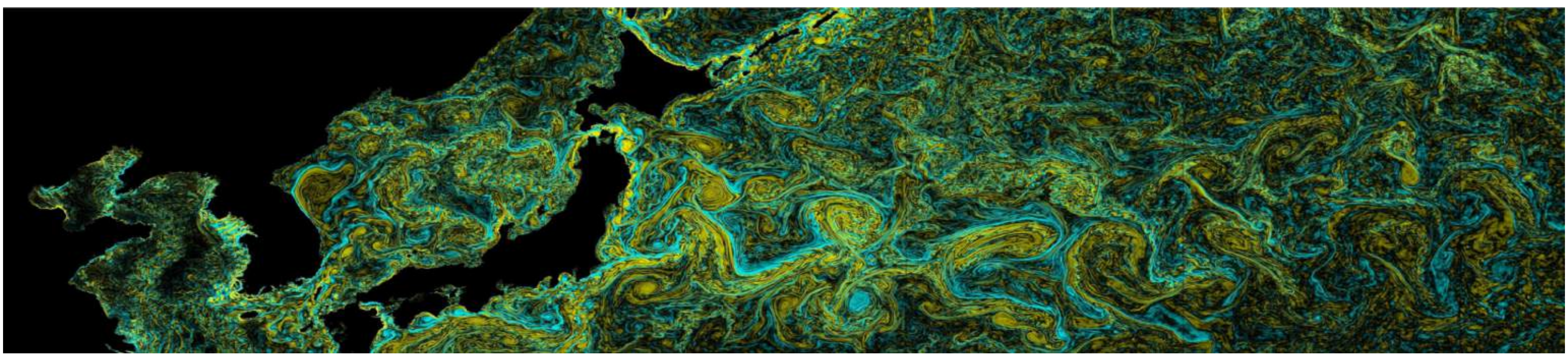
Computational lecture: TBD (TBD)

- Modeling, assimilation and observing
- AI/ML algorithm
- Compute hardware and software
- Collaboration and analysis

Compute is everywhere!



Some topics

- Julia
- GPUs etc….
- Data centers, data sharing and analysis
- AI/ML opportunities

Julia

Julia: A fresh approach to numerical computing

Jeff Bezanson Alan Edelman Stefan Karpinski Viral B. Shah

MIT and Julia Computing*

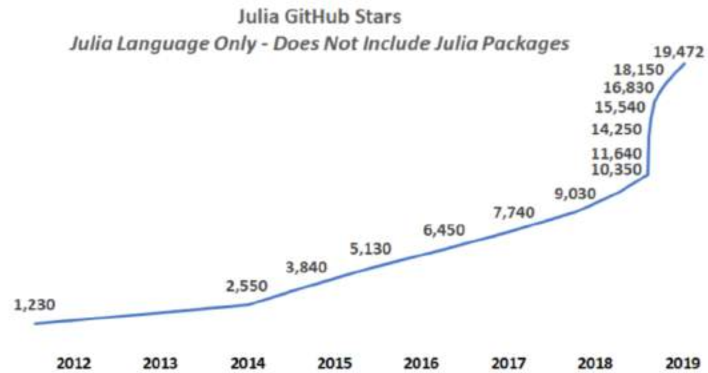
July 7, 2015

Figure 1: Gil Strang's favorite matrix is strang(n) = SymTridiagonal(2*ones(n),-ones(n-1)) Julia only stores the diagonal and off-diagonal. (Picture taken in Gil Strang's classroom.)

6 Conclusion and Acknowledgments

We built Julia to meet our needs for numerical computing, and it turns out that many others wanted exactly the same thing. At the time of writing, not a day goes by where we don't learn that someone else has picked up Julia at universities and companies around the world, in fields as diverse as engineering, mathematics, physical and social sciences, finance, biotech, and many others. More than just a language, Julia has become a place for programmers, physical scientists, social scientists, computational scientists, mathematicians, and others to pool their collective knowledge in the form of online discussions and in the form of code. Numerical computing is maturing and it is exciting to watch!

