Computational Physics (6810): Session 4

Dick Furnstahl
Nuclear Theory Group
OSU Physics Department

January 27, 2017

PS#1, Session 2 and 3 follow-ups

- Continue Session 3 through "Finding the Approximation . . . " (you'll do the rest in PS#2). PS#4 is last "pre-physics" session.
- Problem set comments:
  - Some minor BuckeyeBox/Dropbox issues, but ok now.
  - Comments will be added to your BuckeyeBox/Dropbox folder.
  - Need to *explain* for homework $1 + \frac{1}{2} + \cdots + \frac{1}{N}$ versus $\frac{1}{N} + \frac{1}{N-1} + \cdots + 1$; why is this like the example from class of $1 + a + \cdots + a$ versus $a + a + \cdots + 1$?
    E.g., is this an example of subtractive cancellation?
  - Graphs *must* be labeled, both axes and different curves. "Column 1" and "Column 2" are *not* adequate labels.
  - You don't need to *explain* the slopes for the Bessel function problem but you should understand the explanation in the Session 4 notes.
- Remember: You can always upgrade a check to a plus.
- It's always ok to hold on to Session guides and continue outside class.

PS#1 follow-up: Alternative ways to code a loop

"Best" way to code $S^{(\text{up})} = \sum_{n=1}^{N} \frac{1}{n}$ and $S^{(\text{down})} = \sum_{n=N}^{1} \frac{1}{n}$?

$\Longrightarrow$ *To minimize logical errors and to make it easy to detect typos*, write your code to mimic the original equations (and rewrite for optimization later)

[Note: when you rewrite code, save previous versions (by hand or version control).]

PS#1 follow-up: Alternative ways to code a loop

"Best" way to code $S^{(\mathrm{up})} = \sum_{n=1}^{N} \frac{1}{n}$ and $S^{(\mathrm{down})} = \sum_{n=N}^{1} \frac{1}{n}$?

$\Longrightarrow$ *To minimize logical errors and to make it easy to detect typos*, write your code to mimic the original equations (and rewrite for optimization later)

[Note: when you rewrite code, save previous versions (by hand or version control).]

1. Choosing variables: Is it best to match precisely, e.g., `N` and `n`?
   Recommendations: `Sup` or `S_up`; $N \rightarrow$ `N_max` (or similar);
   `i,j,n` are *usually* ok for dummy indices (e.g., in sums). Exceptions?

PS#1 follow-up: Alternative ways to code a loop

"Best" way to code $S^{(\mathrm{up})} = \sum_{n=1}^{N} \frac{1}{n}$ and $S^{(\mathrm{down})} = \sum_{n=N}^{1} \frac{1}{n}$?

$\Longrightarrow$ *To minimize logical errors and to make it easy to detect typos*, write your code to mimic the original equations (and rewrite for optimization later)

[Note: when you rewrite code, save previous versions (by hand or version control).]

1. Choosing variables: Is it best to match precisely, e.g., `N` and `n`?
   Recommendations: `Sup` or `S_up`; $N \rightarrow$ `N_max` (or similar);
   `i,j,n` are *usually* ok for dummy indices (e.g., in sums). Exceptions?
2. Implementing the loop: Choices and how do you verify?

```
double S_up = 0, S_down = 0;
for (int n=0; i < N_max; n++)
{
  S_up += 1./float(n+1);
  S_down += 1./float(Nmax - n);
}
```

```
double S_up = 0, S_down = 0;
for (int n=1; n <= N_max; n++)
{
  S_up += 1./float(n);
}
for (int n=N_max; n >= 1; n--)
{
  S_down += 1./float(n);
}
```

Session 2 and 3 follow-ups (cont.)

