❑ LECTURE 3

- Connectivity and security of computers
- Learning Linux and its shells
- Languages of High Performance Computing (HPC)
- Simple programs
- and not so simple programs in three languages: *.py .c .f95*

Literature: see our PHYD57/refs and the hidden subpage

We will return to the further historical development of hardware a bit later, after we explore the operating system types that conquered the world (not only of HPC, but also of Internet servers). Incidentally, learn some Internet history here.

Updated list, cf. our resource page /refs and the other recommended page..

Sobell – A practical guide to Linux Commands – 2018
Membrey – Definitive Guide to CentOS – 2009
Negus –Fedora Linux Toolbox. CentOS, Red Hat – 2007
Gedris – Intro to Linux Command Shell for Beginners – 2003

Good web resources:
www.linfo.org - on various topics about Linux, see all the links on that page, among others
www.linfo.org/command_index.html - list commands with further explanations on the use
www.linfo.org/how-to_index.html - how-to guides

Well.. beside its history, how does the networking we use today actually work? We saw examples of domain blocking.

Networking is a separate magic/technology.

Find out yourself by... using it to see the videos below!

Series of informative videos about networking

2. https://www.youtube.com/watch?v=kn7ei2ENJbI
3. https://www.youtube.com/watch?v=fH-fiVm5__o
4. https://www.youtube.com/watch?v=j3LXv7RNFLY
5. https://www.youtube.com/watch?v=mGRClHHgNdk
6. https://www.youtube.com/watch?v=t1MvzaScl9k
7. https://www.youtube.com/watch?v=fyS3QdYuMZI
8. https://www.youtube.com/watch?v=uTE48tWIxuI
9. https://www.youtube.com/watch?v=Cx7foWGm5fo
10. https://www.youtube.com/watch?v=o4xkvyZnhj4
11. https://www.youtube.com/watch?v=O48yc2nYBRA
12. https://www.youtube.com/watch?v=_Cv5OnPHo-k
13. https://www.youtube.com/watch?v=b_9Dg0QYJUg

Why is the knowledge of networking needed in science & business world?

1. We often build and/or maintain our computational & web servers
2. We must be able to securely access them remotely with ssh
3. Transfer data via sftp
4. We would like to sometimes automate access to external sites (e.g. downloads)
5. We need to watch out for bad guys who would like to beak in:

   I will now give you a recent example of break-in attempts on machines on our campus (linux science cluster).

CentOS and other Red Hat derivatives store failed logins & other important events in log file /var/log/secure and similar secure.### (usually date range is attached)
The activity below is known as phishing. As you can see, it was *very frequent*. Cumulatively, more than 1 per second on a few dozen machines. It would happen even much more frequently if wrong passwords did not generate mandatory ~3 s delay.
To circumvent timing hurdles, bots sometimes vary the port numbers (..)

```
Jan 16 03:45:28 art-1 sshd[6785]: Failed password for invalid user root from 185.45.236.154 port 48693 ssh2
Jan 16 03:45:30 art-1 sshd[6785]: Failed password for invalid user root from 185.45.236.154 port 48693 ssh2
Jan 16 03:45:36 art-1 sshd[6788]: Failed password for invalid user test2 from 185.45.236.154 port 48709 ssh2
Jan 16 03:45:38 art-1 sshd[6788]: Failed password for invalid user test2 from 185.45.236.154 port 48709 ssh2
Jan 16 03:45:45 art-1 sshd[6792]: Failed password for invalid user contador from 185.45.236.154 port 48717
Jan 16 03:45:48 art-1 sshd[6792]: Failed password for invalid user contador from 185.45.236.154 port 48717
Jan 16 03:45:53 art-1 sshd[6796]: Failed password for invalid user duni from 185.45.236.154 port 48728 ssh2
Jan 16 08:09:53 art-1 sshd[7494]: Failed password for invalid user pi from 186.149.191.94 port 60334 ssh2
Jan 16 08:09:53 art-1 sshd[7496]: Failed password for invalid user pi from 186.149.191.94 port 60338 ssh2
Jan 16 15:45:38 art-1 sshd[8605]: Failed password for invalid user pi from 24.229.175.230 port 47494 ssh2
Jan 16 15:45:38 art-1 sshd[8608]: Failed password for invalid user pi from 24.229.175.230 port 47500 ssh2
```

Why is the knowledge of networking needed in science & business world?

1. We often build and/or maintain our computational & web servers
2. We must be able to securely access them remotely with ssh
3. Transfer data via sftp
4. We would like to sometimes automate access to external sites
5. We need to watch out for bad guys who would like to beak in:

   I will now give you a recent example of break-in attempt on machines on our campus

Examples of phishing that triggered the linux software to log break-in attempt :

They are of the type:

reverse mapping for 5.58.178.186.static.anycast.cnt-grms.ec [186.178.58.5] (..) failed – POSSIBLE BREAK-IN ATTEMPT!

Attempts came, or masqueraded as coming, from all sort of domains/countries:

.br, .ru, .ec, .es, .tr, .nl, .ca, .us, .id, .vn, .cn, .by, .de, .ar, .it, .uk, .nz etc.

A few examples:

Dec 24 19:20:48 art-7 sshd[15215]: reverse mapping for 188.243.246.181.pool.sknt.ru [188.243.246.181] failed (...)
Jan 1 10:14:12 art-3 sshd[21777]: reverse mapping for 5.58.178.186.static.anycast.cnt-grms.ec [186.178.58.5 ...
Jan 1 10:14:26 art-3 sshd[21780]: reverse mapping for 5.58.178.186.static.anycast.cnt-grms.ec [186.178.58.5 ...
Jan 9 00:17:40 art-3 sshd[22166]: reverse mapping for mail.macmasters.ca [199.243.203.174] failed
Jan 9 00:17:55 art-3 sshd[22169]: reverse mapping for mail.macmasters.ca [199.243.203.174] failed -
Jan 9 00:19:06 art-3 sshd[22186]: reverse mapping for mail.macmasters.ca [199.243.203.174] failed -
Jan 9 08:10:40 art-3 sshd[23629]: reverse mapping for 78.166.122.31.dynamic.ttnet.com.tr [78.166.122.31]
Jan 14 12:48:22 art-4 sshd[14919]: reverse for ip-46-72-134-210.bb.netbynet.ru [46.72.134.210] failed -
Jan 14 12:48:27 art-4 sshd[14922]: reverse mapping for ip-46-72-134-210.bb.netbynet.ru [46.72.134.210] failed -
Jan 14 12:48:47 art-4 sshd[14927]: reverse mapping for ip-46-72-134-210.bb.netbynet.ru [46.72.134.210] failed
etc.

