Numerical Simulation and Scientific Computing II

Lecture 1: Introduction and Distributed Parallel Computing I

Josef Weinbub, Paul Manstetten, Heinz Pettermann, Asur Vijaya Kumar, Pavan Kumar, Jesús Carrete Montana, Francesco Zonta, Kevin Sturm

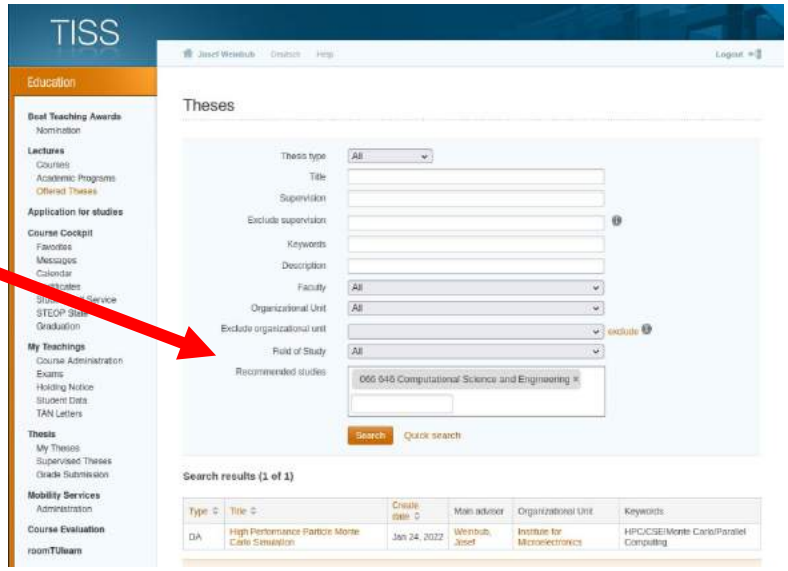
Institute for Microelectronics
TU Wien

nssc@iue.tuwien.ac.at

SS 2023

Master's Thesis Topics: Quantum Monte Carlo Simulator on Supercomputers

- **Development of new highly parallel C++ Monte Carlo Simulator**
- **Several parallelization layers: MPI/OpenMP/CUDA**
- **Tailored to Supercomputers**



Possible Tasks

- **Switchable OpenMP/CUDA kernels**
- **MPI domain decomposition**
- **Structure generator**
- **Performance studies: Algorithms, data structures ..**
- **Automated testing pipelines**
- **and many more ..**

Required knowledge: (at least one of the following)

- **C++ / MPI / OpenMP / CUDA**

Contact: josef.weinbub@tuwien.ac.at

Outline

- **Introduction to the Lecture**
- Distributed Parallel Computing I
  - Primer
  - Overview
  - Process Model and Language Bindings
  - Messages and Point-to-Point Communication
  - Examples
- Quiz

Team Presentation

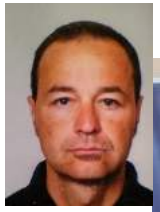
Core Team (Organization, MPI, ODE, FVM)

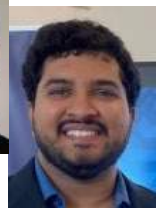
- **Josef Weinbub**
- **Paul Manstetten**
  Institute for Microelectronics

Extended Interdisciplinary Team

- **Francesco Zonta** (Fluid Dynamics)
  Institute of Fluid Mechanics and Heat Transfer

- **Heinz Pettermann,**
  Asur Vijaya Kumar, Pavan Kumar (Finite Elements)
  Institute of Lightweight Design and Structural Biomechanics





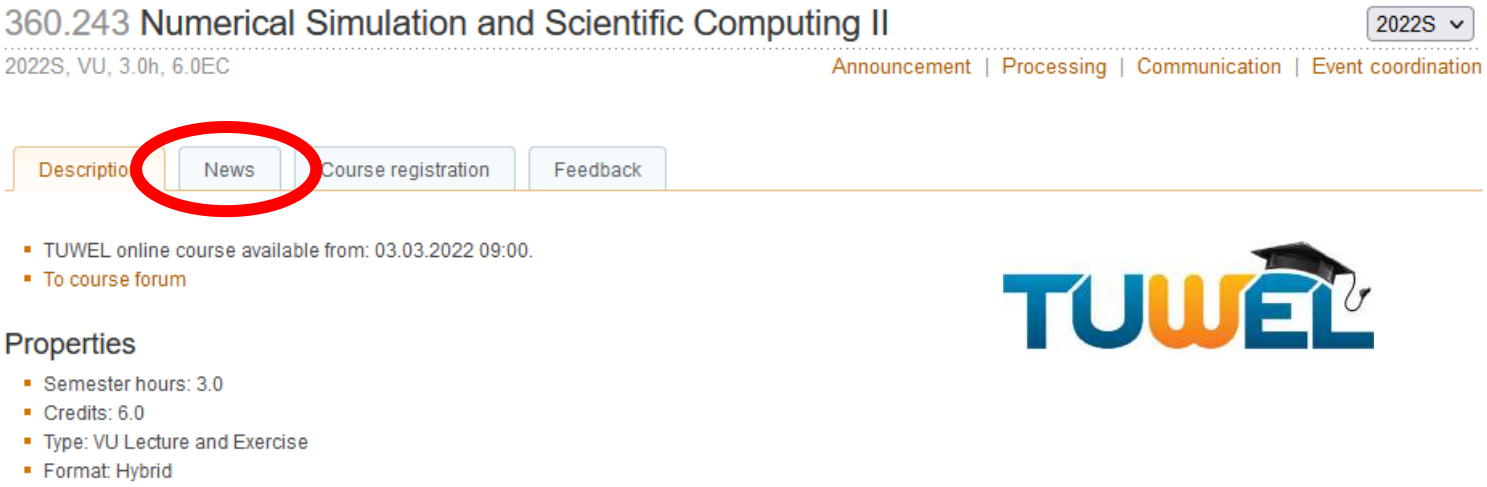
- **Jesús Carrete Montana** (Molecular Dynamics)
  Institute of Materials Chemistry

- **Kevin Sturm** (Partial Differential Equations)
  Institute of Analysis and Scientific Computing

nssc@iue.tuwien.ac.at

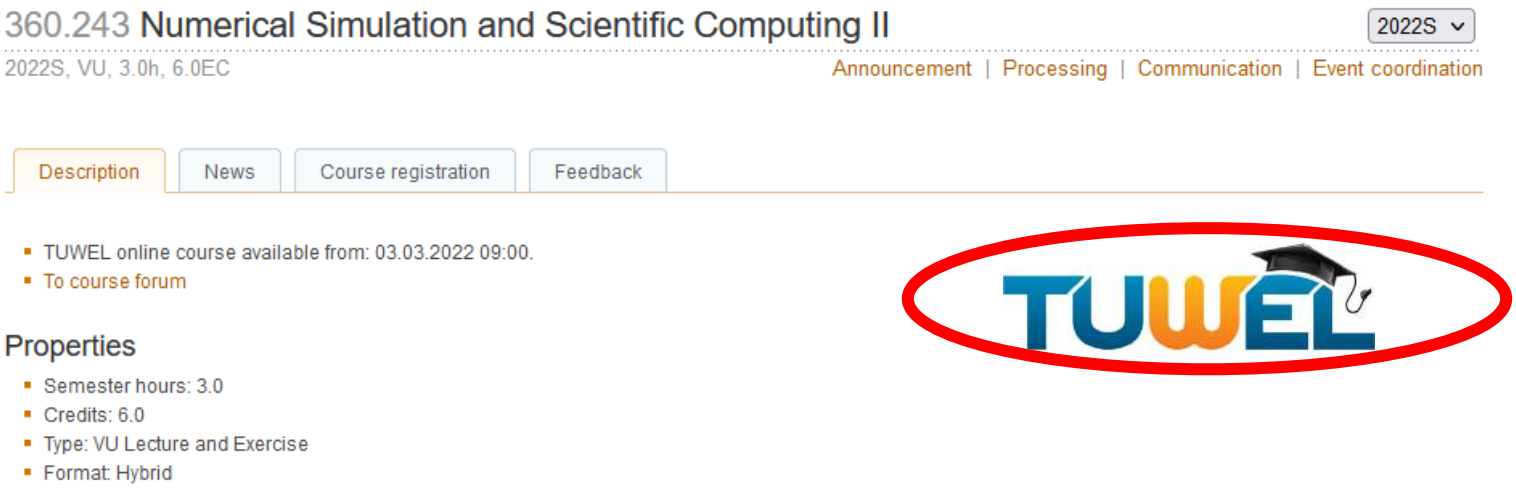
TISS

- **Acceptance to course via TISS: Check today**
- **News will be sent via TISS' notification feature: Regularly monitor your TISS News (at least weekly)!**



TUWEL

- **Course material (slides, handouts, zoom link) provided via TUWEL course**
- **TUWEL course “reachable” via TISS course, watch out for the TUWEL icon:**



General Goals

- **Introduction to advanced methods of CSE**
  - **Distributed Parallel Computing**
  - **Methods for Ordinary Differential Equations**
  - **Classification and Analyses of Partial Differential Equations**
  - **Finite Volume Method**
  - **Finite Element Method**
  - **Fluid Dynamics**
  - **Molecular Dynamics**

Necessary Background

- **C++ and Python! (see TISS)**
  - **If not experienced: Checkout online tutorials asap!**
- **Not a formal requirement**
  yet the expected level of expertise:
  Numerical Simulation and Scientific Computing I
  360.242

Exercise Rules and Course Grade

- 4 mandatory exercises over the whole lecture
- 7 Groups of 3 students and 1 group of 2 students must be formed; student-picked.
  - Go find your group today and inform: nssc@iue.tuwien.ac.at
  - Mind the deadline! (see Course Calendar, slide 10)
  - No changes during the term!
- Code copied from other group: 0 points for both groups!
  - If copied from previous year: also 0 points.
- Each exercise will be graded separately from 0-10 points
- Access for the final exam: sum of the points ≥ 28.0 (same procedure as with NSSC I)
- Course grade: 1/4th Exercises, 3/4th Exam
- Submission details will be communicated with handout

Rules - Quizzes

- In the end of each lecture, you will receive 5 questions
- 3 questions reviewing the current lecture
- 2 questions preparing for the next
- Discussion in the beginning of next class
- Participation is voluntary but encouraged
- These questions might help you learn for the exam!

Course Calendar

March 2	Lecture
March 9	Lecture and Exercise Handout **Send group info to** nssc@iue.tuwien.ac.at
March 16	Lecture and Exercise Support
March 22	(Wednesday!) Lecture (zoom)
March 23	Exercise Support
March 30	Lecture and Exercise Submission
April 21	(Friday! Start at 10:00) Lecture and Exercise Handout (zoom)
April 27	Lecture and Exercise Support
May 4	Lecture and Exercise Submission/Handout
May 11	Lecture and Exercise Support
May 17	(Wednesday!) Lecture and Exercise Submission/Handout
May 25	Exercise Support
June 1	Exercise Support
June 7	(Wednesday!) Exercise Submission
June 15	Backup
June 22	Backup
June 29	Main Exam

- **If not otherwise states, all units start at 14:00 sharp**
- **Exercise submission is done via TUWEL/Email (see exercise handout): No need to show up on submission-only units (e.g., June 7)**
- **Will put lecture slides/exercise material on TUWEL right before slot**
- **Zoom infos in TUWEL**
- **Detailed Schedule also in TISS**

Computer Resources

- It is expected that you have access to a modern, x86-based, personal computer (laptop, home desktop, etc.)
- Linux (virtual machine, dual boot or full OS) is required
  - More information on the development environment for a particular exercise will be made available with the exercise handout

Outline

- Introduction to the Lecture
- **Distributed Parallel Computing I**
  - **Primer**
  - Overview
  - Process Model and Language Bindings
  - Messages and Point-to-Point Communication
  - Examples
- Quiz

Acknowledgments

Thanks to Rolf Rabenseifner's (HLRS)

2014 Parallel Programming Workshop Lecture Material on:

Introduction to the Message Passing Interface (MPI)

Sources

- High Performance Computing Center Stuttgart (HLRS) Online Courses https://www.hlrs.de/about-us/media-publications/teaching-training-material/
- YouTube -- search for “Introduction to MPI”, e.g., https://youtu.be/RoQJNx5npF4 -- Part I of III

Additional Courses at TU Wien

ECTS Courses (count as free electives)

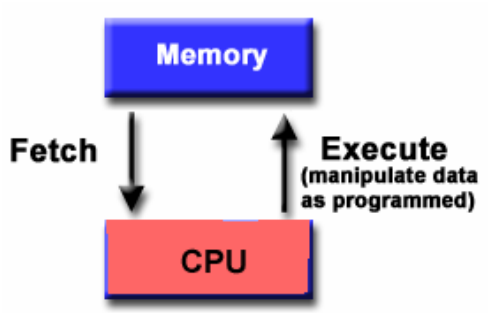
- 057.020 (Winter term)
  VSC-School I Courses in High Performance Computing
- 057.021 (Summer term)
  VSC-School II Courses in High Performance Computing

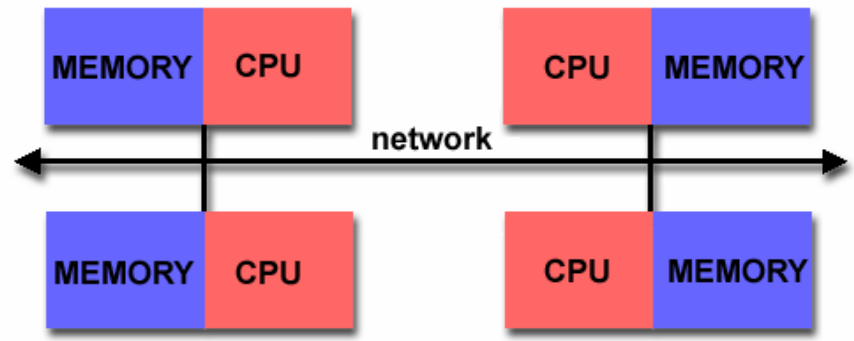
Non-ECTS Trainings (don't count as free electives)

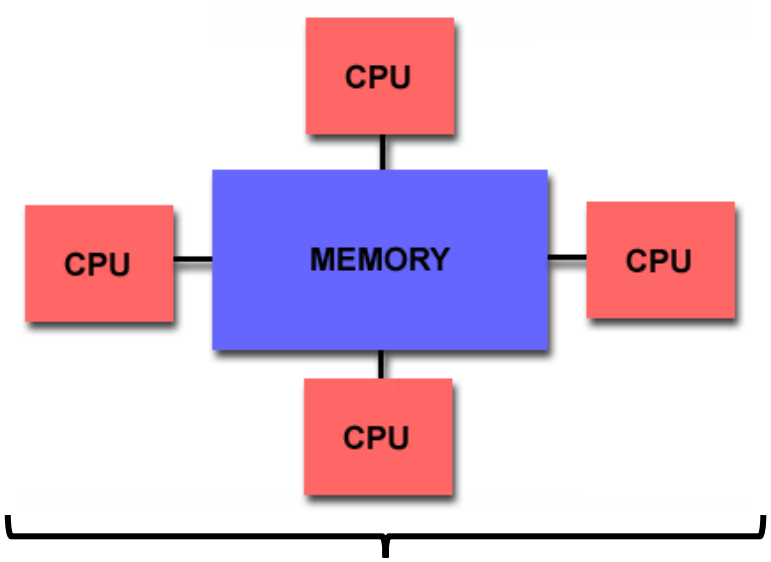
- Node-Level Performance Engineering
- OpenMP
- MPI
- Deep-Learning und GPU programming (OpenACC)
- Hybrid-programming MPI+X

http://typo3.vsc.ac.at/research/vsc-research-center/vsc-school-seminar/

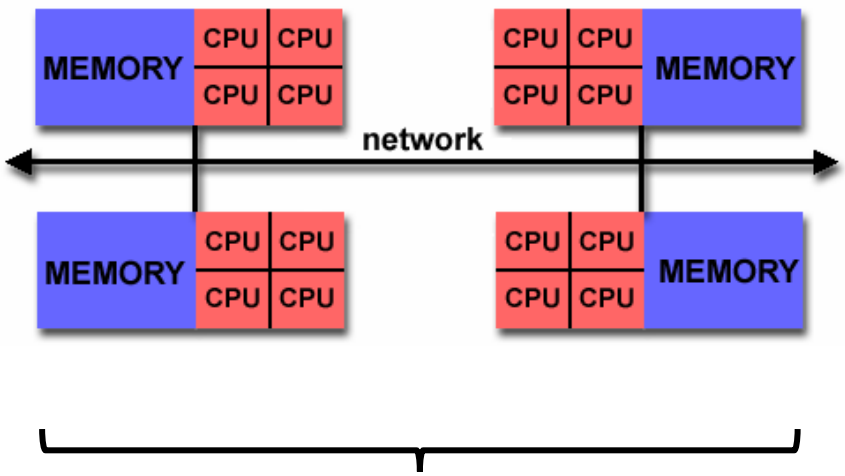
Parallel Computer Architectures







Easy to use, but not scalable



Difficult to use, but scalable

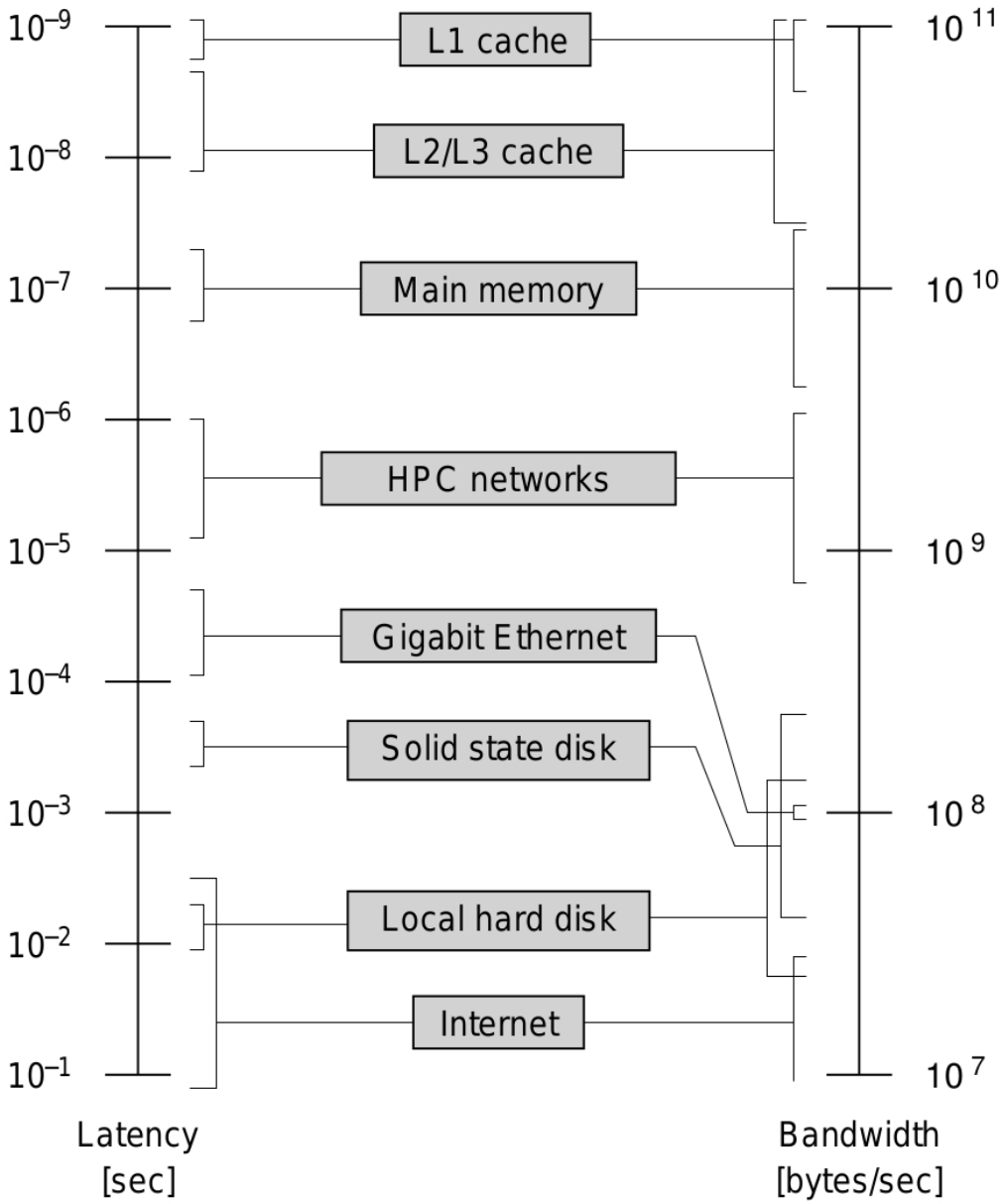
Source: Lyle N. Long, PSU

Distributed Parallel Computing Clusters

IBM Blue Gene/P: Intrepid
Argonne National Laboratory

Source: Wikipedia

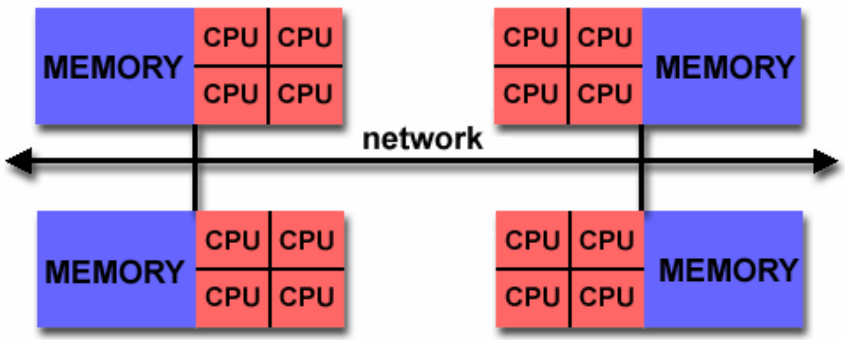
Latency / Bandwidth Hierarchy



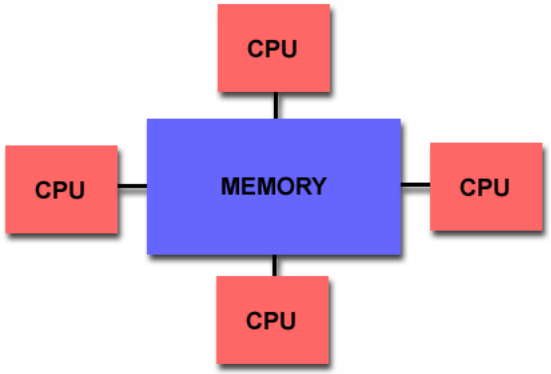
Processes and Threads

- **Process**
  - **A process is a program in execution**

Data exchange happens explicitly via sending messages between processes!



- **Thread**
  - **Unit of CPU utilization**
  - **A single process can contain multiple threads**
  - **Threads belonging to the same process can share resources**



Outline

- Introduction to the Lecture
- Distributed Parallel Computing I
  - Primer
  - **Overview**
  - Process Model and Language Bindings
  - Messages and Point-to-Point Communication
  - Examples
- Quiz

We will gradually increase the level of detail! Topics will re-emerge in later sections in a more detailed form.

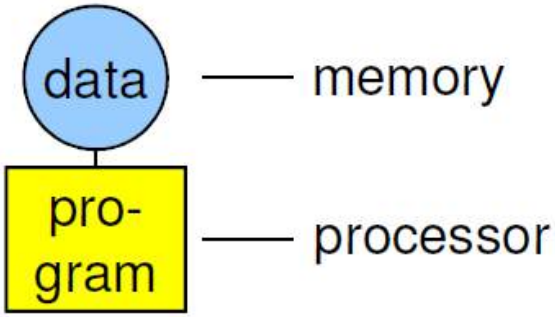
Message Passing Interface (MPI)

```
#include "mpi.h"
#include <stdio.h>
int main( int argc, char *argv[] )
{
  int myid, numprocs;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);
  printf("I am %d out of %d\n", myid, numprocs);
  MPI_Finalize();
  return 0;
}
```

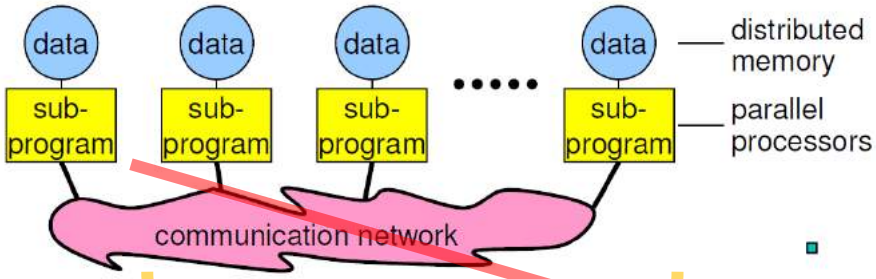
```
> mpiexec –n 4 ./hello
I am 3 out of 4
I am 1 out of 4
I am 0 out of 4
I am 2 out of 4
```

Message Passing Programming Paradigm

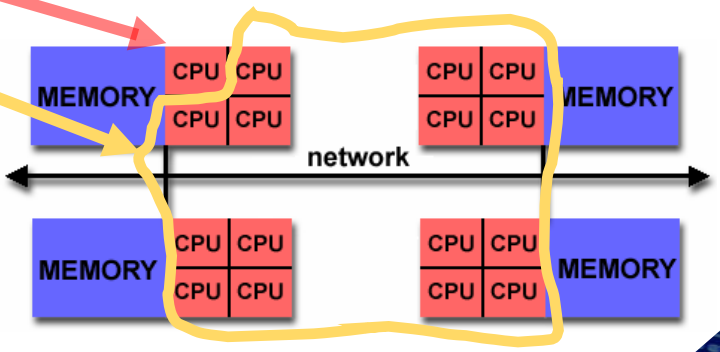
- **Sequential Programming Paradigm**



- **Message Passing Programming Paradigm**

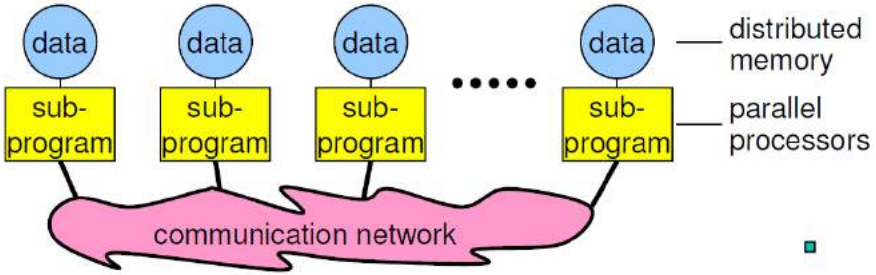



```
// identify specific process which is being executed
// on specific CPU core
if(myrank == 0) {
    // do some central organization
}
else {
    // all the other ranks should do something useful
}
```



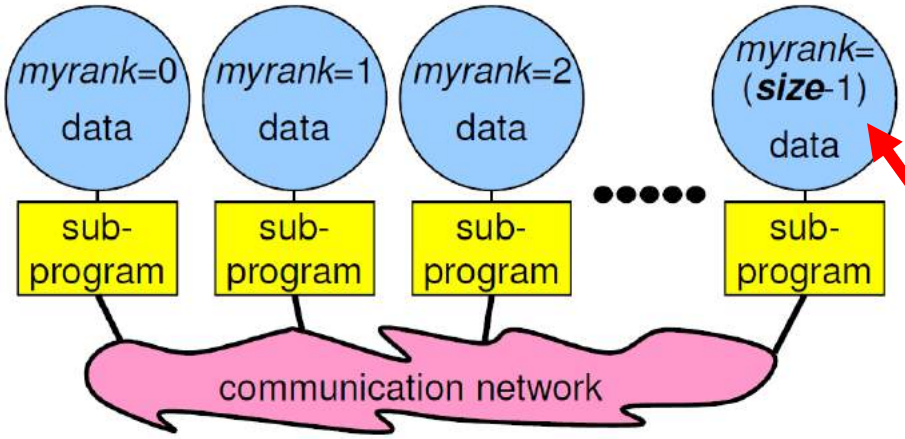
Message Passing Programming Paradigm

- **For instance: Each CPU core in a message passing program runs a *sub-program (SPMD-single program, multiple data):***
  - **written in a conventional sequential language, e.g., C/C++ or Fortran,**
  - **typically the same on each processor,**
  - **the variables of each sub-program have**
    - **the same name**
    - **but different locations (distributed memory) and different data!**
    - **i.e., all variables are private**
- **communicate via special send & receive routines (*message passing*)**



Data and Work Distribution

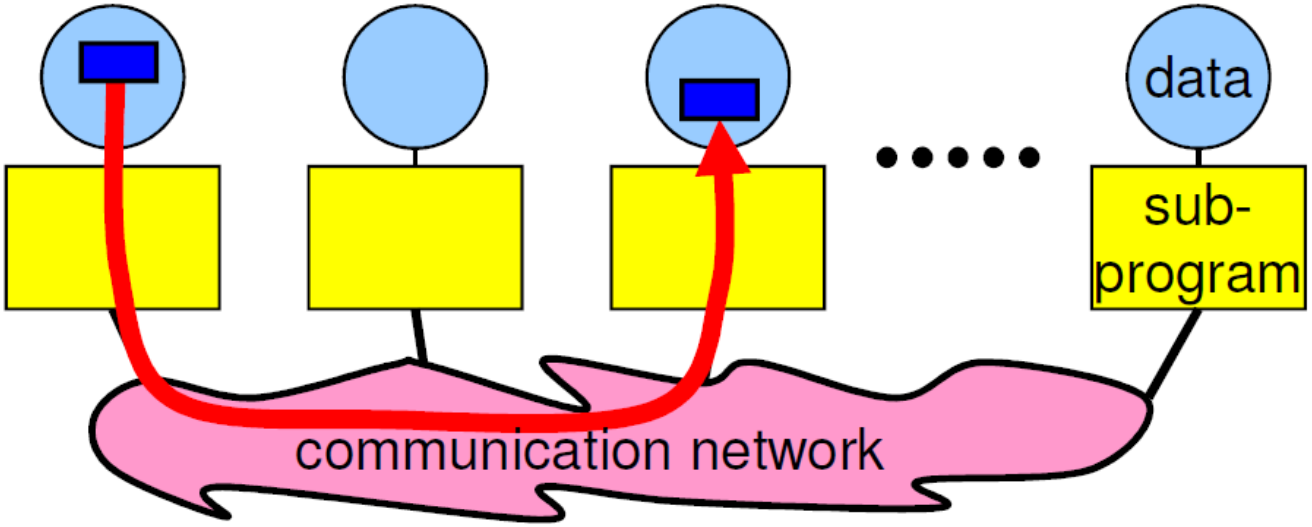
- the value of *myrank* is returned by special library routine
- the system of *size* processes is started by special MPI initialization
- program (*mpirun* or *mpiexec*)
- all distribution decisions are based on *myrank*
- i.e., which process works on which data




```
#include "mpi.h"
#include <stdio.h>
int main( int argc, char *argv[] )
{
  int myid, numprocs;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);
  printf("I am %d out of %d\n", myid, numprocs);
  MPI_Finalize();
  return 0;
}
```

Messages

- **Messages are packets of data moving between sub-programs**
- **Necessary information for the message passing system:**
  - **sending process – receiving process** } **the ranks**
  - **source location – destination location**
  - **source data type – destination data type**
  - **source data size – destination buffer size**



Access

- A sub-program needs to be connected to a message passing system
- A message passing system is similar to:
  - mail box
  - phone line
  - fax machine
  - etc.
- MPI:
  - sub-program must be linked with an MPI library
  - sub-program must use include file of this MPI library
  - the total program (i.e., all sub-programs of the program) must be started with the MPI startup tool

Addressing

- Messages need to have addresses to be sent to.
- Addresses are similar to:
  - mail addresses
  - phone number
  - fax number
  - etc.
- MPI: addresses are ranks of the MPI processes (sub-programs)

Reception

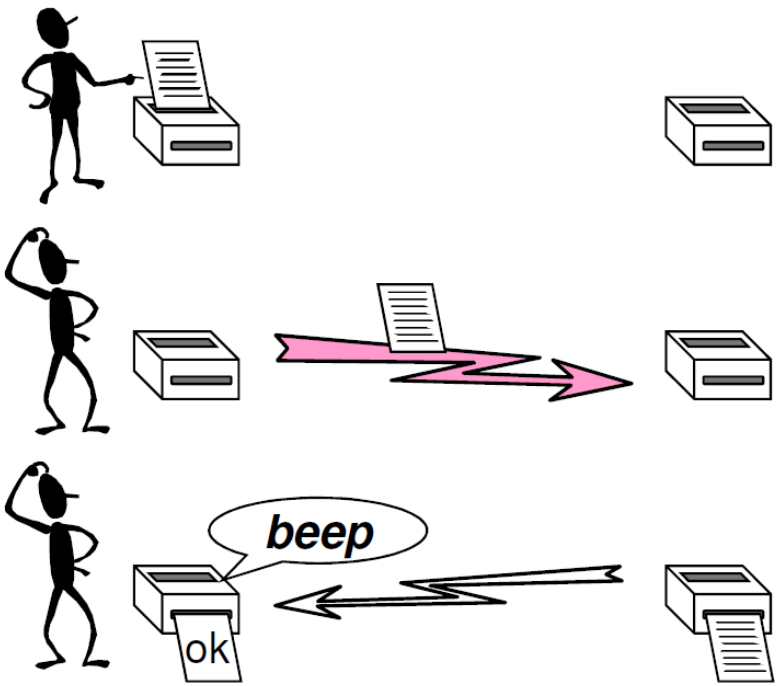
- **All messages must be received.**

Point-to-Point Communication

- Simplest form of message passing
- One process sends a message to another.
- Different types of point-to-point communication modes:
  - synchronous
  - buffered = asynchronous
  - (there is also “standard” and “ready”, more on that later)

Communication Mode: Synchronous

- **The sender gets an information that the message is received.**
- **Analogue to the *beep* or *okay-sheet* of a fax.**



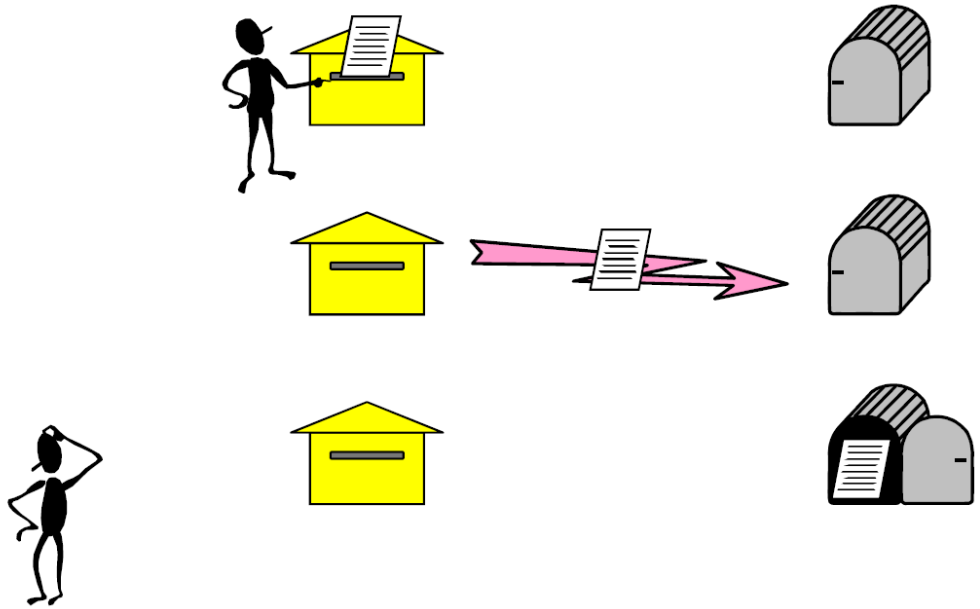
Send can be started whether or not a matching receive was posted.