Julia would not have been possible without the enthusiasm and contributions of the Julia community[22]. We thank Michael La Croix for his beautiful Julia display macros. We are indebted at MIT to Jeremy Kepner, Chris Hill, Saman Amarasinghe, Charles Leiserson, Steven Johnson and Gil Strang for their collegial support which not only allowed for the possibility of an academic research project to update technical computing, but made it more fun too. The authors gratefully



	Total as of Jan 2018	Total as of Jan 2019	
Number of Julia Downloads Initiated via JuliaLang	1.8 million	3.2 million	+78%
Number of Julia Downloads Initiated via DockerHub	Not Available	4 million	Not Available
Julia Packages Available	1,688	2,462	+46%
Number of News Articles Mentioning Julia	93	253	+172%
Julia Discourse Threads + Stack Overflow Questions	8,620	16,363	+90%
GitHub Stars for Julia Language (Not Including Julia Packages)	9,626	19,472	+102%
Published Citations of Julia: A Fresh Approach to Numerical Computing (2017) and Julia: A Fast Dynamic Language for Technical Computing (2012)	613	1,048	+71%

*** Note: Julia can also call C, C++, Fortran, Python, R, Java and MPI libraries**

Julia programming

- Julia is an interactive language (like Matlab/Python).
- The language design allows for so-called "just-in-time" compilation
- ➔you can write loops in native Julia and they can run as fast as C/Fortran.
- People seem more excited about writing Julia code than Fortran
- More people can write code more quickly in Julia
- It remains unclear if more people can write rigorous and correct numerical code – challenge may be human factors not language!

Julia community

- Julia community is very active/large and skews toward technical/applied math-science minded

juliacon 2019

Optim

FFTW

ASR

- Zygote

Edit on GitHub

Checkpointing

Don't Unroll Adjoint: Differentiating SSA-Form Programs

Michael Innes

(Submitted on 18 Oct 2018 (v1), last revised 9 Mar 2019 (this version, v4))

A more advanced example is checkpointing, in which we save memory by re-computing the forward pass of a function during the backwards pass. To wit:

```
julia> checkpoint(f, x) = f(x)
checkpoint (generic function with 1 method)

julia> @adjoint checkpoint(f, x) = f(x), ȳ -> Zygote._forward(f, x)[2](ȳ)

julia> gradient(x -> checkpoint(sin, x), 1)
(0.5403023058681398,)
```

If a function has side effects we'll see that the forward pass happens twice, as expected.

```
julia> foo(x) = (println(x); sin(x))
foo (generic function with 1 method)

julia> gradient(x -> checkpoint(foo, x), 1)
1
1
(0.5403023058681398,)
```

GPUifyLoops.jl

GPUifyLoops tries to solve the problem of code-duplication that can occur when writing performant kernels that target multiple devices.

```
using GPUifyLoops

function kernel(A)
    @loop for i in (1:size(A,1);
                    threadIdx().x)
        A[i] = 2*A[i]
    end
    @synchronize
end
```