How do we react?

- First and foremost: we DO NOT ALLOW USE of account names and passwords easy to guess or to randomly produce (too short or obvious, for instance).

Example: account id = student, password = physics; account id = test, etc.

- Next, we tighten firewalls by using /etc/hosts.deny to exclude connections from IP addresses or address ranges, sometimes from whole top domains if needed (because the attacking bots mutate the subdomain names).

art-1[190]:~$ more /etc/hosts.deny this file now shields the cluster fairly well but not 100%

```
#
# hosts.deny    This file contains access rules which are used to deny connections to network
#               services that either use the tcp_wrappers library or that have been started through a tcp
#               _wrappers-enabled xinetd.  The rules in this file can also be set up in /etc/hosts.allow
#               with a 'deny' option instead.
#               See 'man 5 hosts_options' and 'man 5 hosts_access'
#               for information on rule syntax. See 'man tcpd' for information on tcp_wrappers

sshd :  109.125., 196.50.,  \
222.,218.,209.,201.,199.,197.,193.,192.,191.,190.,183.,185.,186., 187.,189.,188.,\
170.,171.,176.,177.,178.,179.,168.,\
156.,151.,144.,143.,140.,138.,120.,122.,123.,112.,114.,116.,117.,118.,119.,\    my mistake**)
109.,108.,107.,102.,90.,91.,92.,93.,94.,95.,\
 81.,84.,85.,86.,87.,88.,89.,73.,75.,77.,78.,79.,62.,60.,67.,45.,46.,49.,\       domains seen on prev.
37.,38.,39.,31.,24.,14.,5.                                                       slide are shown in color
```

NOTE: 109.125. is equivalent to 109.125.*.* and so on. ** - that's a campus wifi !!

confusingly, %ifconfig does not show "138."

Let's talk about more Linux commands (continued from L2)

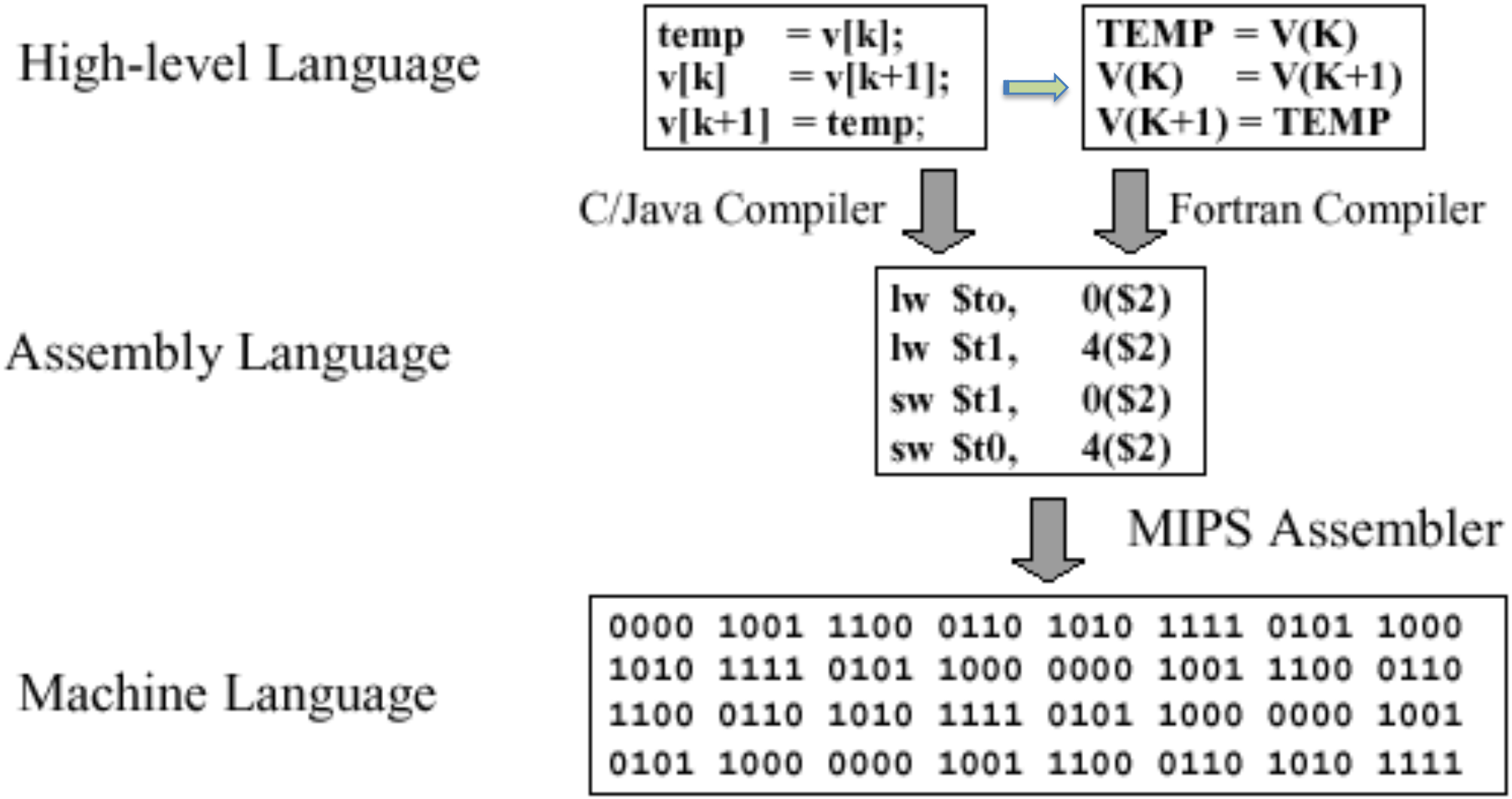
Updated list, cf. our resource page /refs and the other recommended page .../##