Nonetheless, send will only complete successfully <u>if</u> a matching receive is posted <u>and</u> the receive process has started.

Operation is thus <u>non-local</u> as completion depends on occurrence of matching receive.

Communication Mode: Buffered (Asynchronous)

- **Only know when the message has left.**



Send can be started whether or not a matching receive was posted (same as synchronous mode).

However, the buffered send <u>may complete before</u> a matching receive is posted.

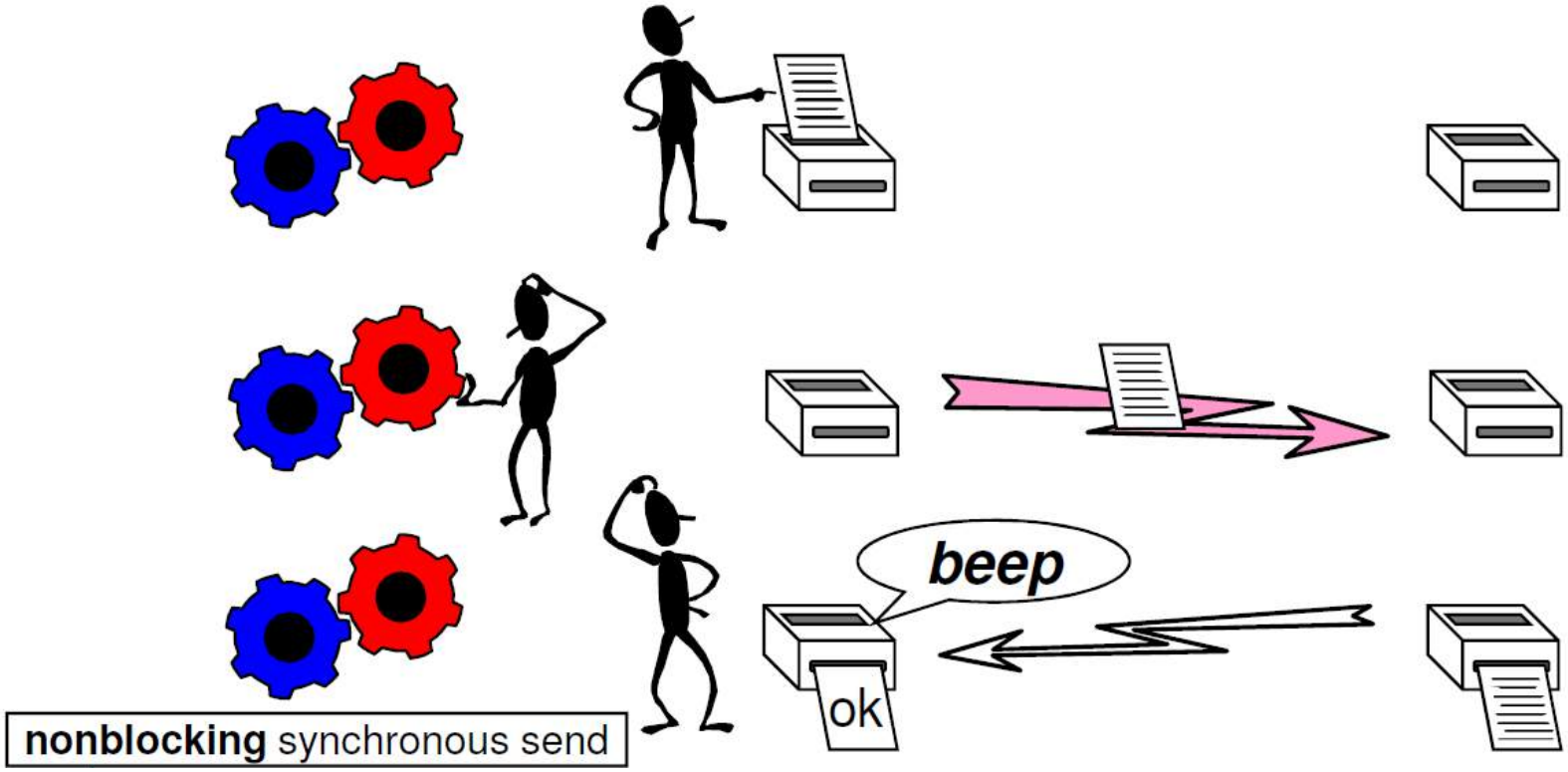
Operation is thus <u>local</u> as completion does not depend on occurrence of matching receive.

Blocking Operations

- Operations are local activities, e.g.,
  - sending (a message)
  - receiving (a message)
- Some operations may block until another process acts:
  - synchronous send operation blocks until receive is posted;
  - receive operation blocks until message was sent.
- Relates to the completion of an operation.
- Blocking subroutine returns only when the send buffer is safe to reuse / recv buffer contains the intended data.

Non-Blocking Operations

- **Non-blocking operation: returns immediately and allow the sub-program to perform other work.**
- **At some later time the sub-program must test or wait for the completion of the non-blocking operation.**



Non-Blocking Operations (cont'd)

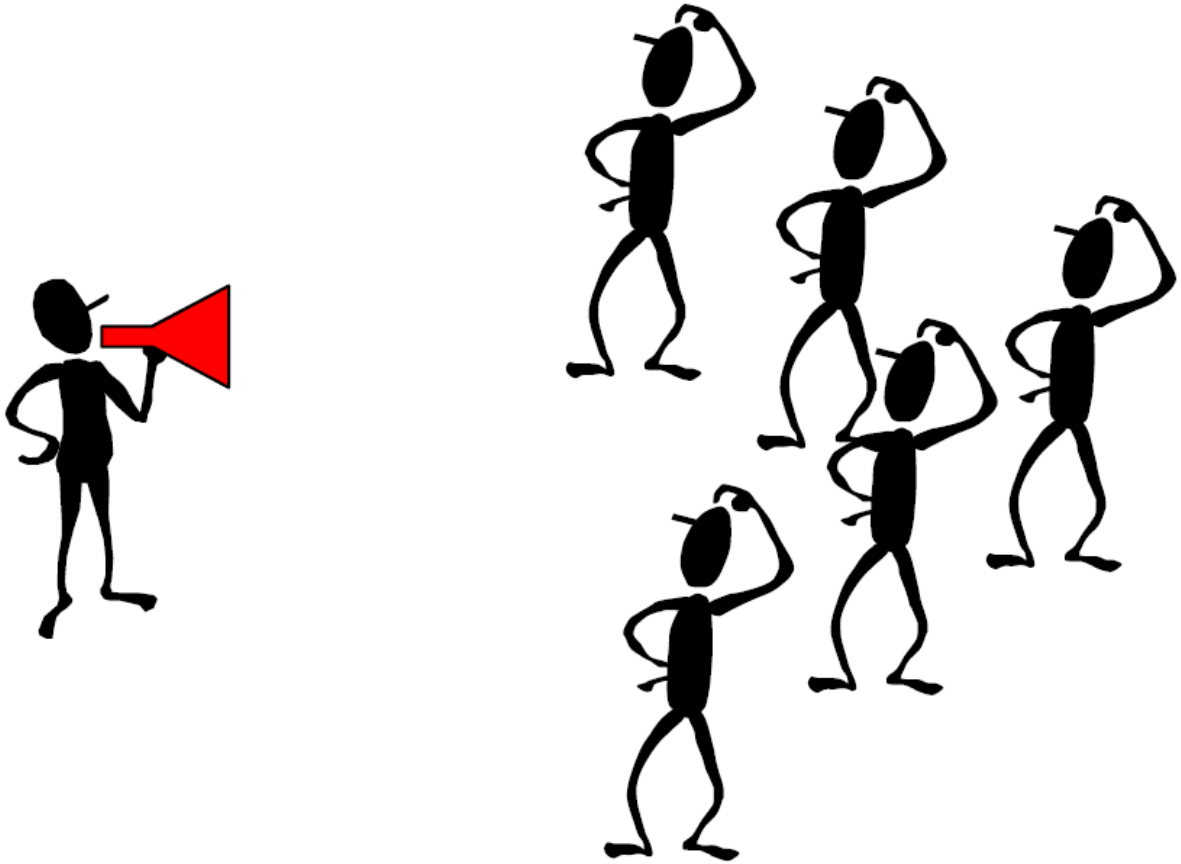
- **All non-blocking operations must have matching wait (or test) operations.**
  (Some system or application resources can be freed only when the non-blocking operation is completed.)
- **<u>A non-blocking operation immediately followed by a matching wait is equivalent to a blocking operation.</u>**
- **Non-blocking is not the same as buffered:**
  - **Blocking send returns after data has been copied out of sender memory**
  - **Non-blocking returns as soon as possible (before completion): it may not have sent/buffered the data!**
  - **Requires separate complete-call; if finished, data can be changed.**

Collective Communications

- Collective communication routines are higher level routines.
- Several processes are involved at a time.
- May allow optimized internal implementations, e.g., tree based algorithms.
- Can be built out of point-to-point communications.

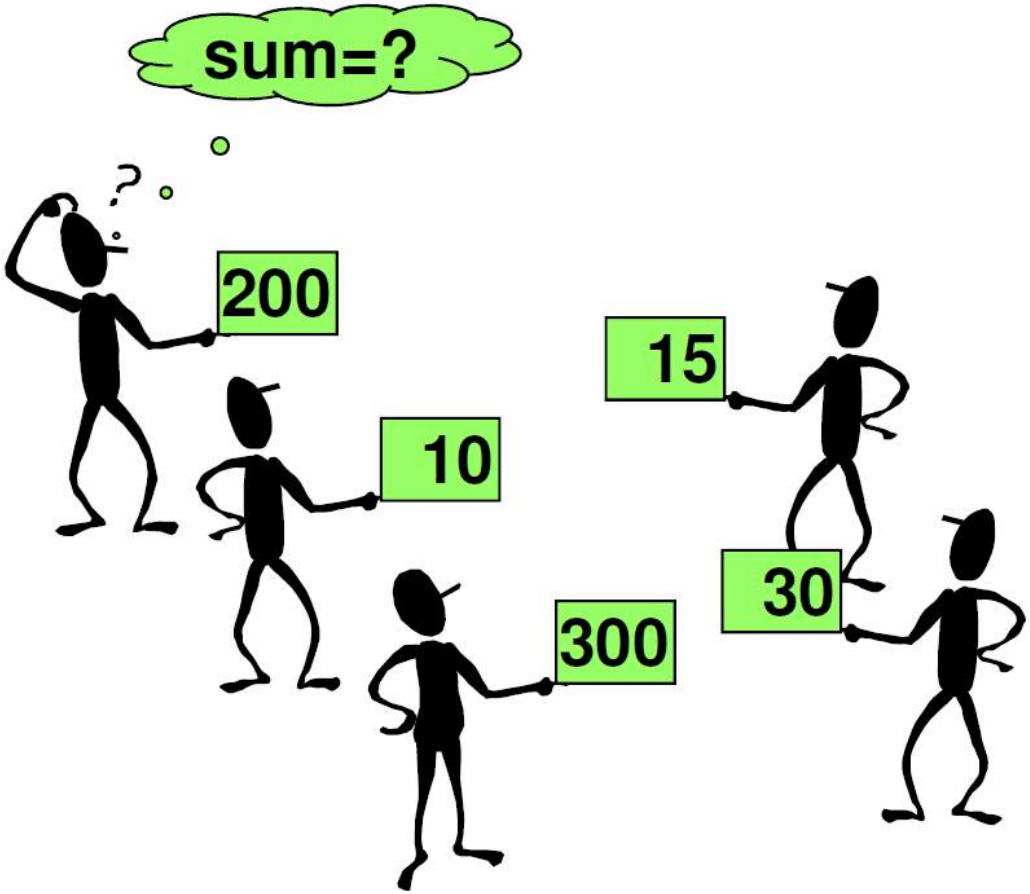
Broadcast

- **A one-to-many communication.**



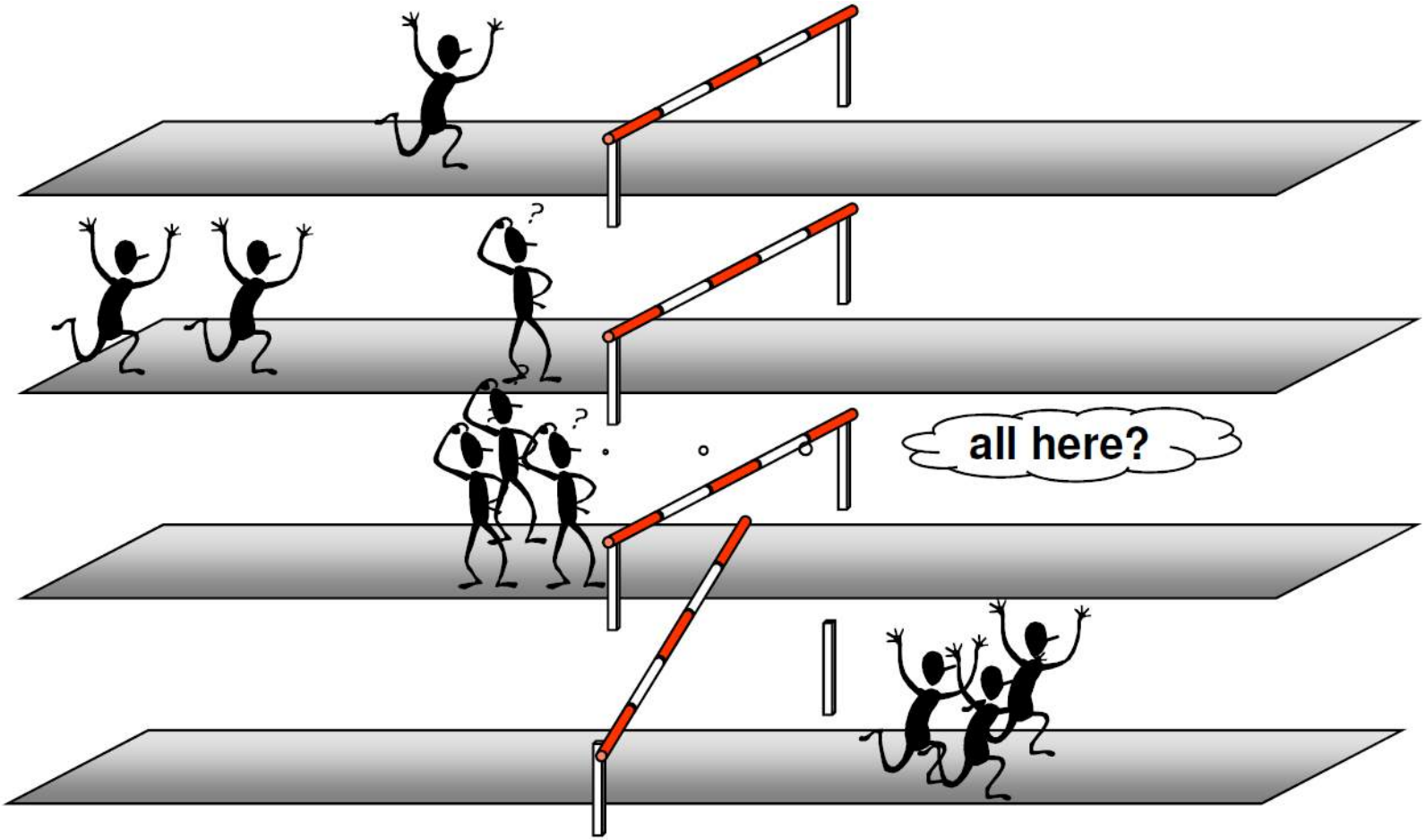
Reduction Operations

- **Combine data from several processes to produce a single result.**



Barriers

- **Synchronize processes.**



MPI Forum

- MPI is a standard
  - Each MPI routine is defined
- MPI Forum: the standardization forum www.mpi-forum.org/
- MPI-1.0 June 1994
- MPI-3.1 June 2015
- MPI-4.0 June 2021
- Many MPI libraries ((mostly) adhering to the standard) are available
  - Open source: OpenMPI, MPICH
  - Native support for C/C++, Fortran (this course focuses only on C/C++)

Outline

- Introduction to the Lecture
- Distributed Parallel Computing I
  - Primer
  - Overview
  - **Process Model and Language Bindings**
  - Messages and Point-to-Point Communication
  - Examples
- Quiz

Header Files and Function Format

- **Header Files**

```
#include <mpi.h>
```

- **MPI Namespace**
  - **MPI_... namespace is reserved for MPI constants and routines**
  - **Application routines and variable names must not begin with MPI_**

- **Function Format**

```
error = MPI_Xxxxxx( parameter, ... );
MPI_Xxxxxx( parameter, ... );
```

Initializing MPI

- **First routine to be called**

```
int MPI_Init( int *argc, char ***argv)
```

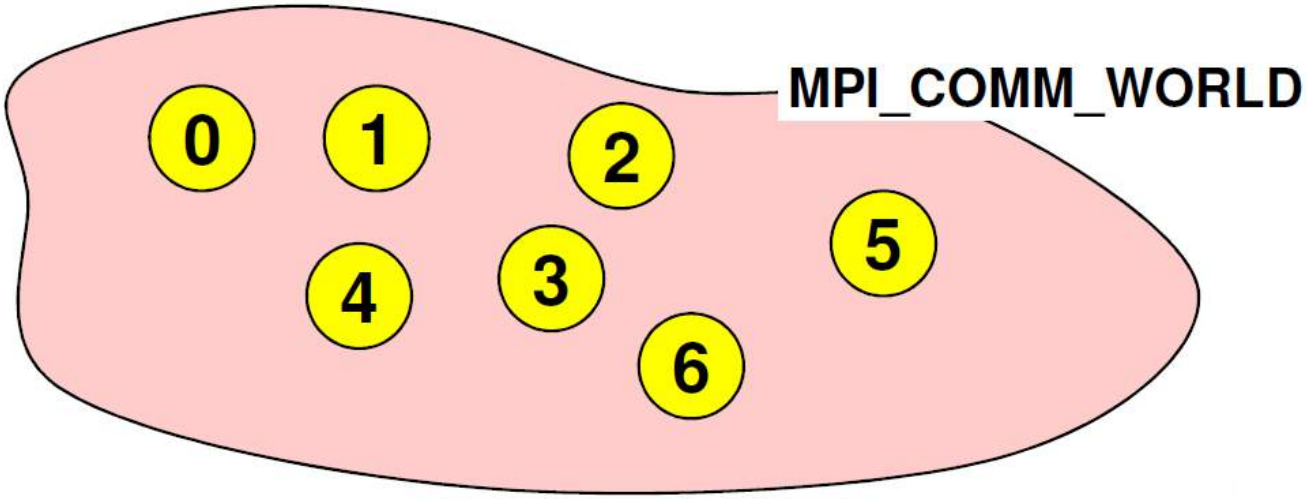
```
#include <mpi.h>
int main(int argc, char **argv)
{
  MPI_Init(&argc, &argv);
  ...
}
```

Starting the MPI Program

- **Start mechanism is implementation dependent**
- **mpirun –np *number_of_processes ./executable***
  (most implementations)
- **mpiexec –n *number_of_processes ./executable***
  (with MPI-2 and later)

Communicator MPI_COMM_WORLD

- All processes (= sub-programs) of one MPI program are combined in the communicator MPI_COMM_WORLD.
- MPI_COMM_WORLD is a predefined handle in mpi.h.
- Each process has its own rank in a communicator:
  - starting with 0
  - ending with (size-1)



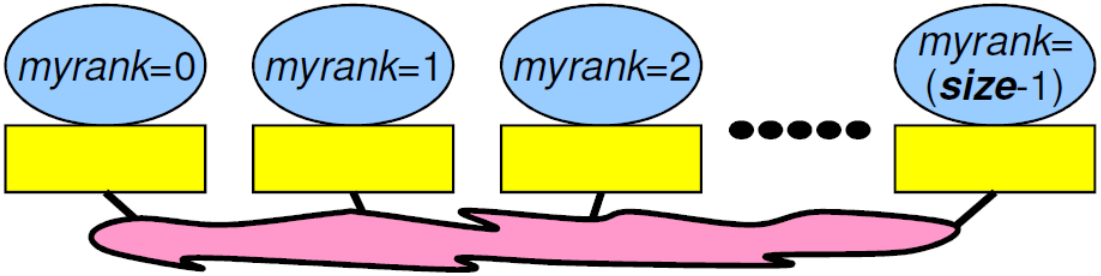
Handles

- <u>Handles</u> identify MPI objects.
- For the programmer, handles are
  - predefined constants in mpi.h
    - Example: MPI_COMM_WORLD
    - Can be used in initialization expressions or assignments.
    - The object accessed by the predefined constant handle exists and does not change only between MPI_Init and MPI_Finalize.
  - values returned by some MPI routines, to be stored in variables, that are defined as

Rank

- **The rank identifies different processes.**
- **The rank is the basis for any work and data distribution.**

```
int MPI_Comm_rank( MPI_Comm comm, int *rank)
```




```
CALL MPI_COMM_RANK( MPI_COMM_WORLD, myrank, ierror)
```

Size

- **How many processes are contained within a communicator?**

```
int MPI_Comm_size( MPI_Comm comm, int *size)
```

Exiting MPI

- **<u>Must be</u> called last by all processes.**
- **User must ensure the completion of all pending communications (locally) before calling finalize**
- **After MPI_Finalize:**
  - **Further MPI-calls are forbidden**
  - **Especially re-initialization with MPI_Init is forbidden**
  - **May abort all processes except “rank==0” in MPI_COMM_WORLD**

```
int MPI_Finalize()
```

Outline

- Introduction to the Lecture
- Distributed Parallel Computing I
  - Primer
  - Overview
  - Process Model and Language Bindings
  - **Messages and Point-to-Point Communication**
  - Examples
- Quiz

Messages

- **A message contains a number of elements of some particular datatype.**
- **MPI datatypes:**
  - **Basic datatype.**
  - **Derived datatypes.**
- **Derived datatypes can be built up from basic or derived datatypes.**
- **Datatype handles are used to describe the type of the data in the memory.**

Example: message with 5 integers

2345	654	96574	-12	7676

MPI Basic Datatypes

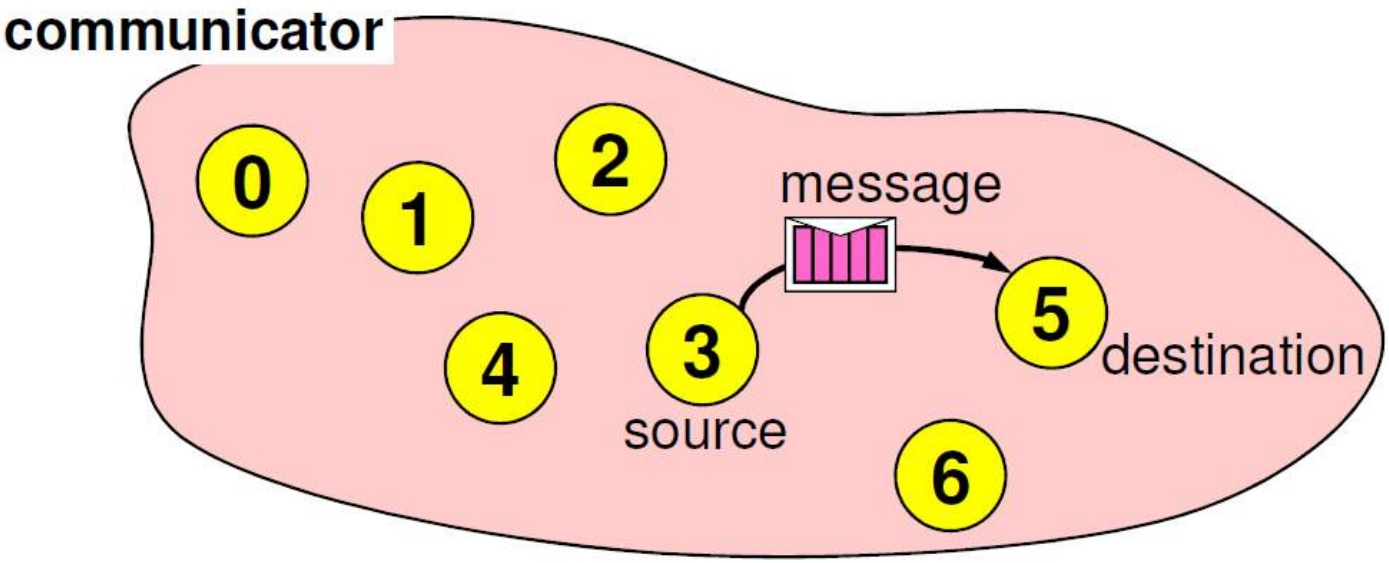
MPI Datatype	C datatype	Remarks
MPI_CHAR	char	Treated as printable character
MPI_SHORT	signed short int	
MPI_INT	signed int	
MPI_LONG	signed long int	
MPI_LONG_LONG	signed long long	
MPI_SIGNED_CHAR	signed char	Treated as integral value
MPI_UNSIGNED_CHAR	unsigned char	Treated as integral value
MPI_UNSIGNED_SHORT	unsigned short int	
MPI_UNSIGNED	unsigned int	
MPI_UNSIGNED_LONG	unsigned long int	
MPI_UNSIGNED_LONG_LONG	unsigned long long	
MPI_FLOAT	float	
MPI_DOUBLE	double	
MPI_LONG_DOUBLE	long double	
MPI_BYTE		
MPI_PACKED		

Further datatypes, see, e.g., MPI-3.0, Annex A.1

Includes also special C++ types, e.g., bool, see page 666

Point-to-Point Communication

- Communication between two processes.
- Source process sends message to destination process.
- Communication takes place within a communicator, e.g., MPI_COMM_WORLD.
- Processes are identified by their ranks in the communicator.



Sending a Message

```
int MPI_Send(    void*          buf,
                 int            count,
                 MPI_Datatype   datatype,
                 int            dest,
                 int            tag,
                 MPI_Comm       comm)
```

Example

```
int number = 4;
int myrank;
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if(myrank == 0)
  // send number to rank 1 using tag 0
  MPI_Send(&number, 1, MPI_INT, 1, 0,
  MPI_COMM_WORLD);
```

- **buf is the starting point of the message with count elements, each described with datatype.**
- **dest is the rank of the destination process within the communicator comm.**
- **tag is an additional nonnegative integer piggyback information, additionally transferred with the message.**
- **The tag can be used by the program to distinguish different types of messages.**

Receiving a Message

```
int MPI_Recv(   void*          buf,
                int            count,
                MPI_Datatype   datatype,
                int            source,
                int            tag,
                MPI_Comm       comm,
                MPI_Status*    status)
```

Example

```
int number;
int myrank;
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if(myrank == 1)
  // recv number from rank 0 using tag 0
  MPI_Recv(&number, 1, MPI_INT, 0, 0,
  MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

- ***buf*/count/datatype describe the receive buffer.**
- **Receiving the message sent by process with rank source in comm.**
- **Envelope information is returned in *status*.**
- **On can pass MPI_STATUS_IGNORE instead of a status argument.**
- **Output arguments are printed *blue-cursive*.**
- **Only messages with matching tag are received.**

Requirements for Point-to-Point Communications

For a communication to succeed:

- **Sender must specify a valid destination rank.**
- **Receiver must specify a valid source rank.**
- **The communicator must be the same.**
- **Tags must match.**
- **Buffer's type must match with the datatype handle (in the send and receive call)**
- **Message datatypes must match.**
- **Receiver's buffer must be large enough.**

Wildcarding

- **Receiver can wildcard.**
- **To receive from any <u>source:</u>**
  source = MPI_ANY_SOURCE
- **To receive from any <u>tag:</u>**
  tag = MPI_ANY_TAG
- **Actual source and tag are returned in the receiver's *status* parameter.**

Example

```
int number;
int myrank;
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if(myrank == 1)
  // recv number from any rank using tag 0
  MPI_Recv(&number, 1, MPI_INT, MPI_ANY_SOURCE, 0,
  MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

Communication Envelope

- **Envelope information is returned from MPI_RECV in *status*.**

```
MPI_Status status;
status.MPI_SOURCE
status.MPI_TAG
status.MPI_ERROR
```

- **Receive Message Count**

```
int MPI_Get_count(      MPI_Status*     status,
                        MPI_Datatype    datatype,
                        int*            count)
```

(Adapted) Example

```
int number, number_amount;
int myrank;
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if(myrank == 1) {
  MPI_Status status;
  // recv number from rank 0 using tag 0
  MPI_Recv(&number, 1, MPI_INT, 0, 0,  MPI_COMM_WORLD, &status);
  MPI_Get_count(&status, MPI_INT, &number_amount);
  printf(“Rank 1 received %d numbers. Message source = %d, tag = %d\n",
          number_amount, status.MPI_SOURCE, status.MPI_TAG);
}
```

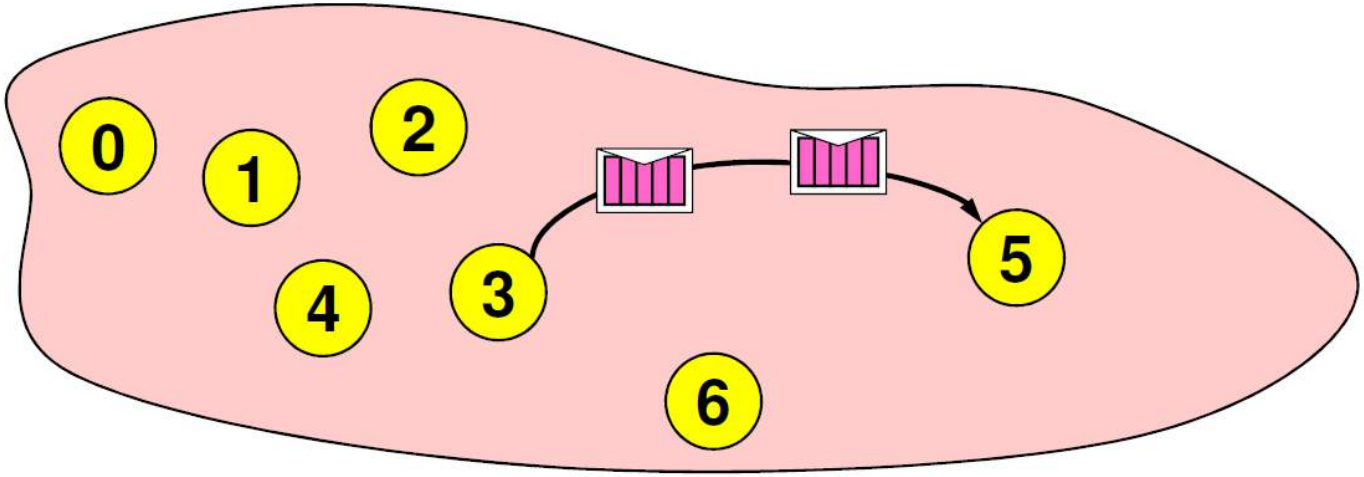
Communication Modes

- **Send communication modes:**
  - **synchronous send → MPI_SSEND**
  - **buffered [asynchronous] send → MPI_BSEND**
  - **standard send → MPI_SEND**
  - **Ready send → MPI_RSEND**
- **Receiving all modes → MPI_RECV**

Sender mode	Definition	Notes
Synchronous send **MPI_SSEND**	Only completes when the receive has started	
Buffered send **MPI_BSEND**	Always completes (unless an error occurs), irrespective of receiver	needs application-defined buffer to be declared with MPI_BUFFER_ATTACH
Standard send **MPI_SEND**	Either synchronous or buffered	uses an internal buffer
Ready send **MPI_RSEND**	May be started **only** if the matching receive is already posted!	highly dangerous!
Receive **MPI_RECV**	Completes when a message has arrived	same routine for all communication modes

Message Order Preservation

- Rule for messages on the same connection,
- i.e., same communicator, source, and destination rank:
- Messages do not overtake each other.
- This is true even for non-synchronous sends.
- If the here shown two receives match both messages, then the order is preserved.



