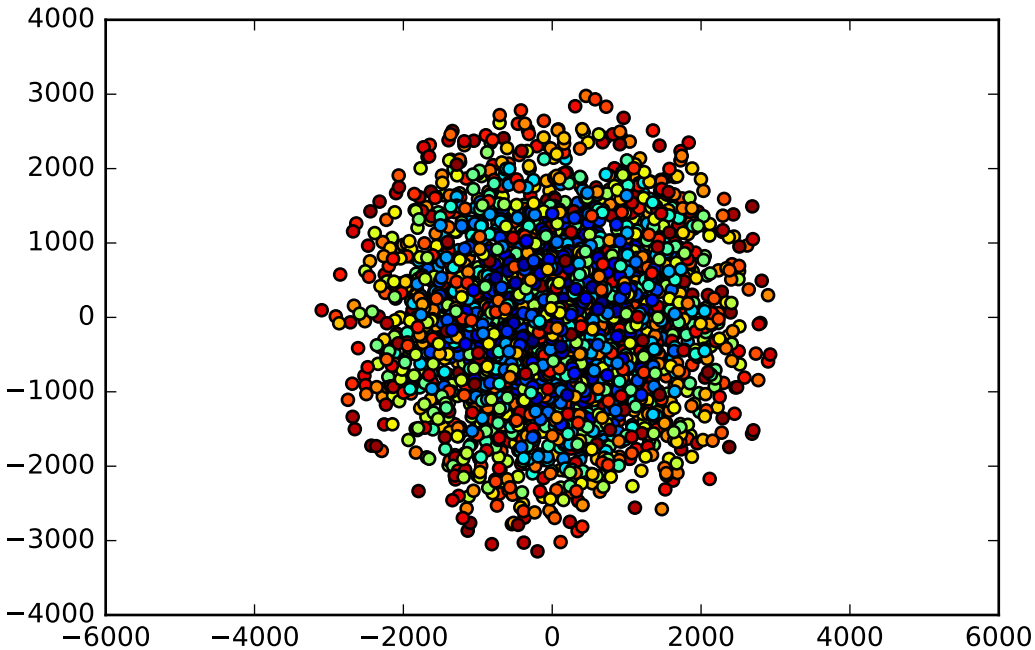
```
tmax = 5.0; tmin = 0.0
intervals = 100; dt = (tmax-tmin) / intervals
nt = intervals + 1; time = np.arange(nt) * dt
aGx = np.ones(nt-2) * gx; aGy = np.ones(nt-2) * gy
aDx = np.zeros(nt-2); aDy = np.zeros(nt-2)
vx = np.zeros(nt-1); vy = np.zeros(nt-1)
vx[0]=vel*np.cos(theta); vy[0]=vel*np.sin(theta)

for it in np.arange(nt-2):
        vMag2=vx[it]**2+vy[it]**2
        accDrag=0.5*Const*densityAir*Area*vMag2/mass
        aDx[it]=-accDrag*vx[it]/vMag2**0.5
        aDy[it]=-accDrag*vy[it]/vMag2**0.5

        vx[it+1] = vx[it]+(aGx[it]+aDx[it])*dt
        vy[it+1] = vy[it]+(aGy[it]+aDy[it])*dt

        pxInc = 0.5*(vx[:-1]+vx[1:])*dt
        pyInc = 0.5*(vy[:-1]+vy[1:])*dt
```



Then use it for a Monte Carlo

```
launches = 5000
landing = np.zeros(launches)
radii = np.zeros(launches)
mass = np.zeros(launches)
for thisTrajectory in np.arange(launches):
    radius=random.uniform(0.05, 0.5)
    density=2500.
    theta=random.uniform(0.,np.pi/2)
    vel=random.uniform(10.,200.)
    py0 = 1300.
    (px,py)=SingleLaunch(py0,radius, density, theta, vel)
    #plt.plot(px,py,label=str(thisTrajectory))
    landing[thisTrajectory]=px[py<0][0]
    radii[thisTrajectory]=radius
    mass[thisTrajectory]=4.0/3.0*np.pi*radius**3*density
```





Continuum Mechanics

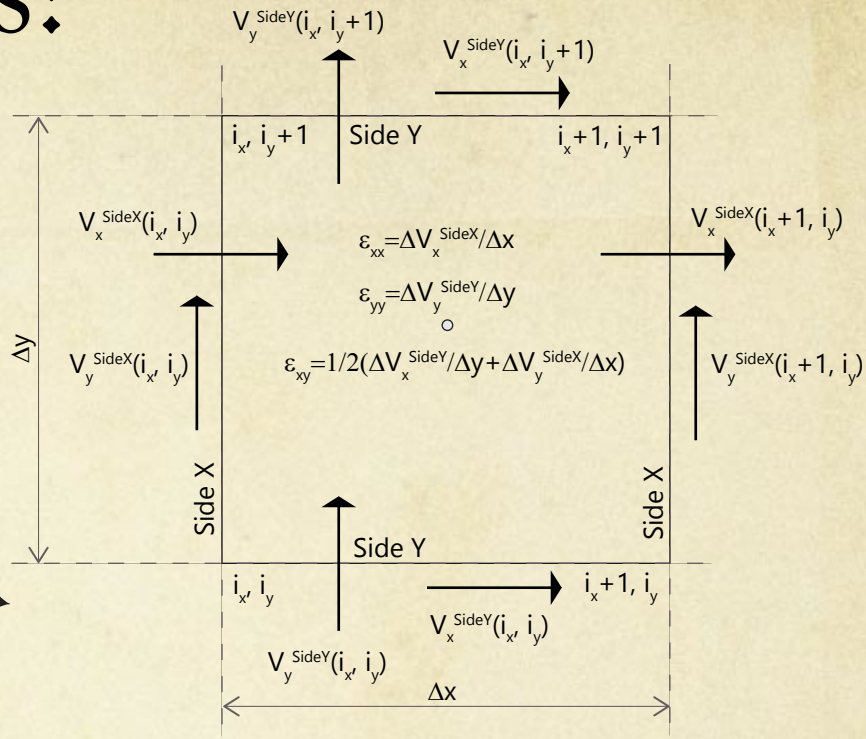
Create your own visual style.

Let it be unique for yourself and yet identifiable for others.

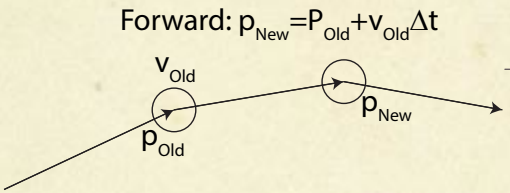
—Orson Welles

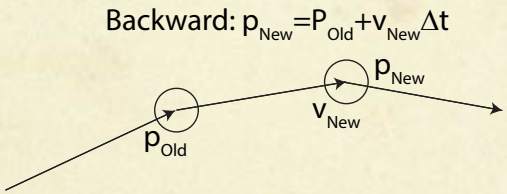
Continuum mechanics: particle in cell

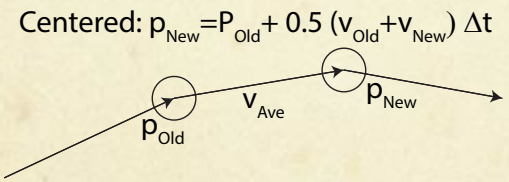
1. PDE's solution in a lattice



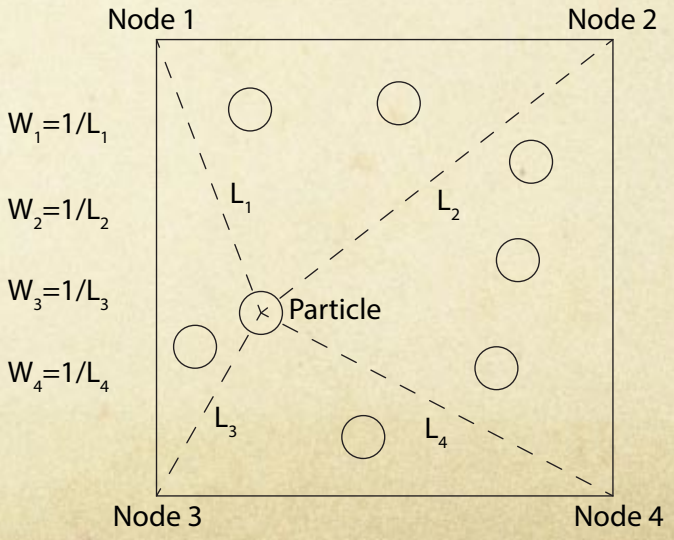
2. Particles advection







3. Projection of fields to and from the lattice



$$F_{Particle} = (F_{N1} * W_1 + F_{N2} * W_2 + F_{N3} * W_3 + F_{N4} * W_4)/(W_1+W_2+W_3+W_4)$$

