Introduction to Matrix Computations

S. Bora

July-November, 2024

Overview

This course will be concerned with the following standard problems:

Overview

This course will be concerned with the following standard problems:

- Linear systems of equations

Overview

This course will be concerned with the following standard problems:

- Linear systems of equations
- Least squares problems

Overview

This course will be concerned with the following standard problems:

- Linear systems of equations
- Least squares problems
- Eigenvalue problems

Overview

This course will be concerned with the following standard problems:

- Linear systems of equations
- Least squares problems
- Eigenvalue problems
- Singular value problems

Overview

This course will be concerned with the following standard problems:

- Linear systems of equations
- Least squares problems
- Eigenvalue problems
- Singular value problems

The methods for solving these problems can be broadly classified as:

Overview

This course will be concerned with the following standard problems:

- Linear systems of equations
- Least squares problems
- Eigenvalue problems
- Singular value problems

The methods for solving these problems can be broadly classified as:

- Direct methods

Overview

This course will be concerned with the following standard problems:

- Linear systems of equations
- Least squares problems
- Eigenvalue problems
- Singular value problems

The methods for solving these problems can be broadly classified as:

- Direct methods
- Iterative methods

General Techniques and Issues

The following general techniques and issues will recur all throughout.

General Techniques and Issues

The following general techniques and issues will recur all throughout.

- Matrix factorizations

General Techniques and Issues

The following general techniques and issues will recur all throughout.

- Matrix factorizations
- Effects of finite precision arithmetic on the algorithms

General Techniques and Issues

The following general techniques and issues will recur all throughout.

- Matrix factorizations
- Effects of finite precision arithmetic on the algorithms
- Stability of the algorithms

General Techniques and Issues

The following general techniques and issues will recur all throughout.

- Matrix factorizations
- Effects of finite precision arithmetic on the algorithms
- Stability of the algorithms
- Perturbation theory and condition numbers

General Techniques and Issues

The following general techniques and issues will recur all throughout.

- Matrix factorizations
- Effects of finite precision arithmetic on the algorithms
- Stability of the algorithms
- Perturbation theory and condition numbers
- Speed of algorithms

Matrix Factorizations

Most of the methods for solving the problems aim to express A as a product of 'simpler' matrices which readily reveal the solution of the problem.

Matrix Factorizations

Most of the methods for solving the problems aim to express A as a product of 'simpler' matrices which readily reveal the solution of the problem.

- LU decomposition ($A = LU$).

Matrix Factorizations

Most of the methods for solving the problems aim to express A as a product of 'simpler' matrices which readily reveal the solution of the problem.

- LU decomposition ($A = LU$).
- QR decomposition ($A = QR$).

Matrix Factorizations

Most of the methods for solving the problems aim to express A as a product of 'simpler' matrices which readily reveal the solution of the problem.

- LU decomposition ($A = LU$).
- QR decomposition ($A = QR$).
- The Schur decomposition ($A = UTU^H$).

Matrix Factorizations

Most of the methods for solving the problems aim to express A as a product of 'simpler' matrices which readily reveal the solution of the problem.

- LU decomposition ($A = LU$).
- QR decomposition ($A = QR$).
- The Schur decomposition ($A = UTU^H$).
- The eigendecomposition ($A = XDX^{-1}$)

Matrix Factorizations

Most of the methods for solving the problems aim to express A as a product of 'simpler' matrices which readily reveal the solution of the problem.

- LU decomposition ($A = LU$).
- QR decomposition ($A = QR$).
- The Schur decomposition ($A = UTU^H$).
- The eigendecomposition ($A = XDX^{-1}$)
- The Singular Value decomposition (SVD) ($A = USV^H$).

Rounding: The Silent Killer

Arithmetic operations on computers are inexact and messy. For example, computing

$$a + b$$

on a computer produces

$$(a + b)(1 + e)$$

where e is called the rounding error.

Rounding: The Silent Killer

Arithmetic operations on computers are inexact and messy. For example, computing

$$a + b$$

on a computer produces

$$(a + b)(1 + e)$$

where e is called the rounding error.
The rounding error is bounded by $|\mathbf{e}| \leq \mathbf{u}$, where $\mathbf{u}$ is the unit roundoff.

