MATLAB Programming

Basic Flow Control

MATLAB is a programming language

- Flow control:
  - if ï else if ï else
  - for
  - while
- Functions

Data parallel approach: donî use flow control when something else is better.

```
Example:
For m in M:
    If m < 50:
        A(m)
    else:
        B(m)
Oré
A (M < 50)
B (M<50)
```

These are equivalent.

Disease statistics over time:

ï One row per patient

ï Columns are hourly responsive T cell counts

ï Replace noisy values with local interpolation

Time	…	12:00	13:00	14:00	15:00	16:00
Person23	é	145.35	135.23	0	128.01	126.34

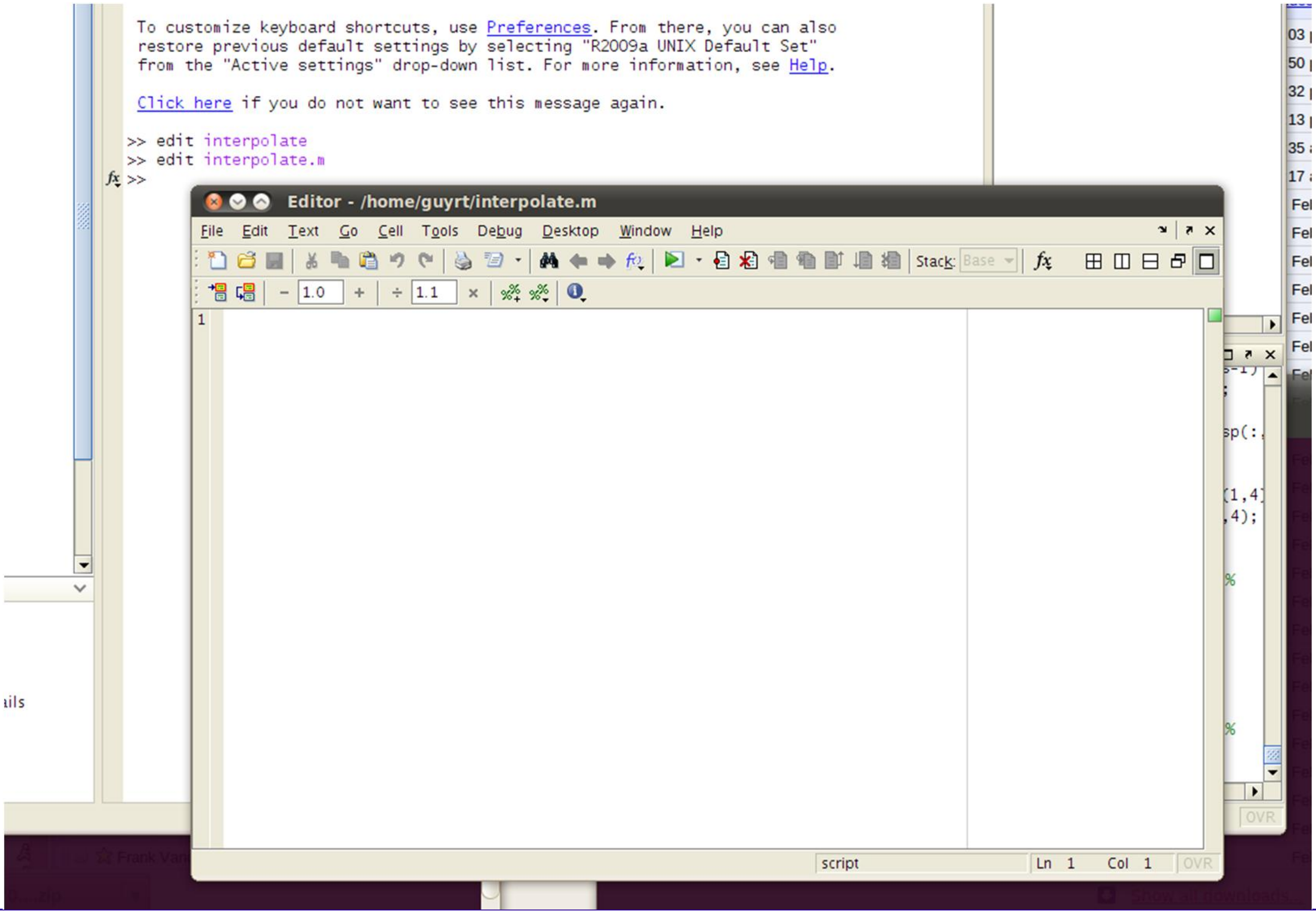
Time	…	12:00	13:00	14:00	15:00	16:00
Person23	é	145.35	135.23	0	128.01	126.34

Use the two closest data points.