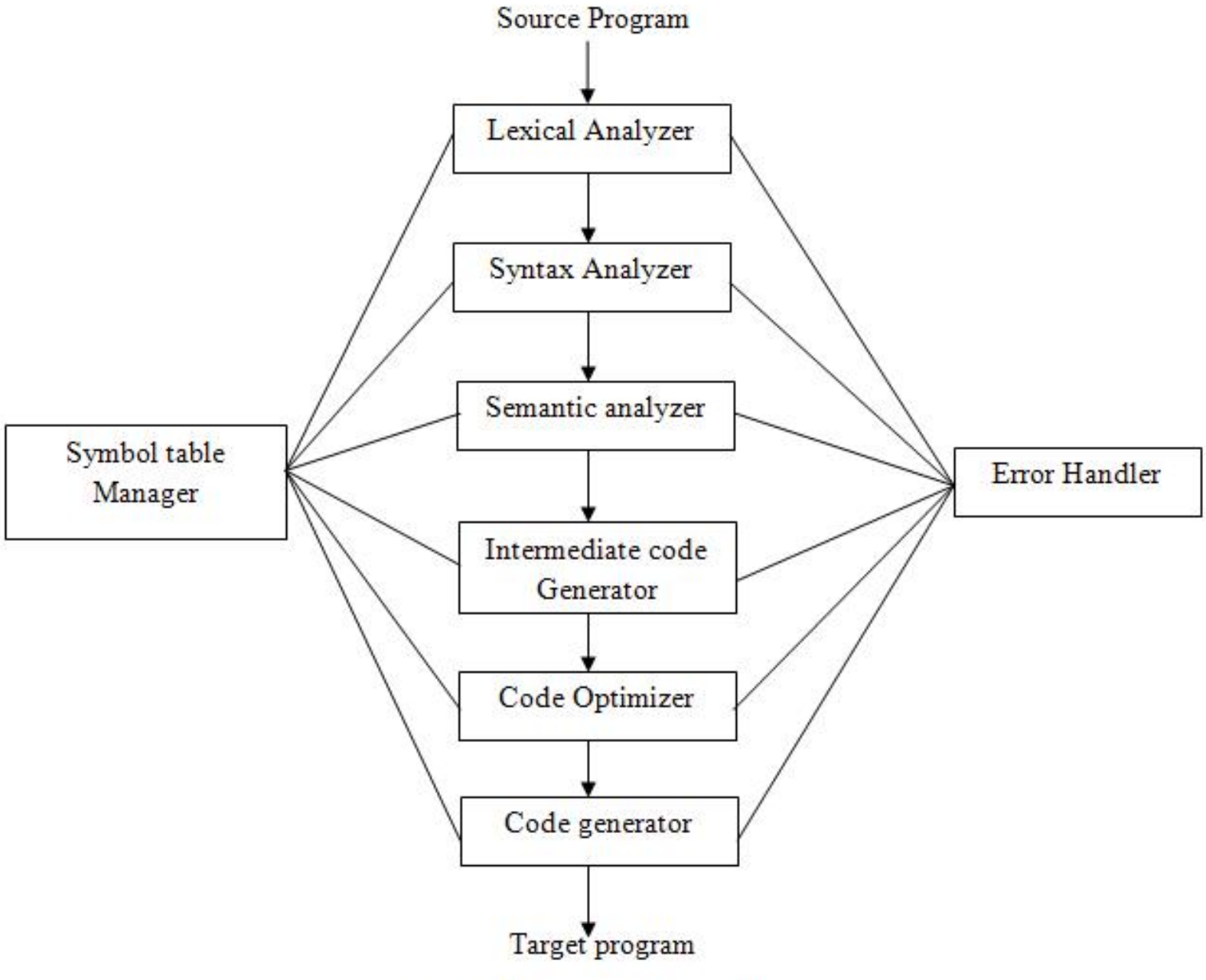
Sobell – A practical guide to Linux Commands – 2018
Membrey -- Definitive Guide to CentOS – 2009
Negus –Fedora Linux Toolbox. CentOS, Red Hat – 2007
Gedris – Intro to Linux Command Shell for Beginners-2003

Good web resources:
www.linfo.org - on various topics about Linux, see all the links on that page, among others
www.linfo.org/command_index.html - list commands with further explanations on the use
www.linfo.org/how-to_index.html - how to guides

COMPILER. What does it do for a living?

Translates from one language into another – lower level one, but its flexible & can be guided by your options/flags (in command line when invoked) and special directives you put inside the program.




Source Program
Lexical Analyzer
Syntax Analyzer
Semantic analyzer
Symbol table Manager
Error Handler
Intermediate code Generator
Code Optimizer
Code generator
Target program

Scientific Programming. Which Compilers to use?

Python 3

Not a compiler, its an interpreter, although a clever one and able to use efficient, pre-compiled libraries and data structures (arrays in Numpy, for instance). It also checks the syntax before running a script.

C/C++

Since C++ is not recommended at our level of usage by the references cited in the Links section of our course home page, we will learn straight C. C++ is a bit more difficult to learn and debug, but may be better is some instances. It usually runs almost as fast, and never much faster than a plain C code.

On the other hand, compilers can mix C and C++ (for instance, many math libraries are currently written in C++). Names of recommended compilers (art-1):

gcc (or g++) - GNU C compiler on your/my machine

icc (or icpc) - Intel compiler

pgcc – Portland Group/Nvidia compiler (now PGI belongs to Nvidia)

nvcc = gcc + CUDA [GPU-specific extension of C/C++] by Nvidia

Programming. Compilers.

All the previously mentioned and these compilers (at least some versions of them) are downloadable for free, for students and educators.

Fortran 95 Recommended compilers

gfortran = gf - GNU compiler. Very good quality. Open source project.
No CUDA yet, but planned.

ifort - Intel compiler(s). Exceptionally good & works with CPU and MIC hardware. No full GPU support, but something is happening...
Provides almost everything in Parallel Studio XE bundle, license may or may not work forever, ~2 updates available per year.

pgfortran =pgf90=pgf95 – Portland Group compiler incl. CUDA Fortran is currently the only way to compile own GPU kernels in extended Fortran
GPUs (PGI is now owned by NVidia, producer of GPUs, which supports scientific research, provides compilers & GPU libraries free of charge to academics.)
Download Community Edition [bundle, C(++)/F95]
license will be valid for 1 year, upload new version afterwards.

It is possible to create multi-language and multi-compiler software.

- How to quickly find out what options are support by a compiler

(Fortran in this case, but the same for C++):

1. % man gfortran % man ifort
2. % gfortran –help % ifort -help % pgfortran –help

One can filter the voluminous output of these commands by piping it to *grep* –i utility [=general regular expression parser].
Or redirecting output to file via % progname > file syntax, and reading it in editor.
Finally, by entering the manual page and searching within it like in vi editor:

For instance,
... | grep –i omp will pick up all lines with “omp” (ignoring case)
and
“/omp” inside a manual page browser (Vi-style) will also find and maybe highlight occurrences of “omp” string.

- How to quickly recall any Linux command name?

Use the automatic *name completion* feature of the linux shell; type one or two first letters that you remember or suspect, and TAB. All system commands and even your executables that are in your path will be shown. Then you can try reading manual pages if they exist, or supply –help or --help modifiers to the command.

It is customary to start with the minimal “Hello World!” program. OK, here is one in C, compiled and run on a Windows machine (right panel) , Python2, C++ and Java, as well as in Fortran.

