# Information on the final exam

*Computational Mathematics course, University of Pisa*

The exam for this course is composed of two parts:

- performing a take-home project, typically in a group of 2 persons, submitting a report of your work, and correcting it according to our comments until it is evaluated positively;
- taking an oral exam.

## 1 Group and project selection

We will give deadlines to form groups and subdivide projects and announce them on Moodle and Teams; the process will take place around November.

### 1.1 Pairing up

We expect the students to form groups of 2 persons each. Singleton groups may be allowed with convincing motivations—such as being away for Erasmus or the total number of students being odd—after discussion with us. A deadline for forming the groups will be announced, and we expect in return a *unique* text file with the description of all groups in the form

Group Number <tab> Name Surname <tab> Matricola <tab> e-mail address

### 1.2 ML and non-ML projects

Project descriptions consist of

- one or more *problems* to solve;
- one or more *algorithms* to be used to solve them.

There will be two lists of projects:

- *ML projects*, on problems that overlap with the topics treated in the Machine Learning course (neural networks, support vector machines);
- *non-ML projects*, which are self-contained on the topics of this course.

You may choose among either list. ML projects require some background on neural networks, but apart from that there is no difference in the difficulty of the two groups of assignments, nor in the grading.

You may be able to re-use parts of the code for this project for your ML exam (further instructions by prof. Micheli). Note, though, that the two exams (and the two reports) are quite different. In this course we focus on the mathematical side of algorithms (quality of the obtained solutions, in terms of "gap" or "residual", and efficiency in obtaining it), while ML typically focuses on the usefulness of the algorithm in learning, which is a different thing. So, while performing a ML project can have a few advantages if you follow both courses, it is not substantially different from a non-ML one.

There will be also some "wildcard" projects that you can personalize (which is very welcome); these cannot be used to copy other projects, even with minor modifications.

### 1.3 Subdividing projects

We will provide a list of available final projects. The students will have time (at least one week) to agree collectively on how to subdivide the projects so that *every group performs a different project*. We expect an *unanimous* agreement to be reached on the matching; should you fail to reach that, with even a single student reporting dissent from the decision, we will step in and give a randomized assignment (which would most likely leave more people unsatisfied). We will only guarantee to keep your preference between ML and non-ML projects (which means, two partial randomized assignments).

In case there is a shortage of projects in one of the lists, we will add more.

## 2 Structure of your report

### 2.1 Avoid plagiarism

Blatant copying from existing material (provided by the instructors or found on the Internet) will be mercilessly crushed upon. We expect all code and text (except for definitions and theorems) to be your own work; exceptions should be explicitly referenced. Ask the instructors if you have doubts on originality and plagiarism issues.

### 2.2 Setting the stage

The first section of your report should contain a description of the problem and the methods that you plan to use. This is just a brief recall, to introduce notation and specify which variants of the methods you plan to use. Your target audience is someone who is already familiar with the content of the course. There is no need to repeat a large part of the theory: we are sure that you know how to do that, given enough time, books, slides, and internet bandwidth.

In case adapting the algorithm to your problem requires some further mathematical derivation (example: developing an exact line search for your problem, when possible, or adapting an algorithm to deal more efficiently with the special structure of your problem), you are supposed to discuss it here with all the necessary mathematical details.

Discuss the reasons behind the choices you make (the ones you can make, that is, since several of them will be dictated by the statement of the project and cannot be questioned).

We suggest to send us a version of this section by e-mail as soon as it is done, so that we can catch misunderstandings as soon as possible and minimize the amount of work wasted. Note that we do not want to see code at this point: that would be premature to produce (for you) and unnecessarily complicated to read (for us).

