A first test

Three bad comments in this Gaussian elimination code. Find them!

\[
A = \begin{bmatrix}
10 & 9 & 8 & 7 \\
1 & 2 & 3 & 2 \\
10 & 5 & 6 & 9 \\
7 & 8 & 10 & 9 \\
\end{bmatrix}
\]

\[
b = \begin{bmatrix}
4 \\
5 \\
6 \\
5 \\
\end{bmatrix}
\]

\[
n = \text{length}(A); \quad \text{computes size of } A
\]

\[
x = \text{zeros}(n, 1);
\]

\[
\text{for } k = 1:4
\]
\[
\quad [s, t] = \text{max}(\text{abs}(A(k, k:n)));
\]
\[
\quad t = t + k - 1;
\]
\[
\quad A([t \ k], :) = A([k \ t], :);
\]
\[
\quad b([t \ k]) = b([k \ t]);
\]
\[
\quad b(k+1:end) = b(k+1:end) - A(k+1:end, k) / A(k, k) \times b(k);
\]
\[
\quad A(k+1:end, k:end) = A(k+1:end, k:end) - A(k+1:end, k) / A(k, k) \times A(k, k:end);
\]
\[
\text{end}
\]

\[
\text{%Rueckwaertseinsetzen}
\]

\[
\text{for } k = 4:-1:1
\]
\[
\quad x(k) = 1 / A(k, k) \times (b(k) - A(k, k+1:n) \times x(k+1:n));
\]
\[
\text{end}
\]
Unit tests in Matlab
and other good coding practices

Federico Poloni

TU Berlin
Supported by Alexander Van

Tools Seminar
What is unit testing?

- Writing procedures to test that your code works
  - Yes, every little piece of code
- Running them automatically
Why should I bother?

“This is just temporary code for testing, it will never be released, so why should I bother?”

- Extremely useful to spot errors immediately
- Forces you to write well-structured code
- Doesn’t cost you much time
  - Saves you a lot of time in the long run! (“future you” will confirm)
Good code always pays off

“This is just temporary code for testing, it will never be released, so why should I bother?”

- After 6 months, the referee report arrives and you need to re-run the experiments
- Another student is taking up your project
- Some random guy reads your article and sends you a mail asking for the code (yes, it happens)
Good code always pays off

“This is just temporary code for testing, it will never be released, so why should I bother?”

- After 6 months, the referee report arrives and you need to re-run the experiments
- Another student is taking up your project
- Some random guy reads your article and sends you a mail asking for the code (yes, it happens)

When you write tests/comments, write them for “future you”
Some more motivation

- You write GaussianElimination, and it works
- You write ComplicatedAlgorithm which uses GaussianElimination
- You modify GaussianElimination. You introduce a bug, for instance in pivoting, but you do not realize immediately because you do not need pivoting in ComplicatedAlgorithm at the moment
- You change ComplicatedAlgorithm. Now you need pivoting
- ComplicatedAlgorithm stops working!
  - You need to debug code in two different places. It’s a mess.

(time spent reading code/debugging) ≫ (time spent writing code)
Solution

There are three wrong comments in this code for Gaussian elimination. Can you find them?

A=[10 9 8 7;1 2 3 2 ; 10 5 6 9 ; 7 8 10 9];
b=[4; 5; 6; 5];

n=length(A); %computes size of A
x=zeros(n,1);
for k=1:4
    [s,t]=max(abs(A(k,k:n)));
t=t+k-1;
    A([t k],:)=A([k t],:);
b([t k])=b([k t]);
    b(k+1:end)=b(k+1:end)-A(k+1:end,k)/A(k,k)*b(k);
    A(k+1:end,k:end)=A(k+1:end,k:end)-A(k+1:end,k)/A(k,k)*A(k,k:end);
end

%%%% Rueckwaertsetzen
for k=4:-1:1
    x(k)=1/A(k,k)*(b(k)-A(k,k+1:n)*x(k+1:n));
end
Solution

There are three wrong comments in this code for Gaussian elimination. Can you find them?