HelloWorld.c

```c
#include <stdio.h>
void main()
{
printf("Hello, World!");
}
```

C

Command Prompt

```
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. All rights reserved

C:\Users\Sahil Agarwal>D:

D:\>cd D:\Programming_In_C

D:\Programming_In_C>gcc HelloWorld.c -o HelloWorld

D:\Programming_In_C>HelloWorld
Hello, World!
D:\Programming_In_C>
```

```
print "Hello World!"
```

Python

```
#include <iostream.h>
int main()
{
cout << "Hello World!";
}
```

C++

```
public class helloWorld
{
    public static void main(String [] args)
    {
    System.out.println("Hello World!");
    }
}
```

Java

print*, “Hello World!”
Or
print*, ‘Hello World!’
Fortran

read Shoerghofer’s
“Lessons in Sci. Computing”
Chapter 4
Programming Languages

Only slightly more complicated example in programs:
C-example.c F-example.f90 (.f95) P-example.py
can be found in art-1: ~/simple, that is in
art-1.utsc.utoronto.ca:/home/phyd57/simple path to directory on the net via *sftp*

Account phyd57 is common to all of you. Don't modify the subdir's that you see there, don't work there. You have your own subdirectory (sandbox ☺) to play in, such as:

% mkdir 309 ← 3 last digits of student # for my ease of identifying you.
Then cd to your new sandbox directory & copy all the files you need from other subdir's like so [~ mean top of the used directory, in this case /home/phyd57]
% cd 309

```
309% cp ~/simple/*   .        ← the dot means destination is current dir i.e. ~/309
309% icc C-example31.c    [-o C-example31.icc.x]
309% pgcc C-example31.c   [-o C-example31.pgcc.x]
309% nvcc C-example31.c   [-o C-example31.nvcc.x]
309% ifort F-example.f90 [-o F-example.ifo.x ]
309% pgf90 F-example.f90 [-o F-example.pgf.x ]
309% pgf95 F-example.f95 [-o F-example.pgf.x ]
309% pgfortran F-example.f95 [-o F-example.pgf.x]
309% F-example.pgf.x          ← execute various .x files or a.out
(..output..)                    [ ] means that this part of command is optional
                                it's the option to name output, i.e. the executable
```

Numerical Calculus: Differentiation: cf. Assignm. A1

Numerical Calculus: Integration

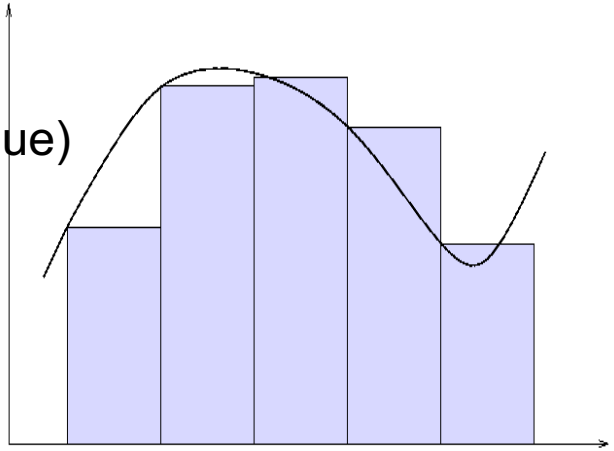
Developing coefficients c_0, c_1, ... of approximation polynomials in a *small* interval covered by just several computational points (samples of data or math function).

Def.: Order of the method is *m* if it integrates polynomials or piecewise polynomials up to order m exactly, and exhibits error proportional to h^{m+1} where *h* is the interval's width, for other functions.

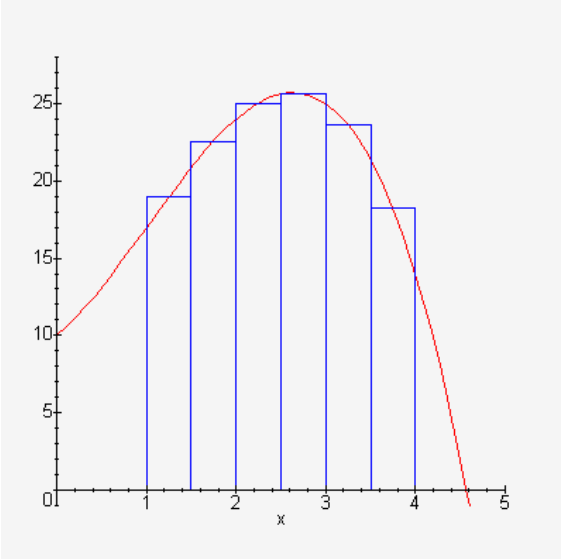
Require that method is of order m, and write m+1 equations (for n=0,1,2,...,m) binding m+1 coefficients of the method. Solve for c_n.

The integration points may be uniformly spaced in the x-interval or their positions in the interval may also be unknown (there must be exactly m+1 unknown x's and c's – read all about Gauss integration formulae on p. 56 of textbook)

