

Numerical Methods in Physics

First Laboratory Session

Orientation and Computing Setup

Hamzeh Khanpour

hamzeh.khanpour@fis.agh.edu.pl

<https://view.fis.agh.edu.pl/staff/hamzeh.khanpour/>

Faculty of Physics and Applied Computer Science
AGH University of Krakow

5 October 2026

Laboratory instructor and contact details



Laboratory instructor

Hamzeh Khanpour

High Energy Particle Physicist, AGH University of Krakow || CERN-TH

How to contact me

E-mail: hamzeh.khanpour@fis.agh.edu.pl; hamzeh.khanpour@agh.edu.pl

CERN e-mail: hamzeh.khanpour@cern.ch

Where to find me

Office & Room: D10 & 218

You are welcome to contact me or stop by for discussion at any time.

Website and course materials

<https://view.fis.agh.edu.pl/staff/hamzeh.khanpour/>

<https://home.agh.edu.pl/~hamzeh.khanpour/>

<https://hamzeh-khanpour.web.cern.ch/>

Materials for Laboratory Session 1

The **nmpLab01.zip** package contains the materials used during the first laboratory session, including the introductory slides, example source code, setup and configuration instructions, compilation and execution examples, and the initial hands-on exercises.

The same package is available from any of the following mirror locations:

<https://view.fis.agh.edu.pl/staff/hamzeh.khanpour/nmpLab01.zip>

<https://home.agh.edu.pl/~hamzeh.khanpour/nmpLab01.zip>

<https://hamzeh-khanpour.web.cern.ch/nmpLab01.zip>

References named in the official course sheet

Obligatory

W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery,
Numerical Recipes: The Art of Scientific Computing, third edition, Cambridge University Press, 2007.

Algorithms, explanations and implementation context. Check code licensing separately.

Optional: numerical linear algebra

G. H. Golub and C. F. Van Loan,
Matrix Computations.

Johns Hopkins University Press.

Fourth edition (2013) is a verified edition choice;
the supplied sheet does not specify an edition.

Optional: engineering methods

J. D. Hoffman,
Numerical Methods for Engineers and Scientists.

Marcel Dekker, as named in the sheet.

The sheet does not specify an edition; a second edition (2001) is documented.

Course-webpage reading: foundations and C++



A. Quarteroni, R. Sacco and F. Saleri

Numerical Mathematics, second edition, Springer, 2007 (copyright year).
Broad mathematical foundations and worked numerical examples.

W. Gautschi

Numerical Analysis, second edition, Birkhäuser, 2012 (copyright year).
Approximation, quadrature and analysis; useful for alternative explanations.

J. Pitt-Francis and J. Whiteley

Guide to Scientific Computing in C++, Springer London, 2012.
A practical bridge from C++ programming to scientific calculations.

These are linked reading resources, not newly imposed purchases. Access may require an institutional subscription.

Course-webpage reading: accuracy and large systems

R. M. Corless and N. Fillion

A Graduate Introduction to Numerical Methods: From the Viewpoint of Backward Error Analysis, Springer, 2013.

Advanced perspective: what nearby problem does a computation solve?

Y. Saad: sparse linear systems

Iterative Methods for Sparse Linear Systems, second edition, SIAM, 2003.

Sparse storage, iterative methods and preconditioning.

Y. Saad: eigenvalue problems

Numerical Methods for Large Eigenvalue Problems, second edition, SIAM, 2011.

Selective reading for large and sparse eigenproblems.

Use advanced books selectively alongside the project instructions; they are not all first-week reading.

Additional reading beyond the course webpage



L. N. Trefethen and D. Bau III

Numerical Linear Algebra, SIAM, 1997.

Compact explanations of factorizations, conditioning and eigenvalue algorithms.

N. J. Higham

Accuracy and Stability of Numerical Algorithms, second edition, SIAM, 2002.

An advanced reference for diagnosing roundoff, conditioning and stability.

M. T. Heath

Scientific Computing: An Introductory Survey, revised second edition, SIAM, 2018.

A broad computational-science perspective with algorithm selection and interpretation.

Supplementary recommendations only. Start with the current project instructions and the lecturer's notes.

Today: orientation, then a working environment



Understand the course

- How projects, activity scores and reports fit together.
- Where to find instructions and how to submit work.
- What the lecture and project roadmaps contain.

Establish one working route

- Access the approved Linux environment.
- Compile a tiny program and check numerical libraries.
- Identify any account or installation problem.

This is not the first numerical project

The setup checks are **ungraded**. No substantial project is solved or assessed today. Published Project 1 keeps its number; the instructor will confirm the meeting schedule.