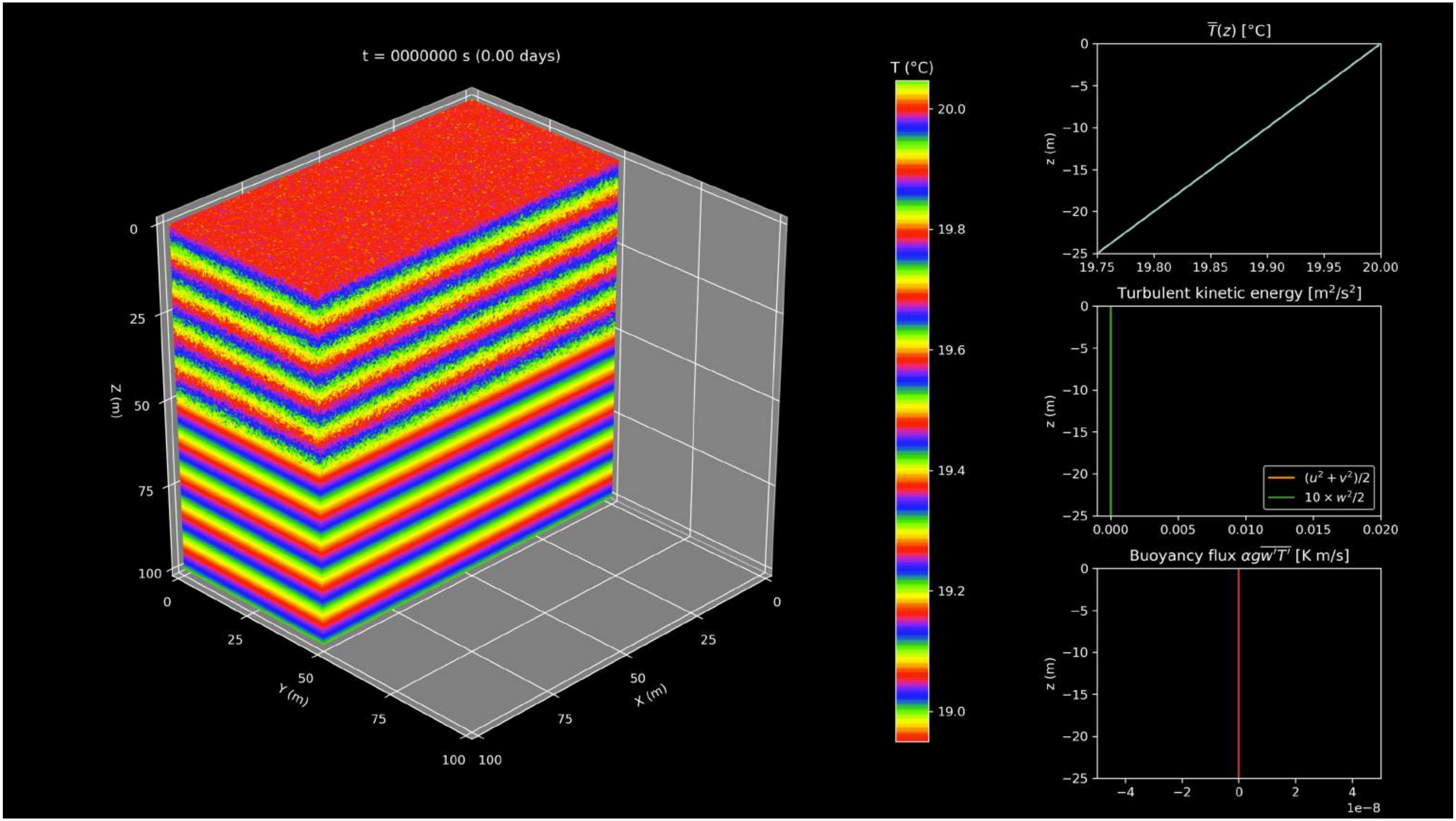
Oceananigans.jl (https://github.com/climate-machine/Oceananigans.jl)

- A fast non-hydrostatic ocean model in Julia that can be run in 2 or 3 dimensions on CPUs and GPUs. The plan is to develop it as a stand-alone large eddy simulation (LES) model which can be used as a source of training data for statistical learning algorithms and/or embedded within a global ocean model as a super-parameterization of small-scale processes, as in Campin et al., 2011.

```
julia>]
(v1.1) pkg> add Oceananigans
```

```
using Oceananigans
Nx, Ny, Nz = 100, 100, 50        # Number of gri
Lx, Ly, Lz = 2000, 2000, 1000    # Domain size (
Nt, Δt = 10, 60                  # Number of tim

model = Model(N=(Nx, Ny, Nz), L=(Lx, Ly, Lz))
time_step!(model, Nt, Δt)
```

