

Homework Problem 1

10 points

Due date: Thursday 22 May 2008

(a) Be sure to download the tridiagonal solver from my web site. If you're not using FORTRAN or C, you'll need to translate that into your language. See me if needed. This is not part of the assignment, but I figure that I have your attention right now, so I'm taking advantage of the opportunity.

(b) This is the assignment. Type the following code, using the editor of your choice (probably WordPad or NotePad in windows, and kyle, emacs or vi in linux). This is to acquaint you with the basics of your compiler, and check on a few basic computational issues that are compiler-dependent. Save it as hw1.f.

```
program hw1
implicit none
double precision x,y,z

write(6,*) 'Test 1'
x = 1
write(6,*) x
x = 1.
write(6,*) x
x = 1.d0
write(6,*) x
write(6,*)

write(6,*) 'Test 2'
x = -3.d0
write(6,*) x**2
write(6,*) x**2.
write(6,*) x**2.d0
write(6,*)

write(6,*) 'Test 3'
write(6,*) x**2.001d0
write(6,*) x**2.001

stop
end
```

Submit a copy of your code and a printout of your output. You can cut and paste into a text document if you wish.

- Test 1. Some older compilers screw up a simple assignment like this.  $x$  is declared as double precision, so it should be stored as 1.000000000000000. In the first line, 1 is an integer and all compilers should convert it to double precision properly. In the second line, 1. is single precision real, and some compilers will convert the single precision to double precision without adding the extra precision! If you have a ‘bad’ compiler, you will see something like 1.0000034958478 – the conversion is accurate only to single precision. **The third line is what you should be doing all the time – write ALL real values as double precision constants.** The ‘d’ is the double precision version of scientific notation. Note that the star format in the write means the number will be printed as efficiently as possible, so an output of 1 means you have the correct value 1.000000000000000 in memory.
- Test 2. You would like to see all 3 answers come out as 9.000000000000000. But again, some compilers handle the type (integer, real or double) of the exponent differently. If you get NAN (not a number) for the second and third lines, this means the compiler is not smart enough to recognize that you want to compute  $x^x$ . Instead it is computing  $x^2 = \exp(2 \ln x)$ , which gives an error because you’ve put a negative number into the natural log. **When the power is an integer, you should type it as an integer – use the first format.**
- Test 3. This forces the compiler to use  $x^2 = \exp(2.001 \ln x)$ , which will give an error. This is more for your benefit than a check of the compiler – the only way to ‘fix’ this is to convert to complex variables, but even then you may have problems with how the branch cut for the natural log is defined.