Example: particle in cell

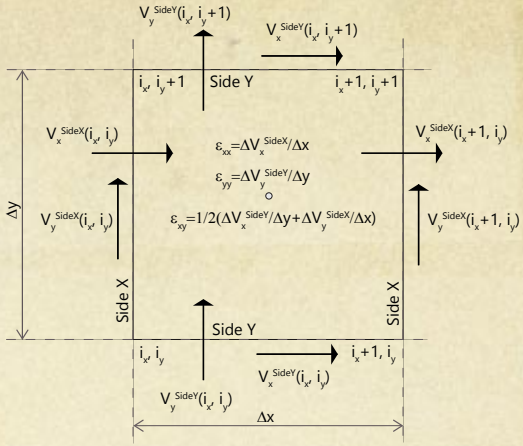
Cell ←→ Particle projections using NumPy.

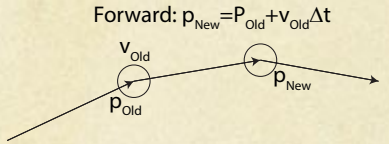
```
def projectLatticeToParticles(w1,w2,w3,w4,trIX,trIY,f,ft):
        ft[:]=(w1[:]*f[trIX[:],
                  trIY[:]]+w2[:]*f[trIX[:]+1,
                  trIY[:]]+w3[:]*f[trIX[:],
                  trIY[:]+1]+w4[:]*f[trIX[:]+1,
                  trIY[:]+1])/(w1[:]+w2[:]+w3[:]+w4[:])
        return (ft)
```

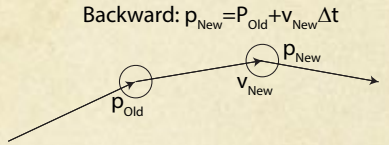
One line of code. Vectorized. Extremely fast.

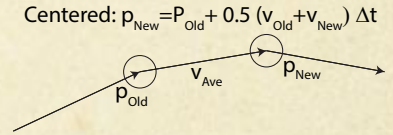
```
def projectParticlesToLattice(w1,w2,w3,w4,trIX,trIY,f,ft,tw):
        f[:,:]=0.0
        tw[:,:]=0.0
        f[trIX[:],trIY[:]]+=ft[:]*w1[:]
        f[trIX[:]+1,trIY[:]]+=ft[:]*w2[:]
        f[trIX[:],trIY[:]+1]+=ft[:]*w3[:]
        f[trIX[:]+1,trIY[:]+1]+=ft[:]*w4[:]
        tw[trIX[:],trIY[:]]+=w1[:]
        tw[trIX[:]+1,trIY[:]]+=w2[:]
        tw[trIX[:],trIY[:]+1]+=w3[:]
        tw[trIX[:]+1,trIY[:]+1]+=w4[:]
        f[:,:]/=tw[:,:]
        return (f)
```

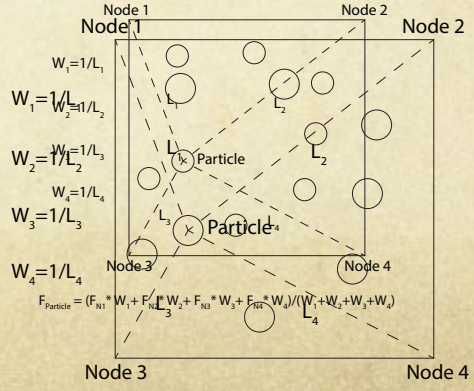
Compact. Easy to Understand and modify.
Minimum memory requirements





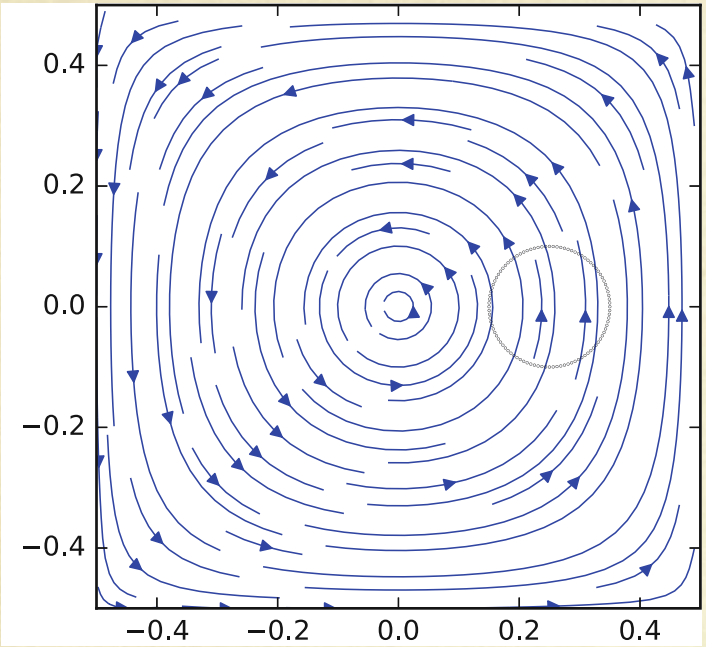


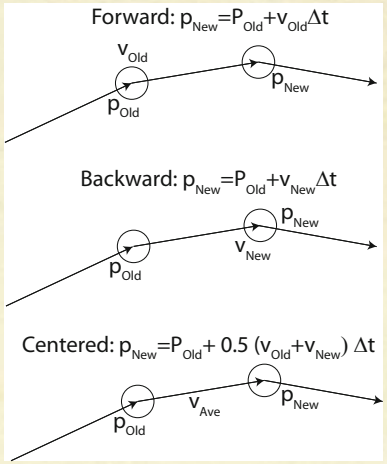


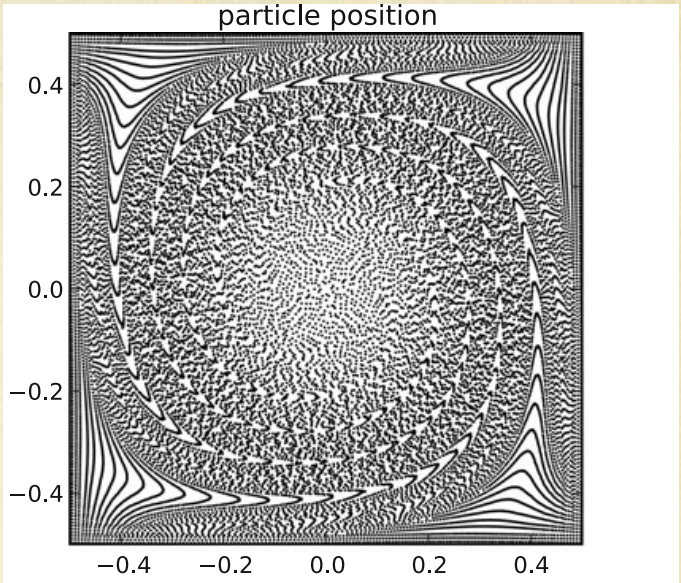


$F_{Particle} = (F_{N1} * W_1 + F_{N2} * W_2 + F_{N3} * W_3 + F_{N4} * W_4)/(W_1+W_2+W_3+W_4)$