Outlook Collective Communications

- Two kinds
  - Data movement (broadcast, scatter, gather, etc.)
  - Collective computation (min, max, sum, logical OR etc.)
- Advantages
  - More convenient
- Can be used as short cut for an ensemble of P2P operations
  - More efficient
- Encapsulate sophisticated algorithms
- Implementation can take advantage of the structure of a machine to optimize and increase parallelism in these operations

Broadcast

- **Broadcasts a message from the process with rank root to all other processes of the group.**
  - **buf = starting address of buffer**
  - **count = number of entries in buffer**
  - **datatype = data type of buffer**
  - **root = rank of broadcast root**
  - **comm = communicator**

```
int MPI_Bcast(    void*          buf,
                  int            count,
                  MPI_Datatype   datatype,
                  int            root,
                  MPI_Comm       comm)
```

Reduce

- **Combines the elements in the input buffer of each process using the operation op and returns the combined value in the output buffer of the process with rank root**
  - **sendbuf = address of send buffer**
  - **recvbuf = address of receive buffer**
  - **count = number of elements in send buffer**
  - **datatype = data type of elements of send buffer**
  - **op = reduce operation**
  - **root = rank of root process**
  - **comm = communicator**

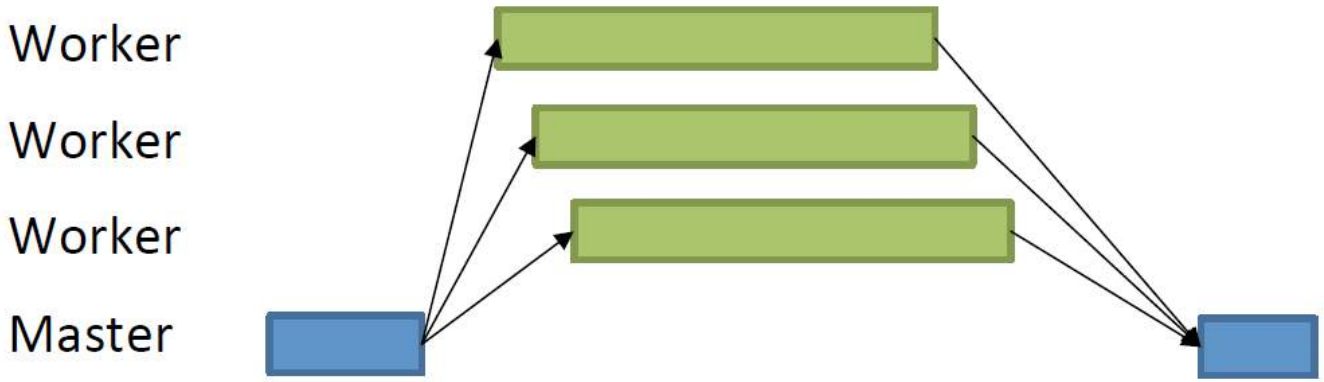
```
int MPI_Reduce(  void*          sendbuf,
                 void*          recvbuf,
                 int            count,
                 MPI_Datatype   datatype,
                 MPI_Op         op,
                 int            root,
                 MPI_Comm       comm)
```

Outline

- Introduction to the Lecture
- Distributed Parallel Computing I
  - Primer
  - Overview
  - Process Model and Language Bindings
  - Messages and Point-to-Point Communication
  - **Example**
- Quiz

Master Worker

- **Idea: self-scheduling algorithm**
  - **Master coordinates processing of tasks by providing input data to workers and collecting results**
- **Suitable if**
  - **Workers need not communicate with one another**
  - **Amount of work each worker must perform is difficult to predict**
- **Example: matrix-vector multiplication**



Matrix-Vector Multiplication

$$A \cdot \vec{b} = \vec{c}$$

Unit of work:

dot product of one row of matrix A with vector b

Master
- Broadcasts b to each worker
- Sends one row to each worker
- Loop
  - Receives dot product from whichever worker sends one
  - Sends next row to that worker
  - Termination of loop if all rows are handed out

Worker
- Receives broadcast value of b
- while-Loop
  - Receives row from A
  - Forms dot product
  - Returns result back to master

Matrix-Vector Multiplication: Common Part 1

```
#include <stdio.h>
#include <stdlib.h>
#include “mpi.h”
#define MAX_ROWS 1000 // upper limits
#define MAX_COLS 1000 // upper limits
#define MIN(a, b) ((a) > (b) ? (b) : (a))
#define DONE MAX_ROWS+1
int main(int argc, char **argv) {
  double** A;
  double* b, * c, * buffer;
  double ans;
  int myid, master, numprocs, i, j, numsent, sender, done, anstype, row, rows, cols;

  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);

  A = (double **)calloc(MAX_ROWS, sizeof(double *));
  for (i=0; i<MAX_ROWS; i++)
    A[i] = (double *)calloc(MAX_COLS, sizeof(double));
  b = (double *)calloc(MAX_COLS, sizeof(double));
  c = (double *)calloc(MAX_ROWS, sizeof(double));
  buffer = (double *)calloc(MAX_COLS, sizeof(double));
  ...
```

This is a minimal working examples, try it yourself!

Matrix-Vector Multiplication: Common Part 2

```
  …
  master = 0;
  rows = 2; // for debugging …
  cols = 2; // for debugging …
  if (myid == master)  {  /* master code: next slides */
  } else               {  /* worker code: next slides */
  }
  // result c vector finished: considering the test values, all element values should be 3.0
  for (i=0; i<MAX_ROWS; i++)
    free(A[i]);
  free(A);
  free(b);
  free(c);
  free(buffer);
  MPI_Finalize();
  return 0;
}
```

Matrix-Vector Multiplication: Master Part 1

Master

- **Broadcasts b to each worker**
- **Sends one row to each worker**
- **Loop**
  - **Receives dot product from whichever worker sends one**
  - **Sends next task to that worker**
  - **Termination if all tasks are handed out**

```
…
if (myid == master)   {
/* Initialize A and b (arbitrary) */
  A[0][0] = 1.0;
  A[0][1] = 2.0;
  A[1][0] = 1.0;
  A[1][1] = 2.0;
  b[0] = 1.0;
  b[1] = 1.0;

numsent = 0;
/* Send b to each worker process */
MPI_Bcast(b, cols, MPI_DOUBLE, master, MPI_COMM_WORLD);
/* Send a row to each worker process; tag with row number */
for (i = 0; i < MIN(numprocs - 1, rows); i++) {
  MPI_Send(&A[i][0], cols, MPI_DOUBLE, i+1, i, MPI_COMM_WORLD);
  numsent++;
}
…
```

Matrix-Vector Multiplication: Master Part 2

```
...
for (i = 0; i < rows; i++) {
  MPI_Recv(&ans, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,
                MPI_COMM_WORLD, &status);
  sender = status.MPI_SOURCE;
  /* row is tag value */
  anstype = status.MPI_TAG;
  c[anstype] = ans;
  /* send another row */
  if (numsent < rows) {
    MPI_Send(&A[numsent][0], cols,
                MPI_DOUBLE, sender,
                numsent, MPI_COMM_WORLD);
    numsent++;
  } else {
    /* Tell sender that there is no more work */
    MPI_Send(MPI_BOTTOM, 0, MPI_DOUBLE, sender, DONE, MPI_COMM_WORLD);
  }
}
} /* end of master specific part*/
```

Master

- **Broadcasts b to each worker**
- **Sends one row to each worker**
- **Loop**
  - **Receives dot product from whichever worker sends one**
  - **Sends next task to that worker**
  - **Termination if all tasks are handed out**

MPI_BOTTOM (indicates end of address space) as placeholder!

Matrix-Vector Multiplication: Worker Part

```
else    {     /* start of worker code*/
MPI_Bcast(b, cols, MPI_DOUBLE, master, MPI_COMM_WORLD);
/* Skip if more processes than work */
done = myid > rows;
while (!done) {
  MPI_Recv(buffer, cols, MPI_DOUBLE, master, MPI_ANY_TAG,
                 MPI_COMM_WORLD, &status);
  done = status.MPI_TAG == DONE;
  if (!done) {
    row = status.MPI_TAG;
    ans = 0.0;
    for (i = 0; i < cols; i++) {
      ans += buffer[i] * b[i];
    }
    MPI_Send(&ans, 1, MPI_DOUBLE, master, row, MPI_COMM_WORLD);
  }
}
} /* end of worker code */
```

Worker

- **Receives broadcast value of b**
- **while-Loop**
  - **Receives row from A**
  - **Forms dot product**
  - **Returns answer back to master**

Outline

- Introduction to the Lecture
- Distributed Parallel Computing I
  - Primer
  - Overview
  - Process Model and Language Bindings
  - Messages and Point-to-Point Communication
  - Examples
- **Quiz**

Quiz

- **Q1: How is it ensured that a specific message is received by a specific process?**
- **Q2: What is the first and last routine to be called in a MPI program?**
- **Q3: Is “MPI_Init” executed by one, several or all MPI processes?**
- **Q4: Name typical reduction operations?**
- **Q5: How can a point-to-point communication be made non-blocking? What potential advantage is there?**

Numerical Simulation and Scientific Computing II

Lecture 2:
Distributed Parallel Computing II

Josef Weinbub, Paul Manstetten, Heinz Pettermann, Asur Vijaya Kumar, Pavan Kumar, Jesús Carrete Montana, Francesco Zonta, Kevin Sturm

Institute for Microelectronics
TU Wien

nssc@iue.tuwien.ac.at

SS 2023

Quiz

Q1: How is it ensured that a specific message is received by a specific process?

→ *tag* and *dest* parameter

Q2: What is the first and last routine to be called in a MPI program?

→ *MPI_Init (…)* and *MPI_Finalize()*

Q3: Is "MPI_Init" executed by one, several or all MPI processes?

→ all

Q4: Name typical reduction operations?

→ sum, min, max, etc.

Q5: How can a point-to-point communication be made non-blocking? What potential advantage is there?

→ MPI_I…, potentially allows to overlap communication with computation

Outline

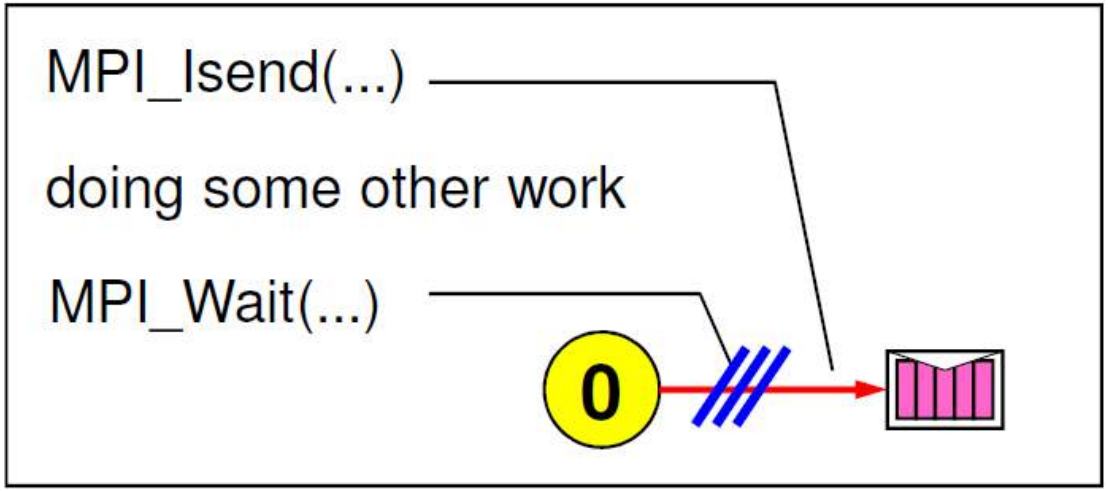
- Distributed Parallel Computing II
  - **Non-Blocking Communication**
  - Collective Communication
  - Derived Datatypes
  - Virtual Topologies
  - Hybrid MPI+OpenMP
- Quiz

Non-Blocking Communications

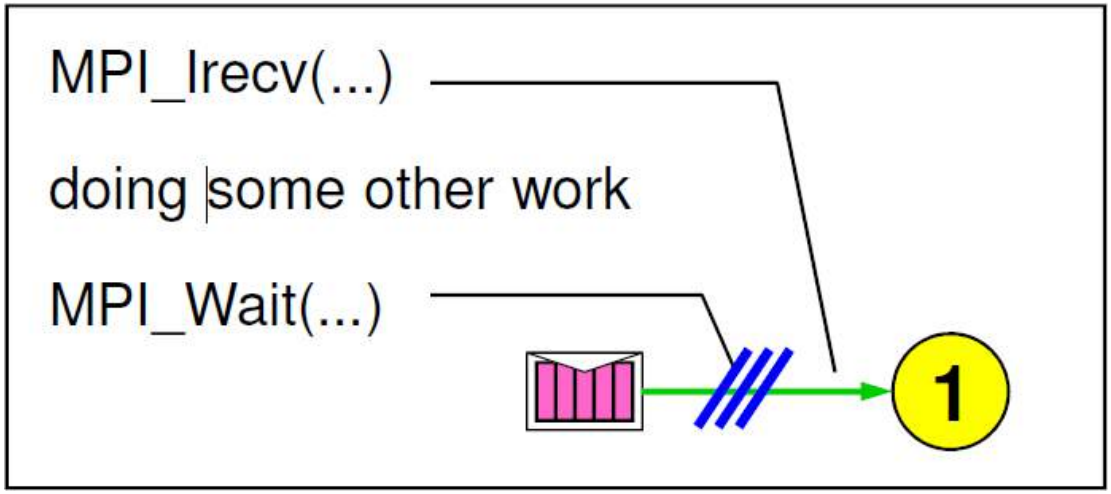
- **Separate communication into three phases:**
  - **Initiate nonblocking communication**
    - **returns Immediately**
    - **routine name starting with MPI_I…**
  - **Do some work (perhaps involving other communications?)**
    - **If cluster hardware doesn't have dedicated MPI communication logic, don't expect efficient overlapping of communication with computation: Communication requires significant computational resources!**
  - **Wait for non-blocking communication to complete**

Non-Blocking Examples

- **Non-blocking <u>send</u>**

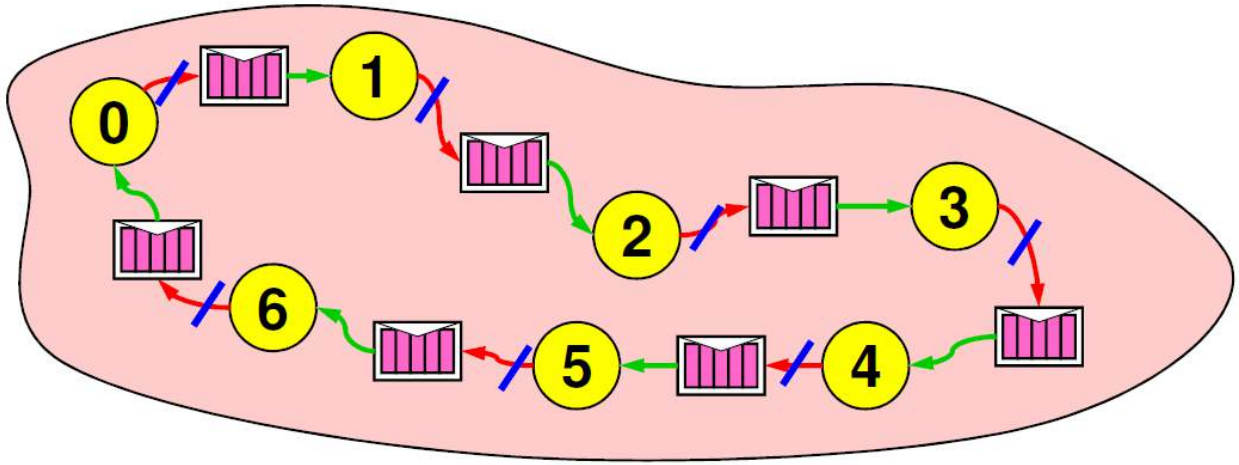


- **Non-blocking <u>receive</u>**



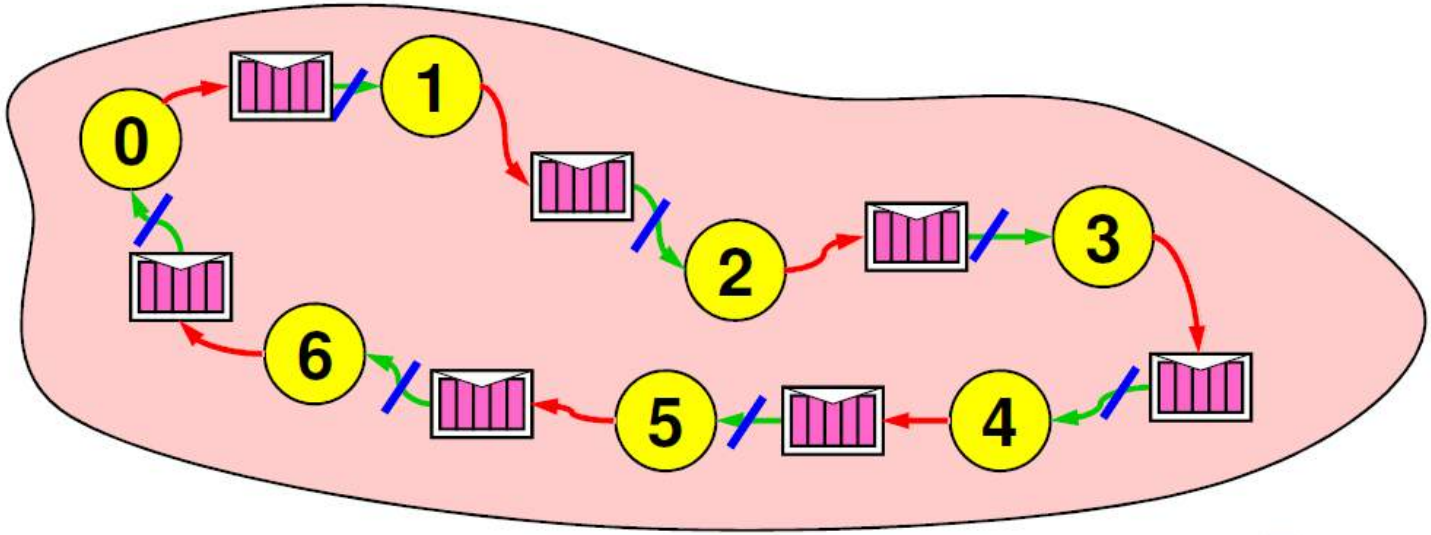
Non-Blocking Send

- **Initiate non-blocking send**
  - **Ring example: Initiate non-blocking send to the right neighbor**
- **Do some work**
  - **Ring example: Receiving the message from left neighbor**
- **Message transfer can be completed**
- **Wait for non-blocking send to complete**



Non-Blocking Receive

- **Initiate non-blocking receive**
  - **Ring example: Initiate non-blocking receive from left neighbor**
- **Do some work**
  - **Ring example: Sending the message to the right neighbor**
- **Message transfer can be completed**
- **Wait for non-blocking receive to complete**



Request Handles

- **Request handles**
  - **Are used for non-blocking communication**
  - **<u>Must</u> be stored in local variables: MPI_Request**
    - **is generated by a non-blocking communication routine**
    - **is used (and freed) in the MPI_WAIT routine**

Non-Blocking Synchronous Send

- **buf must not be modified between Issend and Wait**
- **"Issend + Wait directly after" is equivalent to blocking call (Ssend)**
- **status is not used in Issend, but in Wait (with send: nothing returned)**

```
MPI_Issend( buf, count, datatype, dest, tag, comm, [OUT] &request_handle);
```

```
MPI_Wait( [INOUT] &request_handle, &status);
```

Nonblocking Receive

- **<u>buf</u> must not be used between <u>Irecv</u> and <u>Wait</u>**

```
MPI_Irecv ( buf, count, datatype, source, tag, comm, [OUT] &request_handle);
```

```
MPI_Wait( [INOUT] &request_handle, &status);
```

Blocking and Non-Blocking

- Send and receive can be blocking or non-blocking.
- A blocking send can be used with a non-blocking receive, and vice-versa.
- Non-blocking sends can use any mode
  - standard – MPI_ISEND
  - synchronous – MPI_ISSEND
  - buffered – MPI_IBSEND
  - ready – MPI_IRSEND
- Synchronous mode affects completion, i.e. MPI_Wait / MPI_Test, not initiation, i.e., MPI_I...
- The non-blocking operation immediately followed by a matching wait is equivalent to the blocking operation.

Completion

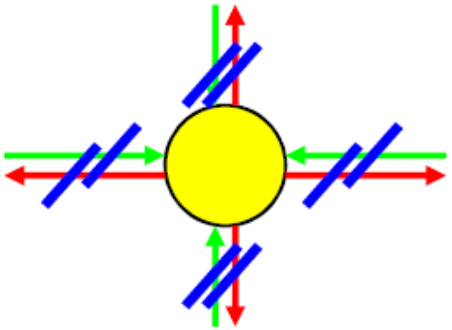
```
MPI_Wait( &request_handle, &status);
MPI_Test( &request_handle, &flag, &status);
```

- **You need to**
  - **WAIT or**
  - **Loop with TEST until request is completed, i.e., flag == 1**

Multiple Non-Blocking Communications

You have several request handles:

- **Wait or test for completion of one message**
  - **MPI_Waitany / MPI_Testany**
- **Wait or test for completion of all messages**
  - **MPI_Waitall / MPI_Testall *)**
- **Wait or test for completion of as many messages as possible (i.e., at least one)**
  - **MPI_Waitsome / MPI_Testsome *)**



***) Each status contains an additional error field.**

This field is only used if MPI_ERR_IN_STATUS is returned (also valid for send operations).

Send-Receive in One Routine

- **MPI_Sendrecv & MPI_Sendrecv_replace**
  - **Combines the triple “MPI_Irecv + Send + Wait” into one routine**

Performance Options

Which is the fastest neighbor communication?

- **MPI_Irecv + MPI_Send**
- **MPI_Irecv + MPI_Isend**
- **MPI_Isend + MPI_Recv**
- **MPI_Isend + MPI_Irecv**
- **MPI_Sendrecv**
- **MPI_Neighbor_alltoall**

No answer by the MPI standard, because:

MPI targets portable and efficient message-passing programming but efficiency of MPI application-programming is not portable!

Example Non-Blocking Communication

- **Integration with MPI non-blocking communications**
  - **Example source:** http://www.bu.edu/tech/support/research/training-consulting/online-tutorials/mpi/example1-2/example1_3/
  - **Background on numerical integration:** http://www.bu.edu/tech/support/research/training-consulting/online-tutorials/mpi/example1-2/
- **Until a matching receive has signaled that it is ready to receive, a blocking send will continue to wait.**

```
void other_work(int myid) {
  printf("more work on process %dn", myid);
}
float integral(float ai, float h, int n){
  int j;
  float aij, integ;
  integ = 0.0;                /* initialize */
  for (j=0;j<j++) {          /* sum integrals */
    aij = ai + (j+0.5)*h;      /* mid-point */
    integ += cos(aij)*h;
  }
  return integ;
}
```

Support Functions

Example Non-Blocking Communication

Common Part

```
int n, p, myid, tag, master, proc, ierr;
float h, integral_sum, a, b, ai, pi, my_int;
MPI_Comm comm;
MPI_Request request;
MPI_Status status;
comm = MPI_COMM_WORLD;
ierr = MPI_Init(&argc,&argv);          /* starts MPI */
MPI_Comm_rank(comm, &myid);            /* get current process id */
MPI_Comm_size(comm, &p);               /* get number of processes */
master = 0;
pi = acos(-1.0);  /* = 3.14159... */
a = 0.;           * lower limit of integration */
b = pi*1./2.;     /* upper limit of integration */
n = 500;          /* number of increment within each process */
tag = 123;        /* set the tag to identify this particular job */
h = (b-a)/n/p;    /* length of increment */
ai = a + myid*n*h;  /* lower limit of integration for partition myid */
my_int = integral(ai, h, n);    /* 0<=myid<=p-1 */
printf("Process %d has the partial result of %fn", myid, my_int);
```

Example Non-Blocking Communication

Master Part

```
if(myid == master) {
  integral_sum = my_int;
  for (proc=1;proc<p;proc++) {
    MPI_Recv(
            &my_int, 1, MPI_FLOAT,    /* triplet of buffer, size, data type */
            MPI_ANY_SOURCE,       /* message source */
            MPI_ANY_TAG,          /* message tag */
            comm, &status);       /* status identifies source, tag */
    integral_sum += my_int;
  }
  printf("The Integral =%fn",integral_sum); /* sum of my_int */
}
```

Example Non-Blocking Communication

Worker Part

```
else {
  MPI_Isend(      /* non-blocking send */
          &my_int, 1, MPI_FLOAT,      /* triplet of buffer, size, data type */
          master,
          tag,
          comm, &request);      /* send my_int to master */
  other_work(myid);
  MPI_Wait(&request, &status);    /* block until Isend is done */
 }
 MPI_Finalize();                  /* let MPI finish up ... */
}
```

Outline

- Distributed Parallel Computing II
  - Non-Blocking Communication
  - **Collective Communication**
  - Derived Datatypes
  - Virtual Topologies
  - Hybrid MPI+OpenMP
- Quiz

Collective Communication

- Communications involving a group of processes.
- Called by all processes in a communicator.
- Examples:
  - Barrier synchronization.
  - Broadcast, scatter, gather.
  - Global sum, global maximum, etc.
  - Neighbor communication in a virtual grid

Characteristics of Collective Communication

- Collective action over a communicator.
- All processes of the communicator must communicate, i.e., must call the collective routine.
- Synchronization may or may not occur, therefore all processes must be able to start the collective routine.
- On a given communicator, the n-th collective call must match on all processes of the communicator.
- In MPI-1.0 – MPI-2.2, all collective operations are blocking. Non-blocking versions since MPI-3.0.
  - Not covered in this lecture
- No tags.
- Receive buffers must have exactly the same size as send buffers.

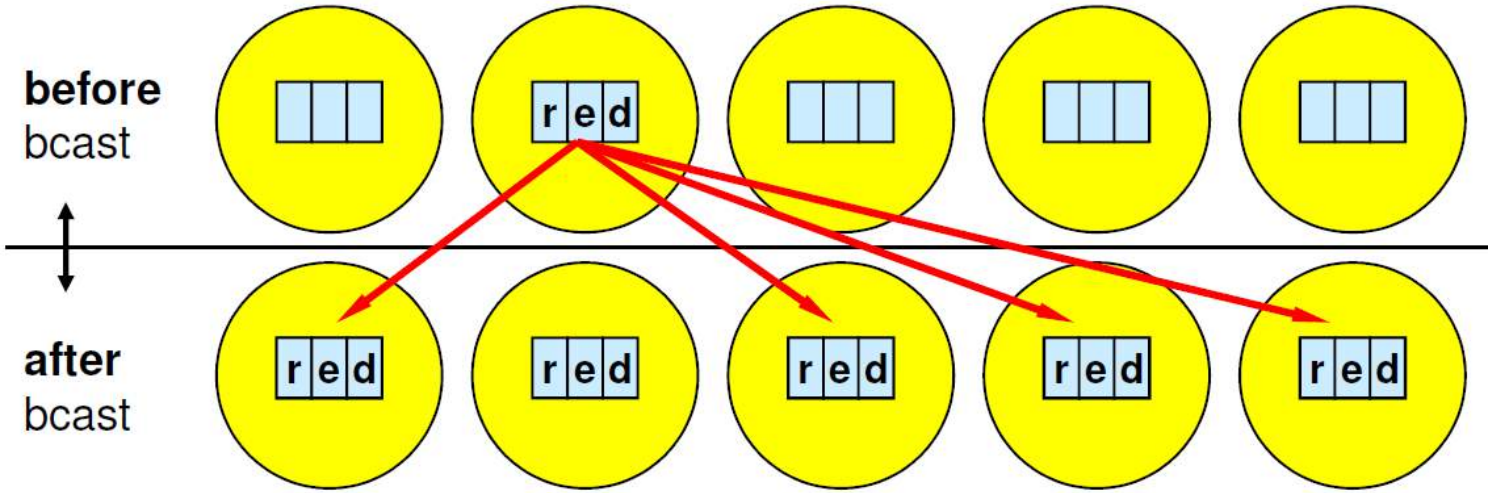
Barrier Synchronization

MPI_Barrier is normally never needed:

- **all synchronization is done automatically by the data communication:**
  a process cannot continue before it has the data that it needs.
- **if used for debugging:**
  please guarantee, that it is removed in production.
- **for profiling: to separate time measurements**

Broadcast

```
int MPI_Bcast(void *buf, int count, MPI_Datatype datatype,
              int root, MPI_Comm comm)
```

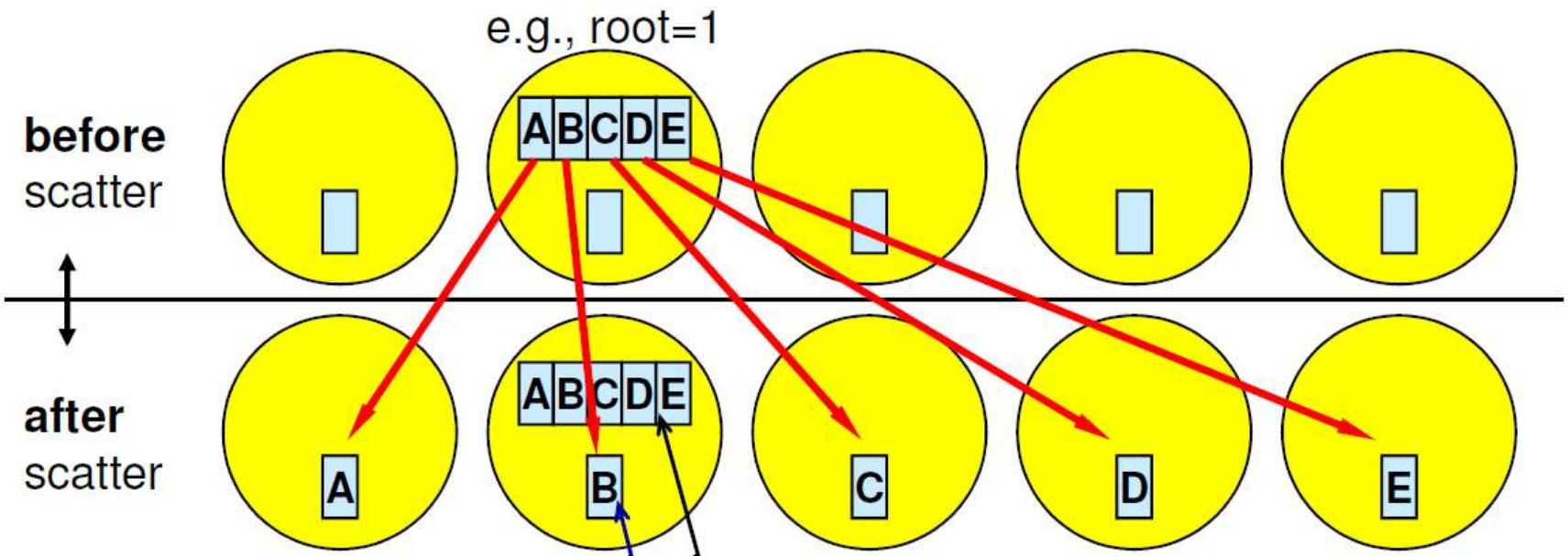


- **E.g. <u>root</u>=1: all processes must use same ”root”**

Example

```
std::vector<char> buf(3);
if(myrank == 1) { buf[0]='r'; buf[1]='e'; buf[2]='d'; }
MPI_Bcast(buf.data(), buf.size(), MPI_CHAR, 1, MPI_COMM_WORLD);
```

Scatter




```
int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype,
                void *recvbuf, int recvcount, MPI_Datatype recvtype,
                int root, MPI_Comm comm)
```

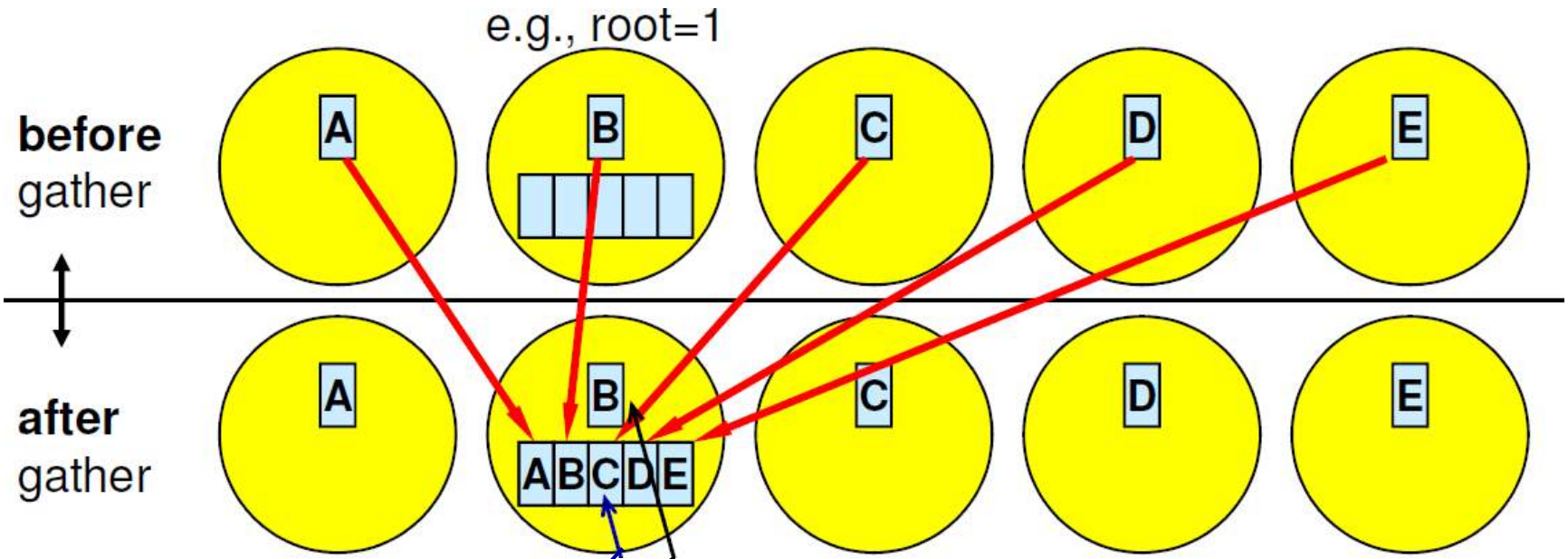
Example

```
std::vector<char> sbuf(5); char rbuf;
if(myrank == 1) sbuf[0]='A'; sbuf[1]='B'; // etc.
MPI_Scatter(sbuf.data(), 1, MPI_CHAR,
            &rbuf, 1, MPI_CHAR,
            1, MPI_COMM_WORLD)
```

If data cannot be equally distributed to all processes, use:
MPI_Scatterv: allows for variable send/recvcounts.