A=[10 9 8 7;1 2 3 2 ; 10 5 6 9 ; 7 8 10 9];
b=[4; 5; 6; 5];

n=length(A); %computes size of A
x=zeros(n,1);
for k=1:4
    [s,t]=max(abs(A(k,k:n)));
    t=t+k-1;
    A([t k],:)=A([k t],:);
    b([t k])=b([k t]);
    b(k+1:end)=b(k+1:end)-A(k+1:end,k)/A(k,k)*b(k);
    A(k+1:end,k:end)=A(k+1:end,k:end)-A(k+1:end,k)/A(k,k)*A(k,k:end);
end
%Rueckwaertseinsetzen
for k=4:-1:1
    x(k)=1/A(k,k)*(b(k)-A(k,k+1:n)*x(k+1:n));
end
Solution

There are three wrong comments in this code for Gaussian elimination. Can you find them?

\[
A = \begin{bmatrix}
10 & 9 & 8 & 7 \\
1 & 2 & 3 & 2 \\
10 & 5 & 6 & 9 \\
7 & 8 & 10 & 9
\end{bmatrix},
\]
\[
b = [4; 5; 6; 5];
\]

\[
n = \text{length}(A); \quad \% \text{computes size of } A
\]
\[
x = \text{zeros}(n, 1);
\]

for \( k = 1:4 \)
\[
[s, t] = \text{max}(\text{abs}(A(k,k:n)));
\]
\[
t = t + k - 1;
\]
\[
A([t \ k], :) = A([k \ t], :);
\]
\[
b([t \ k]) = b([k \ t]);
\]
\[
b(k+1:end) = b(k+1:end) - A(k+1:end, k) / A(k, k) * b(k);
\]
\[
A(k+1:end, k:end) = A(k+1:end, k:end) - A(k+1:end, k) / A(k, k) * A(k, k:end);
\]
end

\%
\[
\text{Rueckwaertseinsetzen}
\]

for \( k = 4:-1:1 \)
\[
x(k) = 1 / A(k, k) * (b(k) - A(k, k+1:n) * x(k+1:n));
\]
end

Wow, I’d never have figured out...
Solution

There are three wrong comments in this code for Gaussian elimination. Can you find them?

```matlab
A=[10 9 8 7; 1 2 3 2; 10 5 6 9; 7 8 10 9];
b=[4; 5; 6; 5];

n=length(A); %computes size of A
x=zeros(n,1);
for k=1:4
    [s,t]=max(abs(A(k,k:n)));
t=t+k-1;
A([t k],:)=A([k t],:);
b([t k])=b([k t]);
b(k+1:end)=b(k+1:end)-A(k+1:end,k)/A(k,k)*b(k);
A(k+1:end,k:end)=A(k+1:end,k:end)-A(k+1:end,k)/A(k,k)*A(k,k:end);
end
%Rueckwaertseinsetzen
for k=4:-1:1
    x(k)=1/A(k,k)*(b(k)-A(k,k+1:n)*x(k+1:n));
end
```

Future you: wait, what's going on here?
Improved version

A=[10 9 8 7;1 2 3 2 ; 10 5 6 9 ; 7 8 10 9];
b=[4; 5; 6; 5];

n=length(A);
x=zeros(n,1);
for k=1:4
    [s,t]=max(abs(A(k,k:n))); %pivoting
    t=t+k-1; %converts relative index in k:n to absolute
    A([t k],:)=A([k t],:);
    b([t k])=b([k t]);
    b(k+1:end)=b(k+1:end)-A(k+1:end,k)/A(k,k)*b(k);
    A(k+1:end,k:end)=A(k+1:end,k:end)-A(k+1:end,k)/A(k,k)*A(k,k:end);
end
%backward elimination
for k=4:-1:1
    x(k)=1/A(k,k)*(b(k)-A(k,k+1:n)*x(k+1:n));
end
You already test your code

