Phys 512
Lecture 1

Jonathan Sievers (jonathan.sievers@mcgill.ca)
https://github.com/sievers/phys512-2022

Schedule

- Lectures - TR 1:05-2:55.
- Tutorial - ?
- Office hours - ?
- Problem sets due - ? #11:59 PM Friday evenings

Default Marking

- 40% problem sets. Approximately one per 1-2 weeks.
- 30% project - larger scale program that can do something useful. Suggested problem n-body particle-mesh solver, but we are open to alternatives (clear first). When would you like this due?
- 30% final exam. Take-home, mix of programming and analytic problems.

Alternative - no project, then 60% problem sets, 40% final?

Assignments

- There are ~50 of you and ~3 TA's.
- Marking code is hard! There are a myriad of ways to be slightly wrong…
- We will suggest function names and input/output arguments. Do not deviate from that. Explain clearly (comments/text output) what you have done. Use functions!
- TA's will set time limit in marking per problem. If they don't know your answer is correct by then, no full marks.

Text Books

- None! However, Numerical Recipes is extremely useful as a pedagogical tool. Many discussions will come out of it. Gezerlis “Numerical methods in physics with Python” available from library as ebook.

- Other useful books (I’m told) include Computational Physics (M. Newman), Intro to Comp. Phys. (T. Pang), Numerical Methods for Physics, (A. Garcia).

Slack/Collaboration

- Feel free to talk amongst yourselves on how to solve problem sets etc.

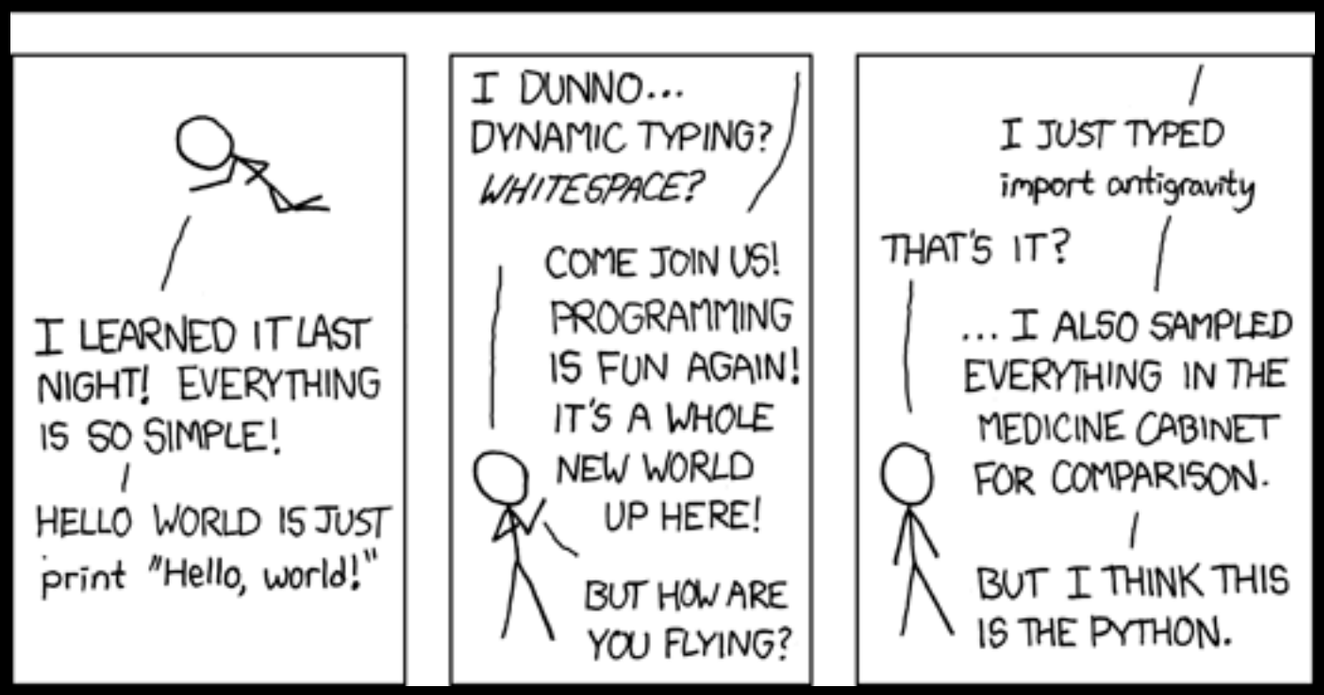
- Each person must write their own code independently, however. It’s really easy for us to tell if you copy...