(135.23+128.01) / 2 = 131.62

Time	…	12:00	13:00	14:00	15:00	16:00
Person23	é	145.35	135.23	131.62	128.01	126.34

Functions in MATLAB should go in files with the same name and a ñmòextension.



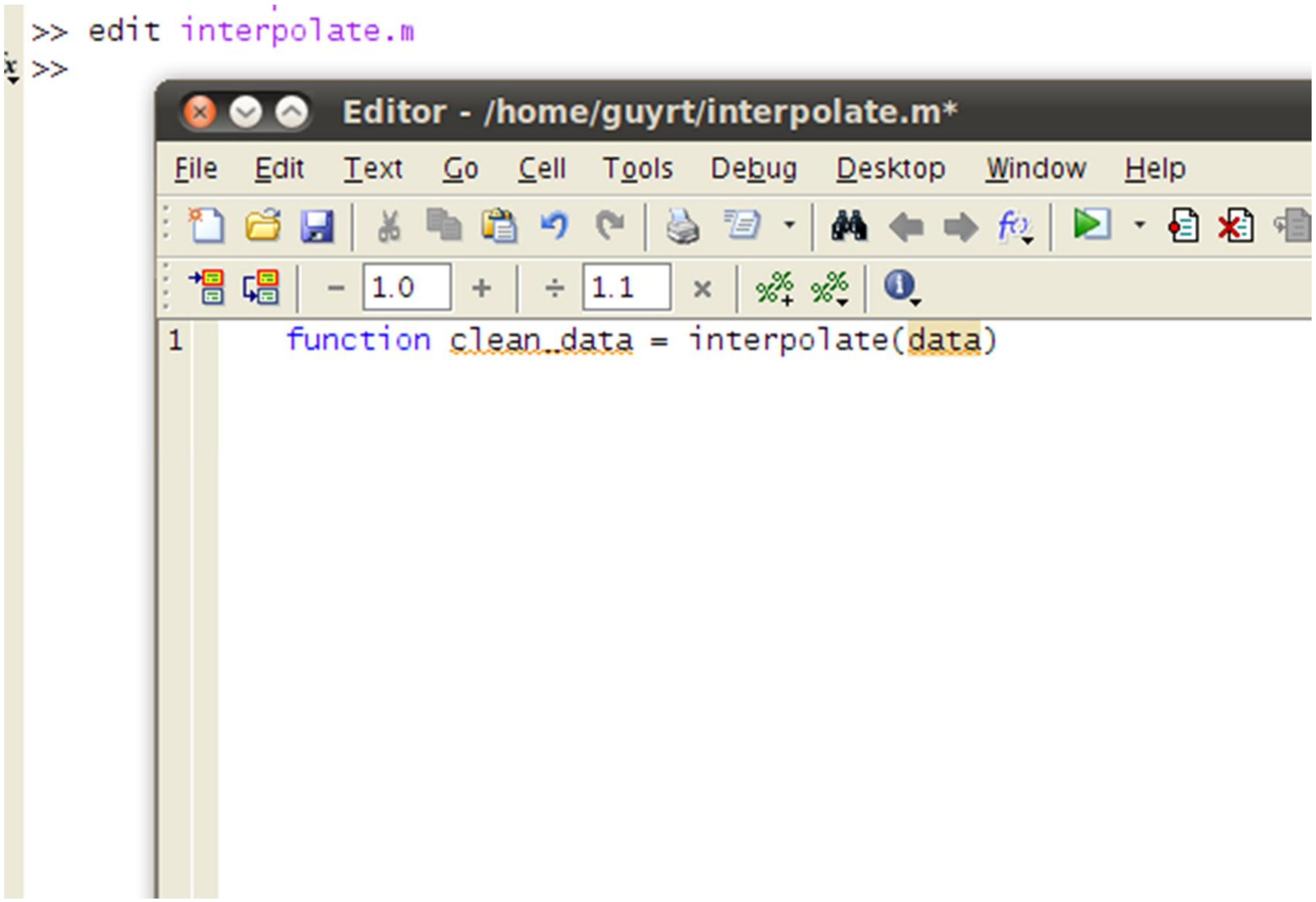
```
To customize keyboard shortcuts, use Preferences. From there, you can also
restore previous default settings by selecting "R2009a UNIX Default Set"
from the "Active settings" drop-down list. For more information, see Help.

Click here if you do not want to see this message again.

>> edit interpolate
>> edit interpolate.m
>>
```

