PHY604 Lecture 3

September 5, 2023

Review: Roundoff and truncation error

- Consider computing exp(-24) via a truncated Taylor series:

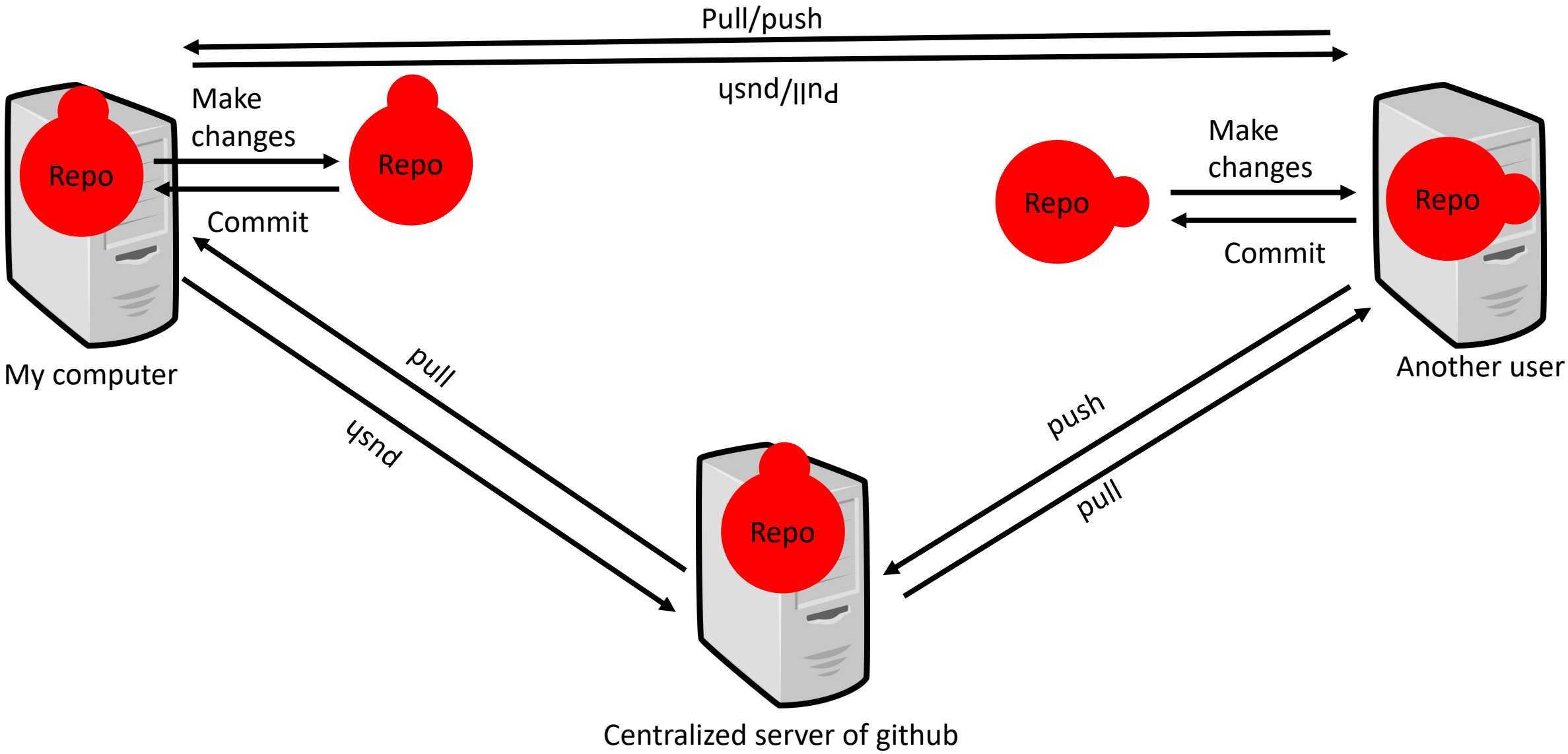
$$e^x \simeq S(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + ... + \frac{x^n}{n!}$$