Sum of transmitted data size must match:
sendcount x sendtype =
recvcount x recvtype
In principle: allowed to mix data types!
But typically: counts and types the same.

Gather




```
int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype sendtype,
               void *recvbuf, int recvcount, MPI_Datatype recvtype,
               int root, MPI_Comm comm)
```

Example

```
if(myrank==0) char sbuf='A';   // etc…'B', 'C', 'D', 'E'
std::vector<char> rbuf(5);
MPI_Gather(&sbuf, 1, MPI_CHAR,
           rbuf.data(), 1, MPI_CHAR,
           1, MPI_COMM_WORLD)
```

See notes for Scatter, same here.

Global Reduction Operations

- To perform a global reduce operation across
- d0 o d1 o d2 o d3 o … o ds-2 o ds-1
  - di = data in process rank i
    - single variable, or
    - vector
  - o = associative operation
  - Example:
    - global sum or product
    - global maximum or minimum
    - global user-defined operation

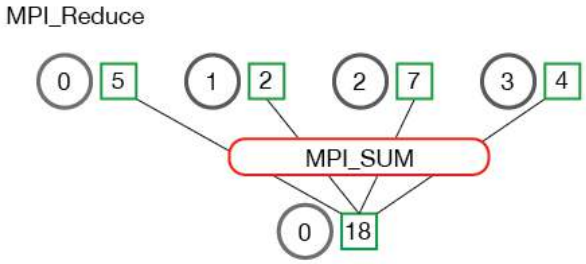
Predefined Reduction Operation Handles

Predefined operation handle	Function
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAXLOC	Maximum and location of the maximum
MPI_MINLOC	Minimum and location of the minimum

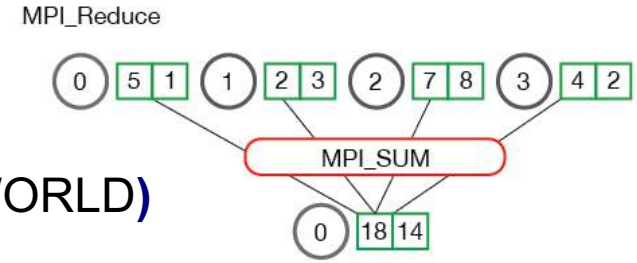
Reduce

```
int MPI_Reduce(void *inbuf, void *result, int count, MPI_Datatype datatype,
               MPI_Op op, int root, MPI_Comm comm)
```

```
int inbuf = 5; // 2, 7, 4 for other ranks
int result;
MPI_Reduce(&inbuf, &result, 1, MPI_INT,
           MPI_SUM, 0, MPI_COMM_WORLD)
```




```
std::vector<int> inbuf {5,1}; // {2,3} … for other ranks
std::vector<int> result (2);
MPI_Reduce(inbuf.data(), result.data(), result.size(),
           MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD)
```



Example Collective Communication

- **Integration with MPI collective communications**
  - **Example source: http://www.bu.edu/tech/support/research/training-consulting/online-tutorials/mpi/example1-2/example1_4/**
  - **Adaption of previous example based on non-blocking communication**

```
float integral(float ai, float h, int n){
  int j;
  float aij, integ;
  integ = 0.0;                /* initialize */
  for (j=0;j<j++) {          /* sum integrals */
    aij = ai + (j+0.5)*h;      /* mid-point */
    integ += cos(aij)*h;
  }
  return integ;
}
```

Support Function

Example Collective Communication

First Part

```
int n, p, myid, tag, proc, ierr, i;
float h, integral_sum, a, b, ai, pi, my_int, buf[50];
int master = 0;  /* processor performing total sum */
MPI_Comm comm;
comm = MPI_COMM_WORLD;
ierr = MPI_Init(&argc,&argv);          /* starts MPI */
MPI_Comm_rank(comm, &myid);            /* get current process id */
MPI_Comm_size(comm, &p);               /* get number of processes */
pi = acos(-1.0);  /* = 3.14159... */
a = 0.;          /* lower limit of integration */
b = pi*1./2.;     /* upper limit of integration */
n = 500;          /* number of increment within each process */
tag = 123;        /* set the tag to identify this particular job */
h = (b-a)/n/p;    /* length of increment */
ai = a + myid*n*h;  /* lower limit of integration for partition myid */
my_int = integral(ai, h, n);   /* 0<=myid<=p-1 */
printf("Process %d has the partial sum of %fn", myid,my_int);
```

buf-size >= p
(see next slide)

Example Collective Communication

Second Part

```
MPI_Gather(      /* collects my_int from all processes to master */
    &my_int, 1, MPI_FLOAT,      /* send buffer, size, data type */
    &buf[0], 1, MPI_FLOAT,      /* receive buffer, size, data type */
    master, comm);

if(myid == master) {
  integral_sum = 0.0;
  for (i=0; i<p; i++) {
    integral_sum += buf[i];
  }
  printf("The Integral =%fn",integral_sum);
}

MPI_Finalize();
```

buf must fit all ranks!

Is there another collective operation which could be used here?

Yes, MPI_Reduce with MPI_SUM

Outline

- Distributed Parallel Computing II
  - Non-Blocking Communication
  - Collective Communication
  - **Derived Datatypes**
  - Virtual Topologies
  - Hybrid MPI+OpenMP
- Quiz

Derived Datatypes

- Description of the memory layout of the buffer
  - for sending
  - for receiving
- Basic types
- Derived types
  - vectors
  - structs
  - others

Data Layout and the Describing Datatype Handle

```
struct buff_layout
  { int    i_val[3];
    double d_val[5];
  } buffer;
```

Compiler

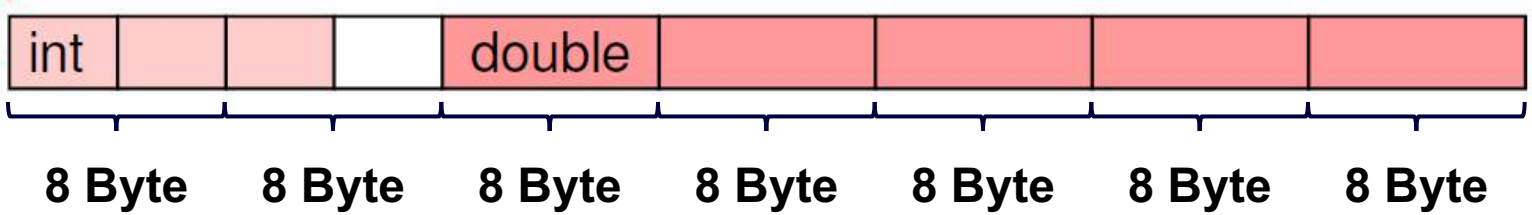
```
array_of_types[0]=MPI_INT;
array_of_blocklengths[0]=3;
array_of_displacements[0]=
array_of_types[1]=MPI_DOUBLE;
array_of_blocklengths[1]=5;
array_of_displacements[1]=
MPI_Type_create_struct(2, array_of_blocklengths,
    array_of_displacements, array_of_types,
                    &buff_datatype);
MPI_Type_commit(&buff_datatype);
```

?

```
MPI_Send(&buffer, 1, buff_datatype, ...)
```

&buffer = the start address of the data

the datatype handle describes the data layout



Compute Displacement

```
array_of_types[0]=MPI_INT;
array_of_blocklengths[0]=3;
array_of_displacements[0]=
array_of_types[1]=MPI_DOUBLE;
array_of_blocklengths[1]=5;
array_of_displacements[1]=
MPI_Type_create_struct(2, array_of_blocklengths,
    array_of_displacements, array_of_types,
                    &buff_datatype);
MPI_Type_commit(&buff_datatype);
```

In principle:
array_of_displacements[i] = address(block_i) – address(block_0);

Use:
int MPI_Get_address(void* location, MPI_Aint *address)

Continuing the example:

```
int address_base, address_0, address_1;
MPI_Get_address(&buffer, &address_base);
MPI_Get_address(&buffer.i_val[0], &address_0);
MPI_Get_address(&buffer.d_val[0], &address_1);
array_of_displacements[0] = address_0 – address_base;
array_of_displacements[1] = address_1 – address_base;
```

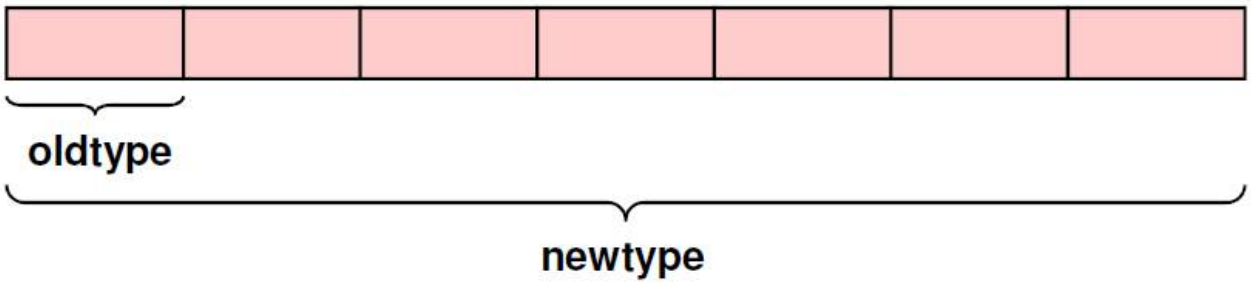
```
struct buff_layout
  { int    i_val[3];
    double d_val[5];
  } buffer;
```

Derived Datatype: Another Example

```
typedef struct {
  char      a;
  int       b;
  double    c;} mystruct;
mystruct mydata;
int baseaddr, addr0, addr1, addr2;
MPI_Get_address ( &mydata,   &baseaddr);
MPI_Get_address ( &mydata.a, &addr0);
MPI_Get_address ( &mydata.b, &addr1);
MPI_Get_address ( &mydata.c, &addr2);
displ[0] = addr0 – baseaddr;
displ[1] = addr1 – baseaddr;
displ[2] = addr2 – baseaddr;
dtype[0] = MPI_CHAR;
blength[0] = 1;
dtype[1] = MPI_INT;
blength[1] = 1;
dtype[2] = MPI_DOUBLE;
blength[2] = 1;
MPI_Type_struct ( 3, blength, displ, dtype, &newtype );
MPI_Type_commit ( &newtype );
```

Contiguous Data

- **The simplest derived datatype**
- **Consists of a number of contiguous items of the same datatype**




```
int MPI_Type_contiguous(int count, MPI_Datatype oldtype,
                        MPI_Datatype *newtype)
```

Example

```
int myvec[4];
MPI_Type_contiguous ( 4, MPI_INT, &mybrandnewdatatype);
MPI_Type_commit ( &mybrandnewdatatype );
MPI_Send ( myvec, 1, mybrandnewdatatype, ... );
```

Committing ad Freeing a Datatype

- **Before a dataytype handle is used in message passing communication, it needs to be committed with MPI_TYPE_COMMIT.**
- **This need be done only once (by each MPI process).**
- **If type no longer required, one may call MPI_TYPE_FREE() to free a datatype and its internal resources.**

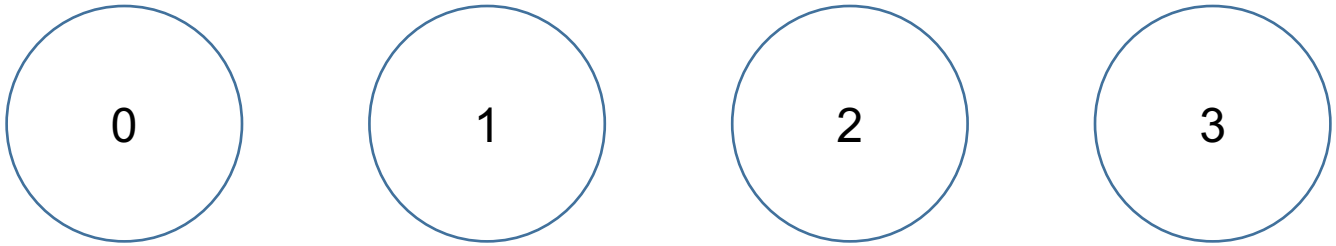
```
int MPI_Type_commit(MPI_Datatype *datatype);
int MPI_Type_free     (MPI_Datatype *datatype);
```

Outline

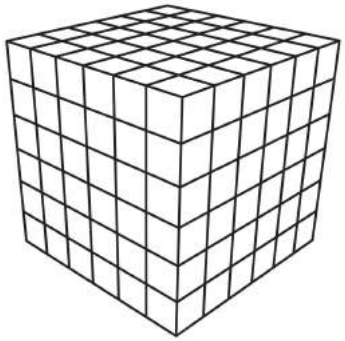
- Distributed Parallel Computing II
  - Non-Blocking Communication
  - Collective Communication
  - Derived Datatypes
  - **Virtual Topologies**
  - Hybrid MPI+OpenMP
- Quiz

Virtual Topologies

- Convenient process naming.
- Naming scheme to fit the communication pattern.
- Simplifies writing of code.
- Can allow MPI to optimize communications.
- Normal topology:



- Now consider that you work with this:

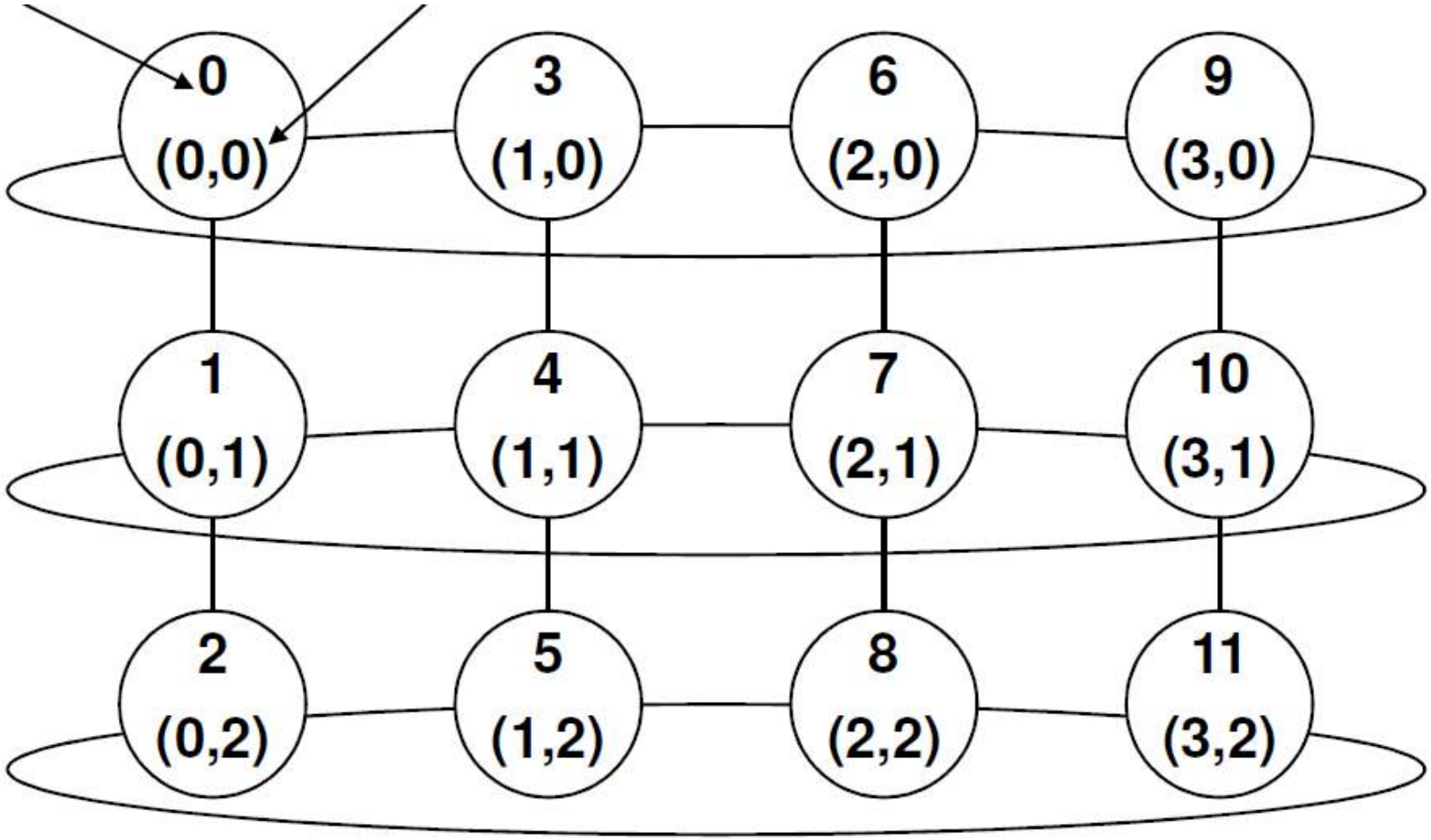


How to Use a Virtual Topology

- **Creating a topology produces a new communicator.**
- **MPI provides mapping functions:**
  - **to compute process ranks, based on the topology naming scheme,**
  - **and vice versa.**

Example – A Two-Dimensional Cylinder

- **<u>Ranks</u> and <u>Cartesian process</u> coordinates**



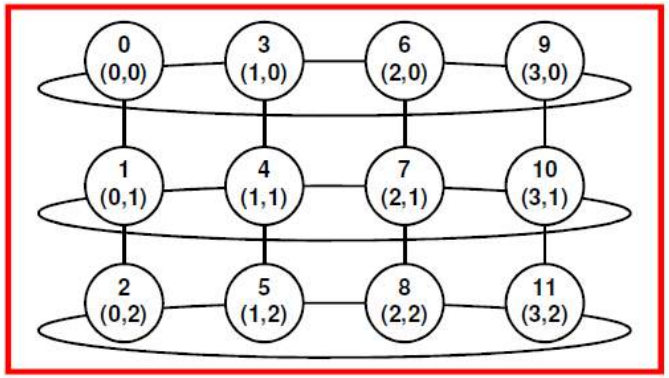
Topology Types

- **Cartesian Topologies**
  - **each process is connected to its neighbor in a virtual grid,**
  - **boundaries can be cyclic, or not,**
  - **processes are identified by Cartesian coordinates,**
  - **of course, communication between any two processes is still allowed.**
- **Graph Topologies**
  - **general graphs,**
  - **two interfaces:**
    - **MPI_GRAPH_CREATE (since MPI-1)**
    - **MPI_DIST_GRAPH_CREATE_ADJACENT &**
    - **MPI_DIST_GRAPH_CREATE (new scalable interface since MPI-2.2)**
  - **not covered here.**

Creating a Cartesian Virtual Topology

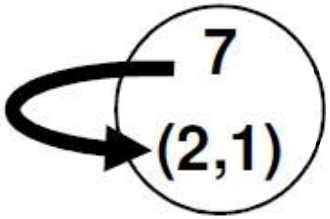
```
int MPI_Cart_create(MPI_Comm comm_old, int ndims,
                    int *dims, int *periods, int reorder,
                    MPI_Comm *comm_cart)
```

- **Comm_old = MPI_COMM_WORLD**
- **ndims = 2**
- **Dims = (4, 3)**
- **Periods = (1, 0)**
- **Reorder = 0 or 1**
  → Reorder: Task MPI backend to optimally assign MPI ranks to specific Cartesian coordinates considering communication ramifications!
  → But then ranks in comm_cart differ from comm_old!



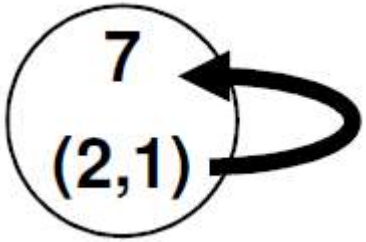
Cartesian Mapping Functions

- **Mapping ranks to process grid coordinates**



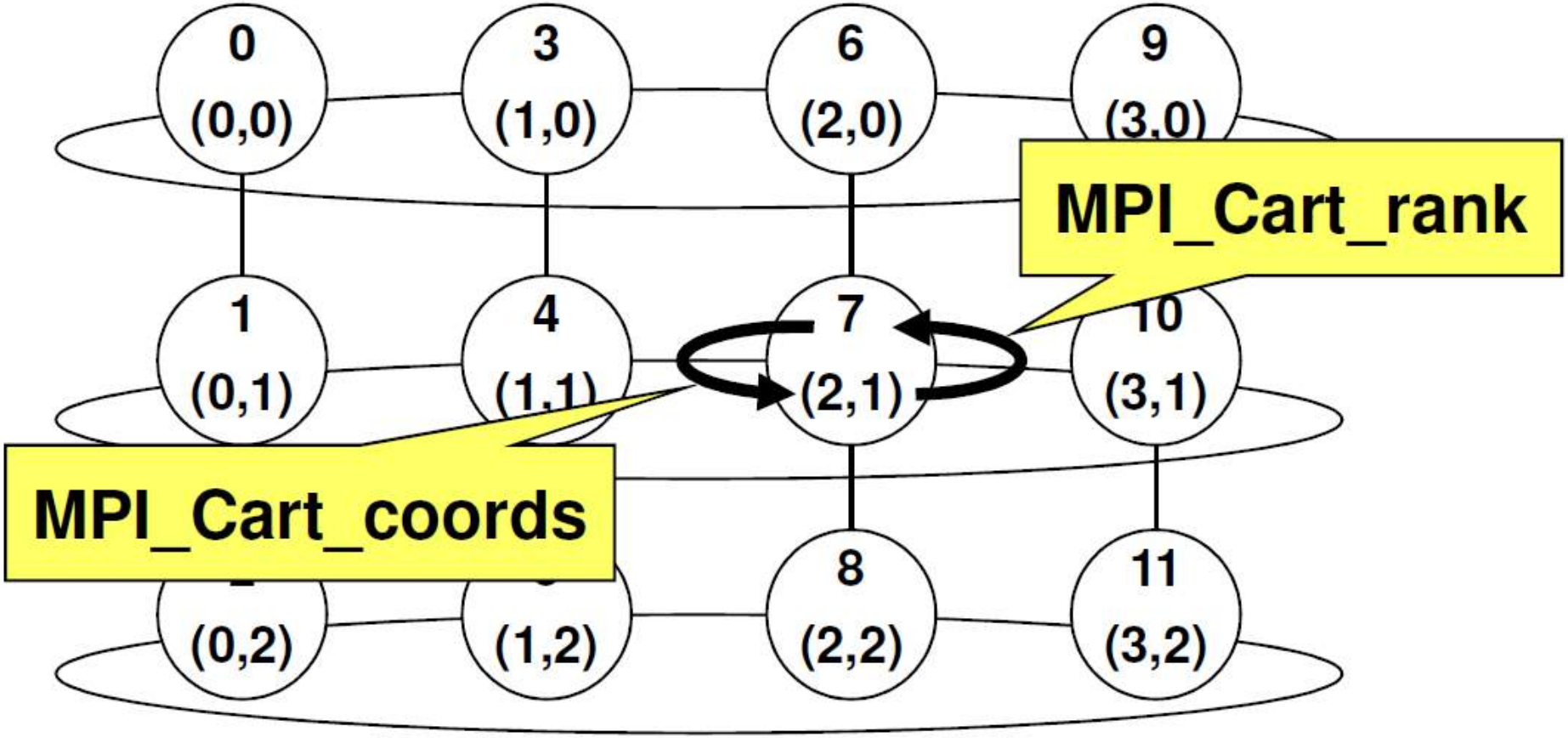

```
int MPI_Cart_coords(MPI_Comm comm_cart, int rank, int maxdims, int *coords)
```

- **Mapping process grid coordinates to ranks**




```
int MPI_Cart_rank(MPI_Comm comm_cart, int *coords, int *rank)
```

Cartesian Mapping Functions

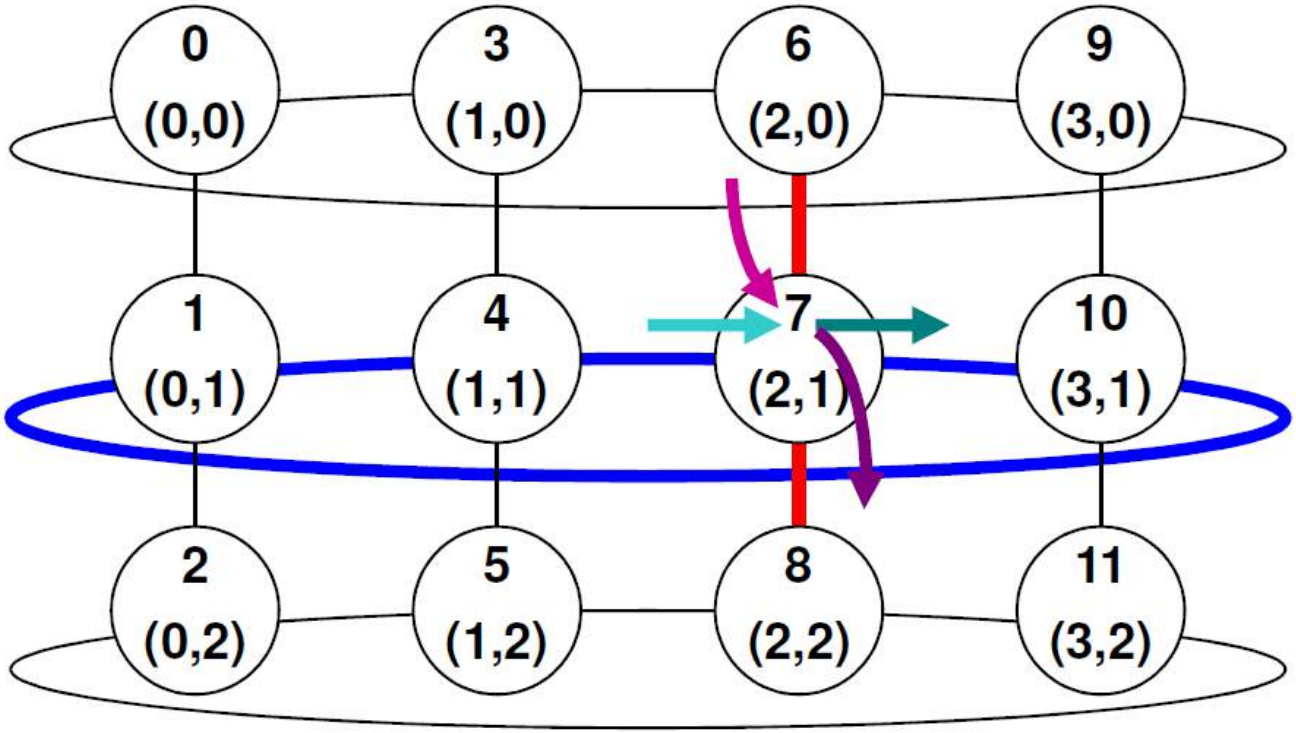


Cartesian Mapping Functions

- **Compute ranks of neighboring processes**
- **Returns MPI_PROC_NULL if there is no neighbor.**
- **In general: MPI_PROC_NULL can be used as source or destination rank in each communication. → This communication will be a no-operation!**

```
int MPI_Cart_shift(MPI_Comm comm_cart, int direction, int disp,
                   int *rank_source, int *rank_dest)
```

Cartesian Mapping Functions



invisible input argument: **my_rank** in cart

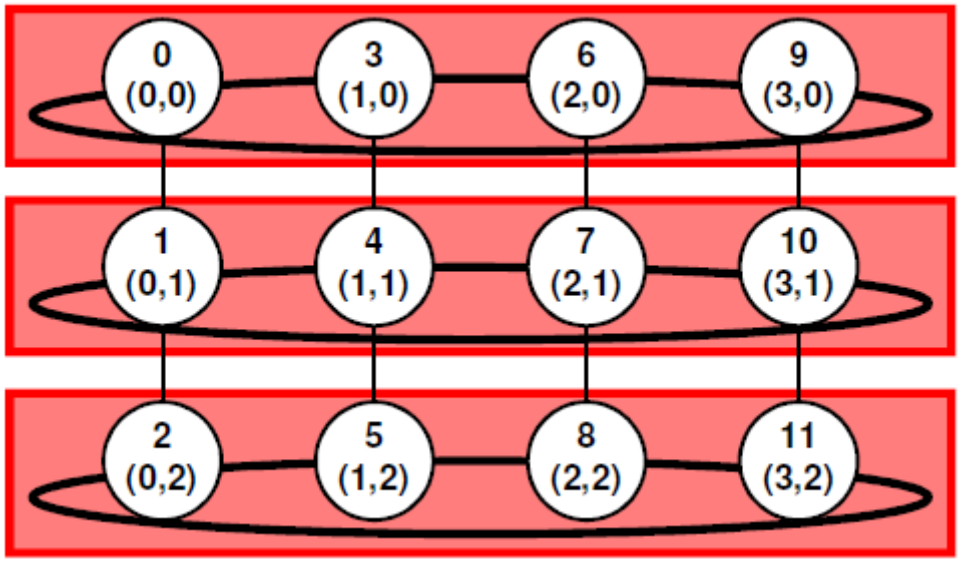
MPI_Cart_shift(cart, direction, displace, *rank_source*, *rank_dest*, *ierror*)

example on	0 or	+1	4	10
process rank=**7**	1	+1	6	8

Cartesian Partitioning

- **Cut a grid up into slices.**
- **A new communicator is produced for each slice.**
- **Each slice can then perform its own collective communications.**

```
int MPI_Cart_sub( MPI_Comm comm_cart, int *remain_dims,
                  MPI_Comm *comm_slice)
```

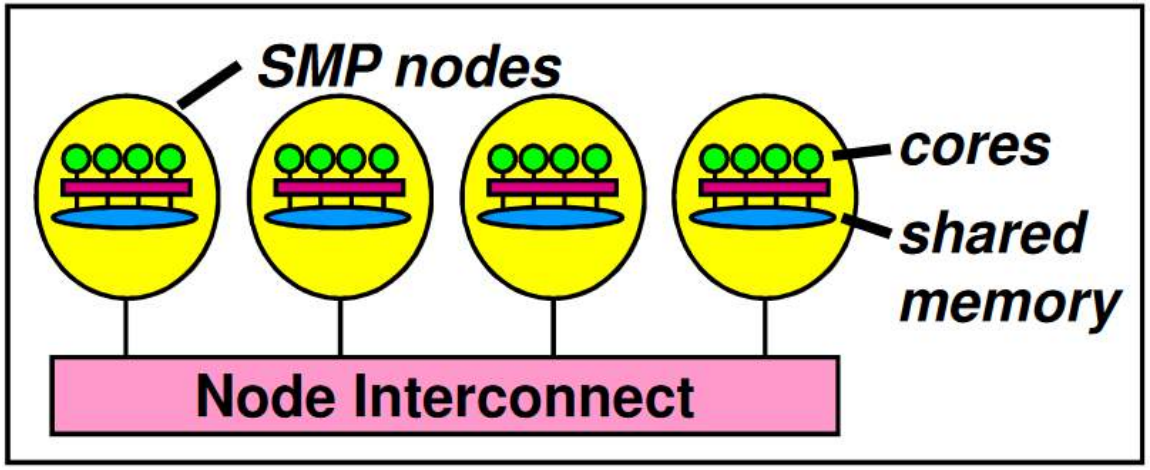


Outline

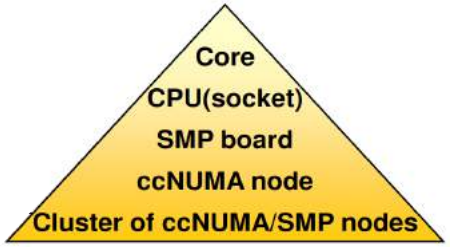
- Distributed Parallel Computing II
  - Non-Blocking Communication
  - Collective Communication
  - Derived Datatypes
  - Virtual Topologies
  - **Hybrid MPI+OpenMP**
- Quiz