Particle in cell of thinning flow

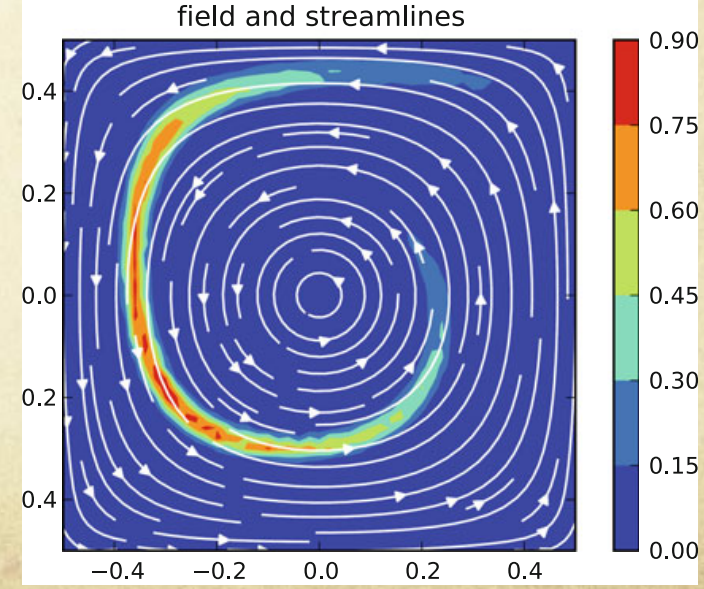






$$vx = -\sin^2(\pi x)\sin(\pi y)\cos(\pi y)$$

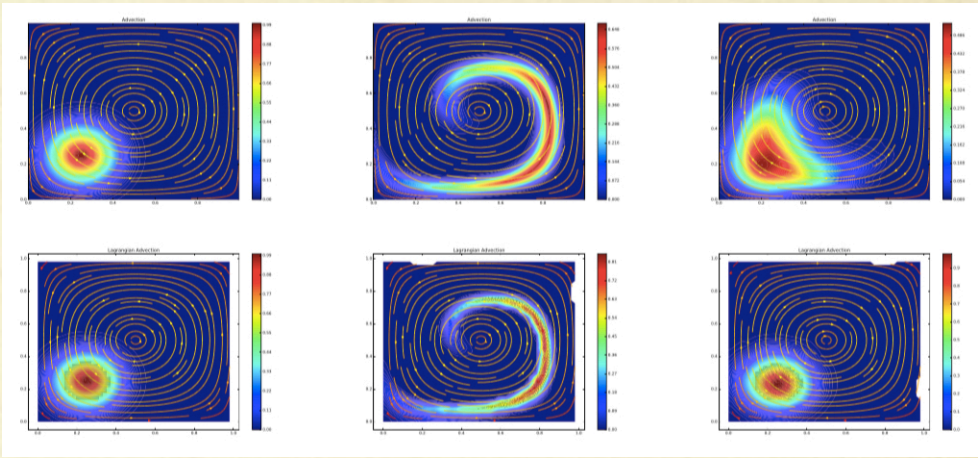
$$vy = \sin^2(\pi y)\sin(\pi x)\cos(\pi x)$$



Upwind vs particle in cell

$$\frac{dF}{dt} = \frac{\partial F}{\partial t} + \frac{\partial x_i}{\partial t}\frac{\partial F}{\partial x_i} = \frac{\partial F}{\partial t} + v_i \frac{\partial F}{\partial x_i}$$

```
def vectorizedUpwind(F,vx,vy,dx,dy,dt):
    (nx,ny) = F.shape
    dFplus = np.zeros((nx,ny),float)
    dFminus = np.zeros((nx,ny),float)
    dF_dt = np.zeros((nx,ny),float)

    ind = vx>0.0  # flag array for x upwind
    ind[0,:] = True; ind[nx-1,:]=False

    dFplus[1:-1,:] = (F[:-2,:]*vx[1:-1,:] - F[1:-1,:]*vx[2:,:])/dx
    dFminus[1:-1,:] = (F[1:-1,:]*vx[1:-1,:] - F[2:,:]*vx[2:,:])/dx
    dF_dt = (ind * dFplus + ~ind * dFminus)

    ind = vy>0.0  # flag array for y upwind
    ind[:,0] = True; ind[:,ny-1]=False

    dFplus[:,1:-1] = (F[:,:-2]*vy[:,1:-1] - F[:,1:-1]*vy[:,2:])/dy
    dFminus[:,1:-1] = (F[:,1:-1]*vy[:,1:-1] - F[:,2:]*vy[:,2:])/dy
```



Operators are (abstract) toys

"Operator! Give me the number for 911!"

