Information on the Didactic Project

Optimization for Data Science (784AA), Università di Pisa

1 Group and project selection

The lecturer will provide deadlines to form groups and define a groups → projects matching, announcing them during the lectures and on the Moodle page (https://elearning.di.unipi.it/course/view.php?id=497); the process will take place around November.

1.1 Pairing up

The interested students are expected 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 the lecturer. Three-students groups can also be allowed, upon request, for performing "significantly heavier" projects than the standard ones. These will have to be autonomously proposed by the students exploiting the "wildcard project" mechanism described in details later on: for instance, a wildcard project can be constructed that group models and/or algorithms from more than one "standard project". Alternatively, `SMS++` projects can easily be constructed to be "significantly heavier" than standard ones, but this requires joining the ranks of `C++` kamikaze.

A deadline for forming the groups will be announced, by which the students will have to return by e-mail a *single* file with a tab-separated list of *all* groups in a format such as

```
Group Number | Name Surname | Matricola | e-mail address
```

for each student (ordered by group number).

1.2 Projects

Project descriptions consist of

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

Some of the problems will be more directly related to Data Science applications, some other will not; however, there will be no difference in the difficulty of the two kinds of assignments.

A list of possible projects will be provided in due time by the lecturer. These will be divided in two lists: "normal" projects (the vast majority) in which the groups have complete freedom about the programming language and libreriers used for the implementation, and a smaller number "SMS++ projects" whereby the students are rather required to use the Structured Modelling System framework (https://gitlab.com/smspp/smspp-project), which implies the choice of `C++` as the programming language. These projects may require either the development of a `Block` (model of a specific optimization problem), or of a `Solver` (specialised solution algorithm) for an existing `Block`, or both. While the overall difficulty of the two kinds projects is analogous, `SMS++` projects have more stringent requirements on both the tools used and the required quality of the produced software, and are therefore by default considered more favorably in term of final grading, plus having a special perk (as detailed below).

There will be also be the possibility to propose "wildcard" projects where you either mix-and match between problems and algorithms proposed in the list in a way that is not done in the existing projects, or propose your own problems / algorithms. This is very welcome, and an accepted way to justify the proposal of forming three-students groups, provided the content of the project warrants it (that is, significantly more problems and/or algorithms than in standard projects, or significantly more complex ones); this is particularly easy for `SMS++` projects. However, "wildcard" projects are not intended, and must not be used, to clone existing projects from the list, even with minor modifications. Anyway, each "wildcard" project will have to be discussed with, and approved by, the lecturer, possibly subject to amendments.

1.3 Subdividing projects

After the groups are formed and the list of available projects has been provided, 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*. An *unanimous* agreement must be reached on the matching; should this fail to materialize, with even a single student

reporting dissent from the decision, the lecturer will step in and give a randomized assignment, which would most likely leave everyone unsatisfied.

In case there is a shortage of projects in any of the two parts of the list, more be will added.

After the initial groups → projects (semi-)assignment has been decided, all remaining projects are up for grabs on a first-come-first-served basis. Students not having previously formed groups can do so, and groups (newly formed or not) can request any of the currently unassigned projects, at any point in time during the year. Just be mindful of the rules governing the switch between the project track and the written+oral track as described in the Moodle page of the course.

2 Workflow of the exam

Once you have received your project, passing the exam requires the following three steps, in temporal sequence:

1. submit a report on the project as a single `pdf` file and have it accepted;
2. submit the code and all the accompanying material (auxiliary files, batches or scripts required to run it, README, data files, logs / csv files / spreadsheets with the detailed results of the experiments, ...) in a single `.zip` or `.tgz` file and have it accepted;
3. pass the oral exam.

All comunications need happen via e-mail.

Submitting the report (and having it accepted) is typically the most time-consuming part of the process. It is *strongly suggested* that you send your reports in partal instalments. Each report should contain a number of separate parts (see the next section for details), and a good practice is to send the report each time you complete one of them. In this way it can be checked that you are on the right track, conceptual mistakes can be corrected and improvements suggested. It is expected that changes *will* be asked for, at least most of the time; it is unusual that the first submitted version of a (part of a) project gets the "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. In particular, it is strongly suggested to complete the "theoretical" part of the report before you invest any serious amount of time into implementations, so as to avoid implementing and testing the wrong approach. However, if you really want to take the chance to subit the report all in one blow when you think it is complete, you can; only, do not give it for granted that it will be accepyed as the final version, and that you will immediately get access to the oral exam.