- Error in the approximation (**i.e., truncation error**) is less than:

$$\frac{|x|^{n+1}}{(n+1)!} \max\{1, e^x\}$$

- But if we compute S(-24) by adding terms until they are less than machine precision (8 byte):
  - S(-24)=3.7814382919759864E-007
  - Exp(-24)=3.7751345442790977E-011
  - Error is larger than the result (much larger than truncation error)!!
  - Looking at terms, we see we are relying on cancellations of terms

Review: Distributed version control with Git



Today’s lecture:

- Good programming practices:
  - Finish discussing version control
  - Testing
  - Misc. good practices
- Numerical differentiation

Example: Simple remote on group server

- We'll look at the example of having people work with a shared remote repository—this is common with groups.
  - Each developer will have their own clone that they interact with, develop in, branch for experimentation, etc.
  - You can push and pull to/from the remote repo to stay in sync with others
  - You probably want to put everyone in the same UNIX group on the server
- Creating a master bare repo:

```
git init --bare --shared myproject.git
chgrp -R groupname myproject.git
```

(to set permissions)

Example: Simple remote on group server

- This repo is empty, and bare—it will only contain the git files, not the actual source files you want to work on
- Each user should clone it
  - In some other directory. User A does:
    - **`git clone /path/to/myproject.git`**
- Now you can operate on it
  - Create a file (README)
  - Add it to your repo: `git add README`
  - Commit it to your repo: `git commit README`
  - Push it back to the bare repo: `git push`
- Note that for each commit you will be prompted to add a log message detailing the change

Example: Simple remote on group server

- Now user B comes along and wants to play too:
- In some other directory. User B does:
  - **`git clone /path/to/myrepo.git`**
- Note that they already have the README file
  - Edit README
  - Commit you changes locally: **`git commit README`**
  - Push it back to the bare repo: **`git push`**
- Now user A can get this changes by doing: **`git pull`**
- In general, you can push to a *bare repo*, but you can pull from anyone

Using github

- Don't want to use your own server? Use github or bitbucket
  - Free for public (open source) projects
  - Pay for private projects
- How to contribute to someone else's project?
  - Since you are not a member of that project, you cannot push back to it
    - You don't have write access
  - Use pull requests:
    - Fork the project into your own account
    - Push back to your fork
    - Issue a *pull-request* asking for your changes to be incorporated

Common Git commands available using `git --help`

```
/Users/cdreyer % git --help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects
```

Some last comments about git and github

- If you put the remote repository on a different server, then you always have a backup of your project
  - Since git is distributed, if your remote server dies, each clone is a backup of the entire repo, so you are safe both ways.
- Free (for open source), online, web-based hosting sites exist (e.g. Github)
- Best with Linux or Mac OS (in terminal).
  - Windows? Try: https://git-for-windows.github.io/
- Github provides tools to share your code broadly and engage with your community
  - Pull requests, issue tracking, etc.

- We'll use git to hand in our homework assignments (more on this later)

Today’s lecture:

- Good programming practices:
  - Finish discussing version control
  - Testing
  - Misc. good practices
- Numerical differentiation

Testing

- Testing is obviously a crucial part of writing programs

- When programs get complicated, testing is not so straight forward:
  - How do I know that a change to one part didn't break another part?
  - How do I know what I did will work on different architectures?
  - My code crashes after running for 78 hours, where did the error originate from?

- Testing involves running the program or part of the program with some inputs and determining if the outputs are those that are expected (or at least consistent)

- Many types of testing. We will discuss unit testing and regression testing

Unit testing

- Unit testing is the practice in which each smallest, self-contained unit of the code is tested independently of the others

- There are unit testing frameworks out there that help automate the procedure for different codes
  - E.g., **`unittest`** for python

Another simple example: Matrix inversion

- Say your code has a matrix inversion routine that computes A^{-1}

- A unit test for this routine can be:
  - Pick a vector x
  - Compute b = A x
  - Compute x = A^{-1} b
  - Does the x you get match (to machine tol) the original x?