Course at a glance



Official profile

Programme	Computer Physics
Level	First-cycle engineer
Semester	3
Language	English
Status	Obligatory / core
Course code	JCMPS.Ii4.14448.25

Workload and preparation

6 ECTS

30 lecture hours + 45 laboratory hours.

Basic vectors/matrices, differential and integral calculus, and C programming.

Why numerical methods in physics?

Questions we want to answer

- How do coupled equations determine a potential?
- Which eigenstates does a model predict?
- What signal is hidden in measured data?
- How accurately can we fit or integrate a function?

Abilities we develop

W1/W2: understand methods and physical applications.

U1: write suitable programs or use numerical packages appropriately.

K1: justify choices and work responsibly.

The scientific workflow

Question → method → implementation → verification → interpretation.

C: direct control and numerical interfaces

Strengths

Compiled code, explicit arrays and a small language core make low-level data layout visible. C interfaces connect many numerical packages.

Be careful about

The programmer must manage bounds, memory and error handling carefully. A fast executable can still produce a wrong numerical answer.

Scientific example

FFTW is a C library for discrete Fourier transforms. Its numerical kernels support signal-processing and scientific-computing workflows.

Relevance here

CBLAS and LAPACK expose numerical routines through C interfaces. This is a useful bridge from our own loops to established libraries.

C++: numerical programs that can grow

Strengths

Compiled execution together with functions, classes, templates and standard containers supports reusable scientific programs.

Be careful about

More language features mean more design and debugging choices. Check array dimensions, ownership and library return codes.

Scientific example

ROOT, **developed at CERN**, provides scientific data storage, processing, visualization and analysis using C++ and interfaces to other languages.

Relevance here

Our setup demonstrations use g++. C++17 is a proposed demonstration convention, not a separately published course requirement.

Python: exploration, arrays and visualization

Strengths

Concise high-level code is convenient for data inspection, plots and independent checks. NumPy supplies numerical arrays and operations.

Be careful about

Python-level loops and compiled array kernels behave differently. Package versions, array shapes and data types still matter.

Scientific example

NumPy supports linear algebra, transforms, statistics and random simulation; many array operations execute in compiled code.

Relevance here

Proposed use: read C/C++ output and plot or cross-check it. Permission must be confirmed; do not replace an assigned implementation with a library call.

Fortran: a scientific-computing foundation



Strengths

A compiled language with array-oriented numerical facilities and a long-established scientific software ecosystem.

Be careful about

Interfacing requires attention to array order, numeric types and calling conventions. Existing code must still be tested and documented.

Scientific example

LAPACK is written in Fortran 90 and supplies factorizations, linear solves, eigenvalue problems and singular-value computations.

Relevance here

We use its capabilities through LAPACK. Learning to call the C interface does not require installing or learning a second language today.

Julia: numerical experimentation and compiled methods

Strengths

A dynamic language designed for scientific and numerical work, with arrays, multiple dispatch and compiled execution of specialized methods.

Be careful about

Compilation and warm-up affect timings; performance depends on how code is written. Package compatibility and reproducibility need attention.

Scientific example

Julia provides numerical arrays and linear algebra. JuMP is a Julia-based modelling language for mathematical optimization, with examples ranging from fitting to power-system models.

Relevance here

An additional environment to know about, not a required installation. Compare tools by the problem and workflow, not a universal speed ranking.

Your weekly responsibilities



Before and during the meeting

- Read the assigned project instructions and review its method.
- Write your own program and work through the stated tasks.
- Keep the required numerical outputs, not just a screenshot.

At the end and afterwards

- Upload source code and required data as the activity ZIP.
- Finish incomplete numerical work independently.
- Prepare and submit the separate PDF report on time.

C/C++

Published programming preference: C/C++; Ask before adopting a different implementation route. Keep each project and its outputs identifiable.

Project workflow: in the laboratory and after the laboratory

During the laboratory class

- Work on the assigned numerical project during the scheduled laboratory.
- Write and test your own C/C++ program and carry out the required numerical tasks.
- Generate and save the required numerical results and data.
- Your **activity is assessed during the laboratory** according to the extent to which the project tasks are completed.
- At the end of the laboratory, submit the required **source code and numerical data as a ZIP archive** through UPeL.

After the laboratory / at home

- Finish any incomplete numerical work independently, if needed.
- Analyse the results and prepare the required figures and tables.
- Write the separate **PDF report**:
 - numerical method,
 - task and implementation,
 - results and discussion,
 - conclusions.
- Submit the report through UPeL **within 7 days after the laboratory**.

Assessment: two components, one laboratory grade

During the project

Activity: 0–100 points.

Assessment reflects the extent to which the stated project tasks are realized.

The first meeting is excluded from activity assessment in the supplied sheet.

After the project

Report: 0–100 points.