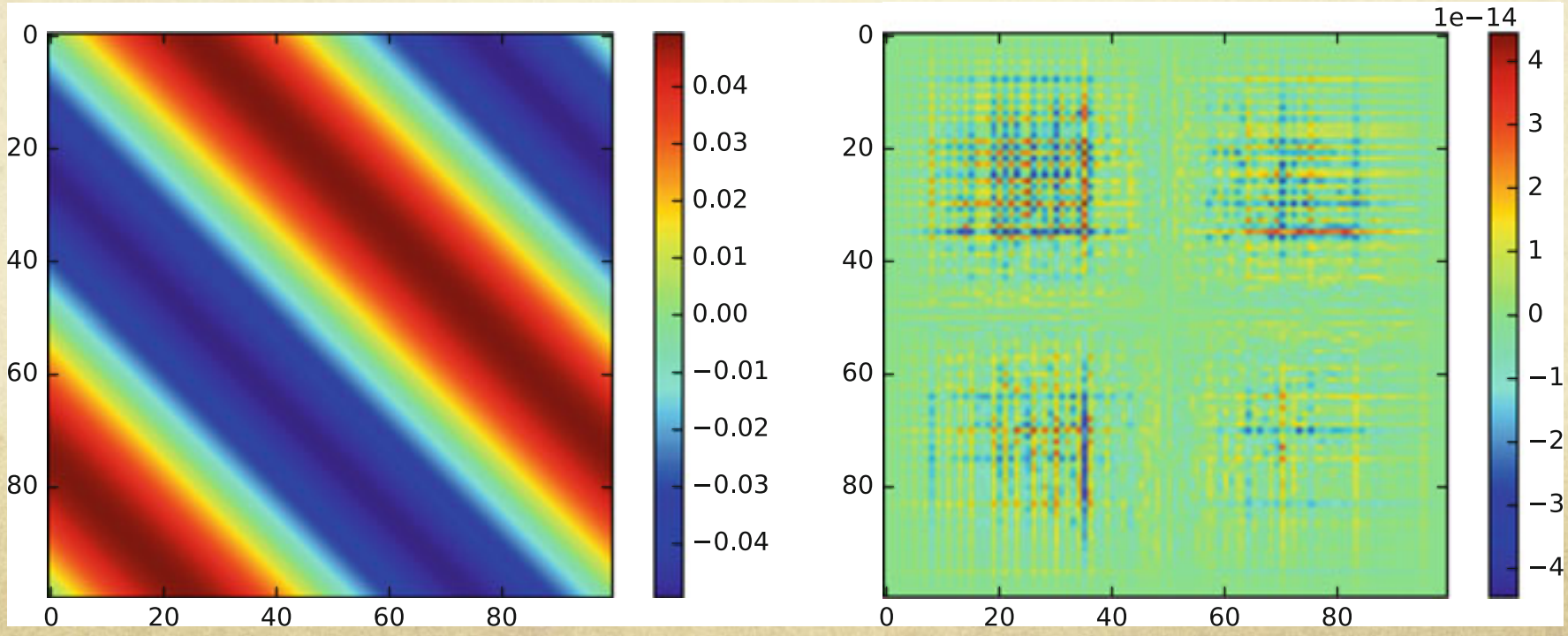
–Homer Simpson

Strain rates from combining operators

```
def buildSideOperatorX(nxp,nyp):
    nyc=nyp-1
    block=np.zeros((nyc,nyp),float);
    block[0:nyc,0:nyc]+=0.5*np.diag(np.ones(nyc),0)
    block[0:nyc,1:nyp]+=0.5*np.diag(np.ones(nyc),0)
    return (np.kron(np.identity(nxp),block))
```

```
def buildDyOperator(nxc,nyp,dy):
    nyc=nyp-1
    block=np.zeros((nyc,nyp),float);
    block[0:nyc,0:nyc]-=1.0/dy*np.diag(np.ones(nyc),0)
    block[0:nyc,1:nyp]+=1.0/dy*np.diag(np.ones(nyc),0)
    return (np.kron(np.identity(nxc),block))
```

```
# calculate the xx strain rate
DxOp = buildDxOperator(nxp,nyc,dx)
DvxDxL = np.dot(DxOp,vxSxL)
DvxDx = DvxDxL.reshape(nxc,nyc)
```



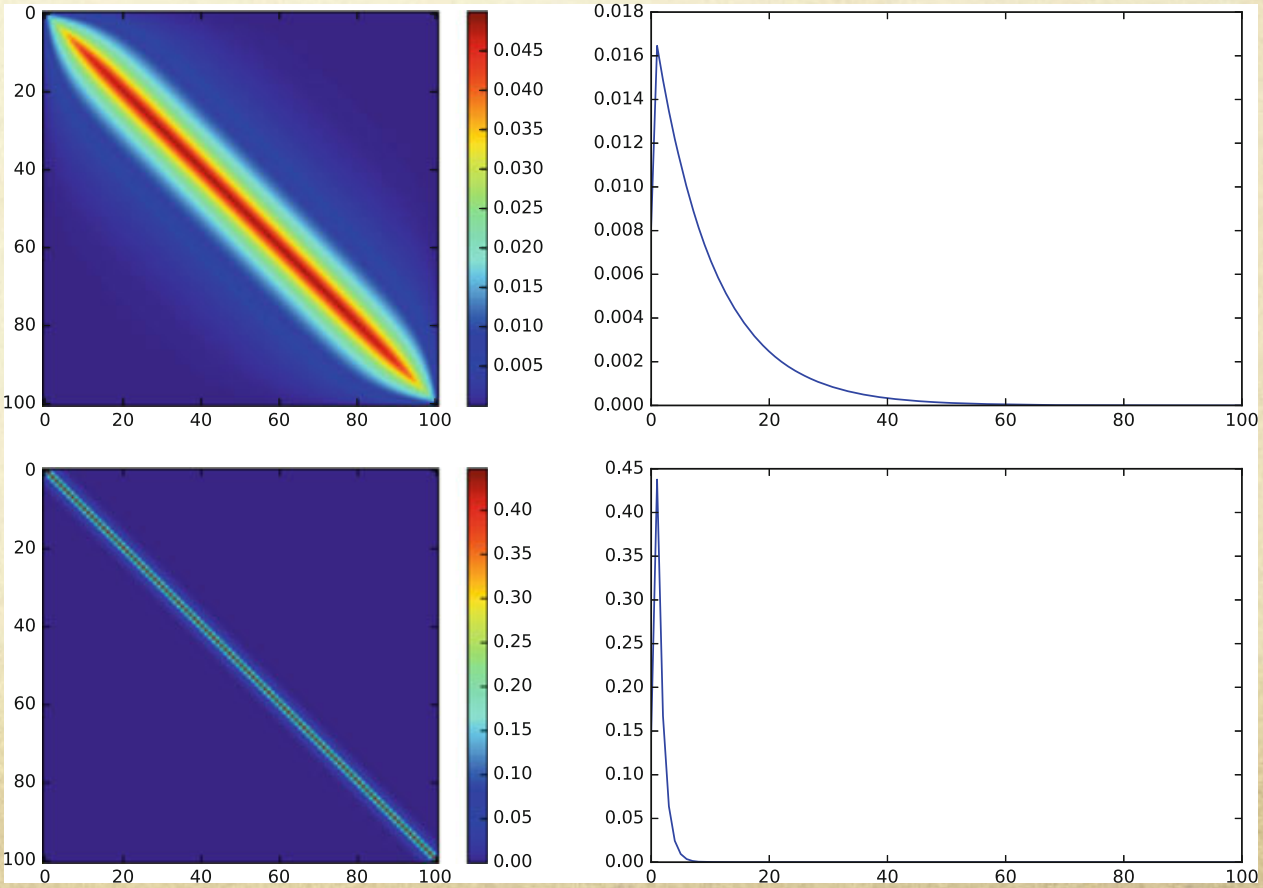
Sparse and Dense Operators

```
def buildSparseOperatorYaxis(nx,ny,k1,k2):
    ny1=ny-1
    firstDiag=np.ones(ny,float)*k1
    secondDiag=np.ones(ny,float)*k2
    offsets=np.array([0,1])
    block = sparse.dia_matrix(([firstDiag,secondDiag],offsets),
    ↪  shape=(ny1,ny)).tocsr()
    return (sparse.kron(sparse.eye(nx),block))
```