Regression Testing

- Imagine you've "perfected" your program (simulation tool, analysis tool, etc.)
  - You are confident that the answer it gives is "right"
  - You want to make sure that any changes you do in the future do not change the output
  - Regression testing tests whether changes to the code change the solution

- Regression testing:
  - Store a copy of the current output (a benchmark)
  - Make some changes to the code
  - Compare the new solution to the previous solution
  - If the answers differ, either:
    - You've introduced a bug → fix it
    - You've fixed a bug → update your benchmark

Regression testing

- Simplest requirements:
  - You just need a tool to compare the current output to benchmark
  - You can build up a more complex system from here with simple scripting

- Big codes need a bunch of tests to exercise all possible options for the code
  - If you spend a lot of time hunting down a bug, once you fix it, put a test case in your suite to check that case
  - If someone implements a new functionality, ask them to submit a test
  - You'll never have complete coverage, but your number of tests will grow with time, experience, and code complexity

Today's lecture:

- Good programming practices:
  - Finish discussing version control
  - Testing
  - Misc. good practices
- Numerical differentiation

Comments and Documentation

- Many in computer science will say that “good code documents itself”
  - **Do not believe it.**
  - Remember, we are often writing code for programming novices (both the developers and users)
  - The better people can understand your code, the more productive science will be done with it

- No hard-and-fast rules. Comments should explain the basic idea of what a block of code does
  - Only comment “single lines” if there is something special or unusual about them
  - Keep comments up to date with the code
  - Think about what information will be useful for you in the future, and other developers of your code

- Can often use tools to turn comments in the source into external documentation
  - Robodoc: https://rfsber.home.xs4all.nl/Robo/
  - FORD: http://fortranwiki.org/fortran/show/FORD
  - Pydoc: https://docs.python.org/3/library/pydoc.html
  - Others for python: https://wiki.python.org/moin/DocumentationTools

Debugging tools

- Simplest debugging: print out information at intermediate points in code execution
- Running with appropriate compiler glags (e.g., `-g` for gnu compilers) can provide debugging information
  - Can make code run slower, but useful for test purposes
- Interactive debuggers let you step through your code line-by-line, inspect the values of variables as they are set, etc.
  - gdb is the version that works with the GNU compilers. Some graphical frontends exist.
  - Lots of examples online
  - Not very useful for parallel code.
- Particularly difficult errors to find often involve memory management
  - **Valgrind** is an automated tool for finding memory leaks. No source code modifications are necessary.

DEMO valgrind

- Ssh –Y rusty
- Cd ~/teaching/PHY604_Fall2021/Lecture2_demos/cxx_val
- Emacs bounds
- g++ -o bounds bounds.cpp
- ./bounds
- Valgrind ./bounds
  - Finds leak, but not sure where
- g++ -o bounds bounds.cpp -g
- valgrind ./bounds
  - Now we can see leak is
- Uncomment std::cout…
- g++ -o bounds bounds.cpp -g
- valgrind ./bounds
  - Uninitialized error

Building your code with, e.g., Makefiles

- It is good style to separate your subroutines/functions into files, grouped together by purpose
  - Makes a project easier to manage (for you and version control)
  - Reduces compiler memory needs (although, can prevent inlining across files)
  - Reduces compile time—you only need to recompile the code that changed (and anything that might depend on it)
- Makefiles automate the process of building your code
  - No ambiguity of whether your executable is up-to-date with your changes
  - Only recompiles the code that changed (looks at dates)
  - Very flexible: lots of rules allow you to customize how to build, etc.
  - Written to take into account dependencies

We have not really discussed general coding style

- Depends very much on the language, and is often a matter of opinion (google it)
- Some general rules:
  - 1. Use a consistent programming style
  - 2. Use brief but descriptive variable and function names
  - 3. Avoid “magic numbers”
    - Name your constants, specify your flags
  - 4. Use functions and/or subroutines for repetitive tasks
  - 5. Check return values for errors before proceeding
  - 6. Share information effectively (e.g., using modules or namespaces)
  - 7. Limit the scope of your variables, methods, etc.
  - 8. Think carefully about the most effective way to input and output data
  - 9. Be careful about memory, i.e., allocating and deallocating
  - 10. Make your code readable and portable, you will thank yourself (or your collaborators will thank you) later.

