Numerical Methods and Algorithms in Chemical Physics

Raghunathan Ramakrishnan
ramakrishnan@tifrh.res.in

TIFR Centre for Interdisciplinary Sciences, Hyderabad, India
August-November, 2016

SO, THE GREATEST HACKER OF OUR ERA IS A COOKIE-BAKING MOM?
OH?
SECOND-GREATEST.
MRS. ROBERTS HAD TWO CHILDREN. HER SON, BOBBY, WAS NEVER MUCH FOR COMPUTERS, BUT HER DAUGHTER ELAINE TOOK TO THEM LIKE A RING IN A BELL.
WHEN ELAINE TURNED 11, HER MOTHER SENT HER TO TRAIN UNDER DONALD KNUTH IN HIS MOUNTAIN HIDEAWAY.
FOR FOUR YEARS SHE STUDIED ALGORITHMS.
CHILD—
WHY IS A* SEARCH WRONG IN THIS SITUATION?
SWISH
MEMORY USAGE!
WHAT WOULD YOU USE?
DIJKSTRA'S ALGORITHM!
UNTIL ONE DAY SHE BESTED HER MASTER
SO OUR LOWER BOUND HERE IS O(n log n)
NOPE. GOT IT IN O(n log(log n))
AND LEFT.
Source: https://xkcd.com

Contents

Part - 1, Foundation Topics

1. General Information

The course outline is here http://tinyurl.com/z45prdn

For demo sessions, we will try to set up some PCs in the computer room adjacent to the class room (third floor). However, students are welcome to bring a laptop with linux installed in it. Please consult Mr. Kalyan kumar (kalyankn@tifrh.res.in) if you need any assistance.

1.1 How to pass this course?

- Attend all classes.
- During the lecture, only those content that are essential for understanding the subject will be demonstrated. You are welcome to interrupt the instructor and ask relevant questions, until you are satisfied with explanations.
- Please read, work through, and understand as much content from this handout as possible. The exercises given at the end of each lecture's handout will require some understanding of the handout content.
- Do not memorize all the linux commands, or the programs given in the handout. Instead, try them out in a computer and note down your observations directly in the handout. Practice the commands regularly to make linux and programming a part of your routine activities.
- Assignment files should be submitted via email on or before the deadline.
- For the first programming exam after part-2, you are free to bring a printed copy of the handout, which can include your handwritten notes. You can also bring some additional (handwritten or printed) notes.

1.2 Grading

- Grading: Assignments (50%), Exam-1 after part-2 (25%), Exam-2 after part-3 (25%)

- The above mark distribution can change if you are asked to work on a short project based on this course, during this semester. More details later.

If you spot any mistake/typo, please write to me at raghu.rama.chem@gmail.com. Thank you.

2. Introduction

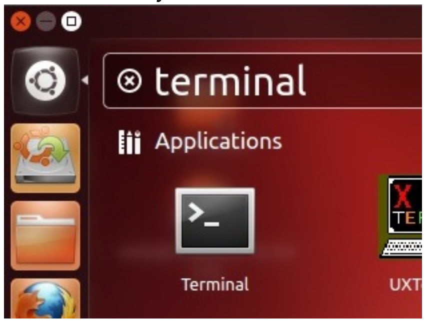
- Computational sciences
- Top-500, and the Top-50
- Coding barrier
- What is this course about ?
- Why Linux + Fortran combo ?

3. Linux

3.1 *Shell and Terminal*

- Shell is a program which enables you to interact with the operating system (e.g. Linux). The shell translates the user-commands received through the user-interface (usually keyboard) to Linux. Some shell programs are `bash`, `csh`, `tcsh`, `ksh`, `zsh`, etc.. We will use `bash` in this course.
- Terminal is a program which enables you to interact with Shell. Some terminal programs are `gnome-terminal`, `konsole`, `xterm`. We will use `gnome-terminal` in this course.
- You may also find using a mouse (with 3-buttons) or touchpad convenient for some tasks.

Here is how you can start `gnome-terminal` in Ubuntu.



3.2 *Commands*

The most commonly used commands in linux are part of the GNU core utilities, a set of utility programs. They are stored (i.e. installed) in the location `/usr/bin`. Here are some commands for you to try in the terminal. You may want to write down some notes in the space next to them.

```
$ pwd
$ date
$ ls
$ echo abc > test
$ ls -l
$ rm test
$ mkdir test
$ rm test
$ rm -r test
$ mkdir test ; rm -r -i test
$ mkdir test ; cd test ; pwd
$ cd ..  ; pwd ; cd test ; pwd
$ echo “Hi”
$ echo “Hi” > file1.txt ; cat file1.txt
$ echo “Hello” >> file1.txt ; cat file1.txt
$ tac file2.txt
$ seq 1 10 >> file2.txt; cat file2.txt
$ seq 1 10 | wc
$ wc file2.txt
$ shuf file2.txt
$ shuf file2.txt | sort
$ shuf file2.txt | sort -r
$ head file2.txt
$ head -2 file2.txt
$ tail -1 file2.txt
$ grep 1 file2.txt
$ grep -v 1 file2.txt
$ seq -w 1 10 | shuf
$ seq -w 1 10 | shuf | sort
$ seq -w 1 10 | shuf | sort -r
$ seq 0 5 50 >> file3.txt; cat file3.txt
$ cat << TheEnd
line 1
line 2
TheEnd
$ factor 100 >> file4.txt; cat file4.txt
$ factor 100 | tee file5.txt
$ diff -q file3.txt file4.txt
$ diff -q file4.txt file5.txt
$ ls
$ ls -l
$ ls -l -S
$ ls -l -S -r
$ mv file1.txt file01.txt; ls
$ cat file*.txt
$ history
$ sleep 2
$ !ca
$ cp file01.txt file01.dat
$ mkdir dir01; cp file01.dat dir01/; ls -l dir01
$ cp dir01 dir02
$ cp -r dir01 dir02
$ cp file01.txt file01.dat dir02/
$ cp dir01/* dir02/
```

- WARNING: `$ rm -r *` will delete all the files in the working directory.
- Files with their names beginning with '.' (dot or period) are hidden. Display them using `$ ls -a`
- File names are case sensitive – file.txt and File.txt are two different files. Try another example and convince yourself.
- Linux does not have the concept of file extension. Extensions become relevant when using a particular software to access/process files.
- As a standard practice, do not use the emply space character in a file name.
- There are many other useful commands like, `find`, `locate`, `paste`, `join`, `split`, etc.. Here is the wiki entry for the full list

  https://en.wikipedia.org/wiki/List_of_Unix_commands
- The command `wc` followed by a filename returns the three integers, corresponding to the number of lines, number of words, and the number of bytes required to store the content of the file. At the end of each line, a new line character is included which takes up 1 byte.
- "here file" is a functionality available in bash which enables passing a series of inputs to a single command (or a script). For instance, try the command `$ awk '{print $1^2}' << x` in your terminal. You can now enter a set of numbers one-by-one followed by 'x' which is the end-of-file character.
- "here string" is a similar functionality which can be used for short inputs. Try `$ awk '{print $1^2}' <<< 1.23`
- Both << and <<< constructs are useful in situations where you may want to execute a command and enter the input directly in the terminal, instead of storing them in a file.

Here are some funny ones for you to try in your free time.

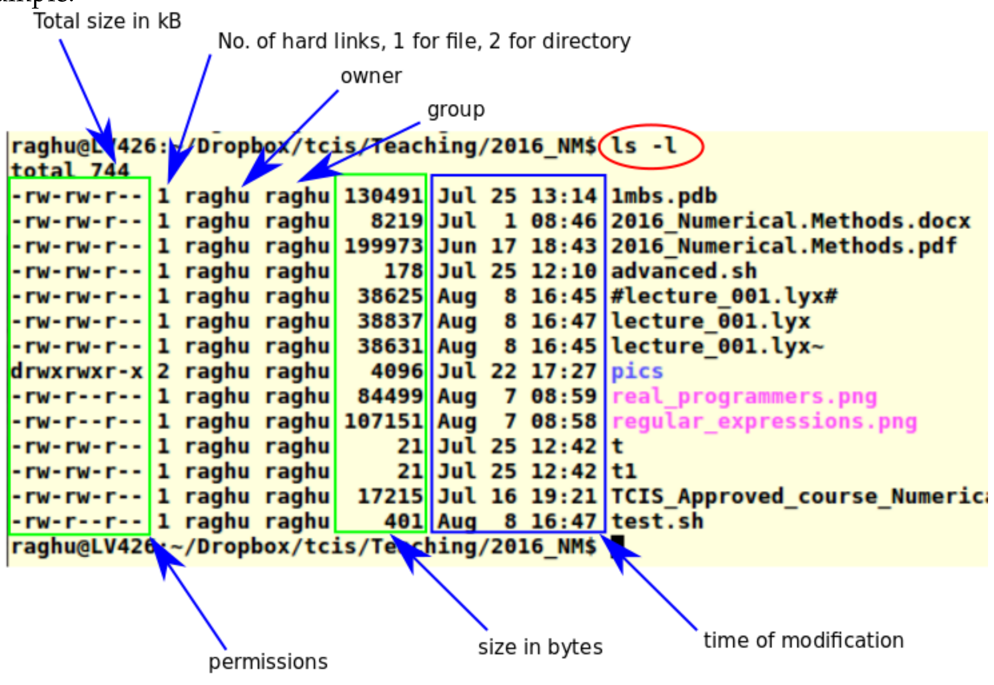
```
$ whatis 'he teaching?'
$ [ Where is my brain?
$ ar -m universe
$ who will passthiscourse?
$ %"Easy teaching?"
$ ^How did the SN1 reaction go?^
$ man superman
```

Later we will learn the proper usage of some of these commands.

- The command `time` returns the time of execution of a command or a program, try `$ time seq 1 1000000000 | tail -3`
- 'real time' is the wall-clock time (or elapsed time), and it is the time period for which you have to wait for the process to complete.
- The sum of 'user time' and 'sys time' is the CPU time, which is reported in the literature. The 'user' value is the time time required for the program execution, including calls to libraries. The 'sys' value is the time needed to execute system calls.
- Typically,
  - when real < user, the program/command executes only CPU-bound tasks (e.g. adding two numbers), and utilizes multiple cores/threads.
  - when real = user, the program/command executes only CPU-bound tasks, and does not utilize multiple cores/threads.
  - when real > user, the task involves mostly serial (non-parallel) tasks, such as input/output (I/O) operations.

3.3 *File system*

List the content of a directory using `$ ls -l`. Below is an annotated example.

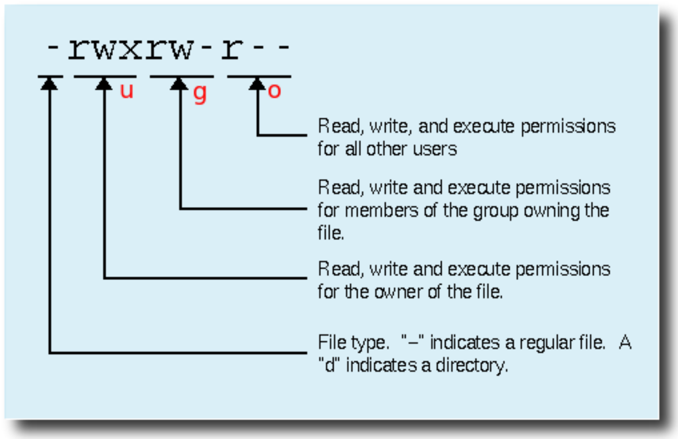


```
raghu@LV426:~/Dropbox/tcis/Teaching/2016_NM$ ls -l
total 744
-rw-rw-r-- 1 raghu raghu 130491 Jul 25 13:14 1mbs.pdb
-rw-rw-r-- 1 raghu raghu   8219 Jul  1 08:46 2016_Numerical.Methods.docx
-rw-rw-r-- 1 raghu raghu 199973 Jun 17 18:43 2016_Numerical.Methods.pdf
-rw-rw-r-- 1 raghu raghu    178 Jul 25 12:10 advanced.sh
-rw-rw-r-- 1 raghu raghu  38625 Aug  8 16:45 #lecture_001.lyx#
-rw-rw-r-- 1 raghu raghu  38837 Aug  8 16:47 lecture_001.lyx
-rw-rw-r-- 1 raghu raghu  38631 Aug  8 16:45 lecture_001.lyx~
drwxrwxr-x 2 raghu raghu   4096 Jul 22 17:27 pics
-rw-r--r-- 1 raghu raghu  84499 Aug  7 08:59 real_programmers.png
-rw-r--r-- 1 raghu raghu 107151 Aug  7 08:58 regular_expressions.png
-rw-rw-r-- 1 raghu raghu     21 Jul 25 12:42 t
-rw-rw-r-- 1 raghu raghu     21 Jul 25 12:42 t1
-rw-rw-r-- 1 raghu raghu  17215 Jul 16 19:21 TCIS_Approved_course_Numeric
-rw-r--r-- 1 raghu raghu    401 Aug  8 16:47 test.sh
raghu@LV426:~/Dropbox/tcis/Teaching/2016_NM$
```

The full path of the working directory can be obtained using `$ pwd`. Since path names are usually long, paths relative to the working directory are assigned the shortcuts "." (dot) and ".." (dot dot). `$ ls ./*` will list all files in the working directory, and `$ ls ../*` will list all files directory, one level below the working directory. Similarly, ~ is the shortcut for `/home/username`. Try the following commands and note down your observations.

```
$ pwd
$ cd .; pwd
$ cd ..; pwd
$ cd ../.; pwd
$ cd ../..; pwd
$ cd ~; pwd
$ cd ~/Desktop; pwd
```

3.4 Permissions


-rwxrw-r--
u
g
o
Read, write, and execute permissions for all other users
Read, write and execute permissions for members of the group owning the file.
Read, write and execute permissions for the owner of the file.
File type. "–" indicates a regular file. A "d" indicates a directory.

The following commands demonstrate how permissions can be changed for various purposes.

```
$ echo 1 > file.txt; ls -l file.txt
$ chmod o+rw file.txt ; ls -l file.txt
$ chmod og+x file.txt ; ls -l file.txt
$ chmod og-x file.txt ; ls -l file.txt
$ chmod uog+x file.txt ; ls -l file.txt
```

- Typical uses of chmod includes disabling write permissions to `u` and `g`, the usage is `$ chmod og-w filename`
- For executable programs (we will generate more of these in this course) set execute permissions using `$ chmod +x filename`

3.5 Elementary bash scripting

Linux commands can be combined with bash progamming structures to write scripts. Simply put, scripting facilitates automatizing tasks. We will look at the four basic programming structures: variables, conditionals, loops, and functions.

Variables:

```
  $ Anything following # is a comment
  $ a=1; echo $a # a is the variable, 1 is assigned to it
  $ a = 1; echo $a # No empty spaces around =
  $ a=one; echo $a # a is now a string
  $ a=2; b=3; let c=a*b; echo $c
  $ i=1; seq 1 3 > file_$i; ls -l
  $ i=1; seq 1 3 > file_$i.txt; ls -l
  $ i=1; mkdir dir_$i; ls -l
  $ str=newfile; i=1; seq 1 10 > “$str”_$i; ls -l
  $ mydir=$(pwd) # output of a cmd can be made a variable
  $ int_list=$(seq 1 10); echo $int_list
```

Conditionals:

```
  $ a=1; b=2; if [ $a -lt $b ]; then echo "a is less than
b"; fi


  $ a=1
  b=2
  if [ $a -lt $b ]; then
   echo "a is less than b"
  fi


  $ a=3
  b=2
  if [ !  $a -lt $b ]; then
   echo "a is not less than b"
  fi


  $ a=1
  b=2
  if [ $a -lt $b ]; then
   echo "a is less than b"
  else
   echo "a is not less than b"
```

```
fi
```

```
$ a=1
b=2
if [ $a -lt $b ]; then
 echo "a is less than b"
elif [ $b -lt $a ]; then
 echo "b is less than a"
else
 echo "a equals b"
fi
```

```
$ a=1
b=”one”
if [ $a != $b ]; then
 echo "a and b differ"
fi
```

```
$ a=1; b=2; c=3
if [ $a -lt $b ] && [ $b -lt $c ]; then
 echo "a is less than c"
fi
```

```
$ echo 1 > file.txt
if [ -a file.txt ]; then
 cat file.txt
fi
```

```
$ mkdir dir01
if [ -d dir01 ]; then
 cd dir01; pwd
fi
```

- Binary operators for comparing strings in lexicographic order: ==, !=, <, >
- Arithmetic binary operators: -eq, -ne, -gt, -ge, -lt, -le
- Combining expressions: &&, ||
- `[ -a filename ]` True if filename exists
- `[ -d filename ]` True if filename exists and is a directory

Loops:

```
$ for i in $( ls ); do
 echo $i
done


$ for a in {1..10}; do
 echo $a
done


$ for ((a=1; a <= 10 ; a++)); do
 echo $a
done


$ for i in $(seq 1 10); do
 echo $i
done


$ cnt=0
while [ $cnt -lt 10 ]; do
 let cnt=cnt+1
 echo $cnt
done


$ cnt=1
until [ $cnt -gt 10 ]; do
 echo $cnt
 let cnt+=1
done
```

The following example shows how a simple bash script can be used to find all prime numbers between 1 and 100. Note a minor detail that `wc -w` returns only the number of words.

```
for i in $(seq 1 100); do
 p=$(factor $i | wc -w)
 if [ $p -eq 2 ]; then
  echo $i is prime
 fi
done
```

Functions:
Functions are few lines of code, performing one particular task, but used multiple times in a script. These lines when coded as a function,

can be used without having to explicitly writing the lines again. Simply put, using functions enables code-recycling.

The following function takes as argument a filename and returns the number of lines, words, and bytes. If the file does not exist, returns a warning.

```
count_file_lines () {
 if [ -a $1 ]; then
  N_lines=$( wc $1 | awk '{print $1}' )
  N_words=$( wc $1 | awk '{print $2}' )
  N_bytes=$( wc $1 | awk '{print $3}' )
  echo “File exists”
  echo “The file has” N_lines, “lines”
  echo “The file has” N_words, “words”
  echo “The file has” N_bytes, “bytes”
 else
  echo “WARNING: The file does not exist”
 fi
}
```

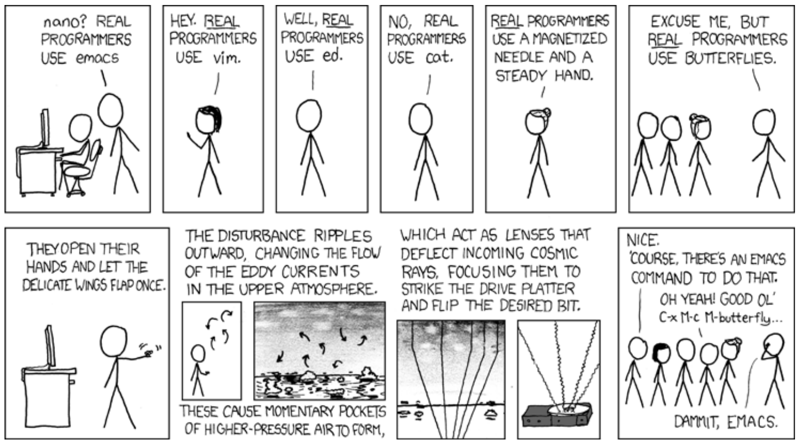
Write the function in a file, `test_fn.sh`, and execute as `$ bash test_fn.sh < test_fn.sh`

3.6 *Text editor*

Later in this course, we will write large programs for which you may want to learn basic text processing using a text editor.

- `gedit` is the editor supplied with the Gnome desktop environment, using which you can create/rename/open a file via mouse clicks. To open a file (new or old), type `$ gedit filename`
- emacs and vi editors include several useful features. Here is a primer for emacs, https://www.gnu.org/software/emacs/tour/.
- A great way to get started with the vi editor is by typing `vimtutor` in the terminal.
- `less` is a program for (only) quickly reading the content of a file. Type `$ seq 1 1000 > test.txt ; less test.txt` in your terminal and try the navigation short-cuts `G`, `100G`, `1G`, `spacebar`, `b`, `/99`, `n`, `n`, `q`. Note down your observations.

Here is an xkcd to help you choose an editor


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

3.7 *bc, an arbitrary precision calculator language*

`bc` (Basic Calculator) is a handy desktop calculator, suitable for arbitrary-precision arithmetic (aka infinite-precision arithmetic). Start `bc` by typing `bc` in your terminal. To exit, type `quit` or `Ctrl+d`. Alternatively, you can pass the input as command line arguments. Executing `bc` with the option `-l` loads the math libraries (with the `-l` option `s(x)`, `c(x)`, `a(x)`, `l(x)`, `e(x)` return sine, cosine, arctangent, natural lagorithm, and exponential of `x`, respectively). Try the following one-liners.

```
  $ echo "2^8" | bc
  $ for i in $(seq 1 10); do echo "2^$i" | bc; done
  $ echo "scale=6; 4*a(1)" | bc -l
```

Write the following 'recursive' function in the `file /home/fact.bc`

```
define fact(x){
   if (x<=1) {
      return (1)
   }
   return (x*f(x-1))
}
```

and try

```
  $ echo “fact(4)” | bc -l /home/fact.bc
  $ for i in $(seq 1 10); do echo "fact(i)" | bc -l /home/fact.bc;
done
```

You may want to store your favorite functions and unit conversion factors in a `start.bc` file while using `bc` as a desktop calculator.

3.8 Text processing

- `sed`, `awk`, and `perl` are advanced tools for text manipulations and more.
- Typical usage includes parsing large ('raw') output file(s), and summarizing the main content. Also useful for string substitutions.

Here are some simple examples for you to try and learn (remember to note down your observations)

```
  $ echo 1 2
  $ echo 1 2 | awk '{print $1}'
  $ echo 1 2 | awk '{print $2}'
  $ echo 1 2 | awk '{print $1 $2}'
  $ echo 1 2 | awk '{print $1 " " $2}'
  $ echo 1 2 | awk '{printf $1 " " $2}'
  $ echo 1 2 | awk '{printf "%15.8f %15.8f", $1, $2 }'
  $ echo 1 2 | awk '{printf "%15.8f %15.8f \n", $1, $2 }'
  $ echo 1 2 | awk '{printf "%15.8f %15.8f %15.8f\n", $1,
$2, $1+$2 }'
  $ echo "A B C"
  $ echo "A B C A A" | sed 's/A/a/'
  $ echo "A B C A A" | sed 's/A/a/g'
  $ echo "A B C A A" | sed 's/A/a/1'
  $ echo "A B C A A" | sed 's/A/a/3'
```

Here is an xkcd showing what you can do with `perl`

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Exercises - 1

Exercise 1.1: squeeze

A string made of 'n' ASCII characters takes up 'n+1' bytes, where 1 byte goes to the terminator (a new line character). Type some random characters in a file and check the file size, `$ echo 1234 > tmp.txt; ls -l tmp.txt`. Create a file containing integers 1 to 100000 in ascending order, and compress the file using

```
$ zip filename.zip filename
$ tar -czf filename.tar.gz filename
$ bzip2 < filename > filename.bz2
```

Now compare the sizes of the compressed files in bytes with that of the original file. Note down your observations. Google and learn how to uncompress the compressed files.

You may also want to check the manual pages of these coreutils by typing `$ man zip`, `$ man tar` or `$ man bzip2`.

The `time` command lets you time the execution of a command – the usage is `$ time 'command'`, e.g. `$ time seq 1 100000`. Discuss the advantages and disadvantages of the three compression schemes.

Exercise 1.2: classify

Classify the various commands you have encountered above according to their purposes, e.g. navigation, viewing the content (file or directory), file manipulation (copy, delete, etc.), input/output (I/O) redirection (recall the use of <, <<, >, >>, |, and tee), and advanced tasks.

Exercise 1.3: random

The bash variable `$RANDOM` returns a random positive integer between 0 and $2^{15} - 1$ (i.e. 32767). Try `$ echo $RANDOM` in your terminal. You can 'pipe' the bash output to `bc` and obtain a real number, try `$ echo "scale=8; $RANDOM" | bc`. Combining `bash` and `bc`, write a 'one liner' for obtaining a random number in the range 0 to 1. Repeat for the range -1 to 1.

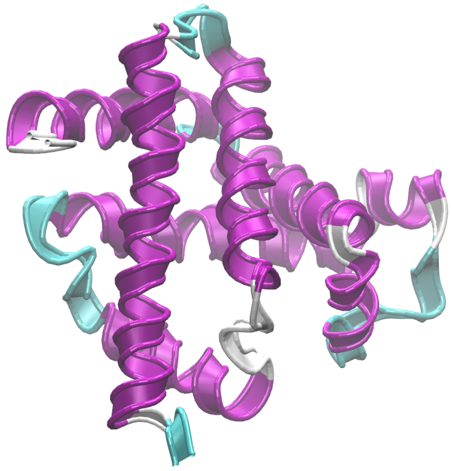
Exercise 1.4: counting atoms

Protein Data Bank (PDB) format is a standard format for collecting atomic coordinates of biological macromolecules. Download the coordinates of a collagen-like protein from `http://files.rcsb.org/view/1mbs.pdb` and 'grep' for the string 'ATOMS' to find out the total number of atoms in this protein. Inspect the file format (using `less`) and write a 'one-liner' to calculate the number of non-organic atoms (i.e., excluding H,C,N,O,S,P).

Using the `wget` command you can download a file from a terminal, without having to use a browser. Try

```
$ wget http://files.rcsb.org/view/1mbs.pdb; ls -l
```

Here the structure of the protein



Exercise 1.5: parse

The output of a geometry relaxation calculation for the water molecule, performed using the program Gaussian, can be downloaded from here http://cccbdb.nist.gov/iofiles/h2o/m8b1.out. The Cartesian coordinates of the atoms, at each step of the geometry update are summarized after the string "Standard orientation". Go to the directory where you have downloaded the file and try `$ grep -A8 'Standard o' m8b1.out`. The last instant corresponds to the minimum energy structure. Write a bash function which takes as input a filename, and prints in a file named `geo.xyz`, the final coordinate in the following format

```
3
some comment in this line
O      0.00000000      0.00000000      0.11974800
H      0.00000000      0.76152100     -0.47899100
H      0.00000000     -0.76152100     -0.47899100
```

This xyz format is very commonly used for collecting the atomic coordinates. The file `geo.xyz` can be processed by various program such as jmol, avogadro, etc., for visualization, and other manipulations.

4. Fortran

4.1 Bits and Bytes

- Bit (binary digit) is the *fundamental unit* of memory. A bit can be physically realized using a two-state device. Possible values a bit can store are the two states of the device, represented typically by 0 or 1.

- A byte, B, (= 8 Bits) is a *derived unit* of memory. Much larger units that are multiples of bytes are commonly encountered in computational sciences.

Metric		IEC	
1000 B	kB, kilobyte	1000 B	kiB, kibibyte
1000^2 B	MB, megabyte	1024^2 B	MiB, mebibyte
1000^3 B	GB, gigabyte	1024^3 B	GiB, gibibyte
1000^4 B	TB, terabyte	1024^4 B	TiB, tebibyte
1000^5 B	PB, petabyte	1024^5 B	PiB, pebibyte

Table 1: International Electrotechnical Commission (IEC) recommends differentiating the units that are powers of 1000 from those that are powers of 1024. However, one commonly encounters same short forms for both units, such as, kB for both kilobyte and kibibyte, etc.

- Primary and secondary memory.

- RAM (Random Access Memory) is a computer's primary storage device. The data stored in RAM will be deleted when the CPU is switched off. Hence RAM is said to have volatile memory. RAM consists of virtual address space (VAS), where each address is a numerical number (in binary, decimal, or hexadecimal representation). A 64-bit system uses 64-bit addresses. A 64-bit (8 B) laptop with 16 GB RAM can have $16\text{x}1024^3/8 = 2.14 \times 10^9$ addresses. Each address can hold only 1 Byte of data.

- To simplify programming using numerical address (binary or hexadecimal), programming languages, such as C and Fortran, have the concept of variables. A variable is a named location that can store a value of a particular type (integer, real, character, logical, etc). For example, a Fortran statement like `integer :: i1`, declares an integer data (of size 4 bytes) for which 4 addresses in the VAS will be reserved.

