Programming (Econometrics)

Lecture 1: Introduction

Tommi Tervonen

Econometric Institute, Erasmus University Rotterdam

Course learning objectives

After this course, you should be able to:

- Program econometrical models in Matlab
- Understand core concepts of imperative programming
- Explain what happens when your Matlab code is executed
- Understand what is an efficient algorithm
- Code efficient algorithms in imperative programming languages

Course organization

- 7 lectures
  - Theoretical contents
  - Provide background for the exercises
- 6 exercise sessions
  - 6 exercises done in pairs - only 3 to submit
  - Come to exercises to ask questions and get help with your code
  - No mandatory attendance

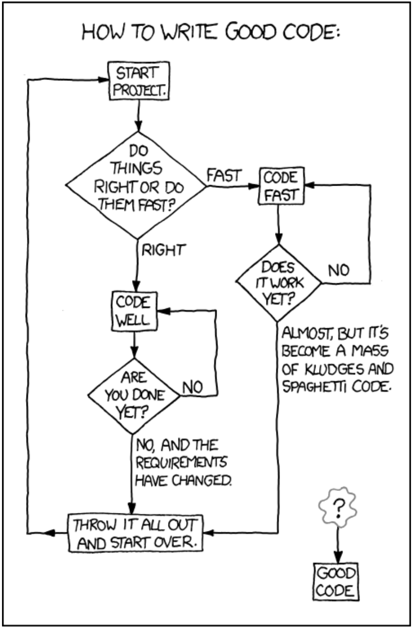
Study load

- 4 ECTS $= 112$h
- 7 lectures $= 14$h
- 6 exercise sessions $= 12$h
- Exam $= 3$h
- $\Rightarrow$ Independent programming 83h $\approx$ 14h/w

Grading

- Exercises: 50% (16.6% each of the 3 that you choose to submit)
  - Done in pairs (can also be done individually)
  - Exercises will be published in BB after Monday's lecture
  - Strict deadline on the following week's Sunday @ 23.59 (last exercise has an earlier deadline)
  - Submission via BB: *only* the source file(s) in the root of a zip. Include a comment in the beginning with your name(s) and student number(s)
- Written exam: 50%
  - Open questions

Making the exercises


HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
DOES IT WORK YET?
NO
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
RIGHT
CODE WELL
ARE YOU DONE YET?
NO
NO, AND THE REQUIREMENTS HAVE CHANGED.
THROW IT ALL OUT AND START OVER.
?
GOOD CODE

```
boolean done=false;
boolean understood=false;
while(!understood) {
    understood = readLN()
     && readExercise();
}
while(!done) {
  done = code();
  if (!done) {
    getHelp();
  }
}
```

Help! I can't code!

1. Read exercise & LN
2. Go to exercise sessions and get help
3. Code @ home
4. Get frustrated
5. Go to exercise sessions and get help
6. Code @ home
7. Get frustrated
8. Get help from BlackBoard forums
9. Code @ home
10. Get frustrated
11. Send Tommi email with topic [FEB22012X] Help!

Plagiarism

- Do not submit anything you haven't written yourself
- Do not submit anything that is not your idea
- We will not give you answers in the tutorials, but merely help you to find the answer
- "But I could've solved this problem myself, it was just faster to google the solution"

Course staff

Tommi Tervonen	Lectures + exercises	H11-26
Carlijn Liqui Lung	Exercises on the fourth week	-

- Also: you! Participate in course discussion forums in BB to get and provide help with the exercises

Required knowledge

Introduction to Programming:

- Variables and methods
- Program flow
- Decisions and branching
- Control structures
- Bitwise operators
- Arithmetic operators
- Scoping

Course contents

L1 Introduction
- Practicalities
- Programming paradigms
- Scripting languages
- Introduction to types

L2 Computing
- Numerical representation
- Introduction to complexity theory
- Insertion sort

L3 Memory organization
- Matrix representation
- Matrix multiplication

Course contents

- L4 Program correctness
  - Side effects
  - Pre- and post-conditions
  - Loop invariants
- L5 Linear data structures
  - Arrays, stacks and queues
  - Linked lists
- L6 Nonlinear data structures
  - Trees
  - Heap
  - Heapsort
- L7 Sorting & searching
  - Mergesort
  - Quick sort
  - Binary search

Literature

- LN-TT-22012-3, available @ `http://smaa.fi/tommi/courses/prog2/` and in print version from the student association, loosely based on a very selected set of material from:
    - Knuth: The Art of Computer Programming (vols 1-3)
    - Cormen, Leicerson, Rivest: Introduction to Algorithms
    - Goulb, van Loan: Matrix Computations
    - Wikipedia
- Matlab book can be useful to own
- All course material is posted in `http://smaa.fi/tommi/courses/prog2/`, and links to exercises also in BB

Software

- The exercise sessions will be guided with Matlab
- You can do most of the exercises with R, Python, or even Octave (though visualization in Octave sucks)
- Other courses require “fluency” in Matlab

Q?

“The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague.”