Project reports (complete or partial) can be submitted at any time until the start of the next version of this course in the next Academic Year (AY). At that time, projects that have already started but are stalled for some reason must be completed before the projects of the new AY are assigned (roughly at the middle of the course). Upon request, a further extension may be granted to groups that have already performed a significant amount of work, as demonstrated by their partial reports, with the idea that they finish "soon" (say, before the end of the calendar year).

Once the complete report is approved, and *only at that moment*, you will submit the corresponding final distribution of your code and of the supporting material. Typically this will be approved right away, since all serious issues should hopefully have been ironed out in the previous phase, but improvements (say, in comments or in the accompanying material) may still be asked if necessary. Once both the report and the code are approved the oral exam can be held and the grading finalised.

The project will contribute substantially to your final mark. Once the project is assigned it cannot be changed, except with the explicit approval by the lecturer for extraordinary reasons, 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 AY). It is always possible at any time to renounce to the project and move to the written + oral exam track, but the project work will then not be taken into account in the final grading, which may therefore be even worse (downto insufficient). Once a project is renounced, it cannot be revived and the written + oral exam track becomes mandatory. Conversely, students having originally chosen the written + oral exam track—that is, not having joined any group—can (form a group and) ask for a project at any point in the AY and choose one among those that have not been taken up yet. Again, this can happen only once: should the project then be renounced, the written + oral exam track will become mandatory again (this time, for good) until the subsequent AY.

The only possible exception to the above rules (save "force majeure" cases to be individually discussed with the lecturer) is the possibility to renounce a "standard" project and pick up instead a `SMS++` one. This will in principle be allowed, but again only once: once the `SMS++` project is either completed or renounced, no other project (of either type) can be required for the current AY. Also, all the work performed for the "standard" project will in principle be lost. The reverse process of switching from a `SMS++` project to a "standard" one is in principle *not* allowed, save for extraordinary reasons under the explicit approval of the lecturer (and even in this fringe case, no more than once).

3 Structure of your report

3.1 Avoid plagiarism

Blatant copying from existing material (provided by the instructors or found on the Internet) will be mercilessly crushed upon. All code and text (except for definitions and theorems) must to be your own work; if you extensively draw from some specific source, this must be explicitly referenced and acknowledged. Ask the lecturer if you have doubts on originality and plagiarism issues.

The lecturer would prefer your work *not* to be make publicly available, e.g. on repo sites like GitHub or GitLab, but since it will be your code this cannot be strictly enforced. Consequently, any attempt at too closely drawing from some existing repository will also be mercilessly crushed upon.

Conversely, the lecturer by default assumes to have the right of using snippets of your code for didactic purposes (although this is not supposed to happen frequently). Should you disagree with this, the objection will have to be explicitly made, and will then be of course complied with.

3.2 Setting the stage

The first section of your report should contain a description of the problem(s) and the algorithm(s) that you plan to use. This is just a brief recall, to introduce notation and specify which variant(s) of the algorithm(s) 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, as it is expected that you know how to do that, given enough time, books, slides, and internet bandwidth.

In case adapting the algorithm(s) to your problem(s) requires some further mathematical derivation (such as developing an exact line search for your function, 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 should not be questioned unless you think you have found a serious conceptual flaw in it).

3.3 What to expect from the algorithm(s)

Next, a brief recall of the algorithmic properties that you expect to see in the experiments is required. Are there any relevant convergence results for your algorithm(s)? 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? What about complexity? 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). Discuss in detail why the assumptions are satisfied in your case and why, or which assumptions are not satisfied or you cannot prove they are.

3.4 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/Octave`, `Python`, `Julia` and `C/C++`, but you can use any other reasonable programming language or system (say, `R`).

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 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), or even using existing solvers to tackle specific optimiation problems that appear as sub-problems in your algorithms, but only as long as this is explicitly permitted by the project statement.

You can (and should) also use existing libraries to compare their results to yours: for instance, you can check if your algorithm is faster or slower than Matlab's `quadprog` (or whatever other applicable off-the-shelf software) and if it produces (up to a tolerance) the same objective value.

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

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, well-written and well-documented code is appreciated. 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 provide the code (especially as it would be appreciated if the repo was never made public).

The exception to the above rules is when performing `SMS++` projects, as in this case the choice of the programming language is fixed (`C++`) and a reasonable test suite, production-quality documentation and the use of `git` *are* required. The specific requirements will be discussed on a project-by-project basis.

3.5 Choose and describe the experimental set-up

Next, a brief description of the data you will test your algorithms on is required.