### 2.3 What to expect from the algorithm(s)

Next, we expect a brief recall of the algorithmic properties that you expect to see in the experiments. Is the algorithm (if it is iterative) guaranteed to converge? Is it going to be stable and return a good approximation of the solution (if it is direct)? What is its complexity? Are there any relevant convergence results? Are the hypotheses of these convergence results (convexity, compactness, differentiability, etc.) satisfied by your problem? If not, what are the "closest" possible results you have available, and why exactly are they *not* applicable? Do you expect this to be relevant in practice? Each time you use some specific result (say, a convergence theorem), please be sure to report in detail what the assumptions of the result are, what consequences exactly you can derive from them, and the source where you have taken it (down to the number of theorem/page). Also, please be sure to discuss in detail why the assumptions are satisfied in your case, or which assumptions are not satisfied or you cannot prove they are.

Again, we suggest to send us a version of this section by e-mail as soon as it is done. No code yet.

### 2.4 Write your code

Coding the algorithms is a major part of the project. Languages that are often convenient for numerical computation are (depending on the task) Matlab, Python, and C/C++, but you can use any reasonable programming language.

You are expected to implement the algorithm yourself; it should *not* be a single line of library call. However, you can use the numerical libraries of your language of choice for some of the individual steps: for instance, you can use `numpy.linalg.norm` to evaluate the error, or Matlab's `A \ b` to solve a linear system that appears as a sub-step (unless, of course, writing a linear solver to compute that solution is a main task in your project).

You can (and should) also use numerical libraries to compare their results to yours: for instance, you can check if your algorithm is faster or slower than Matlab's `quadprog`, if it produces (up to a tolerance) the same objective value, or how the residual of your solution compares with that produced by `A \ b`.

When in doubt if you should use a library, feel free to ask us.

Your goal for this project is implementing and testing numerical algorithms: software engineering practices such as a full test suite, or pages of production-quality documentation, are *not* required. That said, we appreciate well-written and well-documented code (who doesn't?). You are free to use tools such as `git` to ease your work, if you are familiar with them (but giving us a pointer to the `git` repository is not the expected way to communicate with us).

### 2.5 Choose and describe the experimental set-up

Next, we expect a brief description of the data you will test your algorithms on. For "ML projects" this will typically be provided by the ML course, but still a modicum of description is required. For "no-ML projects", it will typically have to be either generated randomly, or picked up from the Internet, or a combination of both. This is not always trivial: the random generation process can be tweaked to obtain "interesting" properties of the data (what kind of solution can be expected, how well or ill-conditioned the problem is, . . . ). These aspects should be described in the report.

You are supposed to test the algorithm on a realistic range of examples, in terms of size, structure, and sparsity: it is typically *not* OK if your largest example is $10 \times 10$. Get a sense of how algorithms scale, and what is the maximum size of problems that you can solve reasonably quickly on an ordinary machine.

Numerical experiments have two purposes:

- Confirm that the methods work as expected: how close do they get to the true solution of the problem? How can you check it? Is there a "residual", or "gap" value that you can check? Do they converge with the rate (linearly, sublinearly, superlinarly, . . . ) that the theory predicts? Does the error decrease monotonically, if it is expected to do so?

- Evaluate trade-offs and compare various algorithms: which algorithm is faster? If algorithm A takes fewer iterations than algorithm B, but its iterations are more expensive, which one is the winner? How does this depend on the characteristics of the problem to be solved (size, density, . . . )?

Comparison with off-the-shelf algorithms is also welcome (and often useful to check correctness) to assess whether your approach could ever be competitive under the right conditions (and what these are).

Setting thresholds and algorithmic parameters is a key and nontrivial aspect. This is one of the "dark secrets" of numerical algorithms: basically any algorithm you can write has parameters, and it will misbehave if you don't set them properly. They can have a huge impact on performance. Thus, a minimal testing activity about the effect of these parameters is almost surely needed. A full-scale test à-la hypermetameters optimization in ML is also possible, and welcome, but this is anyway different from the one in ML, even for "ML projects". The properties of interest are fundamentally different: in ML one looks for learning (accuracy, recall, . . . ), while in this course one is looking at "how close the solution I got is to what I would have liked to get, and how costly was it to get there". Hence, reporting here the same hypermetameters optimization tables done for ML does not cut it. Not that these *cannot* be reported, as they still give some potentially interesting information, but it is not the information we are crucially interested in.