The report communicates the method, results and your interpretation. A project is ordinarily completed through its report.

Course credit and details to confirm

The supplied sheet requires at least half the maximum available points and makes the course grade equal to the laboratory grade. It describes a simple mean of partial evaluations. **The exact averaging and grade bands will be confirmed**; no new weighting is introduced here.

The report: method, evidence and interpretation

Required structure

Method: about one page.

Task: about half a page.

Results: compact figures/tables with short, 2–3 sentence comments.

Conclusions: about half a page of your own observations.

Format and timing

PDF only. Submit to UPeL within **7 days** after the relevant laboratory finishes.

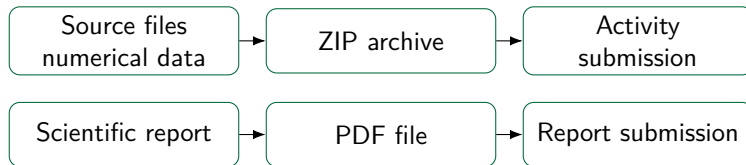
The published late rule deducts **20 points for each subsequent week of delay**.

Good practice

Good practice: label axes and units; state parameters and tolerances; explain errors and convergence. The section lengths are approximate, not a new strict page limit.

UPeL: two separate submission tracks

UPeL: University e-Learning Platform. Sign in through AGH single sign-on (SSO), using AGH email credentials.



Use the correct course and group

UPeL portal | Course link published on the NMP website.

The instructor must confirm the current group and assignment names. Check the recorded submission state; notify the instructor promptly if upload fails.

Prepare a genuine ZIP archive



Official contents

Place the source files and required numerical data in a separate project folder. Archive that folder as **ZIP**; other archive formats are not accepted.

Run from the parent of the project directory.

```
zip -r Surname_Name_Project01.zip Surname_Name_Project01/
```

```
unzip -l Surname_Name_Project01.zip
```

```
unzip -t Surname_Name_Project01.zip
```

Create, list, then test the archive. The PDF report still belongs in its own submission track; renaming a RAR file does not make it a ZIP.

Recommended organization

```
Surname_Name_Project01/  
  src/  data/  
  plots/  README.txt
```

This hierarchy is a recommendation, not an extra official rule.

Absences do not remove the learning task

Justified absence

The webpage waives the activity ZIP and its assessment. Complete the project independently and submit the report within one week after returning to the next class.

The supplied sheet limits the simple activity exemption to **two cases**; further absences require make-up work.

Unexcused absence

Activity receives **0 points**. The project must still be completed at home and the report submitted.

Do not treat an absence as an automatic cancellation of the project or its report.

Confirm early

Confirm early the documentation procedure, any make-up dates and the corresponding UPeL deadlines with the instructor.

Make-up work, retakes and a report exception

Different purposes

Absence make-up: the supplied sheet refers to two additional meetings when justified absences exceed two.

Improving a negative final result: at most two additional projects may replace the two weakest original project evaluations; the final result is recalculated.

Conditional restriction

More than two unexcused absences together with no laboratory credit *may* restrict improvement opportunities. The sheet provides an appeal to the coordinator or dean.

Published P15 exception

The Monte Carlo project explicitly requires **no report**; its partial grade is based on activity. This does not waive other projects' reports.

Individual ownership and reproducible results

Published requirement

Each project is individual work. You are responsible for the program and the numerical results that you submit.

Preserve the source, inputs, parameters and essential outputs.

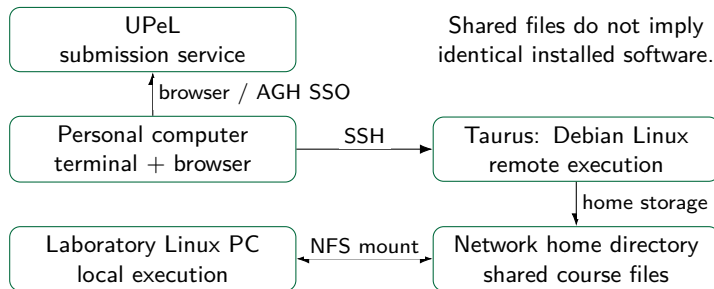
Recommended habits

Keep a short compile/run README. Label figures, document tolerances and explain how you checked the answer. You should be able to explain your code and conclusions.

AI and collaboration boundaries

The course-specific policy for AI tools, code-sharing and permitted assistance must be confirmed with the coordinator. Do not assume unrestricted permission. Never present copied code or fabricated results as your own independent work.

T1 | Where are the files, and where does code run?



Storage and execution are different

SSH (Secure Shell) provides a remote terminal; **NFS** (Network File System) exposes shared files. WFILS recommends workstation applications first. Check libraries separately on the classroom PC and Taurus.