- Why does $\underbrace{5 \times 10^{-8} + 5 \times 10^{-8} + \cdots}_{10^8 \text{ times}} + 1 = 2$?
  - Recall that $1 + 5 \times 10^{-8} = 1$ because it falls off the end:

$$\begin{array}{r} 1.000\cdots00 \\ +0.000\cdots0011010\cdots \\ \hline 1.000\cdots00 \end{array}$$

  - So what do we have after (roughly) 2×10^7 additions? And then what do the rest of the additions do?
- Reminder: it is recommended to use a *plot file* for gnuplot
  - Type your plotting commands in a file instead of at `gnuplot>`
  - Save after changes (but keep the editor open)
  - `gnuplot> load "filename.plt"` after every change
- Compare the *relative* difference to *expected* precision (which is not usually as small as the machine precision):

```
 scale = ( abs(x1) + abs(x1) )/2.     // set the scale
if ( abs(x1-x2) < epsilon*scale ) { < do something> }
```

Round-off versus Approximation errors

- Round-off errors from finite # of *significant figures*
  - about 7 for single precision (float)
  - about 15–16 for double precision (double)
- Sources of round-off errors
  - adding or subtracting very different size numbers, e.g., $(1 + a + a + \cdots)$
  - subtracting two similar numbers ("subtractive cancellation")
- Approximation error for a central derivative

$$\left.\frac{df}{dx}\right|_{x=x_0} \approx \frac{f(x_0 + h/2) - f(x_0 - h/2)}{h} = f'(x_0) + \frac{1}{24}h^2 f^{(3)}(x_0) + \cdots$$

  - Why is this better than forward difference?
    - How many function evaluations?
    - How does the error compare to forward difference?
    - Can you *explain* why it is better?
    - What should the error plot look like?

Numerical integration (or "quadrature")

- General form for approximation of integral:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{N} f(x_i) w_i$$

  - the N x_i's are "nodes" and w_i's are "weights"
  - evenly spaced x_i's $\Longrightarrow$ "Newton-Cotes methods" (see notes)
  - unevenly spaced $\Longrightarrow$ Gaussian quadrature (see Hjorth-Jensen notes)

- Error analysis for trapezoid and Simpson's rule in notes
- Trapezoid weights (Session 3 Table) $\Longrightarrow \frac{1}{2}(h, h)$

$$\int_a^b f(x)\,dx \approx \frac{f_1 + f_2}{2} \cdot h + \frac{f_2 + f_3}{2} \cdot h + \cdots \frac{f_{N-1} + f_N}{2} \cdot h$$

$$\approx \frac{h}{2}[f_1 + 2f_2 + 2f_3 + \cdots + 2f_{N-1} + f_N]$$

Numerical integration (cont.)

- Simpson's rule weights (Session 3 Table) $\Longrightarrow \frac{1}{3}(h, 4h, h)$

$$\int_a^b f(x)\,dx \approx \frac{f_1 + 4f_2 + f_3}{3} \cdot h + \frac{f_3 + 4f_4 + f_5}{3} \cdot h + \cdots \frac{f_{N-2} + 4f_{N-1} + f_N}{3} \cdot h$$

$$\approx \frac{h}{3}[f_1 + 4f_2 + 2f_3 + 4f_4 + \cdots + 2f_{N-2} + 4f_{N-1} + f_N]$$

- Need N to be odd — minimum is 3, then 5, 7, 9, ... $\Longrightarrow (2n + 1)$
- For 3/8ths rule, need N to be 4, 7, 10, ... $\Longrightarrow (3n + 1)$
- If you use the wrong number of points, the answer will not be as good as it should be!

Structures by example (see notes for intro)

- A structure is a precursor to classes. Use it to group info together in a new data type. *Why are structures and functions useful?*
- There are two ways to define; we'll use the first:

```
typedef struct
{
  double a;
  double b;
  double c;
  int num;
} osu_parameters;
```

```
struct parameters
{
  double a;
  double b;
  double c;
  int num;
};
```

- In use:

```
osu_parameters my_coefficients;
my_coefficients.a = 1.1;
my_coefficients.num = 3;
double a = 2.3;  // **independent** of the structure
```

Pointers (also see the handout and notes)

- Analogy: web pages and URL web addresses
  - send a copy of the content of a page ("pass by value")
  - send them the URL ("pass by reference")
- Pointer to function

```
float my_integrand (float x) {return (exp (-x));}
```

  To call `trapezoid_rule` with `my_integrand` as an argument:

```
result = trapezoid_rule (i, lower, upper, &my_integrand);
```

- Note the ampersand & in front of `my_integrand`. That sends to trapezoid the *address* of the `my_integrand` function (i.e., it sends the location in the computer memory).
- The function `trapezoid_rule` is defined as:

```
float trapezoid_rule ( int num_pts, float x_min,
    float x_max, float (*integrand) (float x) )
```

  The `*` indicates a *pointer*, which holds the address that is passed in this case. For now, just note the translation.