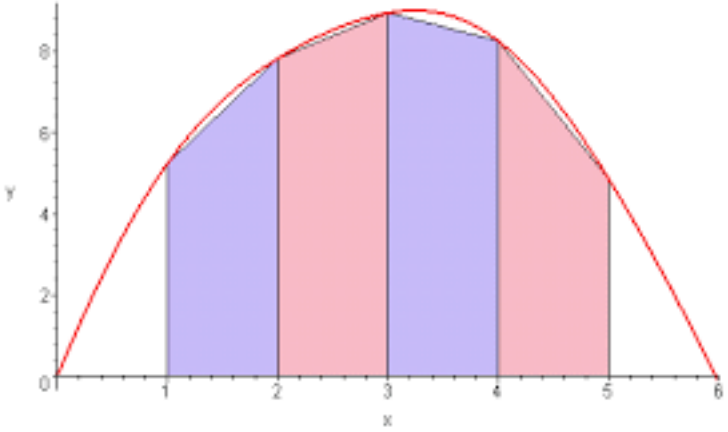
Euler(left value)
1st order

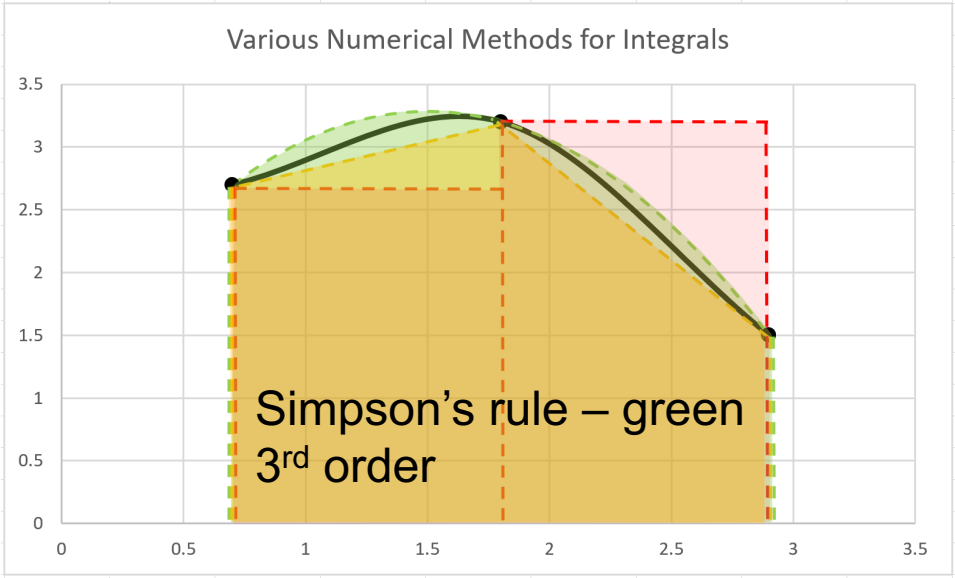


Midpoint method
2nd order



Trapezoid method
2nd order





Numerical integration of functions

see http://planets.utsc.utoronto.ca/~pawel/progD57 (also art-2: ~/progD57)

- Integrate *exp(-x)* from 0 to 1: integ-p124-exp.py
- Integrate $1/(1+x^2)$ from 0 to 1: integ-p124-arc.py
- Integrate the length of a curtain: integ-p124-cur.py
- Integrate ¼ circle (area): integ-p124-Acirc.py
- Integrate ¼ circle (length): integ-p124-Lcirc.py

We implement and compare these methods :

- ❑ Euler's method
- ❑ Midpoint method
- ❑ Trapezoid rule
- ❑ Simpson's rule (so-called 1/3 rule, a 3-point rule)
- ❑ Pythagoras rule for line integrals

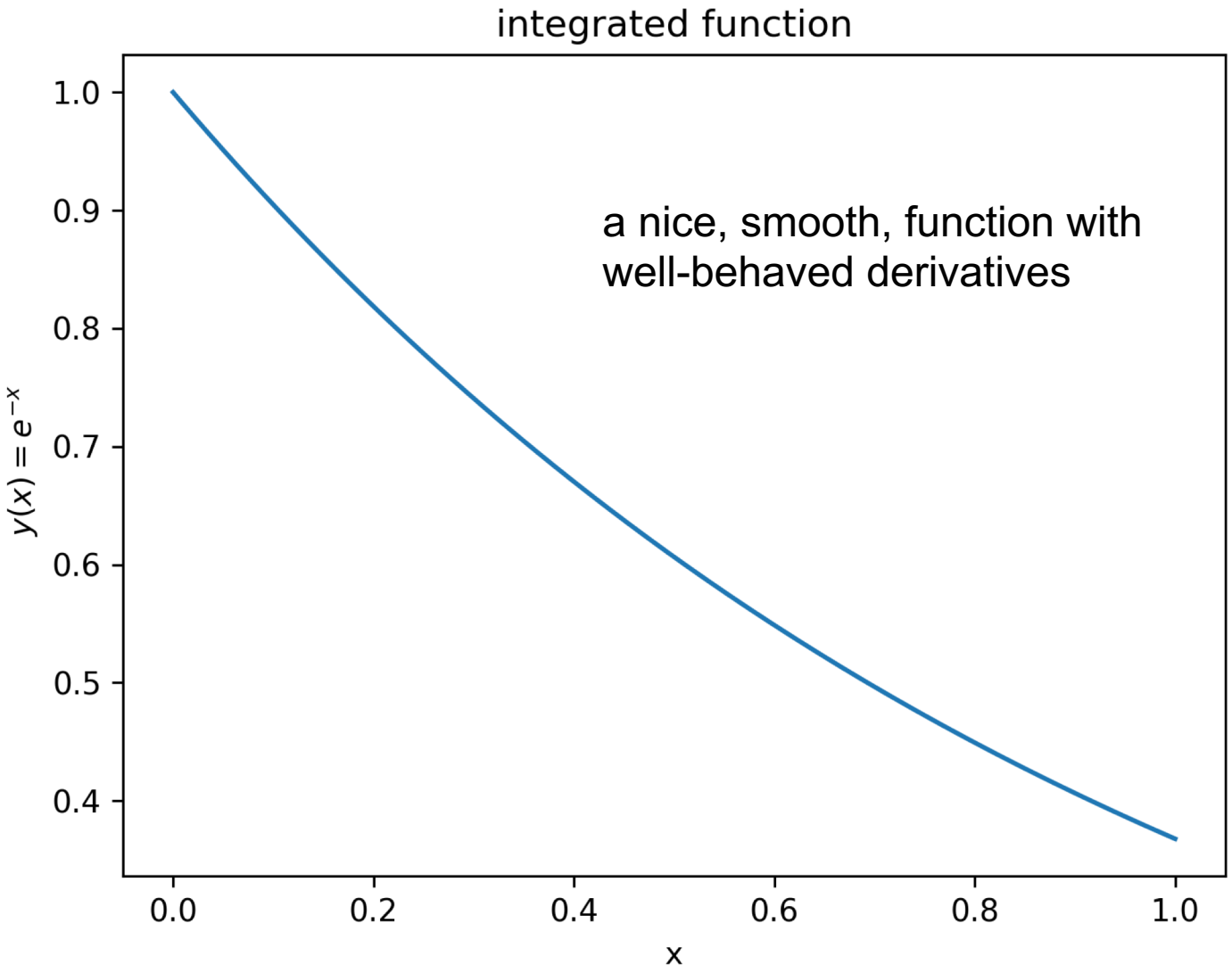
Composite integration formulae – combining N small sub-intervals of width h.

N+1 points, forming N uniform subintervals covering x-interval [a,b].
We have equal integration steps $h = (b-a)/N = x_{n+1} - x_n$, for all intervals n=0,1,2,...