I’m sure you already do this...

```matlab
>> GaussianElimination
>> norm(A*x-b)
ans =
   1.0204e-14
>>
```

We just have to make it run automatically
For this, your code must be reusable
Write a real function, not a script!
Function and test code

```matlab
function x=GaussianElimination(A,b)
    n=length(A);
x=zeros(n,1);
for k=1:4
    [s,t]=max(abs(A(k,k:n))); %pivoting
% [snip]
for k=4:-1:1
    x(k)=1/A(k,k)*(b(k)-A(k,k+1:n)*x(k+1:n));
end
And, in another file

function testGaussianElimination
    A=[10 9 8 7;1 2 3 2 ; 10 5 6 9 ; 7 8 10 9];
b=[4; 5; 6; 5];
x=GaussianElimination(A,b);
if norm(A*x-b) > 1e-10
    error 'Test failed';
end
```
This is basically what unit testing means. Now,

- Add one or two more tests. E.g. special cases ($n==1$)
- Run them often (when you commit, every morning when you start working . . .)
- Take actions when they fail
- Make your code more modular, so that it is easier to test
Benefits

When you add the `n==1` you will find this error

```matlab
for k=1:4  %should have been 1:n
```

As promised,

- You spot errors more promptly
- Your code is more organized and readable

```matlab
function x=GaussianElimination(A,b)
```
Now you may add documentation (think to future you!)

```matlab
function x=GaussianElimination(A,b)
% solves a linear system via Gaussian elimination
%
% x=GaussianElimination(A,b)
%
% A: matrix, b: right-hand side, x: solution
```

- You can use `help`, `lookfor`...
- It helps to write docs before functions: helps you picturing the program flow before writing
xUnit is a small Matlab toolbox that makes testing easier

- Get it from Matlab file exchange
- Unzip somewhere
- Add somewhere/matlab_xunit/xunit to Matlab path (use pathtool)
runtests: runs all functions in the current directory whose name starts or ends with test (or Test)

```matlab
>> runtests
Test suite: /homes/numerik/poloni/somewhere
01-Feb-2012 10:36:55

Starting test run with 2 test cases.
..
PASSED in 0.002 seconds.
>>
```
**xUnit contents**

`runtests`: runs all functions in the current directory whose name starts or ends with `test` (or `Test`)

```
>> runtests
Test suite: /homes/numerik/poloni/somewhere
01-Feb-2012 10:36:55

Starting test run with 2 test cases.
.. 
PASSED in 0.002 seconds.
>>
```
Matlab provides assert:

\[
\text{assert(norm(A*x-b)<1e-10);} \quad \% \text{throws error if argument is false}
\]

xUnit provides something similar:

- `assertTrue`, `assertFalse` (kinda useless, only syntactic sugar)
- `assertEqual` (more useful — Matlab’s vector comparison is a pain)

(not trivial to replace `assertEqual` with a simple MATLAB one-liner
I’ll get you a beer if you get it right at the first attempt)
xUnit: writing a test

```matlab
function testGaussianElimination3 %don’t forget this line
n=4;
A=eye(n); b=[1:n]’; %very simple test case
x=GaussianElimination(A,b);
assertEqual(A*x,b);
```
Comparing with tolerance

Of course, in practice you'll want a tolerance. xUnit provides:

- `assertVectorsAlmostEqual`: check with a normwise relative tolerance of $1 \times 10^{-8}$ (configurable).
  Does what you want in most cases

- `assertElementsAlmostEqual`: similar check, but element-by-element.

```matlab
a=[1e12+1; 1];
b=[1e12+2; 2];
assertVectorsAlmostEqual(a,b); %pass
assertElementsAlmostEqual(a,b); %fail
```
Randomized tests