- To enable you to collaborate, we will have a class Slack channel. Use it!

- Slack is for you to work amongst yourselves, not a venue to pester the TAs at all hours. If you DM the TAs, do not expect an answer. That’s why we have debug sessions.

Course Language

- Python! 3!

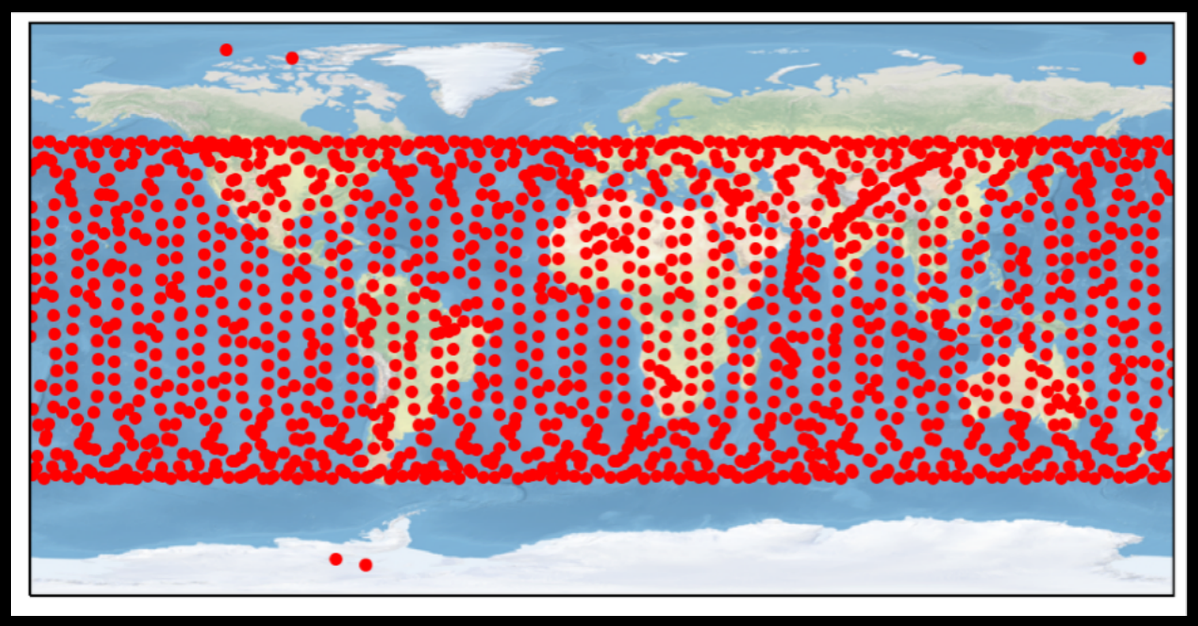


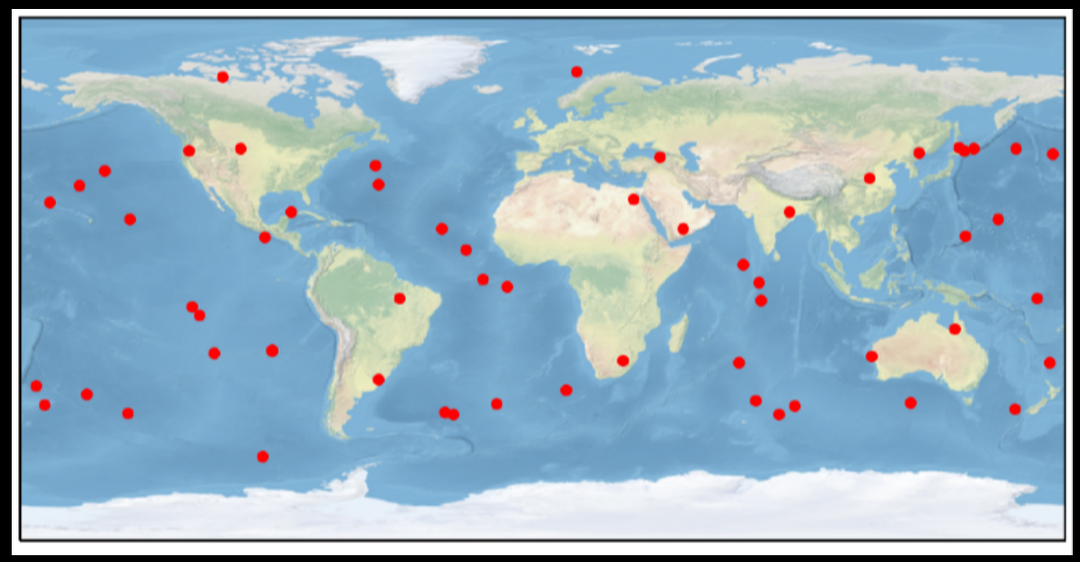
Example: 45 Line Jukebox

```
import sounddevice as sd
import time,glob,numpy,random
import subprocess

def get_key(mystr,key):
    i1=mystr.find(key)
    if i1<0:
        return 'Not found'
    i2=mystr.find('\n',i1+2)
    return mystr[i1+len(key):i2].strip()

def play_song(fname):
    faad='faad'
    to_exec=faad + ' -i '+ fname  #first, read header so we can report song info
    aa=subprocess.check_output(to_exec,stderr=subprocess.STDOUT,shell=True)
    nn=get_key(aa,'\nTille        :')#we'll get name of song
    writer=get_key(aa,'\nComposer     :')  #and composer
    myinfo=get_key(aa,'LC AAC') #we'll get some info about the file
    tags=myinfo.split()
    rate=tags[-2]
    nchan=tags[-4]
    length=tags[0]
    print 'playing ' + nn + ' by ' + writer + ' for ' + length + ' ' + tags[1]

    to_exec=faad+ ' -f2 -w -b4 ' + fname  #this magic command will have faad read data into float arrays
    ff=open('/dev/null','w')
    dd=subprocess.check_output(to_exec,stderr=ff,shell=True) #this calls faad to do the heavy lifting
    ff.close()

    dat=numpy.fromstring(dd,dtype='float32') #convert string output from faad into numpy array
    nchan=numpy.int(nchan)
    dat=numpy.reshape(dat,[len(dat)/nchan,nchan])
    fs=numpy.int(rate) #get the sample rate the speakers are expecting
    sd.play(dat,fs,blocking=False) #play through speakers using sounddevice
    return numpy.float(length) #return length of song.

if __name__=='__main__':
    dr='/Users/sievers/Music/old/ipod_touch/'
    fnames=glob.glob(dr + '/*/*.m4a') #get filenames from a directory
    random.shuffle(fnames) #randomly reorder the file names
    for fname in fnames: #loop over filenames in music library
        try:
            mylen=play_song(fname)  #play the song, and learn how long it is
            time.sleep(mylen)       #wait until the song should have finished
        except:  #when we get a ctrl-c, we'll jump here.  this just
            pass    #goes to next loop iteration, which overwrites current song
```

