ftof — Passing functions to functions

Description

Functions can receive other functions as arguments.

Below is described (1) how to call a function that receives a function as an argument and (2) how to write a function that receives a function as an argument.

Syntax

```
example(..., &somefunction(), ...)
```

where *example*() is coded

```
function example(..., f, ...)
{
        ...
        (*f)(...)
        ...
}
```

Remarks and examples

Remarks are presented under the following headings:

Passing functions to functions

Someone has written a program that receives a function as an argument. We will imagine that function is

```
real scalar fderiv(function(), x)
```

and that `fderiv()` numerically evaluates the derivative of *function*() at *x*. The documentation for `fderiv()` tells you to write a function that takes one argument and returns the evaluation of the function at that argument, such as

```
real scalar expratio(real scalar x)
{
        return(exp(x)/exp(-x))
}
```

To call `fderiv()` and have it evaluate the derivative of `expratio()` at 3, you code

```
fderiv(&expratio(), 3)
```

To pass a function to a function, you code `&` in front of the function's name and `()` after. Coding `&expratio()` passes the address of the function `expratio()` to `fderiv()`.

Writing functions that receive functions, the simplified convention

To receive a function, you include a variable among the program arguments to receive the function—we will use *f*—and you then code `(*`*f*`)(...)` to call the passed function. The code for `fderiv()` might read

```
function fderiv(f, x)
{
        return( ((*f)(x+1e-6) - (*f)(x)) / 1e-6 )
}
```

or, if you prefer to be explicit about your declarations,

```
real scalar fderiv(pointer scalar f, real scalar x)
{
        return( ((*f)(x+1e-6) - (*f)(x)) / 1e-6 )
}
```

or, if you prefer to be even more explicit:

```
real scalar fderiv(pointer(real scalar function) scalar f,
                            real scalar x)
{
        return( ((*f)(x+1e-6) - (*f)(x)) / 1e-6 )
}
```

In any case, using pointers, you type `(*f)(...)` to execute the function passed. See [M-2] **pointers** for more information.

Aside: the function `fderiv()` would work but, because of the formula it uses, would return very inaccurate results.

Passing built-in functions

You cannot pass built-in functions to other functions. For instance, [M-5] **exp()** is built in, which is revealed by [M-3] **mata which**:

```
: mata which exp()
  exp():  built-in
```

Not all official functions are built in. Many are implemented in Mata as library functions, but `exp()` is built in and coding `&exp()` will result in an error. If you wanted to pass `exp()` to a function, create your own version of it

```
: function myexp(x) return(exp(x))
```

and then pass `&myexp()`.

Also see

[M-2] **Intro** — Language definition

For suggested citations, see the FAQ on citing Stata documentation.