Randomized tests are very effective in practice

```matlab
function testGaussianEliminationRandomized

n=7;
for iteration=1:100
    A=randn(n);
    b=randn(n,1);
    x=GaussianElimination(A,b);
    assertVectorsAlmostEqual(x,A);
end
```

Tests with 100 random matrices. If some pivoting case gives error, much likely to catch it.
(Not so) randomized tests

Much easier to debug if we reset the random generator beforehand:

```matlab
function testGaussianEliminationRandomized

reset(RandStream.getDefaultStream);

n=7;

% [snip]
```
Testing for error messages

(Maybe not so important for “paper-quality” code)

```matlab
function testGaussianEliminationException

function f
    GaussianElimination(eye(2),1);
end

assertExceptionThrown(@f,'MATLAB:badsubscript');
end
```
Refactoring

```matlab
function x=GaussianElimination(A,b)
% [snip comment]

n=length(A);
x=zeros(n,1);

for k=1:n-1
    [s,t]=max(abs(A(k,k:n)));
t=t+k-1;
A([t k],:)=A([k t],:);
b([t k])=b([k t]);
b(k+1:end)=b(k+1:end)-A(k+1:end,k)/A(k,k)*b(k);
A(k+1:end,k:end)=A(k+1:end,k:end)-A(k+1:end,k)/A(k,k)*A(k,k:end);
end
solveUpperTriangularSystem(A,b);
```
function x=GaussianElimination(A,b)
% [snip comment]

n=length(A);
x=zeros(n,1);

for k=1:n-1
    [s,t]=max(abs(A(k,k:n)));
t=t+k-1;
A([t k],:)=A([k t],:);
b([t k])=b([k t]);
b(k+1:end)=b(k+1:end)-A(k+1:end,k)/A(k,k)*b(k);
A(k+1:end,k:end)=A(k+1:end,k:end)-A(k+1:end,k)/A(k,k)*A(k,k:end);
end

solveUpperTriangularSystem(A,b);

move this to another procedure
Refactoring

- The back-substitution can now be tested separately
- Less code to test/debug = easier
- More readable

When should I make it a separate function?
- When it’s logically something else...
- If you need to copy and paste code, refactor!
- If your code does not fit in one screen, refactor!
- Often, moving the code in a well-named subfunction is clearer than commenting it

- Should I worry about performance loss? No, 99.99% of the time
Often, well-named variables > comments

```matlab
function x=GaussianElimination(A,b)

n=length(A);
x=zeros(n,1);

for k=1:n-1
    [unused,pivotRow]=max(abs(A(k,k:n))); % converts relative index in k:n to absolute
    pivotRow=pivotRow+k-1;
    A([pivotRow k],:)=A([k pivotRow],:);
    b([pivotRow k])=b([k pivotRow]);
    b(k+1:end)=b(k+1:end)-A(k+1:end,k)/A(k,k)*b(k);
    A(k+1:end,k:end)=A(k+1:end,k:end)-A(k+1:end,k)/A(k,k)*A(k,k:end);
end

solveUpperTriangularSystem(A,b);
```

FP (tools seminar) | Matlab unit tests
Well-named variables

- Not saying that you should always name variables `rowIndexOfTheCurrentPositionInTheSystemMatrix` instead of `i`
- Always think about future you
- `tab-autocompletion` is in most editors, including Matlab’s.
- `(time spent reading code/debugging) \gg (time spent writing code)`
Make errors easier to spot

Some tricks...

- Use warning/error checking when future you will benefit

```matlab
if abs(A(k,k)/A(1,1))<1e-10
    warning 'Matrix may be ill-conditioned'
end
```

```matlab
n=size(b,1);
assertEqual([n,n],size(A),'Wrong matrix dimensions');
```

- `x=nan(n,1)` a better initialization than `x=zeros(n,1)`
Getting “present you” into the picture

- Do you have other good coding practices to suggest?
- Questions on xUnit or unit testing in general?