The following Fortran (to be precise, Fortran90) program (Program 1) demonstrates various integer kinds. The intrinsic (built-in) function `huge` returns the largest integer corresponding to a particular kind, and the intrinsic function `loc` returns the address reserved for a variable in the VAS. The intrinsic function `bit_size` takes as argument an integer variable, and returns the number of

Program 1 Fortran integer types

```
program datatype_int

  implicit none

  integer(kind=1)  :: int_1byte        ! one address
  integer(kind=2)  :: int_2byte        ! two addresses
  integer(kind=4)  :: int_4byte        ! four addresses
  integer(kind=8)  :: int_8byte        ! eight addresses
  integer(kind=16) :: int_16byte       ! sixteen addresses
  integer          :: int1             ! default integer type
  integer(kind=4)  :: int_4bytes(1: 2) ! a 1D array (vector) of 2 elements

  print *, "Largest integer of kind 1 is", huge(int_1byte)       ! 2^7-1
  print *, "Largest integer of kind 2 is", huge(int_2byte)       ! 2^15-1
  print *, "Largest integer of kind 4 is", huge(int_4byte)       ! 2^31-1
  print *, "Largest integer of kind 8 is", huge(int_8byte)       ! 2^63-1
  print *, "Largest integer of kind 16 is", huge(int_16byte)     ! 2^127-1
  print *, "Largest integer of default integer kind", huge(int1) ! kind=4

  print *, "integer, kind=16 needs ",bit_size(int_16byte)," bits"
  print *, "integer, kind=16 needs ",bit_size(int_16byte)/8," bytes"

  ! address of the first element of the vector int_4bytes
  print *, loc(int_4bytes(1))
  ! address of the second element of the vector int_4bytes
  print *, loc(int_4bytes(2))

end program datatype_int
```

bits needed to store this variable. Save the code in a file named, say, `example_int.f90`, and compile the program as `$ gfortran example_int.f90 -o example_int.x` . The program `example_int.x` can be executed as `./example_int.x`

Note that in the code listed above, anything following the exclamation mark "`!`" is a comment, and will be ignored by the compiler.

- In Fortran, an N-bit integer variable can be used to store any of the 2^N integers in the range $-2^{N-1}$to $2^{N-1}-1$. A hypothetical 1-bit integer can only store the two integers -1, and 0.

4.2 *Ariane 5 and the overflow-bug*

"On June 4, 1996 an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after its lift-off from Kourou, French Guiana. Ariane explosion The rocket was on its first voyage, after a decade of development costing $7 billion. The destroyed rocket and its cargo were valued at $500 million. A board of inquiry investigated the causes of the explosion and in two weeks issued a report. It turned out that the cause of the failure was a software error in the inertial reference system. Specifically a 64 bit floating point number relating to the horizontal velocity of the rocket with respect to the platform was converted to a 16 bit signed integer. The number was larger than 32,767, the largest integer storeable in a 16 bit signed integer, and thus the conversion failed."
Source: https://www.ima.umn.edu/~arnold/disasters/ariane.html.
Here is the youtube video: https://www.youtube.com/watch?v=gp_D8r-2hwk

The 'software-bug' that was responsible for the failure of Ariane 5 is an overflow error, which arises when a variable declared as a low memory datatype (i.e. fewer addresses in the VAS) is assigned a numerical value requiring large memory (requiring more addresses in the VAS). Modern Fortran compilers can detect such scenarios and either warn the user during compilation or simply fail to compile with a relevant error message. Program 2 demonstrates overflow by assigning an 4-byte integer to a 1-byte integer variable. Compiling this program with the `gfortran` compiler (version 5.4.0) fails to produce an executable program, and returns the following error:

Program 2 Overflow error

```
program demo_overflow

  implicit none

  integer(kind=1) :: hori_speed

  hori_speed = 323
  print *, hori_speed

end program demo_overflow
```

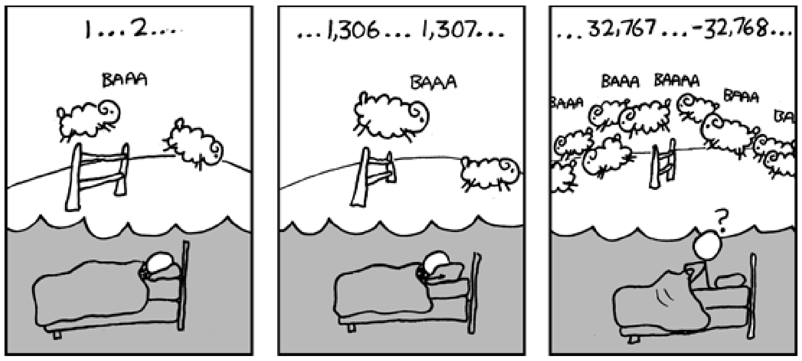
```
overflow.f90:7:10:
  speed = 323
```

```
          1
   Error:  Arithmetic overflow converting INTEGER(4) to
INTEGER(1) at (1).  This check can be disabled with the
option '-fno-range-check'
```

However, compiling the same program (Program 2) with the ifort compiler (version 14.0.2) returns only a warning

```
   overflow.f90(7):  warning #6047:  The BYTE / LOGICAL(KIND=1)
/ INTEGER(KIND=1) value is out-of-range.  [323]
     speed = 323 ----------^
```

and compiles the program. The resulting program when executed returns the answer 67, which is the remainder in the integer division 323/256. Integer overflow happens when an N-bit variable is assigned a value that is either less than -2^{N-1} or greater than $2^{N-1} - 1$. Specifically, integer overflow causes the number to be reduced modulo a power of two — in our case after reaching 127, the couting starts at -128, goes through -127,-126, and so on, eventually ending at 67 as the three-hundred-and-twenty-third integer. The following xkcd illustrates integer overflow in a humorous way.



4.3 *Floating point arithmetic*

- Recall that the linux program, bc, performs arbitrary precision arithmetic. Such arithmetic is used in applications where the speed of computation is not a key factor, or where precise results are required at the expense of speed. For faster execution, calculations are performed using a computer's arithmetic logic unit (ALU) or floating-point unit (FPU). Both ALU and FPU are part of the processor hardware. Calculations performed on these units use numbers between 8 and 64 bits of precision (hence finite precision).

- It may be realized that due to finite precision, we will only work with a subset of all real numbers. Arithmetic performed using such numbers is called as floating point arithmetic, where a real number x is represented by a *machine number* closest to it. This process is called *rounding*

$$\begin{aligned} x &\approx \widetilde{x} \\ &= m \times 10^r, \end{aligned}$$

where r is an integer (called *characteristic* of the number), and m a real numer is called the *mantissa*. Formally there is no limit to the magnitude of r or to the number of digits in m. However, later we will see that memory limitations of datatypes will impose constraints on both these aspects. In Fortran, we will call the number of digits of m as the `precision`, and the exponent range $10^{-r} - 10^r$ as the `range`.

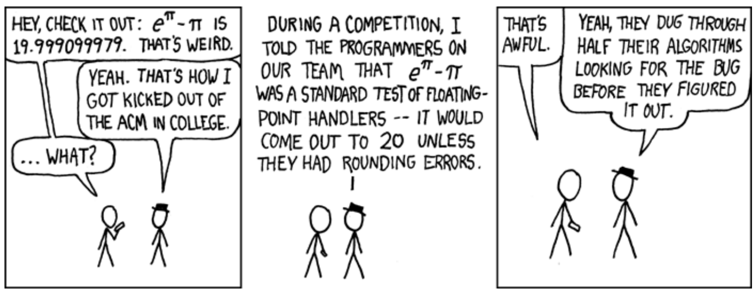
- Notation for real, and floating point number spaces are $\mathbb{R}$, and $\mathbb{F}$, respectively. Obviously, $\mathbb{F} \in \mathbb{R}$, i.e., $\mathbb{F}$ is a subset of $\mathbb{R}$.
- $\widetilde{x}$ is said to be in *normalized engineering form* if $0.1 \leq |m| < 1$, for example, $\pi \approx 0.314159 \times 10^1$.
- $\widetilde{x}$ is said to be in *normalized scientific form* if $1 \leq |m| < 10$, for example, $\pi \approx 3.14159 \times 10^0$.
- For $z = x \times y$, due to rounding in floating point arithmetic, we find $\widetilde{z} = \widetilde{(\widetilde{x} \times \widetilde{y})}$, i.e. first x and y are rounded to $\widetilde{x}$ and $\widetilde{y}$, respectively. Then the product of both these numbers is rounded.
- **Rounding to even**: The procedure for rounding a number, x, to t decimal places is as follows
  - Represent the number in normalized scientific form.
  - Let ϕ be the digit at the t th decimal, and let η be the part of x that corresponds to positions to the right of the t th decimal. For example, when rounding the irrational number π (3.1415926...) to 3 decimal places, $\phi = 1$ (at the third decimal place) and $\eta = 0.0005926$ (after the third decimal place).
  - Now
    * if $\eta > 0.5 \times 10^{-t}$, set $\phi = \phi + 1$.
    * if $\eta < 0.5 \times 10^{-t}$ ϕ is unchanged.
    * if $\eta = 0.5 \times 10^{-t}$, set $\phi = \phi + 1$, only if ϕ is odd.
- The relative error due to rounding is given by

$$\rho = \frac{|x - \widetilde{x}|}{|x|} \leq \varepsilon,$$

where ε is called *machine accuracy*. Note that ε is the smallest real number for which $1 + \varepsilon \neq 1$. When a number is rounded to t

decimal places, the upper bound to ε is 0.5×10^{-t}. In practical computations, the exact value of ε can be generated and can be seen to be less than 0.5×10^{-t}.

- If you can appreciate the humour in the following xkcd, then you are doing well in floating point arithmetic.


HEY, CHECK IT OUT: $e^{\pi} - \pi$ IS 19.999099979. THAT'S WEIRD.
YEAH. THAT'S HOW I GOT KICKED OUT OF THE ACM IN COLLEGE.
... WHAT?
DURING A COMPETITION, I TOLD THE PROGRAMMERS ON OUR TEAM THAT $e^{\pi} - \pi$ WAS A STANDARD TEST OF FLOATING-POINT HANDLERS -- IT WOULD COME OUT TO 20 UNLESS THEY HAD ROUNDING ERRORS.
THAT'S AWFUL.
YEAH, THEY DUG THROUGH HALF THEIR ALGORITHMS LOOKING FOR THE BUG BEFORE THEY FIGURED IT OUT.

Program 3 summarizes the real floating point datatypes available in Fortran90. The `loc` function that returns memory addresses can also be used for real arguments. Note that both semi-quadruple precision (kind=10), and quadruple precision (kind=16) real numbers require 16 addresses for their storage.

- Significant digits (or significant figures) is a concept we had encountered in elementary chemistry/physics courses. Formally, significant digits of a number are those digits which we know for certain to be true. We will look at a more robust mathematical definition of this concept. A number x is said to be approximated by $\widetilde{x}$ to t significant figures if t is the largest non-negative integer for which
$$\frac{|x - \widetilde{x}|}{|x|} \leq 5 \times 10^{-t}.$$
Here are some examples: The number of significant digits in the following numbers
  - $0.001234 \pm 0.5 \times 10^{-5}$
  - $56.789 \pm 0.5 \times 10^{-3}$
  - 210000 ± 5000

  are 3, 5, and 2, respectively. Note that in the first number, $|x - \widetilde{x}|$ is the uncertainty 0.5×10^{-5}, and $x = 0.001234$.

Program 3 Fortran real types

```
program datatype_float

  implicit none

  real(kind=4)     :: r4    ! single precision real number, 4 Bytes
  real(kind=8)     :: r8    ! double precision, 8 Bytes
  real(kind=10)    :: r10   ! semi-quadruple precision, 16 Bytes
  real(kind=16)    :: r16   ! quadruple precision, 16 Bytes
  real             :: r     ! same as real(kind=4)
  double precision :: dp    ! same as real(kind=8)

  print *, '=== Double Precision Real ==='
  r8 = 1.0
  print *, r8
  print *, epsilon(r8)      ! machine accuracy
  print *, precision(r8)    ! decimal precision
  print *, range(r8)        ! decimal exponent range, 10^-r  to 10^r
  print *, tiny(r8)         ! smallest positive real number
  print *, digits(r8)       ! no. of significant binary digits
  print *, minexponent(r8)  ! minimal binary exponent range
  print *, maxexponent(r8)  ! maximal binary exponent range

  ! shortcut to fix precision for constant real numbers
  print *, "==== demo of r4, r8, r16 1.0 ===="
  print *, "single precision: ", 1.e0
  print *, "double precision: ", 1.d0
  print *, "quadruple precision: ", 1.q0

end program datatype_float
```

A parameterized variable is an integer or a real variable declared with an optional `kind` parameter. The performance of `kind` may differ across compilers. Hence it is recommended to determine the kind-type dynamically in a program.

- `selected_int_kind(r)` returns the smallest integer-kind with maximum range $\geq 10^r$.
- `selected_real_kind(r,p)` returns the smallest real-kind with a minimum of p decimal digits, and with maximum range $\geq 10^r$.

4.4 "Problem condition" and "Algorithm stability"

- For a given problem (or a given equation) $P(i) \rightarrow o$, and $P(i + \Delta i) \rightarrow o + \Delta o$,
  - if $|\Delta i/i| = \varepsilon$, and if $|\Delta o/o|$ is small, then the problem is *well-conditioned*. The resulting deviation in output is called as *unavoidable error*.
  - if $|\Delta i/i| = \varepsilon$, and if $|\Delta o/o|$ is large, then the problem is *ill-conditioned*.
- A problem may be ill-conditioned only for a narrow range of x, e.g. plot $\sin^2 x/x^2$ vs. $(1 - \cos^2 x)/x^2$.
- An algorithm is a sequence of elementary operations which generate an o for a given x, typically via some intermediate quantities q_j. Let us simply represent the execution of an $N-$step algorithm as $A(i) \rightarrow q_1 \rightarrow q_2 \ldots q_N \rightarrow o$.
  - if i or q_j are rounded, and if the error in $|\Delta o/o|$ is not much larger than the unavoidable error, then the algorithm is numerically stable.
- Numerical instability, e.g.
  - $e = \lim\limits_{n\rightarrow\infty} (1 + 1/n)^n$
  - $e^x = \lim\limits_{n\rightarrow\infty} (1 + x/n)^n$
- Later we will revisit these concepts, with some modifications, while discussing linear algebra topics.

Exercises - 2

Exercise 2.1: neither distributive nor associative

Show that the law of associativity, and the law of distributivity are *not* obeyed in floating point arithmetic.

Exercise 2.2: weighing the Captain with the ship

1. Show that the sum $z = x + y$ is well-conditioned for $xy \geq 0$
2. Show that the sum $z = x + y$ is ill-conditioned in floating point arithmetic for $x \approx -y$ and $x \neq 0 \neq y$.

3. Show that the system of linear equations

$$\begin{aligned} ax + y &= 1, \\ x + ay &= 0, \end{aligned}$$

is ill-conditioned for $|a| \approx 1$.

4. The term

$$z = x - \sqrt{x^2 + y}, \qquad x \gg y > 0$$

is well-conditioned. If the relative error in calculating the square root equals ε (the machine accuracy) then

$$z = x - \sqrt{x^2 + y}$$

is not a numerically stable algorithm, but

$$z = \frac{-y}{x + \sqrt{x^2 + y}}$$

is. Why?

Exercise 2.3: mind the digits

1. Round to even the following numbers to 4 decimals: 1.23456, 0.002556, 1.4564500, 8.5963500, 98.22229000

2. Find the number of significant digits in the following physical constants:

   (a) Planck constant, $h = 6.626070040 \pm 0.000000081 \times 10^{-34}$ J s

   (b) Atomic mass constant, $m_u = 1.660539040 \pm 0.000000020 \times 10^{-27}$ Kg

   (c) Faraday constant, $F = 96485.33289 \pm 0.00059 \quad \mathrm{C\,mol^{-1}}$

Exercise 2.4: Factorials Reloaded

What is the largest integer for which factorial can be computed, exactly without rounding errors, when working exclusively with Fortran90 integers of kind 1, 2, 4, 8, and 16, as well as real numbers of kind 4, 8, and 16? An example Fortran90 implementation using the loop structure "`do...end do`" can be found at the bottom of the page, http://www.tutorialspoint.com/fortran/fortran_do_loop.htm

5. Elementary algorithms

Swap

$$a, b = b, a \tag{1}$$

```
Input:  a, b
Output:  b, a
c = a
a = b
b = c
```

Works also for swapping arrays.

Dot Product

$$c = \mathbf{a} \cdot \mathbf{b} \rightarrow c = \sum_{i=1}^{n} a_i \cdot b_i \tag{2}$$

```
Input:  n, a(1:n), b(1:n)
Output:  c
c = 0
do i = 1, n
  c = c + a(i) * b(i)
end do
```

Matrix-vector multiplication

$$\mathbf{y} = \mathbf{A} \cdot \mathbf{x} \rightarrow x_i = \sum_{k=1}^{n} A_{i,k} \cdot x_k \tag{3}$$

```
Input:  n, A(1:n,1:n), x(1:n)
Output:  y(1:n)
do i = 1, n
  y(i) = 0
  do k = 1, n
    y(i) = y(i) + A(i,k) * x(k)
  end do
end do
```

Matrix-matrix multiplication

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} \rightarrow C_{i,j} = \sum_{k=1}^{n} A_{i,k} \cdot B_{k,j} \tag{4}$$

```
Input:  n, A(1:n,1:n), B(1:n,1:n)
Output:  C(1:n,1:n)
do i = 1, n
  do j = 1, n
    C(i,j) = 0
    do k = 1, n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    end do
  end do
end do
```

Vector normalization

$$\mathbf{x} = \mathbf{x}/\sqrt{\mathbf{x}\cdot\mathbf{x}} \rightarrow x_i = x_i/\sqrt{\mathbf{x}\cdot\mathbf{x}} \tag{5}$$

```
Input:  n, x(1:n)
Output:  x(1:n)
do i = 1, n
  x(i) = 0
  do k = 1, n
    x(i) = x(i)/sqrt(dot_product(x,x))
  end do
end do
```

For all the four algorithms can be implemented easily using the Fortran intrinsic functions `dot_product` and `matmul`

- `c = dot_product(a,b)`
- `y = matmul(A,x)`
- `C = matmul(A,B)`
- `x = x/sqrt(dot_product(x,x))`

Gaussian Elimination

The linear system of equations $\mathbf{A}_{N\times N}\cdot\mathbf{x}_N = \mathbf{a}_N$ with $|\mathbf{A}| \neq 0$, has exactly one solution. The solution can be obtained by matrix inversion, $\mathbf{x} = \mathbf{A}^{-1}\mathbf{a}$. To solve the linear system by Gaussian elimination, the problem is transformed into a new problem

$$\mathbf{U}_{N\times N}\cdot\mathbf{y}_N = \mathbf{b}_N, \tag{6}$$

where $\mathbf{U}$ is a non-singular upper-triangular matrix.

```
Input:  n, U(1:n,1:n), b(1:n)
Output:  y(1:n)
do i = n, -1, 1
  y(i) = ( b(i)-dot_product( U(i,i+1:n),y(i+1:n) ) )/ U(i,i)
end do
```

This procedure is called backward substitution, because one starts by finding from the n-th element of $\mathbf{y}$. The transformation of equations from $\mathbf{A} \cdot \mathbf{x} = \mathbf{a}$ to the equivalent problem $\mathbf{U} \cdot \mathbf{y} = \mathbf{b}$ is possible because the solution $\mathbf{x}$ is invariant, up to permutation of its elements, when

- a multiple of any equation is added to another equation, or
- if two equations are swapped

This procedure to convert $\mathbf{A} \cdot \mathbf{x} = \mathbf{a}$ to $\mathbf{U} \cdot \mathbf{y} = \mathbf{b}$ is due to Gauss.

6. Nonlinear equations

6.1 *Some definitions*

A function f is continuous at x_0 if

$$\lim_{x \to x_0} f(x) = f(x_0).$$

The function f is continuous on the set X if it is continuous at each element in X. The set of all functions that are continous on the set X is denoted by $C(X)$. The set of all functions continuous on the closed interval $[a,b]$ is denoted by $C\left[a,b\right]$. The function f is differentiable at x_0 if

$$f'\left(x_0\right) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

exists. The set of all functions that have first, second until $n-$th order derivatives on X is denoted $C^n(X)$. Polynomial, rational, trigonometric, exponential and logarithmic functions are in $C^\infty(X)$.

6.2 *The problem*

We will study methods for solving an equation

$$f(x) = 0, \tag{7}$$

where f is a real-valued function, and x is a scalar-valued real variable. The value of x for which the above equation is satisfied is called as the *solution or root of the equation*, and the same number is called as the *zero of the function* f. Sometimes, additional conditions such as f being differentiable are given. Typically, a nonlinear equation $f(x) = 0$ cannot be solved analytically, i.e., the solution cannot be expressed in a form involving only elementary artihmetic operations and square roots. Even when $f(x)$ is a polynomial, the above equation cannot be generally solved analytically when the degree of the polynomial is larger than 4. Analytic solution is not possible also for a transcendental equation, i.e., when $f(x)$ comprises of exponential or trigonometric functions.

Let x^* be a solution to the nonlinear equation. Most numerical methods for solving nonlinear equations involve *iteratively* finding approximations for x^* by constructing a sequence $\{x_k\}_{k=1}^{\infty}$that converges to the root

$$\lim_{k \to \infty} x_k = x^* \tag{8}$$

A rule of thumb in choosing an iterative procedure is: The convergence can be made faster by using more information about f.

6.3 *Bisection method*

6.3.1 *Theorem:*

If $f \in C[a,b]$ and $f(a) \cdot f(b) < 0$, bisection method generates a sequence $\{x_k\}_{k=1}^{\infty}$ approximating the root x^* of f with

$$|x_n - x^*| \leq \frac{b-a}{2^n} \tag{9}$$

$\{x_k\}_{k=1}^{\infty}$ converges to x^* at a rate of $\mathcal{O}\left(1/2^n\right)$.

6.3.2 *Code:*

```
Input:  a0, b0, tol, maxeval
External:  f
Output:  x, neval
Call:  bisec(f, a0, b0, tol, maxeval, x, neval)
a = a0
b = b0
neval = 0
loop1:  do
  x = (a + b)/2.0d0
  fx = f(x)
  fb = f(b)
  neval = neval + 2
  if (fx*fb > 0.0d0) then
    b = x
  else
    a = x
  endif
  if ( (abs(fx) <= tol) .or.  (neval >= maxeval) ) &
  exit loop1
end do loop1
```

6.4 *Fixed-point iteration*

The number x is a fixed-point for a function g if $x = g(x)$. A root-finding problem $f(x) = 0$, can be converted into a fixed-point problem $g(x) = x$ by simple manipulations. The converged fixed-point will then be the root of the original equation. For example, $f(x) = \exp(-x) + x/5 - 1 = 0$, can be written as $x = 5(1 - \exp(-x)) = g(x)$.

6.4.1 *Theorem:*

Let $g \in C[a,b]$ and $g(x) \in [a,b]$ for all $x \in C[a,b]$. In addition, let $g'(x)$ exists on (a,b) and that a positive constant $0 < k < 1$ exists with $|g'(x)| \leq k$, for all $x \in C\,(a,b)$. Then for any number $x_0 \in [a,b]$, the sequence defined by

$$x_n = g\left(x_{n-1}\right), \qquad n \geq 1,$$

converges to the unique fixed point x^* in $[a,b]$. Further, $\{x_k\}_{k=1}^{\infty}$ converges to x^* at a rate of $\mathcal{O}\left(k^n\right)$. More precisely, the convergence follows

$$|x_n - x^*| \leq \frac{k^n}{1-k}\left|x_1 - x_0\right|. \qquad (10)$$

To verify this inequality, in practise, one can estimate k as $\left|g\left(x_1\right) - g\left(x_0\right)\right| / \left|x_1 - x_0\right|$.

6.4.2 *Code:*

```
Input:  x, tol, maxeval
External:  g
Output:  x, neval
Call:  fixpt(g, x, tol, maxeval, neval)
neval = 0
loop1:  do
  gx = g(x)
  neval = neval + 1
  if ( (abs(x-gx) <= tol) .or.  (neval >= maxeval) ) &
  exit loop1
  x = gx
end do loop1
```

6.5 *Newton-Raphson method*

6.5.1 *Theorem:*

Let $f \in C^2[a,b]$. If $x \in (a,b)$ is such that $f(x) = 0$ and $f'(x) \neq 0$, then there exists a $\delta > 0$ such that Newton's method defined by

$$x_n = x_{n-1} - \frac{f\left(x_{n-1}\right)}{f'\left(x_{n-1}\right)}, \qquad n \geq 1$$

generates a sequence $\{x_k\}_{k=1}^{\infty}$ converging to x^* for any initial approximation $x_0 \in \left[x^* - \delta, x^* + \delta\right]$.

6.5.2 *Code:*

```
Input:  x, tol, maxeval
External:  f, df
Output:  x, neval
Call:  nr(f, df, x, tol, maxeval, neval)
neval = 0
loop1:  do
  fx = f(x)
  dfx = df(x)
  neval = neval + 2
  if ( (abs(fx) <= tol) .or.  (neval >= maxeval) ) &
  exit loop1
  x = x-fx/dfx
end do loop1
```

6.6 *Modified Newton-Raphson method*

6.6.1 *Multiple root:*

The function $f \in C^m[a,b]$ has a zero of multiplicity m at x^*in (a,b) if and only if

$$f(x^*) = f'(x^*) = \ldots = f^{(m-1)}(x^*) = 0, \quad \text{but} \quad f^m(x^*) \neq 0.$$

For example the function $f(x) = (x-1)^2$ has a double root at 1. Newton-Raphson method will fail to find the root of this function because at $x = 1$, the derivative of this function is zero, i.e., $f'(1) = 2(1-1) = 0$. A more general approach to tackle an equation with multiple roots is to replace the function $f(x)$ in the Newton-Raphson iteration by a new function $u(x)$ which has a single root at the same point where the original function $f(x)$ has a multiple root. One such function is $u(x) = f(x)/f'(x)$.

6.6.2 *Theorem:*

Let $f \in C^m[a,b]$. If we define a new function $u(x) = f(x)/f'(x)$, then $u \in C[a,b]$. For any $x \in (a,b)$ such that $f(x) = 0$ and $f'(x) = 0$, there exists a $\delta > 0$ such that Newton's method defined by

$$x_n = x_{n-1} - \frac{f(x_{n-1})\, f'(x_{n-1})}{[f'(x_{n-1})]^2 - f(x_{n-1})\, f''(x_{n-1})}, \qquad n \geq 1$$

generates a sequence $\{x_k\}_{k=1}^{\infty}$ converging to x^* for any initial approximation $x_0 \in [x^* - \delta, x^* + \delta]$.

6.6.3 Code:

```
Input:  x, tol, maxeval
External:  f, df, d2f
Output:  x, neval
Call:  mnr(f, df, d2f, x, tol, maxeval, neval)
neval = 0
loop1:  do
  fx = f(x)
  dfx = df(x)
  d2fx = d2f(x)
  neval = neval + 3
  if ( (abs(fx) <= tol) .or.  (neval >= maxeval) ) &
  exit loop1
  x = x - f*df/( df**2 - f*d2f )
end do loop1
```

Exercises - 3

Exercise 3.1: Wien's law

Planck's law of radiation in the wavelength representation is given by

$$\rho_\lambda(\lambda, T) = \frac{8\pi hc}{\lambda^5} \frac{1}{\exp(hc/\lambda k_B T) - 1}.$$

Google and find the definitions of various symbols, and constants entering this equation. The wavelength corresponding to maximum energy density is obtained by solving

$$\frac{d\rho_\lambda(\lambda, T)}{d\lambda} = 0$$

which results in the Wien's law

$$\lambda_{\max} T = b.$$

The experimentally deduced value of b is $(2.897729 \pm 0.0000017) \times 10^{-3}$ m K. Find the same constant numerically with maximal precision possible in standard Fortran90.

Exercise 3.2: find all roots

An object file containing an one-dimensional real function can be downloaded from here:

https://drive.google.com/file/d/0Bx3T4IcMYKH3Tk9md1FaZjFWaWM/view?usp=sharing
This 'double precision' function can be envoked as `fn(x)`, where `x` is an input variable of double precision type. Use your favourite numerical method, with various initial guesses, and find all the roots of this function. Note that all roots of this function are singular.

Exercise 3.3: fixed point to Newton

Show that the Newton-Raphson iteration formula

$$x_n = x_{n-1} - \frac{f\left(x_{n-1}\right)}{f'\left(x_{n-1}\right)}$$

can be written as the fixed-point iteration formula

$$x_n = g\left(x_{n-1}\right).$$

Exercise 3.4: multiple roots

Let x^* denote a root of the nonlinear function f, then we can write

$$f(x) = (x - x^*)^q g(x).$$

If $q = 1$, then x^* is a simple root, while if $q > 1$, x^* is a multiple root with multiplicity q.