Hybrid Parallel Programming

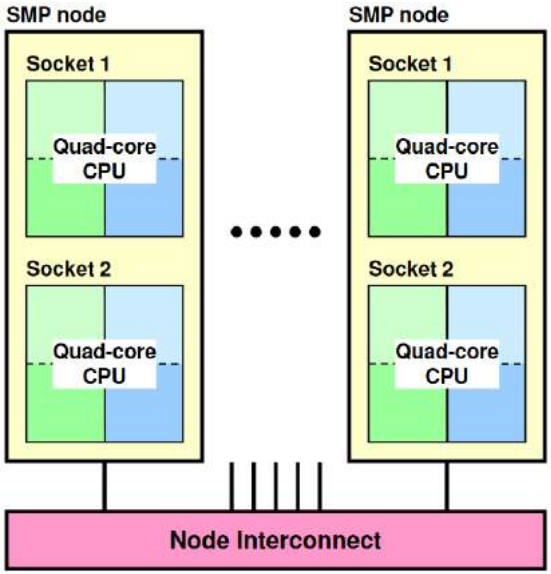
- **Hybrid parallel programming:**
  Mix different parallel programming approaches to efficiently use available computing platforms (heterogeneous computing resources)
- **Typical example:**
  Programming of clusters of shared memory nodes



Hybrid Parallel Programming



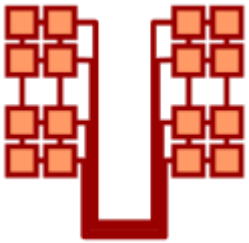
- **Hierarchical system layout**
- **Hybrid programming seems *natural***
  - **MPI between the nodes**
  - **Shared memory programming inside of each SMP node**
    - **<u>OpenMP</u>**
    - **MPI-3 shared-memory programming: not covered here**
    - **Accelerator programming (CUDA, OpenACC, etc): not covered here**
    - **And others.**



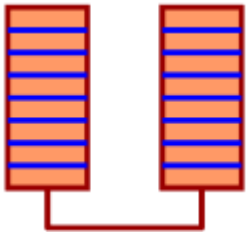
Hybrid Parallel Programming

- **Which programming model is best?**

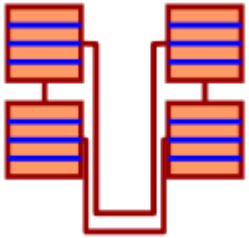
- **"Pure" MPI?**
  (a.k.a. 1 MPI process per CPU core, no shared-memory at all)



- **"Fully hybrid"?**
  (a.k.a. 1 MPI process per node, shared-memory within nodes)

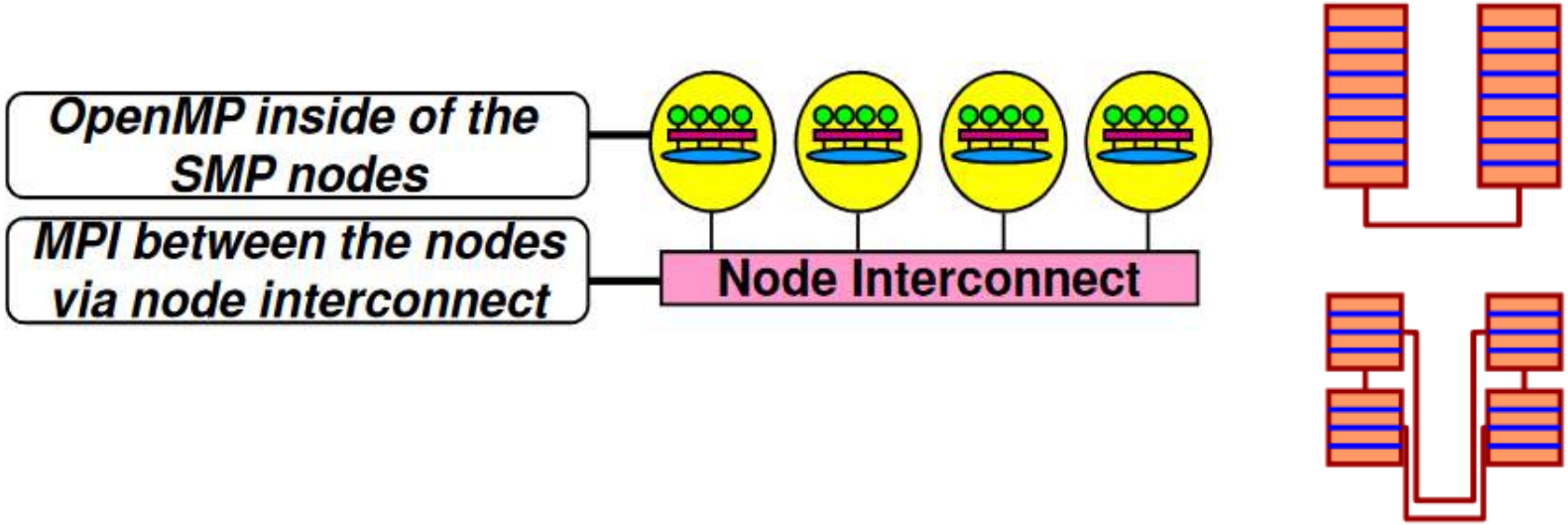


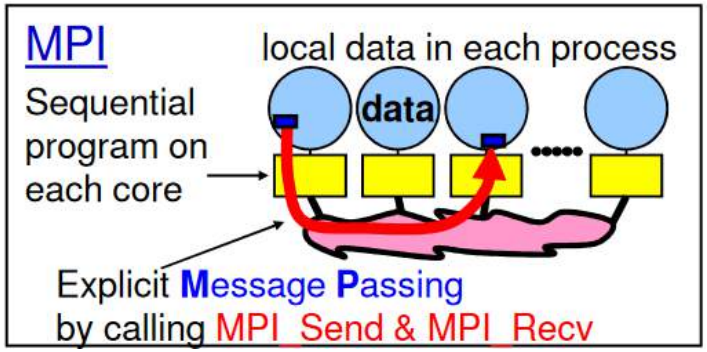
- **"Mixed hybrid"?**
  (a.k.a. 1 MPI process per, e.g., CPU, shared-memory within the CPUs; other combinations possible, e.g., (cc)NUMA specific)

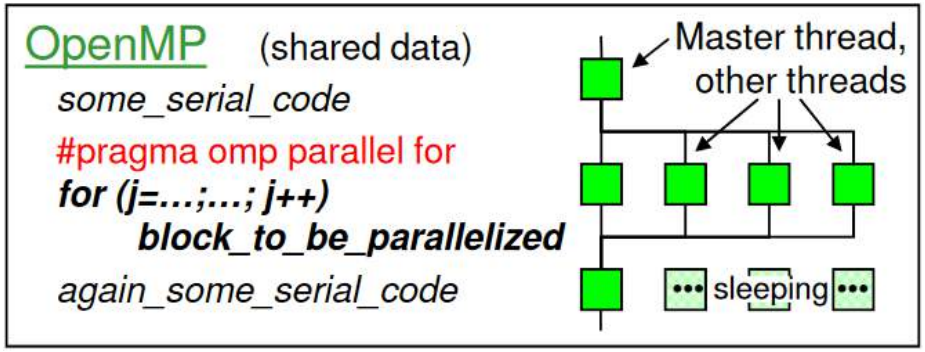


Problem specific …

Typical Case: MPI+OpenMP







Hybrid MPI+OpenMP *Masteronly*

- **<u>Master</u>only:**
  MPI only outside parallel OpenMP regions → only the <u>master</u> thread issues MPI calls

```
#pragma omp parallel
  numerical code
/*end omp parallel */

/* on master thread only */
MPI_Send (original data to halo areas in other SMP nodes)
MPI_Recv (halo data from the neighbors)
```

Hybrid MPI+OpenMP *Masteronly*

- **Major Advantages**
  - **No message passing inside of the SMP nodes**
  - **No topology problem**
- **Major Disadvantages**
  - **All other threads are sleeping while master thread communicates**
  - **All communicated data passes through the cache where the master thread is executing**
  - **Strictly speaking: MPI-library must have been compiled with threading support (configure OpenMPI library build with: `--enable-mpi-threads`)**
    and must have been initialized with threading support
    → But masteronly approach will likely work without as well.

MPI Threading Support Handles

- **MPI_THREAD_SINGLE: Only one thread will execute (→ similar to calling MPI_Init ())**
- **MPI_THREAD_FUNNELED: Only master thread will make MPI-calls**
- **MPI_THREAD_SERIALIZED: Multiple threads may make MPI-calls, but only one at a time (not covered here)**
- **MPI_THREAD_MULTIPLE: Multiple threads may call MPI, with no restrictions (not covered here)**

```
int MPI_Init_thread( int * argc, char ** argv[],
                     int thread_level_required,
                     int * thead_level_provided);
```

Outline

- Distributed Parallel Computing II
  - Non-Blocking Communication
  - Collective Communication
  - Derived Datatypes
  - Virtual Topologies
  - Hybrid MPI+OpenMP
- **Quiz**

Quiz

- **Q1: How many Bytes would be required to store an MPI derived datatype based on a struct of two characters and one double?**
- **Q2: Can a non-blocking receive be combined with a blocking send?**
- **Q3: Give an example setup for defining a Cartesian MPI communication topology for a three-dimensional setup if you could use up to 1000 MPI processes (make assumptions for all other properties).**
- **Q4: What is the order of the forward/backward Euler method?**
- **Q5: Are Runge-Kutta methods single-step or multi-step methods?**

Numerical Simulation and Scientific Computing II

Lecture 3:
ODE Methods, Finite Volume Method

Josef Weinbub, Paul Manstetten, Heinz Pettermann, Asur Vijaya Kumar, Pavan Kumar, Jesús Carrete Montana, Francesco Zonta, Kevin Sturm

Institute for Microelectronics
TU Wien

nssc@iue.tuwien.ac.at

SS 2023

Questions

ODE Methods

- Q1: What is the order of the forward/backward Euler method?
- Q2: Are the Runge-Kutta methods single-step or multistep methods?
- Q3: How can methods for first order ODEs be applied to higher order ODEs?

Finite volume method

- Q4: Why are finite volume schemes ‘conservative’?
- Q5: What are advantages of the finite volume method over the finite difference method?

Outline

- ODE Methods
  - Example: first order ODEs
  - Single-step methods
  - Multistep methods
  - Embedded methods
  - Higher order ODEs/systems of ODEs
- Finite volume method
  - Concept
  - Domain decomposition
  - Boundary/interface conditions

First order ODE

- First order ODE + initial value

$$\frac{dy}{dt} = f\big(t, y(t)\big)$$
$$y(t_a) = t_a$$

- Simple example (with solution)

$$y' = -y, \qquad y = e^{-t+C}$$
$$y(t_a = 0) = e^1, \qquad y = e^{-t+1}$$

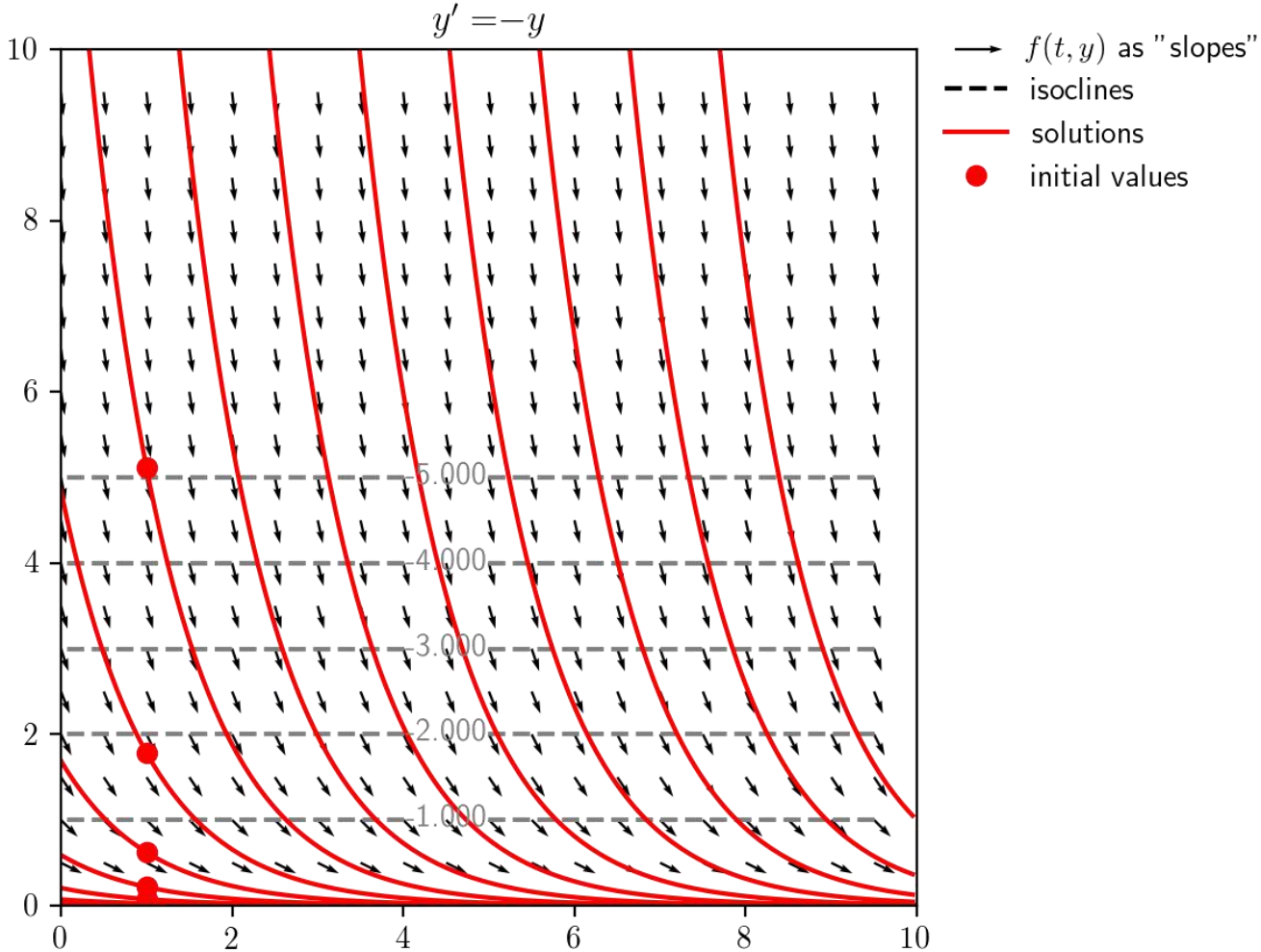
- Methods for numerical solutions from t_a to t_b (or only at t_b)
  - Large collection of methods for problems of this type
  - For “not so nice” f, numerical methods are the only choice
  - “Not so nice” f stem from arbitrary dependencies on t and y

Examples

- Simple example

$$y' = -y, \qquad y = e^{-t+C}$$

- Visualization of solutions in the t, y plane

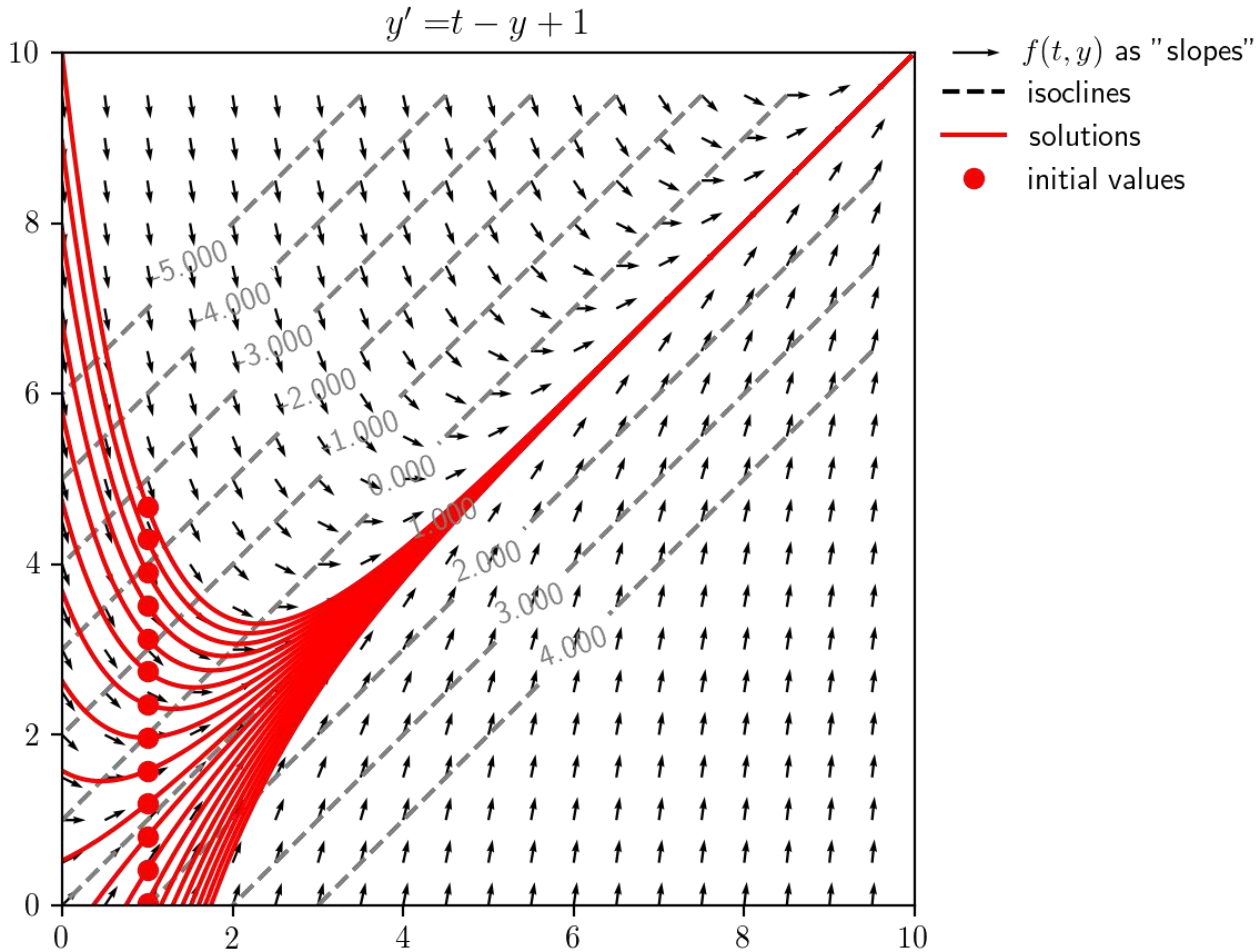


Examples

- Simple example

$$y' = t - y + 1, \qquad y = \frac{te^t + C}{e^t}$$

- Visualization of solutions in the t, y plane (isoclines)



Explicit Euler Method

- Given

$$y' = f\big(t, y(t)\big)$$

- Equidistant discretization in t domain from t_a to t_b

$$t_j = t_a + jh, \quad j = 0,1 \ldots, N, \quad h = \frac{t_b - t_a}{N}$$

- Forward Difference (first order approximation)

$$y'(t_{j-1}) = \frac{y(t_j) - y(t_{j-1})}{h} + O(h^2)$$

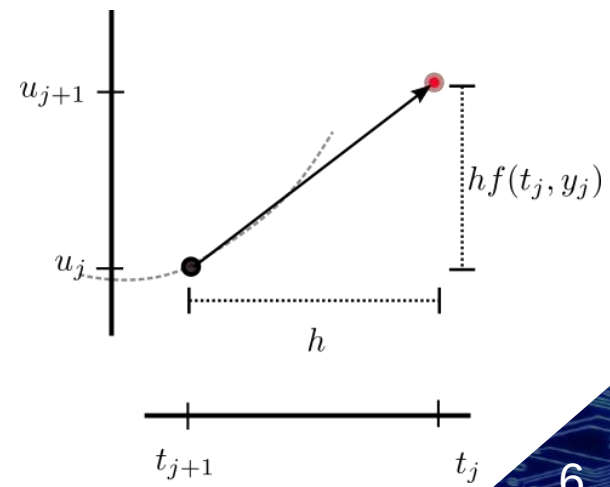
$$y(t_j) = y(t_{j-1}) + hy'(t_{j-1}) + O(h^2)$$

$$y(t_j) = y(t_{j-1}) + hf\left(t_{j-1}, y(t_{j-1})\right) + O(h^2)$$

- Explicit scheme

$$u_o = y_o$$

$$u_{j+1} = u_j + hf(t_j, u_j)$$

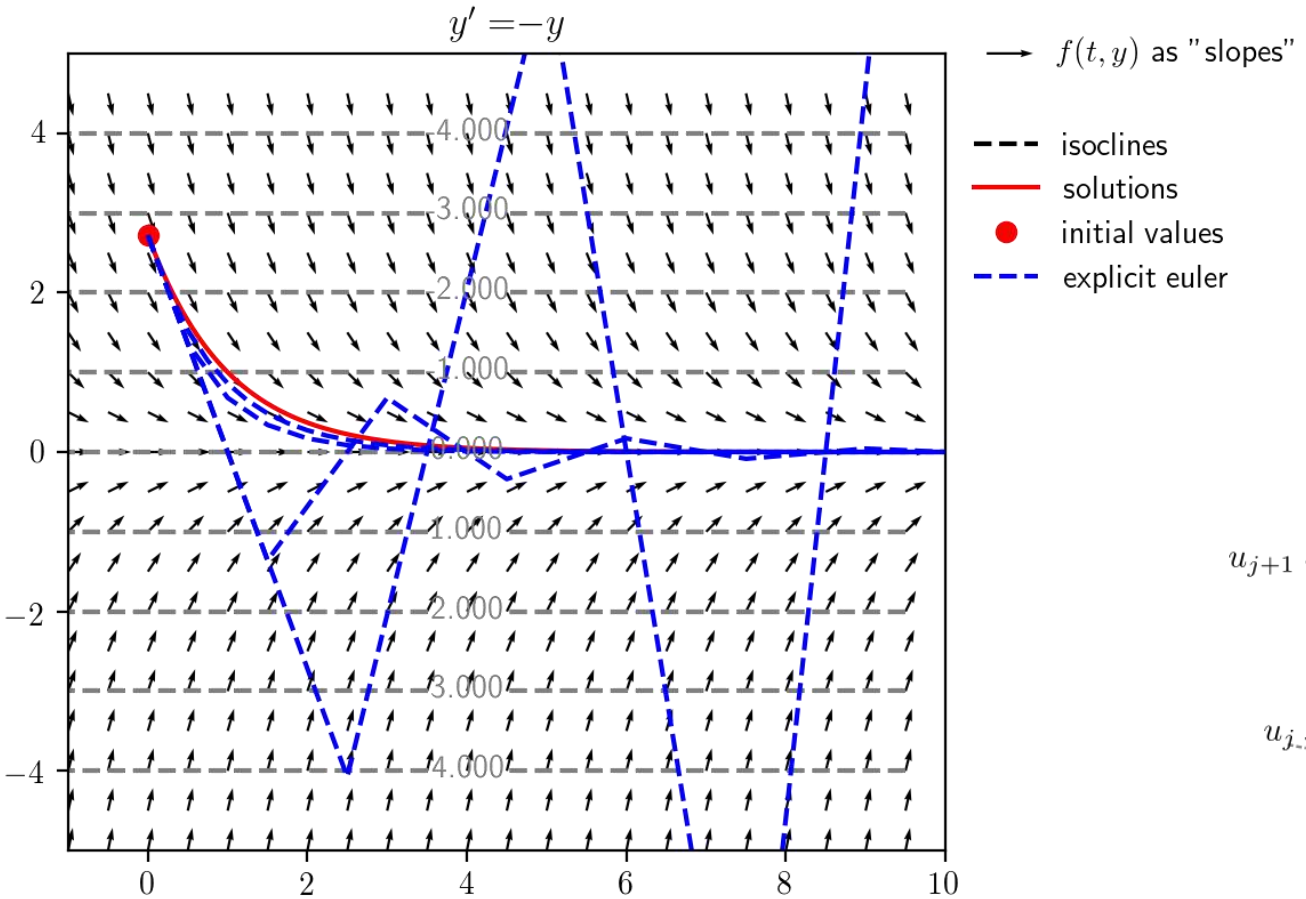


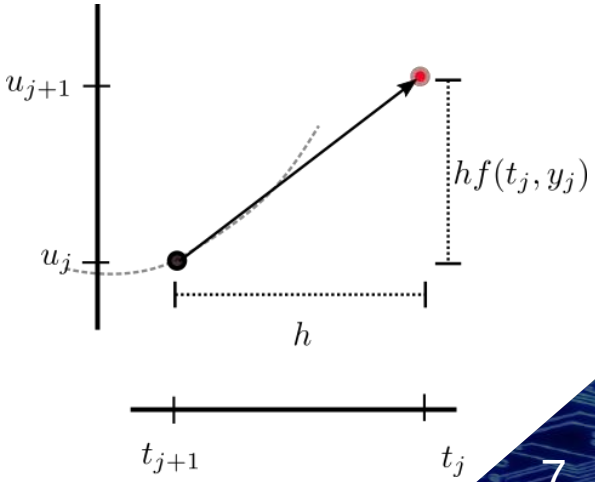
Explicit Euler Method

- Simple example

$$y' = -y, \qquad y = e^{-t+1}$$

- Visualization of approximations in the t, y plane



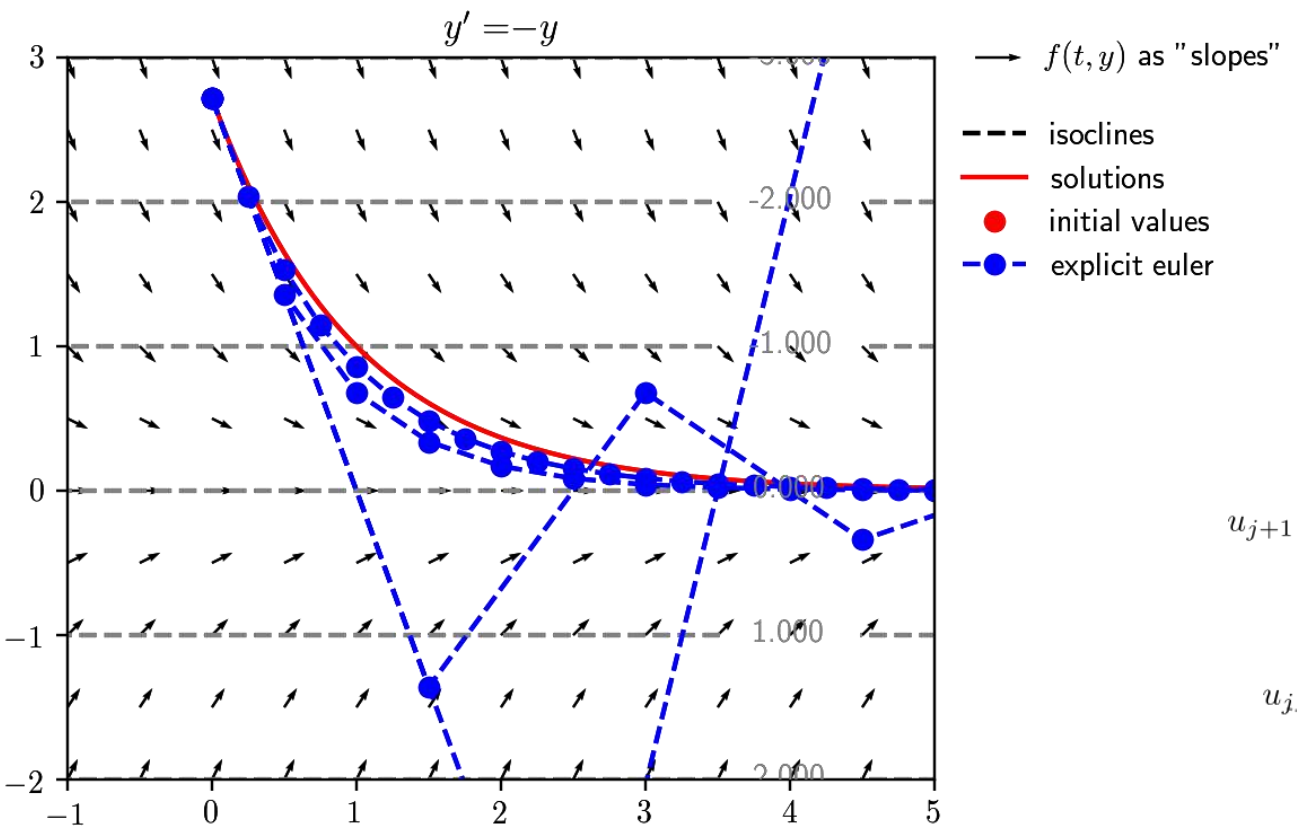


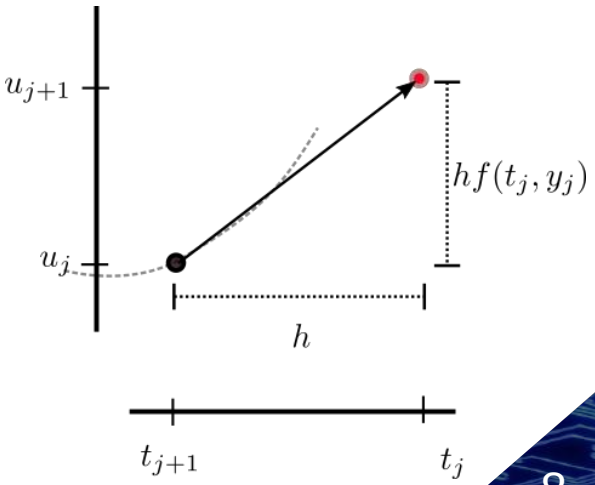
Explicit Euler Method

- Simple example

$$y' = -y, \qquad y = e^{-t+1}$$

- Visualization of approximations in the t, y plane



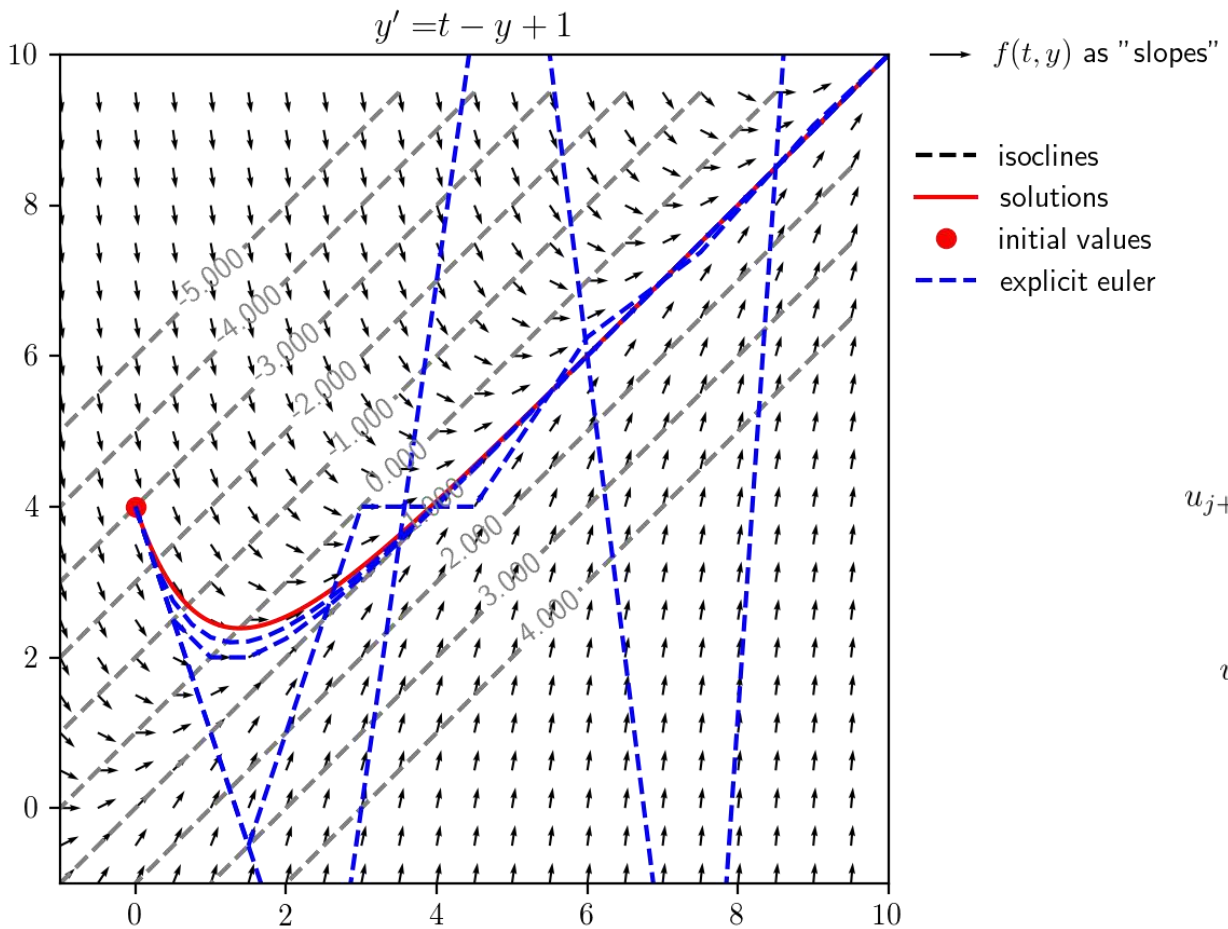


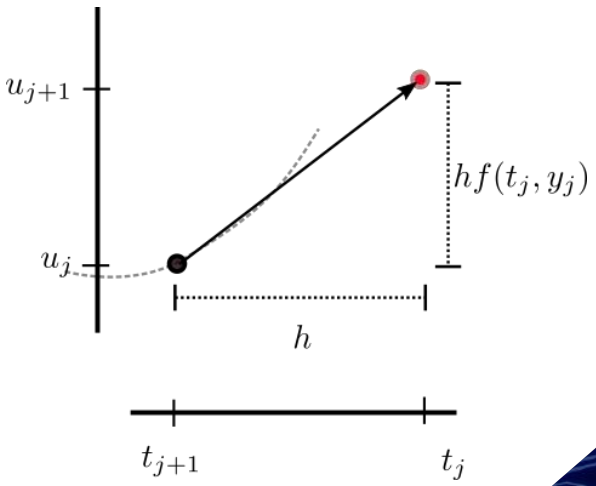
Explicit Euler Method

- Simple example

$$y' = t - y + 1, \qquad y = \frac{te^t + 4}{e^t}$$

- Visualization of approximations in the t, y plane





Implicit Euler Method

- Given

$$y' = f\big(t, y(t)\big)$$

- Equidistant discretization in t domain from t_a to t_b

$$t_j = t_a + jh, \quad j = 0,1 \ldots, N, \quad h = \frac{t_b - t_a}{N}$$

- Backward Difference (first order approximation)