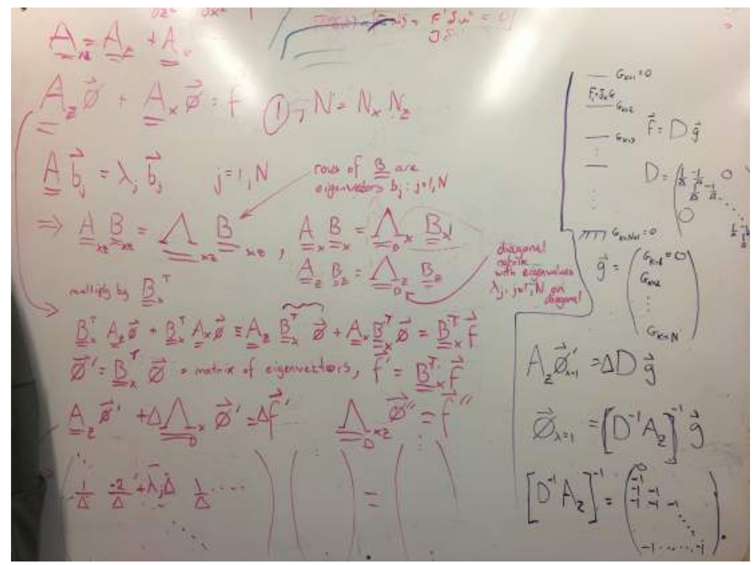

t = 0000000 s (0.00 days)
T (°C)
20.0
19.8
19.6
19.4
19.2
19.0
Z (m)
Y (m)
X (m)
T̄(z) [°C]
z (m)
Turbulent kinetic energy [m²/s²]
(u² + v²)/2
10 × w²/2
Buoyancy flux αgw′T′ [K m/s]
1e−8

Solver in Julia

```julia
function solve_poisson_3d_ppn_planned!(solver::PoissonSolver, g::RegularCartesianGrid, f::CellField, ϕ::CellField)
    solver.DCT!*f.data  # Calculate DCTᶻ(f) in place.
    solver.FFT!*f.data  # Calculate FFTˣʸ(f) in place.

    for k in 1:g.Nz, j in 1:g.Ny, i in 1:g.Nx
        @inbounds ϕ.data[i, j, k] = -f.data[i, j, k] / (solver.kx²[i] + solver.ky²[j] + solver.kz²[k])
    end
    ϕ.data[1, 1, 1] = 0

    solver.IFFT!*ϕ.data  # Calculate IFFTˣʸ(ϕ) in place.
    solver.IDCT!*ϕ.data  # Calculate IDCTᶻ(ϕ) in place.
    @. ϕ.data = ϕ.data / (2*g.Nz)
    nothing
end
```



GPU and CPU in one code

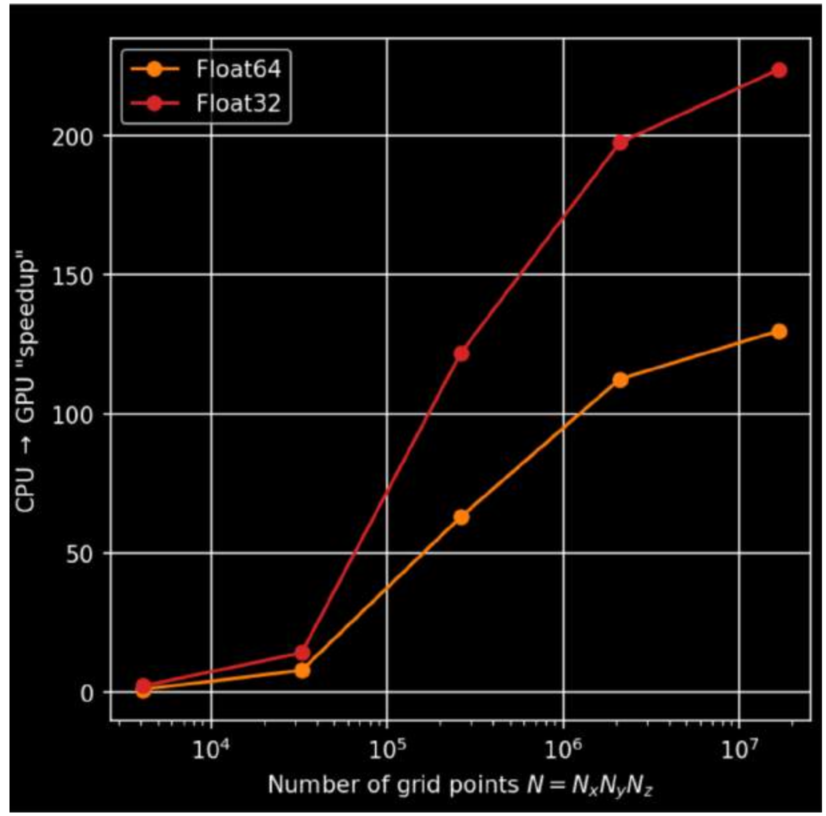
```
using GPUifyLoops
```

```
"Kernel for computing the solution `ϕ` to Poisson equation for source term `f` on a GPU."
function f2ϕ!(grid::Grid, f, ϕ, kx², ky², kz²)
    @loop for k in (1:grid.Nz; blockIdx().z)
        @loop for j in (1:grid.Ny; (blockIdx().y - 1) * blockDim().y + threadIdx().y)
            @loop for i in (1:grid.Nx; (blockIdx().x - 1) * blockDim().x + threadIdx().x)
                @inbounds ϕ[i, j, k] = -f[i, j, k] / (kx²[i] + ky²[j] + kz²[k])
            end
        end
    end
    @synchronize
end
```