1. Show that $f'(x^*) \neq 0$ only when $q = 1$
2. If the function $f(x)$ has a multiple root at $x = x^*$, show that at the same point the function $u(x) = f(x)/f'(x)$ has a single root.
3. Discuss various strategies by which one can determine the multiplicity of a root.

6.7 *Contour diagram*

6.7.1 *Inflection point*

An inflection point is a point on a surface at which the sign of the curvature changes. Inflection points may be stationary points (i.e. vanishing first derivative), but are not local maxima or local minima (which have non-vanishing second derivative). A necessary condition for x to be an inflection point is $f''(x) = 0$. For example, for the curve $y = (x-a)^3$ plotted above, the point $x = a$ is an inflection point.

Exercises - 4

Exercise 4.1: gas equations

Some of the equation of states which can capture the non-ideal behaviour of real gases are given by

$$P = \frac{RT}{\overline{V} - b} - \frac{a}{\overline{V}^2} \quad \text{van der Waals} \tag{11}$$

$$P = \frac{RT}{\overline{V} - B} - \frac{A}{T^{1/2}\overline{V}\left(\overline{V} + B\right)} \quad \text{Redlich} - \text{Kwong} \tag{12}$$

$$P = \frac{RT}{\overline{V} - \beta} - \frac{\alpha}{\overline{V}\left(\overline{V} + \beta\right) + \beta\left(\overline{V} - \beta\right)} \quad \text{Peng} - \text{Robinson} \tag{13}$$

1. Write these equations as cubic equations in $\overline{V}$.

2. A liquid-vapour critical point, $\left(\overline{V}_C, \overline{P}_C, \overline{T}_C\right)$, designates conditions under which a liquid and its vapor can coexist. For any temperature below $\overline{T}_C$ liquid and vapour can coexist for more than one value of $\overline{V}$. At $T = T_C$ the above three equations (in the form of cubic equations in $\overline{V}$), will have a real root with multiplicity 3. The value of this root will give $\overline{V}_C$. Using these information, derive expressions for $\overline{V}_C, \overline{P}_C$, and $\overline{T}_C$, for all three equations of states, as a function of the constants entering the equations.

3. Compressibility factor of a gas is given by $Z = P\overline{V}/RT$. The constants van der Waals constants for the gases N_2, CH_4, and He are listed in the following table. Using these values, compute Z for various values of P (in bar), at $T = 300$ K. Plot the Z vs. P data as three curves (with appropriate units) and report your observations.

gas	a ($dm^6 \cdot atm \cdot mol^{-2}$)	b ($dm^3 \cdot mol^{-1}$)
N_2	1.3483	0.038577
CH_4	2.2725	0.043067
He	0.034145	0.023733

Exercise 4.2: H_2^+

H_2^+ is the simplest of all molecules comprising of two protons bind together by a single electron. With in Born-Oppenheimer approximation, the Hamiltonian operator for this molecule, in atomic units, is given by

$$\hat{H} = -\frac{1}{2}\nabla^2 - \frac{1}{r_A} - \frac{1}{r_B} + \frac{1}{R},$$

where r_A and r_B are distances of the electron from nuclei A and B, while R is the internuclear separation. In the above formula, the last term on the right side, $1/R$, corresponds to the repulsion energy between the two protons. To calculate the ground state electronic energy of this molecule, we can use the variational theorem. To begin with, let us expand the ground state (i.e., bonding molecular orbital) as the linear combination

$$\psi_{\text{bond}} = c_1 1s_A + c_2 1s_B$$

where, by symmetry, $c_1 = c_2 = 1/\sqrt{2}$, while $1s_A$ and $1s_B$ are hydrogen atomic orbitals centered on nuclei A and B. The energy is computed as the expectation value

$$E_{\text{bond}} = \frac{\int d\tau\ \psi^*_{\text{bond}} \hat{H} \psi_{\text{bond}}}{\int d\tau\ \psi^*_{\text{bond}} \psi_{\text{bond}}},$$

where the integration is typically performed in elliptical coordinates:

$$\int d\tau = \frac{R^3}{8} \int_1^\infty du \int_{-1}^{1} dv \int_0^{2\pi} d\phi \quad u^2 - v^2$$

After plugging in the function $1s_X = \sqrt{\zeta^3/\pi} \exp(-\zeta r_X)$ (where $X = A, B$), and some manipulations, the expression for energy simplifies to

$$E_{\text{bond}} = \frac{J + K}{1 + S} + \frac{1}{R},$$

where $J(\zeta, R) = \zeta^2/2 - \zeta - 1/R + (\zeta + 1/R)\exp(-2\zeta R)$, $K(\zeta, R) = -\zeta^2 S(\zeta, R)/2 - \zeta(1 + \zeta R)(2 - \zeta)\exp(-\zeta R)$, and finally $S(\zeta, R) = (1 + \zeta R + (\zeta R)^2/3)\exp(-\zeta R)$. Now the optimal value of ζ and R can be computed by a two-dimensional minimization of the function $E_{\text{bond}}(\zeta, R)$. The corresponding energy at the optimal value of ζ and R is the electronic energy of H_2^+. If the energy of a H atom is subtracted from E_{bond} one arrives at the bond energy ΔE_{bond}.

1. Report ζ_{opt} , R_{opt}, ΔE_{bond} and compare these values with their exact non-relativistic counterparts, $R^{\text{exact}}_{\text{opt}} = 105.69$ pm and $\Delta E^{\text{exact}}_{\text{bond}} = -0.1026$ hartree.

2. For various fixed values of R in the range 1.0 to 10 bohr, minimize E_{bond} by optimizing ζ. Plot the resulting ΔE_{bond} as a function of R. Explain the features of this plot.

7. GNUPLOT

Gnuplot is a robust open source program for visualizing data. Here let us look at some elementary examples.

7.1 Basics

Let us start with simple one-line scripts that demonstrate various things that one can do with gnuplot. To start gnuplot, type `gnuplot` in your terminal. In the gnuplot terminal you will see the prompt `gnuplot>` where you can try the following commands (and note down your observations), where the prompt is denoted by >. The line `Terminal type set to 'x11'` means that the display will be shown in the graphical user interface (GUI).

```
> # comment
> plot x**2
> plot sin(x)
> plot x**2, sin(x)
> plot [-5:5][-1:5] x**2, sin(x)
> save "test001.gp"
> exit
```

Inspect the content of the file "`test001.gp`".

7.2 Data from a file

Instead of using analytic functions, if you want to use x vs. $f(x)$ data stored in a file, the syntax for the plot command is

```
> plot "datafile.txt" using 1:2
```

where the syntax `1:2` implies plotting column-2 as a function of column-1. If you wish to add a constant 5.0 to $f(x)$,try

```
> plot "datafile.txt" using 1:($2+5.0)
```

The file "`datafile.txt`" is assumed to have entries listed in the format

```
Filename:  datafile.txt
# x f(x)
1.00 1.00
2.00 4.00
3.00 9.00
...
```

7.3 *The load function*

The `load` function lets you load a script contaning commands. Enter the following one-line command in a file called "`test002.gp`".

```
plot x**2, sin(5*x)
```

Now `load` the script from gnuplot as

```
> load "test002.gp"
```

7.4 *The call function*

The `call` function is more flexible than the `load` function. The `call` function lets you pass up to 10 arguments to a script. Enter the following one-line command in a file called "`test003.gp`".

```
plot x**$0, sin($1*x)
```

Now call the function from gnuplot as

```
> call "test003.gp" 2 5
```

The arguments are referred in the commands as `$0, $1,..., $9`.

From now on, let us assume that the commands are stored in a file and is "loaded" or "called". So, I will leave out ">", the symbol for the prompt.

7.5 *Script file, example 001*

This example shows how to make simple plots (by executing commands from a file) and save the output in a png file.

```
Filename:  example_001.gp
```

```
Execution:  gnuplot> load "example_001.gp"
or
Execution from bash:  gnuplot example_001.gp
```

```
set terminal png
set output "example_001.png"
set xrange [-5:5]
set yrange [-1:5]
plot x**2, sin(x)
```

To plot the output in a file, as well as to display it on the GUI, add the following three lines at the end of `example_001.gp`.

```
set output
set terminal x11
replot
```

This example script can be made recycleable by including variables via `$0, $1,..., $9`.

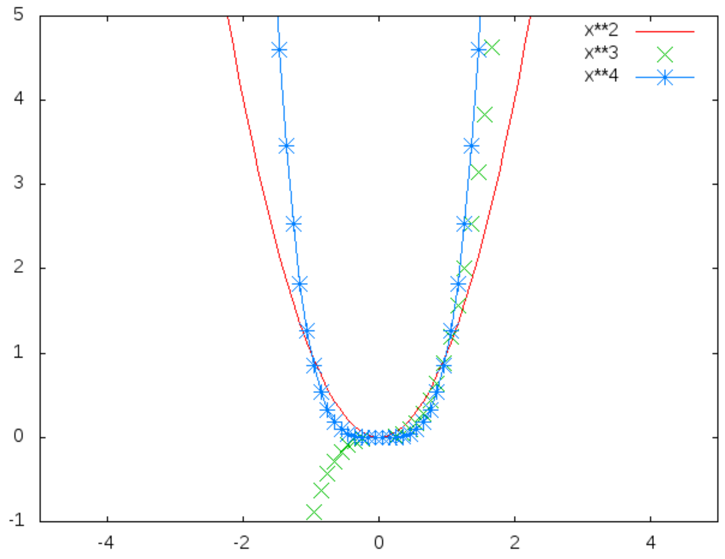
```
Filename:  example_001_2.gp
Execution:  gnuplot> call "example_001_2.gp" st_line 1
Execution:  gnuplot> call "example_001_2.gp" parabola 2
set terminal png

# dynamically define output name
set output "$0.png"

set xrange [-5:5]
set yrange [-1:5]
# to enter multiple commands in a single line
# separate them by semicolon ";"
# xrange [-5:5]; set yrange [-1:5]

# dynamically define function
plot x**$1
```

7.6 *Color & Style - lines, points*

To plot multiple curves with different styles, replace the command `plot x**2` in `example_001.gp` by the following line.

```
plot x**2 with lines, x**3 with points, x**4 with
linespoints
```

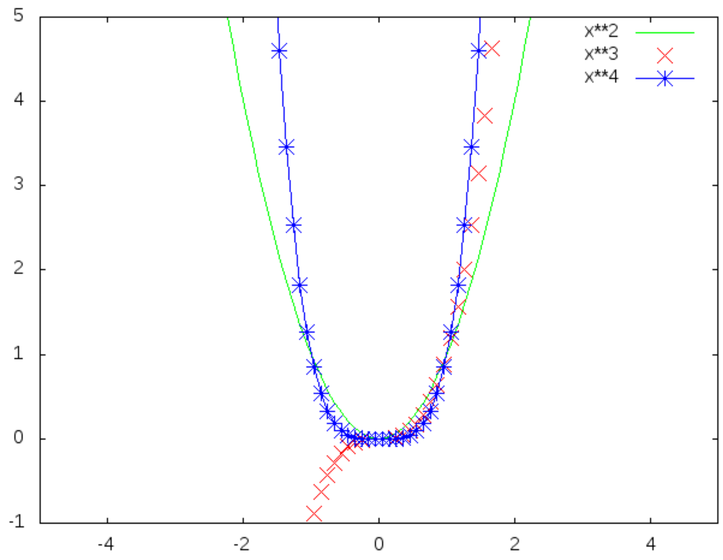
Executing the resulting `example_001.gp` script file with gnuplot produces the following output



The colors of the lines and points can be controlled as follows

```
plot x**2 with lines linecolor “green”, \
x**3 with points linecolor “red”, \
x**4 with linespoints linecolor “blue”
```

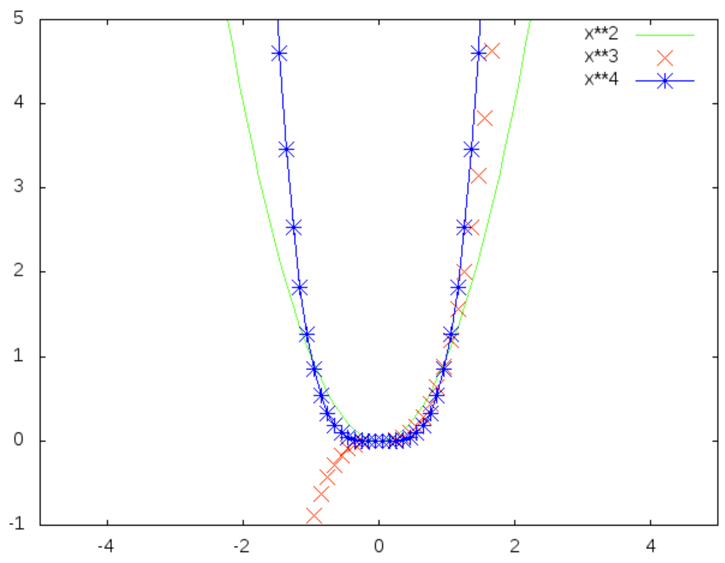
The symbol “\” enables splitting a single-line command to multiple lines. Overall, this modification of the `plot` command produces the following output



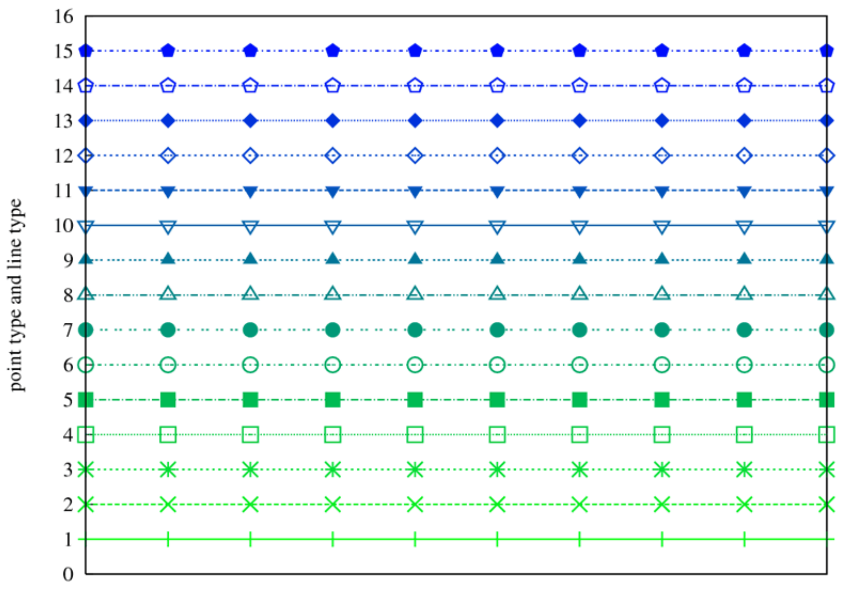
Colours can also be specifed using the HTML value for a colour in hexadecimal form. Here is an online colour picker to choose various colors: http://www.w3schools.com/colors/colors_picker.asp. When using these values, the above `plot` command can be modified as

```
plot x**2 with lines linecolor “#66ff33”, \
x**3 with points linecolor “#ff3300”, \
x**4 with linespoints linecolor “#0000ff”
```

This produces the following output



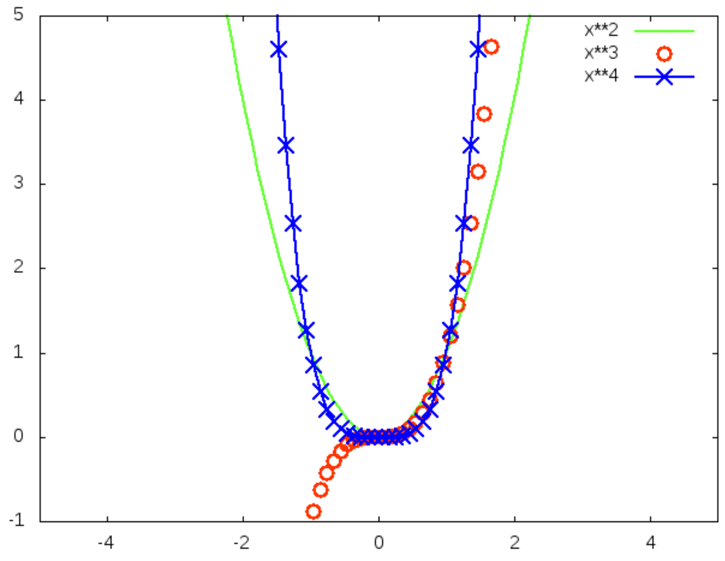
Line width, point size, and line/point styles can also be controlled. The following chart presents the point and line types available in gnuplot



The usage is demonstrated in the following block

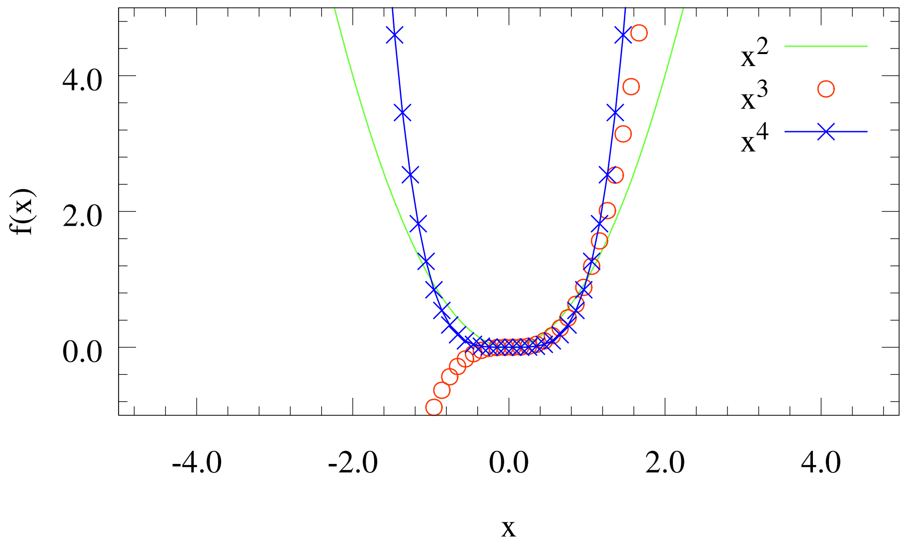
```
set pointsize 2
plot x**2 with lines linetype 1 linewidth 2 linecolor
"#66ff33", \
x**3 with points pointtype 6 linewidth 2 linecolor
"#ff3300", \
x**4 with linespoints linetype 2 linewidth 2 linecolor
"#0000ff"
```

This produces the following output



7.7 *Font, format, labels, tics, line titles*

Here is a more complete example to generate the following plot

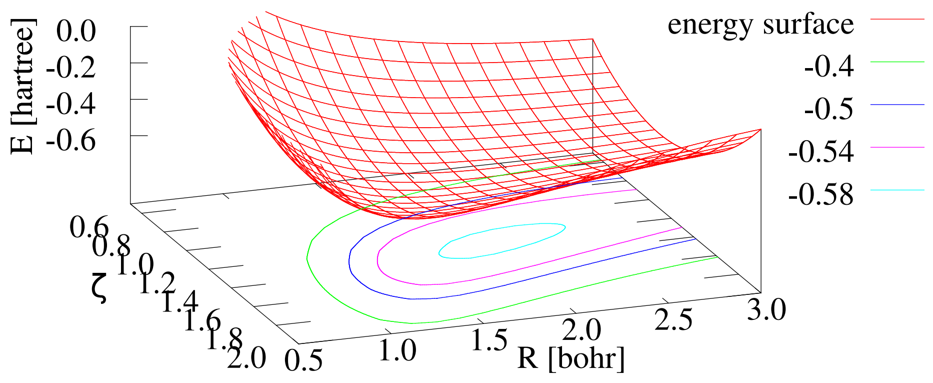


```
File:  example_002.gp
set terminal eps enhanced font 'Times,18'
set output "example_002.eps"

set xrange [-5:5]
set yrange [-1:5]

set xlabel "x"
set ylabel "f(x)"

# tics every 2 units
set xtic 2
set ytic 2
# sub-divide into 5 units
set mxtic 5
set mytic 5

set format x "%2.1f"
set format y "%2.1f"

set pointsize 2

plot x**2 title "x^2" with lines linetype 1 linewidth 2
linecolor "#66ff33", \
x**3 title "x^3" with points pointtype 6 linewidth 2
linecolor "#ff3300", \
x**4 title "x^4" with linespoints linetype 2 linewidth 2
linecolor "#0000ff"
```

Here is a good online reference for advanced commands: http://folk.uio.no/hpl/scripting/doc/gnuplot/Kawano/index-e.html

7.8 Surfaces, contours

Exercise 4.2 describes the method to variationally calculate the ground state electronic energy of the H_2^+ molecule. In this exercise, the energy function $E_{\text{bond}}(\zeta, R)$ is minimized with respect to the variational parameters ζ and R. The following example shows how this 2D surface can be plotted along with its contour projected to the base.



```
File:  example_003.gp
set terminal eps enhanced font 'Times,18'
set output "H2plus.eps"
S(x,y)=(1.0 + x*y + (1.0/3.0)*(x*y)**2) * exp(-x*y)
J(x,y)=(1.0/2.0)*x**2 - x - 1.0/y + ( 1.0/y + x ) *
exp(-2.0*x*y)
K(x,y)=-(1.0/2.0)*S(x,y)*x**2 - x*( 1.0+ x*y
)*(2.0-x)*exp(-x*y)
E(x,y) = (J(x,y)+K(x,y))/(1.0+S(x,y)) + 1.0/y
set xrange [0.5:2]; set yrange [0.5:3]; set zrange
[-0.65:0.0]
set xlabel "{/Symbol z}"
set ylabel "R [bohr]"
set zlabel "E [hartree]" rotate by -270 left
set format x "%2.1f"
set format y "%2.1f"
set format z "%2.1f"
set view 50.0, 70.0
set samples 1000
set isosample 20,20
set xtic 0.2
set ytic 0.5
set ztic 0.2
set key at 1, 4.2
set contour base
set cntrparam levels discrete -0.58, -0.54, -0.50, -0.40
splot E(x,y) title "energy surface" with lines
```

8. Interpolation and polynomial approximation

8.1 Algebraic polynomials

Algebraic polynomials are functions of the form

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n = \sum_{k=0}^{n} a_k x^k, \tag{14}$$

where $n > 0$, and $a_k \in \mathbb{R}$ (real number line). The integer n is called the degree of the polynomial. Polynomials are most commonly used for interpolation because they are easy to

- evaluate
- differentiate and integrate

Recall, that the derivative of a polynomial $P_n(x) = \sum_{k=0}^{n} a_k x^k$ is $P_n'(x) = \sum_{k=1}^{n} a_k x^{k-1}$. Similarly the indefinite integral of a polynomial is also another polynomial, $\int dx \quad P_n(x) = \sum_{k=0}^{n} a_k \frac{x^{k+1}}{k+1}$. These properties enable applying polynomials to derive useful formulas to use with various numerical methods.

8.2.1 Horner's scheme for polynomial evaluation

Direct brute-force evaluation of a polynomial can be very inefficient. The following example code computes a polynomial using $n(n-1)/2$ multiplications and n additions.

```
Input:  n, x0, a[0:n]
Output:  poly_bad
Call:  fn = poly_bad(n, x0, a[0:n])
poly = a(0)
do k = 1, n
  poly = poly + a(k) * x0**k
end do
```

When evaluating x^k as $x^{k-1}x$, and multiplying with the corresponding a_k, the number of FLOPs can be decreased to $3n - 1$, resulting from $2n - 1$ multiplications and n additions. This scheme is illustrated in the following code.

```
Input:  n, x0, a[0:n]
Output:  poly_better
Call:  fn = poly_better(n, x0, a[0:n])
poly = a(0) + a(1) * x0
xk = x0
do k = 2, n
  xk = xk * x0
  poly = poly + a(k) * xk
end do
```

Horner's scheme to evaluate a polynomial, demonstrated in the following code, can further decrease the flop count to $2n$.

```
Input:  n, x0, a[0:n]
Output:  poly_horner
Call:  fn = poly_horner(n, x0, a[0:n])
poly = a(n)
do k = n-1, 0
  poly = poly * x0 + a(k)
end do
```

8.2 Weierstrass approximation theorem

Suppose a function f is defined and is continuous in the closed interval [a,b]. For any $\varepsilon > 0$, there exists a polynomial $P(x)$, with the property that

$$|f(x) - P(x)| < \varepsilon, \qquad \forall x \in [a,b] \tag{15}$$

8.3 Taylor polynomials

Taylor polynomials, expanded around a point x_0 are defined as

$$P_n^{\text{Taylor}}(x) = \sum_{k=0}^{n} \frac{f^k(x_0)}{k!} (x - x_0)^k \tag{16}$$

Taylor polynomials concentrate their accuracy near the point x_0. Their accuracy away from x_0 need not converge with increasing n.

8.4 Lagrange interpolating polynomials

Lagrange interpolating polynomials, or shortly Lagrange polynomials are class of functions that are more suited when the value of the function which we want to approximate as a polynomial is known for some abscissae (i.e., for some x values).

$$\begin{aligned} P_n^{\text{Lagrange}}(x) &= \sum_{k=0}^{n} f(x_k) L_{n,k}(x) \\ &= \sum_{k=0}^{n} f(x_k) \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i} \end{aligned} \tag{17}$$

The Lagrange polynomial of degree n, i.e. $P_n^{\text{Lagrange}}(x)$, is the unique polynomial of degree at most n, that passes through $n+1$ points, namely, $(x_0, f(x_0)) \ldots (x_n, f(x_n))$.

8.4.1 Case $n = 1$, two-point interpolation

For the special case of $n = 1$, i.e., with two points at $(x_0, f(x_0))$ and $(x_1, f(x_1))$, we know that this unique polynomial of degree 1 must be a straight line. Let us prove this in the following.

The equation of a straight line passing through two points (x_0, y_0) and (x_1, y_1) is given by

$$\frac{y - y_1}{y_0 - y_1} = \frac{x - x_1}{x_0 - x_1}, \tag{18}$$

which on rearranging results in

$$\begin{aligned} y &= (y_0 - y_1)\frac{x - x_1}{x_0 - x_1} + y_1 \\ &= y_0 \frac{x - x_1}{x_0 - x_1} + y_1 - \frac{x - x_1}{x_0 - x_1} y_1 \\ &= y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \left[1 - \frac{x - x_1}{x_0 - x_1}\right] \\ &= y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \left[\frac{x_0 - x}{x_0 - x_1}\right] \\ &= y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \frac{x - x_0}{x_1 - x_0} \end{aligned} \tag{19}$$

Expanding P_1^{Lagrange}, we arrive at

$$\begin{aligned} f(x) \approx P_1^{\text{Lagrange}}(x) &= \sum_{k=0}^{1} f(x_k) \prod_{i=0, i\neq k}^{1} \frac{x - x_i}{x_k - x_i} \\ &= f(x_0)\frac{x - x_1}{x_0 - x_1} + f(x_1)\frac{x - x_0}{x_1 - x_0}, \end{aligned} \tag{20}$$

8.5 Runge's Phenomenon

When a function f is approximated by interpolation in any interval, the error at the interpolation abscissae is zero, but at other points, the truncation error need not decrease with increasing degree of the interpolating polynomial (n). In fact, there is no guarantee that a good approximation is possible with a high degree polynomial and suitably sampled interpolation points. In can be shown that for any choice of interpolation points one can find a function for which error in polynomial interpolation tends to infinity.

Exercises - 5

Exercise 5.1: Horner with derivative

Write a Fortran90 subroutine which takes as input $\{a_k\}$ and x_0 (a value for x), and returns the value of the polynomial, along with its derivative evaluated at x_0. Make sure that both $P_n(x)$ and $P_n'(x)$ are evaluated using the Horner's scheme. Comment on the number of FLOPs.

Exercise 5.2: Taylor vs. Lagrange

Expand the functions $f(x) = \exp(x)$ and $f(x) = 1/x$ about $x_0 = 1$, with Taylor polynomials of degrees $n = 0, \ldots, 3$.

1. In two separate plots, display the given functions along with their Taylor approximations.
2. Comment on how Taylor polynomials with increasing n approximate $f(x)$ at $x = 3$.
3. Using the abscissae $x = 1, 2, 5/2, \quad \text{and} \quad 4$, and the values of $f(x) = \exp(x)$ and $f(x) = 1/x$ at these points, expand both these functions as P_3^{Lagrange}. Using this polynomial estimate the function value at $x = 3$. Report relative errors, and compare with those obtained using P_3^{Taylor}.