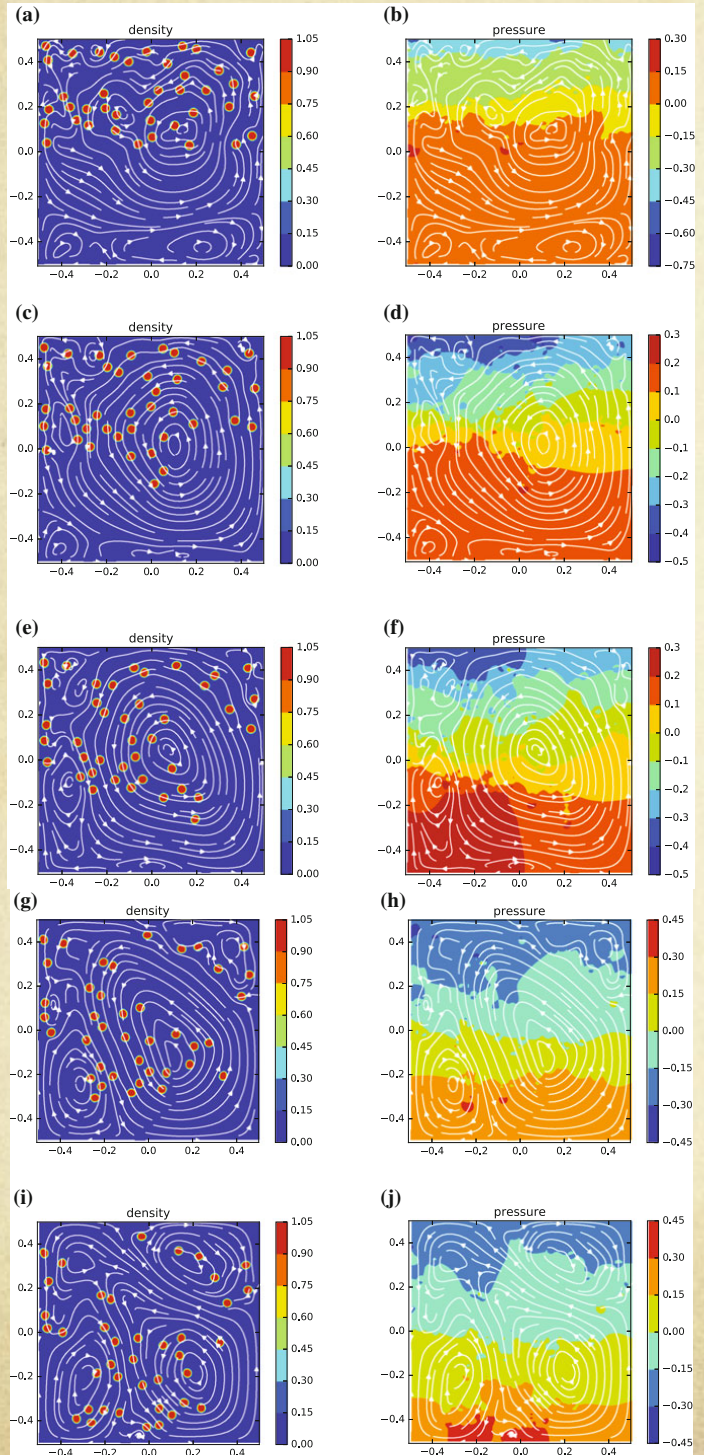
$$(I + A)T^{t+\Delta t} = T^t$$

$$T^{t+\Delta t} = (I+A)^{-1}T^t = BT^t$$

Left 2D representation of the matrix B = I + A–1 for r = 100.0 (top) and r = 1.0 (bottom). One observes that for an implicit algorithm with increasing r (and therefore being far beyond the limiting explicit case), the terms far from the diagonal also increase and the sparse structure is completely lost. Right central section of the B matrix where it is visible its decay from the diagonal, smoother with increasing r . For large r matrix sparsity for the inverse is not an advantage anymore



Non-linear



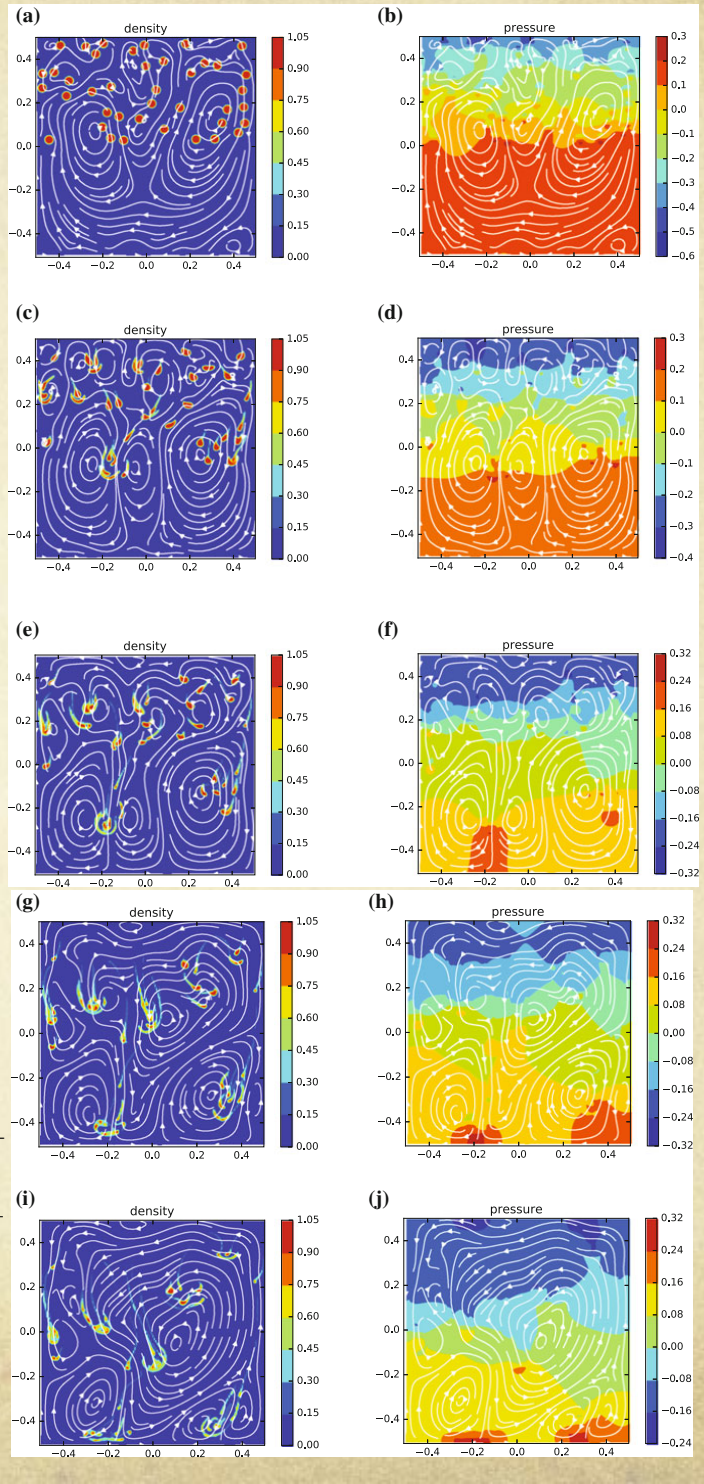
$$\begin{bmatrix} \frac{\partial}{\partial x}\left(2\eta\frac{\partial}{\partial x}\right)+\frac{\partial}{\partial y}\left(\eta\frac{\partial}{\partial y}\right) & \frac{\partial}{\partial y}\left(\eta\frac{\partial}{\partial x}\right) & -\frac{\partial}{\partial x} \\ \frac{\partial}{\partial x}\left(\eta\frac{\partial}{\partial y}\right) & \frac{\partial}{\partial x}\left(\eta\frac{\partial}{\partial x}\right)+\frac{\partial}{\partial y}\left(2\eta\frac{\partial}{\partial y}\right) & -\frac{\partial}{\partial y} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & 0 \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ p \end{bmatrix} = \begin{bmatrix} -\rho g_x \\ -\rho g_y \\ 0 \end{bmatrix}$$

(10)

$$\frac{\eta^{i+1,j+1}\left(v_y^{i+\frac{3}{2},j+1}-v_y^{i+\frac{1}{2},j+1}\right)-\eta^{i+1,j}\left(v_y^{i+\frac{3}{2},j}-v_y^{i+\frac{1}{2},j}\right)}{\Delta x\,\Delta y}$$

$$2\frac{\eta^{i+\frac{3}{2},j+\frac{1}{2}}\left(v_x^{i+2,j+\frac{1}{2}}-v_x^{i+1,j+\frac{1}{2}}\right)-\eta^{i+\frac{1}{2},j+\frac{1}{2}}\left(v_x^{i+1,j+\frac{1}{2}}-v_x^{i,j+\frac{1}{2}}\right)}{\Delta x^2}+$$

$$\frac{\eta^{i+1,j+1}\left(v_x^{i+1,j+\frac{3}{2}}-v_x^{i+1,j+\frac{1}{2}}\right)-\eta^{i+1,j}\left(v_x^{i+1,j+\frac{1}{2}}-v_x^{i+1,j-\frac{1}{2}}\right)}{\Delta y^2}$$

