Solving Systems of Nonlinear Equations: Getting to know `fsolve`

1 The Problem

Write a `MATLAB` function `foobar` to solve the following system of two nonlinear equations in two unknowns, x and y:

$$g(x,y) = \sin(xy) + ax + by \tag{1}$$
$$h(x,y) = e^{x+y} + cx + dy \tag{2}$$

where the parameters $\{a, b, c, d\}$ are user-specified, on-the-fly, as arguments to a function that we are going to write. We want to find x and y such that $g = 0$ and $h = 0$ simultaneously[1].

2 Application Programming Interface

2.1 Function: `foobar.m`

2.1.1 Usage

```
[z] = foorbar(zo,a,b,c,d)
```

2.1.2 Input Arguments

- **zo** is a (2×1) matrix containing the user-specified initial guesses

$$\mathbf{zo} = \begin{bmatrix} xo \\ yo \end{bmatrix} \tag{3}$$

- the four parameters $\{a, b, c, d\}$ are user-specified scalars.

2.1.3 Output Arguments

- **z** is the (2×1) matrix of the solution to the system of nonlinear equations.

$$\mathbf{z} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{4}$$

3 Meet `fsolve`

We will use the `fsolve` command in `MATLAB` to complete this task. See `http://www.mathworks.com/help/optim/ug/fsolve.html` for a complete description of `fsolve`. The following description is extracted from this Mathworks site.

[1] This is similar to a problem given in CE 3101, but the inclusion of four user-specified parameters adds an additional level of difficulty.

`fsolve` finds a root (zero) of a system of nonlinear equations.

- `x = fsolve(fun,x0)` starts at x0 and tries to solve the equations described in fun.
- `x = fsolve(fun,x0,options)` solves the equations with the optimization options specified in the structure options. Use optimset to set these options.
- `[x,fval] = fsolve(fun,x0)` returns the value of the objective function fun at the solution x.
- `[x,fval,exitflag] = fsolve(...)` returns a value exitflag that describes the exit condition.

The input argument `fun` defines the nonlinear system of equations to solve. The following description is extracted from the Mathworks site.

> fun is a function that accepts a vector x and returns a vector f, the nonlinear equations evaluated at x. The function fun can be specified as a function handle for a file
>
> `x = fsolve(@myfun,x0)`
>
> where myfun is a MATLAB function such as
>
> ```
> function f = myfun(x)
> f = ...            % Compute function values at x
> ```

4 Version 0.1

It looks as though `fsolve` does everything that we need. Perhaps we should dispense with the extra layer defined as `foobar` and simply call `fsolve` directly on the command line.

```
>> [z] = fsolve(@myfun,zo);
```

Of course, we still need to write the problem-defining function `myfun`. Looking at the documentation we see that `myfun` "is a function that accepts a vector x and returns a vector f, the nonlinear equations evaluated at x." We create our `myfun.m` file so that is looks like the following.

```
function [f] = myfun(z)
    f(1) = sin(z(1)*z(2)) + a*z(1) + b*z(2);
    f(2) = exp(z(1)+z(2)) + c*z(1) + d*z(2);
end
```

Then we execute the following on the command line.

```
>> zo = [0;0];
>> [z] = fsolve(@myfun,zo);
```

The result is a bunch of red ink.

```
Undefined function or variable 'a'.

Error in myfun (line 2)
    f(1) = sin(z(1)*z(2)) + a*z(1) + b*z(2);

Error in fsolve (line 241)
    fuser = feval(funfcn{3},x,varargin{:});

Caused by:
    Failure in initial user-supplied objective function evaluation.
    FSOLVE cannot continue.
```

5 Version 0.2

Ah ha! Our `myfun` does not know the four user-defined parameters, {`a,b,c,d`}. Let's pass these in as parameters to `myfun`. We edit `myfun` to read as follows.

```
function [f] = myfun(z, a,b,c,d)
    f(1) = sin(z(1)*z(2)) + a*z(1) + b*z(2);
    f(2) = exp(z(1)+z(2)) + c*z(1) + d*z(2);
end
```

so that all of the necessary parameters are defined. Then we execute the following on the command line.

```
>> zo = [0;0];
>> [z] = fsolve(@myfun,zo);
```

Again, the result is a bunch of red ink. A new error, to be sure, but an error nonetheless.

```
Error using myfun (line 2)
Not enough input arguments.

Error in fsolve (line 241)
    fuser = feval(funfcn{3},x,varargin{:});

Caused by:
    Failure in initial user-supplied objective function evaluation.
    FSOLVE cannot continue.
```

6 Rooting out the problem

The source of this error is an incompatibility between what `fsolve` demands and what `myfun` provides. The built-in `MATLAB` function `fsolve` is a generic routine that is meant to work for

a wide range of problems. The `fsolve` function does NOT know anything about the extra parameters that `myfun` wants.