Exercise 5.3: Runge's phenomenon

Approximate the function $f(x) = 1/(1 + 25x^2)$ in the region $[-1, 1]$ by P_n^{Lagrange} with n values 2, 4, 8, and 16. To choose the interpolation abscissae use the formula

$$x_i = -1 + \frac{2}{n} i, \qquad i = 0, 1, \ldots, n.$$

Make a plot using Gnuplot with xrange $[-1, 1]$, displaying the function, and the four interpolating polynomials. Include the gnuplot script as a part of your solution.

9. Variational solutions to the time-independent Schrödinger equation

9.0 Notations

- Exact eigenfunction (which by definition is normalized) of a Hamiltonian is $\Psi(\tau)$
  - corresponding eigenvalue is $E = \int d\tau \, \Psi^\dagger(\tau)\hat{H}\Psi(\tau)$, hence $\hat{H}\Psi(\tau) = E\Psi(\tau)$
- Trial function, $\widetilde{\Psi}(\tau)$ (in general, unnormallized)

$$\widetilde{E} = \frac{\int d\tau \, \widetilde{\Psi}^\dagger(\tau)\hat{H}\widetilde{\Psi}(\tau)}{\int d\tau \, \widetilde{\Psi}^\dagger(\tau)\widetilde{\Psi}(\tau)} \tag{21}$$

- Ground state wavefunction and energy: $\Psi_0(\tau)$, E_0
  - their approximation by a trial function: $\widetilde{\Psi}_0(\tau)$, $\widetilde{E}_0$
  - wavefunction and energy of the first excited state: $\Psi_1(\tau)$, E_1
  - Note that the subscript usually corresponds to a quantum number, and the lowest quantum number is not always 0. For the particle-in-a-box the quantum number begins from 1, where as for a Harmonic oscillator it begins from 0. So you can denote the ground state wavefunction as Ψ_1 for the particle-in-a-box, and Ψ_0 for a harmonic oscillator.
- Basis functions, $\phi_j(\tau)$; $\qquad j = 1, \ldots, N_b$
- Matrix elements
  - $O_{j,k} = \int d\tau \, \phi_j^\dagger(\tau)\hat{O}\phi_k(\tau)$, for any operator $\hat{O}$
  - Overlap, $S_{j,k} = \int d\tau \, \phi_j^\dagger(\tau)\phi_k(\tau)$ here the operator is simply the unit operator, $\hat{I}\phi_k = \phi_k$.
  - Kinetic energy of one-particle, $T_{j,k} = \int d\tau \, \phi_j^\dagger(\tau)\left(-\frac{\hbar^2}{2m}\nabla^2\right)\phi_k(\tau)$, where m is the mass of this particle. In the Harmonic oscillator problem, m corresponds to the reduced mass of the oscillator.
  - Potential energy, $V_{j,k} = \int d\tau \, \phi_j^\dagger(\tau)\hat{V}(\tau)\phi_k(\tau)$
  - Hamiltonian, $H_{j,k} = T_{j,k} + V_{j,k}$

9.1 *Variational method*

9.1.1 *Ritz theorem*

For a variational trial function $\widetilde{\Psi}$, the Ritz theorem states that as

$$\widetilde{E}_0 = \frac{\int d\tau\, \widetilde{\Psi}^\dagger(\tau)\hat{H}\widetilde{\Psi}(\tau)}{\int d\tau\, \widetilde{\Psi}^\dagger(\tau)\widetilde{\Psi}(\tau)} \geq E_0, \tag{22}$$

where the equality holds only when the trial function is equal to the lowest eigenfunction of the Hamiltonian. The corresponding eigenvalue E_0 is the ground state energy of the system. For an arbitrary trial function, $\widetilde{E}_0$ is an upper bound.

9.2 *Linear variational problem*

In a linear variational problem, the trial function will depend linearly on variational parameters. In practice, this problem is posed as finding an optimal wavefunction that is a linear combination of fixed basis functions.

$$\widetilde{\Psi}(\tau) = \sum_{j=1}^{N_b} c_j\phi_j(\tau) \tag{23}$$

Note that the variational parameters are now the expansion coefficients $\{c_j\}$. Let us work with the assumption that the basis functions are orthonormal.

$$\int d\tau\, \phi_j^*(\tau)\phi_k(\tau) = \delta_{jk}^{\text{Kronecker}} = \begin{array}{ll} 1 & \text{if} \quad j = k \\ 0 & \text{if} \quad j \neq k \end{array} \tag{24}$$

9.2.1 *From operators/wavefunctions to matrices/vectors*

Let us briefly look at the notations for representation

$$\begin{aligned}
\int d\tau\, \phi_k^*(\tau)\widetilde{\Psi}(\tau) &= \int d\tau\, \phi_k^*(\tau)\sum_{j=1}^{N_b} c_j\phi_j(\tau) \\
&= \sum_{j=1}^{N_b} c_j \int d\tau\, \phi_k^*(\tau)\phi_j(\tau) \\
&= \sum_{j=1}^{N} c_j\delta_{j,k} \\
&= c_k
\end{aligned}$$

which is the projection of the trial wavefunction along the $k-$th basis function. So, the projection of the wavefunction along all N_b functions is a vector (arbitrarily chosen as the column vector).

$$\int d\tau \begin{bmatrix} \phi_1^*(\tau) \\ \phi_2^*(\tau) \\ \vdots \\ \phi_{N_b}^*(\tau) \end{bmatrix} \widetilde{\Psi}(\tau) = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{N_b} \end{bmatrix} = \mathbf{c} \qquad (25)$$

This equation is nothing but writing that

$$\begin{aligned} \widetilde{\Psi}(\tau) &= \begin{bmatrix} \phi_1(\tau) & \phi_2(\tau) & \dots & \phi_{N_b}(\tau) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{N_b} \end{bmatrix} && (26) \\ &= \mathbf{\Phi c} && (27) \end{aligned}$$

and

$$\int d\tau \begin{bmatrix} \phi_1^*(\tau) \\ \phi_2^*(\tau) \\ \vdots \\ \phi_{N_b}^*(\tau) \end{bmatrix} \widetilde{\Psi}(\tau) = \int d\tau\, \mathbf{\Phi}^\dagger \mathbf{\Phi c} = \begin{bmatrix} \int d\tau\, \phi_1^\dagger(\tau)\phi_1(\tau) & \dots \\ \vdots & \ddots \\ & \end{bmatrix} \mathbf{c} = \mathbf{c} \qquad (28)$$

Now let us see how to represent an expectation value.

$$\begin{aligned} \int d\tau\, \widetilde{\Psi}^\dagger(\tau)\hat{O}\widetilde{\Psi}(\tau) &= \int d\tau\, \mathbf{c}^\mathrm{T}\mathbf{\Phi}^\mathrm{T}\hat{O}\mathbf{\Phi c} && (29) \\ &= \mathbf{c}^\mathrm{T}\left[\int d\tau\, \mathbf{\Phi}^\mathrm{T}\hat{O}\mathbf{\Phi}\right]\mathbf{c} && (30) \\ &= \mathbf{c}^\mathrm{T} \begin{bmatrix} \int d\tau\, \phi_1^\dagger(\tau)\hat{O}\phi_1(\tau) & \dots \\ \vdots & \ddots \\ & \end{bmatrix} \mathbf{c} && (31) \\ &= \mathbf{c}^\mathrm{T} \begin{bmatrix} O_{11} & \dots & O_{1N} \\ \vdots & \ddots & \\ O_{N1} & & O_{NN} \end{bmatrix} \mathbf{c}. && (32) \end{aligned}$$

9.2.2 Connection to eigenvalue problem

The linear variational problem is to find the optimal set of linear expansion coefficients, $\{c_j\}$, which minimizes the energy functional. We should also note that this becomes a constrained minimization problem, because not all the N_b coefficients are independent. From the normalization constraint, $\int d\tau\, \widetilde{\Psi}(\tau)^\dagger\widetilde{\Psi}(\tau) = 1$, we should be able to find the N_b-th coefficient, if we know the other $N_b - 1$ coefficients. This constrained optimization problem can be tackled using Lagrange's method of undetermined multipliers.

Let us define the Lagrangian function as the energy expectation value minus the normalization constraint multiplied by a constant number E, which is the Lagrange multiplier. For convenience, let us work with real valued trial function.

$$\begin{aligned}
\mathcal{L}\left(c_1,\ldots,c_{N_b};E\right) &= \int d\tau\,\widetilde{\Psi}(\tau)^{\dagger}\hat{H}\widetilde{\Psi}(\tau) - \\
&\quad E\left(\int d\tau\,\widetilde{\Psi}(\tau)^{\dagger}\widetilde{\Psi}(\tau)-1\right) \qquad (33)\\
&= \int d\tau\,\left[\sum_{j=1}^{N_b} c_j\phi_j(\tau)\right]\hat{H}\sum_{k=1}^{N_b} c_k\phi_k(\tau) - \\
&\quad E\left(\int d\tau\,\left[\sum_{j=1}^{N_b} c_j\phi_j(\tau)\right]\sum_{k=1}^{N_b} c_k\phi_k(\tau)-1\right)\\
&= \sum_{j=1}^{N_b}\sum_{k=1}^{N_b} c_jc_kH_{j,k} - E\left(\sum_{j=1}^{N_b}\sum_{k=1}^{N_b} c_jc_k\delta_{j,k}-1\right)\\
&= \sum_{j=1}^{N_b}\sum_{k=1}^{N_b} c_jc_kH_{j,k} - E\left(\sum_{j=1}^{N_b} c_jc_j-1\right) \qquad (34)
\end{aligned}$$

Let us take the partial derivative of both sides w.r.t. c_k which we will later equate to zero to solve for stationary values of c_k.

$$\begin{aligned}
\frac{\partial}{\partial c_q}\mathcal{L}\left(c_1,\ldots,c_{N_b};E\right) &= \frac{\partial}{\partial c_q}\left[\sum_{j=1}^{N_b}\sum_{k=1}^{N_b} c_jc_kH_{j,k} - E\left(\sum_{j=1}^{N_b} c_j^2-1\right)\right]\\
&= \sum_{k=1}^{N_b} c_kH_{q,k} + \sum_{j=1}^{N_b} c_jH_{j,k} - 2Ec_q\\
&= 2\sum_{j=1}^{N_b} c_jH_{q,j} - 2Ec_q \qquad (35)
\end{aligned}$$

where we have used the fact that our Hamiltonian matrix is symmetric, $H_{j,k} = H_{k,j}$. Now

$$\begin{aligned}
\frac{\partial}{\partial c_q}\mathcal{L}\left(c_1,\ldots,c_{N_b};E\right) &= 0 \qquad (36)\\
\Rightarrow 2\sum_{j=1}^{N_b} c_jH_{q,j} - 2Ec_q &= 0\\
\sum_{j=1}^{N_b} H_{q,j}c_j &= Ec_q \qquad (37)
\end{aligned}$$

The last equation can be generalized for all values of q, i.e., by replacing the single element c_q by a column vector $\mathbf{c}$:

$$\begin{aligned}
\mathbf{Hc} &= E\mathbf{c} \qquad (38)\\
\Rightarrow \mathbf{c}^{\mathrm{T}}\mathbf{Hc} &= E. \qquad (39)
\end{aligned}$$

Thus solving for an N_b-dimensional normalized vector which minimizes the energy expectation value amounts to finding the eigenvectors of the Hamiltonian matrix. The proof can be continued for excited states by including further constraints such as the first excited state must be orthogonal to the ground state, etc., finally arriving at

$$\begin{aligned} \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_{N_b} \end{bmatrix}^{\mathrm{T}} \mathbf{H} \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_{N_b} \end{bmatrix} &= \mathbf{E} \text{ (diagonal)} \\ \mathbf{C}^{\mathrm{T}}\mathbf{H}\mathbf{C} &= \mathbf{E}. \end{aligned} \tag{40}$$

9.3 *Finite-basis representation*

An arbitrary trial function can be expanded as linear combinations of any complete set of orthonormal basis functions. Let us call the set of all basis functions as the basis set. In principle, a complete set comprises of infinite functions

$$\widetilde{\Psi}(x) = \sum_{j=1}^{\infty} c_j \phi_j(x). \tag{41}$$

However, in practice, we cannot work with infinite number of basis functions, and we will have to truncate the trial function expansion with a finite number of basis functions.

$$\widetilde{\Psi}(x) = \sum_{j=1}^{N_b} c_j \phi_j(x) \tag{42}$$

We can increase N_b until all the expectation values of interest converge, i.e., $\langle \hat{O} \rangle$ does not change up to required precision, when N_b is increased by 1.

9.3.1 *Particle-in-a-box eigenfunctions as orthonormal basis functions for the Harmonic oscillator problem*

For one-dimensional problems, such as harmonic oscillator or Morse potentials, we can use the lowest few eigenfunctions of the particle-in-a-box as basis functions.

$$\phi_n(x) = \sqrt{\frac{2}{L}} \sin\left(\frac{n\pi x}{L}\right) \quad \text{and} \quad n = 1, 2, \dots, N_b. \tag{43}$$

These basis functions are by definition confined to the region $0 \leq x \leq L$.

For a potential energy function, centered at $x = 0$, we can use as basis functions, the eigenstates of a particle-in-a-box confined in the range $-L \leq x \leq L$.

$$\phi_n(x) = \sqrt{\frac{1}{L}} \sin\left[\frac{n\pi(x-L)}{2L}\right] \quad \text{and} \quad n = 1, 2, \dots, N_b. \tag{44}$$

Let us now collect various matrix elements, in this representation.
Kinetic energy:

$$\begin{aligned} T_{i,j} &= \int_{-L}^{L} dx\, \phi_i^\dagger(x) \left(-\frac{\hbar^2}{2m} \frac{d^2}{dx^2} \right) \phi_j(x) \\ &= 0; \quad i \neq j & (45) \\ &= \frac{1}{8m} \left(\frac{i\pi\hbar}{L} \right)^2; \quad i = j & (46) \end{aligned}$$

Harmonic potential energy:

$$\begin{aligned} V_{i,j}^{\text{harmonic}} &= \int_{-L}^{L} dx\, \phi_i^\dagger(x) \left(\frac{1}{2} k x^2 \right) \phi_j(x) \\ &= \left(\frac{L}{(i^2 - j^2)\pi} \right)^2 8kij \left[1 + (-1)^{i+j} \right]; \quad i \neq j & (47) \\ &= \left(\frac{L}{i\pi} \right)^2 \left(\frac{k\pi^2 i^2}{6} - 1 \right); \quad i = j & (48) \end{aligned}$$

Once the Hamiltonian matrix is built, as $H_{i,j} = T_{i,j} + V_{i,j}$, eigenvector/eigenvalue pairs can be computed via matrix-diagonalization. The code-handout shows how the entire procedure can be programmed by calling the eigensolver, DSYEV, from the Lapack library. Note that DSYEV stands for Double precision-SYmmetric-EigenValue.

The following table demonstrates the convergence of the lowest few eigenvalues with increasing N_b. This table provides the numerical proof for the variational theorem, i.e., we should see the trend

$$E_0^{N_b=1} > E_0^{N_b=2} > \ldots > E_0^{N_b=\infty}.$$

But in contrast, what we find from the data is

$$E_0^{N_b=2} > \ldots > E_0^{N_b=\infty}$$

and fortuitously, $E_0^{N_b=1} = E_0^{N_b=2}$, which can be explained by a simple analysis. The first basis function ϕ_1 is an even function, while the second function, ϕ_2, is an odd function. Since we are targetting only the ground state with even symmetry, we should only inspect how E_0 converges with increase in the number of even basis functions.

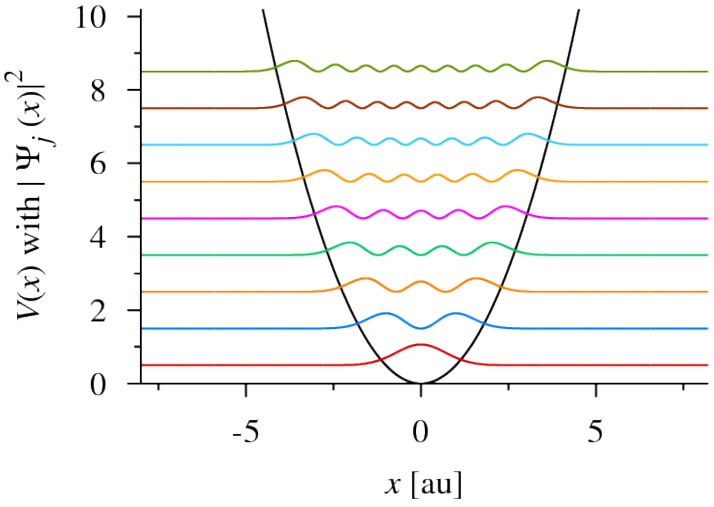
The Fortran90 program also generates the file `'pot.dat'` containing the value of the potential $V(x)$ along with the probability densities of the lowest few states for discrete values of x. For every value of x, the value of an eigenfunction is computed as

$$\Psi_j(x) = \sum_{k=1}^{N_b} C_{k,j} \phi_k(x) \tag{49}$$

E_i	$N_b = 1$	$N_b = 2$	$N_b = 4$	$N_b = 8$	$N_b = 16$	$N_b = 32$
E_0	6.5468	6.5468	2.2409	0.7712	0.5026	0.5000
E_1		14.1829	6.1425	2.4244	1.5159	1.5000
E_2			19.9578	6.1599	2.6561	2.5000
E_3			24.2712	8.7798	3.8151	3.5000
E_4				16.6123	5.6128	4.5000

Table 2: Convergence of eigenvalues as a function of number of basis functions, N_b. Energies of only the lowest 5 eigenstates are shown.

and the probability densities are compued as $|\Psi_j(x)|^2$. The following plot has been generated using Gnuplot using the data from the file 'pot.dat'.



9.4 Fortran90 code:

Separate handout.

9.5 Löwdin orthogonalization

In case a basis set is not orthonormal,

$$\int d\tau\, \phi_j^\dagger(\tau)\phi_k(\tau) = S_{j,k}, \tag{50}$$

one should reformulate the working-equations. One can start from orthonormalizing the basis set through a simple procedure called Löwdin orthogonalization. The key quantity here is the overlap matrix.

$$\begin{aligned} \int d\tau\, \widetilde{\Psi}^\dagger(\tau)\widetilde{\Psi}(\tau) &= 1 & (51)\\ \Rightarrow \int d\tau\, \mathbf{c}^{\mathrm{T}}\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{c} &= \mathbf{c}^{\mathrm{T}}\mathbf{S}\mathbf{c} & (52)\end{aligned}$$

Note that the task now is to choose the coefficient vector such that the quantity $\mathbf{c}^{\mathrm{T}}\mathbf{S}\mathbf{c}$ becomes $\mathbf{c}^{\mathrm{T}}\mathbf{c} = 1$. Löwdin suggested that one can choose the new vector as

$$\begin{aligned} \widetilde{\mathbf{c}} &= \mathbf{S}^{-1/2}\mathbf{c} && (53) \\ \Rightarrow \widetilde{\mathbf{c}}^{\mathrm{T}}\mathbf{S}\widetilde{\mathbf{c}} &= \mathbf{c}^{\mathrm{T}}\mathbf{S}^{-1/2}\mathbf{S}\mathbf{S}^{-1/2}\mathbf{c} && (54) \\ &= 1 && (55) \end{aligned}$$

The new basis set will satisy the usual orthonormal relations.

9.6 *Function of a matrix*

For the Löwdin orthonormalization, we will have to compute the inverse-square-root of the overlap matrix. Note that, in general

$$\left(S^{-1/2}\right)_{i,j} \neq \left(S_{i,j}\right)^{-1/2}, \tag{56}$$

unless $\mathbf{S}$ is a diagonal matrix. i.e., we cannot evaluate $\mathbf{S}^{-1/2}$, by computing inverse-square-root element-by-element. This fact is true for any function of a non-diagonal matrix

$$(f(S))_{i,j} \neq f\left(S_{i,j}\right). \tag{57}$$

The function of a matrix can be defined in its eigen representation, $\mathbf{a} = \mathbf{V}^{\mathrm{T}}\mathbf{A}\mathbf{V}$, where $\mathbf{a}$ is a diagonal matrix with eigenvalues along the diagonal, and the columns of the matrix $\mathbf{V}$ correspond to the eigenvectors.

$$f(\mathbf{A}) = f\left(\mathbf{V}\cdot\mathbf{a}\cdot\mathbf{V}^{\mathrm{T}}\right) \tag{58}$$

If we assume that the function can be expanded as a series,

$$f(\mathbf{A}) = \sum_{k=1}^{\infty} c_k \mathbf{A}^k \tag{59}$$

we arrive at

$$\begin{aligned} f(\mathbf{A}) &= \sum_{k=1}^{\infty} c_k \left(\mathbf{V}\cdot\mathbf{a}\cdot\mathbf{V}^{\mathrm{T}}\right)^k \\ &= \mathbf{V}\cdot\left(\sum_{k=\mathbf{1}}^{\infty} c_k \mathbf{a}^k\right)\cdot\mathbf{V}^{\mathrm{T}} && (60) \end{aligned}$$

Hence

$$f(\mathbf{A}) = \mathbf{V}\cdot f(\mathbf{E})\cdot\mathbf{V}^{\mathrm{T}} \tag{61}$$

Returning to our original problem, we write

$$\mathbf{S}^{-1/2} = \mathbf{V}\cdot\mathbf{s}^{-1/2}\cdot\mathbf{V}^{\mathrm{T}} \tag{62}$$

where $\mathbf{s}$ is a diagonal matrix, with eigenvalues of $\mathbf{S}$ along the diagonal, and the columns of the matrix $\mathbf{V}$ correspond to the eigenvectors of $\mathbf{S}$.

Exercises - 6

Exercise 6.1: Analytic solution to 2×2 *eigenvalue problem*

We know that diagonalizing a real symmetric matrix to compute the eigenvalue/eigenvector pairs is nothing but performing an orthogonal transformation which renders the matrix diagonal. i.e., in the equation

$$\mathbf{a} = \mathbf{V}^{\mathrm{T}}\mathbf{A}\mathbf{V}$$

$\mathbf{V}$ is an orthogonal matrix (i.e., $\mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I}$). For a 2×2 problem, a general orthogonal matrix can be written as

$$\mathbf{V} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(\theta) & -\cos(\theta) \end{bmatrix}.$$

1. Using the above equation, show that $\mathbf{V}$ is an orthogonal matrix.

2. For a symmetrix 2×2 matrix,

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} \\ A_{12} & A_{22} \end{bmatrix},$$

show that the off-diagonal elements of $\mathbf{a} = \mathbf{V}^{\mathrm{T}}\mathbf{A}\mathbf{V}$ vanish, i.e., $a_{1,2} = a_{2,1} = 0$, at an angle

$$\theta_0 = \frac{1}{2}\tan^{-1}\left(\frac{2A_{12}}{A_{11} - A_{22}}\right).$$

Note that for this value of θ_0 the two columns of $\mathbf{V}$ correspond to the two eigenvectors.

3. Use θ_0 in $\mathbf{a} = \mathbf{V}^{T}\mathbf{A}\mathbf{V}$, and show that the eigenvalues can be analytically written as

$$a_1 = a_{1,1} = A_{1,1}\cos^2(\theta_0) + A_{2,2}\sin^2(\theta_0) + A_{1,2}\sin(2\theta_0)$$

and

$$a_2 = a_{2,2} = A_{1,1}\sin^2(\theta_0) + A_{2,2}\cos^2(\theta_0) - A_{1,2}\sin(2\theta_0).$$

Exercise 6.2: H-atom with STO-nG

The Schrödinger equation (in atomic units) for the hydrogen atom is

$$\left(-\frac{1}{2}\nabla^2 - \frac{1}{r}\right)\Psi(r) = E\Psi(r),$$

where

$$\nabla^2 f(r) = \frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d}{dr}\right)f(r).$$

For this problem the exact wavefunction is $\Psi(r) = \sqrt{\zeta^3/\pi}\exp(-\zeta r)$, where the optimal ζ which minimizes the ground state energy of H atom turns out to be 1. The corresponding ground state energy of H atom is $E = \int_0^\infty dr\, 4\pi r^2\, \Psi^\dagger(r)\, \hat{H}\Psi(r) = -0.5$ hartree.

1. Use a trial function $\widetilde{\Psi}(r) = (2\alpha/\pi)^{3/4}\exp\left(-\alpha r^2\right)$ and minimize the energy expectation value. Report the optimal value of α along with $\widetilde{E}$. Since the trial function varies non-linearly with respect to the variational parameter α, this problem is a non-linear variational problem.

2. Estimate α_{opt} by maximizing the overlap integral, S, between the trial function $\widetilde{\Psi}(r)$ and the exact wavefunction, $\Psi(r) = \sqrt{1/\pi}\exp(-r)$

$$S(\alpha) = \int_0^\infty dr\, 4\pi r^2\, (2\alpha/\pi)^{3/4}\exp\left(-\alpha r^2\right)\cdot\sqrt{1/\pi}\exp(-r).$$

   Report this α_{opt} along with $\widetilde{E}$ computed using the corresponding trial function.
   NOTE: When using α_{opt} this trial function made with one Gaussian approximates best the hydrogen wavefunction (which is also known as the Slater-type orbital, STO). For this reason, $\widetilde{\Psi}(r) = \left(2\alpha_{\text{opt}}/\pi\right)^{3/4}\exp\left(-\alpha_{\text{opt}}r^2\right)$ is known as STO-1G basis function (where 1G stands for one-Gaussian) for H atom.

3. Use a trial function $\widetilde{\Psi}(r) = c_1\phi_1(r) + c_2\phi_2(r)$, where $\phi_j(r) = \left(2\alpha_{\text{j}}/\pi\right)^{3/4}\exp\left(-\alpha_{\text{j}}r^2\right)$ and minimize the energy expectation value as a linear variational problem for fixed $\alpha_1 = 0.151623$ and $\alpha_2 = 0.851819$. Note that the basis functions $\phi_j(r)$ are normalized but not orthogonal to each other.
   NOTE: This trial function made of two Gaussian functions, when used with optimal c_j, approximates best the hydrogen wavefunction. The exponents α_j have been calculated by maximizing the overlap between the trial function $\widetilde{\Psi}(r)$ and the exact wavefunction, $\Psi(r) = \sqrt{1/\pi}\exp(-r)$. Hence this trial function is known as the STO-2G basis function (where 2G stands for two-Gaussians) for H atom. Early seminal electronic structure calculations have

employed the STO-2G function to model H atom in molecules such as H_2O, CH_4, C_6H_6, etc.

Useful integrals:

$$\int_0^\infty dr\, r^{2m} \exp\left(-\alpha r^2\right) = \frac{(2m)!\sqrt{\pi}}{2^{2m+1} m! \alpha^{m+1/2}}$$

$$\int_0^\infty dr\, r^{2m+1} \exp\left(-\alpha r^2\right) = \frac{m!}{2\alpha^{m+1}}$$

10. Numerical derivatives

10.1 Polynomial truncation error

10.1.1 Taylor's theorem

Suppose $f \in C^n\left[a,b\right]$, that $f^{(n+1)}$ exists on $[a,b]$, $x_0 \in [a,b]$. For every $x \in \left[a,b\right]$, there exists a number $\xi\left(x\right)$ between x_0 and x with

$$f(x) = P_n^{\text{Taylor}}(x) + R_n^{\text{Taylor}}(x),$$

where

$$R_n^{\text{Taylor}}(x) = \frac{f^{(n+1)}\left(\xi\left(x\right)\right)}{(n+1)!}\left(x-x_0\right)^{n+1}. \tag{63}$$

The term $R_n^{\text{Taylor}}(x)$ is called the remainder term associated with $P_n^{\text{Taylor}}(x)$.

10.1.2 Remainder for the Lagrange interpolation formula

If we use a Lagrange polynomial of degree n to approximate the function, the remainder term is given by

$$R_n^{\text{Lagrange}}(x) = \frac{f^{(n+1)}\left(\xi\left(x\right)\right)}{(n+1)!}\Pi_{i=0}^{n}\left(x-x_i\right). \tag{64}$$

In the case of both remainder terms, the function ξ need not be identified. All that is necessary to use these terms is to assume the existence of this unspecified function.