- Single precision: $\mathbf{u} \simeq \mathbf{5.96} \times \mathbf{10^{-8}}$
- Double precision: $\mathbf{u} \simeq \mathbf{1.11} \times \mathbf{10^{-16}}$

Rounding: The Silent Killer

Arithmetic operations on computers are inexact and messy. For example, computing

$$a + b$$

on a computer produces

$$(a + b)(1 + e)$$

where e is called the rounding error.
The rounding error is bounded by $|\mathbf{e}| \leq \mathbf{u}$, where $\mathbf{u}$ is the unit roundoff.

- Single precision: $\mathbf{u} \simeq \mathbf{5.96} \times \mathbf{10^{-8}}$
- Double precision: $\mathbf{u} \simeq \mathbf{1.11} \times \mathbf{10^{-16}}$

IEEE standard allows to track small errors made when two numbers are added, subtracted, multiplied or divided on a computer.

Hazards of automatic computation

The rules of arithmetic are lost on computers! For example, generally

Hazards of automatic computation

The rules of arithmetic are lost on computers! For example, generally

$$\begin{aligned}
\mathbf{a} + (\mathbf{b} + \mathbf{c}) &\neq (\mathbf{a} + \mathbf{b}) + \mathbf{c} \\
\mathbf{a} \times (\mathbf{b} + \mathbf{c}) &\neq \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c} \\
\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) &\neq (\mathbf{a} \times \mathbf{b}) \times \mathbf{c}
\end{aligned}$$

Hazards of automatic computation

The rules of arithmetic are lost on computers! For example, generally

$$\begin{aligned} \mathbf{a} + (\mathbf{b} + \mathbf{c}) &\neq (\mathbf{a} + \mathbf{b}) + \mathbf{c} \\ \mathbf{a} \times (\mathbf{b} + \mathbf{c}) &\neq \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c} \\ \mathbf{a} \times (\mathbf{b} \times \mathbf{c}) &\neq (\mathbf{a} \times \mathbf{b}) \times \mathbf{c} \end{aligned}$$

On my computer MATLAB produces:

$$\begin{aligned} (\frac{4}{3} - 1) * 3 - 1 &= -2.2204 \times 10^{-16} \\ 5 \times \frac{(1 + \exp(-50)) - 1}{(1 + \exp(-50)) - 1} &= \mathbf{NaN}. \\ \frac{\log(\exp(750))}{100} &= \mathbf{Inf}. \end{aligned}$$

Hazards of automatic computation

For day-to-day ordinary computations things are not that bad. But things can go wrong. Safeguards must be taken against unpleasant results.

Hazards of automatic computation

For day-to-day ordinary computations things are not that bad. But things can go wrong. Safeguards must be taken against unpleasant results.

Prevention is better than cure!

Stability of algorithms

Analysing the errors caused by the algorithm itself requires knowing the effect of rounding errors during the execution of the algorithm. A desirable property of algorithms is *backward stability:*

> *If an algorithm alg*(x) *is used to compute* $f(x)$, *then including the effect of rounding error, alg*(x) *is said to be backward stable if alg*$(x) = f(x + \delta x)$ *for small* δx. *Here* δx *is called the backward error.*

Stability of algorithms

Analysing the errors caused by the algorithm itself requires knowing the effect of rounding errors during the execution of the algorithm. A desirable property of algorithms is *backward stability:*

> *If an algorithm alg*(x) *is used to compute* $f(x)$, *then including the effect of rounding error, alg*(x) *is said to be backward stable if alg*$(x) = f(x + \delta x)$ *for small* δx. *Here* δx *is called the backward error.*

Thus a backward stable algorithm provides the exact answer to a slightly perturbed problem.

Perturbation Theory and Condition Numbers.

The computed solution to a problem provided by an algorithm is seldom its exact solution. The difference may be largely attributed to:

Perturbation Theory and Condition Numbers.

The computed solution to a problem provided by an algorithm is seldom its exact solution. The difference may be largely attributed to:

- Error in input data arising from prior calculations, measurement errors and rounding errors due to finite precision arithmetic.

Perturbation Theory and Condition Numbers.

The computed solution to a problem provided by an algorithm is seldom its exact solution. The difference may be largely attributed to:

- Error in input data arising from prior calculations, measurement errors and rounding errors due to finite precision arithmetic.
- Errors introduced by the algorithm when making approximations.

