1 Worksheet 6: Arrays and Libraries

In this worksheet we will reuse the code from the previous worksheet in which we timed the execution time of our error function $\mathrm{erf}(x)$ for different values of x. In that worksheet you simply printed the results to standard output in the form of a table. But of course it would be nice to visualize these results.

We will now plot the execution time of `ErfOfReal`, $t(x)$, versus x, using a ready made graphics library called DISLIN and a module that Aleksandar Donev made with easy-to-use plotting routines. This will introduce you to using ready-made libraries of Fortran functions. Since the routines in DISLIN accept arrays and strings as arguments, we will need to introduce simple arrays, discussed in sections 1.9 (exclude 1.9.3 for now) and 2.6 (read subsections 1,2,3,4 and 9) in the Fortran manual, as well as section 1.5 on character strings (not neccessary).

1.1 Arrays

In the previous assignment, we had a `DO` loop (different people used different strategies) which invoked the function `ErfOfReal` for several values of x, storing the result into a temporary variable `erf`, printing the result, and then going on to a new iteration.

Now, assume you really needed to save the results for some post-processing. Since there are an unknown and possibly large number of values we need to store, arrays are an obvious task for the choice. So we will declare two new allocatable arrays, say `x_values` and `elapsed_time`, and store the values of x and the values of $t(x)$ in them. Make these arrays single precision since this is the kind of arrays DISLIN accepts:

```
REAL(KIND=sp), DIMENSION(:), ALLOCATABLE :: x_values, elapsed_time
...
ALLOCATE(x(n_points),elapsed_time(n_points)) ! Allocate
...
dx=(x_max-x_min)/REAL(n_points-1) ! The step size
x=x_min ! Initialize
DO i=1,n_points
  elapsed_time(i)=ErfTiming(x) ! In us
  x_values(i)=x ! Store x in memory
  x=x+dx ! Increment
```

```
END DO
```

Now, we have two arrays in memory holding the values x_i and $t(x_i)$, and we are going to pass them as arguments to plotting routines to vizualize them. Before you try that though, print your arrays to make sure they contain the correct numbers, simply using `WRITE`:

```
WRITE(*,*) ''x   :'',x_values
WRITE(*,*) ''t(x):'',elapsed_time
```

Only after this works, go to the next section.

1.2 Using external libraries

One of the most important things about Fortran is that there are literally thousands of high-quality libraries of scientific (especially numerical) procedures written and ready-to-be-used in Fortran. Using these libraries is a balance between knowledge and ignorance. A beginner need only know how to use the libraries. In Fortran 90 terms, he needs to know the `INTERFACE` of each of the routines in the library—is it a `FUNCTION` or a `SUBROUTINE`, how many and what type arguments does it accept, what does each argument mean, etc. As you use a library for more and more sophisticated tasks, you need to learn more about its inner functionings. In this worksheet you will go through the first two steps in using a library—learning how to call the routine you need from the provided documentation (in this case HTML), and then actually using the routine with your Fortran compiler (this requires *linking libraries*).

1.3 The `SimpleGraphics` module

The DISLIN library is a high-level graphics library that has a Fortran 90 interface. However using is not a trivial task. Therefore, we made a simple module called `SimpleGraphics` that you need to `USE` it in your program. This module has documentation provided in HTML format at:

http://computation.pa.msu.edu/phy201/SimpleGraphics.html

Read this documentation and in particular take a look at the example `TestSimple_2D.f90`. You need not worry about the 3D routine `SurfPlot` at this time. As any documentation, this document may contain information that you can not understand or need not use at the moment. But it is good excercise to learn how to extract the relevant info from the document.

We will not give an example of using this library in this worksheet. The example in the documentation is enough. Just remember which arrays you need to pass to the routine `Plot2D` as `x` and `y` data points.

1.4 Compiling the program

After you write your program that `USE`s the module `SimpleGraphics` and its routines, you need to compile it. This process is tricky, so we have made some shortcuts for you.

First, at the Linux command prompt, type:

```
>  source /classes/phy201/SimpleGraphics.init
```

which will read in an environmental variable `$DISLINvf90` which has all the neccessary commands to link in the appropriate files (if you logged to `gauss` using your class account this would have been done automatically!). Just append `$DISLINvf90` at the end of your compilation line, as in:

```
>  f90-vast erf_plot.f90 -o erf_plot.x erf_series.o erf_timing.o $DISLINvf90
```

The last worksheet explained why you need to link the `.o` files. Compiled working programs for both the example `TestSimple_2D.f90` and `erf_plot.f90` are in our `/classes/phy201` directory under the same name but with a `.x` extension. Try these programs to see how they work. Notice the proper labeling of the axis, the labels and the legends on the plots. Add as many of these embelishments as you have time to.