52 Line Real-time Satellite Viewer

```
import numpy as np
import ephem
import time
from matplotlib import pyplot as plt
import cartopy.crs as ccrs
op Store .lib import request

def read_latest_tles(url='https://celestrak.com/NORAD/elements/orbcomm.txt'):
    raw=request.urlopen(url)
    lines=[line.decode('utf-8').strip() for line in raw.readlines()]
    nsat=len(lines)//3
    tles=[None]*nsat
    for i in range(nsat):
        tles[i]=lines[3*i:3*(i+1)]
    return tles


def ctime2mjd(tt=None,type='Dublin'):
    if tt is None:
        tt=time.time()
    jd=tt/86400+2440587.5
    return jd-2415020
```





```
tles=read_latest_tles()

lat=np.empty(len(tles))
lon=np.empty(len(tles))
observer=ephem.Observer()

plt.clf()
ax=plt.axes(projection=ccrs.PlateCarree())
ax.stock_img()
for iter in range(100):
    tt=time.time()
    djd=ctime2mjd(tt)
    observer.date=djd
    t1=time.time()
    plt.ion()
    for i,tle in enumerate(tles):
        tle_rec=ephem.readtle(*tle)
        tle_rec.compute(observer)
        lat[i]=tle_rec.sublat
        lon[i]=tle_rec.sublong
    t2=time.time()
    if iter==0:
        plot_data,=ax.plot(lon*180/np.pi,lat*180/np.pi,'r.')
    else:
        plot_data.set_data(lon*180/np.pi,lat*180/np.pi)
    plt.pause(0.001)
    time.sleep(10)
```

Corollary

- First rule of coding: Don't!
- Someone has probably spent their career doing (almost) exactly what you're trying to do. Go use what they did!
- Your code will (almost certainly) be slower, buggier, less flexible, and more painful to use.

Course Outline

- essentials of math on computers "what every computer scientist should know about floating point arithmetic" Code style, unit tests, classes, never write the same block twice, measure twice/code once, VCS/git.

- numerical derivatives, interpolation/extrapolation - poly/rational function/spline

- numerical integration - simpson's rule, accuracy scaling with step size, adaptive. what can go wrong?

- ODEs - linear systems, RK4, stiff sets of equations.

- linear algebra - matrix multiplication, inverses, useful factorizations (Cholesky, eigen, SVD). χ^2 and linear least squares. Legendre/Chebyshev polynomials. Conjugate gradient, preconditioning.

- Nonlinear least-squares. Levenberg-Marquardt. MCMC. Statistical significance of results. function minimization/maximization, Newton's method, (nonlinear?) conjugate-gradient

Outline, part 2

- FFTs. Description of basic properties, interpretation as matrix multiply/coordinate rotation. Convolution/correlations with FFTs in 1 and 2+ dimensions. Nyquist theorem. real2complex FFTs

- root finding - bisection, newton's method. Random variables - uniform, generating via inverting CDF, Gaussians, rejection method, ratio of uniforms

- PDEs - boundary value problems, hyperbolic/elliptic equations, courant condition/stability. advection.

- Brief machine learning, pytorch

- Performance considerations - memory layout/access pattern consiterations. loops vs. vectorized calls in python. Intro to numba. GPU acceleration