Perturbation Theory and Condition Numbers.

The computed solution to a problem provided by an algorithm is seldom its exact solution. The difference may be largely attributed to:

- Error in input data arising from prior calculations, measurement errors and rounding errors due to finite precision arithmetic.
- Errors introduced by the algorithm when making approximations.

This makes it necessary to understand the sensitivity of the solution to perturbations in the input data. The condition number of a problem is a measure of this sensitivity.

Perturbation Theory and Condition Numbers.

The computed solution to a problem provided by an algorithm is seldom its exact solution. The difference may be largely attributed to:

- Error in input data arising from prior calculations, measurement errors and rounding errors due to finite precision arithmetic.
- Errors introduced by the algorithm when making approximations.

This makes it necessary to understand the sensitivity of the solution to perturbations in the input data. The condition number of a problem is a measure of this sensitivity.

Thus if the algorithm is backward stable, then the *forward error* which is the difference between its exact and computed solutions is small if the solution is not too sensitive to perturbation.

Speed of algorithms

The speed of an algorithm is traditionally measured in terms of the number of *floating point operations* or *flops* it performs. The design of every algorithm will be accompanied by an estimate of its *flop count.*

Speed of algorithms

The speed of an algorithm is traditionally measured in terms of the number of *floating point operations* or *flops* it performs. The design of every algorithm will be accompanied by an estimate of its *flop count.*

However it may take significantly longer to move the data to the point at which it is operated upon than to actually perform the operations. Therefore, the speed also greatly depends upon the order and manner of implementation of the operations.

Speed of algorithms

The speed of an algorithm is traditionally measured in terms of the number of *floating point operations* or *flops* it performs. The design of every algorithm will be accompanied by an estimate of its *flop count.*

However it may take significantly longer to move the data to the point at which it is operated upon than to actually perform the operations. Therefore, the speed also greatly depends upon the order and manner of implementation of the operations.

If an algorithm is *iterative*, then it is necessary to know the number of iterations necessary to accept any approximate solution as an answer. This is decided by the quality of the convergence, whether *linear, quadratic or cubic....*

Class timings, Texts and References

Class Timings:
Theory classes:
Room 2202, Monday (10-11), Tuesday (11-12), Friday (9-10)
Practical classes:
Department Lab, Monday (2-4).

Class timings, Texts and References

Class Timings:
Theory classes:
Room 2202, Monday (10-11), Tuesday (11-12), Friday (9-10)
Practical classes:
Department Lab, Monday (2-4).

Textbook:

- Matrix Computations by D. S. Watkins (2nd or 3rd edition)

Class timings, Texts and References

Class Timings:
Theory classes:
Room 2202, Monday (10-11), Tuesday (11-12), Friday (9-10)
Practical classes:
Department Lab, Monday (2-4).

Textbook:

- Matrix Computations by D. S. Watkins (2nd or 3rd edition)

Reference books:

- Applied Numerical Linear Algebra by James Demmel.
- Numerical Linear Algebra by Trefethen and Bau.
- Numerical Computing with IEEE floating point arithmetic by Michael Overton.

Evaluation Schedule & Policy

Evaluation Schedule & Policy

Theory (120):

10 (Quiz I, tentatively within 26-28 August)
40 (Midsem)
10 (Quiz II, tentatively within 28-30 October)
60 (Endsem)

Practical (80):

40 (Midsem on 9 September)
40 (Endsem on 11 November)

Total: 200 marks.

Evaluation Schedule & Policy

Theory (120):

10 (Quiz I, tentatively within 26-28 August)
40 (Midsem)
10 (Quiz II, tentatively within 28-30 October)
60 (Endsem)

Practical (80):

40 (Midsem on 9 September)
40 (Endsem on 11 November)

Total: 200 marks.

Evaluation Schedule & Policy

Theory (120):

10 (Quiz I, tentatively within 26-28 August)
40 (Midsem)
10 (Quiz II, tentatively within 28-30 October)
60 (Endsem)

Practical (80):

40 (Midsem on 9 September)
40 (Endsem on 11 November)

Total: 200 marks.

To pass the course you will need:

Class attendance $\geq$ 75%
&
A score of $\geq$ 10 in Lab assessments
&
A score of $\geq$ 20 in Theory assessments