When designing your experiments, and later your graphs and/or tables, you should have these goals in mind. To quote a famous mathematician, "the purpose of computing is insight, not numbers."

### 2.6 Report results of your experiments

A few plots and/or tables are required to display your results. Have a purpose in mind: as for the choice of experiments, the plots should display the important features of the algorithm: convergence speed, residual, computational time, . . .

Plots should be readable. If 90% of your plot are barely-visible horizontal or vertical lines, look for a better way to display information. *Logarithmic scales* are usually a good idea to display quantities (such as residuals or errors) that vary by orders of magnitude. In particular, a sequence with exact linear convergence ($\text{error}_{k+1} = C \cdot \text{error}_k$) becomes a straight line in logarithmic plots; hence they display well the convergence speed of algorithms.

For "ML projects", the quantities you want to plot may be quite different from these you would in a ML course. While the "learning rate" (here "convergence rate") is the same, the out-of-sample performance is much less relevant for us.

## 3 Submitting your project

Submit your report to us by e-mail, as a `pdf` file. We suggest using LaTeX, but this is not mandatory.

You can send us partial versions of your projects at any time ("release early, release often"); ideally, after each major milestone (theoretical analysis, data set definition, before starting the experiments, after the parameters tuning phase, . . . ), so that we can check that you are on the right track and suggest improvements. Most of the times we *will* ask for a few changes; it is unusual that the first submitted version of a (part of a) project gets our "stamp of approval" immediately. Getting feedback early may allow you to avoid building on erroneous assumptions and having to redo large parts of the project. However, if you really want to take the chance to let us have the project all in one blow at the end, you can; only, do not give it for granted that we will agree that that is the final version, and that you will immediately get access to the oral exam.

Project reports can be submitted at any time until the start of the next version of this course in the next academic year. At that time, projects that have already started but are stalled for some reason must be completed before the projects of the new year are assigned (roughly at the middle of the course).

The project will contribute substantially to your final mark. Once the project is assigned it cannot be changed, except for extraordinary reasons to be discussed with us, nor is it possible to require a second project to improve your marks up until the end of the process (the re-starting of it in the next Academic Year).

## 4 The oral exam

Once we are satisfied with your project, we give our stamp of approval and we can proceed with the oral exam. Send us a final e-mail containing:

- the final version of your report;

- all the source code of your project, including any auxiliary file, batch or script required to run it, with a quick README describing how to run it;

- results of the experiments in spreadsheet/databases/text files.

We will agree with you a date and time for the exam. There are no "appelli" for this course: you may ignore the dates on `https://esami.unipi.it`. In general we will be available to check your (partial) project and make the oral exam at any time of the year, but keep in mind that we have other commitments, too (say, in August our reaction times could be significantly slower). Typically all students of a group are expected to take the oral exam at the same time, but motivated exceptions are possible.

There is no need to register for the oral exam on `https://esami.unipi.it`, but please *do submit your course evaluation* there; it is very important feedback for us.

The oral exam will start with a discussion of your project, and possible improvements. Usually, other parts of the course's syllabus emerge during this discussion. You are expected to be familiar with the main ideas and topics of the course. The main purpose of this course is insight, not proofs, and this is what you will be tested on.

It is of course possible to refuse the vote that you are offered. However, an accepted project cannot be changed or improved. Hence, the only way to improve your vote is to re-take the oral exam. If this happens, the exam will no longer focus on the project, and therefore it will necessarily have to go more in detail on the mathematical theory of the course. So make sure that you return well prepared on the theoretical part. There is no guarantee that a new oral exam will improve the vote that you are finally offered.

Feel free to ask for clarifications at any time. Address *all* e-mail to both of us, even if it looks like it concerns only one of the two parts of the course.

Good luck! We hope that you enjoyed the course, and that it will be useful in your future.

Federico and Antono