10.2 Finite derivatives using Taylor polynomials

Taylor polynomials are typically used to evaluate several arbitrary order derivatives at a given point.

10.2.1 First-order derivative with two points: Forward/backward-difference formulae

Suppose we know the value of a function, f, at two points x_0 and $x_0 - h$, how can we estimate the value of $f^{(1)}$ at the point x_0?

We can start by expanding the function as a Taylor polynomial of degree 1 around $x_0 - h$.

$$\begin{aligned} f(x_0-h) &= f(x_0) - hf^{(1)}(x_0) + \frac{h^2}{2}f^{(2)}\left(\xi\left(x_0\right)\right) + \ldots \\ &\approx f(x_0) - hf^{(1)}(x_0) \\ \Rightarrow f^{(1)}(x_0) &\approx \frac{f(x_0) - f(x_0-h)}{h}. \end{aligned}$$

This is the two-point backward-differerence formula. This short derivation can be presented more compactly in the matrix form

$$\begin{aligned}
\begin{bmatrix} f(x_0) \\ f(x_0-h) \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} f(x_0) \\ hf^{(1)}(x_0) \end{bmatrix} && (65) \\
\begin{bmatrix} f(x_0) \\ hf^{(1)}(x_0) \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix}^{-1} \begin{bmatrix} f(x_0) \\ f(x_0-h) \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} f(x_0) \\ f(x_0-h) \end{bmatrix} && (66)
\end{aligned}$$

Similarly, $f^{(1)}(x_0)$ can also be estimated using $f(x_0)$ and $f(x_0+h)$

$$\begin{aligned}
f(x_0+h) &= f(x_0) + hf^{(1)}(x_0) + \frac{h^2}{2} f^{(2)}(\xi(x_0)) + \ldots \\
&\approx f(x_0) + hf^{(1)}(x_0) \\
\Rightarrow f^{(1)}(x_0) &\approx \frac{f(x_0+h) - f(x_0)}{h},
\end{aligned}$$

which is the two-point forward-differerence formula. This formula can also be expressed in matrix form as follows.

$$\begin{aligned}
\begin{bmatrix} f(x_0) \\ f(x_0+h) \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} f(x_0) \\ hf^{(1)}(x_0) \end{bmatrix} && (67) \\
\begin{bmatrix} f(x_0) \\ hf^{(1)}(x_0) \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} f(x_0) \\ f(x_0+h) \end{bmatrix} && (68)
\end{aligned}$$

10.2.2 Some five-point formulae

Let us see why the matrix notation is so powerful by considering the following five truncated polynomials, where the remainder term has been ignored.

$$\begin{aligned}
f(x_0-2h) &= f(x_0) - 2hf^{(1)}(x_0) + \frac{(2h)^2}{2} f^{(2)}(x_0) - \frac{(2h)^3}{3!} f^{(3)}(x_0) + \frac{(2h)^4}{4!} f^{(4)}(x_0) \\
f(x_0-h) &= f(x_0) - hf^{(1)}(x_0) + \frac{h^2}{2} f^{(2)}(x_0) - \frac{h^3}{3!} f^{(3)}(x_0) + \frac{h^4}{4!} f^{(4)}(x_0) \\
f(x_0) &= f(x_0) \\
f(x_0+h) &= f(x_0) + hf^{(1)}(x_0) + \frac{h^2}{2} f^{(2)}(x_0) + \frac{h^3}{3!} f^{(3)}(x_0) + \frac{h^4}{4!} f^{(4)}(x_0) \\
f(x_0+2h) &= f(x_0) + 2hf^{(1)}(x_0) + \frac{(2h)^2}{2} f^{(2)}(x_0) + \frac{(2h)^3}{3!} f^{(3)}(x_0) + \frac{(2h)^4}{4!} f^{(4)}(x_0)
\end{aligned}$$

Now, how can we solve these equations to find an arbitrary order derivative, $f^{(n)}(x_0)$, say $n = 2$? For this purpose, we can use the matrix version of the same set of equations.

$$\begin{bmatrix} f(x_0-2h) \\ f(x_0-h) \\ f(x_0) \\ f(x_0+h) \\ f(x_0+2h) \end{bmatrix} = \begin{bmatrix} 1 & -2 & \frac{2^2}{2} & -\frac{2^3}{3!} & \frac{2^4}{4!} \\ 1 & -1 & \frac{1}{2} & -\frac{1}{3!} & \frac{1}{4!} \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & \frac{1}{2} & \frac{1}{3!} & \frac{1}{4!} \\ 1 & 2 & \frac{2^2}{2} & \frac{2^3}{3!} & \frac{2^4}{4!} \end{bmatrix} \begin{bmatrix} f(x_0) \\ hf^{(1)}(x_0) \\ h^2f^{(2)}(x_0) \\ h^3f^{(3)}(x_0) \\ h^4f^{(4)}(x_0) \end{bmatrix} \quad (69)$$

which is nothing but a matrix-inversion problem, and be tackled easily using a Lapack subroutine for matrix inversion.

$$\begin{bmatrix} f(x_0) \\ hf^{(1)}(x_0) \\ h^2f^{(2)}(x_0) \\ h^3f^{(3)}(x_0) \\ h^4f^{(4)}(x_0) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ \frac{1}{12} & -\frac{2}{3} & 0 & \frac{2}{3} & -\frac{1}{12} \\ -\frac{1}{12} & \frac{4}{3} & -\frac{5}{2} & \frac{4}{3} & -\frac{1}{12} \\ -2 & 1 & 0 & -1 & 2 \\ 1 & -4 & 6 & -4 & 1 \end{bmatrix} \begin{bmatrix} f(x_0-2h) \\ f(x_0-h) \\ f(x_0) \\ f(x_0+h) \\ f(x_0+2h) \end{bmatrix} \quad (70)$$

From the matrix equation we can easily pull out the expression that we want, for example,

$$f^{(2)}(x_0) = \frac{1}{h^2}\left[-\frac{f(x_0-2h)}{12} + \frac{4}{3}f(x_0-h) - \frac{5}{2}f(x_0) + \frac{4}{3}f(x_0+h) - \frac{f(x_0+2h)}{12}\right]$$

An important extension of this method is to also include the residuals, or the Taylor polynomial truncation errors. We can include these terms explicitly on the left side.

$$\begin{bmatrix} f(x_0-2h) - \frac{(2h)^5}{5!}f^{(5)}(\xi(x_0)) \\ f(x_0-h) - \frac{h^5}{5!}f^{(5)}(\xi(x_0)) \\ f(x_0) \\ f(x_0+h) + \frac{h^5}{5!}f^{(5)}(\xi(x_0)) \\ f(x_0+2h) + \frac{(2h)^5}{5!}f^{(5)}(\xi(x_0)) \end{bmatrix} = \begin{bmatrix} 1 & -2 & \frac{2^2}{2} & -\frac{2^3}{3!} & \frac{2^4}{4!} \\ 1 & -1 & \frac{1}{2} & -\frac{1}{3!} & \frac{1}{4!} \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & \frac{1}{2} & \frac{1}{3!} & \frac{1}{4!} \\ 1 & 2 & \frac{2^2}{2} & \frac{2^3}{3!} & \frac{2^4}{4!} \end{bmatrix} \begin{bmatrix} f(x_0) \\ hf^{(1)}(x_0) \\ h^2f^{(2)}(x_0) \\ h^3f^{(3)}(x_0) \\ h^4f^{(4)}(x_0) \end{bmatrix} \quad (71)$$

$$\begin{bmatrix} f(x_0) \\ hf^{(1)}(x_0) \\ h^2f^{(2)}(x_0) \\ h^3f^{(3)}(x_0) \\ h^4f^{(4)}(x_0) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ \frac{1}{12} & -\frac{2}{3} & 0 & \frac{2}{3} & -\frac{1}{12} \\ -\frac{1}{12} & \frac{4}{3} & -\frac{5}{2} & \frac{4}{3} & -\frac{1}{12} \\ -2 & 1 & 0 & -1 & 2 \\ 1 & -4 & 6 & -4 & 1 \end{bmatrix} \begin{bmatrix} f(x_0-2h) - \frac{(2h)^5}{5!}f^{(5)}(\xi(x_0)) \\ f(x_0-h) - \frac{h^5}{5!}f^{(5)}(\xi(x_0)) \\ f(x_0) \\ f(x_0+h) + \frac{h^5}{5!}f^{(5)}(\xi(x_0)) \\ f(x_0+2h) + \frac{(2h)^5}{5!}f^{(5)}(\xi(x_0)) \end{bmatrix} \quad (72)$$

In the right side of the above equation, the error term $f^{(5)}(\xi(x_0))$ is just a constant number c. Now evaluating $f^{(1)}(x_0)$ yields

$$\begin{aligned} f^{(1)}(x_0) &= \frac{1}{h}\left[\frac{f(x_0-2h)}{12} - \frac{2}{3}f(x_0-h) + \frac{2}{3}f(x_0+h) - \frac{f(x_0+2h)}{12}\right] + \\ &\quad c\frac{1}{h}\left[-\frac{1}{12}\cdot\frac{32}{120} + \frac{2}{3}\cdot\frac{1}{120} + \frac{2}{3}\cdot\frac{1}{120} - \frac{1}{12}\cdot\frac{32}{120}\right]h^5 \\ &= \frac{1}{h}\left[\frac{f(x_0-2h)}{12} - \frac{2}{3}f(x_0-h) + \frac{2}{3}f(x_0+h) - \frac{f(x_0+2h)}{12}\right] - c\frac{h^4}{30} \end{aligned}$$

We can simply state the error to be $\propto \frac{h^4}{30}$ (proportional to) or $\mathcal{O}\left(h^4\right)$ (order of magnitude of).

Note that the error for the second order derivative, using this 5-point formula seemingly vanishes – this is called as *error cancellation*. This does not mean that there is no truncation error in this approximation. We will have to now consider the next higher order truncation error.

$$\begin{aligned} f^{(6)}\left(\xi\left(x_0\right)\right)\frac{1}{h}\left[\frac{1}{12}\cdot\frac{64}{720}+\frac{4}{3}\cdot\frac{1}{720}+\frac{4}{3}\cdot\frac{1}{720}+\frac{1}{12}\cdot\frac{64}{720}\right]h^6 &\propto \frac{h^5}{54} \\ &\approx \mathcal{O}\left(h^5\right. \end{aligned}$$

The truncation error is of a lower order, for this reason central differences with uniform steps are preferred over end-point or forward/backward differences.

10.2.3 *Code*

Here is an interface for the Lapack subroutine for matrix inversion.

```
subroutine inverse_LU(N, MAT, IMAT)

  implicit none

  integer, intent(in) ::  N
  double precision, intent(in) ::  MAT(N,N)
  double precision, intent(out) ::  IMAT(N,N)

  integer ::  info
  double precision ::  work(N)
  integer ::  IPIV(N)

  if (N .eq.  1) then
    IMAT(1,1) = 1.0d0/MAT(1,1)
    return
  endif

  IMAT = MAT

  call dgetrf( N, N, IMAT, N, IPIV, info )
  call dgetri(N, IMAT, N, IPIV, work, N, info)

end subroutine inverse_LU
```

Note that the procedure will fail if a singular matrix is passed to the subroutine.

10.3 Finite derivatives using Lagrange interpolation polynomials

The Lagrange interpolation polynomials are typically used when only few formulae of a specified order deriviate are required (quickly) using few interpolation points. This approach is also very handy for estimating the truncation error in a specified finite derivative formula.

10.3.1 First-order derivatives

Recall that a function passing through n-interpolation-abscissae $\{x_0, x_1, \ldots, x_n\}$ can be approximated by using Lagrange interpolation formula as

$$\begin{aligned} f(x) &= \sum_{k=0}^{n} f(x_k) L_{n,k}(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{i=0}^{n} (x - x_i) \\ &= \sum_{k=0}^{n} f(x_k) \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i} + \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{i=0}^{n} (x - x_i) . \end{aligned}$$

Taking the first derivative of this formula results in

$$\begin{aligned} f^{(1)}(x) &= \sum_{k=0}^{n} f(x_k) L_{n,k}^{(1)}(x) + \prod_{i=0}^{n} (x - x_i) \frac{d}{dx} \left(\frac{f^{(n+1)}(\xi(x))}{(n+1)!} \right) + \\ &\quad \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \frac{d}{dx} \left(\prod_{i=0}^{n} (x - x_i) \right) . \end{aligned}$$

If the derivative is computed at one of the interpolation abscissae, this formula simplifies to

$$f^{(1)}(x_q) = \sum_{k=0}^{n} f(x_k) L_{n,k}^{(1)}(x_q) + \frac{f^{(n+1)}(\xi(x_q))}{(n+1)!} \prod_{i=0, i\neq q}^{n} (x_q - x_i) .$$

Let us use this formula to evaluate $f^{(1)}(x_0)$ using the five points $f(x_0 + 2h)$, $f(x_0 + h)$, $f(x_0)$, $f(x_0 - h)$, and $f(x_0 - 2h)$.

$$\begin{aligned} f^{(1)}(x_0) &= \sum_{k=0}^{4} f(x_k) L_{4,k}^{(1)}(x_0) + \frac{f^{(5)}(\xi(x_q))}{5!} (2h \cdot h \cdot (-h) \cdot (-2h)) \\ &= \sum_{k=0}^{4} f(x_k) L_{4,k}^{(1)}(x_0) + \frac{f^{(5)}(\xi(x_q))}{5!} \left(4h^4\right) \\ &= \sum_{k=0}^{4} f(x_k) L_{4,k}^{(1)}(x_0) + \mathcal{O}\left(\frac{h^4}{30}\right) . \end{aligned}$$

We have quickly obtained the error term which we had previously estimated using Taylor polynomials, and matrix inversion. Evaluating $\sum_{k=0}^{4} f(x_k) L_{4,k}^{(1)}(x)$ is left as an exercise.

Exercises - 7

Exercise 7.1: Arbitrary order derivative

The Taylor polynomial based approach for evaluating arbitrary order central derivatives, based on $2M+1$ points and a $2M$-degree Taylor polynomial, can be written as the matrix equation

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b},$$

where the size of the vectors is $2M+1$, and the size of the matrix is $(2M+1) \times (2M+1)$. Marix and vector elements are defined as

$$b_i = f\left(x_0 + (i - M - 1)h\right),$$

$$x_i = h^{i-1} f^{(i-1)}\left(x_0\right), \qquad \text{and}$$

$$A_{ij} = \frac{(i - M - 1)^{j-1}}{(j-1)!}.$$

1. Using these formulae, along with the Fortran90 subroutine for matrix inversion, compute $\mathbf{C} = \mathbf{A}^{-1}$ for $M = 1, 2, \ldots, 5$, and collect the formulae for $f^{(1)}$ and $f^{(2)}$ with increasing M.
2. Graphically represent these formulae.
3. Numerically verify that the matrix $\mathbf{C}$ has a property that the sum of all elements in each row adds to zero.

Exercise 7.2: Numerical stability

Using the formulae from the previous exercise, compute the first, and second-derivatives of the function $f = x^2$ at $x_0 = 0$ using step sizes $h = 0.1, 0.01, 0.001$. Which combination of formula and step size gives the least error compared to the exact value $f^{(1)}(0) = 0$ and $f^{(2)}(0) = 2$. Using relative error based on exact values, comment on why a certain combination works the best.

Exercise 7.3: Truncation error

Find possible finite-derivative formulae for $f^{(1)}(x_0)$, $f^{(1)}(x_0 + h)$, and $f^{(1)}(x_0 + 2h)$, along with the corresponding truncation error, using the value of a function f at the three points

1. x_0, $x_0 + h$, and $x_0 + 2h$
2. x_0, $x_0 + h$, and $x_0 - h$

Comment on which choice of points is most suited for a given $f^{(1)}(x_0 + mh)$, where $m = 0, 1, 2$.

Exercise 7.4: Derivatives via Lagrange

Find the derivative of the Lagrange funtion $L_{n,k}(x) = \prod_{i=0, i \neq k}^{n} \frac{x - x_i}{x_k - x_i}$ w.r.t x. Using the five points $f(x_0 + 2h)$, $f(x_0 + h)$, $f(x_0)$, $f(x_0 - h)$, and $f(x_0 - 2h)$, evaluate $f^{(1)}(x_0) = \sum_{k=0}^{4} f(x_k) L_{4,k}^{(1)}(x_0)$.

11. DVR, Tunneling splitting, vibrational spectra of diatomics

11.1 Discrete variable representation

Formally the simplest basis set choice for expanding a trial function is the position variable x, itself. In practice, this is done by discretizing the spatial variable as discrete points, or grids (separated by Δx) and placing a Dirac delta function at each grid.

$$\widetilde{\Psi}(x_0) = \int_{-\infty}^{\infty} dx\, c(x)\delta(x - x_0) \tag{73}$$

$$\sum_{j=1}^{N_g} \Delta x c_j \delta(x - x_j) \tag{74}$$

The value of the position variable x at a certain location is obtained projecting the variable on a delta function localized at that location: $\int dx\, x\delta(x - x_j) = x_j$, where we have used the property, $\int dx\, f(x)\delta(x - x_0) = f(x_0)$, i.e., the delta function localized at x_0 plucks out the value of the function at this point.

If a region $a \leq x \leq b$ is divided into N_g grids, the lattice points $\{x_j\}$ are separated by a step-size of $\Delta x = (b - a)/(N_g - 1)$. The $j-$th grid position can be computed as $x_j = a + (j - 1)\,\Delta x$.

Note that the Dirac delta function has the property

$$\int dx\, \delta(x - x_i)\delta(x - x_j) = \delta(x_i - x_j) = \begin{array}{cc} \infty & i = j \\ 0 & i \neq j \end{array} . \tag{75}$$

which is the orthonormal relation for a continuous basis set. When $x_i = x_j$, we find the overlap to be ∞. For our purpose, we can think of the delta function as a limiting case of function that peaks at a position, such as a rectangular peak with height $1/(2\epsilon)$, and width 2ϵ.

$$\delta(x - x_j) = \lim_{\epsilon \to 0} f_\epsilon(x - x_j) = \begin{array}{ll} \frac{1}{2\epsilon}; & -\epsilon \leq x - x_j \leq \epsilon \\ 0; & \text{otherwise} \end{array}$$

In the case of discrete representation, the width is $\epsilon = \Delta x/2$. This will ensure the usual normalization condition.

11.1.1 Matrix elements for Harmonic oscillator

As pointed out in an earlier lecture, computing the matrix elements in the "grid- representation" is trivial. For a $\hat{V}$ which is a function of x, the potential matrix becomes diagonal.

$$\int dx\, \delta(x-x_i)\hat{V}(x)\delta(x-x_j) = V(x_j)\delta(x_j-x_i) \qquad (76)$$

The discretized version of the above equation is

$$V_{i,j} = \begin{matrix} V(x_i) & i=j \\ 0 & i \neq j \end{matrix} \qquad (77)$$

The kinetic energy operator, however, is not in its diagonal form in the position representation. Let us start by discretizing the Laplacian operator, $\hat{D2} = d^2/dx^2$.

$$\hat{D2}f(x) = \frac{1}{\Delta x^2}\left[f\left(x-\Delta x\right) - 2f\left(x\right) + f\left(x+\Delta x\right)\right] \qquad 3-\text{point}$$

$$\begin{aligned}\hat{D2}f(x) &= \frac{1}{\Delta x^2}[-(1/12)f\left(x-2\Delta x\right) + (4/3)f\left(x-\Delta x\right) \\ &\quad -(5/2)f\left(x\right) + (4/3)f\left(x+\Delta x\right) - (1/12)f\left(x-2\Delta x\right)] \qquad 5-\text{point}\end{aligned}$$

If $f(x) = x$,

$$\hat{D2}x = \frac{1}{\Delta x^2}\left[\left(x-\Delta x\right) - 2\left(x\right) + \left(x+\Delta x\right)\right] \qquad 3-\text{point.}$$

The discretized version of the kinetic energy operator, $\hat{T} = -\left(\hbar^2/2m\right)d^2/dx^2$ becomes

$$\begin{aligned}\hat{T}x_j &= -\frac{\hbar^2}{2m}\frac{1}{\Delta x^2}\left[\left(x_j-\Delta x\right) - 2x_j + \left(x_j+\Delta x\right)\right] &(78)\\ &= -\frac{\hbar^2}{2m}\left[x_{j-1} - 2x_j + x_{j+1}\right] &(79)\end{aligned}$$

Now

$$\int dx\, x_i\hat{T}x_j = -\frac{\hbar^2}{2m}\left[\delta(x_i-x_{j-1}) - 2\delta(x_i-x_j) + \delta(x_i-x_{j+1})\right] \qquad (80)$$

The discretized version of the above equation is

$$T_{i,j} = \begin{matrix} -\frac{\hbar^2}{2m} & i=j-1 \text{ or } i=j+1 \\ \frac{\hbar^2}{m} & i=j \\ 0 & \text{else where} \end{matrix} \qquad (81)$$

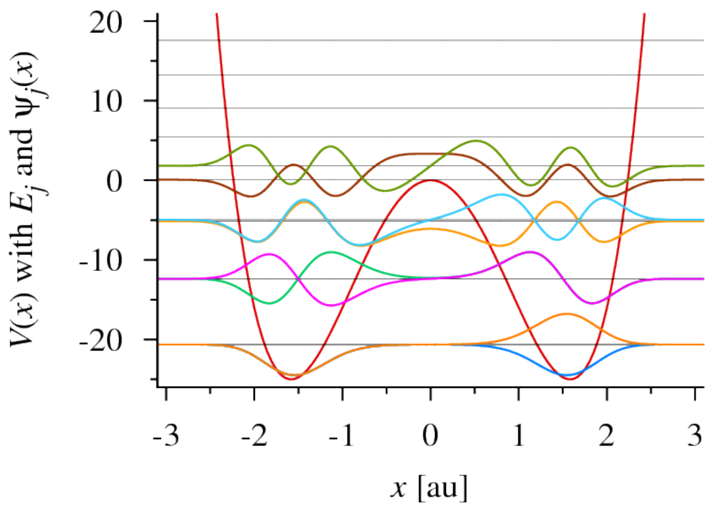
11.2 Fortran90 code

The code in the handout presents the implementation of the computation of the Hamiltonian matrix element of a harmonic oscillator in the grid representation. The code can discretize the kinetic energy operator using the 3, 5, or 7-point central difference formula. Note that towards the end of the region, for example, at $x = a$ or $x = b$, the 3-point central difference does not hold, but this must not affect the implementation, because we will use a larger range, such as $-10 \leq x \leq 10$, so that towards the boundary the wavefunction sufficiently becomes zero.

11.3 Tunneling splitting

Let us look at the vesatility of the DVR approach by considering a more complicated 1D potential that corresponds to the symmetric double well, $\hat{V} = -20x^2 + 4x^4$. This function has a maximum at $x = 0$, and two symmetric minima at $x = \pm\sqrt{2.5}$. Each minimum can be thought of as a well. Wavefunctions of the states that have energies below the barrier (maximum), will interact across the two wells to show tunneling splitting. The following figure shows how the splitting is more pronounced with increase in energy. The ground and the first excited states are perfectly doubly degenerate. The second excited state, with energy about -5 atomic units, show noticeable tunneling splitting, and the degeneracy is slightly broken. States above are totally non-degenerate because the wavefunctions are no longer confined into any of the wells.

You can reproduce the energy levels shown in the figure, by simply replacing the harmonic potential by the double well potential in the code presented in the handout. No other changes are needed.



12. Numerical integration

Let us start by recalling the formula for Lagrange interpolation polynomial that passes through n-interpolation-abscissae $\{x_0, x_1, \ldots, x_n\}$

$$f(x) = \sum_{k=0}^{n} f(x_k) L_{n,k}(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{i=0}^{n} (x - x_i).$$

Integrating this equation gives

$$\int_{x_0}^{x_n} dx\, f(x) = \sum_{k=0}^{n} f(x_k) \int_{x_0}^{x_n} dx\, L_{n,k}(x) + \int_{x_0}^{x_n} dx \left(\frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{i=0}^{n} (x - x_i) \right).$$

As usual, the first term on the r.h.s. is the approximation to the integral, and the second term is the truncation error or remainder.

12.1 Trapezoidal rule, $n = 1$, $\{x_0, x_0 + h\}$

Let us derive the 2-point formula, starting from the l.h.s.

$$\begin{aligned}
\sum_{k=0}^{1} f(x_k) L_{1,k}(x) &= \frac{(x - x_1)}{(x_0 - x_1)} f(x_0) + \frac{(x - x_0)}{(x_1 - x_0)} f(x_1) \\
\sum_{k=0}^{1} f(x_k) \int_{x_0}^{x_1} dx\, L_{1,k}(x) &= \left[\frac{(x - x_1)^2}{2(x_0 - x_1)} f(x_0) + \frac{(x - x_0)^2}{2(x_1 - x_0)} f(x_1) \right]_{x_0}^{x_1} \\
&= \frac{h}{2} [f(x_1) + f(x_0)]
\end{aligned}$$

which is the Trapezoidal rule for integration. The error term is given by

$$\begin{aligned}
\int_{x_0}^{x_1} dx \left(\frac{f^{(2)}(\xi(x))}{2!} \prod_{i=0}^{1} (x - x_i) \right) &= \frac{f^{(2)}(\xi(x))}{2!} \int_{x_0}^{x_1} dx\ (x - x_0) . (x - x_1) \\
&= \frac{f^{(2)}(\xi(x))}{2!} \left(-\frac{h^3}{6} \right),
\end{aligned}$$

the derivation is left as an exercise. The error term is proportional to $f^{(2)}$, hence the Trapezoidal rule is exact for a function with vanishing second-derivative, i.e., a polynomial of degree one or less. Similar conclusions can be drawn for higher-order formulae.

12.2 Closed Newton-Cotes formulae

Trapezoidal rule and their extensions, which are derived using $n + 1$ equidistant abscissae are called as Newton-Cotes formulae.

12.2.1 $n = 1$, *Trapezoidal rule*

$$\int_{x_0}^{x_1} dx\, f(x) = \frac{h}{2}\left[f(x_0) + f(x_1)\right] - \frac{1}{12}h^3 f^{(2)}(\xi(x)) \tag{82}$$

12.2.2 $n = 2$, *Simpson's rule*

$$\int_{x_0}^{x_2} dx\, f(x) = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] - \frac{1}{90}h^5 f^{(4)}(\xi(x)) \tag{83}$$

Note that the error term is not dependent on h^4 but on h^5. The dependence on h^4 vanishes by error cancellation, and we will have to consider the next lower order term, which is h^5, the error is $\mathcal{O}(h^5)$. This is the reason why Simpson is so widely applied. Recall that we had already encountered error cancellation in the derivation of 3-point central difference formula.

12.2.3 $n = 3$, *Simpson's Three-Eighths rule*

$$\int_{x_0}^{x_3} dx\, f(x) = \frac{3h}{8}\left[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right] - \frac{3}{80}h^5 f^{(4)}(\xi(x)) \tag{84}$$

12.3 *Composite rules*

In any typical application, we will divide the region $a \le x \le b$, into a large number of regions. Here it is not necessary to derive closed-form expressions using higher-degree Legendre polynomials, but to integrate the function, piecewise, using a lower degree polynomial.

$$\int_a^b dx\, f(x) = \int_a^{y_1} dx\, f(x) + \int_{y_1}^{y_2} dx\, f(x) + \int_{y_2}^{b} dx\, f(x), \tag{85}$$

where each integral on the r.h.s. can be approximated using an $n+1$-point formula.

12.3.1 *Composite trapezoidal rule*

$$\int_a^b dx\, f(x) \approx \frac{h}{2}\left[f(a) + 2\sum_{j=1}^{n-1} f(x_j) + f(b)\right] \tag{86}$$

12.3.2 *Composite Simpson's rule*

$$\int_a^b dx\, f(x) \approx \frac{h}{3}\left[f(a) + 2\sum_{j=1}^{(n/2)-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(b)\right] \tag{87}$$

12.4 Implementation

12.4.1 Composite trapezoidal rule

In practice, these formulae are applied in an iterative fashion. At each iteration, i, (starting from $i = 1$) the step size is determined by $h = (b-a)/2^i$. The number of intermediate points is then $n - 1 = 2^i - 1$.

Fortran90 implementation of a trapezoidal method, along with a driver program is presented in a separate handout.