T2 | First login and account access



From a terminal on your current computer

```
ssh YOUR_LOGIN@taurus.fis.agh.edu.pl  
# YOUR password.  
exit
```

What this does

WFiS requires a first remote Taurus login to initialize the home directory. Afterwards, the same Taurus account is used on the Linux workstation.

Password entry may show no characters.

Check before continuing

Confirm the hostname and account. Ask IT about unexpected host-key warnings; do not disable checking.

UPeL uses AGH SSO, not an assumed interchangeable Taurus login.

T3 | Your course directory and terminal workflow



Commands to recognize

`pwd`: print the current directory.
`ls`: list files.
`cd`: change directory.
`mkdir -p`: create a directory path.

Keep setup separate

```
pwd
mkdir -p ~/nmp/setup
cd ~/nmp/setup
ls
```

Read the path, not just the command

`~` means your home directory. Linux filenames are case-sensitive. A command after SSH runs remotely. Keep orientation checks in `setup/` and numbered work in `Project01/`, `Project02/`, etc.; do not overwrite existing work.

T4 | Source code to executable



A complete tiny C++ program

```
#include <iostream>
int main() {
    std::cout << "C++ setup OK\n";
    return 0;
}
```

```
g++ -std=c++17 -Wall -Wextra -O2 hello.cpp -o hello
./hello
```

Expected

Expected: C++ setup OK. C++17 is the setup convention; this is an ungraded check, not a numerical project.

The toolchain

GCC: GNU Compiler Collection.

gcc: C-oriented driver.

g++: C++ compilation/linking.

-o names the executable; warnings help reveal mistakes.

T5 | BLAS and its C interface



What is it?

BLAS: Basic Linear Algebra Subprograms. It provides vector and matrix kernels.

CBLAS is a C interface, not a separate numerical method. The installed implementation determines the actual kernels.

```
g++ -std=c++17 cblas_test.cpp -o cblas_test -lblas -lm  
./cblas_test
```

The complete source includes `cblas.h` and checks the result. P01 later compares your own matrix operations with library calls.

One small test

```
const double x[] = {1,2,3};  
const double y[] = {4,5,6};  
const double d =  
    cblas_ddot(3,x,1,y,1);
```

Expected: $1(4) + 2(5) + 3(6) = 32$.

T6 | LAPACK, LAPACKe and the dependency chain



Library roles

LAPACK: Linear Algebra PACKage.

LAPACKe: its C interface.

BLAS: underlying vector/matrix kernels.

Examples: LU, eigenvalues, SVD and tridiagonal solves.

Calling a routine

A **header** declares the interface. Match row/column ordering and the leading dimension. Check the returned info status before trusting the result.

```
g++ -std=c++17 lapacke_test.cpp -o lapacke_test \  
-llapacke -llapack -lblas -lm \  
./lapacke_test
```

Correct linker explanation

GNU tools process inputs in command-line order. For ordinary archive linking, put a dependency **after** what uses it: program → LAPACKe → LAPACK → BLAS. The old webpage's right-to-left explanation is incorrect.

T7 | LAPACK: test first, then choose a route



System-wide library works

Use the standard command from T6. No private copy is needed.

Missing on an AGH machine? Ask the instructor or administrator; do not run a system installation yourself.

```
g++ -std=c++17 -I"$HOME/bin/lapacke/include" \  
-L"$HOME/bin/lapacke" lapacke_test.cpp -o lapacke_test \  
-llapacke -llapack -lblas -lm \  
./lapacke_test
```

-I: headers; -L: library directory; -l: library name. The fallback archive was not retrievable during production. **Do not assume it is needed or compatible.**

Course-provided fallback

The webpage offers lapacke.zip. Its layout and binary compatibility must be checked before use. The documented location is \$HOME/bin/lapacke.

T8 | FFTW and the libraries we actually need



Fourier-transform software

FFTW: Fastest Fourier Transform in the West. P11 uses it for the discrete Fourier transform.

Our tiny check transforms an impulse, inverts it and divides by the transform length.

```
g++ -std=c++17 fftw_test.cpp -o fftw_test -lfftw3 -lm  
./fftw_test
```

Expected: the four-point impulse is recovered after division by 4. Library availability still needs checking on the actual classroom PC and Taurus.

Required versus additional

Core for published projects: BLAS/CBLAS, LAPACK/LAPACKGE, FFTW3.

Linked references, not blanket requirements: GSL, Intel MKL, MUMPS, ARPACK-ng.

T9 | One working route on your own computer

Choose one route

- Use approved remote SSH access, when available.
- Use your own compatible Linux installation.
- On Windows, optionally use WSL2 (Windows Subsystem for Linux).
- On macOS, use native compatible libraries or remote access.