Today's lecture:

- Good programming practices:
  - Finish discussing version control
  - Testing
  - Misc. good practices
- Numerical differentiation

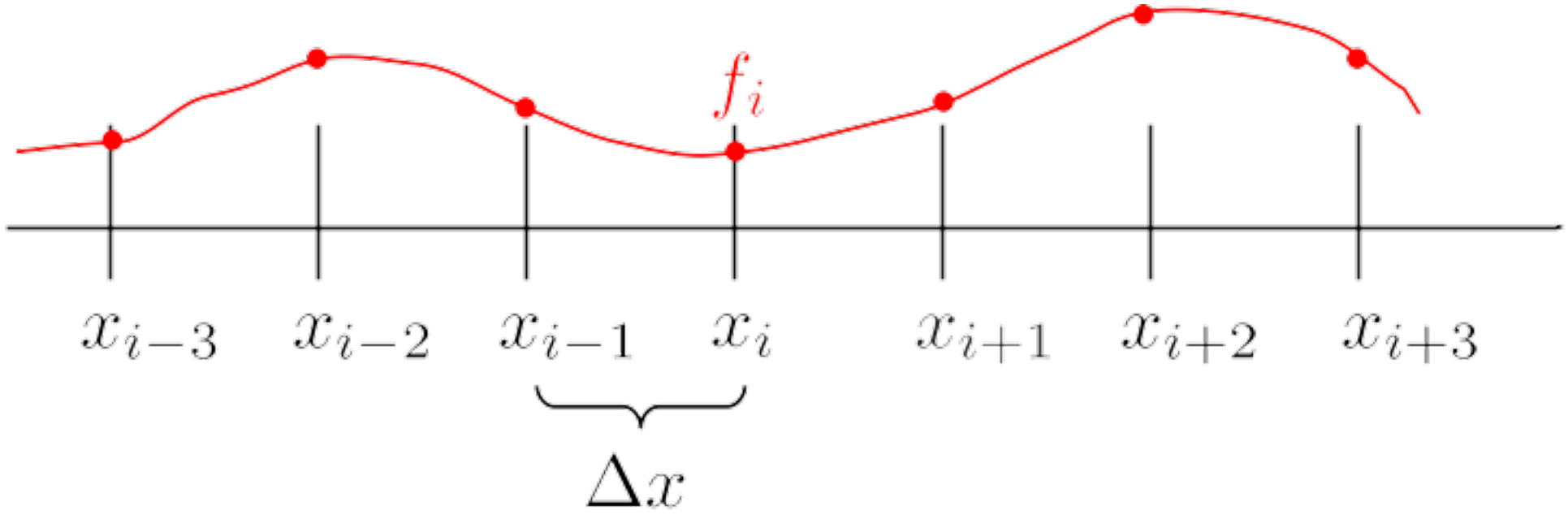
Numerical differentiation, Two situations:

- We have data defined only at a set of (possibly regularly spaced) points
  - Generally speaking, asking for greater accuracy for the derivative involves using more of the discrete points

- We have an analytic expression for f(x) and want to compute the derivative numerically
  - If possible, it would be better to take the analytic derivative of f(x), but we can learn something about error estimation in this case.
  - Used, for example, in computing the numerical Jacobian for integrating a system of ODEs (we'll see this later)