GPUs

```
CPU -> GPU speedup:
32x 32x 32 static ocean (Float32): 14.138
32x 32x 32 static ocean (Float64): 7.829
64x 64x 64 static ocean (Float32): 121.806
64x 64x 64 static ocean (Float64): 62.924
128x128x128 static ocean (Float32): 197.906
128x128x128 static ocean (Float64): 112.417
256x256x256 static ocean (Float32): 223.748
256x256x256 static ocean (Float64): 129.923
```

Apples to apples
Google CPU core hour:GPU hour ➔ GPU is ~2-3x cheaper than CPU



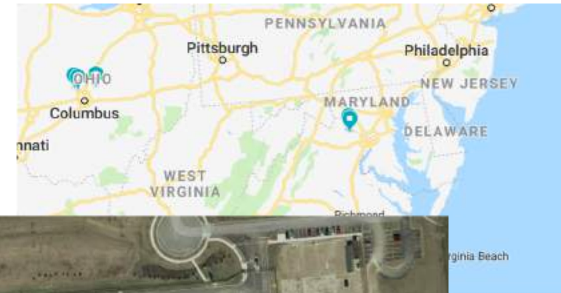
Single CPU core v V100 GPU (2650 64-bit GPU units)

Data centers, data analysis


$

v.

$$$





1/3 AWS US East2 (approx. 1/200 AWS globally)

$$$

$


Investing $10-20B+/year, every year

Cloud analysis

jupyter ecco_v4_example (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Pangeo ECCO Examples

This Jupyter notebook demonstrates how to use xarray and xgcm to analyze data from the ECCO v4r3 ocean state esti
Pangeo Binder from the pangeo_ecco_examples github repository.

First we import our standard python packages:

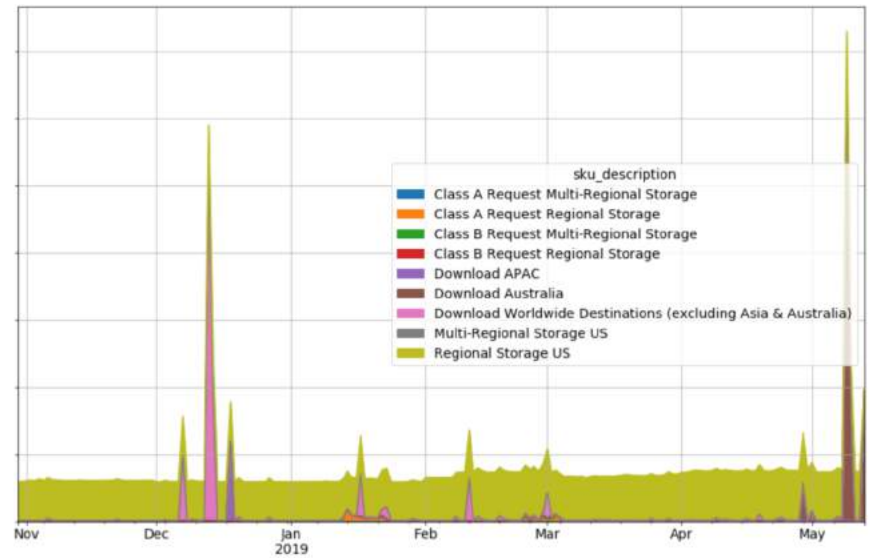
```
In [ ]: import xarray as xr
        import numpy as np
        from matplotlib import pyplot as plt
        %matplotlib inline
```

```
In [2]: import intake
        ds = intake.cat.ECCOv4r3.to_dask()
        ds
```