Personal Linux installation only

```
sudo apt update
sudo apt install \
    build-essential zip unzip \
    libblas-dev liblapack-dev \
    liblapacke-dev libfftw3-dev
```

Permission matter

these installation commands are for an authorized personal Debian/Ubuntu-like machine, not Taurus or managed lab PCs. Windows-side WSL commands and Linux commands are different; see the setup guide.

T10 | Diagnose the failure, then ask the right question



What failed?

Header not found: compilation cannot see a declaration.

Undefined reference / cannot find -l: link flags or library installation.

Loader/architecture error: binary compatibility or runtime search path.

What to report

Which machine? Which directory?
Exact command? First error message?

A nonzero numerical-library info value needs its own interpretation; it is not necessarily a missing package.

Record status honestly

Account / compiler / BLAS / LAPACK / FFTW / UPeL: mark each **working**, **pending** or **needs assistance**. A local test does not verify Taurus, and shared files do not verify the classroom compiler.

Central question

What is the course trying to teach?

In the notes

The lecturer's introduction connects numerical methods with programming practice and independent laboratory work. It outlines the main mathematical families and useful reading.

Why it matters

Read with the current course sheet. The linked introductory PDF gives an older 15+30-hour allocation. The current public programme gives 30+45 hours and 6 ECTS.

Laboratory connection: Orientation connects all projects. A lecture-note unit is not automatically one scheduled lecture or laboratory meeting.

Central question

Why can two reasonable computations disagree?

In the notes

Finite floating-point representation; round-off and cancellation; truncation in approximations; numerical differentiation; accuracy and stability. The notes motivate these through physical and mathematical calculations.

Laboratory connection: A recurring check across P01–P15. For example, compare accuracy as a grid or tolerance is refined, rather than trusting extra printed digits.

Why it matters

Distinguish the causes. Conditioning concerns the problem; stability concerns the algorithm. Increasing precision reduces roundoff but does not automatically repair an unstable method.

Anchor idea

$$\epsilon_{\text{abs}} = |x_{\text{num}} - x_{\text{ref}}|$$

Central question

How do we solve many coupled linear equations?

In the notes

Gaussian elimination, triangular solves, lower-upper (LU) factorization, symmetric factorizations and tridiagonal systems. QR factorization also supports least-squares problems.

Laboratory connection: P02 uses pivoted LU; P09 uses a tridiagonal solve. QR is in the syllabus/notes but has no separately titled project on the current webpage.

Why it matters

Discretized field equations and fitted coefficients often become matrix problems. Reusing a factorization can help when several right-hand sides share the same matrix.

Anchor idea

$$Ax = b$$

L04 | Diagonalization of dense matrices



Central question

Which special modes are encoded in a matrix?

In the notes

Eigenvalues/eigenvectors, transformations to simpler matrix forms, tridiagonal reduction and QR-type eigenvalue methods; generalized problems and singular value decomposition also appear.

Laboratory connection: P03: a tridiagonal physical model. P06: singular value decomposition (SVD). A residual checks whether a computed eigenpair satisfies its defining equation.

Why it matters

Matrix eigenproblems represent physical modes and discretized quantum problems. Matrix symmetry and structure determine which routines are appropriate.

Anchor idea

$$Av = \lambda v$$

Central question

Can a sparse problem be solved without dense factorization?

In the notes

Sparse storage, including coordinate and compressed sparse row (CSR) formats; Jacobi, Gauss–Seidel, successive over-relaxation (SOR), steepest descent and conjugate gradients (CG).

Laboratory connection: P04 implements SOR and follows residual histories. No universal solver ranking is assumed; convergence depends on the matrix, parameters and stopping test.

Why it matters

Large discretizations may contain mostly zero entries. Store and multiply the meaningful entries, then track convergence instead of forming a dense inverse.

Anchor idea

$$r_k = b - Ax_k$$

L06 | Iterative diagonalization of sparse matrices



Central question

How can we find selected eigenpairs rather than the whole spectrum?

In the notes

Power iteration with orthogonalization; shift-and-inverse ideas; Lanczos and Arnoldi methods for reducing large eigenproblems.

Laboratory connection: P05 implements the power method and orthogonalization. Check residuals and orthogonality, not only whether the reported eigenvalue has stopped changing.

Why it matters

A physical calculation may need only a few modes or eigenstates. Iterative methods can target part of a spectrum, but spectral separation and starting vectors matter.

Anchor idea

$$v_{k+1} = Av_k / \|Av_k\|_2$$

Central question

How do we locate a root when a direct formula is unavailable?

In the notes

Bisection, false position, secant and Newton–Raphson methods; convergence concepts; systems of nonlinear equations and multidimensional Newton iteration.