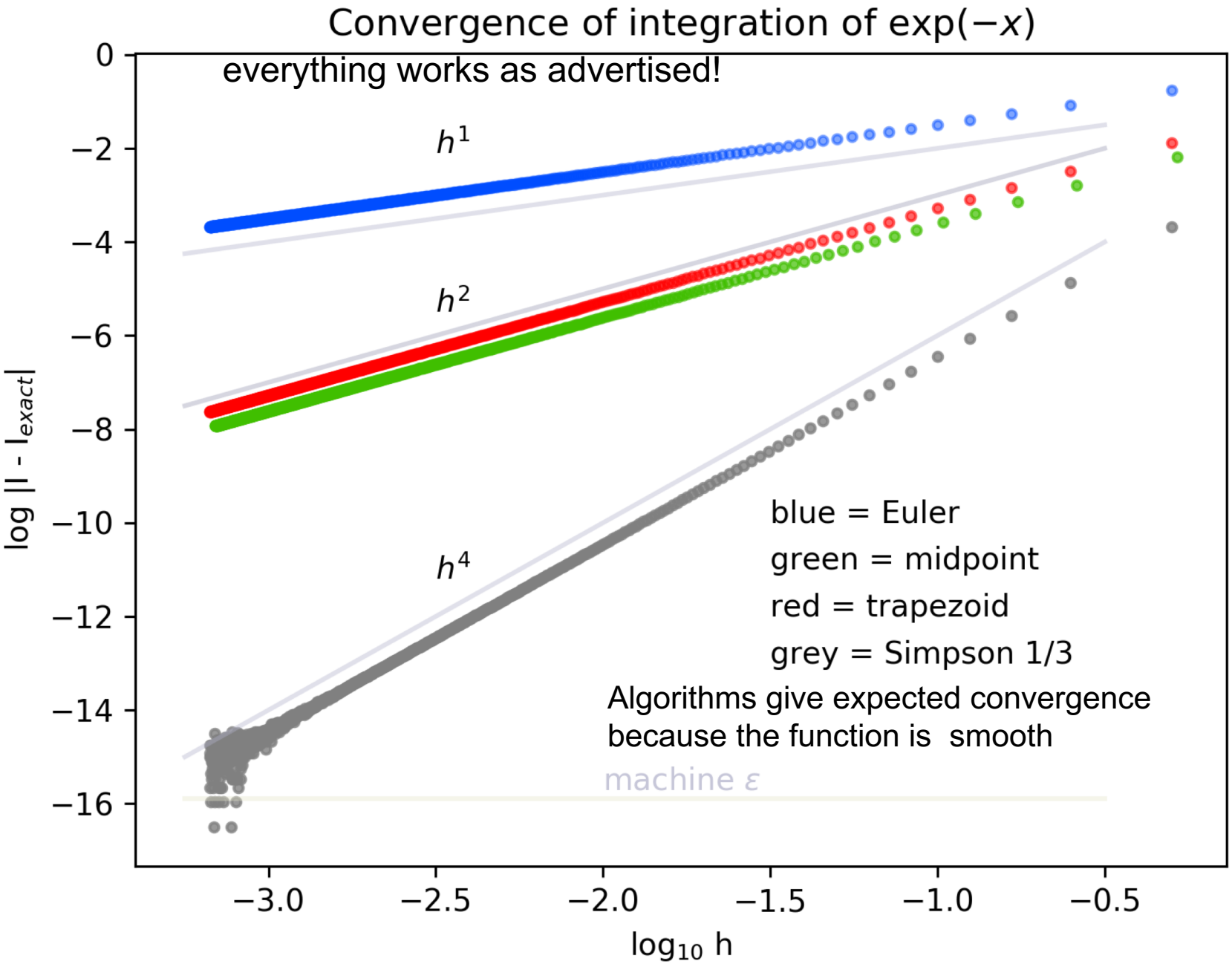
- Euler (left) $S = (f_0 + f_1 + f_2 + \ldots f_{n-1})\, h$
- Euler (right) $S = (f_1 + f_2 + f_3 + \ldots f_n)\, h$
- Midpoint $S = (f_{1/2} + f_{3/2} + \ldots + f_{n-3/2} + f_{n-1/2})\, h$, $f_{1/2} := f((x_0+x_1)/2)$ etc.
- Trapezoid $S = ((1/2)f_0 + f_1 + f_2 + f_3 + \ldots + f_{n-1} + (1/2)f_n)\, h$
- Simpson's 1/3 rule: $S = (1/3)\,(f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \ldots + 4f_{n-1} + f_n)\, h$
- Pythagoras length summation: $\Sigma_{n=1..N} \; [(x_{n+1}-x_n)^2 + (f_{n+1}-f_n)^2]^{1/2}$

- Notice that integration errors accumulate, and will be N times larger than the single-subinterval error of basic formulae, i.e. 1/h times larger because $N = (b-a)/h \sim h^{-1}$.
- If the error on one sub-interval was $\sim h^p$, then the error on bigger interval [a,b], made of N sub-intervals, will be $\sim N\, h^p \sim h^{p-1}$ (method of order p-1)
- Thus, in interval [a,b] we lose one power of h in accuracy scaling, or one unit in the order of the method, compared to each subinterval.

Integrate exp(-x) from 0 to 1: integ-p124-exp.py



integ-p124-exp.py




Galaxies M82 and M81
Copyright Astrosib250

2-D diffusion, un-sharp masking, a method of mage processing – and its use as benchmark

Finally, consider programs in other languages below,
placed in art-1 subdirectory ~/progD57, i.e. /home/phyd57/progD57/
(Reminder: art-1 has linux flavor CentOS 6, and defaults to shell called tcsh)

They perform the same task: to repeatedly, locally blur a monochromatic NxM pixel image, according to a 3x3 cross stencil of a Laplacian operator $\Delta f = \nabla^2 f$

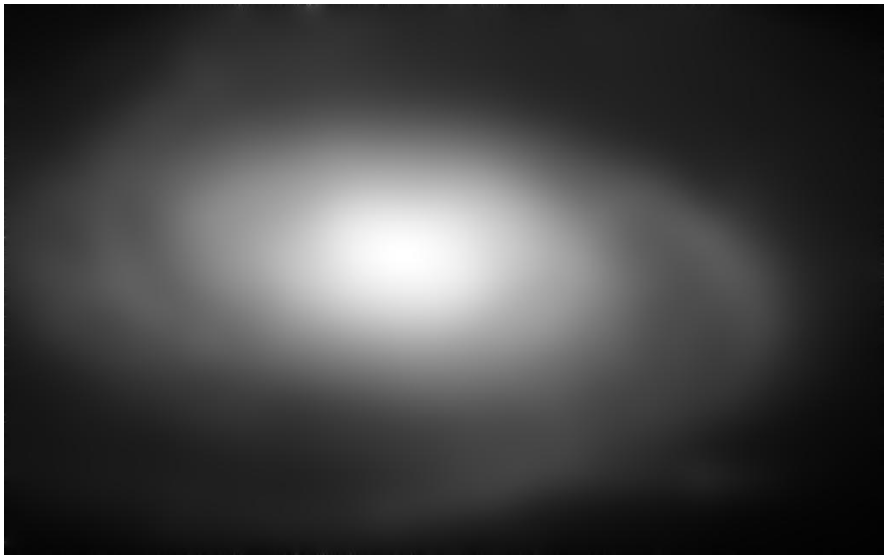
```
      q
q   (1-4q)  q
      q
```

2-D stencil of the Δf operator. Coefficients must sum to 1 (check!)