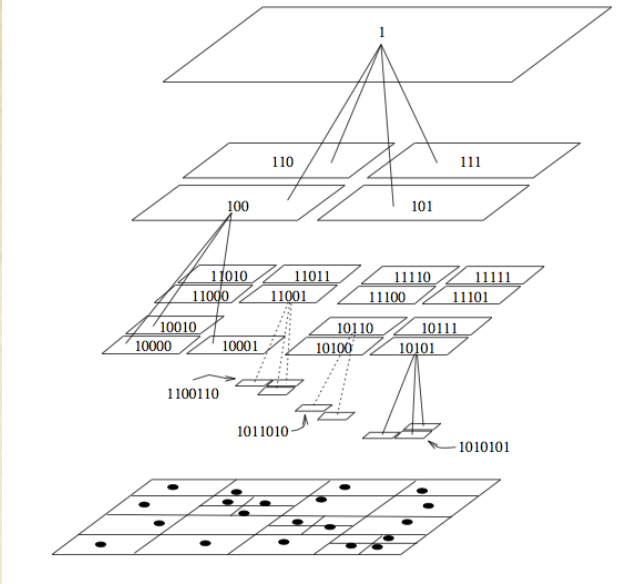


3D

"Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

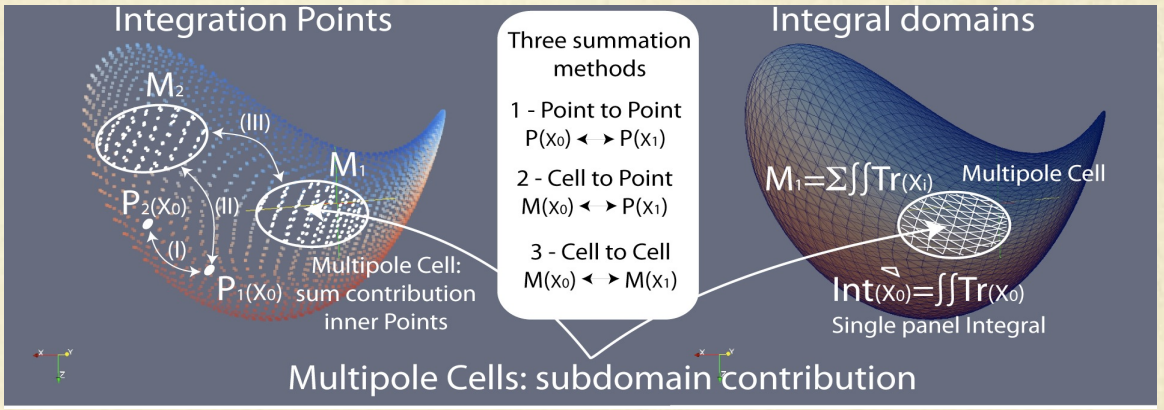
— Linus Torvalds

Immediate 3d modeling with NumPy



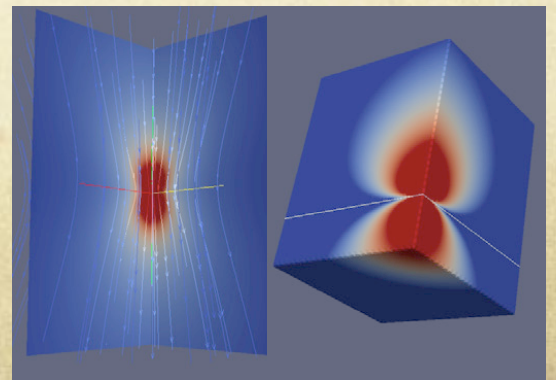
1. Tree representation

2. Fast Integration



3. Lagrangian motion

4. Cartesian representation



Immediate 3d modeling with NumPy

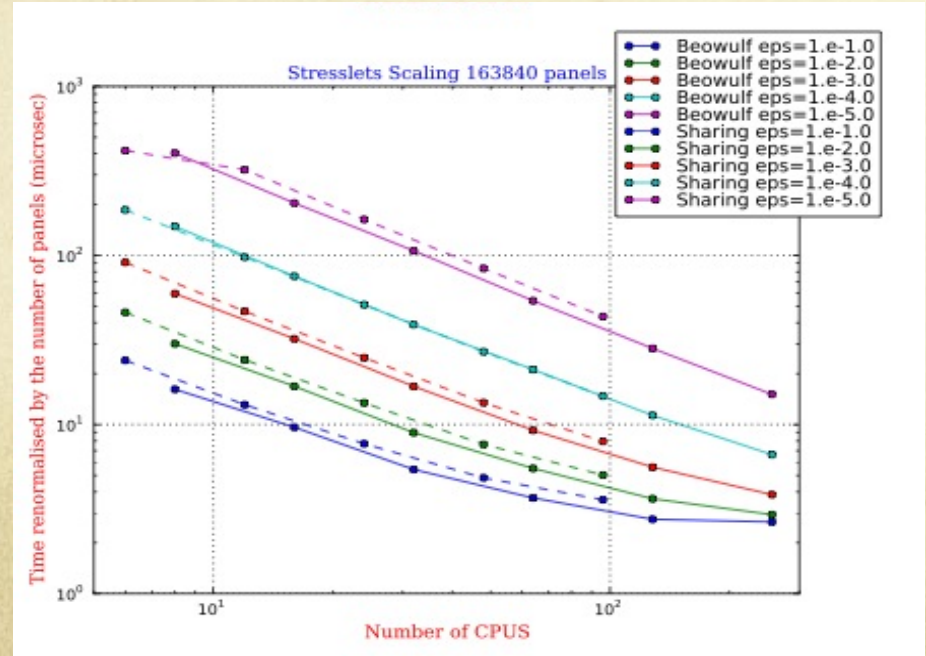
1. Tree representation:

```
from scipy import spatial
x, y, z = np.mgrid[0:5, 2:8, 3:7]
tree = spatial.KDTree(zip(x.ravel(), y.ravel(), z.ravel()))
```