$$y'(t_j) = \frac{y(t_j) - y(t_{j-1})}{h} + O(h^2)$$

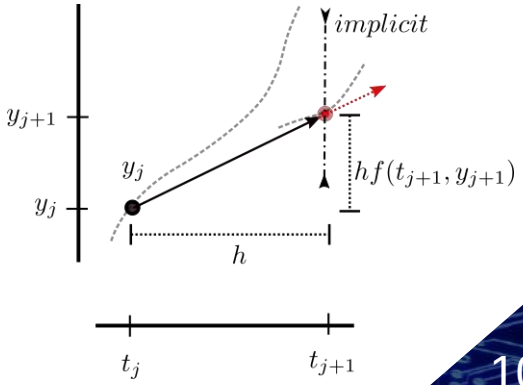
$$y(t_j) = y(t_{j-1}) + hy'(t_j) + O(h^2)$$

$$y(t_j) = y(t_{j-1}) + hf\left(t_j, y(t_j)\right) + O(h^2)$$

- Implicit scheme (in general: solution of nonlinear equation)

$$u_o = y_o$$

$$u_{j+1} = u_j + hf(t_{j+1}, u_{j+1})$$

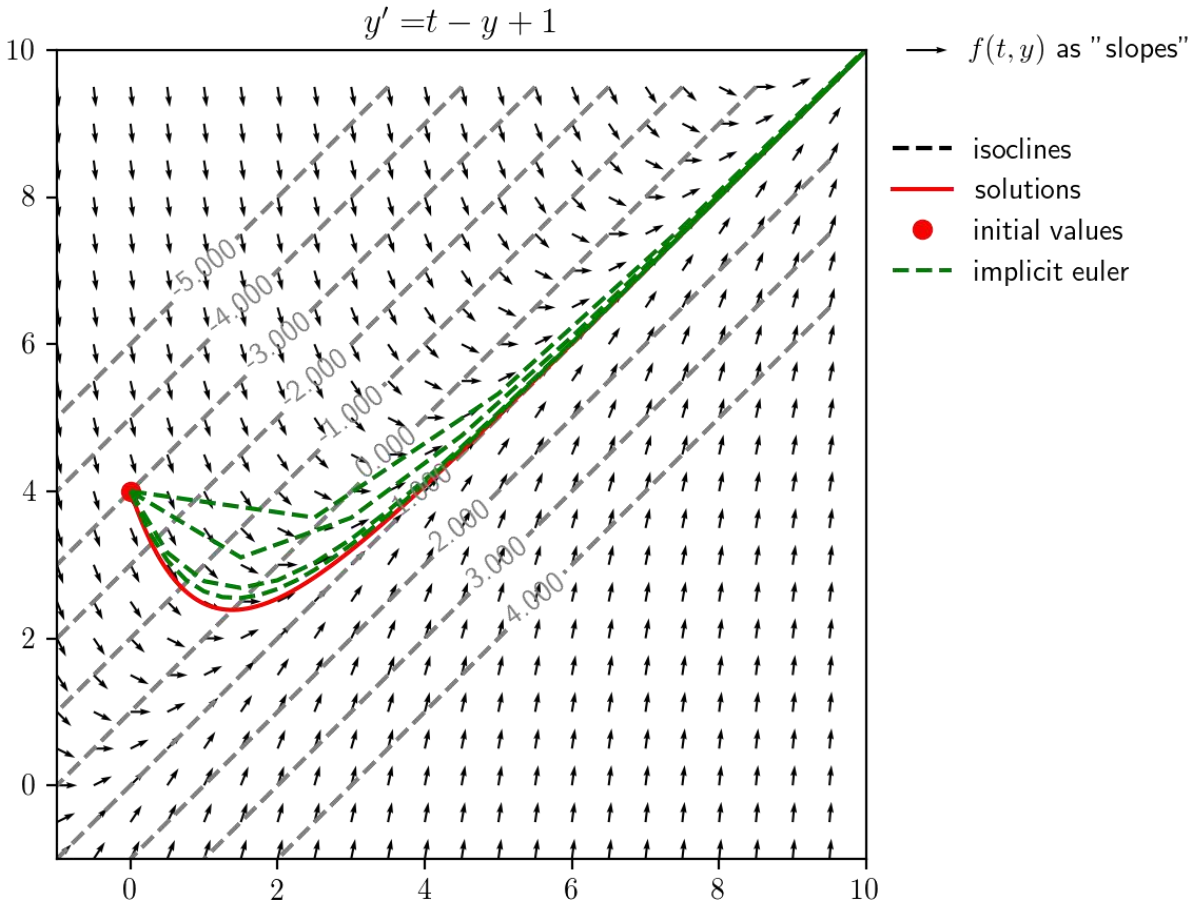


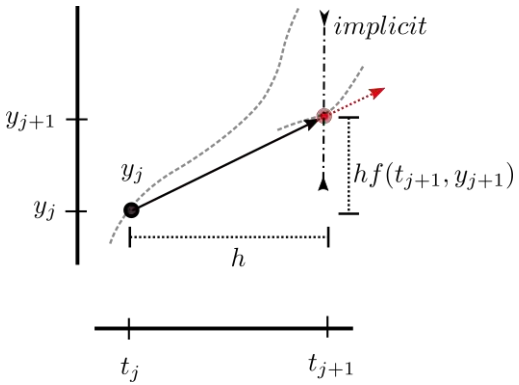
Implicit Euler Method

- Simple example

$$y' = t - y + 1, \qquad y = \frac{te^t + 4}{e^t}$$

- Visualization of approximations in the t, y plane



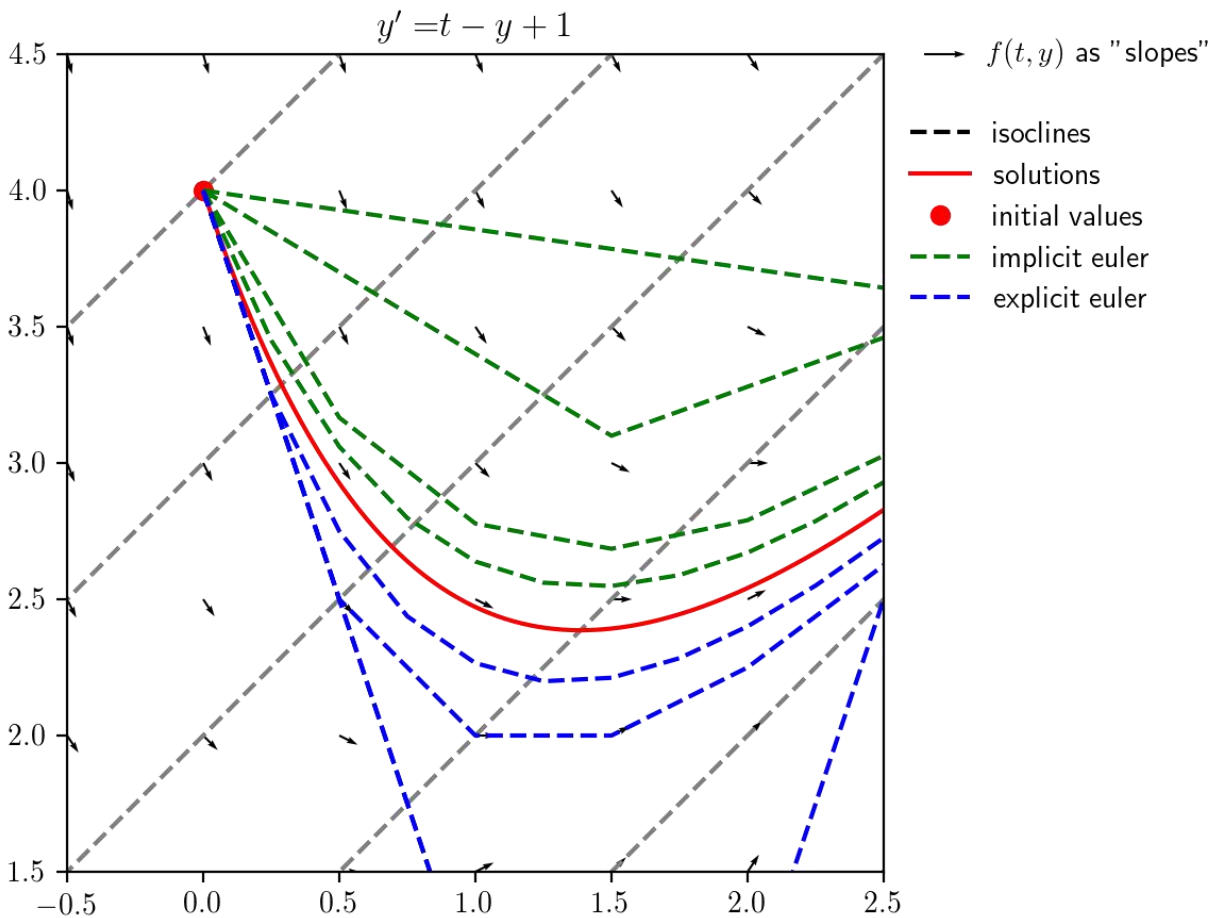


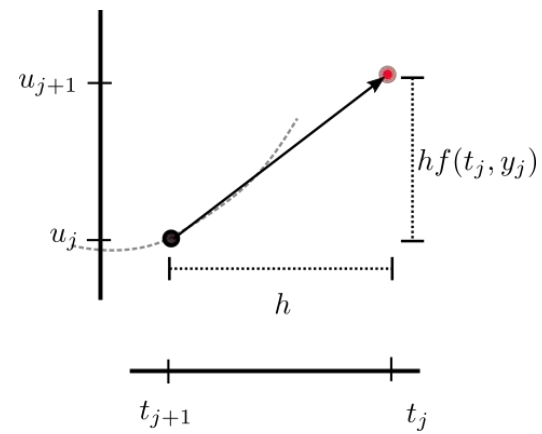
Euler Method

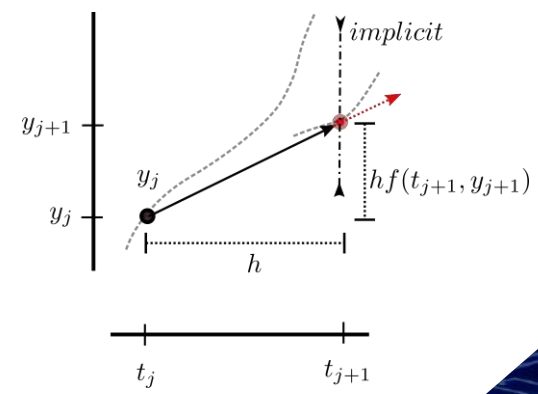
- Simple example

$$y' = t - y + 1, \qquad y = \frac{te^t + 4}{e^t}$$

- Comparison of implicit and explicit approximations







Euler Methods

- Forward Euler method
  - Explicit method
  - Demands small step sizes
- Backward Euler method
  - Implicit method
  - Stable also for large step sizes
- Both methods
  - Are single step methods
  - Produce local truncation error of $O(h^2)$
  - Produce a global error of $O(h)$

Implicit Trapezoidal Rule

- Given

$$y' = f\big(t, y(t)\big)$$

- Equidistant discretization in t domain from t_a to t_b

$$t_j = t_a + jh, \quad j = 0,1 \ldots, N, \quad h = \frac{t_b - t_a}{N}$$

- Reformulation and integration (using Trapezoidal rule)

$$y\big(t_{j+1}\big) - y\big(t_j\big) = \int_{t_j}^{t_{j+1}} f\big(t, y(t)\big)\, dt$$

$$y\big(t_{j+1}\big) = y\big(t_j\big) + \frac{h}{2}\big\{f\big(t_j, y_j\big) + f\big(t_{j+1}, y_{j+1}\big)\big\}$$

- Implicit scheme

$$u_o = y_o$$

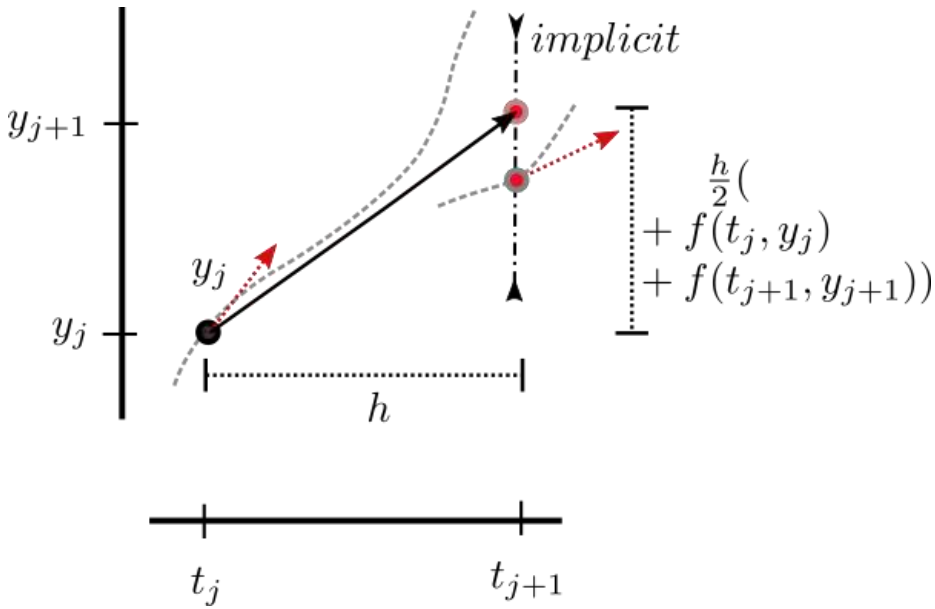
$$u_{j+1} = u_j + \frac{h}{2}\big\{f\big(t_j, u_j\big) + f\big(t_{j+1}, u_{j+1}\big)\big\} + O(h^3)$$

Implicit Trapezoidal Rule

- Implicit scheme

$$u_o = y_o$$

$$u_{j+1} = u_j + \frac{h}{2}\{f(t_j, u_j) + f(t_{j+1}, u_{j+1})\} + O(h^3)$$



Explicit Trapezoidal Rule

- Given

$$y' = f\big(t, y(t)\big)$$

- Equidistant discretization in t domain from t_a to t_b

$$t_j = t_a + jh, \quad j = 0,1 \ldots, N, \quad h = \frac{t_b - t_a}{N}$$

- Reformulation and integration (using Trapezoidal rule)

$$y\big(t_{j+1}\big) - y\big(t_j\big) = \int_{t_j}^{t_{j+1}} f\big(t, y(t)\big)\, dt$$

$$y\big(t_{j+1}\big) = y\big(t_j\big) + \frac{h}{2}\{f\big(t_j, y_j\big) + f(t_{j+1}, y_{j+1})\}$$

- Explicit scheme

$$u_o = y_o$$

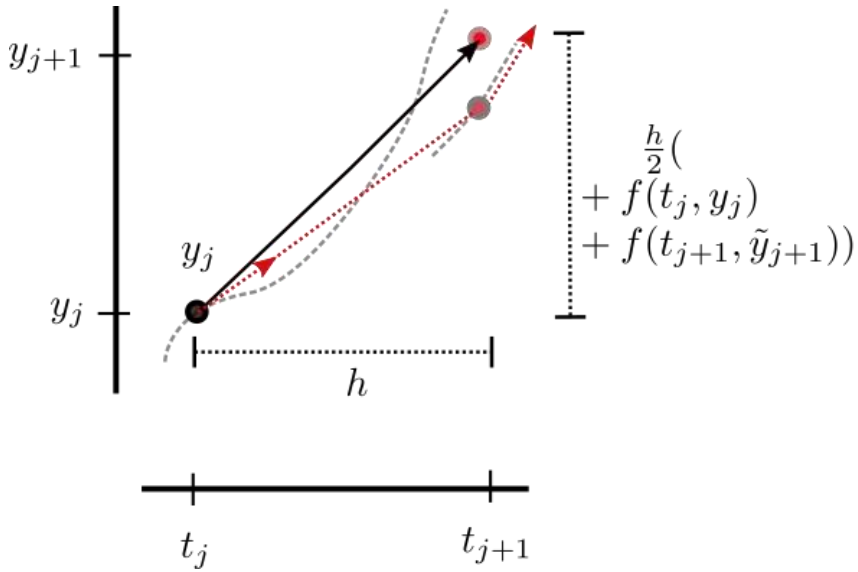
$$\tilde{u}_{j+1} = u_j + hf\big(t_j, u_j\big) + O(h^2)$$

$$u_{j+1} = u_j + \frac{h}{2}\{f\big(t_j, u_j\big) + f(t_{j+1}, \tilde{u}_{j+1})\} + O(h^3)$$

Explicit Trapezoidal Rule

- Explicit scheme

$$u_o = y_o$$
$$\tilde{u}_{j+1} = u_j + hf(t_j, u_j) + O(h^2)$$
$$u_{j+1} = u_j + \frac{h}{2}\{f(t_j, u_j) + f(t_{j+1}, \tilde{u}_{j+1})\} + O(h^3)$$

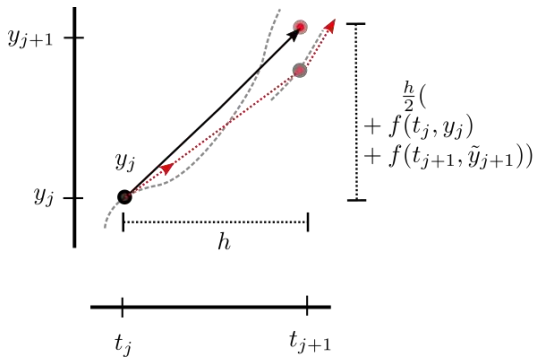


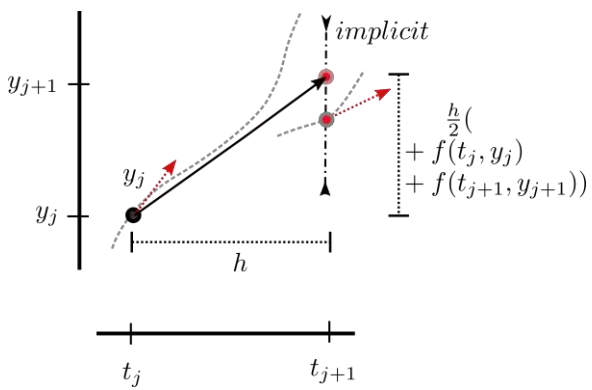
Trapezoidal Rule

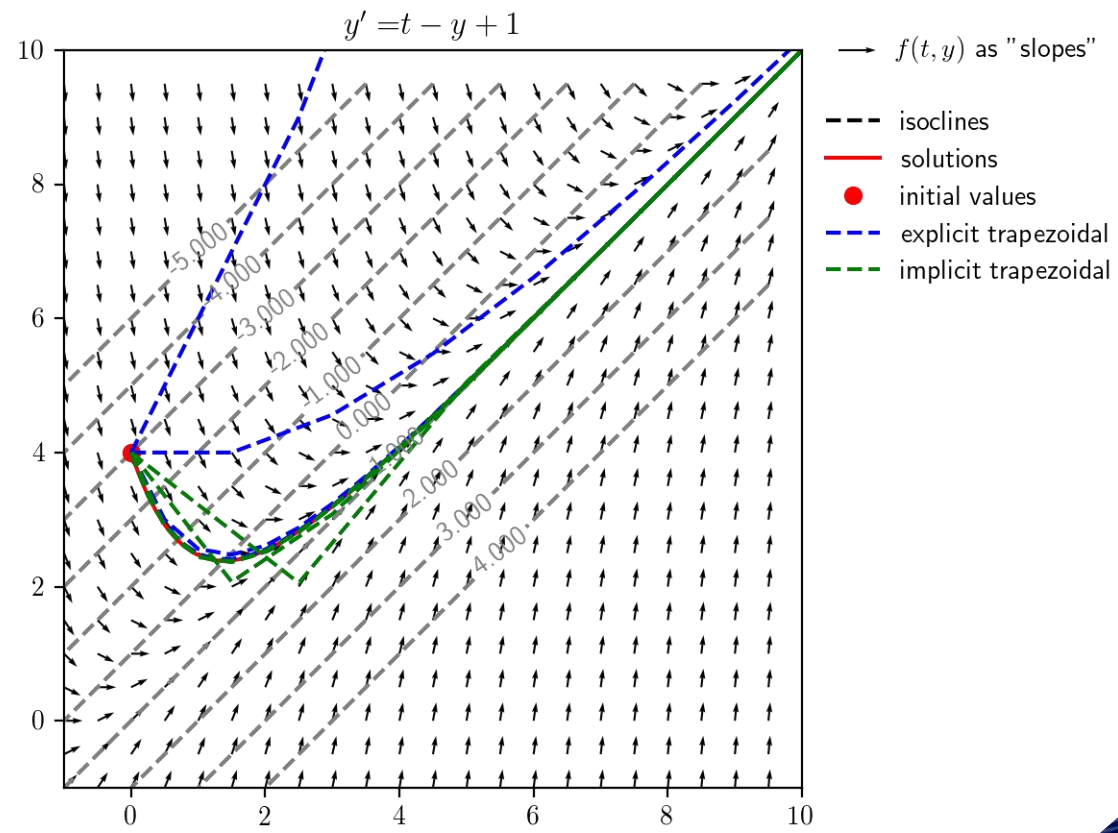
- Simple example

$$y' = t - y + 1, \qquad y = \frac{te^t + 4}{e^t}$$

- Comparison of implicit and explicit approximations





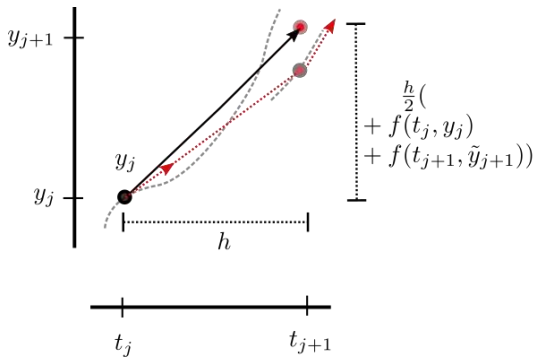


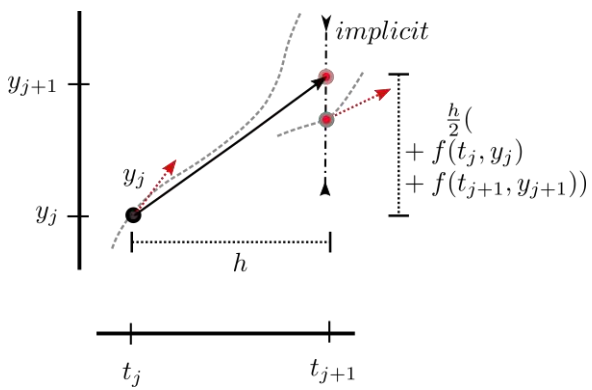
Trapezoidal Rule

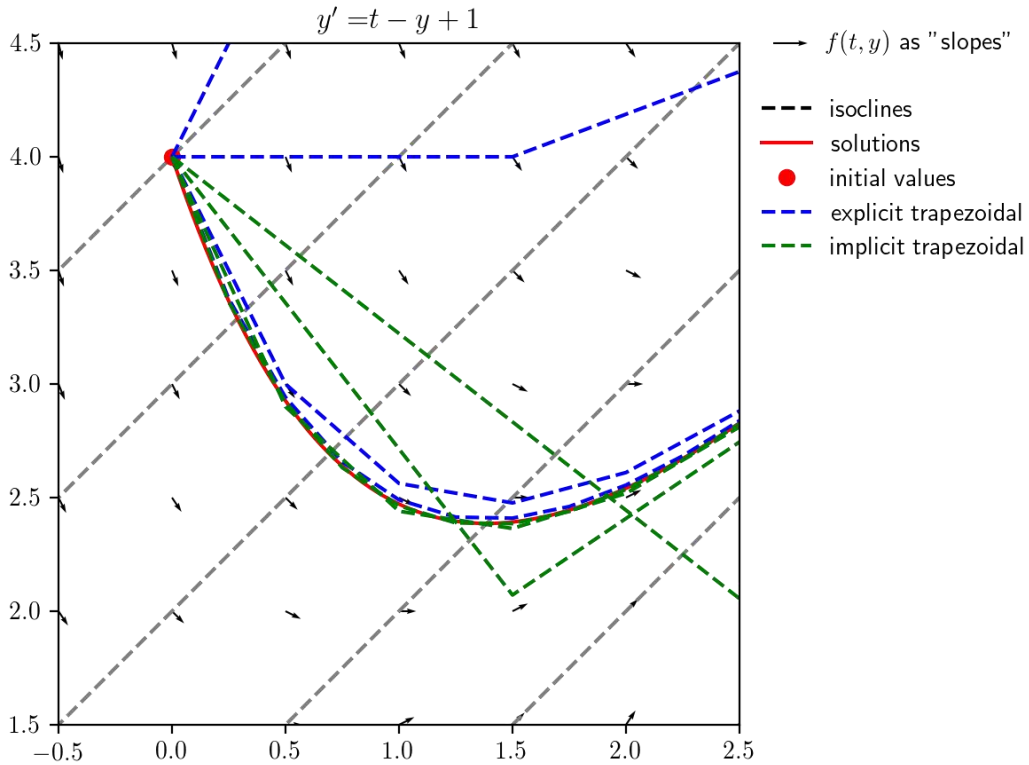
- Simple example

$$y' = t - y + 1, \qquad y = \frac{te^t + 4}{e^t}$$

- Comparison of implicit and explicit approximations







Trapezoidal Rule

- Implicit Trapezoidal Rule
  - Implicit method
  - Stable also for large steps
  - Suitable for stiff problems
- Explicit Trapezoidal Rule
  - Explicit method
  - Improved accuracy compared the forward Euler method
- Both methods
  - Are single step methods
  - Produce local truncation error of $O(h^3)$
  - Produce a global error of $O(h^2)$ and are therefore second order methods

Classic Runge-Kutta Method

- Given

$$y' = f\big(t, y(t)\big)$$

- Equidistant discretization in t domain from t_a to t_b

$$t_j = t_a + jh, \quad j = 0,1 \ldots, N, \quad h = \frac{t_b - t_a}{N}$$

- Reformulation and integration (using Simpson's rule)

$$y\big(t_{j+1}\big) - y\big(t_j\big) = \int_{t_j}^{t_{j+1}} f\big(t, y(t)\big)\, dt$$

$$y\big(t_{j+1}\big) = y\big(t_j\big) + \frac{h}{6}\{f^j + 4f^{j+1/2} + f^{j+1}\}$$

$$y\big(t_{j+1}\big) = y\big(t_j\big) + \frac{h}{6}\{f^j + 2\hat{f}^{j+1/2} + 2\tilde{f}^{j+1/2} + f^{j+1}\}$$

- Idea: successively approximate f for increasing t

Classic Runge-Kutta Method

$$y(t_{j+1}) = y(t_j) + \frac{h}{6}\{f^j + 2\hat{f}^{j+1/2} + 2\tilde{f}^{j+1/2} + \bar{f}^{j+1}\}$$

- Forward Euler

$$\hat{f}^{j+1/2} = f\left(t_{j+1/2}, u_j + \frac{h}{2} f(t_{j,} u_j)\right)$$

- Backward Euler

$$\tilde{f}^{j+1/2} = f\left(t_{j+1/2}, u_j + \frac{h}{2}\hat{f}^{j+1/2}\right)$$

- Midpoint method

$$\bar{f}^{j+1} = f\left(t_{j+1}, u_j + h\tilde{f}^{j+1/2}\right)$$

- Explicit scheme

$$u_o = y_o$$

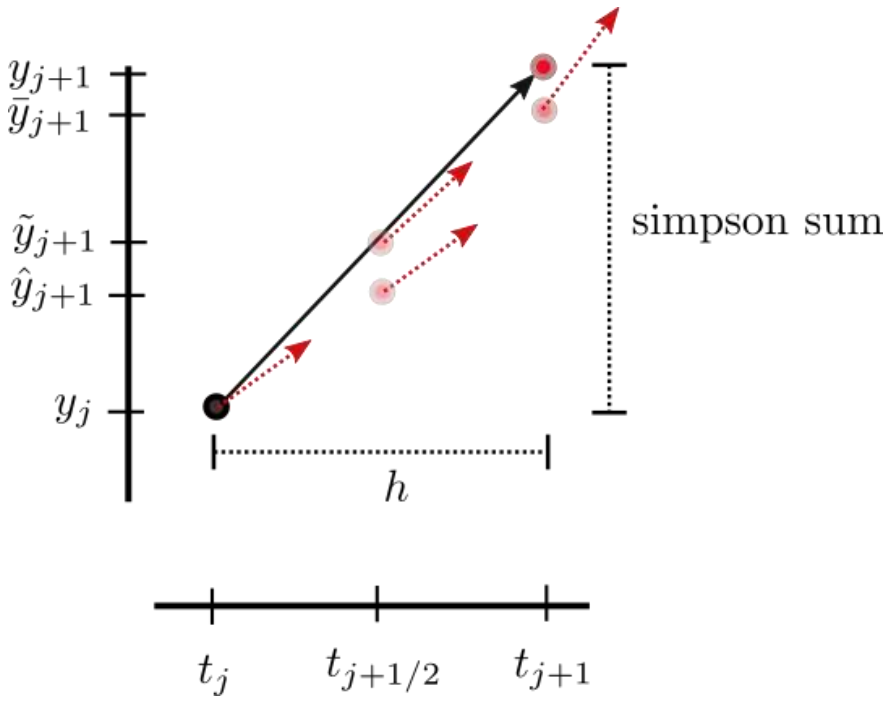
$$u_{j+1} = u_j + \frac{h}{6}\{f^j + 2\hat{f}^{j+1/2} + 2\tilde{f}^{j+1/2} + \bar{f}^{j+1}\}$$

Classic Runge-Kutta Method

- Explicit scheme

$$u_o = y_o$$

$$u_{j+1} = u_j + \frac{h}{6}\left\{f^j + 2\hat{f}^{j+1/2} + 2\tilde{f}^{j+1/2} + \bar{f}^{j+1}\right\}$$