E.W. Dijkstra

Programming paradigms

- Programming paradigms refer to the philosophy behind designing programming languages
- When you know to program with 1 language of a paradigm, others of the same paradigm are easy to learn (mostly just syntax)

Programming paradigms

1. Procedural / imperative paradigm (C, Pascal, Matlab, R, Fortran, Algol, Python)
2. Object-oriented paradigm (Java, Smalltalk, C++ partially)
3. Declarative paradigm, including
   - Functional programming (ML, Lisp, Haskell, Erlang, Scala, Scheme)
   - Logic programming (Prolog)

OO vs Procedural

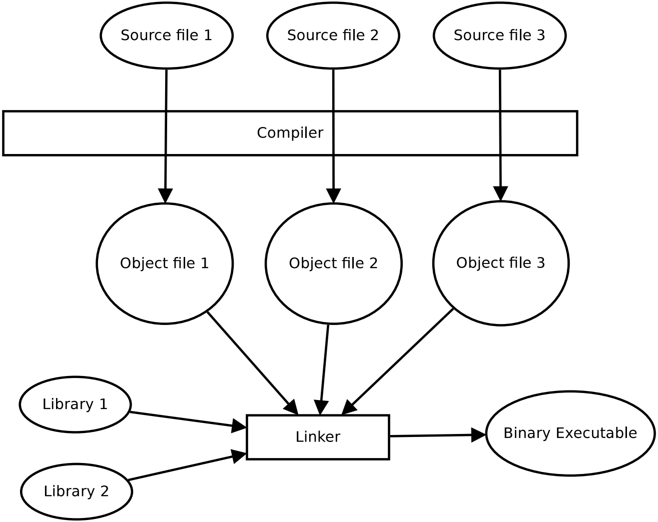
Object-oriented	Procedural
Design classes that communicate	Design global methods
Classes with fields	Data structures
Suitable for large programs	For “small” programs
Access control in language	Programmer has full access

- Both are part of imperative paradigm: control flow consists of *statements* that change the state of the program
- x = 2;
- Imperative paradigm makes program correctness hard to prove, as $x = 2 \neq x \leftarrow 2$

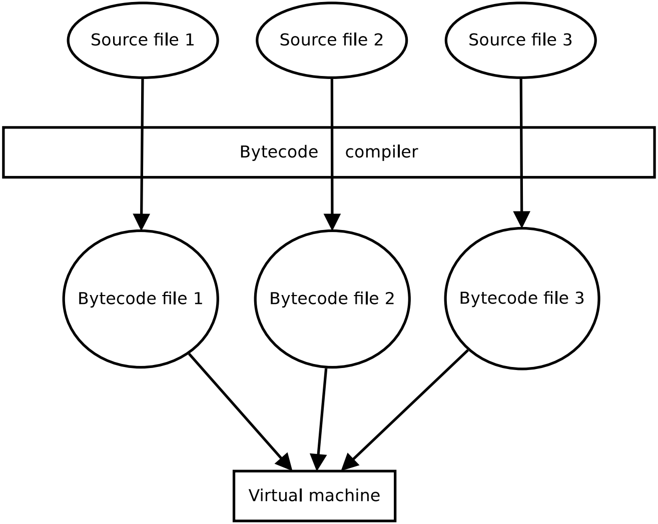
Compilation of languages

- Before source code can be executed, it needs to be *compiled* into an executable format
- The compilation can be made
  1. Completely in advance to a binary executable (fast)
  2. Partially in advance to bytecode to be executed in a virtual machine (Java, quite fast and portable)
  3. Run-time (slow but allows easy “modify & execute” cycles)

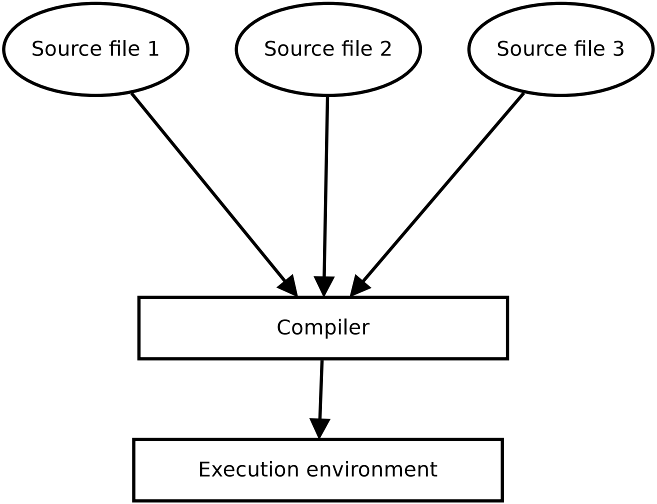
Fully compiled languages (e.g. C)



Bytecode compiled languages (e.g. Java)



Runtime compiled languages (e.g. Matlab)



Scripting languages

- In scripting languages the instructions are compiled run-time into execution statements
- Slow, as less optimization can be made
- In languages of statistical / scientific computation, you have to understand what happens “under the hood” to make efficient *and* correct code

Introduction to types

- Typing systems form the core of programming languages - they allow construction of abstractions
- Differences in electric currency $\rightarrow$ bits $\rightarrow$ numbers $\rightarrow$ characters $\rightarrow$ objects

Strong and weak typing

Strong typing: each variable has a type associated with it

```
int x = 2; // ok
x = 3; // ok
x = ``s''; // error
```

Weak typing : a single variable can be assigned varying types of values

```
y = 3; % ok - no type declaration required
y = 't'; % ok
```

Typing in Matlab

- Matlab is a weakly typed language, and the following are valid expressions:

```
x = 1;
y = '1';
z = x + y;
```

- Now z = ?

Next

- Get your copy of LN from student association
- Check the first exercise in the course page
- Make sure you understand the exercise
- Familiarize yourself with Matlab

... and get coding!