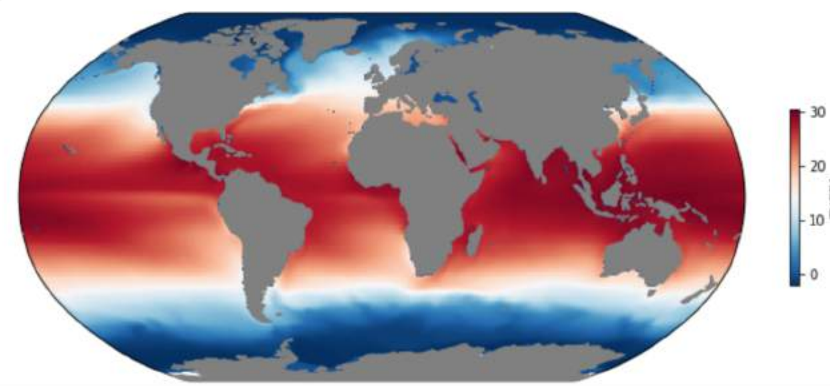
```
In [6]: sst = ds.THETA.se
```

```
In [8]: mean_sst = ds.THETA.sel(k=0).mean(dim='time')
```

Very slick, currently impractically expensive – by any honest analysis

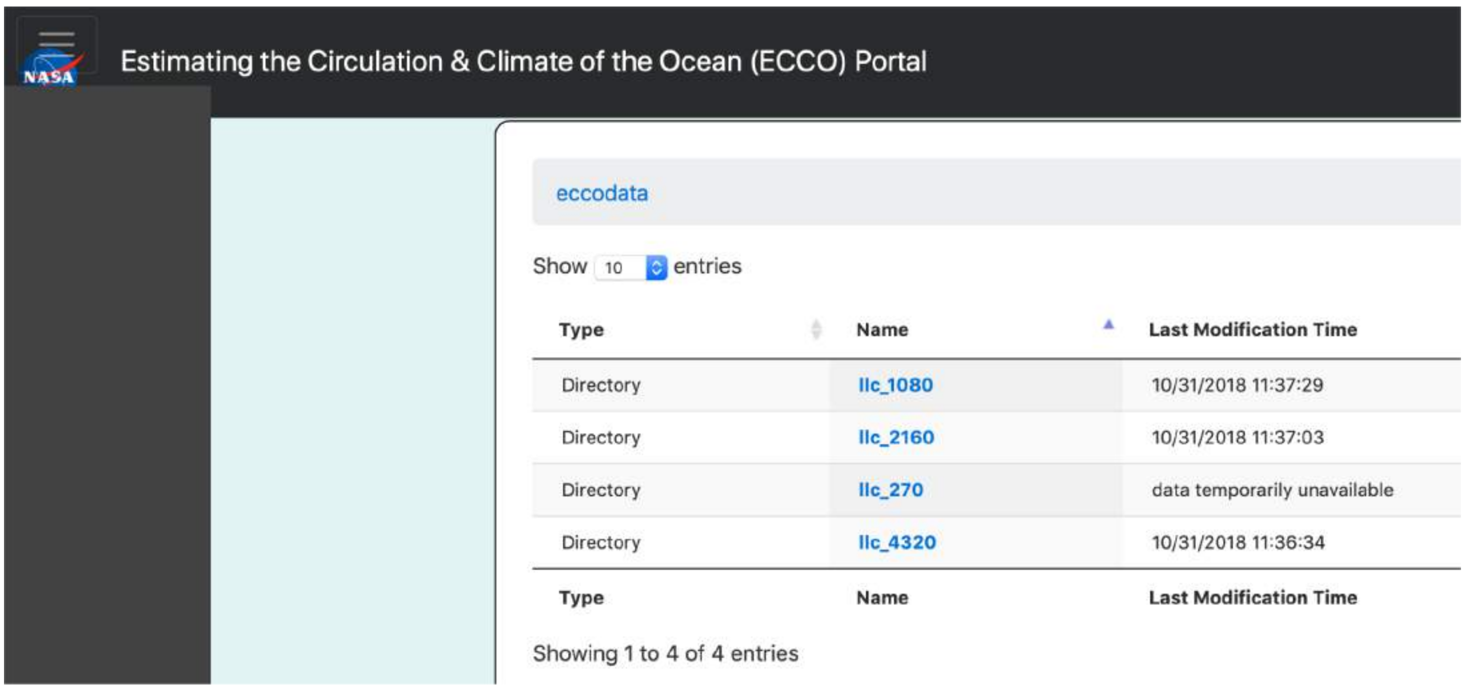



```
In [10]: mapper(mean_sst, cmap='RdBu_r');
```