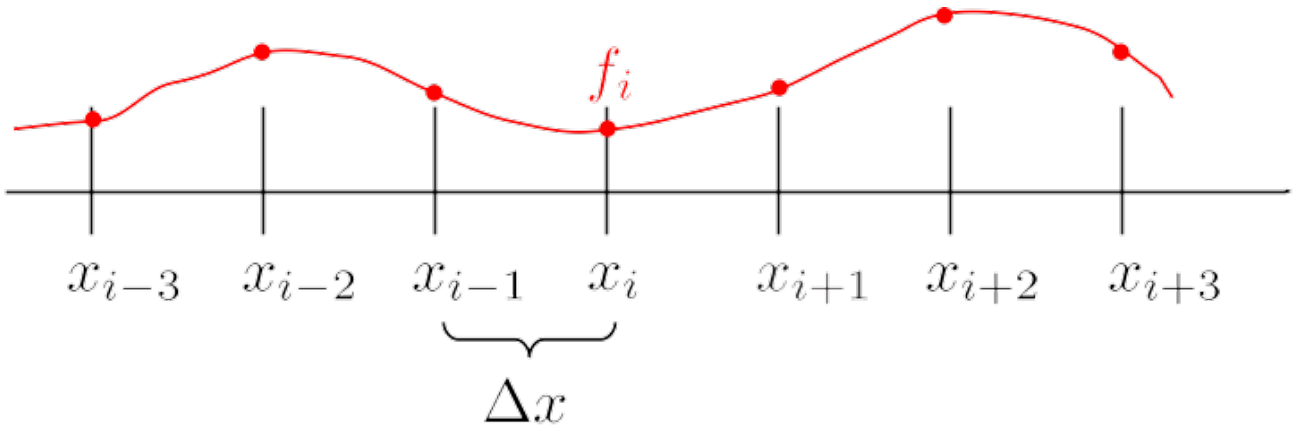
OTB: Gridded data

- Discretized data is represented at a finite number of locations
  - Integer subscripts are used to denote the position (index) on the grid
  - Structured/regular: spacing is constant



- Data is known only at the grid points: $f_i = f(x_i)$

OTB: First derivative



- Taylor expansion:

$$f_{i+1} = f(x_i + \Delta x) = f_i + \left.\frac{df}{dx}\right|_{x_i} \Delta x + \frac{1}{2}\left.\frac{d^2 f}{dx^2}\right|_{x_i} \Delta x^2 + \ldots$$

- Solve for the first derivative:

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f_{i+1} - f_i}{\Delta x} - \frac{1}{2}\left.\frac{d^2 f}{dx^2}\right|_{x_i} \Delta x$$

Discrete approx. of f'

Leading term in the truncation error

OTB: Order of accuracy

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f_{i+1} - f_i}{\Delta x} - \left.\frac{1}{2}\frac{d^2 f}{dx^2}\right|_{x_i} \Delta x$$

- The accuracy of the finite difference approximation is determined by size of Δx
- So this finite difference expression is accurate to "order" Δx: $\mathcal{O}(\Delta x)$

- However: Making Δx small means that we are subtracting numbers that are very close to each other, which can result in significant rounding errors

OTB Maximizing the accuracy

- Say we can evaluate the function to accuracy *C f(x)* [also *C f*(*x*+Δ*x*)]
  - For double precision: $C \simeq 10^{-16}$
- Worst-case rounding error on derivative is 2*C*|*f*(*x*)|/ Δ*x*
  - Also need to worry about associative errors: $(x + \Delta x) - x \stackrel{?}{=} \Delta x$
- So total error is: $\left| \left.\frac{df}{dx}\right|_{x_i} - \frac{f_{i+1} - f_i}{\Delta x} \right| \leq \left. \frac{1}{2}\frac{d^2 f}{dx^2} \right|_{x_i} \Delta x + \frac{2C|f_i|}{\Delta x}$
- We can minimize to find: $\Delta x = \sqrt{4C \left| \frac{f_i}{f_i''} \right|} \sim 10^{-8}$
- So “minimum” error: $\epsilon = \sqrt{4C\,|f_i f_i''|} \sim 10^{-8}$

Increasing accuracy with more points in the "stencil"

- First-order "forward" or "backward":

$$f' = \frac{f_{i+1} - f_i}{\Delta x} \qquad f' = \frac{f_i - f_{i-1}}{\Delta x}$$

2-point stencil

- Second-order "central":

$$f' = \frac{-\frac{1}{2} f_{i-1} + 0 f_i + \frac{1}{2} f_{i+1}}{\Delta x}$$

3-point stencil

Second-order central

- Consider two Taylor expansions:

$$f_{i+1} = f_i + \left.\frac{df}{dx}\right|_{x_i} \Delta x + \frac{1}{2}\left.\frac{d^2 f}{dx^2}\right|_{x_i} \Delta x^2 + ...$$

$$f_{i-1} = f_i - \left.\frac{df}{dx}\right|_{x_i} \Delta x + \frac{1}{2}\left.\frac{d^2 f}{dx^2}\right|_{x_i} \Delta x^2 + ...$$