12.5 Gaussian quadrature

Gaussian quadrature chooses optimal points for evaluating the function, rather than equally spaced points. In the quadrature equation

$$\int_a^b dx\, f(x) \approx \sum_{i=1}^{n} c_i f(x_i), \tag{88}$$

the nodes $x_1, x_2, \ldots, x_n$, (all lie in $[a, b]$) as well as the coefficients $c_1, c_2, \ldots, c_n$, are chosen to minimize the approximation error. Overall, there are $2n$ free parameters to choose. Recall that a polynomial of degree n depends on $n+1$ free parameters (the coefficients). Thus $2n$ parameters can define a polynomial of degree $2n-1$ or less. Thus, an n-point Gaussian quadrature formula calculates integrals exactly for polynomials of order $2n-1$ or less.

12.5.1 Two point Gauss-Legendre quadrature

Consider the two-point quadrature problem

$$\int_{-1}^{1} dx\, f(x) \approx c_1 f(x_1) + c_2 f(x_2). \tag{89}$$

For the optimal c_j, and x_j, the quadrature must be exact if $f(x)$ is a polynomial of degree 3.

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3. \tag{90}$$

Let us integrate this equation and compare to the quadrature equation

$$\int_{-1}^{1} dx \left[a_0 + a_1 x + a_2 x^2 + a_3 x^3\right] = c_1 f(x_1) + c_2 f(x_2) \tag{91}$$

This means that quadrature will be exact for $f(x) = a_0, a_1 x, a_2 x^2, a_3 x^3$ which gives us four equations:

$$\int_{-1}^{1} dx\, a_0 = 2a_0 = c_1 a_0 + c_2 a_0 \Rightarrow 2 = c_1 + c_2 \tag{92}$$

$$\int_{-1}^{1} dx\, a_1 x = 0 = c_1 a_1 x_1 + c_2 a_1 x_2 \Rightarrow 0 = c_1 x_1 + c_2 x_2 \tag{93}$$

$$\int_{-1}^{1} dx\, a_2 x^2 = \frac{2}{3} a_2 = c_1 a_2 x_1^2 + c_2 a_2 x_2^2 \Rightarrow \frac{2}{3} = c_1 x_1^2 + c_2 x_2^2 \tag{94}$$

$$\int_{-1}^{1} dx\, a_3 x^3 = 0 = c_1 a_3 x_1^3 + c_2 a_3 x_2^3 \Rightarrow 0 = c_1 x_1^3 + c_2 x_2^3 \tag{95}$$

In one of the exercises, you will show these equations to have a unique solution at $c_1 = c_2 = 1$, $x_1 = -1/\sqrt{3} = -0.577350269189626$, and $x_2 = 1/\sqrt{3} = 0.577350269189626$. The roots $x_1$and x_2 are nothing but the roots of the Legendre polynomial of degree 2, $P_2^{\text{Legendre}}(x) = x^2 - \frac{1}{3}$. The coefficients are obtained from the roots from the formula

$$c_j = \int_{-1}^{n} dx\, \Pi_{k=1,k\neq j}^{n} \frac{x - x_k}{x_j - x_k}; \quad \text{where}\, n = 1. \tag{96}$$

12.5.2 Change of limits

The Legendre polynomials are defined in the range $[-1, 1]$. The quadrature formulae can be changed to any arbitrary range $[a, b]$ by

$$\int_{a}^{b} dx\, f(x) = \frac{b-a}{2} \int_{-1}^{1} dt\, f\left(\frac{(b-a)\,t + (b+a)}{2}\right). \tag{97}$$

12.5.3 Gauss-Legendre/Gauss-Hermite/Gauss-Laguerre quadratures

General Gaussian-quadrate formula are of the form

$$\int_{a}^{b} dx\, w(x) f(x) \approx \sum_{i=1}^{n} c_i f(x_i), \tag{98}$$

The weighting function $w(x)$, the abscissae x_j, and the coefficients c_j are derived for orthogonal polynomials. The weighting function is required so that the polynomials satisfy usual orthonormal relations. The abscissae x_j are the roots of the polynomials, and the coefficients c_j are given (as mentioned before)

$$c_j = \int_{-1}^{n} dx\, \Pi_{k=1,k\neq j}^{n} \frac{x - x_k}{x_j - x_k}. \tag{99}$$

The choice of a polynomial depends on the integration range as summarized below.

Polynomial	limit	$w(x)$
Legendre	$[-1, 1]$	1
Hermite	$[-\infty, \infty]$	$\exp\left(-x^2\right)$
Laguerre	$[0, \infty]$	$\exp\left(x^2\right)$

Depending on x_j and c_j, we call the quadrature formula as Gauss-Legendre, Gauss-Hermite, or Gauss-Laguerre quadratures.

The abscissae and coefficients for Gauss-Legendre, Gauss-Hermite, or Gauss-Laguerre quadratures are listed in a separate handout.

12.5.4 Example: Gauss-Legendre

Approximate $I = \int_{-1}^{1} dx \, \exp(x) \cos(x)$ using Gauss-Legendre quadratre with $n = 2$, 3, and 4.

Exact value of this integral can be computed using the trapezoid program, and it can be found to be $I^{\text{exact}} = 1.93342150$. Using x_j and c_j from the handout we find

$$\begin{aligned} I_{2-\text{point}}^{\text{Gauss}-\text{Legendre}} &= c_1 f(x_1) + c_2 f(x_2) \\ &= 1.96297276 \end{aligned}$$

$$\begin{aligned} I_{3-\text{point}}^{\text{Gauss}-\text{Legendre}} &= c_1 f(x_1) + c_2 f(x_2) + c_2 f(x_3) \\ &= 1.93339047 \end{aligned}$$

$$\begin{aligned} I_{4-\text{point}}^{\text{Gauss}-\text{Legendre}} &= c_1 f(x_1) + c_2 f(x_2) + c_2 f(x_3) + c_4 f(x_4) \\ &= 1.93341689 \end{aligned}$$

12.5.4 Example: Gauss-Legendre with variable limit

Approximate $I = \int_{1}^{3} dx \, \left[x^6 - x^2 \sin(2x)\right]$ using Gauss-Legendre quadratre with $n = 2$, 3, and 4.

Exact value of this integral can be computed using the trapezoid program, and it can be found to be $I^{\text{exact}} = 317.34424667$. Using x_j and c_j from the handout, and the formula

$$\int_a^b dx \, f(x) = \frac{b-a}{2} \int_{-1}^{1} dt \, f\left(\frac{(b-a)\,t + (b+a)}{2}\right) \tag{100}$$

we find

$$\begin{aligned} I_{2-\text{point}}^{\text{Gauss}-\text{Legendre}} &= c_1 f(x_1) + c_2 f(x_2) \\ &= 306.81993449 \end{aligned}$$

$$\begin{aligned} I_{3-\text{point}}^{\text{Gauss}-\text{Legendre}} &= c_1 f(x_1) + c_2 f(x_2) + c_2 f(x_3) \\ &= 317.26415173 \end{aligned}$$

$$\begin{aligned} I_{4-\text{point}}^{\text{Gauss}-\text{Legendre}} &= c_1 f(x_1) + c_2 f(x_2) + c_2 f(x_3) + c_4 f(x_4) \\ &= 317.34539033 \end{aligned}$$

Exercises - 8

Exercise 8.1: Remainder for trapezoid

Show that

$$\int_{x_0}^{x_1} dx \left(\frac{f^2\left(\xi\left(x\right)\right)}{2!} \prod_{i=0}^{1} \left(x - x_i\right) \right) \quad = \quad \frac{f^2\left(\xi\left(x\right)\right)}{2!} \left(-\frac{h^3}{12}\right)$$

Exercise 8.2: 2-point Gauss-Legendre

Show that the equation

$$\begin{bmatrix} 1 & 1 \\ x_1 & x_2 \\ x_1^2 & x_2^2 \\ x_1^3 & x_2^3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ \frac{2}{3} \\ 0 \end{bmatrix}$$

has a unique solution $c_1 = c_2 = 1$, $x_1 = -1/\sqrt{3} = -0.577350269189626$, and $x_2 = 1/\sqrt{3} = 0.577350269189626$. You may start with the known constraint $c_j \neq 0$.

Exercise 8.3: Singularity

Use the trapezoidal rule program given in the handout, to evaluate the integral

$$\int_0^{0.64} dx \, \frac{\arctan(x)}{\sqrt{(x)}}.$$

Try the same integral after a change of variable

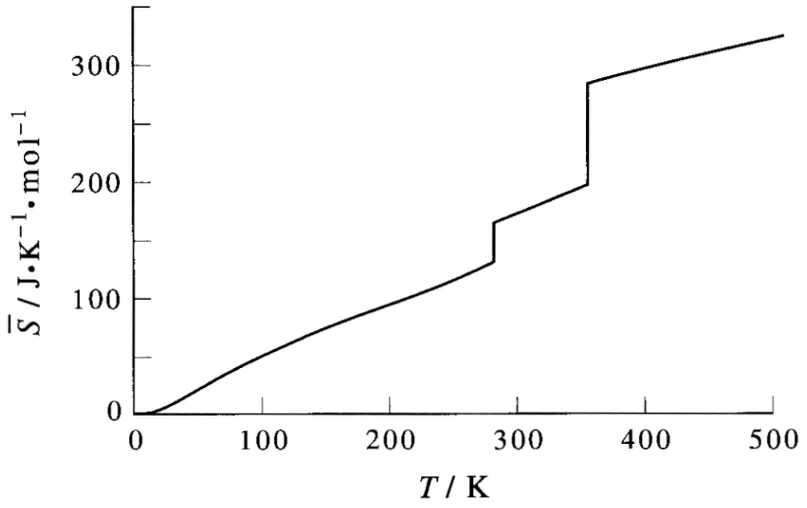
$$t = \sqrt{X}.$$

Exercise 8.4: Entropy

To calculate molat entropy, $\overline{S}$ at a temperature T_0, we integrate $\overline{C_p}(T)/T$ up to the first phase (e.g. solid) of transition temperature, add $\Delta_{trs}\overline{H}/T_{trs}$ (solid to liquid, melt) at the phase transition temperature, then repeat for the second phase, and so on until T_0 is reached.

$$\begin{aligned}\overline{S}(T) \;=\;& \overline{S}^{Debye}(T_{low}) + \int_{T_{low}}^{T_{melt}} \frac{\overline{C}_p^{solid}(T)dT}{T} + \frac{\Delta_{melt}\overline{H}}{T_{melt}} + \\ & \int_{T_{melt}}^{T_{boil}} \frac{\overline{C}_p^{liquid}(T)dT}{T} + \frac{\Delta_{boil}\overline{H}}{T_{boil}} + \int_{T_{boil}}^{T_0} \frac{\overline{C}_p^{gas}(T)dT}{T}\end{aligned}$$

Here is how the plot of $\overline{S}(T)$ looks for benzene.



The heat capacity of solid CH_3CH_2Cl up to 15 K can be determined using Debye theory as $\overline{S}^{Debye}(T_{low}) = 1.88$ J K^{-1}mol^{-1}. From $T = 15$ K to 286.2 K, at a constant pressure of 1 bar, the molar heat capacity of CH_3CH_2Cl is tabulated below. Further, at the same pressure this compound melts at 134.4 K with $\Delta_{melt}\overline{H} = 4.45\,$kJ/mol, and boils at 286.2 K with $\Delta_{boil}\overline{H} = 24.65\,$kJ/mol.

Using all these information calculate, by numerical integration (you may design a procedure, if you wish), the molar entropy of CH_3CH_2Cl at its boiling point.

T/K	$\overline{C}_P/\mathrm{J\cdot K^{-1}\cdot mol^{-1}}$	T/K	$\overline{C}_P/\mathrm{J\cdot K^{-1}\cdot mol^{-1}}$
15	5.65	130	84.60
20	11.42	134.4	90.83 (solid)
25	16.53		97.19 (liquid)
30	21.21	140	96.86
35	25.52	150	96.40
40	29.62	160	96.02
50	36.53	180	95.65
60	42.47	200	95.77
70	47.53	220	96.04
80	52.63	240	97.78
90	55.23	260	99.79
100	59.66	280	102.09
110	65.48	286.2	102.13
120	73.55		

Exercise 8.5: Virial coefficient

The virial equation of state is the most fundamental equation of state. Here the compressibility factor $Z = P\overline{V}/RT$ is expressed as a polynomial in $1/\overline{V}$

$$Z = \frac{P\overline{V}}{RT} = 1 + \frac{B_{2v}(T)}{\overline{V}} + \frac{B_{3v}(T)}{\overline{V}^2} + \dots$$

Such an expansion is called as virial expansion. When all $B_{nV}(T)$ are zero, we get back the compressibility factor of the ideal gas, $Z_{\text{ideal}} = 1$. The second virial coefficient $B_{2V}(T)$ marks the first deviation from the ideal nature, hence it is a very important quantity. This quantity has also been measured accurately for many gases.

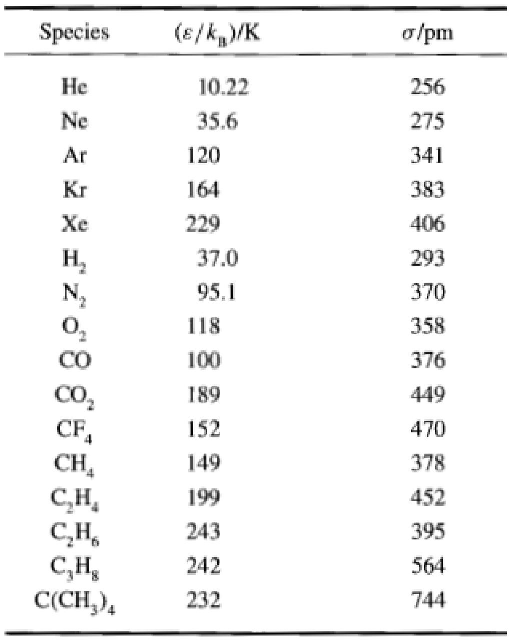
The virial coefficient is a measure of intermolecular interactions. As a simple approximation, the interaction between two molecules can be assumed to be dependent only on the distance (r) between them, i.e., we have averaged over all orientations.

$$B_{2v}(T) = -2\pi N_A \int_0^\infty dr\, r^2 \left[\exp\{-u(r)/(k_B T)\} - 1\right]$$

If $u(r) = 0$, (no interactions) then $B_{2v}(T) = 0$ (ideal gas). A simple two-parameter potential that is widely used for modeling intermolecular interactions is the Lennard-Jones (LJ) potential (see wiki entry for definitions of the constants).

$$u(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right].$$

1. Find analytically the r for which the LJ potential is minimum, i.e. r_{min}. What is the value of $u(r_{min})$? Plot $u(r)$ with the unit of u as ϵ, and unit of r as σ. What is the significance of the quantity σ ?
2. Write a Fortran90 program, which uses a numerical integration procedure, and calculate B_{2v} for the gases, He, H_2, CO, N_2, and CH_4. Use LJ constants from the following table.



Species	(ε/k_B)/K	σ/pm
He	10.22	256
Ne	35.6	275
Ar	120	341
Kr	164	383
Xe	229	406
H_2	37.0	293
N_2	95.1	370
O_2	118	358
CO	100	376
CO_2	189	449
CF_4	152	470
CH_4	149	378
C_2H_4	199	452
C_2H_6	243	395
C_3H_8	242	564
$C(CH_3)_4$	232	744

13. Least squares regression

Regression, regression analysis, or curve fitting is a very useful technique to fit (or represent) experimental data to determine parameters entering a mathematical model of a system. Here we will only look at linear models, i.e., models that are linearly dependent on the parameters. The simplest linear model is the model of a straight line.

13.1 Straight-line least-squares fitting

Suppose we want to model some experimental observations (e.g. molecular properties, reaction rates, etc.), y_i as a linear function of inputs, x_i (e.g. temperature, etc.,), then the simplest model that one can use corresponds to that of a straight-line

$$y_i = a + bx_i.$$

If we want to find the best values for a, and b, we have to optimize these parameters to decrease the error in the model. One definition of prediction error is the least square error or least-square residual, defined by

$$R = \sum_{i=1}^{N} (y_i - a - bx_i)^2 = \langle \mathbf{y} - a - b\mathbf{x} | \mathbf{y} - a - b\mathbf{x} \rangle,$$

where N corresponds to the number of observations or number of data points. Our objective is to find the values of a and b so that R is a minimum. We can do this analytically as follows.

$$\frac{\partial R}{\partial a} = 0 \qquad (101)$$

$$\Rightarrow \sum_{i=1}^{N} \frac{\partial}{\partial a} (y_i - a - bx_i)^2 = -2\sum_{i=1}^{N} (y_i - a - bx_i) = 0$$

$$\Rightarrow Na + b\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} y_i \qquad (102)$$

$$\frac{\partial R}{\partial b} = 0 \qquad (103)$$

$$\Rightarrow \sum_{i=1}^{N} \frac{\partial}{\partial b} (y_i - a - bx_i)^2 = -2\sum_{i=1}^{N} (y_i - a - bx_i)\, x_i = 0$$

$$\Rightarrow a\sum_{i=1}^{N} x_i + b\sum_{i=1}^{N} x_i^2 = \sum_{i=1}^{N} x_i y_i \qquad (104)$$

Both equations can be collectively presented as

$$\begin{bmatrix} N & \sum_{i=1}^N x_i \\ \sum_{i=1}^N x_i & \sum_{i=1}^N x_i^2 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^N y_i \\ \sum_{i=1}^N x_i y_i \end{bmatrix} \tag{105}$$

By now we are very familiar with such equations, where the unknown vector can be computed through a matrix inversion

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} N & \sum_{i=1}^N x_i \\ \sum_{i=1}^N x_i & \sum_{i=1}^N x_i^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^N y_i \\ \sum_{i=1}^N x_i y_i \end{bmatrix}. \tag{106}$$

13.2 Polynomial/Curve least-squares fitting

The straight-line model can be generalized to that of a polynomial of degree D. If we want to model N outputs (y_i) from N observations, using a polynomial in x_i of degree D, i.e.,

$$y_i \quad = \quad a_0 + a_1 x + \ldots + a_D x^D = \sum_{k=0}^{D} a_k x^k,$$

all we have to do is to compute the unknown vector via matrix inversion.

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_D \end{bmatrix} = \begin{bmatrix} N & \sum_{i=1}^N x_i & \cdots & \sum_{i=1}^N x_i^N \\ \sum_{i=1}^N x_i & \sum_{i=1}^N x_i^2 & & \sum_{i=1}^N x_i^{N+1} \\ \vdots & & \ddots & \\ \sum_{i=1}^N x_i^D & \sum_{i=1}^N x_i^{D+1} & & \sum_{i=1}^N x_i^{D\times N} \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^N y_i \\ \sum_{i=1}^N x_i y_i \\ \vdots \\ \sum_{i=1}^N x_i^D y_i \end{bmatrix} \tag{107}$$

13.3 Vandermonde matrix

Let us start with the 2×2 matrix and see if we can simplify the above equations. The RHS of the 2×2 case can be written as

$$\begin{bmatrix} \sum_{i=1}^N y_i \\ \sum_{i=1}^N x_i y_i \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{108}$$

The matrix on the RHS is called as the transpose of the Vandermonde matrix, $\mathbf{X}^{\mathrm{T}}$. Using $\mathbf{X}$, we can also simplify the LHS of the 2×2 case

$$\begin{aligned}
\mathbf{X}^{\mathrm{T}}\cdot\mathbf{X} &= \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \end{bmatrix}\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} && (109)\\
&= \begin{bmatrix} N & \sum_i^N x_i \\ \sum_i^N x_i & \sum_i^N x_i^2 \end{bmatrix} && (110)
\end{aligned}$$

So the 2×2 case (i.e., a straight-line fit) can be directly expressed using the Vandermonde matrix as

$$\begin{aligned}
\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \end{bmatrix}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \end{bmatrix}^{\mathrm{T}}\begin{bmatrix} a \\ b \end{bmatrix} &= \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} && (111)\\
\mathbf{X}^{\mathrm{T}}\cdot\mathbf{X}\cdot\mathbf{a} &= \mathbf{X}^{\mathrm{T}}\cdot\mathbf{b} && (112)\\
\Rightarrow \mathbf{a} &= \left(\mathbf{X}^{\mathrm{T}}\cdot\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\cdot\mathbf{b} && (113)
\end{aligned}$$

For fitting data to a polynomial of degree D, the Vandermonde matrix takes the form

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^D \\ 1 & x_2 & \cdots & x_2^D \\ \vdots & \vdots & \ddots & \\ 1 & x_N & \cdots & x_N^D \end{bmatrix}, \qquad (114)$$

where each row corresponds to input x_i from $i-$th observation , and in each column, j, we take the $j-$th power of the input. By now it must be clear that the Vandermonde matrix need not be square, i.e., you can have $\{x_i, y_i\}$, from N observations (or N equations) and fit them to a polynomial of any degree D. When $N > D + 1$, we have an overdetermined system of equations (fewer unknowns from more equations), and when $N < D + 1$, we are dealing with an underdetermined system (more unknowns from fewer equations).

13.4 Pearson product-moment correlation coefficient

If the two variables x and y vary together, then they are said to be correlated. The Pearson product-moment correlation coefficient, r, gives the strength of a linear relationship between the N values of two variables, $\{x_i, y_i\}$.

$$r = \frac{\sum_{i=1}^{N} x_i y_i - \left(\sum_{i=1}^{N} x_i\right)\left(\sum_{i=1}^{N} y_i\right)}{\sqrt{\left[\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2\right]}\sqrt{\left[\sum_{i=1}^{N} y_i^2 - \left(\sum_{i=1}^{N} y_i\right)^2\right]}}$$

This expression can be simplified as

$$r = \frac{\mathrm{cov}_{\mathrm{XY}}}{\sigma_{\mathbf{x}}\sigma_{\mathbf{y}}},$$

where $\mathrm{cov}_{\mathrm{XY}}$ is the covariance of two-vatiables$x$ and y

$$\mathrm{cov}_{\mathrm{XY}} = \langle (\mathbf{x} - \langle \mathbf{x} \rangle)(\mathbf{y} - \langle \mathbf{y} \rangle) \rangle \quad (115)$$

and σ_{X} is the standard deviation

$$\sigma_{\mathrm{X}} = \sqrt{\langle \mathbf{x}^2 \rangle - (\langle \mathbf{x} \rangle)^2}. \quad (116)$$

Exercises - 9

Exercise 9.1: Parabola fit

Fit the following data to a polynomial of degree at most 2.

i	1	2	3	4	5
x_i	0.0	0.25	0.50	0.75	1.00
y_i	1.0000	1.2840	1.6487	2.1170	2.7183

1. Make a plot (using GNUPLOT) containing the given data x vs. y^{given} along with the fitted function, x vs. y^{fit}.

2. Make a scatter plot, i.e., y^{given} vs. y^{fit}

3. Report the fitting errors (MAE and RMSE), and the Pearson product-moment correlation coefficient.

Exercise 9.2: H_2^+: Potential energy curve fitting and vibrational spectrum

1. Compute the electronic energy of the ground state of H_2^+ for various values of $r = R - R_0$, where R_0 is the optimal bond length, and R is the bond length away from R_0. For instance, you may choose 11 points: $R_0 =$ 1.50, 1.60, 1.70, 1.80, 1.90, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0 bohr. NOTE: At each value of R, ζ must be optimized (by 1D minimization) to get the variationally least energy. Let us call the resulting potential energy curve as $V^{\text{variational}}(r)$.

2. Fit the potential energies from the 11 points to the formula $V^{\text{quartic}}(r) = E(r) - E(R_0, \zeta_0) = \sum_{k=0}^{4} V_k r^k$. Explain the significance of each V_k.

3. The Morse potential is given by $V^{\mathrm{Morse}}(R) = D_e\left[1 - \exp\left(-a\left\{R - R_0\right\}\right)\right]^2$, where D_e is the dissociation energy, and the exponent a, is related to the second derivative of $V(R)$ at R_0

$$a = \sqrt{V''(R_0)/(2D_e)}.$$

How is V'' related to V_k from sub-problem 2?

4. Plot $V^{\mathrm{Morse}}(R)$ along with $V^{\mathrm{quartic}}(R)$ and $V^{\mathrm{variational}}(R)$ in the range $1 \leq R \leq 10$. Explain your observations.

5. Use both the fitted potentials ($V^{\mathrm{quartic}}(r)$ and $V^{\mathrm{Morse}}(r)$ in the 1D DVR program to compute the bound states. The resulting energy levels correspond to the vibrational energy levels of H_2^+including anharmonicity, and must correlate well with the experimental spectrum. Compare your predicted spectra with the following measured energy levels. Which potential gives you better comparison with the experiment, and explain why?.

n	E_n (cm^{-1})
0	1148.1
1	2195.2
2	4260.7
3	6201.6
4	8022.2
5	9726.0

14. Molecular entropy

14.1 Ideal gas, harmonic oscillator, rigid rotor approximation

The absolute entropy of a gas with $N-$particles is given by

$$S = k_\mathrm{B} \ln Q\left(N,V,T\right) + k_\mathrm{B} T \left(\frac{\partial \ln Q\left(N,V,T\right)}{\partial T}\right)_{N,V}, \tag{117}$$

where

$$Q\left(N,V,T\right) = \frac{\left[q\left(V,T\right)\right]^N}{N!}. \tag{118}$$

For a monoatomic ideal gas, in a non-degenerate electronic state the translational partition function is given by

$$q_\mathrm{trans}\left(V,T\right) = \left(\frac{2\pi m k_\mathrm{B} T}{h^2}\right)^{3/2} \cdot V, \tag{119}$$

where m is the mass of one molecule, and you can calculate V as $k_\mathrm{B}T/P$ when the pressure is known. The vibrational partition function (of a non-linear molecule) can be approximated using the harmonic oscillator model

$$q_\mathrm{vib}\left(V,T\right) = \Pi_{i=1}^{3N-6}\left[1 - \exp\left(-\frac{h\nu_i}{k_\mathrm{B}T}\right)\right], \tag{120}$$

we are summing over only the fundamental energy levels with in the harmonic approximation. The rotational partition function approximated by considering the molecule as a rigid $N-$particle system is given by

$$q_\mathrm{rot}\left(V,T\right) = \frac{1}{\pi\sigma}\left[\frac{8\pi^3 k_\mathrm{B} T \left(I_\mathrm{A} I_\mathrm{B} I_\mathrm{C}\right)^{1/3}}{h^2}\right]^{3/2}, \tag{121}$$

where I_A, I_B, and I_C are the three components of the moment of inertia tensor. The contant σ is called the symmetry number of the molecule and corresponds to the number of indistinguishable orientations of the molecule.

Exercises - 10

Exercise 10.1: Entropy of ethyl chloride

The minimal energy geometry of ethyl chloride has been modelled using the Hartree-Fock approach using the basis set STO-3G. The corresponding Cartesian coordinates in Angstrom are as follows.

```
 C       1.529258    0.659115    0.000000
```

```
C        0.000000    0.821909    0.000000
H        1.989916    1.643466    0.000000
H        1.856538    0.119187    0.883039
H        1.856538    0.119187   -0.883039
Cl      -0.834894   -0.792768    0.000000
H       -0.342674    1.354532    0.887255
H       -0.342674    1.354532   -0.887255
```

At the same geometry, the vibrational wavenumbers have been calculated with in the harmonic oscillator approximation. The corresponding values in cm^{-1} are

```
 248.1337               363.1202               860.6740
 908.6836              1178.4259              1260.0304
1275.2972              1485.6994              1570.0785
1720.3905              1780.0599              1809.0720
1818.7459              3576.9723              3617.9807
3743.0422              3756.3694              3769.1069
```

Calculate the total partition function

$$q = q_{\text{trans}} \cdot q_{\text{vib}} \cdot q_{\text{rot}}$$

and compute the molecular entropy of ethyl chloride using finite derivatives using 3,5, and 7-point central difference formulae. Analyse your results and compare with the value of entropy estimated using calorimetry values from an earlier exercise.

15. Initial-Value Problems for Ordinary Differential Equations

The range of differential equations that can be solved by analytical methods is limited. However, when a differential equation and some boundary conditions are given, an approximate solution at discrete values of independent variables can be obtained by numerical methods.

15.1 First-order differential equations

Numerical solution of $y' = f(x,y)$ with the initial condition that $y(x_0) = y_0$.

