# Writing your own functions into PDL

## Using PDL Functions

PDL shares the Perl method for building functions for code that perform a commonly repeated function - you can define a function with `sub`, a function name, and a pair of curly braces.

Here's a simple function in a PDL script:

```
#!/usr/bin/perl
use PDL;
$a = sequence(10);
$b = $a * 4;

$result = my_sums($a, $b);
print $result;

print my_sums(pdl(10,20,30), pdl(3,4,5) );

sub my_sums {
    my ($a1, $a2) = @_; # pdls passed in the perl array @_

    my $c = $a1 + $a2;

    my $difference = $a1 - $a2;

    return($c, $difference);
}
```

As you can see, the function is called `my_sums` and it is defined at the end of this script, but you can be define functions anywhere in the script. In the example above, we call `my_sums` twice, printing out the answers as we go.

You can return as many PDLs from the function as you want, by passing them out in a comma separated list.

You can define the functions in any part of your Perl script, even after the point in the program that you started using the function. The input variables to the function are passed through the `@_` array, and we can put any set of piddles into the function. The function also has local scope, so variables inside the routine are not seen by anything calling that function. Remember, though, that the variables *outside* the function **can** be seen inside the function! It's good practice to have a `use strict;` inside your functions whilst writing them, though, as this will help catch bugs.

## Moving Functions into Separate Files

It gets tiring to copy and paste your useful functions from script to script, and so PDL provides a way to have your functions stored in a file that can be read by many scripts.

Two important notes:

- The filename has `.pdl` at its end, not `.pl`
- The file has `1;` as the last line outside the curly braces of the statement.

Use a file editor and cut and paste the text below into a file with the name `my_sums.pdl`.

```
sub my_sums {
    ($a1, $a2) = @_;
    my $c = $a1 + $a2;
    my $difference = $a1 - $a2;
```

```
    return($c, $difference);
}
1;
```

Now create a separate script in the same directory:

```
#!/usr/bin/perl
use PDL;
use PDL::AutoLoader;
$a = sequence(10);
$b = $a * 4;
$result = my_sums($a, $b);
print $result;
print my_sums(pdl(10,20,30), pdl(3,4,5) );
```

Running this Perl script will include PDL, and PDL will automatically look for a file called `my_sums.pdl` (remember the extension has to have `.pdl`) and use it.

### Getting PDL to look for your functions in other places

After a while, you will have many PDL functions scattered over many directories, and so it makes sense to collect all your functions into a separate directory and have PDL look for them there.

You can set an environment variable in your shell called `PDLLIB` to look within a given directory. One convention is to use `PDLLIB=${HOME}/lib/pdl+` to store all your functions. When defined in your system shell, this will inform PDL where you've put your commonly used functions. The directory path is specified to your PDL library directory. The + sign at the end of the path tells PDL to also look in all the subdirectories below the PDLLIB directory.

### Passing Options to your function with Hashes

Sometimes you need to pass in one or more options into your PDL function, and these are typically passed into a PDL routine using an anonymous hash. As an example of this, we'll modify the routine above to optionally print the result of the function:

```
sub my_sums {
    ($a1, $a2) = @_;
    my $c = $a1 + $a2;
    my $difference = $a1 - $a2;
    return($c, $difference);
}
1;
```

### Documenting your Functions

Just like any other Perl script, you can add Plain Old Documentation (POD) inside your PDL functions.

For a detailed look at what you can have in a POD, look at *perlpod* with `perldoc perlpod` or look online for a tutorial.

At the bare minimum, the POD should say what the PDL function does, what are its inputs and outputs. Further detail may include one or two examples so that a new user can test it and check they understand what the function behaves, look at *perlpod* with `perldoc perlpod` or look online for a tutorial.