Pointers (cont.)

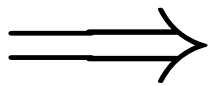
- Examples of declaring and using pointers

```
double alpha; // alpha holds a value
double * beta_ptr; // * means pointer: holds an address

alpha = 3.
beta_ptr = alpha;  // FAILS! (address != value)
beta_ptr = &alpha; // & means "the address of"
gamma = *beta_ptr + 12.1 // * means "dereference" (look up)
```

- void pointer: point to an address, but don't specify what is there (could be an int, double, structure, class)

```
double alpha;
void * params_ptr;
params_ptr = &alpha
```



```
double alpha_fails, alpha_works;
alpha_fails = *params_ptr; \\ error!
alpha_works = *(double *)params_ptr;
```

Pointers (cont.)

- Two ways to recover a struct from a void pointer

```
typedef struct       // define a simple struct
{
  double a;
  double b;
} osu_parameters;

double f_osu_parameters (double x, void *params_ptr)
{
  \\ Here we define a local pointer to an osu_parameters struct
  osu_parameters *passed_ptr;
  passed_ptr = (osu_parameters *) params_ptr;

  double passed_double_1 = ((osu_parameters *) params_ptr)->a;
  double passed_double_2 = passed_ptr->b;  // using local struct

  cout << "Passed:   a = " << passed_double_1
    << ", b = " << passed_double_2
    << endl;
```

Other comments on C++ programming

- When defining parameters that do not change, use `const`:

```
const int Nstart = 100;
const int Nmax = 1e8;
const int Ntimes = 2;
for (N=Nstart; N<=Nmax; N*=Ntimes) {
  <do stuff>
}
```

- Do not *hardcode* values for these! [good programming practice]
- Equal spacing on a log scale
  - If you have `N` points, incrementing `N++` or `N += 10` gives an uneven distribution on a log scale (why??)
  - Consider $N = N_0, aN_0, a^2N_0, a^3N_0 \ldots$ instead
    $\Longrightarrow \log_{10} a^i N = \log_{10} N + i \log 10a \Longrightarrow$ evenly spaced
  - So could use `N*=2` or `N=2*N+1` $\Longrightarrow$ which for Simpson's?

Gnuplot skills you should have (if you don't, learn today!)

- Color! `set term postscript color`
- Using a plot file: `gnuplot> load "filename.plt"`
  - See `gnuplot_plotfile_example.pdf`
  - Operating on columns: use `$1,$2,...` and lots of `()`'s

```
plot 'test.dat' using 1:2
plot 'test.dat' using ($1):($2)
# what does this do?  [Note the ()'s]
plot 'test.dat' using (2*($1)):(($2)**2)
```