2. Many-body calculations enable N-logN scaling.

3. Fast Integrals with NumPy

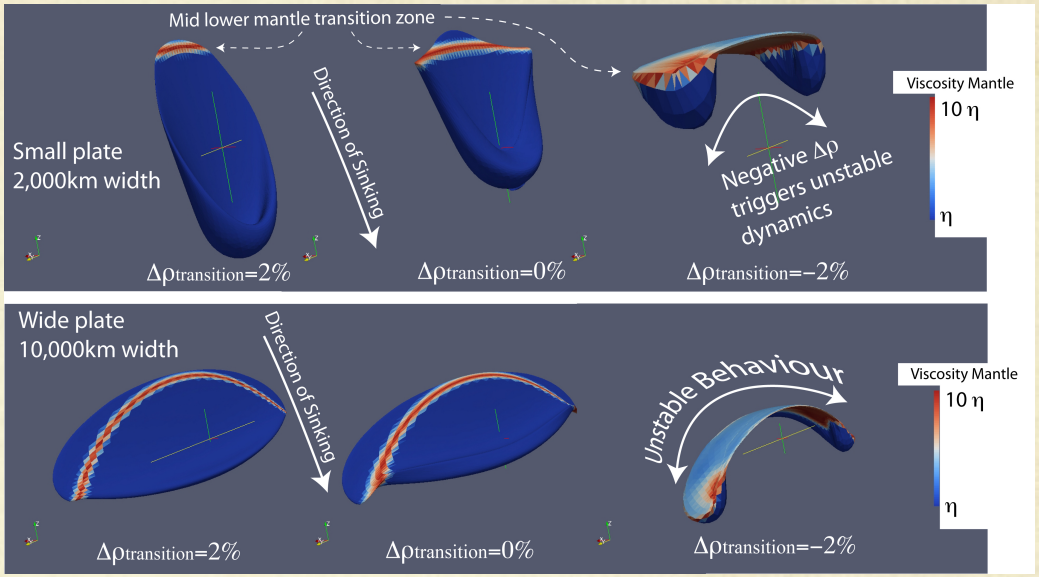
4. MPI Parallelization

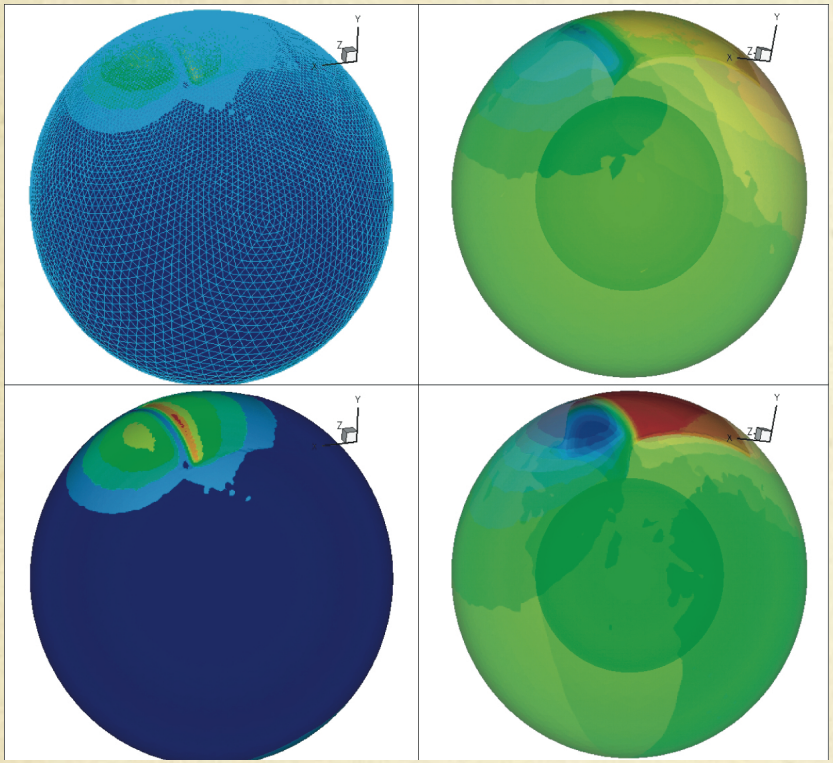



```
def integralOverTriangle(collocationPosition, element,
  ↪  gaussPoints,gaussWeights,xiCoord,etaCoord,
  ↪  coords,nodesOnElement, alpha,beta,gamma):

# Integrates the Green's function over a non-singular triangle
# GE[0:3,0:3] are the integrated component
# gaussPoints is the order of triangle quadrature

   GE=np.zeros((3,3),float)

   (gaussPosition,
   gaussNormalVector,
   gaussSurfaceMetrics,
   gaussDerivatives)=interpolate(
      coords[node1],coords[node2],coords[node3],
      coords[node4],coords[node5],coords[node6],
      alpha,beta,gamma,
      gaussXi,gaussEta,
      1)

   d = gaussPosition-collocationPosition
   dd = np.outer(d,d)
   i = np.identity(3)
   r = np.linalg.norm(d)
   velocityGreenFunction = i / r + dd / r**3

   prefactor=0.5*gaussSurfaceMetrics*gaussWeights[gaussPoint]
   GE += prefactor*velocityGreenFunction

   return GE
```

Applications in global geodynamics and multiphase flow

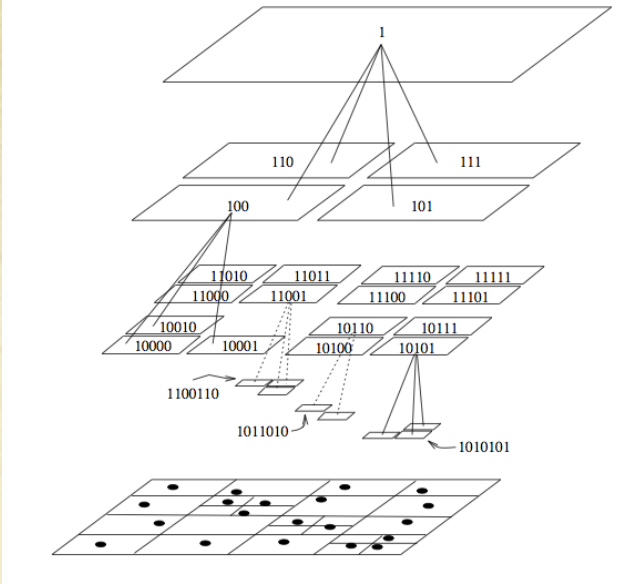
Morra et al., PEPI, 2010





Morra et al., 2012

Example II: 3D BEM + NumPy + KD Tree + PyMPI

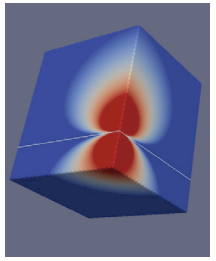


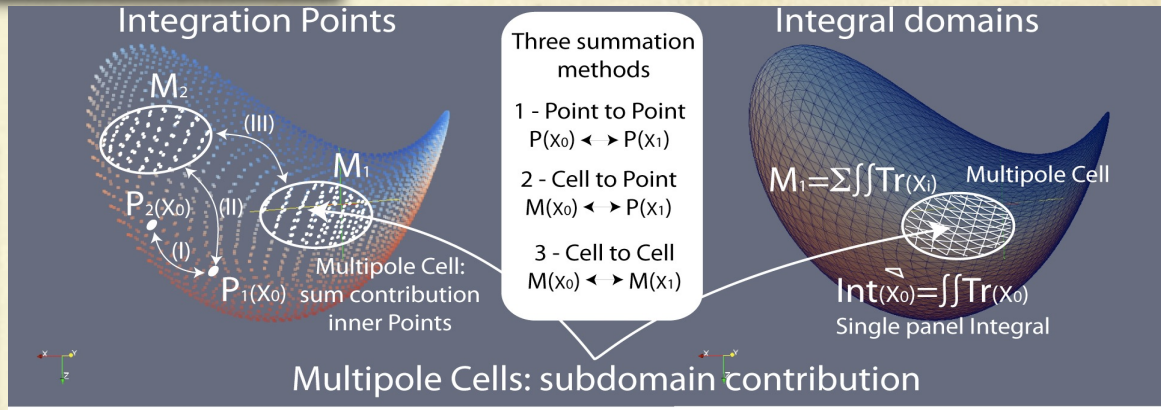
1. Tree representation (e.g KD Tree)

```
from scipy import spatial
x, y, z = np.mgrid[0:5, 2:8, 3:7]
tree = spatial.KDTree(zip(x.ravel(), y.ravel(), z.ravel()))
```

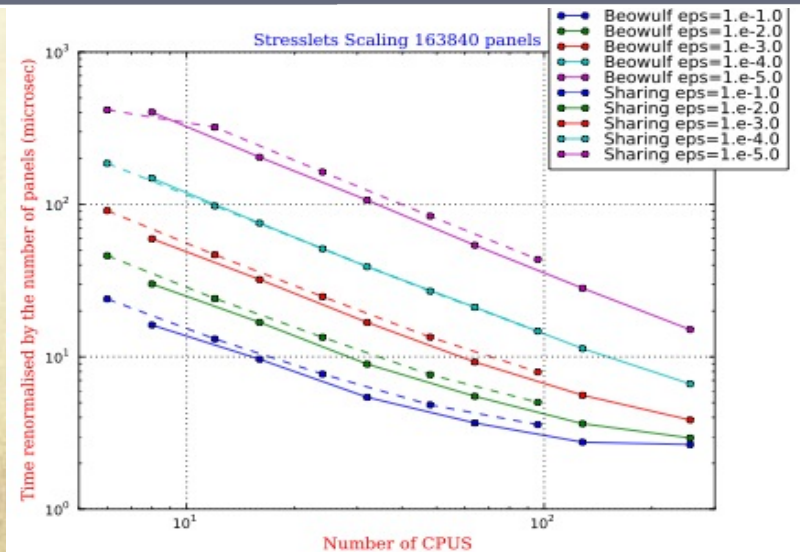
2. Fast Integration

```
(gaussPosition,
gaussNormalVector,
gaussSurfaceMetrics,
gaussDerivatives)=interpolate(
   coords[node1],coords[node2],coords[node3],
   coords[node4],coords[node5],coords[node6],
   alpha,beta,gamma,
   gaussXi,gaussEta,
   1)

d = gaussPosition-collocationPosition
dd = np.outer(d,d)
i = np.identity(3)
r = np.linalg.norm(d)
velocityGreenFunction = i / r + dd / r**3

prefactor=0.5*gaussSurfaceMetrics*gaussWeights[gaussPoint]
GE += prefactor*velocityGreenFunction

return GE
```