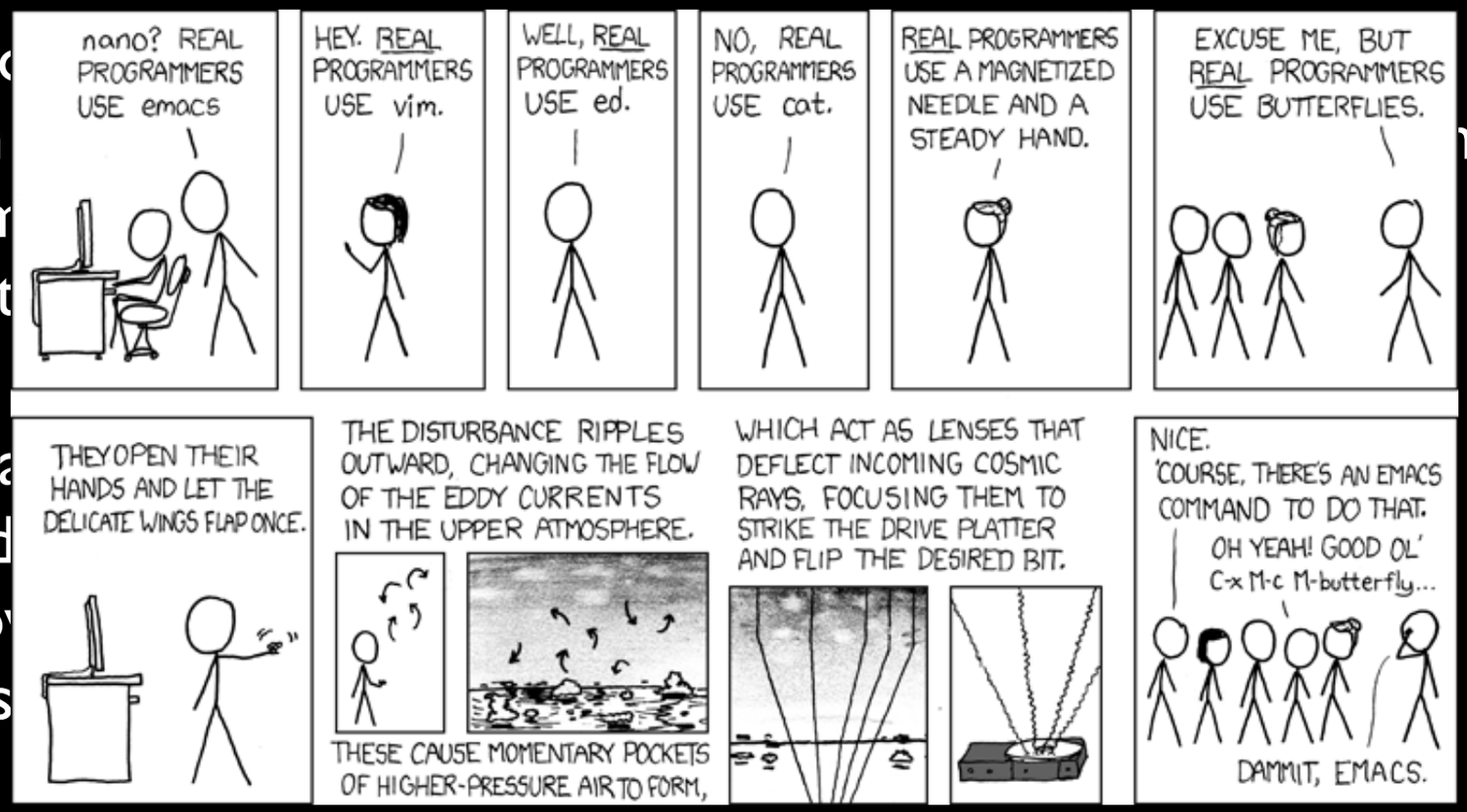
- basics of parallel computing. Embarrassingly parallel/python threading. What is already parallelized in python. Intro to mpi4py

- Do any of these sound boring? Are there things *you* want to see?

Text Editors

- At so
with
is em
gedit

- "ema
wind
follo
exits



Version Control

- Let's role play.

Version Control ctd.

- What did we just learn? It's important to be able to undo changes
- Merging changes from different places/different people important. Should be done in a systematic way. If I change one part of a file while you change another, would like to combine those changes automatically.
- What happens if someone steals your computer? The more places code lives, the safer you are.

Version Control 3

- Several version control (VC) systems exist, common include CVS, SVN, mercurial, and git. We will use git.

- VC lets many people edit files. A set of files being tracked is called a "repository." A VC system can keep your local files in a repository synced with the central repository.

- When you are done making changes, you "commit" them to your local repository, then "push" them to the central repository. When someone else makes changes, you "pull" them from the central repository.

- If you both edited the same part of a file, the VCS will thrown an error and tell you to fix the conflict by hand.

- VCSs usually require your local repository to be up-to-date with the central one before you're allowed to push changes.

Git

- git was de lds (the “Li” in linux)
- very pow
- Basic con itialize an empty repositor add those files to the repositor
- When re <message>” “ will commit your char .

If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of 'It's really pretty simple, just think of branches as...' and eventually you'll learn the commands that will fix everything.” https://xkcd.com/1597/

Github

- Github provides free repository support. Much of the code released today is done on github.