Editor - /home/guyrt/interpolate.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
1.0
1.1
script
Ln 1 Col 1
OVR

The first line of your function file is the header:



```
>> edit interpolate.m
>>
```

Editor - /home/guyrt/interpolate.m*
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1

```
function clean_data = interpolate(data)
function clean_data = interpolate(data)
```

Naive implementation:

```
For each person
  For each data value
     If it the value is 0, replace it
     with the local interpolate.
```

MATLAB for loops:

```
>> for i = 1:5
>>   i
>> end
```

Loop variable

MATLAB for loops:

```
>> for i = 1:5
>>    i
>> end
```

Loop sequence

Loop sequences:

a:b Loop between a and b inclusive, jumping by one.

a:c:b Loop between a and b inclusive, jumping by c.

MATLAB for loops:

```
>> for i = 1:5
>>  i
>> end
```

What will this display?

MATLAB for loops:

```
>> for i = 1:5
>>  i
>> end
   1
   2
   3
   4
   5
```

What will this display?

```
function clean_data = interpolate(data)


[ppl_count, measure_count] = size(data);
```

```
function clean_data = interpolate(data)

[ppl_count, measure_count] = size(data);
clean_data = data;
```

```
function clean_data = interpolate(data)
[ppl_count, measure_count] = size(data);
clean_data = data;
for person = 1:ppl_count
  for measurement = 2:(measure_count-1)
     if clean_data(person, measurement) < EPS
        clean_data(person, measurement) =
           (data(person, measurement - 1) +
           data(person, measurement + 1))/2;
     end
  end
end
```

```
if clean_data(person, measurement) < EPS
```

EPS is roughly 2e-16.

Why not test whether the value == 0?

Tests of equality with floating point numbers are often wrong:

0.0 != 0.000000000000000000000000000001

What about a return statement?

When a function ends, the last value assigned to each return variable is returned.

How much work does the program do?

```
for person = 1:ppl_count
  for measurement = 1:measure_count
    if clean_data(person, measurement) < EPS
```

…

Is this data parallel?

Data Parallel:

Let MATLAB identify the locations that are zero and only loop over those locations.

134.23	139.34	145.35	123.94	0	126.41	121.04
135.31	0	145.35	135.23	133.42	128.01	126.34

$\downarrow$ MATLAB function ñfind non-zerosò

Row	Column
1	5
2	2

MATLABôs find function returns all non-zero elements.

[I,J] = find(M) # Return locations such that M(I,J) are all nonzero entries.

Use ñhelp findò to investigate other parameter and return sets that find provides.

How can we use ôfindô to find zeros rather than nonzeros?

```
function clean_data = interpolate(data)

[I, J] = find( data < EPS );
```

134.23	139.34
0	143.2
135.31	0

data < EPS

0	0
1	0
0	1

[I,J] = find(é)

I	J
2	1
3	2

```
function clean_data = interpolate(data)

[I, J] = find( data(2:end-1) < EPS );

clean_data = data;

for i = 1:length(I)

  person = I(i);

  measurement = J(i);

  clean_data(person, measurement) =
     data(person,measurement-1) +
     data(person,measurement+1) / 2;

end
```

Have we reduced the amount of work that the computer has to do?

Not really: somewhere, a loop is checking for all entries in ódataô that are zero.

But we are using a library routine, so:

- It is faster.
- It has been debugged.

Have we reduced the amount of work that the computer has to do?

Not really: somewhere, a loop is checking for all entries in ódataô that are zero.

But we are using a library routine, so:

It is faster.

It has been debugged.

Can we do even better?

```
function clean_data=interpolate(data)

[I, J] = find( data(2:end-1) < EPS );

clean_data = data;

clean_data(I,J) = data(I,J-1) +
  data(I,J+1) / 2;
```

MATLAB provides a complete set of flow control structures:

While

If ï else if ï else

For

But you should think carefully before you use them.

Before you write a function:

Is there a pre-built function that I can use instead?

Use MATLABôs help function to get a long list of available functions.

Before you write a loop or an if statement:

Is there a data parallel way to do the same thing?

created by

Richard T. Guy

February 2011