- We see that:

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f_{i+1} - f_{i-1}}{2\Delta x} + \mathcal{O}(\Delta x^2) + ...$$

Error in Second order central

$$\left| \left. \frac{df}{dx} \right|_{x_i} - \frac{f_{i+1} - f_{i-1}}{2\Delta x} \right| \leq \left. \frac{1}{6} \frac{d^3 f}{dx^3} \right|_{x_i} \Delta x^2 + \frac{C|f_i|}{\Delta x}$$

- Minimize WRT Δx: $\Delta x = \sqrt[3]{6C \left| \frac{f(x_i)}{f'''(x_i)} \right|} \sim 10^{-5}$

Assuming double prec.

- Minimum error: $\epsilon \propto \sqrt[3]{C^2 f(x_i)^2 |f'''(x_i)|} \sim 10^{-11}$

Assuming double prec.

Higher order first derivatives

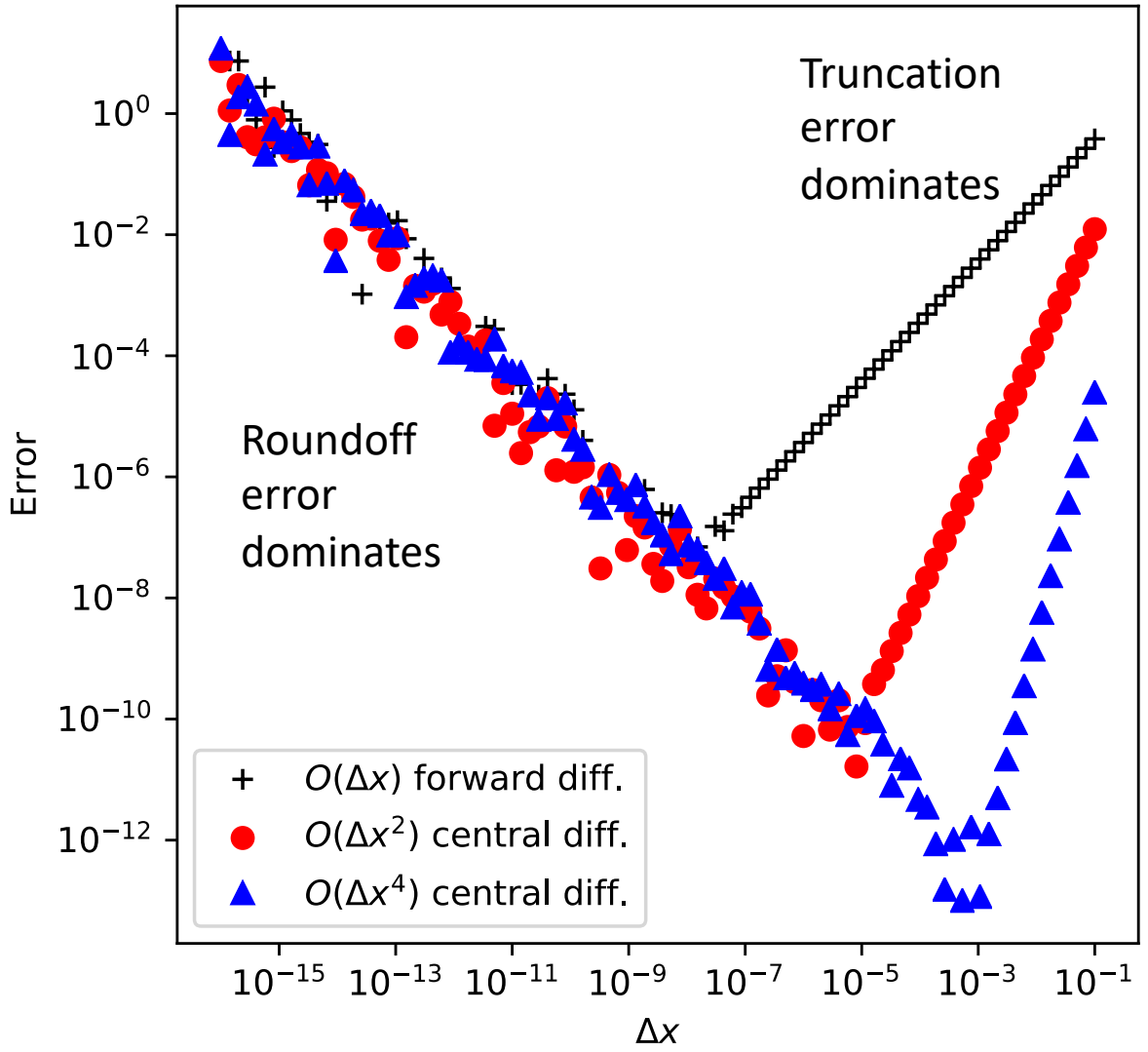
- To get accuracy to order *n* [i.e., $\mathcal{O}(\Delta x^n)$] follow a similar strategy:
  - 1. Write down Taylor expansion for *n+1* finite difference points up to order *n*+1
  - 2. Solve set of polynomial equation in Δx for *f'*
  - 3. Obtain an expression involving weighted sum of function evaluated at *n+1* points (some weights may be zero)
- Note: may be central, forward, or backward
- For example, for central:

Derivative	Accuracy	–5	–4	–3	–2	–1	0	1	2	3	4	5
1	2					–1/2	0	1/2				
	4				1/12	–2/3	0	2/3	–1/12			
	6			–1/60	3/20	–3/4	0	3/4	–3/20	1/60		
	8		1/280	–4/105	1/5	–4/5	0	4/5	–1/5	4/105	–1/280	

https://en.wikipedia.org/wiki/Finite_difference_coefficient

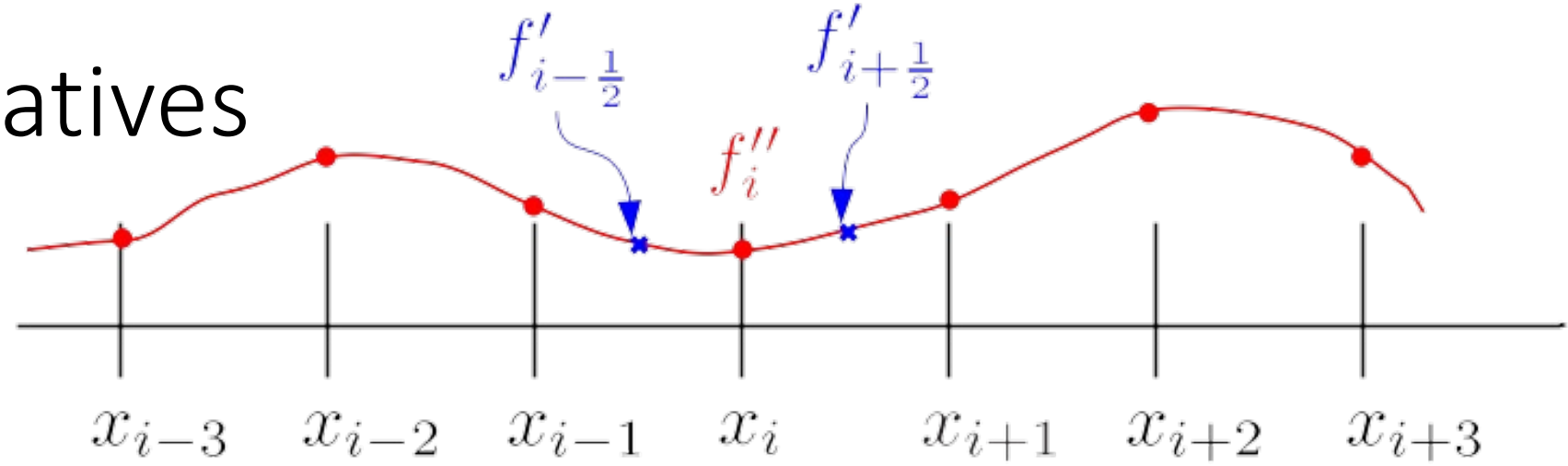
D

Example: Derivative of exp(x)