- You can make an account, then github will walk you through the steps you need to do what you want.

- “git clone” will copy a repository. I have made a github page for this class - you can get the slides plus a linux tutorial cheat sheet there.

- In the browser, go to “https://github.com/sievers/phys512-2024"

- Everyone should make their own github page, and homeworks etc. should be submitted via it.

What Every Computer Scientist Should Know About Floating-Point Arithmetic

DAVID GOLDBERG

Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, California 94304

- Never forget you are doing math/physics on a computer.

- Computers store floating point numbers as a set of bits, usually defined by IEEE-754.

- Single precision - 32 bits, 24 for mantissa, 8 for exponent.

- Double precision - 64 bits, 53 for mantissa, 11 for exponent.

You Should Care About Precision

```python
import numpy as np
import time
n=5000
a_double=np.random.randn(n,n) #generates a matrix in double
a_float=a_double.astype('float32') #convert to single

t1=time.time()
b_double=a_double@a_double #multiply double precison
t2=time.time()
b_float=a_float@a_float #multiply single precison
t3=time.time()
print('Time to multiply single/double precision: ',t3-t2,t2-t1)

#You will have to decide if the increased error from single is worth it
print('fractional difference: ',np.std(b_float-b_double)/np.std(b_double))
```

```
MacBook-Pro-8:lecture1 sievers$ python3 matrix_precision.py
Time to multiply single/double precision:  0.3189167976379394 5 0.6369478702545
fractional difference:  3.4133977107234 26e-07
```

Single precision is twice as fast! (and half the memory usage!). It's your job as the physicist to decide on tradeoffs.

With machine learning, many different precision options now supported.

Floating Point Representation

- What is the largest number we can represent in single precision?
- 8 bits in exponent, goes from -128 to +127. $2^{127}=127*\log10(2)$. $\log10(2)=0.301\ldots$ so, largest number ~ 10^{38}.
- In double? 11 bits, so $2^{(11-1)}=1024$, or 10^{308}.
- How about smallest *fractional* difference between two numbers?
- 24 bits in mantissa, so $1+2^{-24}$ (single) can't be represented. Double, $1+2^{-53}$ (base 10, fractional tolerance is ~10^{-7}/10^{-16} for single/double).

Numerical Derivative

- So, how would I take a numerical derivative?
- f' ~[f(x+dx)-f(x)]/dx. How should I pick dx to make this accurate?
- $f(x+dx)=f(x)+f'(x)dx+0.5*f''(x)dx^2+\ldots$
- First source of error - neglecting higher order terms in Taylor series. Will this get better or worse as dx gets smaller?
- Second source of error - in general, I can't represent f(x) exactly. So, f(x+dx)-f(x) will have error due to roundoff. Will this get more or less important as dx gets smaller?

Simplest Derivative

- deriv ~[(f(x)+f'(x)dx+0.5f"(x)dx^2)(1+$g_+\varepsilon$)-f(x)(1+$g_0\varepsilon$)]/dx
  ε is accuracy (10^{-7} for single, 10^{-16} for double), g_+,g_0 are order unity random numbers.

- Leading order, deriv=f'+f(gε)/dx+0.5f"dx. Absolute value ~fε/dx+0.5f"dx.

- Differentiate w.r.t dx: -εf/dx^2+0.5f"=0, or dx~$(\varepsilon f/f")^{1/2}$.

- What should dx be for exp(x) around x=0? Around x=10?

- We can do better by taking deriv ~ (f(x+dx)-(f(x-dx))/2dx. How should we pick dx now?

2-sided derivative

- Roundoff/numerical accuracy error the same
- 2nd order term killed in Taylor series
- Leading expansion error now dx^2 f”’
- Total error: ~fε/dx+f”’dx^2.
- Differentiate: -fε/dx^2 + f”’dx.
- Set equal to zero & solve: dx ~(fε/f”’)$^{1/3}$.
- What should dx be for exp(x) in single and double precision?
- Would you do anything differently for sin/cos?

Numerical Derivative Errors