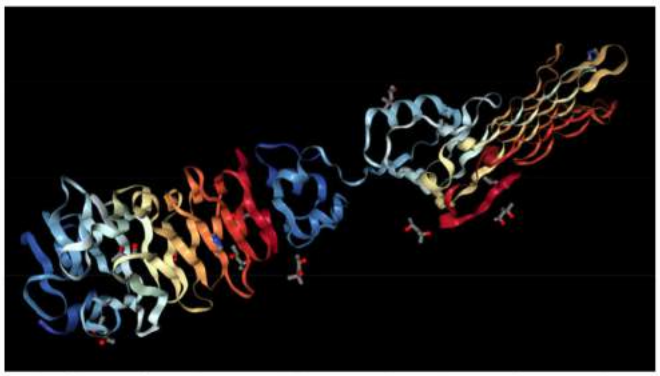
Home grown data center analysis

Most cost effective problems, but not as slick.



ML/AI – is mostly just statistics + lots of data + compute

- Some interesting papers/ideas



Complex of bacteria-infecting viral proteins modeled in CASP 13. The complex contains four separate subunits that were modeled individually. PROTEIN DATA BANK

Google's DeepMind aces protein folding

By Robert F. Service | Dec. 6, 2018 , 12:05 PM

Turns out mastering chess and Go was just for starters. On 2 December, the Google-owned artificial intelligence firm DeepMind took top honors in the 13th Critical Assessment of Structure Prediction (CASP), a biannual competition aimed at predicting the 3D structure of proteins.

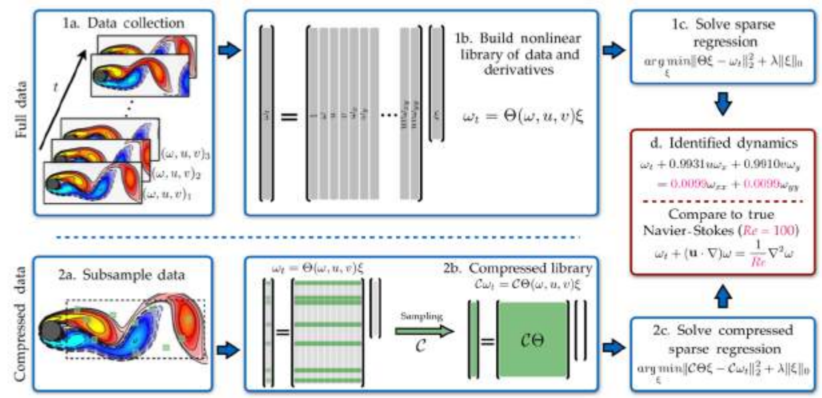
ML deep neural

APPLIED MATHEMATICS

Data-driven discovery of partial differential equations

Samuel H. Rudy,[1]* Steven L. Brunton,[2] Joshua L. Proctor,[3] J. Nathan Kutz[1]

We propose a sparse regression method capable of discovering the governing partial differential equation(s) of a given system by time series measurements in the spatial domain. The regression framework relies on sparsity-

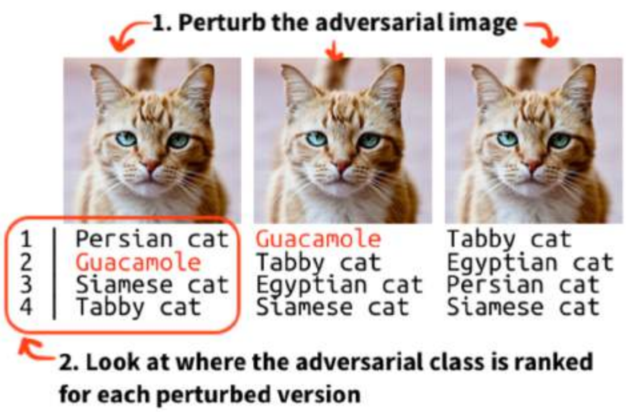


Guided Bayesian like search

Possible way forward for mesoscale experimentation?