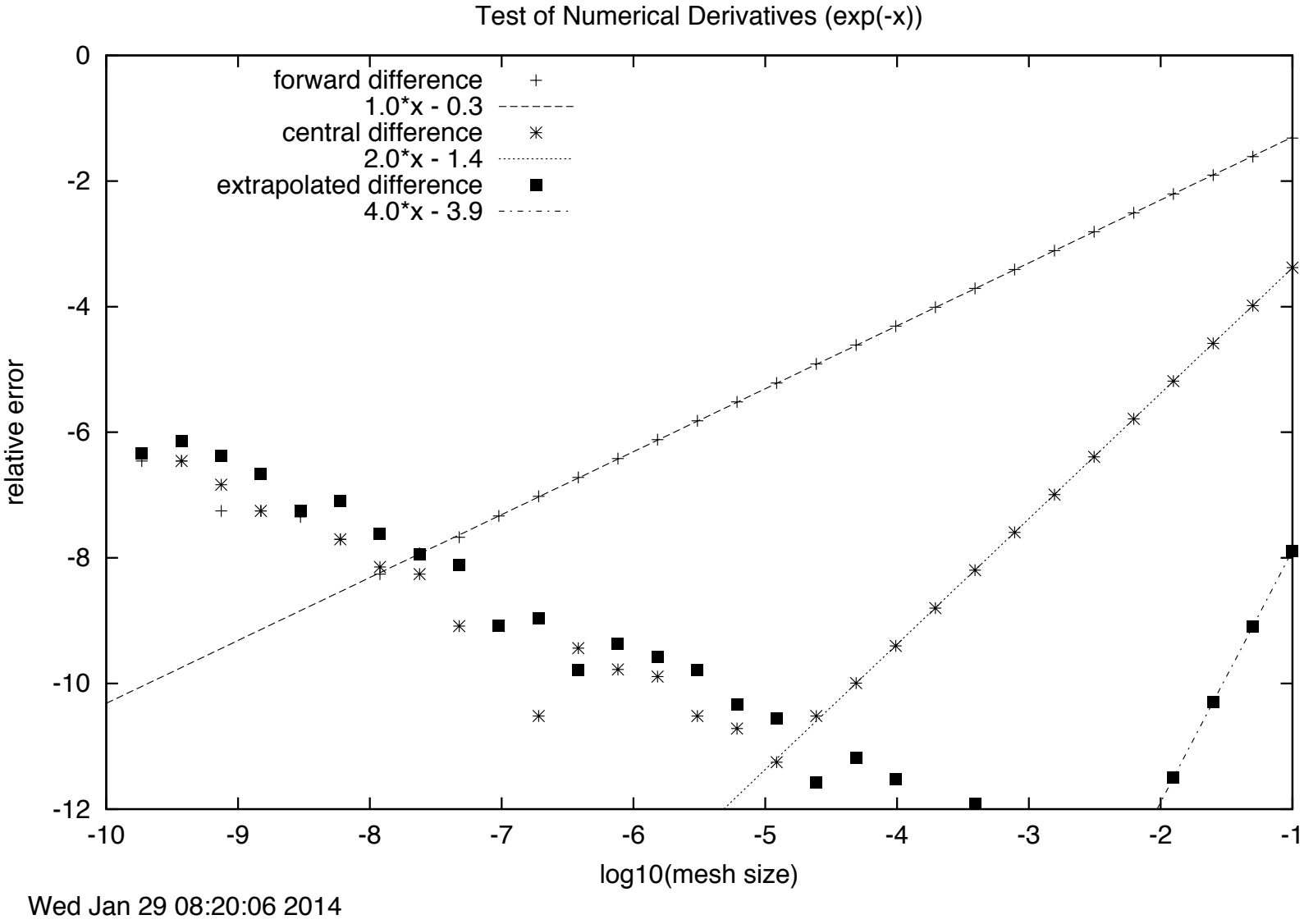
- Fitting lines: over a range, multiple fits, adding fit lines to plot

```
f1(x) = a1*x + b1
fit [-8:-1] f1(x) "derivative_test.dat" using ($1):($2) via a1,b1
fit_title1 = sprintf("%-+4.1f*x %-+4.1f",a1,b1)

f2(x) = a2*x + b2
fit [-5:-1] f2(x) "derivative_test.dat" using ($1):($3) via a2,b2

plot "derivative_test.dat" using ($1):($2), f1(x) title fit_title1
```

Numerical derivatives and Richardson extrapolations



Challenge: explain the left-side slope *and* intercept

Numerical derivatives and Richardson extrapolations

- If you know the precise *form* of the error ...
- Consider the central difference derivative

$$D_c[f,h] \equiv \frac{f(x_0+h/2)-f(x_0-h/2)}{h} = f'(x_0) + \frac{1}{24}h^2 f^{(3)}(x_0) + Ch^4 f^{(5)}(x_0) + \cdots$$

- The only h dependence in the error is explicit. What if $h \to 2h$?

$$D_c[f,2h] = \frac{f(x_0+h)-f(x_0-h)}{2h} = f'(x_0) + \frac{1}{24}4h^2 f^{(3)}(x_0) + 16Ch^4 f^{(5)}(x_0) + \cdots$$

- Richardson: consider the error term as an unknown and eliminate:

$$\begin{aligned}\frac{4D_c[f,h]-D_c[f,2h]}{3} &= \frac{1}{3}\left\{\left[4f'(x_0) + \frac{4}{24}f^{(3)}(x_0)\right] - \left[f'(x_0) + \frac{4}{24}f^{(3)}(x_0)\right]\right\} \\ &= f'(x_0) - 4Ch^4 f^{(5)}(x_0) = f'(x_0) + \mathcal{O}(h^4)\ !\end{aligned}$$

- How can you get rid of the $\mathcal{O}(h^4)$ term?