

# MATLAB debugger example

- `debug_example.m`: runs the function `unit_scale_data` on two arrays; it fails on the second call because the input is 2-D.

```
function debug_example
```

```
x = [1 2 3 4]
```

```
xn = unit_scale_data(x)
```

```
% second call will cause an error - x is 2-D.
```

```
x = magic(2)
```

```
xn = unit_scale_data(x)
```

# MATLAB debugger example

- Output will be:

```
>> debug_example
x =
     1     2     3     4
xn =
         0    0.3333    0.6667    1.0000
x =
     1     3
     4     2
??? Error using ==> minus
Matrix dimensions must agree.
```

Note that the “correct” answer for the second call should be:

```
x =
     1     3
     4     2
x =
     0    0.6667
 1.000    0.3333
```

# MATLAB debugger example

- Setting `dbstop if error` halts execution at the location of the error, but the function **cannot** be resumed (see `help dbstop`)
- If you wrap the call within a `try ... catch ... end`, it is possible to resume the function
- Change the calls inside `debug_example` to `unit_scale_data_catch`, and run again

# try / catch

- The try block is executed; if a run time error occurs, the remainder of try block is **skipped** and the catch block is executed.
- If no error occurs, the catch block is not executed

```
try
    output_data = output_data - data_min;
    output_data = output_data / (data_max - data_min);
catch
    disp('Error during data rescaling')
end
```

# MATLAB debugger example

- New result – error is “caught”, x is returned, unmodified:

```
>> debug_example
x =
     1     2     3     4
xn =
     0   0.3333   0.6667   1.0000
x =
     1     3
     4     2
Error during data rescaling
xn =
     1     3
     4     2
```

Now, set the debugger to catch a caught error:  
`dbstop if caught error`

# Caught errors

- MATLAB is now halted at line 24, but when you resume, it will skip past the catch block to line 26. So, if we want continue the function, we need to do two things:
  - 1. Correct the problem that leads to the error – in this case it is the fact that `data_min` and `data_max` are not scalars. Here is one possible fix:
    - `data_min=min(min(data));`  
`data_max=max(max(data));`
  - 2. Rerun the code inside the catch block, that generated the error, which is being skipped:
    - `output_data=output_data-data_min;`  
`output_data=output_data/(data_max-data_min);`
- Then you can continue the function with `dbcont`

- Here is what a “corrected” run might look like:

These commands would be the ones typed in at the command line, while the program is halted. Note the “K>>” prompt.

```
>> debug_example
```

```
x =
```

```
    1    2    3    4
```

```
xn =
```

```
    0    0.3333    0.6667    1.0000
```

```
x =
```

```
    1    3
```

```
    4    2
```

```
Caught-error breakpoint was hit in unit_scale_data_catch at  
line 24. The error was:
```

```
Error using ==> minus
```

```
Matrix dimensions must agree.
```

```
24      output_data = output_data - data_min;
```

```
K>> data_max = max(max(data)); data_min = min(min(data));
```

```
K>> output_data = output_data - data_min;
```

```
K>> output_data = output_data / (data_max - data_min);
```

```
K>> dbcont
```

```
Error during data rescaling
```

```
xn =
```

```
    0    0.6667
```

```
  1.0000    0.3333
```

```
>>
```



# Bottom line(s)

- Don't depend on this to “fix” code. If you find a runtime bug, it is always better to actually edit the code to fix the problem, and re-start the execution from the beginning.
- But, if you have a troublesome function call (meaning, you think it might cause a runtime error), you can put `try...catch...end` around it, and potentially “recover” a program run.
- This assumes the fix is easy enough to be done at the command line.
- The example was very simple – a real program might not be fixable at the command line!