Function evaluation

Spring 2019

fplot

- `plot` plots a function by connecting defined data points.

fplot

- `plot` plots a function by connecting defined data points.
- poorly chosen data points may result in inacurate plots.

```
1                    x = 0:pi/8:2*pi
2                    y = sin(8*x);
3                    plot(x,y,'b')
```

- This code results in a straight line, which is clearly wrong!

Definining function - Anonymous functions

- A function that is *not* stored in a program file, but is asscociated with a variable whose data type is `function_handle`, e.g

Defininging function - Anonymous functions

- A function that is *not* stored in a program file, but is asscociated with a variable whose data type is `function_handle`, e.g
- Scalar anonymous function

```
>> f1=@(x) cos(x).*sin(2*x)
f1 =
    @(x)cos(x).*sin(2*x)
>> f1(-1)
ans =
    -0.4913
```

Anonymous functions

- 2D anonymus function

```
>> g1=@(x,y) sin(x.^2 + y.^2)./(x.^2 +y.^2)
g1 =
    @(x,y)sin(x.^2+y.^2)./(x.^2+y.^2)
>> g1(0.01,0.01)
ans =
    1.0000
```

fplot(f,xinterval)

- `fplot(f,xinterval)` - plots y=f(x) over the specified interval

```
1                f = @(x) sin(8*x);
2                fplot(f,[0 2*pi],'r');
3                title('$sin(8x)$','Interpreter','latex')
```

- `fplot` has a relative error tolerance of 2e-03

Defining functions in MATLAB

Inline functions

- Useful for defining a function that will only be used in the current MATLAB session. e.g.

Defining functions in MATLAB

Inline functions

- Useful for defining a function that will only be used in the current MATLAB session. e.g.
- scalar functions:

```
>> f=inline('cos(x).*sin(2*x)')
f =
     Inline function:
     f(x) = cos(x).*sin(2*x)
>> f(-1)
ans =
     -0.4913
```

inline functions

- 2D functions

```
>> g=inline('sin(x.^2 + y.^2)./(x.^2 +y.^2)')
g =
     Inline function:
     g(x,y) = sin(x.^2 + y.^2)./(x.^2 +y.^2)
>> g(0.1,0.1)
ans =
     0.9999
```

- 3D functions

```
>> h=inline('sin(x+y+z)')
h =
     Inline function:
     h(x,y,z) = sin(x+y+z)
>> h(1,1,1)
ans =
     0.1411
```

Indirect function evaluation

```
[y1,. . . ,yN] = feval(fun,x1,. . .,xM)
```

- Evaluates a function using its name or its handle using inputs `x1,. . .,xM`, e.g

```
>> feval('cos',0.5)
ans =
    0.8776
```

Indirect function evaluation

```
[y1,. . . ,yN] = feval(fun,x1,. . .,xM)
```

- Evaluates a function using its name or its handle using inputs `x1,. . .,xM`, e.g

```
>> feval('cos',0.5)
ans =
    0.8776
```

```
>> g=inline('sin(x.^2 + y.^2)./(x.^2 +y.^2)');
>> feval(g,0.01,0.01)
ans =
     1.0000
```

Indirect function evaluation

```
[y1,. . . ,yN] = feval(fun,x1,. . .,xM)
```

- Evaluates a function using its name or its handle using inputs `x1,. . .,xM`, e.g

```
>> g1=@(x,y) sin(x.^2 + y.^2)./(x.^2 +y.^2);
>> feval(g1,0.01,0.01)
ans =
    1.0000
```

Indirect function evaluation

```
[y1,. . . ,yN] = feval(fun,x1,. . .,xM)
```

- Evaluates a function using its name or its handle using inputs `x1,. . .,xM`, e.g

```
1 >> g1=@(x,y) sin(x.^2 + y.^2)./(x.^2 +y.^2);
2 >> feval(g1,0.01,0.01)
3 ans =
4      1.0000
```

- Note: if you define your own scalar function in a file `f1.m` you can also use feval to evaluate it as `>>feval('f1',x)`

Other useful commands

Command	Result
`fplot('f',[x1, x2])`	`plots the function f on [x1,x2]`
`fminbnd('f',x1,x2)`	`returns x, a local min of f on [x1,x2]`
`fzero('f',x0)`	`returns the root of f near x0`

Other useful commands

Command	Result
`fplot('f',[x1, x2])`	`plots the function f on [x1,x2]`
`fminbnd('f',x1,x2)`	`returns x, a local min of f on [x1,x2]`
`fzero('f',x0)`	`returns the root of f near x0`

`Anonymous` vs `inline`? - use `Anonymous`, it appears MATLAB will remove `inline` in future releases.

bisect.m

Code a function `bisect.m` that implements the bisection method.

```
1                function [c, iteration_count] = bisect(f,a, ...
                     b, epsilon)
2  % bisect.m
3  % Implements the bisection method
4  % Input: a -- nonlinear function such that f(x)=0
5  %        a -- the left endpoint of the interval
6  %        b -- the right endpoint of the interval
7  %        epsilon --  the error tolerance/accuracy
8  %
9  % Output: c --  the approxmiation to the root of f
10 %          iteration_count  --  the number of iterations ...
       to obtain convergence
11 % usage: >>bisect(f,a, b, epsilon)
```

- Test your code with $f(x) = e^x - sin(x)$ on $[-4, 3]$, use `fzero` to confirm that your root is correct.
- Modify your code to implement Newton's method.