TopEdu

PROJECT 10 INSTRUCTIONS

The race between solvers

Make a simulation twice as sharp and it can take many times longer. Race the methods and find out which wins, and when.

START HERE

Set up in ten minutes

This project comes with one short Python file, `10_solver_race.py`. It solves the project's equation with simple methods you can read line by line, checks itself against an answer that is known exactly, and ends with a list of first experiments. The research is what you do after that.

1. **Install Python 3** from python.org (on a Mac it may already be there: type `python3 --version` in Terminal).
2. **Download the starter** from the project's page, or all ten as a zip from topedu.club/topedu/research/starter-code/.
3. **Install the two libraries** in Terminal (Mac) or PowerShell (Windows):

   ```
   python3 -m pip install numpy matplotlib
   ```

   On Windows, type `py` where this guide says `python3`.
4. **Run it:**

   ```
   python3 10_solver_race.py --show
   ```

   The CHECK line must say PASS. `--show` opens the plots; without it they are saved in the `output` folder.

No computer of your own? Google Colab works in a browser for free: open a new notebook, upload the file with the folder icon on the left, and run `!python 10_solver_race.py` in a cell.

The rules that make it research

- **Predict before you run.** Write your guess, and why, in your log first. A wrong prediction you can explain is worth more than a right one you cannot.
- **Keep a log.** Date, what you changed, what you expected, what happened. Your paper is written from it.
- **Check the grid.** Run your key result again with twice the points. If the answer moves, report by how much.
- **Never edit the CHECK to make it pass.** A FAIL after your change is information: undo, confirm PASS, then change one thing at a time.
- **Own every line.** Your mentor explains; you write and run. Name everyone and everything that helped, including this starter.

The full guide, with all ten projects and five numerical ideas the starters use, is at topedu.club/topedu/research/starter-code/.

PROJECT 10 · NUMERICAL ANALYSIS AND COMPUTING

The race between solvers

Make a simulation twice as sharp and it can take many times longer. Race the methods and find out which wins, and when.

FILE	MATH	YOU NEED
`10_solver_race.py`	Algebra II, logarithms; some programming helps	A laptop; some programming helps

The question

Doubling a simulation's resolution makes a basic solver far slower. How do accuracy, iteration count and run time scale, and at what size do higher-order methods, multigrid and GPUs actually pay off?

Objectives

Weeks 1 to 3	**Learn to trust the model.** Run 10_solver_race.py until its CHECK line says PASS, and explain every line of it to your mentor. Then reproduce a known answer: The solution has a known exact form, so you can measure the real error. Our test run gives rates of 2.00, 3.00 and 4.95 for orders 2, 4 and 6. Part of your project is explaining why the labelled order and the measured rate differ. Checkpoint: you reproduce a known answer and report how close you got.
Weeks 4 and 5	**Ask your own question.** Narrow the question above to one sentence that could turn out to be wrong, and write your prediction before you run anything. The first experiments below are a place to start. Checkpoint: a one-page plan with your question and prediction.
Weeks 6 to 9	**Run it, then measure the world.** Measure error and iterations as the grid grows for orders 2, 4 and 6, then race conjugate gradients against multigrid, and CPU against GPU, with the sbp-sat-suite tools. Then step away from the screen: time everything on real hardware and fit the power laws yourself. In our test, CG needed 127 to 416 iterations from 65 to 513 points while multigrid stayed near 15, and multigrid won on time only past about 257 points. Your machine's break-even point is your own result. Checkpoint: one chart comparing your model with reality.
Weeks 10 to 12	**Explain the gap, and share it.** Work out why your model and your measurements disagree, test at least one explanation, and write it up yourself. Say which numbers came from the starter and which from the lab. Keep the honest limit in view: the online lab covers conjugate gradients only. Multigrid and GPU runs use the sbp-sat-suite tools with a mentor. Checkpoint: a final paper, your code and a ten-minute talk.

The model in the starter

The test problem: -laplacian(u) = f on the unit square, u = 0 on the edge, with f chosen so the exact answer is u = g(x) g(y), g(s) = s (1 - s) e^s. Knowing the answer lets you measure the real error of every method. (The lab uses u = sin(pi x) sin(pi y). Try it here, in problem(): CG then finishes in one iteration.

Why?)

This file races three ways of solving the same grid equations (plain second-order finite differences): Jacobi: each point becomes the average the equation wants. Simple, and very slow on big grids. Conjugate gradients: the standard method for this kind of system; its iterations grow with the grid. Multigrid (V-cycle): fixes smooth errors on coarser grids; its iterations should stay flat.

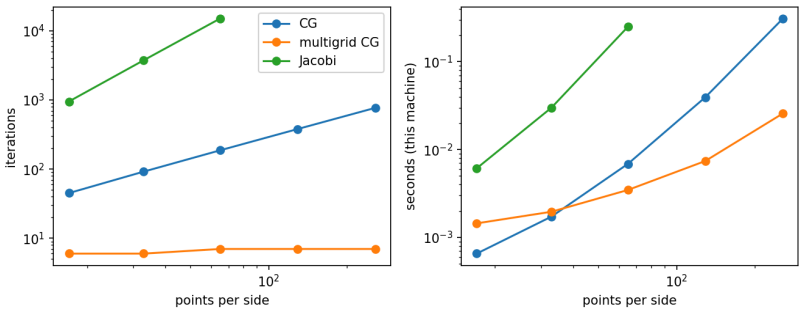
What the starter does

- Stage 1: run it. It checks that the error falls like h^2 (rate near 2). The CHECK line should say PASS.
- Stage 2: time everything as the grid grows, fit the power laws, and find your own machine's break-even point. The online lab uses higher-order SBP-SAT methods; sbp-sat-suite has a mentor-run multigrid and GPU version.