15.1.1 Euler's method

Once we know the initial condition, y_0, let us start expanding the value of y at $x = x_0 + h = x_1$ as a truncated Taylor polynomial

$$y_1^{\text{Taylor}} = y_0 + hy^{(1)}(x_0) + \frac{h^2}{2}y^{(2)}\left(\zeta\left(x_0\right)\right)$$

Euler's method truncates the above expansion at the term that is first-order in h.

$$y_1^{\text{Euler}} \approx y_0 + hy^{(1)}(x_0)$$

Note that we do not have to compute $y^1(x_0)$ explicitly because the ODE already is in the form $y' = f\left(x,y\right)$. Hence we can use the RHS of the original problem.

$$y_1^{\text{Euler}} \approx y_0 + hf(x_0,y_0).$$

The local truncation error of the Euler method is error made in a single step. It is the difference between the numerical solution (above equation) and the exact solution as a Taylor series (the equation further above)

$$\begin{aligned} \text{LTE} = \frac{y_1^{\text{Taylor}} - y_1^{\text{Euler}}}{h} &= \frac{h}{2}y^{(2)}\left(\zeta\left(x_0\right)\right) \\ &= \mathcal{O}\left(h\right) \end{aligned}$$

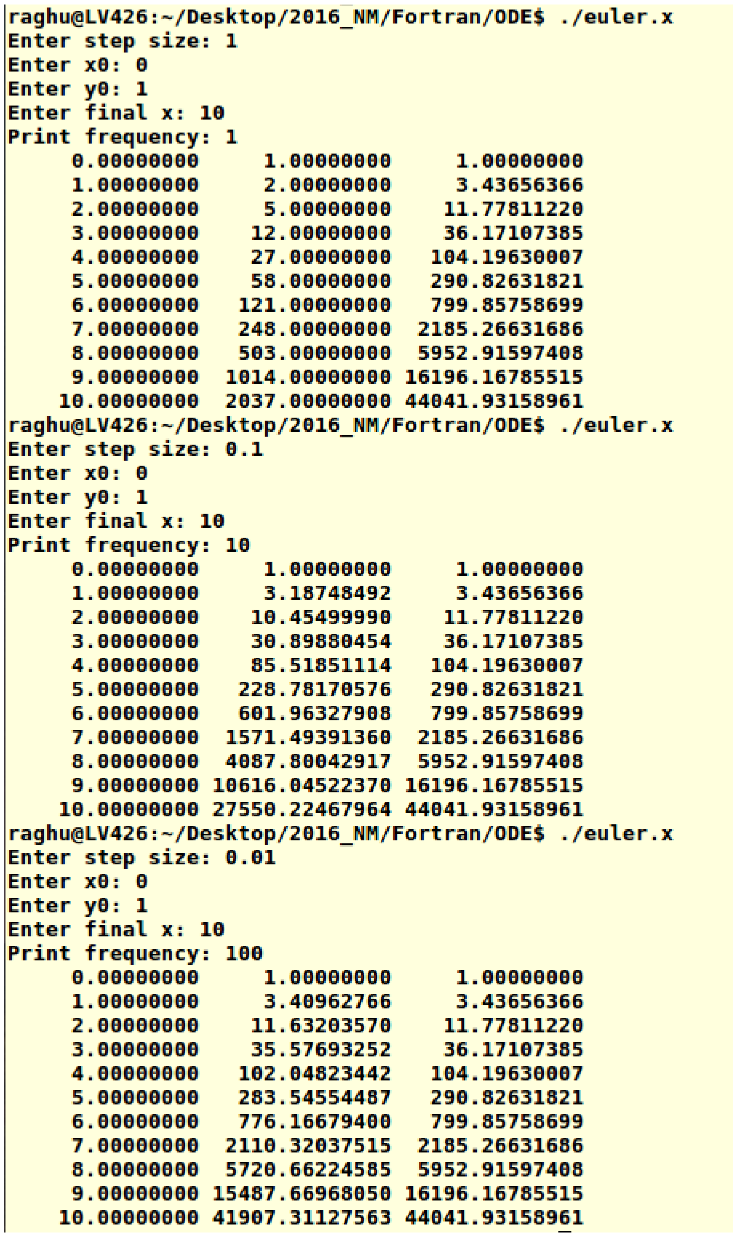
We can continue this procedure and calculate y_2^{Euler}, $y_3^{\text{Euler}}, \ldots$ and so on.

Example 1

Let us look at the ODE $y' = x + y$ with the initial conditions, $x_0 = 0$ and $y(x_0) = y_0 = 1$. The exact solution of this ODE is $y(x) = 2e^x - x - 1$. Here is an implementation of the Euler method which you can adapt. Note that the RHS is included as a separate function procedure.

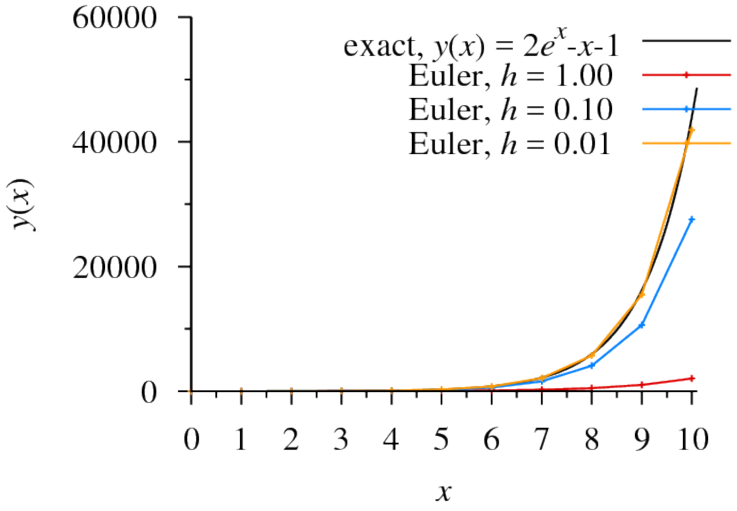
```
program euler_ode

  implicit none
  double precision           :: h, x0, y0, xf
  double precision           :: x, y
  double precision, external :: f, yexact
  integer                    :: ih, dh_print, Nstep

  write(*,'(a)', advance='no') 'Enter step size: '
  read *, h
  write(*,'(a)', advance='no') 'Enter x0: '
  read *, x0
  write(*,'(a)', advance='no') 'Enter y0: '
  read *, y0
  write(*,'(a)', advance='no') 'Enter final x: '
  read *, xf
  write(*,'(a)', advance='no') 'Print frequency: '
  read *, dh_print

  Nstep = nint( (xf - x0)/h )
  ! initial value
  x = x0
  y = y0
  print '(3f15.8)', x, y, yexact(x)

  ! propogation
  do ih = 1, Nstep
    y = y + h * f(x, y) ! Euler
    x = x + h
    if ( mod(ih, dh_print) .eq. 0 ) then
      print '(3f15.8)', x, y, yexact(x)
    endif
  enddo

end program

double precision function f(x,y)

  implicit none
  double precision, intent(in) :: x, y

  f = x + y

end function

double precision function yexact(x)

  implicit none
  double precision, intent(in) :: x

  yexact = 2d0 * exp(x) - x - 1d0

end function
```

Using this program for $h = 1$, 0.1 and 0.01, we get the following output



```
raghu@LV426:~/Desktop/2016_NM/Fortran/ODE$ ./euler.x
Enter step size: 1
Enter x0: 0
Enter y0: 1
Enter final x: 10
Print frequency: 1
     0.00000000     1.00000000     1.00000000
     1.00000000     2.00000000     3.43656366
     2.00000000     5.00000000    11.77811220
     3.00000000    12.00000000    36.17107385
     4.00000000    27.00000000   104.19630007
     5.00000000    58.00000000   290.82631821
     6.00000000   121.00000000   799.85758699
     7.00000000   248.00000000  2185.26631686
     8.00000000   503.00000000  5952.91597408
     9.00000000  1014.00000000 16196.16785515
    10.00000000  2037.00000000 44041.93158961
raghu@LV426:~/Desktop/2016_NM/Fortran/ODE$ ./euler.x
Enter step size: 0.1
Enter x0: 0
Enter y0: 1
Enter final x: 10
Print frequency: 10
     0.00000000     1.00000000     1.00000000
     1.00000000     3.18748492     3.43656366
     2.00000000    10.45499990    11.77811220
     3.00000000    30.89880454    36.17107385
     4.00000000    85.51851114   104.19630007
     5.00000000   228.78170576   290.82631821
     6.00000000   601.96327908   799.85758699
     7.00000000  1571.49391360  2185.26631686
     8.00000000  4087.80042917  5952.91597408
     9.00000000 10616.04522370 16196.16785515
    10.00000000 27550.22467964 44041.93158961
raghu@LV426:~/Desktop/2016_NM/Fortran/ODE$ ./euler.x
Enter step size: 0.01
Enter x0: 0
Enter y0: 1
Enter final x: 10
Print frequency: 100
     0.00000000     1.00000000     1.00000000
     1.00000000     3.40962766     3.43656366
     2.00000000    11.63203570    11.77811220
     3.00000000    35.57693252    36.17107385
     4.00000000   102.04823442   104.19630007
     5.00000000   283.54554487   290.82631821
     6.00000000   776.16679400   799.85758699
     7.00000000  2110.32037515  2185.26631686
     8.00000000  5720.66224585  5952.91597408
     9.00000000 15487.66968050 16196.16785515
    10.00000000 41907.31127563 44041.93158961
```

The following figure presents the numerical solutions along with the exact analytic solution.



15.1.2 Higher order Taylor methods

We have derived Euler's method using a Taylor polynomial of degree 1. So, we can call this method as first order Taylor method. We can include more terms in the Taylor polynomial and derive more accurate equations of higher order.

Taylor method of order 1, aka Euler's method

$$\begin{aligned} y_k &\approx y_{k-1} + hy^{(1)}(x_{k-1}) \\ &= y_{k-1} + hf(x_{k-1}, y_{k-1}); \qquad k = 1, 2, \ldots \end{aligned}$$

Note that for $k = 0$, $y = y_0(x_0)$ is the initial value and must be defined to obtain a specific solution.

Taylor method of order 2

$$\begin{aligned} y_k &\approx y_{k-1} + hy^{(1)}(x_{k-1}) + \frac{h^2}{2}y^{(2)}(x_{k-1}) \\ &= y_{k-1} + hf(x_{k-1}, y_{k-1}) + \frac{h^2}{2}f'(x_{k-1}, y_{k-1}); \qquad k = 1, 2, \ldots \end{aligned}$$

Taylor method of order N

$$y_k = y_{k-1} + hf(x_{k-1}, y_{k-1}) + \frac{h^2}{2}f'(x_{k-1}, y_{k-1}) + \ldots + \frac{h^n}{n!}f^{(n-1)}(x_{k-1}, y_{k-1}); \qquad k = 1, 2, \ldots$$

In practice it is not necessary to use a higher-order Taylor approximation to every problem. Note that there is a computational overhead associated with the evaluation of the higher order derivatives of the function f. In fact, there is a theorem to clarify when to apply higher order formulae and expect improvement.

Theorem

If Taylor's method of order n is used to approximate the solution to

$$y'(t) = f\left(x, y(x)\right), \quad a \leq x \leq b, \quad y(a) = x_a$$

with step size h one can expect the local truncation error (LTE) to be $\mathcal{O}\left(h^n\right)$ only if $y \in C^{n+1}\left[a, b\right]$.

15.1.3 Code for $N-th$ order Taylor

```
Nstep = nint( (xf - x0)/h )
x = x0
y1 = y0
y2 = y0
y3 = y0
y4 = y0
do ih = 1, Nstep
  !  Euler, First order Taylor
  y1 = y1 + h * f(x, y1)

  !  Second order Taylor
  y2 = y2 + h * f(x, y2) + (h**2/2.0d0) * df(x,y2)

  !  Third order Taylor
  y3 = y3 + h * f(x, y3) + (h**2/2.0d0) * df(x,y3) + &
                           (h**3/6.0d0) * d2f(x,y3)

  !  Fourth order Taylor
  y4 = y4 + h * f(x, y4) + (h**2/2.0d0) * df(x,y4) + &
                           (h**3/6.0d0) * d2f(x,y4) + &
                           (h**4/24.0d0) * d3f(x,y4)

  x = x + h
enddo
```

15.1.3 Runge-Kutta methods

One of the drawbacks of the Taylor methods is the need for derivatives of the function, f. The Runge-Kutta (RK) methods tackle this problem by evaluating more function values at intermediate (fractional) steps. The $N-$th order RK method (RKN method) involves

N function evaluations. Before we look at the expressions at various orders let us look at the best possible LTE that can be obtained with RK formulas that are based on N function evaluations.

N	1	2	3	4	$5 \leq N \leq 7$	$8 \leq N \leq 9$	$10 \leq N$
Best possible LTE	$\mathcal{O}(h)$	$\mathcal{O}(h^2)$	$\mathcal{O}(h^3)$	$\mathcal{O}(h^4)$	$\mathcal{O}(h^{N-1})$	$\mathcal{O}(h^{N-2})$	$\mathcal{O}(h^{N-3})$

From this Table it is clear that the best cost-to-performance ratio can be expected for the RK4 method, which is in fact the reason why this method is being widely applied.

The general formula for the solution according to RKN is given by

$$y_1 = y_0 + h \sum_{k=1}^{N} b_k \phi_k, \tag{122}$$

where each term ϕ_k is a function evaluated as follows

$$\phi_k = f\left(x_0 + c_k h, y_0 + h \sum_{s=1}^{k-1} a_{k,s}\phi_s\right) \tag{123}$$

The values of the constants b_k, c_k, and $a_{k,s}$ are tabulated for various N and can be found in, for example, Wikipedia.

15.1.4 RK1 aka Euler's method

The RK expression for $N = 1$ is

$$\begin{aligned} y_1 &= y_0 + hb_1\phi_1 && (124) \\ &= y_0 + hb_1 f(x_0 + c_1 h, y_0) && (125) \end{aligned}$$

We would like to use the function value at $\{x_0, y_0\}$, which results in $c_1 = 0$. If we compare the resulting equation to a Taylor expansion

$$\begin{aligned} y_1 &= y_0 + hy^{(1)}(x_0) + \mathcal{O}\left(h^2\right) \\ &= y_0 + hf(x_0, y_0), \end{aligned}$$

we note that the constant b_1 has to be 1. Thus we see that the RK1 method is nothing but the Euler's method

$$y_1^{\text{RK1}} = y_0 + hf(x_0, y_0). \tag{126}$$

15.1.5 RK2

The RK expression for $N = 2$ is

$$\begin{aligned} y_1 &= y_0 + hb_1\phi_1 + hb_2\phi_2 && (127)\\ &= y_0 + hb_1 f\left(x_0 + c_1 h, y_0\right) + \\ &\quad hb_2 f\left(c_0 + c_2 h, y_0 + ha_{2,1}\phi_1\right) \\ &= y_0 + hb_1 f\left(x_0 + c_1 h, y_0\right) + \\ &\quad hb_2 f\left(c_0 + c_2 h, y_0 + ha_{2,1} f\left(x_0 + c_1 h, y_0\right)\right) && (128) \end{aligned}$$

Again, we would like to use the function value at $\{x_0, y_0\}$, so let us make $c_1 = 0$, which simplifies the above equation to

$$\begin{aligned} y_1 &= y_0 + hb_1 f\left(x_0, y_0\right) + \\ &\quad hb_2 f\left(x_0 + c_2 h, y_0 + ha_{2,1} f\left(x_0, y_0\right)\right) && (129) \end{aligned}$$

The term $f\left(x_0 + c_2 h, y_0 + ha_{2,1} b_1 f\left(x_0, y_0\right)\right)$ corresponds to a function evaluated with displacement in both x and y variables.

$$\begin{aligned} f\left(x_0 + c_2 h, y_0 + ha_{2,1} b_1 f\left(x_0, y_0\right)\right) &\approx f\left(x_0, y_0\right) + c_2 h \left.\frac{\partial f}{\partial x}\right|_{x_0, y_0} \\ &\quad + ha_{2,1} f\left(x_0, y_0\right) \left.\frac{\partial f}{\partial y}\right|_{x_0, y_0} \end{aligned}$$

Using this approximation, we have

$$\begin{aligned} y_1 &= y_0 + hb_1 f\left(x_0, y_0\right) + \\ &\quad hb_2 f\left(x_0, y_0\right) + b_2 c_2 h^2 \left.\frac{\partial f}{\partial x}\right|_{x_0, y_0} + \\ &\quad b_2 h^2 a_{2,1} f\left(x_0, y_0\right) \left.\frac{\partial f}{\partial y}\right|_{x_0, y_0} && (130) \end{aligned}$$

Now our task is to find the constants b_1, b_2, c_1, c_2 $a_{2,1}$. To find these constants, again, we have to expand y_1 as Taylor polynomial.

$$\begin{aligned} y_1 &= y_0 + hy^{(1)}(x_0) + \frac{h^2}{2} y^{(2)}(x_0) + \mathcal{O}\left(h^3\right) \\ &\approx y_0 + hf\left(x_0, y_0\right) + \frac{h^2}{2} \left.\frac{df}{dx}\right|_{x_0, y_0} \end{aligned}$$

We now expand $\frac{df}{dx}$ as the total derivative

$$\begin{aligned} \frac{df}{dx} &= \frac{\partial f}{\partial x}\frac{dx}{dx} + \frac{\partial f}{\partial y}\frac{dy}{dx} \\ &= \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx} && (131) \end{aligned}$$

$$\begin{aligned}
y_1 &\approx y_0 + hf(x_0, y_0) + \frac{h^2}{2}\left\{\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx}\right\}\bigg|_{x_0,y_0} \\
&= y_0 + hf(x_0, y_0) + \frac{h^2}{2}\frac{\partial f}{\partial x}\bigg|_{x_0,y_0} + \frac{h^2}{2}\frac{\partial f}{\partial y}\frac{dy}{dx}\bigg|_{x_0,y_0} \\
&= y_0 + hf(x_0, y_0) + \frac{h^2}{2}\frac{\partial f}{\partial x}\bigg|_{x_0,y_0} + \frac{h^2}{2}\frac{\partial f}{\partial y}\bigg|_{x_0,y_0} f(x_0, y_0) \quad (132)
\end{aligned}$$

Comparing equations 130 and 132, we find that the constants b_1, b_2, c_1, c_2 $a_{2,1}$ must satisfy the equations

$$\begin{aligned}
b_1 + b_2 &= 1 \\
b_2 c_2 &= 1/2 \\
b_2 a_{2,1} &= 1/2
\end{aligned}$$

For various values of $b_1$we can derive different formula as follows

15.1.5.1 RK2, $b_1 = 0$ aka Mid-point method

With this choice we get $b_2 = 1$, and $c_2 = a_{2,1} = 1/2$. We can plug these in equation 129, to get the RK2 equation

$$y_1^{\mathrm{RK1-mid-point}} = y_0 + hf\left(x_0 + \frac{h}{2}, y_0 + \frac{h}{2}f(x_0, y_0)\right). \quad (133)$$

15.1.5.2 RK2, $b_1 = 1/2$ aka Heun-Euler method

With this choice we get $b_2 = 1/2$, and $c_2 = a_{2,1} = 1$. The corresponding RK2 equation is

$$\begin{aligned}
y_1^{\mathrm{RK1-Heun}} &= y_0 + \frac{h}{2}f(x_0, y_0) + \\
&\quad \frac{h}{2}f(x_0 + h, y_0 + hf(x_0, y_0)) \quad (134)
\end{aligned}$$

15.1.5.3 RK2, $b_1 = 1/3$ aka Ralston's method

With this choice we get $b_2 = 2/3$, and $c_2 = a_{2,1} = 3/4$. The corresponding RK2 equation is

$$\begin{aligned}
y_1^{\mathrm{RK1-Ralston}} &= y_0 + \frac{1}{3}hf(x_0, y_0) + \\
&\quad h\frac{2}{3}f\left(x_0 + \frac{3}{4}h, y_0 + h\frac{3}{4}f(x_0, y_0)\right) \quad (135)
\end{aligned}$$

15.1.6 Code for various RK2

```
Nstep = nint( (xf - x0)/h )
x = x0
y1_mid = y0
do ih = 1, Nstep
  !  RK2, mid-point
  f1 = h * f(x, y1_mid)
  f2 = h * f(x+h/2.0d0, y1_mid+f1/2.0d0)
  y1_mid = y1_mid + f2

  !  RK2, Heun
  f1 = h * f(x, y1_heun)
  f2 = h * f(x+h, y1_heun+f1)
  y1_heun = y1_heun + f1/2.0d0 + f2/2.0d0
  x = x + h

  !  RK2, Ralston
  f1 = h * f(x, y1_rals)
  f2 = h * f(x+(3.0d0/4.0d0)*h, y1_rals+(3.0d0/4.0d0)*f1)
  y1_rals = y1_rals + (1.0d0/3.0d0) * f1 + (2.0d0/3.0d0) *
f2

  x = x + h
enddo
```

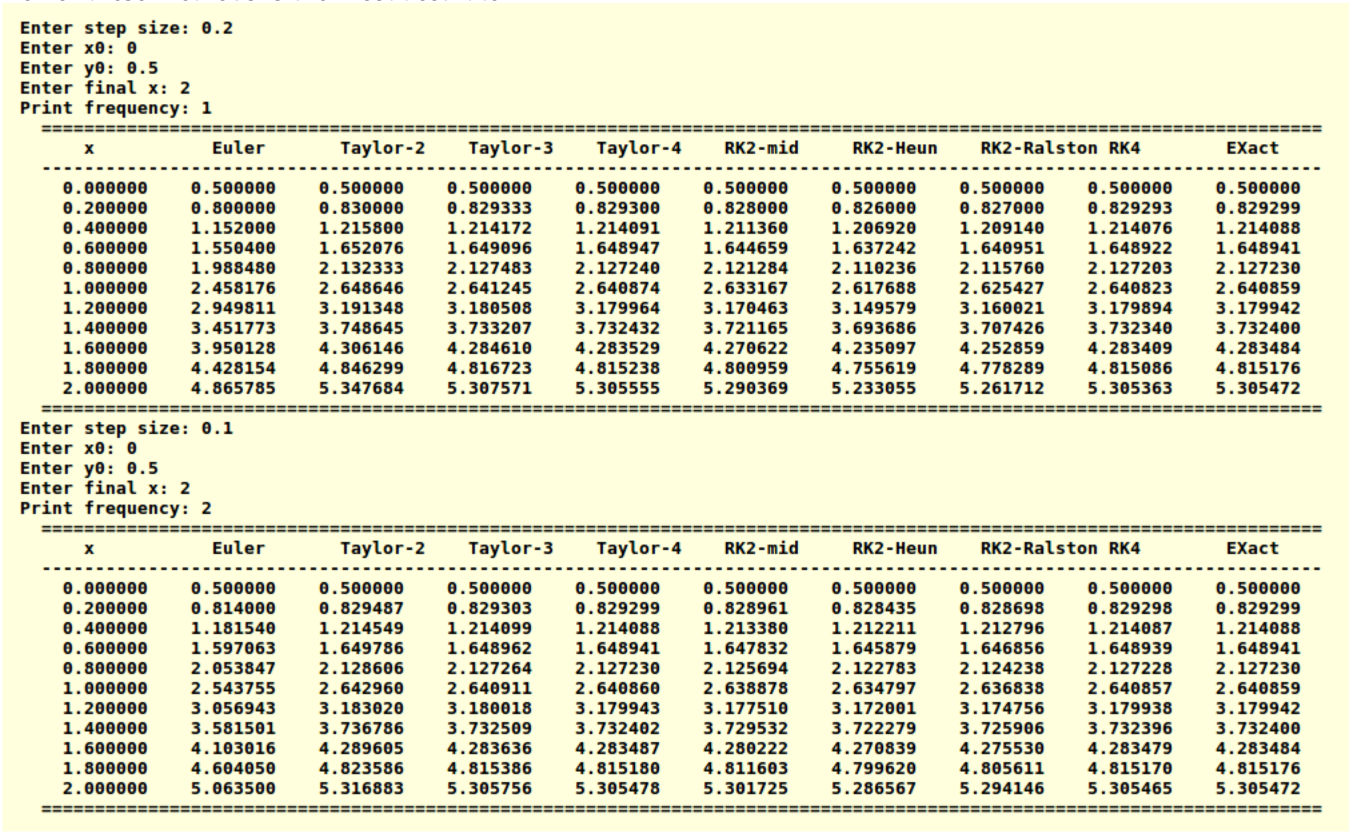
15.1.7 RK4

As we had discussed, RK4 method gives the best performance among the lower order RK methods. This method involved four function evaluations, and results in an LTE which is $\mathcal{O}\left(h^4\right)$. Let us directly look at the code for RK4.

```
Nstep = nint( (xf - x0)/h )
x = x0
y1_mid = y0
do ih = 1, Nstep
  !  RK4
  f1 = h * f(x, y1_mid)
  f2 = h * f(x+h/2.0d0, y1_mid+f1/2.0d0)
  f3 = h * f(x+h/2.0d0, y1_mid+f2/2.0d0)
  f4 = h * f(x+h, y1_mid+f3)
  y1_mid = y1_mid + (f1 + 2.0d0*f2 + 2.0d0*f3 + f4)/6.0d0

  x = x + h
enddo
```

Example 2

Let us look at the ODE $y'(x) = y(x) - x^2 + 1, \quad 0 \leq x \leq 1, \quad y(0) = 0.5$. The exact solution of this ODE is $y(x) = (x+1)^2 - e^x/2$. Here are the results from all the methods that we have discussed so far. Which of these methods is the most accurate?



```
Enter step size: 0.2
Enter x0: 0
Enter y0: 0.5
Enter final x: 2
Print frequency: 1
  =================================================================================================================
     x          Euler       Taylor-2    Taylor-3    Taylor-4    RK2-mid     RK2-Heun    RK2-Ralston RK4         EXact
  -----------------------------------------------------------------------------------------------------------------
    0.000000    0.500000    0.500000    0.500000    0.500000    0.500000    0.500000    0.500000    0.500000    0.500000
    0.200000    0.800000    0.830000    0.829333    0.829300    0.828000    0.826000    0.827000    0.829293    0.829299
    0.400000    1.152000    1.215800    1.214172    1.214091    1.211360    1.206920    1.209140    1.214076    1.214088
    0.600000    1.550400    1.652076    1.649096    1.648947    1.644659    1.637242    1.640951    1.648922    1.648941
    0.800000    1.988480    2.132333    2.127483    2.127240    2.121284    2.110236    2.115760    2.127203    2.127230
    1.000000    2.458176    2.648646    2.641245    2.640874    2.633167    2.617688    2.625427    2.640823    2.640859
    1.200000    2.949811    3.191348    3.180508    3.179964    3.170463    3.149579    3.160021    3.179894    3.179942
    1.400000    3.451773    3.748645    3.733207    3.732432    3.721165    3.693686    3.707426    3.732340    3.732400
    1.600000    3.950128    4.306146    4.284610    4.283529    4.270622    4.235097    4.252859    4.283409    4.283484
    1.800000    4.428154    4.846299    4.816723    4.815238    4.800959    4.755619    4.778289    4.815086    4.815176
    2.000000    4.865785    5.347684    5.307571    5.305555    5.290369    5.233055    5.261712    5.305363    5.305472
  =================================================================================================================
Enter step size: 0.1
Enter x0: 0
Enter y0: 0.5
Enter final x: 2
Print frequency: 2
  =================================================================================================================
     x          Euler       Taylor-2    Taylor-3    Taylor-4    RK2-mid     RK2-Heun    RK2-Ralston RK4         EXact
  -----------------------------------------------------------------------------------------------------------------
    0.000000    0.500000    0.500000    0.500000    0.500000    0.500000    0.500000    0.500000    0.500000    0.500000
    0.200000    0.814000    0.829487    0.829303    0.829299    0.828961    0.828435    0.828698    0.829298    0.829299
    0.400000    1.181540    1.214549    1.214099    1.214088    1.213380    1.212211    1.212796    1.214087    1.214088
    0.600000    1.597063    1.649786    1.648962    1.648941    1.647832    1.645879    1.646856    1.648939    1.648941
    0.800000    2.053847    2.128606    2.127264    2.127230    2.125694    2.122783    2.124238    2.127228    2.127230
    1.000000    2.543755    2.642960    2.640911    2.640860    2.638878    2.634797    2.636838    2.640857    2.640859
    1.200000    3.056943    3.183020    3.180018    3.179943    3.177510    3.172001    3.174756    3.179938    3.179942
    1.400000    3.581501    3.736786    3.732509    3.732402    3.729532    3.722279    3.725906    3.732396    3.732400
    1.600000    4.103016    4.289605    4.283636    4.283487    4.280222    4.270839    4.275530    4.283479    4.283484
    1.800000    4.604050    4.823586    4.815386    4.815180    4.811603    4.799620    4.805611    4.815170    4.815176
    2.000000    5.063500    5.316883    5.305756    5.305478    5.301725    5.286567    5.294146    5.305465    5.305472
  =================================================================================================================
```

Exercises - 11

Exercise 11.1: Higher-order Taylor

In this problem you will learn how Taylor approximations with increasing order result in convergence towards the exact solution.