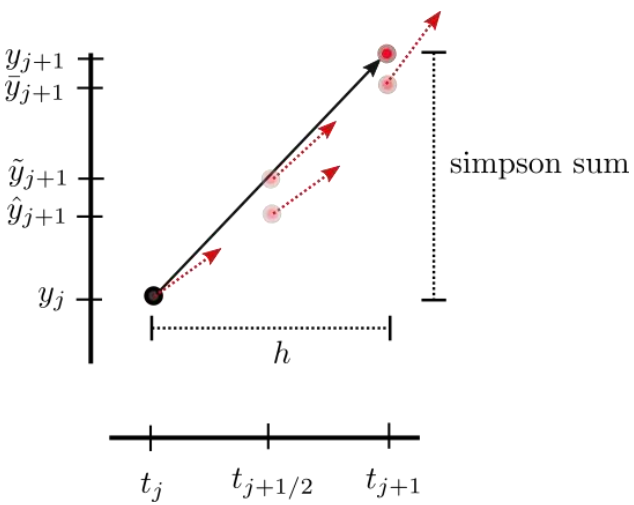


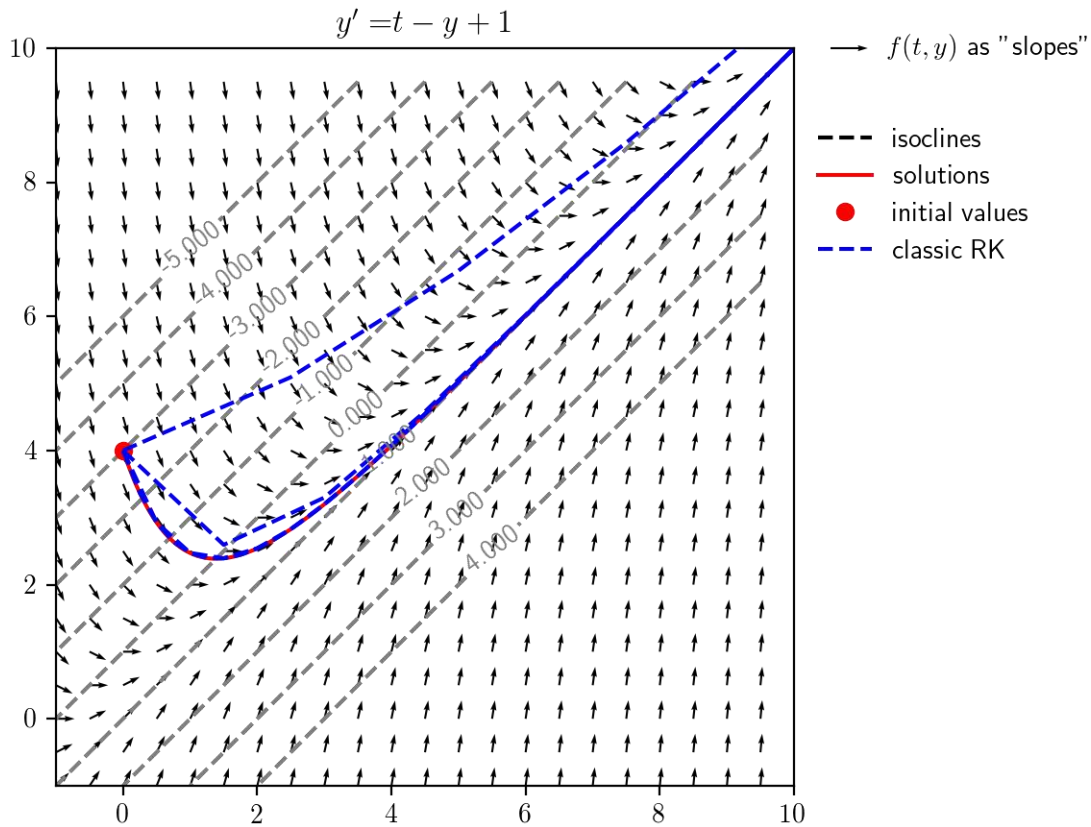
Classic Runge-Kutta Method

- Simple example

$$y' = t - y + 1, \qquad y = \frac{te^t + 4}{e^t}$$

- Approximation using classic Runge-Kutta method





Classic Runge-Kutta Method

- Simple example

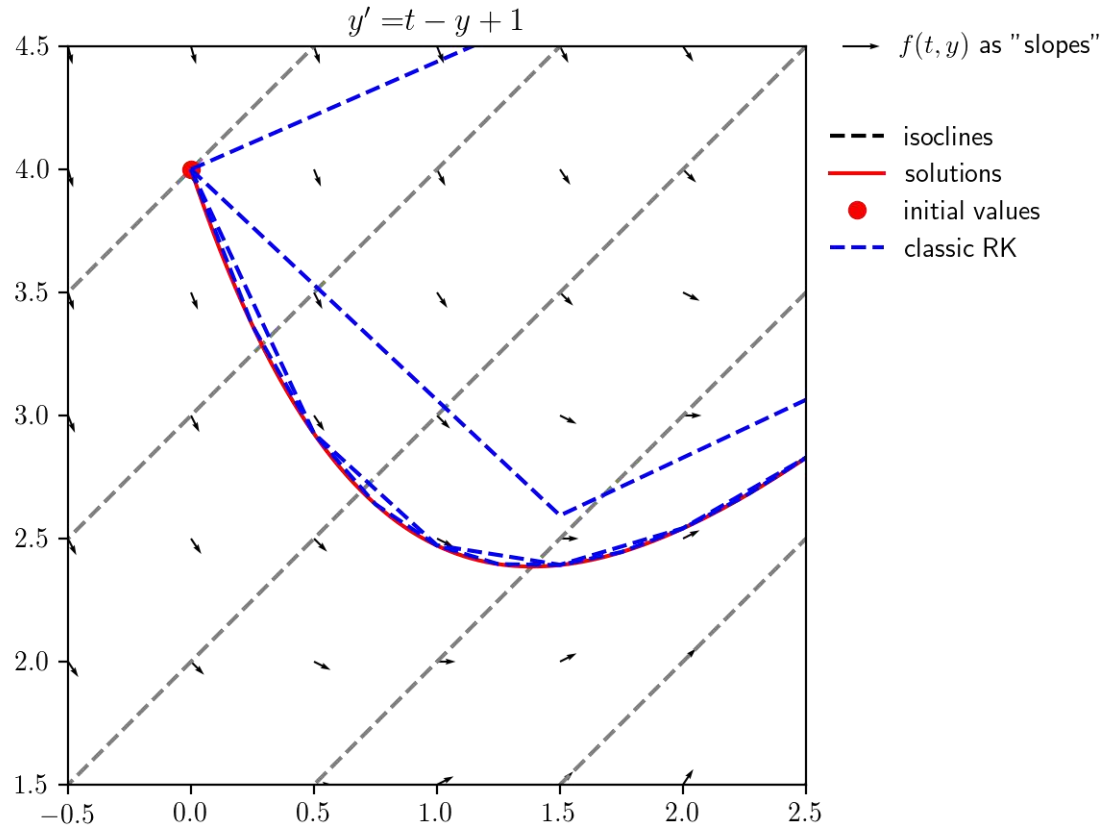
$$y' = t - y + 1, \qquad y = \frac{te^t + 4}{e^t}$$

- Approximation using the classic Runge-Kutta method

- Simple example

$$y' = t - y + 1, \qquad y = \frac{te^t + 4}{e^t}$$

- Approximation using the classic Runge-Kutta method



Runge-Kutta Methods

- Classic Runge-Kutta method
  - Single step method
  - Explicit method
  - Produces a local truncation error of $O(h^5)$
  - Produces a global error of $O(h^4)$
  - Widely used
  - Not useful for stiff problems
- Generalization of Runge-Kutta methods
  - Butcher-Tableau: array notation of the characteristic coefficients
  - (Semi)Implicit Runge-Kutta methods
- Embedded Runge-Kutta methods
  - Runge-Kutta-Fehlberg (RK45) method is a prominent example
    - 4th order method using an embedded 5th order method to approximate local truncation error (adaptive step width control)

Multistep Methods

- Single-step methods
  - Use only the previous evaluation point
  - Intermediate values (to construct higher order methods) are withdrawn after a step is performed
- Multistep methods
  - Previous evaluation points are reused to construct higher order methods (to gain efficiency)
  - Linear multistep methods use a linear combination of previous evaluation points
  - Adams-Bashforth Methods (explicit)
  - Adams-Moulton Methods (implicit)
  - Backward differentiation formulas (implicit)

Adams-Bashforth Methods

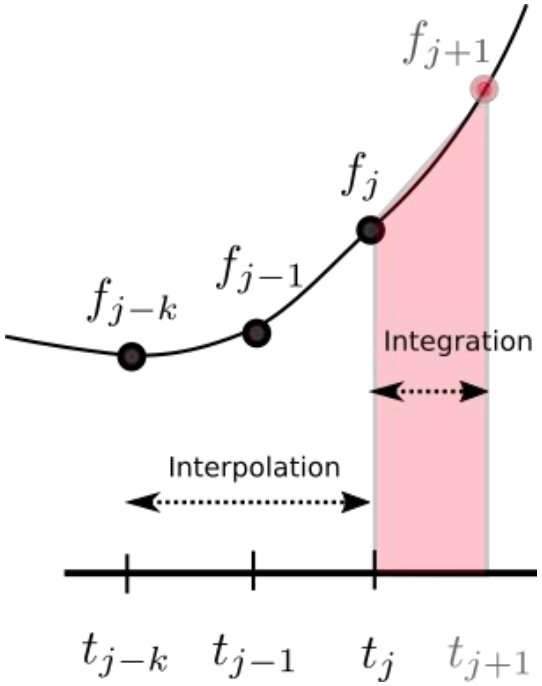
- Given

$$y' = f\big(t, y(t)\big)$$

- Reformulation and polynomial approximation

$$y(t_{j+1}) = y(t_j) + \int_{t_j}^{t_{j+1}} \underbrace{f\big(t, y(t)\big)}_{P_k(t)}\, dt$$

- $P_k(t)$ for f constructed from t_j to t_{j-k} (explicit)



Adams-Moulton Methods

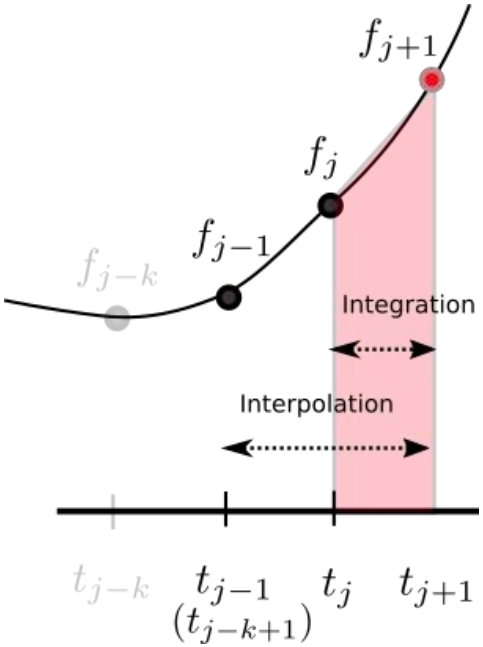
- Given

$$y' = f\big(t, y(t)\big)$$

- Reformulation and Polynomial approximation

$$y(t_{j+1}) = y(t_j) + \int_{t_j}^{t_{j+1}} \underbrace{f\big(t, y(t)\big)}_{P_k(t)}\, dt$$

- $P_k(t)$ for f constructed from t_{j+1} to t_{j-k+1} (implicit)



Backward Differentiation Formulas

- Given

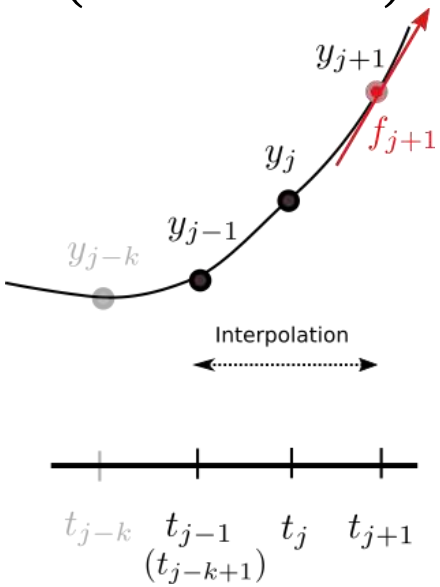
$$y' = f\big(t, y(t)\big)$$

- Reformulation and polynomial approximation

$$\underbrace{y\big(t_{j+1}\big)}_{P_k} = f\,(t_{j+1}, y_{j+1})$$

- $P_k(t)$ directly for y using t_{j+1} to t_{j-k+1} (implicit)
- Condition for $P_k(t)$ at t_{j+1}

$$P_k'\,(t_{j+1}) = f\,\Big(t_{j+1}, P_k\big(t_{j+1}\big)\Big) = f\,\Big(t_{j+1}, y\big(t_{j+1}\big)\Big)$$



Higher Order ODEs

- Any explicit (system of) higher order ODEs can be transformed into an equivalent system of first order ODEs

$$\frac{d^2y}{dt^2} = -ky, \qquad \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} y \\ y' \end{bmatrix}, \quad \begin{bmatrix} z_1' \\ z_2' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

$$my'' + by' + ky = f(t) \qquad \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} y \\ y' \end{bmatrix}, \quad \begin{bmatrix} z_1' \\ z_2' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -b/m & -k/m \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} 0 \\ f(t)/m \end{bmatrix}$$

  - Semi-implicit schemes
    - Sequential evaluation of coupled ODEs
    - Incorporation of already updated values
- Partial Differential Equations
  - Spatial discretization leads to system of coupled ODEs
  - Time integration schemes for ODEs can be used
  - Tailored/adopted schemes for specific problem domains

Conclusions: ODE Methods

- Single-Step
  - Explicit
    - Small step size required, but ‘just’ evaluations, no equations to solve
  - Implicit
    - Large step size, requires solution to (in general) non-linear equation
  - Intermediate evaluations: Runge-Kutta Methods
    - Classic Runge-Kutta: Higher order accuracy using multiple evaluations
- Multistep
  - Idea: efficiency through reuse of previous steps
  - Explicit/Implicit Methods
  - Reuse of past values
- Embedded methods
  - Idea: perform multiple (two) approximations of different order
  - Model local truncation error as difference between the approximations
  - Use error model for ‘automatic’ step size control

Next: finite volume method

Finite Volume Method

- Idea: Reformulate a differential equation using divergence theorem
- Example: Poisson equation

$$\nabla^2 u = \nabla \cdot \nabla u = f$$

- Apply divergence theorem

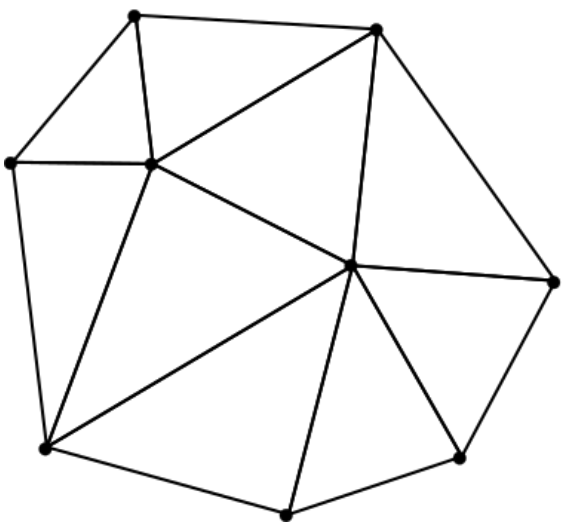
$$\int \nabla \cdot \nabla u dV = \oint \nabla u dA = \int f dV$$

- Finite Volume Method
  - Decomposing the domain into discrete cells (e.g., triangles, boxes)
  - Construct a corresponding dual representation (control volumes)
  - Setup of neighbour information between control volumes sharing surface regions
  - Approximate surface integrals on shared surfaces
  - Define surface integrals on domain boundary according to boundary conditions
  - Assemble linear system of equations

Domain Decomposition

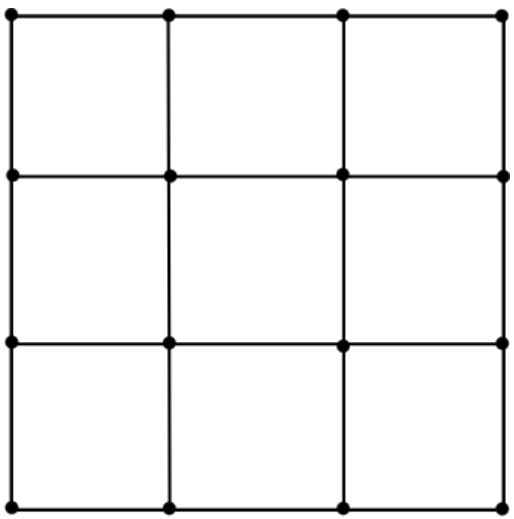
Triangles

Regular Grid

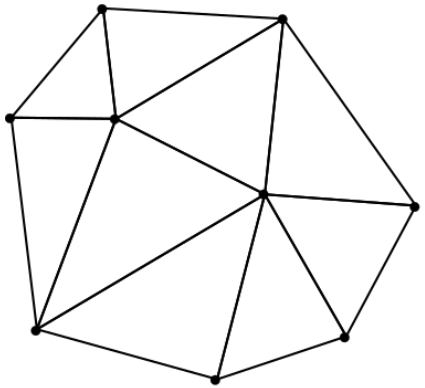


cells

discretized solution

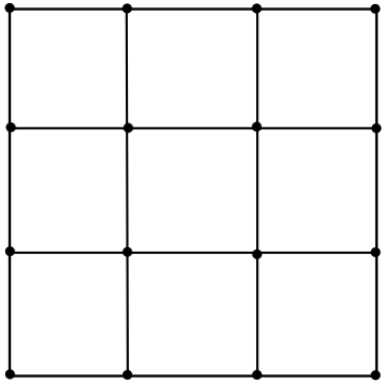


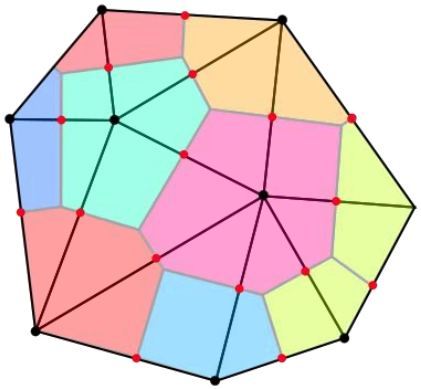
Control Volumes



cells

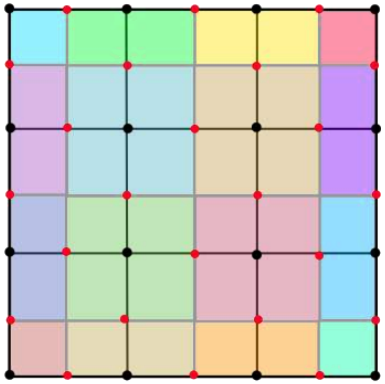
discretized solution

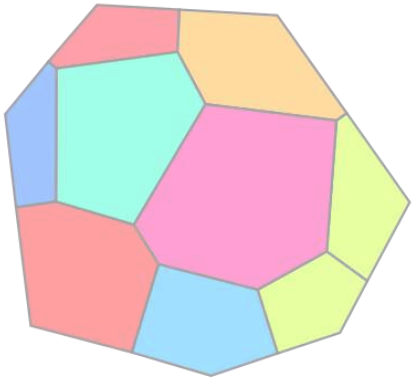




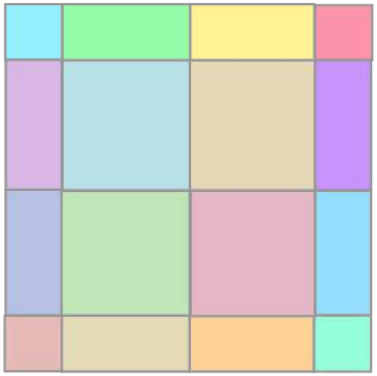
gradient evaluation

discretized solution





control volumes



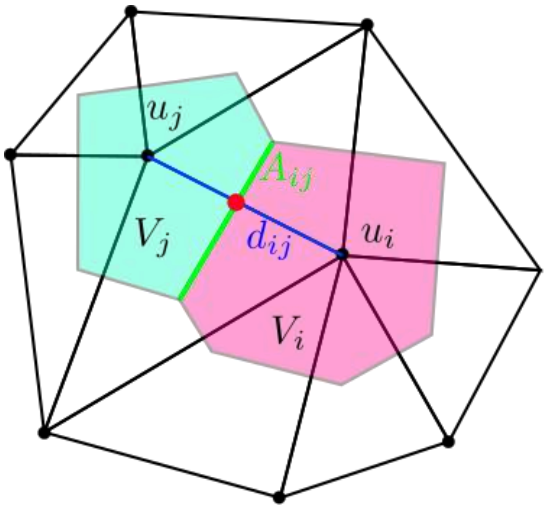
Discrete Gradient

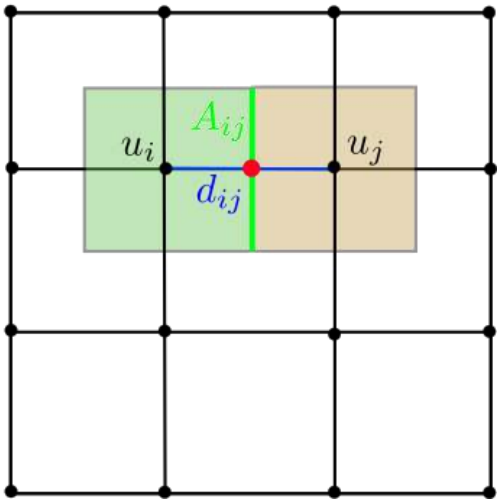
- First order difference approximation of gradient normal (outward) to shared surfaces

$$\oint \nabla u dA = \int f dV$$

$$\sum_j \underbrace{\frac{u_j - u_i}{d_{ij}}}_{\nabla u} A_{ij} = f_i V_i$$

- Conservative discretization





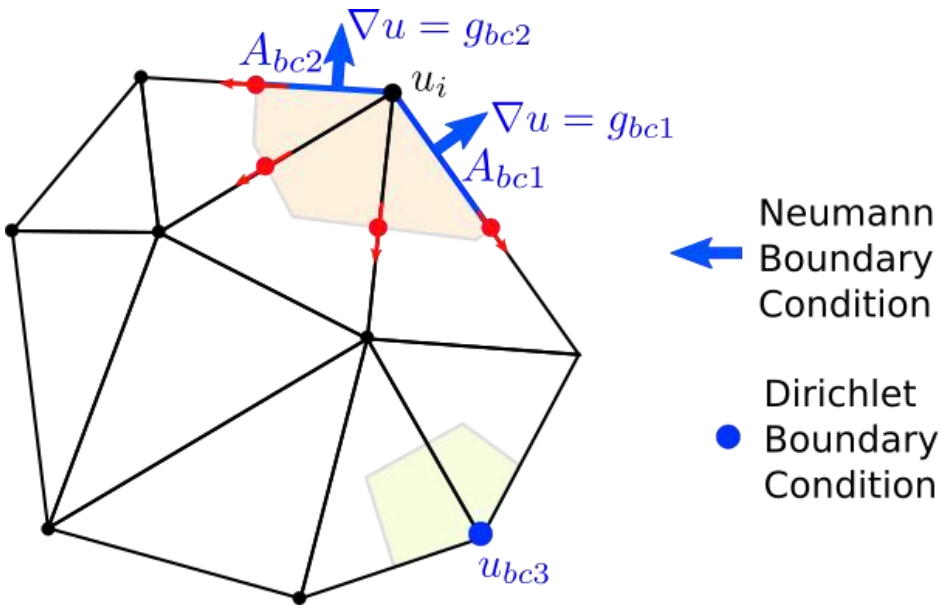
Boundary/Interface Conditions

- Neumann boundary condition

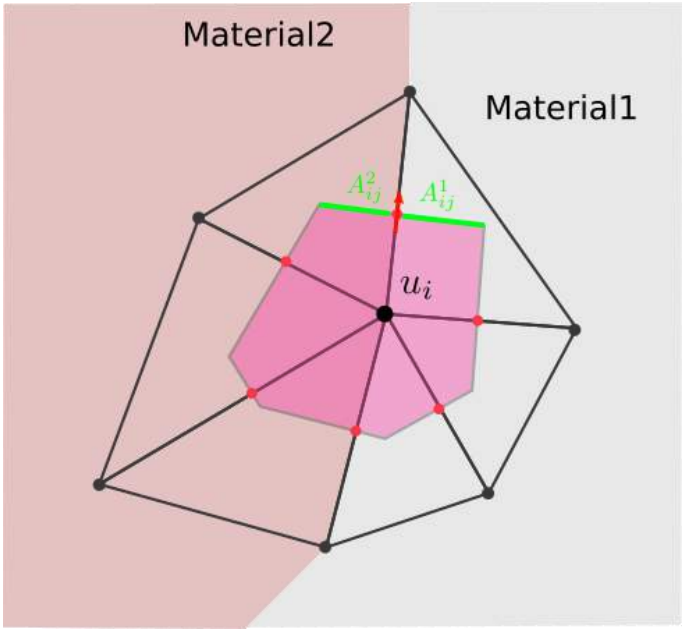
$$\oint \nabla u dA = \int f dV$$

$$\sum_j \frac{u_j - u_i}{d_{ij}} A_{ij} + g_{bc1} A_{bc1} + g_{bc2} A_{bc2} = f_i V_i$$

Boundary condition



Interface condition



Relation to FDM

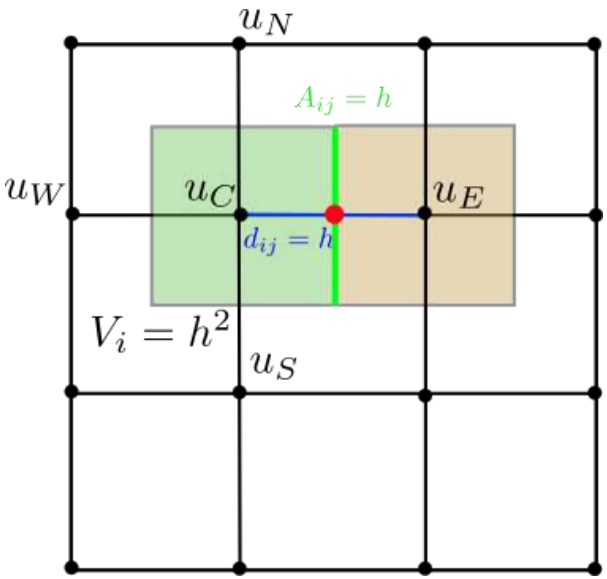
- For regular grid: FVM equal to FDM discretization
  - Finite Volume

$$\sum_{j=N,S,E,W} \underbrace{\frac{u_j - u_C}{h}}_{\nabla u} h = \frac{u_N + u_S + u_E + u_W - 4u_C}{h} h = f_C h^2$$

  - Finite Differences (second order central)

$$(u_{xx} + u_{yy}) \approx \left(\frac{(u_N - 2u_C + u_S)}{h^2} + \frac{(u_W - 2u_C + u_E)}{h^2} \right) =$$

$$= \frac{1}{h^2}\left(u_N + u_S + u_E + u_W - 4u_C\right) = f_C$$



Conclusions: Finite Volume Method

Using triangles (2D) or tetrahedral (3D) as cells:

- Advantages
  - Complex domains possible
  - Straightforward handling of discontinuous properties/interfaces
  - Adaptive discretization resolution straightforward
  - Mesh Requirements: Delaunay triangulation
  - Conservative method

- Disadvantages
  - No implicit connectivity (explicit neighbourhood information required)
  - No constant stencil coefficients (coefficients stored for each control volume)

Conclusions

- Q1: What is the order of the forward/backward Euler method?
- Q2: Are the Runge-Kutta methods single-step or multistep methods?
- Q3: How can first order ODEs methods be applied to higher order ODEs?
- Q4: Why are finite volume schemes ‘conservative’?
- Q5: What are advantages of the Finite Volume Method over the Finite Difference Method?

References

- Norbert Köckler – Numerische Mathematik (2006)
- Svein Linge – Programming for Computations (2016)

1 Classification of PDEs

In this lecture we will discuss some classifications of partial differential equations.

1.1 Notation

We let $u(x_1, \ldots, x_d)$ be a scalar function, that is, $u(x_1, \ldots, x_d) \in \mathbf{R}$ for all $(x_1, \ldots, x_d) \in \mathbf{R}^d$. Let us define the partial derivative of u:

$$\partial_{x_i} u(x_1, \ldots, x_d) := \lim_{h \to 0} \frac{u(x_1, \ldots, x_i + h, \ldots, x_d) - u(x_1, \ldots, x_i, \ldots, x_d)}{h} \tag{1}$$

or equivalently with the notation $x = (x_1, \ldots, x_d)$

$$\partial_{x_i} u(x) = \lim_{h \to 0} \frac{u(x + he_i) - u(x)}{h} \tag{2}$$

where $e_i = (0, \ldots, 1, \ldots, 0)$ (1 at ith entry). Recall that $\partial_{x_i} u(x) = \nabla u \cdot e_i$.

Other notations for partial derivative are (and which we will henceforth use)

$$u_{,i}, \quad u_{x_i}, \quad \frac{\partial}{\partial x_i}, \quad D_{x_i}. \tag{3}$$

Similarly higher order partial derivatives are denoted as follows:

$$u_{,ij}, \quad u_{x_i x_j}, \quad \frac{\partial^2}{\partial x_i \partial x_j}, \quad D_{x_i x_j}. \tag{4}$$

We denote by ∇u the gradient of u and by $\nabla^2 u$ the transposed of the Jacobian of ∇u, that is, $\nabla^2 u := D(\nabla u)^\top$.

1.2 Linear elliptic equations

Under a second order linear partial differential operator in divergence form we understand:

$$Lu(x) := -\operatorname{div}(A(x)\nabla u(x)) + b(x) \cdot \nabla u(x) + c(x)u(x) \quad \text{for } x \in \Omega, \tag{5}$$

where $A : \overline{\Omega} \to \mathbf{R}^{d \times d}$ is a matrix-valued function, $b : \overline{\Omega} \to \mathbf{R}^d$ and $c : \overline{\Omega} \to \mathbf{R}$ are given functions. For given $f : \overline{\Omega} \to \mathbf{R}$ one obtains the associated differential equation:

$$Lu(x) = f(x) \quad \text{for } x \in \Omega. \tag{6}$$

We note that (5) can be equivalently written as:

$$Lu(x) = -\sum_{i,j=1}^{d} (a_{ij}u_{x_j})_{x_i} + \sum_{j=1}^{d} b_j(x)u_{x_j}(x) + c(x)u. \tag{7}$$

Here $a_{ij}(x)$ are the entries of the matrix $A(x)$ and $b(x) = (b_1(x), \ldots, b_d(x))$.

Definition 1.1. We say the differential operator L is elliptic if A is uniformly positive definite on $\overline{\Omega}$, that is, there is $c > 0$, such that

$$A(x)v \cdot v \geqslant c\|v\|^2 \quad \text{for all } v \in \mathbf{R}^d \tag{8}$$

or equivalently

$$\sum_{i,j=1}^{d} a_{ij}(x)v_i v_j \geqslant c\|v\|^2. \tag{9}$$

Here $\|v\| := \sqrt{\sum_{i=1}^{d} v_i^2}$ denotes the Euclidean norm of $v = (v_1, \ldots, v_d) \in \mathbf{R}^d$.

Example 1.1. As an example we consider $A(x) = I$, $b = 0$ and $c = 0$, where $I \in \mathbf{R}^{d\times d}$ denotes the identity matrix. This yields the Poisson equation

$$-\Delta u = f \quad \text{in } \Omega. \tag{10}$$

In order for (10) to have a unique solution we have to impose boundary conditions. For instance

- $u = 0$ on $\partial\Omega$ (homogeneous Dirichlet boundary conditions)
- $\nabla u \cdot n = 0$ on $\partial\Omega$ (homogeneous Neumann boundary conditions)

where n denotes the outward pointing normal vector field along $\partial\Omega$

Example 1.2. A second example is $A(x) = \alpha_1\chi_\omega(x) + \alpha_2\chi_{\Omega\setminus\omega}(x)$, where χ_ω denotes the characteristic function of the set ω, which is 1 if $x \in \omega$ and 0 otherwise. With this

$$\begin{aligned} -\alpha_1\Delta u &= f \quad \text{in } \Omega^+ := \omega \\ -\alpha_2\Delta u &= f \quad \text{in } \Omega^- := \Omega \setminus \overline{\omega}. \end{aligned} \tag{11}$$

In this case a solution u to (11) must satisfy the following transmission conditions:

$$\begin{aligned} u^+ = u^- &\quad \text{on } \partial\Omega^+ \cap \partial\Omega^- \\ \alpha_1\nabla u^+ \cdot n = \alpha_2\nabla u^- \cdot n &\quad \text{on } \partial\Omega^+ \cap \partial\Omega^- \end{aligned} \tag{12}$$

where $u^+ := u|_{\Omega^+}$ and $u^- := u|_{\Omega^-}$ denote the restrictions of u to Ω^+ and Ω^-, respectively.

Notice that if the a_{ij} are differentiable, then one can write by the product rule

$$Lu(x) = -\sum_{i,j=1}^{d} a_{ij} u_{x_i x_j} + \sum_{i=1}^{d} \tilde{b}_i(x) u_{x_i}(x) + c(x)u. \tag{13}$$

where $\tilde{b}_i(x) := b_i(x) - \sum_{j=1}^{d} (a_{ij})_{x_i} u_{x_j}$. One also says in this case that the operator L is in non-divergence form.

For the sake of completeness we associate a bilinear form with an elliptic operator.

Definition 1.2. We define the bilinear form $B : H_0^1(\Omega) \times H_0^1(\Omega) \to \mathbf{R}$ by

$$B(u, v) := \int_\Omega A(x)\nabla u(x) \cdot \nabla v(x) + b(x) \cdot u(x)v(x) + c(x)u(x)v(x). \tag{14}$$

With this bilinear form one can formulate for instance (6) when imposing Dirichlet boundary conditions on $\partial\Omega$ as follows: find $u \in H_0^1(\Omega)$, such that

$$B(u, v) = \int_\Omega fv \, dx \quad \text{for all } v \in H_0^1(\Omega). \tag{15}$$

1.3 Linear parabolic equations

Under a linear parabolic equation we understand an equation

$$u_t + Lu(x) = f(x) \quad \text{in } (0, T) \times \Omega, \tag{16}$$

where $u(t, x_1, \ldots, x_d)$ is the unknown, $f : \overline{\Omega} \to \mathbf{R}$ is given and L is a second order differential operator of divergence form

$$Lu(x) = -\sum_{i,j=1}^{d} (a_{ij}(t, x)u_{x_j})_{x_i} + \sum_{j=1}^{d} b_j(t, x)u_{x_j}(x) + c(t, x)u \tag{17}$$

or non-divergence form

$$Lu(x) = -\sum_{i,j=1}^{d} a_{ij}(t, x)u_{x_j x_i} + \sum_{j=1}^{d} b_j(x)u_{x_j}(x) + c(x)u \tag{18}$$

Example 1.3. For $Lu = \Delta u$ we obtain the heat equation

$$u_t - \Delta u = f \quad \text{in } (0, T) \times \Omega. \tag{19}$$

In order for (16) to have a unique solution one has to impose and initial "temperature" and boundary conditions:

- $u(0,x) = g(x) \quad$ for $x \in \Omega$
- $u(t,x) = 0 \quad$ for $(t,x) \in (0,T] \times \partial\Omega$

where $g : \overline{\Omega} \to \mathbf{R}$ is given.