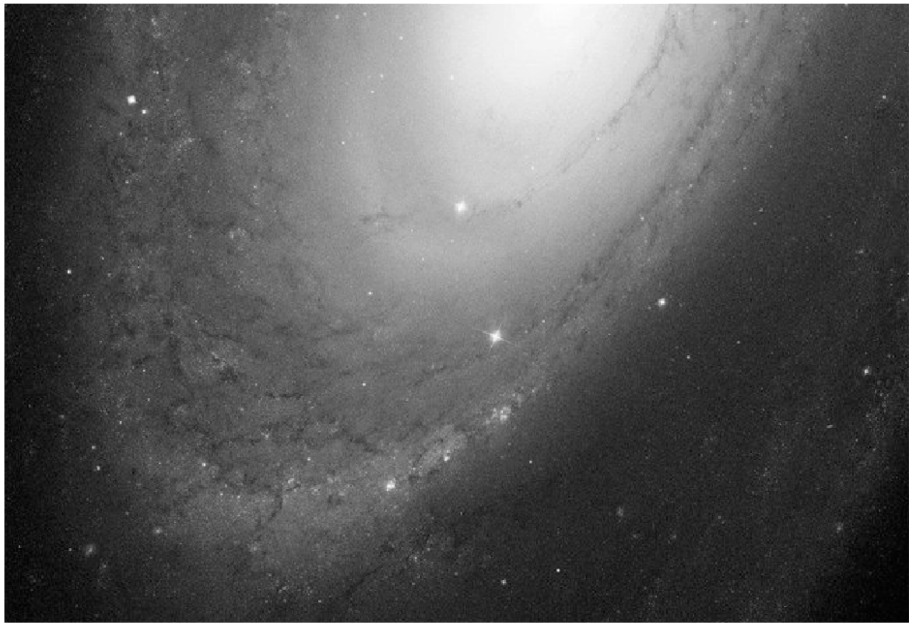
Read, copy, compile with all possible compilers, and execute these programs

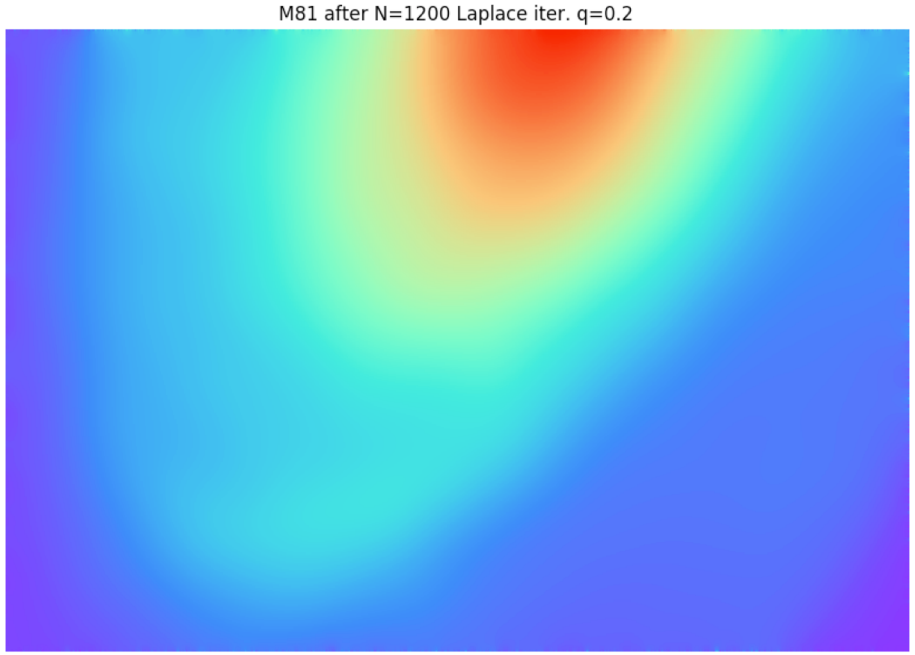
laplacian-5.py*	laplacian-5t.py*
ifor-laplace3-sp.f90	ifor-laplace3-dp.f90
cudafor-laplace3-sp.f95	cudafor-laplace3-dp.f95