What you should see

Run `python3 10_solver_race.py`. This is the output from the run that built this guide. The numbers that depend on your computer's speed may differ; the CHECK line must say PASS.

```
points   method          iterations   seconds   max error
    17   CG                    45     0.001   3.62e-04
    17   multigrid CG           6     0.001   3.62e-04
    17   Jacobi               950     0.006   3.62e-04
    33   CG                    92     0.002   9.07e-05
    33   multigrid CG           6     0.002   9.07e-05
    33   Jacobi              3750     0.030   9.07e-05
    65   CG                   187     0.007   2.27e-05
    65   multigrid CG           7     0.003   2.27e-05
    65   Jacobi             15050     0.250   2.27e-05
   129   CG                   379     0.039   5.68e-06
   129   multigrid CG           7     0.007   5.68e-06
   257   CG                   768     0.310   1.42e-06
   257   multigrid CG           7     0.026   1.42e-06
measured convergence rate: 2.00 (second-order differences should give 2)
CHECK convergence rate near 2 -> PASS (limit 2 +/- 0.1)
```



Your first experiments

From the YOUR TURN list at the bottom of the file. Write your prediction in your log *before* you run each one.

1. Predict first: if CG's iterations grow like n^p, what is p? Fit it from the table (polyfit on the logs). Do the same for time. Why does time grow faster than iterations?
2. Add 513 points. From what size does multigrid win on time here? Compiled code (sbp-sat-suite) shifts that point, because each method's cost per iteration changes. Your machine's break-even point is your own result.
3. In the online lab, run orders 2, 4 and 6 at 17, 33, 65 and 129 points and measure each order's rate. Order 4 measures about 3 in our tests. Why might the boundary be the reason?
4. With a mentor: race CPU against GPU in sbp-sat-suite. Report where your numbers differ from its README.

The data

Settings in the online lab

Setting	Range	Starts at
Grid points per side	9 to 129	33
Accuracy order	2 to 6	4

Checked on the research solver

Solver (Poisson, 2D; sbp-sat-suite for multigrid and GPU)	order 2: rate 2.00, iterations 29 > 52 > 91 > 154; order 4: rate 3.00, iterations 31 > 61 > 113 > 200; order 6: rate 4.95, iterations 36 > 77 > 145 > 265
Known answer	error falls at least as fast as the order's boundary rate (order 2 gives about 2)
Error	0.03% (limit 10.00%): passed

The real world

Time everything on real hardware and fit the power laws yourself. In our test, CG needed 127 to 416 iterations from 65 to 513 points while multigrid stayed near 15, and multigrid won on time only past about 257 points. Your machine's break-even point is your own result.

The honest limit

The online lab covers conjugate gradients only. Multigrid and GPU runs use the sbp-sat-suite tools with a mentor.

How the full project checks itself: The solution has a known exact form, so you can measure the real error. Our test run gives rates of 2.00, 3.00 and 4.95 for orders 2, 4 and 6. Part of your project is explaining why the labelled order and the measured rate differ. The online lab runs the same project on the research solver (Poisson, 2D; sbp-sat-suite for multigrid and GPU).

REFERENCE

When something goes wrong

You see	Try this
`No module named numpy`	Run the install line again with the same Python you use to run the file: `python3 -m pip install -r requirements.txt`
Numbers become nan or huge	A time step is past its speed limit, usually after you made the grid finer. Each solver works out its step from the grid; check you did not fix it by hand.
`did not settle`	An iterative solver needs more sweeps for a finer grid. Raise its limit, or keep the grid smaller while you explore.
A CHECK line says FAIL after your change	Good: your change moved the model away from the case with a known answer. Undo it, confirm PASS, then change one thing at a time.
No window opens	Add `--show`, or open the PNG files in the `output` folder.
A recording gives strange notes	Record in a quiet room, save as .wav (not .m4a), trim silence at the start, and set the note you expect in the file.

How the starters relate to the online lab

The online lab, for members, runs each project on the research solver (AutoSBPSAT): high-order SBP-SAT methods, the full 3D elasticity for project 9, and frames of a time series. The starters use plain second-order methods instead, so every line is short enough to read. Both are checked against the same kind of exact answers. Use the starter to understand and to explore quickly; use the lab for results you will quote, and say in your paper which one produced each number.

How these starters were checked

Every number in "What you should see" came from running the starter while this guide was built, on Python 3.9.23, numpy 1.24.4, matplotlib 3.5.2. `check_all.py` also feeds each optional data reader (the soil-station CSV, the ruler CSV and two .wav recordings) a made-up file with a known answer, and confirms the answer comes back. They have not yet been used by a student. The twelve-week plan and the time estimates in it are a design, to be revised after the first group: tell your mentor what was unclear.