The requirement specified by `fsolve` is `myfun` "is a function that accepts a vector x and returns a vector f." This statement from Mathsoft is a bit too loose, it should say: "... is a function that accepts **one and only one argument**, a vector x, and returns a vector f." Our version 0.2 of `myfun` does NOT satisfy this specification requirement.

7 Solving the paradox

"*What we've got here is failure to communicate.*"[2] Our `myfun` wants the four user-defined parameters, {`a,b,c,d`}. `MATLAB`'s `fsolve` says "*I know nothing, I hear nothing, I see nothing*"[3] about these parameters, and they are not my responsibility.

This is a very common problem in computer programming when using generic functions like `fsolve`. Mathsoft provides an extensive discussion of the issue and offers multiple solutions. Mathsoft call this the "passing extra parameters" problem. See `http://www.mathworks.com/help/optim/ug/passing-extra-parameters.html`.

8 Using *a nested function*

We are going to solve this problem using a *nested function* inside of a call to our driver function `foobar`.

```
function [z] = foobar(zo,a,b,c,d)
    [z] = fsolve(@foo,zo);

    function [f] = foo(z)
        x = z(1);
        y = z(2);

        f(1) = sin(x*y) + a*x + b*y;
        f(2) = exp(x+y) + c*x + d*y;
    end
end
```

Note that the function `foo` is nested inside of function `foobar`. Function `foo` may have its own private variables – like `x` and `y`. Function `foo` may also use the variables that its parent, function `foobar`, knows – like the four parameters {`a,b,c,d`}.

Using this nested function approach our function `foo` is serving in the role as `myfun`, not the parent function *foobar*. Function `foo` satisfies all of the strict requirements demanded by `MATLAB`'s `fsolve`, but still has access to the user-specified parameters.

[2]This is a pop-culture reference that most of you do not recognize, since it was introduced long before you were born. Hmmm... so is old pop culture still pop culture? Google "failure to communicate" if you would like more background, but skip over the *Guns N' Roses* references as they came later.

[3]This is another ancient pop culture reference, sorry. I will try and work "Thrift Shop" into a future lecture.

9 Version 1.0

We write the nested function version of `foobar`, as given above, and then we execute the following on the command line.

```
>> foobar([0;0], -1, -1, 1, -1)
```

The resulting output is not red, but it is rather wordy.

```
Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>

ans =

   -0.5068
    0.3369
```

10 Version 1.1

While it is nice that `MATLAB` tells us the happy message "Equation solved"... well, it is nice the first time, especially after our previous failures. But, it is not so nice each and every time. Usually, we simply want the answer.

We will use the `options` to turn the display messages off, and then check the `exitflag` to make certain the we converged. We only print out a message if there is a possible problem.

```
function [z] = foobar(zo,a,b,c,d)
    [z,~,exitflag] = fsolve(@foo,zo, optimset('Display','off'));
    if exitflag ~= 1
        warning('exitflag = %d', exitflag);
    end

    function [f] = foo(z)
        x = z(1);
        y = z(2);

        f(1) = sin(x*y) + a*x + b*y;
        f(2) = exp(x+y) + c*x + d*y;
    end
end
```

11 How does `fsolve` work?

As an interesting aside, if you set the 'Display' option to 'iter' instead of 'off', MATLAB writes out detailed, iteration-by-iteration results. This is a useful tool for debugging and for understanding how `fsolve` works.

If we make this dispay change and execute the following on the command line

```
>> foobar([0;0], -1, -1, 1, -1)
```

MATLAB produces the following output.

```
                                        Norm of      First-order   Trust-region
 Iteration  Func-count     f(x)          step         optimality    radius
     0          3              1                          2              1
     1          6       0.0612087       0.707107          0.367          1
     2          9     0.000173451       0.166663         0.0238       1.77
     3         12     5.34561e-10     0.00770501       3.33e-05       1.77
     4         15     1.77696e-20    1.40058e-05       1.94e-10       1.77

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>


ans =

   -0.5068
    0.3369
```

There are many other options for `fsolve`. You should make yourself aware of the range of possibilities. The most important are the options used to define the tolerances and stopping criteria.