* - run python3 program on your own machine, it's not fully installed on art-1

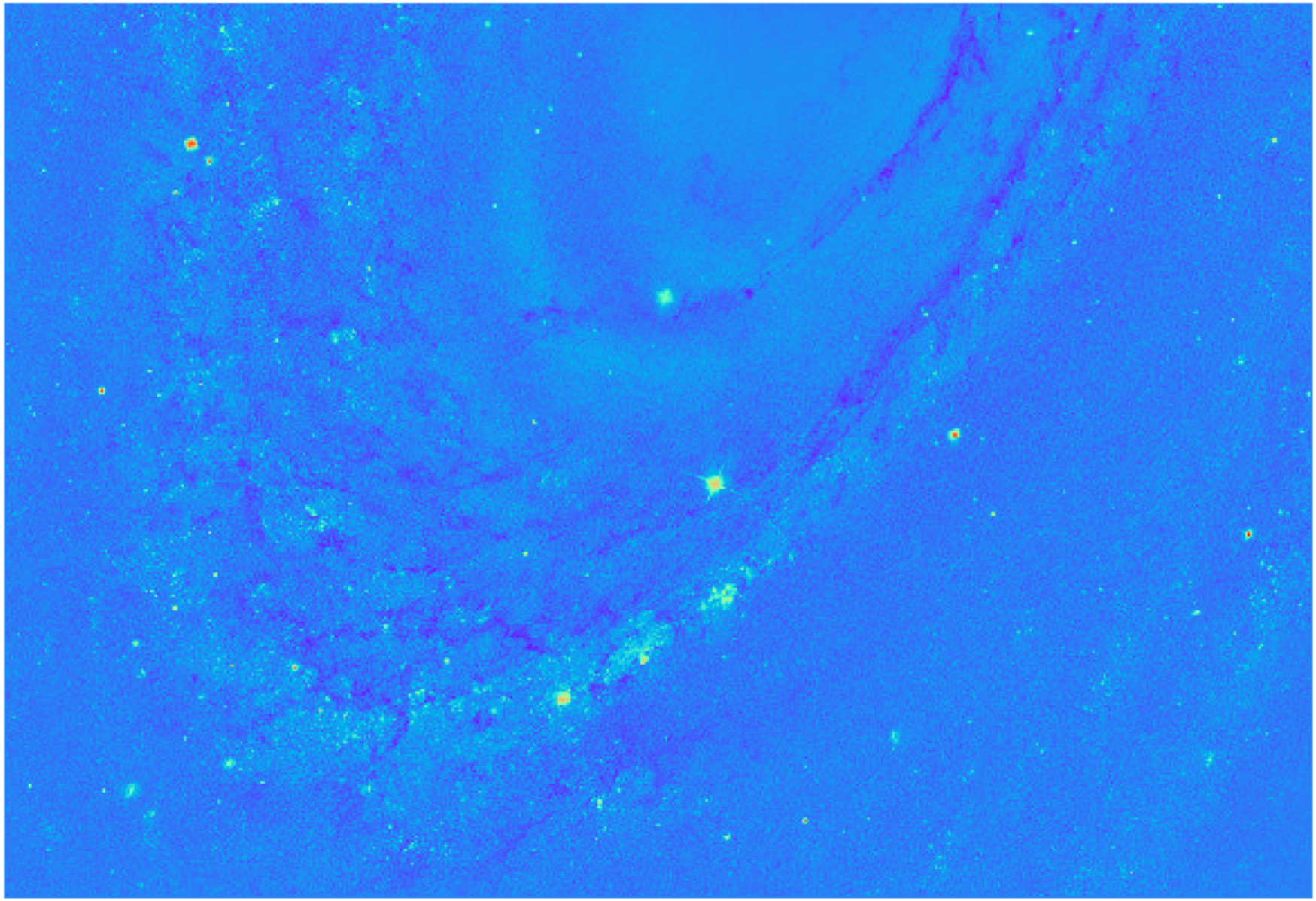


Or the same subtraction in more detailed view





=



Running a code after compilation of blurring algorithm with Intel ifort compiler.
Fortran code is written in double precision (dp), Python is ver. 3.

```
art-1[174]:~/progD57$ ifor-laplace3-dp.x
t=  2.797E-04 s      3654 fps,  val=  0.2769022      host OMP
t=  2.691E-04 s      3716 fps,  val=  0.2769022      host OMP
t=  3.757E-04 s      2661 fps,  val=  0.2769022      host OMP i
t=  3.264E-04 s      3063 fps,  val=  0.2769022      host slices
t=  2.532E-04 s      3949 fps,  val=  0.2769022      host slices 1
t=  9.323E-04 s      1072 fps,  val=  0.2769022      host squares
t=  8.695E-04 s      1150 fps,  val=  0.2769022      host
t=  3.720E-03 s       268 fps,  Numpy (written in C or F)
t=  0.1562400 s       6.4 fps,  plain Python
```

Time t is the execution of one Laplacian blurring pass on array of 1 M pixels (1 Mpix).
The corresponding number of sweeps per second is given as fps = frames per sec.
„val“ is one pixel value printed to make sure all programs produce the same result.

The task is done extremely slowly by 2 nested Python loops (6.4 Mpix/s), 42 times faster by NumPy methods of vectorized array operations (268 Mpix/s),
& even much faster on CPU in Fortran (different methods mentioned in the rightmost column).

This performance is great, but is beaten by single precision versions of the program and by CUDA (GPU versions). Graphics, by the way, does not need double precision.

Example compilation with pgf95 PGI compiler. Code in double precision (dp) run on CPU = i7 6 cores, 4GHz overclock, and Nvidia GTX 1080ti graphics card.

```
art-1[177]:~/progD57$ nvidia-smi –p 1  (it's good to make sure that the
    so-called persistence mode of the Nvidia driver is on, or switch it
    on as above)
art-1[177]:~/progD57$ cudafor-laplace3-dp.x
t=   5.7930E-05 s      17261 fps,  val=   0.3449662 CUDA kernel 0 on GPU
t=   5.7928E-05 s      17262 fps,  val=   0.3449662 CUDA kernel 1 on GPU
t=   3.1855E-04 s       3139 fps,  val=   0.3449662 host OMP slices
t=   3.7507E-04 s       2666 fps,  val=   0.3449662 host OMP
t=   8.8273E-04 s       1132 fps,  val=   0.3449662 host
t=   3.7199E-03 s        268 fps,  Numpy dp
t=   0.1562400  s        6.4 fps,  Python out of the box
```

Time t of one sweep of Laplacian blurring operator on array of 1 M pixels in sec, and the corresponding fps (frames/sec) are shown (equivalent to Mpix/s)

Results show that this problem is solved extremely fast by PGI CUDA Fortran (~1000x faster than Python, and 23x faster than Numpy)

This can only be beaten by a single precision CUDA run....

Example compilation with pgf95 PGI compiler. Code in single precision (sp) runs on: CPU host = i7 (6 cores), 4GHz clock, and on Nvidia GTX 1080ti GPU.

```
art-2[183]:~/progD57$  cudafor-laplace3-sp.x
 t=   3.4939E-05        28621 fps,  val=   0.3449661 CUDA kernel 0
 t=   3.4926E-05        28631 fps,  val=   0.3449661 CUDA kernel 1
 t=   3.4950E-05        28612 fps,  val=   0.3449661 CPU, OMP slices
 t=   1.0031E-04         9963 fps,  val=   0.3449661 CPU, OMP
 t=   3.6699E-04         3756 fps,  val=   0.3449661 CPU host
 t=   3.7199E-03          268 fps,  Numpy (dp)
 t=   1.5624E-01          6.4 fps,  Python3 loops
```

Maximum speed (sp) of Laplacian blurring is 29k fps, or equivalently 29 Gpixels/s. We are near the resolution of the timer, so I don‘t actually trust these precise numbers. But the order of magnitude is correct. If each pixel requires about 10 FLOP (dp floating-point operations) to be updated using values from its neighborhood, then CUDA Fortran reaches 0.3 TFLOPs ~ 0.3e12 FLOPS.

This is an impressive speed, although lower than ~7 TFLOPs, of which the 1080ti card is actually capable in applications that do not rely (as this code does) on the GPU-RAM bandwidth.

On the same server, the sp calculation can be done ~4500x faster on GPU than in (dp) Python on CPU, and ~100x faster than on the same CPU in Numpy.

This record can further be beaten by parallel computation on a multi-node cluster, especially with a faster hardware. (But the best hardware, CPUs and GPUs, in 2024 was no more than about 3 times faster than we have on art-1.)

We have proven that PYTHON IS NOT AN HPC LANGUAGE. That‘s ok, since it was never meant for large computations.

This is also one of the main reasons why you take this course: to go beyond Python in programming, as well as encounter new algorithms, methods, and Linux shells.