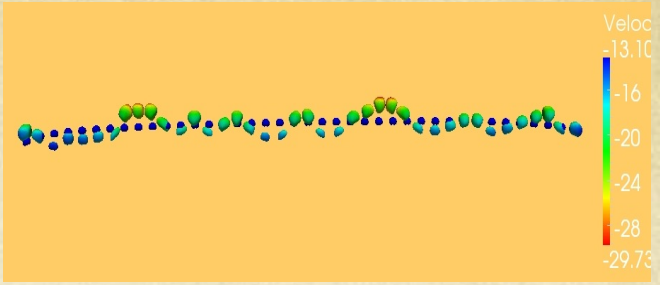




3. MPI Parallelization

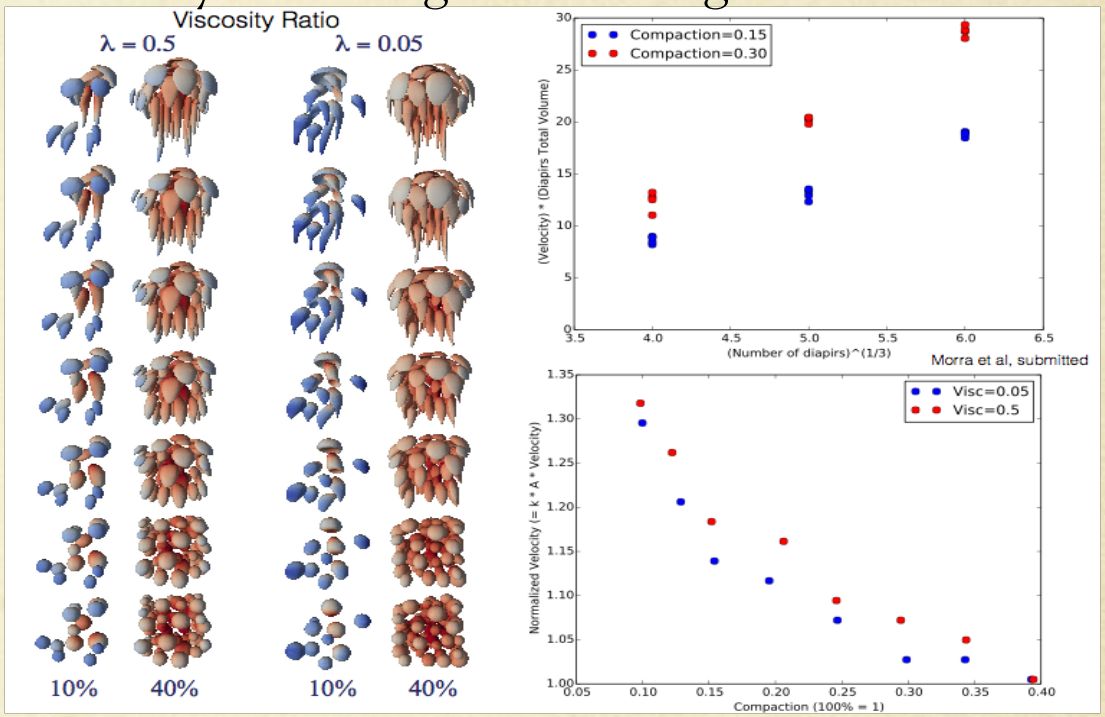


Example II: 3D BEM + NumPy + KD Tree + PyMPI

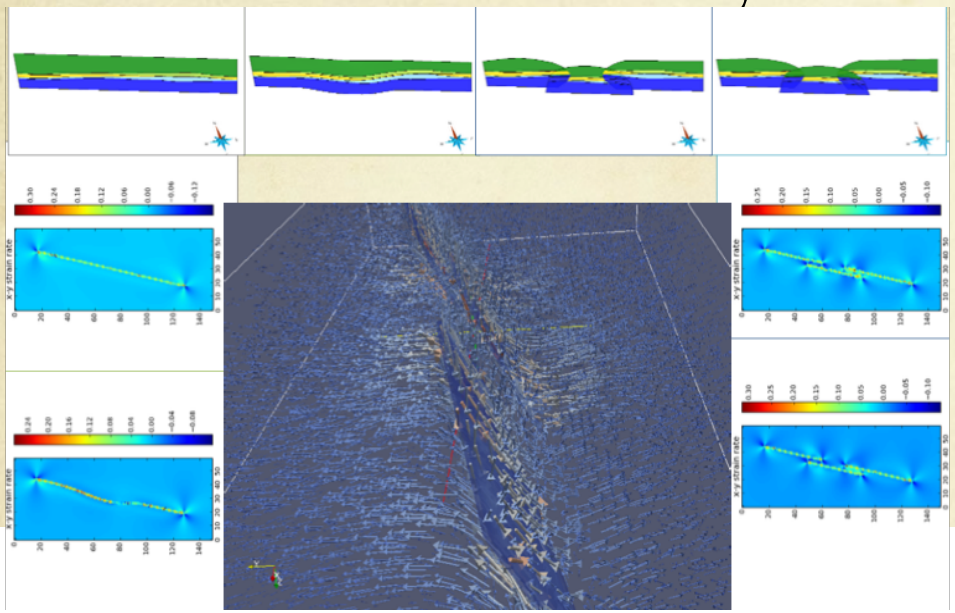


1. Tree representation (e.g KD Tree)
2. Fast Integration
3. MPI parallelization

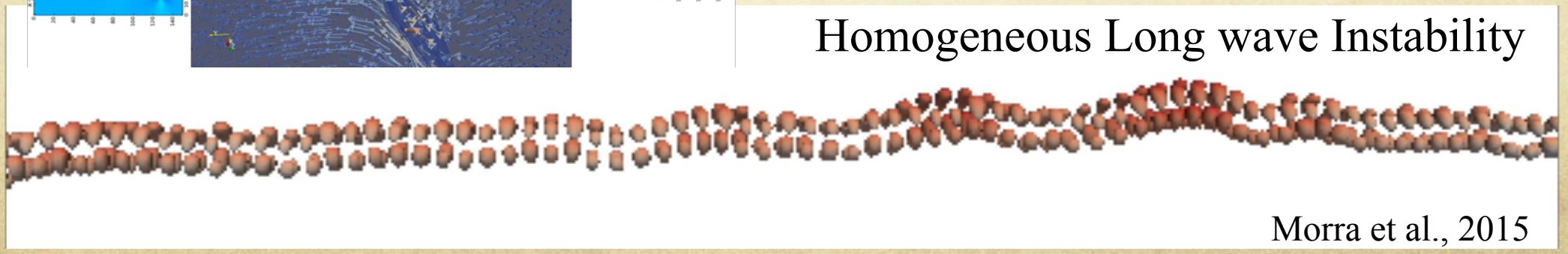
Bubbly flow of gas in a magmatic conduit



Crustal Dynamics



Homogeneous Long wave Instability



Morra et al., 2015

Conclusions and Perspectives

- Students and professionals have now more accessible tools to learn programming, which are simple and accessible new languages.
- Jupyter represents a real revolution in Applied Computer science pedagogy.
- By using Python, geodynamics codes in 2D as well as in 3D are compact, run fast, are parallelized in a straightforward way. But they need careful design from the start to avoid bottlenecks.
- Parallel Computing is easily accessible as well, by using modules such as mpi4py, PyCuda, PyPETSc
- Students learn very quickly using these tools, and acquire an immediate understanding of how computers work. Some students show resistance. They do not want to “jump” into the game.