The data will typically have to be either picked up from the Internet (repositories of AI/ML datasets, repositories of optimization instances, ...), or generated randomly, or a combination of both (such as in taking a dataset having most of the required data and randomly generating the few missing pieces). 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 badly a given approach can be expected to perform, ...). 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/or sparsity: it is typically *not* OK if your largest example is 10×10 (whatever "10" measures). 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. Using HPC systems or machines otherwise equipped with special features (such as high-end GPU) should not be required, save possibly for `SMS++` projects (in which case assistance to get access to the required hardware will be provided).

Numerical experiments have two purposes:

- Confirm that the algorithms work as expected: how close do they get to the true solution of the problem? How can you check it? Is there a "gap" value that you can check? Do they converge with the rate (linearly, sublinearly, superlinearly, ...) 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 software 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 that can have a huge impact on performance, and it will misbehave if you do not set them properly. Thus, a minimal testing activity about the effect of these parameters is almost surely needed. A full-scale test a-la hypermetameters optimization in ML is also possible, and welcome, but typically not necessary. Those already familiar with AI/ML techniques should also consider that, even in projects where the underlying model in a ML one (say, a NN, a SVR, ...) the properties of interest are fundamentally different from the ones one would be concerned with in a ML setting. Indeed, ML is interested in 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". Besides, it is common occurrence in ML to do hypermetameters tuning at the same time on *model parameters* (say, the weight of the regularization term or the topology of the NN) and *algorithmic parameters* (say, the fixed stepsize); this is unacceptable here, where the model parameters need be fixed (in any reasonable way) so that all the algorithms solve exactly the same problem. Hence, any hypermetameters optimization would need to be properly reasoned.

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."

3.6 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(s): convergence speed, gap/accuracy, 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 gaps or gradient norms that vary by orders of magnitude. In particular, they display well convergence speed, since a sequence with linear convergence ($v_{k+1} \leq r \cdot v_k$) should become a(n approximately) straight line.

Always keep in mind what is the information that is important for you to show and do your best to show it effectively and efficiently: three pages-long tables of 7pt figures or 10 pages filled of very many small plots are typically neither efficient nor effective at conveying the information to your readers.

4 The oral exam and grading registration

Once your project report and the final distribution of the code are accepted, a date and time for the exam wll be agreed upon. There are no "appelli" for this course if you are enrolled in the "didactic project + oral" track, hence

you may safely ignore the dates on `https://esami.unipi.it`, and there is no need to register for the oral exam when a custom date is agreed upon. However, *please do submit your course evaluation* on `https://esami.unipi.it` before the oral exam, as it is no longer possible to do it after the grade is formally registered. Since you are not required to register this cannot be imposed, but any feedback about what does work and what does not in the course is a service you do on your fellow students of the following AYs.

In general, submitting (partial) reports and asking for the oral exam is allowed at any time of the year, but keep in mind that the lecturer has other commitments, too; say, in August reaction times could be significantly slower. Typically all students of a group are expected to take the oral exam at the same time, but exceptions are possible upon well-motivated request.

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 all the main ideas and topics of the syllabus, but the main purpose of this course is insight, not proofs, and this is what you will be tested on. The idea is that the relevant theoretical parts will have been discussed in details in the report, possibly incorporating feedback, and therefore there should be little point in delving upon them again, unless the discussion strictly warrents this. Of course, this does not apply to oral exams in the standard "written + oral" track.

If you agree with the proposed final grading, it will be immediately registered in the system. It is of course always possible to refuse the vote that you are offered. However, an accepted project cannot be changed or improved. Hence, the only way to significantly improve your grade in case of a low-quality project is to re-take the oral exam. If this happens, the exam can obviously no longer focus on the project, and therefore it will necessarily have to go more in detail on the mathematical theory of the course, as is the case in the "written + oral" track. Hence, should you want to try to improve your grading this way you are better 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. There also remains the option to completely renounce the project and move to the "written + oral" track, where each written + oral attempt is independent and therefore it is in principle possible to improve (or deteriorate) at will the final grading. As previously discussed, students originally in the "written + oral" track can freely move to the "didactic project + oral" track at any point in time of the AY, but only once; should they eventually renounce the project they are given, they would be back to the "written + oral" track for all the remainder of the AY.

5 Final remarks

Interacting with the lecturer during the development of your project is a useful mean of discussing and clarifying the contents of the course, especially of those more strictly related with the project itself; you are strongly encouraged to exploit this option. However, this is not a good substitute of asking questions during lectures and office hours (ricevimento): you are strongly encouraged to exploit these options, too. The lecturer will always be available and happy to discuss the course topics at any time, insomuch as permitted by the other teaching, research, and organisational tasks.

Best of luck for your work, and I hope you will enjoy the course and the project.

Antonio