Use the ODE that has been discussed as an example in the class, $y'(x) = x + y(x) \quad 0 \leq x \leq 10, \quad y(0) = 1$. The exact solution of this ODE is $y(x) = 2e^x - x - 1$. Modify the program listed in the handout to include Taylor methods of orders $n = 1$, 2, 3, and 4. Use $h = 1$. Comment on the convergence of the solution with increasing n. Make a plot of relative errors in each order, i.e, x vs. $\left| \frac{y^{\text{exact}}(x) - y^{\text{num.}}(x)}{y^{\text{exact}}(x)} \right|$.

15.2 Systems of differential equations

An $M-$th order (or M-dimensional) system of first-order initial-value problems has the form

$$\begin{aligned} y_1' &= f_1\left(x, y_1, y_2, \ldots, y_M\right) \\ y_2' &= f_2\left(x, y_1, y_2, \ldots, y_M\right) \\ &\vdots \\ y_N' &= f_M\left(x, y_1, y_2, \ldots, y_M\right) \quad a \leq x \leq b, \end{aligned} \tag{136}$$

with M initial conditions $y_1(a) = a_1$, $y_2(a) = a_2$, etc. Solving this system of ODEs is a very straight-forward extension of the framework we had established previously for a first-order ODE, $y' = f(x, y)$. All we have to do is to implement the "function" as an array, and solve for the vector $\mathbf{y}$.

$$\mathbf{y}' = \mathbf{f}\left(x, \mathbf{y}\right) \tag{137}$$

15.2.1 RK4 for Reaction Rates

Let us see how we can apply RK4 for this problem using an example. Consider the following reversible reaction

$$\mathrm{A} \underset{k_2}{\overset{k_1}{\rightleftharpoons}} \mathrm{B} \tag{138}$$

The differential rate equations give the rate at which the concentration of A and B vary

$$\begin{aligned} y_1' = \frac{d\,[\mathrm{A}]\,(t)}{dt} &= -k_1\,[\mathrm{A}]\,(t) + k_2\,[\mathrm{B}]\,(t) = f_1(t, y_1, y_2) \\ y_2' = \frac{d\,[\mathrm{B}]\,(t)}{dt} &= k_1\,[\mathrm{A}]\,(t) - k_2\,[\mathrm{B}]\,(t) = f_2(t, y_1, y_2) \end{aligned} \tag{139}$$

The exact solution to this system of ODEs can be found in a physical chemistry book as

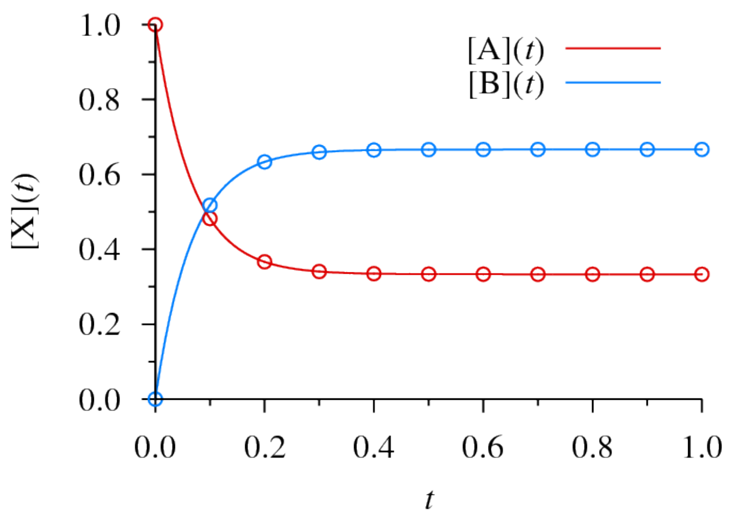
$$\begin{aligned} [\mathrm{A}]\,(t) &= [\mathrm{A}]\,(0) - \left(k_1\,[\mathrm{A}]\,(0) - k_2\,[\mathrm{B}]\,(0)\right)\left[1 - \exp\left(-\tau t\right)\right] \\ [\mathrm{B}]\,(t) &= [\mathrm{B}]\,(0) + \left(k_1\,[\mathrm{A}]\,(0) - k_2\,[\mathrm{B}]\,(0)\right)\left[1 - \exp\left(-\tau t\right)\right], \end{aligned} \tag{140}$$

where τ is the relaxation time, $\tau = k_1 + k_2$. Here is a simple program implementing RK4 to solve the rate equation.

```fortran
program rate_rk4

  implicit none
  double precision            :: x
  double precision            :: x0, y0(1:2), xf
  double precision            :: h
  double precision            :: f1(1:2), f2(1:2), f3(1:2), f4(1:2)
  double precision            :: y(1:2)
  double precision            :: k(1:2)

  integer                     :: dh_print, Nstep
  integer                     :: ih

  write(*,'(a)', advance='no') 'Enter step size: '
  read *, h
  write(*,'(a)', advance='no') 'Enter x0: '
  read *, x0
  write(*,'(a)', advance='no') 'Enter y0(1:2) : '
  read *, y0(1:2)
  write(*,'(a)', advance='no') 'Enter rate constants, k(1:2) : '
  read *, k(1:2)
  write(*,'(a)', advance='no') 'Enter final x: '
  read *, xf
  write(*,'(a)', advance='no') 'Print frequency: '
  read *, dh_print

  Nstep = nint( (xf - x0)/h )
  x = x0
  y(1:2) = y0(1:2)
  print '(3f12.6)', x, y(1:2)

  do ih = 1, Nstep

    ! RK4
    call diff_rate(x,y,k,f1,h)
    call diff_rate(x+h/2.0d0, y+f1/2.0d0, k, f2,h)
    call diff_rate(x+h/2.0d0, y+f2/2.0d0, k, f3,h)
    call diff_rate(x+h, y+f3, k, f4,h)
    y = y + (f1 + 2.0d0 * f2 + 2.0d0 * f3 + f4)/6.0d0

    x = x + h
    if ( mod(ih, dh_print) .eq. 0 ) then
      print '(3f12.6)', x, y(1:2)
    endif

  enddo

end program

subroutine diff_rate(x,y,k,f,h)
  implicit none
  double precision, intent(in)  :: x, y(1:2), k(1:2), h
  double precision, intent(out) :: f(1:2)
  f(1) = -k(1) * y(1) + k(2) * y(2)
  f(2) =  k(1) * y(1) - k(2) * y(2)
  f = f * h
end subroutine
```

- Note the difference in the structure of this program and the previously discussed program for first-order ODE
- The RHS function is coded in a subroutine
- Note down the data that are declared as arrays

For the initial values of $[\mathrm{A}](0) = 1$, $[\mathrm{B}](0) = 0$, and with the rate constants $k_1 = 10$, and $k_2 = 5$, the following plot shows the exact solution, along with RK4 values computed with a step size of 0.01.



15.2.2 *Complex problems*

Recall that exact analytic solutions to differential rate equations require steady-state-approximation. Numerical solutions on the other hand, will give numerically exact solutions. Let us inspect couple of interesting reactions

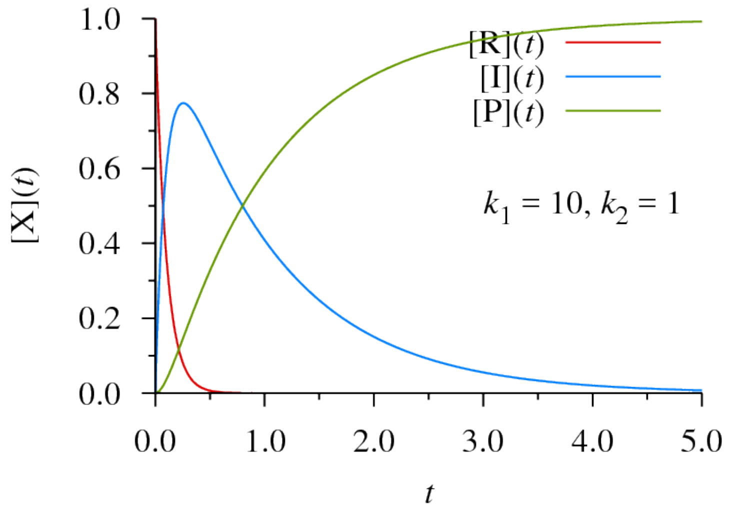
Example 1: Reactant-to-Intermediate-to-Product

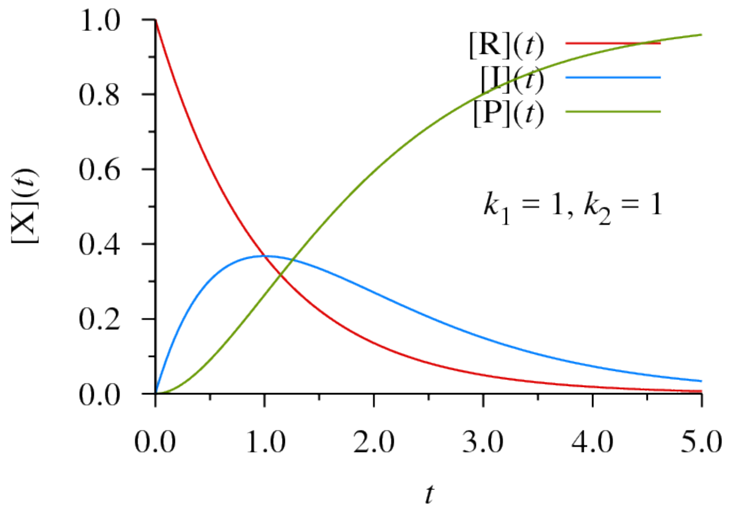
$$\mathrm{R} \overset{k_1}{\Rightarrow} \mathrm{I} \overset{k_2}{\Rightarrow} \mathrm{P} \tag{141}$$

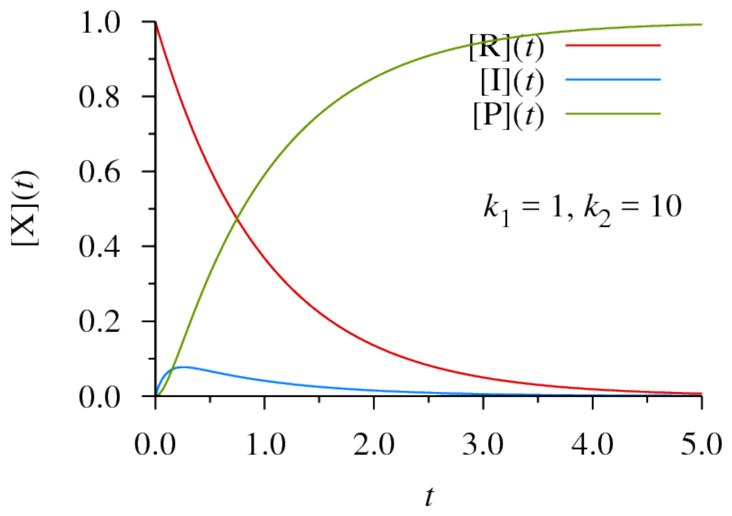
The differential rate equations are

$$\begin{aligned} \frac{d\,[\mathrm{R}]\,(t)}{dt} &= -k_1\,[\mathrm{A}]\,(t) \\ \frac{d\,[\mathrm{I}]\,(t)}{dt} &= k_1\,[\mathrm{A}]\,(t) - k_2\,[\mathrm{I}]\,(t) \\ \frac{d\,[\mathrm{B}]\,(t)}{dt} &= k_2\,[\mathrm{I}]\,(t) \end{aligned} \tag{142}$$

The following plot shows the results for the three interesting limits $k_1 \gg k_2$, $k_1 = k_2$ and $k_1 \ll k_2$.







Example 2: A complex problem

The inorganic redox reaction

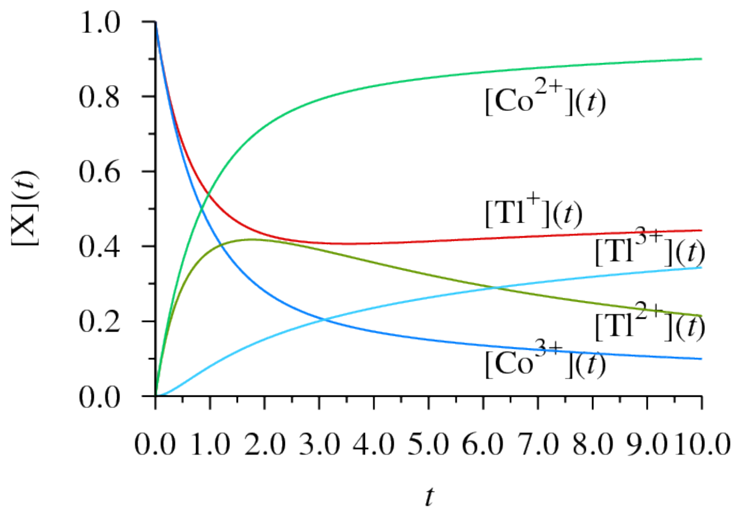
$$\mathrm{Tl}^{+} + 2\mathrm{Co}^{3+} \longrightarrow \mathrm{Tl}^{3+} + 2\mathrm{Co}^{2+} \tag{143}$$

proceeds with a complicated mechanism

$$\mathrm{Tl}^{+} + \mathrm{Co}^{3+} \underset{k_2}{\overset{k_1}{\rightleftharpoons}} \mathrm{Tl}^{2+} + \mathrm{Co}^{2+} \tag{144}$$

$$\mathrm{Co}^{3+} + \mathrm{Tl}^{2+} \xrightarrow{k_3} \mathrm{Co}^{2+} + \mathrm{Tl}^{3+} \tag{145}$$

For the values of $[\mathrm{Tl}^{+}](0) = 1$, $[\mathrm{Co}^{3+}](0) = 1$, and with the rate constants $k_1 = 1$, $k_2 = 0.25$, and $k_3 = 0.5$, we get the following numerical solution using RK4.



Exercises - 12

Exercise 12.1: butadiene: cis-trans

The (cis-trans) isomerization of butadiene is first order in both directions. At 25°C, the equilibrium constant K is 0.406 and the forward (cis to trans) rate constant is $4.21 \times 10^{-4}\,\mathrm{s}^{-1}$. Recall that $K = k_{\text{forward}}/k_{\text{reverse}}$.

1. Starting with a sample of the pure cis isomer with initial concentration $0.115\,\mathrm{mol.dm}^{-3}$, numerically find the time it would take for half the equilibrium amount of the trans isomer to form.

2. Make a plot of the instantaneous concentration of the reactant and the product.

15.3 Higher-order differential equations

Several physical problems such as vibration of a string, motion of a pendulum/spring, time-independent Schrödinger equation, are initial value problems whose equations have order larger than 1. In general an $m-$th order initial value problem has the form

$$y^{(m)} = f\left(x, y, y^{(1)}, \ldots, y^{(m-1)}\right), \quad a \leq x \leq b, \tag{146}$$

along with m initial conditions $y(a) = a_1$, $y^{(m)}(a) = a_m$, etc. In order to solve this equation, we have to consider $y(x)$ and its $m-1$ derivatives as m arbitrary functions of x, i.e., $\left[\begin{array}{cccc} y & y^{(1)} & \cdots & y^{(m-1)} \end{array}\right] = \left[\begin{array}{cccc} u_1 & u_2 & \cdots & u_m \end{array}\right] = \mathbf{u}$. Thus, we can transform a single $m-$th order initial value problem into a system with m first-order equations, i.e., $y^{(m)} = f\left(x, y, y^{(1)}, \ldots, y^{(m-1)}\right) = \mathbf{u}' = \mathbf{f}(x, \mathbf{u})$. The last equation corresponds to that of a system of first-order equations, written in vector notation.

15.3.1 Example

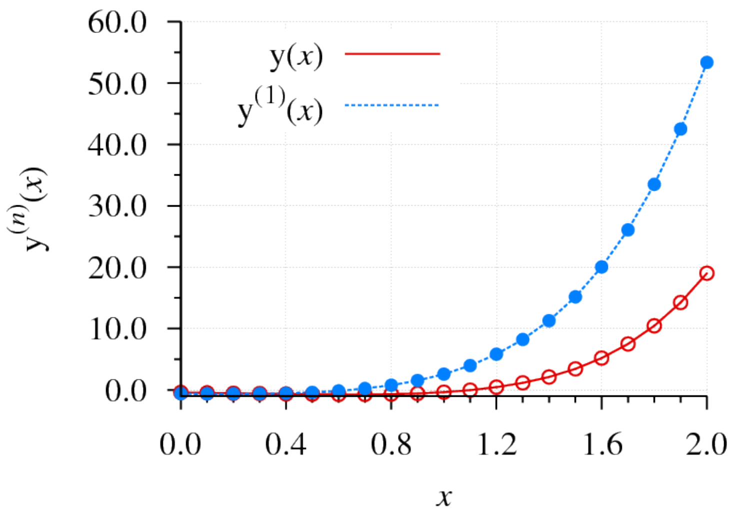
Let us solve the second-order differential equation $y^{(2)} - 2y^{(1)} + 2y = \exp(2x)\sin(x), \quad 0 \leq x \leq 2$, with the two initial conditions, $y(0) = -0.4$, and $y^{(1)}(0) = -0.6$. The exact solution of this equation is $y(x) = \frac{1}{5}\left[e^{2x}\left(\sin(x) - 2\cos(x)\right)\right]$. Let us start with the substitutions

$$u_1(x) = y(x) \qquad \text{and} \qquad u_2(x) = u^{(1)}(x) = y^{(1)}(x). \tag{147}$$

The second-order ODE can now be written as a system of the two first-order ODEs

$$\begin{aligned} u_1^{(1)}(x) &= u_2(x) = f_1(x, u_1, u_2) \\ u_2^{(1)}(x) &= \exp(2x)\sin(x) + 2u_2(x) - 2u_1(x) = f_2(x, u_1, u_2) \end{aligned} \tag{148}$$

Using the RK4 program discussed in the last class, we can readily compute the numerical solution. The following plot shows the exact analytic solution, its analytic derivative (lines in the plot) along with their numerical (RK4) approximations (circles in the plot) computed with step size 0.1.



15.3.2 Simple pendulum - trajectories and phase portrait

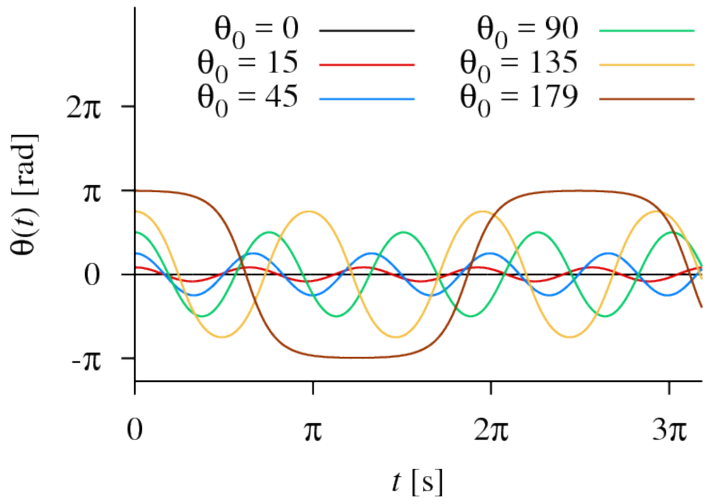
The motion of a swinging pendulum, under certain conditions (no air-friction, uniform gravity, planar motion, bob is a particle, massless rod) can be derived as

$$\theta^{(2)}\left(t\right) + \frac{g}{L}\sin\left(\theta\right) = 0, \tag{149}$$

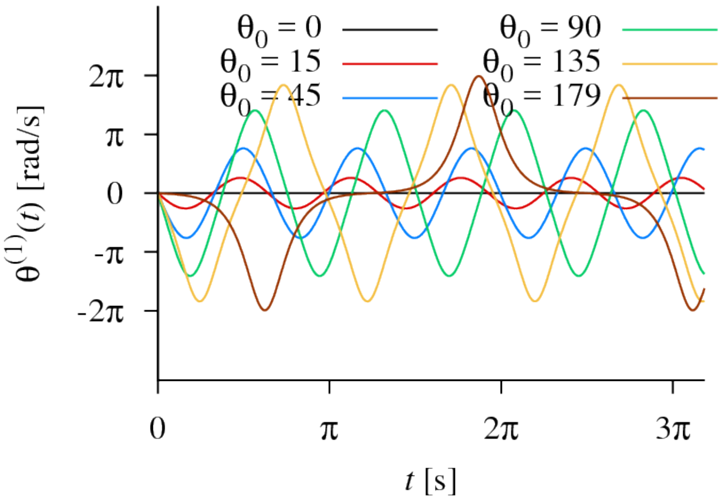
where g is the acceleration due to gravity $9.8\,\mathrm{m/s^2}$., and L is the pendulum's length. Check out the wiki article (`https://en.wikipedia.org/wiki/Pendulum_(mathematics)`) to learn about many ways to derive this equation.

Analytic solution of this equation is possible only when further approximations are incorporated, such as the harmonic approximation that is valid for small anglular displacements. When the angle is small, we have $\sin\left(\theta\right) \approx \theta$, then the analytic solution is given by $\theta\left(t\right) = \theta_0 \cos\left(\sqrt{\frac{g}{L}}t\right), \qquad \theta_0 \ll 1.$

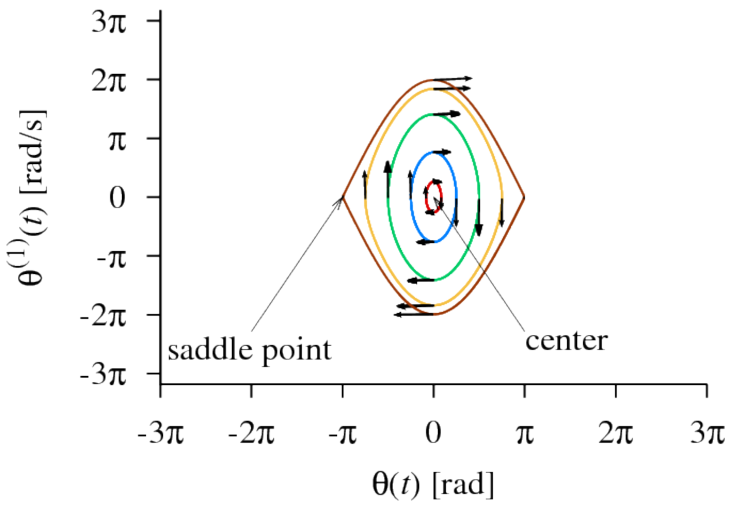
Using the numerical approach we can easily compute the solution for any arbitrary initial value. Let us fix the pendulum's length to be $L = 1\,\mathrm{m}$. The following plot shows the time-dependent variation of the angle of the pendulum, for zero initial velocity, $\theta_0' = 0$, and for different initial displacements, $\theta_0 = 0, 10, 45, 90, 135, 179, 180$ degrees.



For the same initial conditions, angular velocities, $\theta'(t)$, have also been computed in the same calculation, and are shown in the following plot.

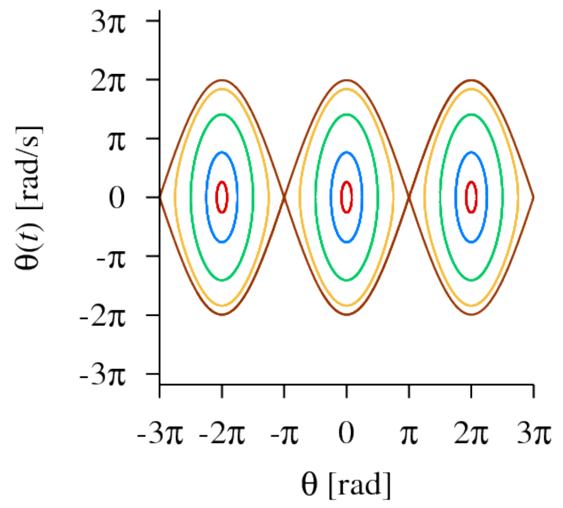


When the variation of velocities (here angular velocities) is plotted as a function of displacement (here angular displacements), we get a trajectory. When this trajectory is plotted for various initial conditions, we obtain a phase portrait. The following plot shows part of the phase-portait of the swinging pendulum. The arrows in the plot point to the directions along which the displacements and velocities change.



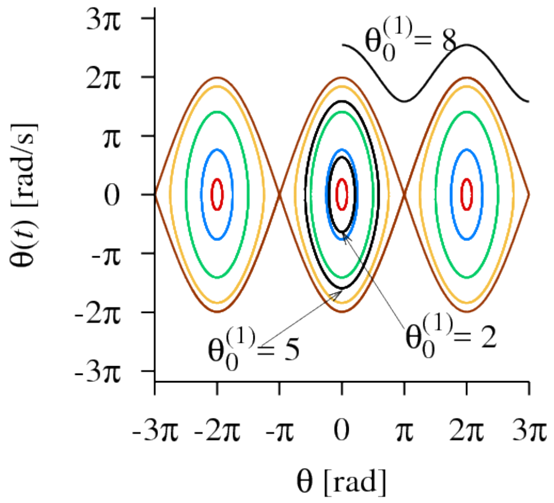
The points where the potential has extrema, are called as critical points. The two critical points shown in the above figure are stable and unstable equilibrium points. The stable equilibrium point is also called as the center. Any small displacement from the center will take the pendulum back to its initial position – hence stable point. On the other hand, a small displacement from an unstable equilibrium point (saddle point) will take the pendulum, away from this point.

When the initial displacements are varied in the range $-3\pi \leq \theta \leq -\pi$ or $\pi \leq \theta \leq 3\pi$, we obtain phase portraits which shows repeated regions.



If we simulate the dynamics with non-zero initial velocities (but with zero displacement), we see some interesting dynamics. For small initial velocities such as 2 or 5 rad/s, the motion remains periodic, i.e., the pendulum swings from $\theta_0 = 0$ to 90 degrees (where the velocity becomes zero), and turns back and reaches 0 degrees again (with the velocity reversed, i.e., oppposite direction), and reaches -90 degrees (where the velocity is zero again). Again the pendulum changes its direction eventually crossing the initial position $\theta_0 = 0$ with the same initial velocity. Overall the motion is periodic and oscillatory. Such "closed-loop" trajectories are also known as finite solutions.

For larger initial velocities, the periodicity is lost. As shown in the plot below, for an initial velocty of 8 rad/s, the pendulum crosses 90 degrees and starts to rotate instead of swinging back. This rotational motion will continue for ever, because we do not have air-resistance in the model to dampen the motion. Such trajectories are called as infinite solutions, and the corresponding motion is non-periodic. The $\theta_0 \approx 180°$ trajectory that separates qualitatively different dynamical regions is called as the separatrix curve. A separatrix always passes through a point of unstable equilibrium.



Useful references

1. Uncle Google
2. http://linuxcommand.org/
3. http://www.unixcl.com/
4. http://www.yolinux.com/
5. http://tldp.org/LDP/Bash-Beginners-Guide/html/index.html
6. http://www.gnu.org/software/coreutils/manual/html_node/index.html
7. http://www.tutorialspoint.com/unix/index.htm
8. http://www.tutorialspoint.com/fortran/index.htm
9. https://gcc.gnu.org/onlinedocs/gfortran/Intrinsic-Procedures.html
10. Fortran 90/95 for Science and Engineering, edition 2, S. J. Chapman, McGraw Hill Education India Private Limited (Indian Edition, ~Rs. 800).
11. Numerical Methods, W. Boehm, H. Prautzsch, Universities Press, 2000. (Indian Edition, ~Rs. 200).
12. Numerical Analysis, R. L. Burden, J. D. Faires, edition 9, Cengage Learning (Indian Edition, ~Rs. 600).
13. Introduction to Numerical Computation, L. Eldén, L. Wittmeyer-Koch, H.B. Nielsen, Overseas Press, 2006 (Indian Edition, ~Rs. 500).
14. Gnuplot in Action, Understanding data with graphs, P. K. Janert, Manning Publications Co., 2010.
15. http://folk.uio.no/hpl/scripting/doc/gnuplot/Kawano/index-e.html
16. Numerical Mathematics, M. Grasselli, D. Pelinovsky, Narosa, 2009. (Indian Edition.~Rs. 500).