Why it matters

Bracketing and local iteration offer different robustness/cost trade-offs. Newton's quadratic convergence is local and requires suitable conditions near a simple root.

Anchor idea

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Laboratory connection: P07 compares root-finding methods, including a multiple root. Report a function residual and a step or bracket test; keep an iteration limit.

Central question

What should a curve do between tabulated values?

In the notes

Polynomial interpolation, the Lagrange formula, interpolation error, Chebyshev nodes and piecewise cubic splines with boundary conditions.

Laboratory connection: P08 compares node choices and the Runge effect. P09 constructs cubic splines. Keep interpolation inside its supported interval distinct from extrapolation.

Why it matters

Interpolation estimates intermediate values from tabulated data. Matching every node is the defining constraint; it does not guarantee small error everywhere between nodes.

Anchor idea

$$p(x_i) = y_i$$

Central question

How do we choose a useful approximation without matching every point?

In the notes

Basis functions, norms, weighted mean-square approximation, monomial and trigonometric bases, and Pad'e rational approximants.

Laboratory connection: P06 uses SVD for a least-squares fit. P10 builds rational approximants. Compare residuals and inspect the approximation over its intended domain.

Why it matters

Fitting experimental data and simplifying known functions are related but different tasks. A chosen norm and basis define what good agreement means.

Anchor idea

$$\min_a \sum_j w_j \left[y_j - \sum_i a_i \phi_i(x_j) \right]^2$$

Central question

Which parameters make an objective function small?

In the notes

Local/global minima, constraints, golden-section search, parabolic interpolation, steepest descent, conjugate-gradient and Newton ideas.

Laboratory connection: There is no separately titled optimization project in the current webpage inventory. Its laboratory treatment must be confirmed; this slide does not announce its removal from the syllabus.

Why it matters

Objectives can represent energy, fit quality or an engineering design criterion. A converged search does not automatically establish a global minimum.

Anchor idea

$$x^* \in \arg \min_x f(x)$$

L11 | Fast Fourier Transform



Central question

How is a sampled signal represented by frequencies?

In the notes

Fourier series and the discrete Fourier transform (DFT); Nyquist frequency and aliasing; frequency indexing; radix-2 FFT, inverse and multidimensional transforms.

Laboratory connection: P11 filters a synthetic noisy signal using FFTW. Confirm the transform convention and verify a forward/inverse round trip before interpreting spectra.

Why it matters

The fast Fourier transform (FFT) is an efficient way to compute a DFT, not a different physical transform. Sampling and normalization conventions matter.

Anchor idea

$$c_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j e^{-2\pi i k j / N}$$

Central question

How do we turn function samples into a reliable integral?

In the notes

Closed Newton–Cotes rules; composite formulas; Richardson/Romberg extrapolation; Gaussian rules and their nodes/weights; multidimensional integration.

Why it matters

Charges, energies and normalization constants often involve integrals. Choose the rule with the integration domain, weights and smoothness in mind.

Anchor idea

$$\int_a^b f(x) dx \approx \sum_i w_i f(x_i)$$

Laboratory connection: P12 explores one-dimensional rules. P13 compares Gaussian quadratures in an electrostatic integral. Convergence against a reference is more informative than a single number.

L13 | Random number generators



Published status: IN PREPARATION

The course webpage lists this lecture unit, but its linked teaching material was not retrievable during production. This frame preserves the advertised topic without inventing its syllabus.

Introductory context only

Random-number generators produce samples used in stochastic calculations. A reproducible computation needs to identify its generator and setup.

Information still to be announced

The precise generator families, examples, notes and any required preparation will come from the lecturer. P14 is also currently listed as in preparation.

L14 | Monte Carlo integration



Published lecture notes: IN PREPARATION

The lecture topic is listed, but the linked notes were not retrievable. Do not infer the future lecture's detailed sequence from this overview.

Introductory context only

Monte Carlo methods estimate quantities from samples. Reporting sampling uncertainty is part of the result, not an optional decorative error bar.

A separate source is available

P15 already has a published project sheet on integration, variance and sampling. The project is available even though the lecture notes remain pending.

For suitable independent sampling and finite variance, standard error typically scales as $N^{-1/2}$. This statement is supported by the available project, not an unread lecture PDF.

What we compute

Perform repeated matrix–vector products, then compare handwritten matrix multiplication with a BLAS calculation.

Method and implementation

Write matrix/vector loops and compare matrix multiplication with CBLAS dgemm.

Software: BLAS / CBLAS

Learning goal: Arrays, memory layout and the distinction between an algorithm and an optimized numerical kernel.

Submission: Report follows the normal rule.

Outputs and verification

Save norm histories and timings. First compare numerical answers; only then compare speed.