D

OTB: Higher derivatives



- Write second derivative as: $$f''_i = \frac{f'_{i+1/2} - f'_{i-1/2}}{\Delta x}$$
- Insert central difference first derivatives, e.g.: $f'_i = \frac{f_{i+1} - f_i}{\Delta x}$
- So we get: $$f''_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2}$$

Higher derivatives and error

- We can also use the Taylor expansion strategy:

$$f_{i+1} = f_i + \Delta x f_i' + \frac{1}{2}\Delta x^2 f_i'' + \frac{1}{6}\Delta x^3 f_i''' + \frac{1}{24}\Delta x^4 f_i'''' + \ldots$$

$$f_{i-1} = f_i - \Delta x f_i' + \frac{1}{2}\Delta x^2 f_i'' - \frac{1}{6}\Delta x^3 f_i''' + \frac{1}{24}\Delta x^4 f_i'''' + \ldots$$

- Add together and rearrange: $f_i'' = \dfrac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2} - \dfrac{1}{12}\Delta x^2 f_i''''$

- Error: $\epsilon = \sqrt{\dfrac{4}{3} C |f_i f_i''''|} \sim 10^{-8}$

Assuming double prec.

Partial and mixed derivatives

- Partial derivatives are a simple generalization
- E.g., central differences for function of two variables *f*(*x*,*y*)

$$\frac{\partial f}{\partial x} = \frac{f(x+\Delta x, y) - f(x-\Delta x, y)}{2\Delta x} \qquad \frac{\partial f}{\partial y} = \frac{f(x, y+\Delta y) - f(x, y-\Delta y)}{2\Delta y}$$

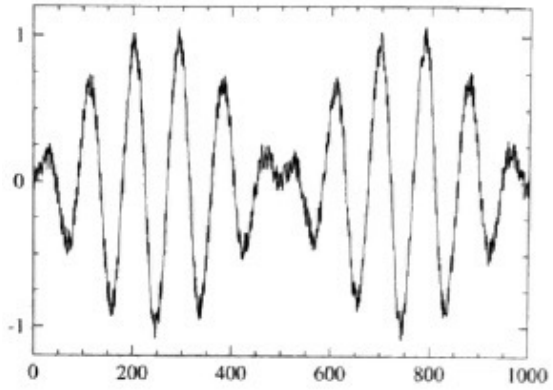
- Mixed second derivative:

$$\frac{\partial^2 f}{\partial y \partial x} = \frac{f(x+\Delta x, y+\Delta y) - f(x-\Delta x, y+\Delta y) - f(x+\Delta x, y-\Delta y) + f(x-\Delta x, y-\Delta y)}{4\Delta x \Delta y}$$

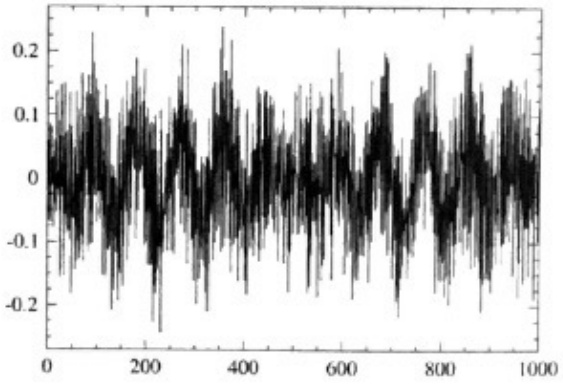
Some final comments on numerical derivation

- Taking derivatives of noisy data makes the noise much worse!
  - Fit to a smooth curve and take the derivative of that
  - Smooth the data, e.g., with a Fourier transform
- We can treat data on uneven grids with the same strategy as before, taking into account the different Δx's between points

Noisy data



Derivative



(Newman)

After class tasks

- If you do not already have one, make an account on github: https://github.com/

- Readings:
  - Wikipedia artical on makefiles
  - Fortran best practices
  - Good Enough Practices in Scientific Computing
  - Blog on numerical differentiation
  - Wikipedia page of finite difference coefficients
  - Newman Chapter 5
  - Garcia Section 10.2