Adversarial DL applied to pets….

Bus Ostrich



Synthesizing Robust Adversarial Examples

Anish Athalye [* 1 2] Logan Engstrom [* 1 2] Andrew Ilyas [* 1 2] Kevin Kwok [2]

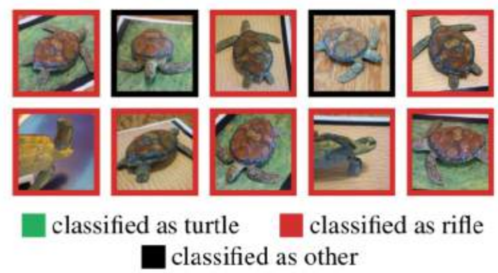


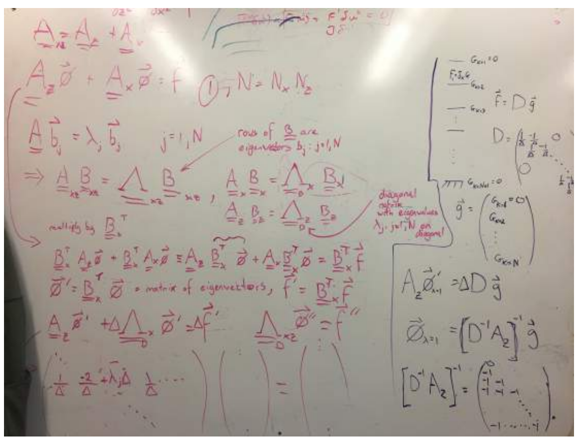
Figure 1. Randomly sampled poses of a 3D-printed turtle adversarially perturbed to classify as a rifle at every viewpoint[2]. An unperturbed model is classified correctly as a turtle nearly 100% of the time.

Incompressible Navier-Stokes - ML?

Accelerating Eulerian Fluid Simulation With Convolutional Network

Jonathan Tompson [1] **Kristofer Schlachter** [2] **Pablo Sprechmann** [2 3] **Ken Perlin** [2]

Convolutional network has some similarities with spectral filter, but computer solves for the algorithm….



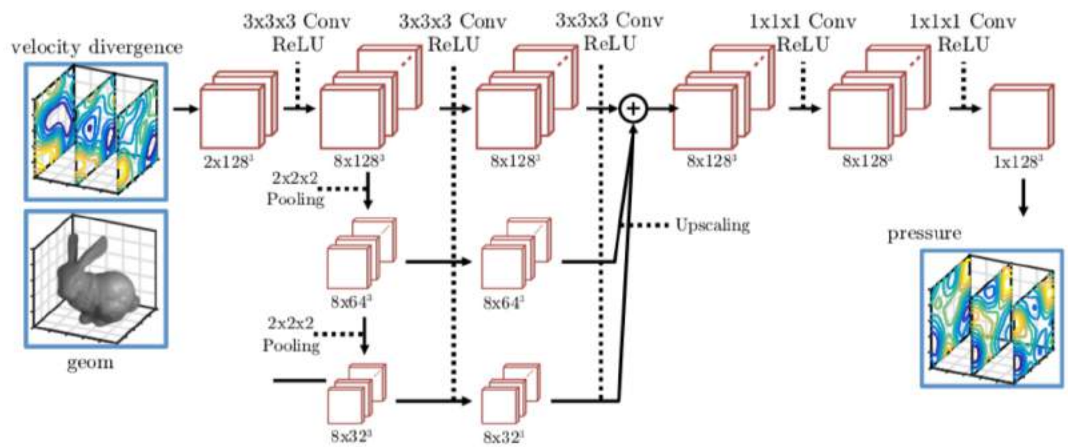


Figure 3. Convolutional Network for Pressure Solve