What we compute

Determine a shaping polynomial from prescribed values and derivatives, then study what happens as interpolation nodes approach one another.

Method and implementation

Construct the constraint matrix and call LAPACKE_dgetrf / LAPACKE_dgetrs.

Software: LAPACKE + LAPACK + BLAS

Outputs and verification

Plot the polynomial and sensitivity diagnostics. Check the residual; a determinant from pivoted LU needs the permutation sign.

Learning goal: Relate a physically motivated shape constraint to a linear system and distinguish ill-conditioning from a library failure.

Submission: Report follows the normal rule.

P03 | Diagonalization of a tridiagonal matrix



What we compute

Discretize the one-dimensional quantum harmonic oscillator and compute its spectrum and wavefunctions in a finite box.

Method and implementation

Build the tridiagonal Hamiltonian and call LAPACKE_dstev; normalize with the grid spacing.

Software: LAPACKE + dependencies

Learning goal: See how a differential equation becomes an eigenvalue problem with measurable approximation errors.

Submission: Report follows the normal rule.

Outputs and verification

Compare spectra and wavefunctions with analytic results. Check eigenpair residuals and separate grid/box effects; align eigenvector signs.

What we compute

Solve a sparse system motivated by a discretized Poisson problem, using compressed sparse row storage.

Method and implementation

Implement CSR storage, sparse matrix-vector operations and SOR; vary the prescribed relaxation parameter.

Software: Standard C/C++; own routines

Learning goal: Connect sparse storage, convergence and the choice of relaxation parameter.

Submission: Report follows the normal rule.

Outputs and verification

Plot residual and solution-norm histories and the requested potential/source. Follow the assigned tolerance and iteration limit; optionally add a scaled residual.

P05 | Diagonalization using the power method



What we compute

Find eigenpairs of the specified symmetric matrix by repeated matrix–vector multiplication and orthogonalization.

Method and implementation

Write matrix/vector routines, normalize iterates and apply Gram–Schmidt orthogonalization.

Software: No external numerical library required

Learning goal: Learn an iterative algorithm directly, rather than treating a library call as the method itself.

Submission: Report follows the normal rule.

Outputs and verification

Show eigenpairs, convergence and reconstruction results. Check residuals and orthogonality; dominant eigenvalues are selected by magnitude.

What we compute

Fit samples of $x \cos(x)$ with a polynomial basis and compare fits as the amount of data and basis size change.

Method and implementation

Construct the polynomial design matrix and use `LAPACKE_dgesvd`; vary data and basis sizes.

Software: LAPACKE + dependencies

Learning goal: Understand overdetermined fitting and the information revealed by singular values.

Submission: Report follows the normal rule.

Outputs and verification

Show singular values, fitted curves and reconstruction information. Check fit residuals and inspect small singular values.

What we compute

Compare root-finding methods on a polynomial containing a simple root and a multiple root.

Method and implementation

Implement the three methods for the prescribed polynomial, including the modified Newton comparison.

Software: Standard C/C++; own routines

Learning goal: Distinguish convergence speed from robustness and recognize the effect of root multiplicity.

Submission: Report follows the normal rule.

Outputs and verification

Record iterations and step changes. Check function residuals; even-multiplicity roots do not provide the usual sign-change bracket.

What we compute

Interpolate $1/(1 + x^2)$ on $[-5,5]$ and compare equidistant nodes with Chebyshev nodes.

Method and implementation

Write a Lagrange interpolant and compare the specified equidistant/Chebyshev node sets.

Software: Standard C/C++; own routines

Learning goal: Understand the Runge effect and why more equidistant nodes need not improve a high-degree polynomial.

Submission: Report follows the normal rule.

Outputs and verification

Plot the function, interpolant and error. Check both input nodes and independent points; node agreement alone is insufficient.

What we compute

Construct a natural cubic spline for the test function and compare it with the known function and derivatives.

Method and implementation

Construct the second-derivative system and call LAPACKE_dgtsv with the specified natural boundaries.

Software: LAPACKE + dependencies

Learning goal: Combine local polynomial modelling with an efficient structured linear solve.

Submission: Report follows the normal rule.

Outputs and verification

Compare splines, function errors and second derivatives as node count changes. Check node values and continuity.

P10 | Padé approximation of a Gaussian



What we compute

Approximate $\exp(-x^2)$ using a ratio of polynomials for several numerator/denominator orders.

Method and implementation

Build the coefficient system and use LAPACKE_dsylv for the specified numerator/denominator orders.

Software: LAPACKE + dependencies

Learning goal: Recognize rational approximation as an alternative to polynomial interpolation or fitting.

Submission: Report follows the normal rule.

Outputs and verification

Plot the rational approximations and compare their errors and behaviour. Inspect denominators for zeros before dividing.

What we compute

Construct a periodic signal, contaminate it with noise, filter its Fourier components and reconstruct a cleaned signal.

Method and implementation

Use FFTW plans/execution, apply the specified frequency threshold, then invert and divide by N .

Software: FFTW3

Learning goal: Interpret sampled frequencies and connect a numerical transform with signal processing.

Submission: Report follows the normal rule.

Outputs and verification

Plot signal and spectral comparisons. First check a round trip; record the random setup for reproducible comparisons.

P12 | One-dimensional Newton–Cotes integration



What we compute

Integrate the specified oscillatory function using trapezoidal, Simpson and the hand-out's five-node Newton–Cotes rule, then apply extrapolation.

Method and implementation

Implement the assigned quadratures, then construct the Romberg-style refinement table.

Software: Standard C/C++; own routines

Learning goal: Compare quadrature order and extrapolation using evidence, not only a final decimal value.

Submission: Report follows the normal rule.

Outputs and verification

Show integral estimates and error-versus-node-count plots. Compare with the handout reference and discuss observed convergence.

P13 | Two-dimensional electrostatic integration



What we compute

Evaluate the interaction of two one-dimensional Gaussian charge distributions through a double integral.

Method and implementation

Compare Gauss–Legendre and Gauss–Hermite integration using the supplied node/weight helper.

Software: LAPACK helper; source retrieval pending

Learning goal: Match weights and integration domains to a physical integral, rather than using one quadrature indiscriminately.

Submission: Normal report rule; no explicit waiver found.

Outputs and verification

Show convergence and interaction versus separation; compare with the point-charge limit. The helper still needs retrieval and testing.

P14 | Random number generators



Published status: IN PREPARATION

The project appears on the course webpage, but its linked instructions were not retrievable. The title is retained; no generator, tasks, output format or grading rule is invented.

Before this project

The instructor will announce the finished handout, prerequisites, software and submission requirements. Read the published version before beginning implementation.

Connection to the course

Random sampling is used in the available Monte Carlo project P15. This explains the topic's relevance without predicting P14's final task list.

What we compute

Estimate the assigned integrals from random samples. Compare ordinary and anti-thetic sampling, then use an exponential distribution.

Method and software

Generate samples, form averages and estimate sampling variance in your own C/C++ program. Record the generator and seed.

For suitable sampling and finite variance, standard error scales as $N^{-1/2}$.

Outputs and verification

Plot integral estimates and uncertainties as sample size grows. Compare with the analytic references in the handout.

No report required: the published P15 grade is based on activity.

Learning goal: distinguish the estimate, sampling variance and uncertainty of the mean.

How lectures connect to projects

Concept family	Published projects
Numerical linear algebra	P01–P06: operations, LU, eigenproblems, SOR, SVD
Nonlinear equations	P07: root finding
Interpolation and approximation	P08–P10: Lagrange, splines, Padé
Fourier methods	P11: signal filtering
Quadrature and sampling	P12–P13 and P15; P14 remains pending

This is a concept map, not the weekly timetable

QR and optimization occur in the supplied syllabus, but have no separately titled current project. Their laboratory treatment and the orientation/project calendar must be confirmed by the coordinator.

Documentation, resources and help



Course and access

Numerical Methods course page
UPeL portal and official login guidance
Taurus and account instructions
Linux laboratory-PC instructions

Software and setup

BLAS documentation
LAPACKE interface documentation
FFTW manual
GNU compiler: linking options

Ask with useful information

For course or submission questions, contact the laboratory instructor. Include the machine, exact command, filename and first error. Consult `Student_Setup_Guide.md` for copyable commands and test files.

Before leaving: your readiness checkpoint

One route is usable

- My account and working directory are accessible.
- I can compile and run the tiny C++ test.
- I know the status of BLAS, LAPACK and FFTW.
- I know whether I am working locally or remotely.

The workflow is clear

- I can locate the correct UPeL group.
- I can create and inspect a ZIP archive.
- I know the report PDF track and deadline.
- I know whom to contact about a problem.

A pending check needs an action, not a guessed pass

Mark each item **working** / **pending** / **needs assistance**. Record the blocker and help route. No student needs to install all five languages or resolve an administrator-only failure alone.

Next: the first numerical project

Published Project 1

Simple numerical operations with vectors and matrices.

We move from setup checks to the actual matrix/vector tasks in the published handout.

Prepare beforehand

- Review arrays, loops, functions and matrix dimensions.
- Read P01 and its BLAS/CBLAS instructions.
- Bring a working account and record any unresolved setup issue.

The instructor will confirm

The next meeting's schedule, the current UPeL activity/report assignments and any remaining language or assistance-policy details.

Questions before we begin project work?