Definition 1.3. We say that the operator $\frac{\partial}{\partial t} + L$ is uniformly parabolic if

$$\sum_{i,j=1}^{d} a_{ij}(t,x)v_i v_j \geqslant c\|v\|^2, \tag{20}$$

for all $v \in \mathbf{R}^d$, $(t,x) \in [0,T] \times \overline{\Omega}$.

1.4 Linear hyperbolic equations

Under a linear hyperbolic equation we understand an equation

$$u_{tt} + Lu(x) = f(x) \quad \text{ in } (0,T) \times \Omega, \tag{21}$$

where $u(t, x_1, \ldots, x_d)$ is the unknown, $f : \overline{\Omega} \to \mathbf{R}$ is given and L is a second order differential operator of divergence form

$$Lu(x) = -\sum_{i,j=1}^{d} (a_{ij}(t,x)u_{x_j})_{x_i} + \sum_{j=1} b_j(t,x)u_{x_j}(x) + c(t,x)u \tag{22}$$

or non-divergence form

$$Lu(x) = -\sum_{i,j=1}^{d} (a_{ij}(t,x)u_{x_j})_{x_i} + \sum_{j=1} b_j(t,x)u_{x_j}(x) + c(t,x)u \tag{23}$$

Example 1.4. The choice $L = -\Delta$ (that is $A = I$, $b = 0$ and $c = 0$) give the wave equation

$$u_{tt} - \Delta u = f.$$

Definition 1.4. We say that the operator $\frac{\partial^2}{\partial^2 t} + L$ is uniformly hyperbolic if

$$\sum_{i,j=1}^{d} a_{ij}(t,x)v_i v_j \geqslant c\|v\|^2, \tag{24}$$

for all $v \in \mathbf{R}^d$, $(t,x) \in [0,T] \times \overline{\Omega}$.

Similarly to the parabolic equation we need to prescribe initial data and boundary data:

- $u(0,x) = g_1(x) \quad$ for $x \in \Omega$
- $u_t(0,x) = g_2(x) \quad$ for $x \in \Omega$
- $u(t,x) = 0 \quad$ for $(t,x) \in [0,T] \times \partial\Omega$

where $g_1, g_1 : \overline{\Omega} \to \mathbf{R}$ are given.

1.5 First order fully nonlinear equations

Let $\Omega \subset \mathbf{R}^d$ be a bounded domain and let $F : \overline{\Omega} \times \mathbf{R} \times \mathbf{R}^d \to \mathbf{R}$. Under a fully nonlinear second order partial differential equation we understand:

$$F(x_1, \ldots, x_d, u(x_1, \ldots, x_d), \nabla u(x_1, \ldots, x_d)) = 0 \tag{25}$$

for all $(x_1, \ldots, x_d) \in \Omega$, or shorter with the notation $x := (x_1, \ldots, x_d)$

$$F(x, \nabla u(x)) = 0 \quad \text{for all } x \in \Omega. \tag{26}$$

Example 1.5 (Eikonal equation)**.** The Eikonal equation is a first order equation: find $u : \overline{\Omega} \to \mathbf{R}$, such that,

$$\begin{aligned} |\nabla u| &= 1 \quad \text{on } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned} \tag{27}$$

The interpretation of $u(x)$ at a point $x \in \Omega$ is the distance to the boundary $\partial\Omega$, that means, $u(x) = \min_{y \in \partial\Omega} \|x - y\|$.

1.6 Second order fully nonlinear equations

Let $\Omega \subset \mathbf{R}^d$ be a bounded domain and let $F : \overline{\Omega} \times \mathbf{R} \times \mathbf{R}^d \times \mathbf{R}^{d \times d} \to \mathbf{R}$. Under a fully nonlinear second order partial differential equation we understand:

$$F(x_1, \ldots, x_d, u(x_1, \ldots, x_d), \nabla u(x_1, \ldots, x_d), \nabla^2 u(x_1, \ldots, x_d)) = 0 \tag{28}$$

for all $(x_1, \ldots, x_d) \in \Omega$, or shorter with the notation $x := (x_1, \ldots, x_d)$

$$F(x, \nabla u(x), \nabla^2 u(x)) = 0 \quad \text{for all } x \in \Omega. \tag{29}$$

Example 1.6 (Monge-Ampère)**.** An example of a fully nonlinear PDE is the Monge-Ampère equation:

$$u_{xx}(x,y) u_{yy}(x,y) - (u_{xy}(x,y))^2 = 0 \quad \text{for } (x,y) \in \mathbf{R}^2. \tag{30}$$

A generalisation of this equation to arbitrary space dimension is

$$\det(\nabla^2 u) = \psi \quad \text{in } \mathbf{R}^d, \tag{31}$$

where $\psi : \mathbf{R}^d \to \mathbf{R}$ is some given function.

Example 1.7 (Nonlinear heat equation)**.** Find $u : (0, \infty) \times \mathbf{R} \to \mathbf{R}$, such that

$$u_t = \frac{u_{xx}}{u_x^2}. \tag{32}$$

Assume that u is a solution to (32) and $u_x > 0$. For fixed t we denote the inverse of $x \mapsto u(t, x)$ by $v(t, x) = u^{-1}(t, x)$. The v solves the <u>linear equation</u>:

$$v_t = v_{xx} + v_x. \tag{33}$$

1.7 Semilinear equations

Roughly speaking semilinear equation are equations where the nonlinearity does not appear in the leading part of the operator.

Definition 1.5. We call

$$Lu(x) + \rho(x, u(x)) = f(x) \quad x \in \Omega \tag{34}$$

second order semilinear equation, where $\rho = \rho(x, y) : \overline{\Omega} \times \mathbf{R} \to \mathbf{R}$ is a given function and L is a second order differential operator as in Definition 5 or 7. We say the operator $L + \rho$ is monotone provided

- L is uniformly elliptic
- $\partial_y \rho(x, y) \geqslant 0$ for all $(x, y) \in \overline{\Omega} \times \mathbf{R}$

Example 1.8. A typical example is $L = -\Delta$ (that is $A = I$, $b = 0$ and $c = 0$) and $\rho(x, y) = y^3$. Then (34) becomes:

$$-\Delta u + u^3 = f \quad \text{ in } \Omega. \tag{35}$$

Of course also in this case we have to prescribe boundary conditions to get unique solvability. For instance homogeneous Neumann boundary conditions $u = 0$ on $\partial\Omega$.

1.8 Quasilinear equations

Finally let us turn our attention to quasilinear equations.

Definition 1.6. Under a second order quasilinear equation we understand

$$Qu(x) := -\sum_{i,j=1}^{d} a_{ij}(x, u(x), \nabla u(x)) u_{x_i x_j} + b(x, u(x), \nabla u(x)) = f(x) \quad \text{ in } \Omega, \tag{36}$$

where $a_{ij}, b : \overline{\Omega} \times \mathbf{R} \times \mathbf{R}^d \to \mathbf{R}$ are given functions. The operator Q is called *elliptic* if there is a constant $c >$, such that

$$\sum_{i,j=1}^{d} a_{ij}(x, z, p)\nu_i\nu_j \geqslant c\|\nu\|^2, \tag{37}$$

where $\nu = (\nu_1, \ldots, \nu_d)$.

Similarly to linear second order PDE on says the operator Q is in divergence form if we can write it as

$$Qu(x) = \sum_{i,j=1}^{d} (a_{ij}(x, u(x), \nabla u(x))u_{x_j})_{x_i} + b(x, u(x), \nabla u(x)) \tag{38}$$

or with $A(x, u(x), \nabla u(x)) := (a_{ij})$ as

$$Qu(x) = -\operatorname{div}(A(x, u(x), \nabla u(x))\nabla u(x)) + b(x, u(x), \nabla u(x)). \tag{39}$$

Example 1.9. As an example we consider

$$Qu = -\Delta u + \sum_{i,j=1}^{d} \frac{u_{x_i}u_{x_j}}{(1 + |\nabla u|^2)} u_{x_i x_j}. \tag{40}$$

and $b(x, u(x), \nabla u(x)) = f(x)$.

Example 1.10 (Equation of Capillarity)**.** We consider the equilibrium shape of a liquid surface with constant surface tension. The liquid is supposed to be in a homogeneous gravity field. Then the surface can be described by the equation

$$\operatorname{div}\left(\frac{\nabla u}{\sqrt{1 + |\nabla u|^2}}\right) = \kappa u. \tag{41}$$

Here u denotes the perturbation of a reference shape $S \subset \mathbf{R}^2$. That means every point $x \in S$ is moved to the equilibrium point $(x, u(x))$.

Example 1.11. Another example is the p-Laplacian

$$-\operatorname{div}(|\nabla u|^{p-2}\nabla u) = f \quad \text{in } \Omega. \tag{42}$$

This equation is the Euler-Lagrange equation of the energy

$$E(\varphi) := \frac{1}{p}\int_\Omega |\nabla\varphi|^p - \int_\Omega f\varphi \, dx. \tag{43}$$

Literature

- L. Evans - Partial Differential Equations
- D. Gilbarg, N.S. Trudinger - Elliptic Partial Differential Equations of Second Order

Introduction to Molecular Dynamics

Numerical Simulation and Scientific Computing II

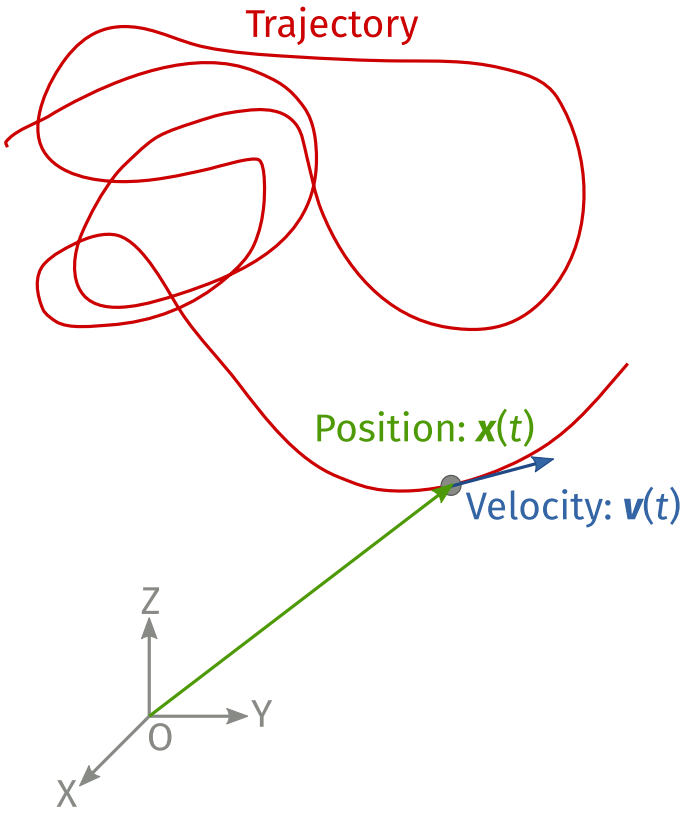
Jesús Carrete Montaña
jesus.carrete.montana@tuwien.ac.at

Lecture 5

TECHNISCHE UNIVERSITÄT WIEN

Prelude: A refresher on Newtonian mechanics

Newtonian mechanics of a single particle



Entity: Structureless, point-like particle with fixed mass m

Goal: Obtain a trajectory $\boldsymbol{x}(t)$

Inputs: Forces and initial conditions

Newton's 2nd law:

$$m\frac{\mathrm{d}^2\boldsymbol{x}}{\mathrm{d}t^2} = \boldsymbol{f}$$

Some vector quantities:

Force: $\boldsymbol{f}$

Position: $\boldsymbol{x}$

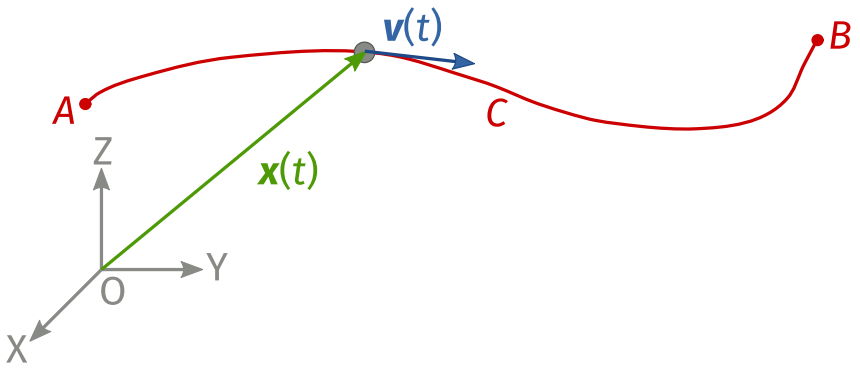
Velocity: $\boldsymbol{v} = \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}$

Acceleration: $\boldsymbol{a} = \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \frac{\mathrm{d}^2\boldsymbol{x}}{\mathrm{d}t^2}$

Linear momentum: $\boldsymbol{p} = m\boldsymbol{v}$

Angular momentum: $\boldsymbol{l} = \boldsymbol{x} \times \boldsymbol{p}$

Work and kinetic energy



The work performed by the force $\boldsymbol{f}$ on the particle as it moves along a trajectory described by the curve C from A to B is defined as the line integral:

$$W_{A\to B}[C] = \int_C \boldsymbol{f}\cdot \mathrm{d}\boldsymbol{x}$$

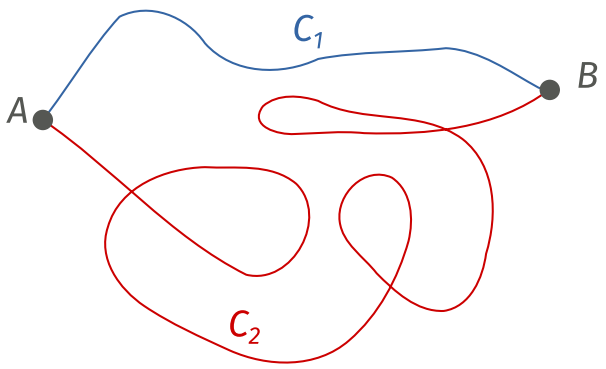
Using the parametrization $\boldsymbol{x}(t)$ for C:

$$W_{A\to B}[C] = \int_{t_A}^{t_B} \boldsymbol{f}\cdot\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}\mathrm{d}t = m\int_{t_A}^{t_B}\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}\cdot\boldsymbol{v}\mathrm{d}t = \int_{t_A}^{t_B}\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{2}m|\boldsymbol{v}|^2\right)\mathrm{d}t$$

The work is equal to the variation of the kinetic energy $\boxed{E_{\mathrm{kin}} = \frac{1}{2}m|\boldsymbol{v}|^2}$ along the trajectory.

Conservative force fields and potential energy

- Force field: Function $\boldsymbol{f} : V \subset \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ assigning a force to each possible position in a subset V of space.
- $\boldsymbol{f}$ is conservative if V is simply connected and $\oint_C \boldsymbol{f} \cdot \mathrm{d}\boldsymbol{x} = 0$ for any closed trajectory C in V.



- For any conservative force field,
$$\oint_{C_1 \cup C_2} \boldsymbol{f} \cdot \mathrm{d}\boldsymbol{x} = 0 = W_{A \to B}[C_2] + W_{B \to A}[C_1]$$
- Therefore, $W_{A \to B}[C]$ is the same for any trajectory C joining A and B

If we choose a point P as the origin of energies,

$$\boxed{E_{\mathrm{pot}}(\boldsymbol{x}) = -W_{P \to \boldsymbol{x}}}$$

is a function of $\boldsymbol{x}$ called the potential energy.

Extracting (conservative) forces from the potential energy

Let us consider one of the directional derivatives of $E_{\text{pot}}(\boldsymbol{x})$ at $\boldsymbol{x}_0$:

$$\left(\frac{\partial E_{\text{pot}}}{\partial y}\right)_{\boldsymbol{x}_0} = \lim_{\Delta y \longrightarrow 0} \frac{-W_{P\to \boldsymbol{x}_0+\hat{\mathbf{y}}\Delta y} + W_{P\to \boldsymbol{x}_0}}{\Delta y} = \lim_{\Delta y \longrightarrow 0} -\frac{W_{\boldsymbol{x}_0\to \boldsymbol{x}_0+\hat{\mathbf{y}}\Delta y}}{\Delta y}$$

Work is independent from the path $\implies$ we choose a straight segment in a natural parametrization: $\boldsymbol{x}(s) = \boldsymbol{x}_0 + \hat{\mathbf{y}}s, s \in [0, \Delta y]$

$$\left(\frac{\partial E_{\text{pot}}}{\partial y}\right)_{\boldsymbol{x}_0} = \lim_{\Delta y \longrightarrow 0} -\frac{1}{\Delta y}\int_0^{\Delta y} \boldsymbol{f}\cdot\hat{\mathbf{y}}\mathrm{d}s = -f_y$$

Valid for all coordinates $\implies$ The force at a point is equal the gradient of the potential energy at that point with its sign reversed

$$\boxed{\boldsymbol{f} = -\nabla E_{\text{pot}}}$$

The conservation of mechanical energy

Two expressions for the work performed between A and B:

- $W_{A\to B} = \Delta_{A\to B} E_{\text{kin}}$
- $W_{A\to B} = -\Delta_{A\to B} E_{\text{pot}}$ (conservative forces only!)

$\Rightarrow \Delta_{A\to B}\big(E_{\text{kin}} + E_{\text{pot}}\big) = 0.$

The total mechanical energy

$$\boxed{E = E_{\text{kin}} + E_{\text{pot}}}$$

is conserved.

Conservation of linear and angular momentum

Conservation of linear momentum: when $\boldsymbol{f} = 0$, then

$$\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = 0 \implies \boldsymbol{p} \text{ is conserved}$$

Conservation of angular momentum: when the torque $\boldsymbol{x} \times \boldsymbol{f}$ is zero, then

$$\frac{\mathrm{d}\boldsymbol{l}}{\mathrm{d}t} = 0 \implies \boldsymbol{l} \text{ is conserved}$$

Important example: conservative central forces, where $E_{\mathrm{pot}}(\boldsymbol{x})$ only depends on $|\boldsymbol{x}| \implies \boldsymbol{f} = -\nabla E_{\mathrm{pot}}$ is parallel to $\boldsymbol{x}$.

Newtonian mechanics of a system of M point particles

Same quantities, defined particle by particle

- Masses $\{m_L\}_{L=1}^{M}$
- Positions $\{\boldsymbol{x}_L\}_{L=1}^{M}$
- Velocities $\{\boldsymbol{v}_L\}_{L=1}^{M}$, with $\boldsymbol{v}_L = \frac{\mathrm{d}\boldsymbol{x}_L}{\mathrm{d}t}$
- Accelerations, forces, linear momenta, angular momenta, etc.

Newton's 2nd law:

$$m_L \frac{\mathrm{d}^2\boldsymbol{x}_L}{\mathrm{d}t^2} = \boldsymbol{f}_L, \; L \in \{1 \dots M\}$$

Mechanical energy for M particles

Work: $W_{A\to B}[C] = \int_C \boldsymbol{f} \cdot \mathrm{d}^{3M}\boldsymbol{l}$, where $\boldsymbol{f}$ includes the $3M$ components of the forces on all particles.

- Kinetic energy: $E_{\mathrm{kin}} = \sum_{L=1}^{M} \frac{1}{2} m_L |\boldsymbol{v}_L|^2$
- Potential energy (only for conservative forces!): $E_{\mathrm{pot}} = -W_{P\to\boldsymbol{x}}$, where $\boldsymbol{x}$ contains all positions $\implies$ not a sum over particles

If a potential energy can be defined, $\boldsymbol{f}_L = -\boldsymbol{\nabla}_{\boldsymbol{x}_L} E_{\mathrm{pot}}$ for each particle.

Conservation laws for M particles

Conservation of energy: For conservative forces, the total mechanical energy $E_{\mathrm{kin}} + E_{\mathrm{pot}}$ is conserved.

Conservation of linear momentum: if the net force on the system $\sum_L \boldsymbol{f}_L$ is zero, the total linear momentum $\sum_L \boldsymbol{p}_L$ is conserved.

Conservation of angular momentum: if the net torque $\sum_L \boldsymbol{x}_L \times \boldsymbol{f}_L$ is zero, the total angular momentum $\sum_L \boldsymbol{x}_L \times \boldsymbol{p}_L$ is conserved.

In more advanced formulations of mechanics, these conservation laws emerge directly from the fundamental symmetries of free space: translation symmetry ($\boldsymbol{p}$), rotation symmetry ($\boldsymbol{l}$) and time translation invariance (E).

For a more in-depth refresher

- H. Goldstein, C. P. Poole & J. L. Safko, *Classical Mechanics*, 3rd edition, Wesley (2002). ISBN: 0201657023
- T. T. Thornton & J. B. Marion, *Classical Dynamics of Systems and Particles*, 5th edition, Brooks/Cole (2004). ISBN: 0534408966

Introduction

Molecular dynamics (MD)

Narrow definition:

- Family of atomistic simulation methods
- State of "atoms" described by:
  1. Positions $\{\boldsymbol{x}_L\}_{L=1}^{M}$
  2. Velocities $\left\{\boldsymbol{v}_L = \frac{\mathrm{d}\boldsymbol{x}_L}{\mathrm{d}t}\right\}_{L=1}^{M}$
  3. Internal variables (*e.g.* electric dipole moment)
- Numerical integration of Newton's equations of motion for conservative forces:
  $m_L \frac{\mathrm{d}^2\boldsymbol{x}_L}{\mathrm{d}t^2} = -\boldsymbol{\nabla}_{\boldsymbol{x}_L} E_{\mathrm{pot}} \Leftrightarrow m_L \frac{\mathrm{d}^2 x_L^{(\alpha)}}{\mathrm{d}t^2} = -\frac{\partial E_{\mathrm{pot}}}{\partial x_L^{(\alpha)}}$ where $\alpha \in \{x, y, z\}$
- Result: trajectories in phase space $\{\boldsymbol{x}_L(t), \boldsymbol{v}_L(t)\}_{L=1}^{M}$

Wait a minute...

Isn't classical mechanics wrong?

- Wronger than some things, less wrong than others [I. Asimov, *The Relativity of Wrong*, Doubleday (1988). ISBN: 0385244738]
- Better question: Can it be applied here?

The challenges:

Relativistic effects:
- Weak as long as $|\boldsymbol{v}_L| \ll c$ and no intense gravitational fields involved
- (Mostly) not very relevant for our simulations

Quantum mechanics:
- Fundamental physics of electrons
- Totally necessary at the atomistic level
- Effects survive in the macroscale: indistinguishability, quantization...
- Underpins all of materials science

Is MD even worth discussing?

The ~~sad~~ honest answer

A proper quantum-mechanical treatment of large system is unfeasible.
Even for a stationary solution:

"Quantum-chemical" (wave-function-based) methods:

- Exponential barrier (curse of dimensionality)
- $\lesssim$ Hundreds of electrons

Density functional theory:

- Overcomes the exponential barrier
- Still $\lesssim$ thousands of atoms

Time-dependent quantum methods are even more constrained

$\Longrightarrow$ Until conceptual and algorithmic revolutions,
MD is here to stay.

Some support from quantum mechanics

Born-Oppenheimer approximation:

- The dynamics of electrons and nuclei can be decoupled (conditions apply)
- Fix the nuclei, solve for the electrons, get effective potential energy

Ehrenfest theorem:

$$\frac{\mathrm{d}\langle \boldsymbol{p}_L \rangle}{\mathrm{d}t} = -\boldsymbol{\nabla}_{\boldsymbol{x}_L} E_0$$

$\Rightarrow$ Newton's 2nd law holds in an average sense and a potential energy for the atomic nuclei can be defined.

Ab-initio (Born-Oppenheimer) molecular dynamics

Recipe to generate a trajectory:

Initialization

1. Fix a set of positions $\{\boldsymbol{x}_L\}$ for nuclei

Main loop

1. Obtain the electronic ground state energy $E_0(\{\boldsymbol{x}_L\})$
2. Use E_0 an effective potential energy for the nuclei
3. Compute its derivatives (forces)
4. Integrate equations of motion
5. Update coordinates and velocities

Extremely useful, but

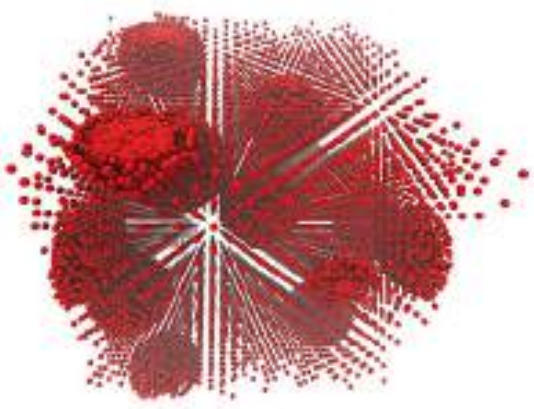
- Limited by the quantum solver step
- Cannot reproduce quantum effects on the nuclei

Classical molecular dynamics

Orientative limits of ab-initio molecular dynamics:

- Hundreds of atoms
- Some picoseconds

Classical molecular dynamics is based on a parametrization of the potential energy, abstracting away the electrons.



Reference: M. Bernreuther *et al.*, *Innovative HPC Methods and Application to Highly Scalable Molecular Simulation (IMEMO)*, Inside 10 (2012) 50:

- Simulation of nucleation
- 4.13×10^{12} particles

Classical MD simulations with $10^5 - 10^6$ particles are within everyone's reach.

The ingredients of MD

The ingredients

At least four elements are necessary for an MD study:

1. A way to construct $E_{\mathrm{pot}}(\boldsymbol{x}_L)$ (a potential)
2. An integration scheme
3. Boundary conditions (in a general sense)
4. A method to extract information from the trajectory

Wish list for a numerical integrator

Indispensable compatibility with classical mechanics:

- Time-reversible
- Symplectic (preserves phase-space volumes)
- Energy- and momentum-conserving

Desirable properties:

- Stable (no exponential deviation from actual solution)
- Fast
- No high-order derivatives or additional evaluations (E_{pot} **expensive**, $> 85\%$ of simulation time)

Good to have:

- Accurate
- Compact in terms of memory

A first attempt

Taylor expansion based on the available information at time t:

The Euler integrator

$$\boldsymbol{x}(t+\Delta t) = \boldsymbol{x}(t) + \boldsymbol{v}(t)\Delta t + \frac{1}{2}\boldsymbol{a}(t)(\Delta t)^2$$
$$\boldsymbol{v}(t+\Delta t) = \boldsymbol{v}(t) + \boldsymbol{a}(t)\Delta t$$

See results in Elliptical orbit.ipynb

The velocity Verlet integrator

With just a small change:

$$\boldsymbol{x}(t+\Delta t) = \boldsymbol{x}(t) + \boldsymbol{v}(t)\Delta t + \frac{1}{2}\boldsymbol{a}(t)(\Delta t)^2$$

$$\boldsymbol{v}(t+\Delta t) = \boldsymbol{v}(t) + \frac{\boldsymbol{a}(t) + \boldsymbol{a}(t+\Delta t)}{2}\Delta t$$

See results in Elliptical_orbit.ipynb

Questionable error analysis

The velocity Verlet algorithm has a more favorable truncation error:

Taylor:

$$\boldsymbol{x}(t+\Delta t) = \boldsymbol{x}(t) + \boldsymbol{v}(t)\Delta t + \frac{1}{2}\boldsymbol{a}(t)(\Delta t)^2 + \mathcal{O}\left[(\Delta t)^3\right]$$
$$\boldsymbol{v}(t+\Delta t) = \boldsymbol{v}(t) + \boldsymbol{a}(t)\Delta t + \mathcal{O}\left[(\Delta t)^2\right]$$

Velocity Verlet:

$$\boldsymbol{x}(t+\Delta t) = \boldsymbol{x}(t) + \boldsymbol{v}(t)\Delta t + \frac{1}{2}\boldsymbol{a}(t)(\Delta t)^2 + \mathcal{O}\left[(\Delta t)^3\right]$$
$$\boldsymbol{v}(t+\Delta t) = \boldsymbol{v}(t) + \frac{\boldsymbol{a}(t)+\boldsymbol{a}(t+\Delta t)}{2}\Delta t + \boxed{\mathcal{O}\left[(\Delta t)^3\right]}$$

Proof (I)

Expansion of $\boldsymbol{v}$ around $t + \frac{\Delta t}{2}$, evaluated at t and $t + \Delta t$:

$$\boldsymbol{v}(t) = \boldsymbol{v}\left(t + \frac{\Delta t}{2}\right) - \boldsymbol{a}\left(t + \frac{\Delta t}{2}\right)\frac{\Delta t}{2} + \frac{1}{2}\boldsymbol{j}\left(t + \frac{\Delta t}{2}\right)\left(\frac{\Delta t}{2}\right)^2 + \mathcal{O}\left[(\Delta t)^3\right]$$
$$\boldsymbol{v}(t + \Delta t) = \boldsymbol{v}\left(t + \frac{\Delta t}{2}\right) + \boldsymbol{a}\left(t + \frac{\Delta t}{2}\right)\frac{\Delta t}{2} + \frac{1}{2}\boldsymbol{j}\left(t + \frac{\Delta t}{2}\right)\left(\frac{\Delta t}{2}\right)^2 + \mathcal{O}\left[(\Delta t)^3\right],$$

where $\boldsymbol{j} = \frac{\mathrm{d}^3\boldsymbol{x}}{\mathrm{d}t^3}$ (called "jerk"). Subtracting both:

$$\boldsymbol{v}(t + \Delta t) = \boldsymbol{v}(t) + \boldsymbol{a}\left(t + \frac{\Delta t}{2}\right)\Delta t + \mathcal{O}\left[(\Delta t)^3\right]$$

Proof (II)

For the acceleration:

$$\boldsymbol{a}(t) = \boldsymbol{a}\left(t + \frac{\Delta t}{2}\right) - \boldsymbol{j}\left(t + \frac{\Delta t}{2}\right)\frac{\Delta t}{2} + \mathcal{O}\left[(\Delta t)^2\right]$$
$$\boldsymbol{a}(t + \Delta t) = \boldsymbol{a}\left(t + \frac{\Delta t}{2}\right) + \boldsymbol{j}\left(t + \frac{\Delta t}{2}\right)\frac{\Delta t}{2} + \mathcal{O}\left[(\Delta t)^2\right]$$

Adding both together:

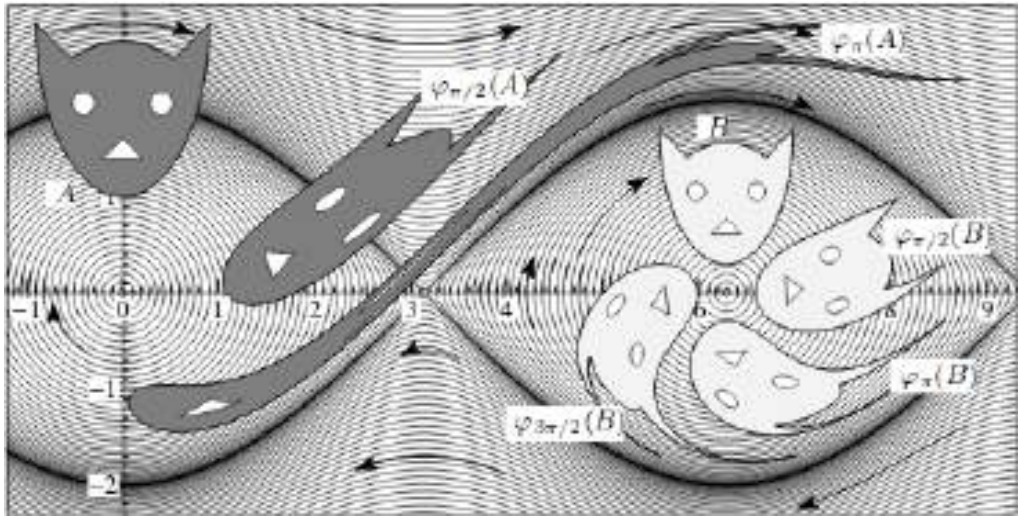
$$\boldsymbol{a}\left(t + \frac{\Delta t}{2}\right) = \frac{\boldsymbol{a}(t) + \boldsymbol{a}(t + \Delta t)}{2} + \mathcal{O}\left[(\Delta t)^2\right]$$

Substituting this into the expansion of $\boldsymbol{v}$:

$$\boldsymbol{v}(t + \Delta t) = \boldsymbol{v}(t) + \frac{\boldsymbol{a}(t) + \boldsymbol{a}(t + \Delta t)}{2}\Delta t + \mathcal{O}\left[(\Delta t)^3\right]$$

The real reason why velocity Verlet works

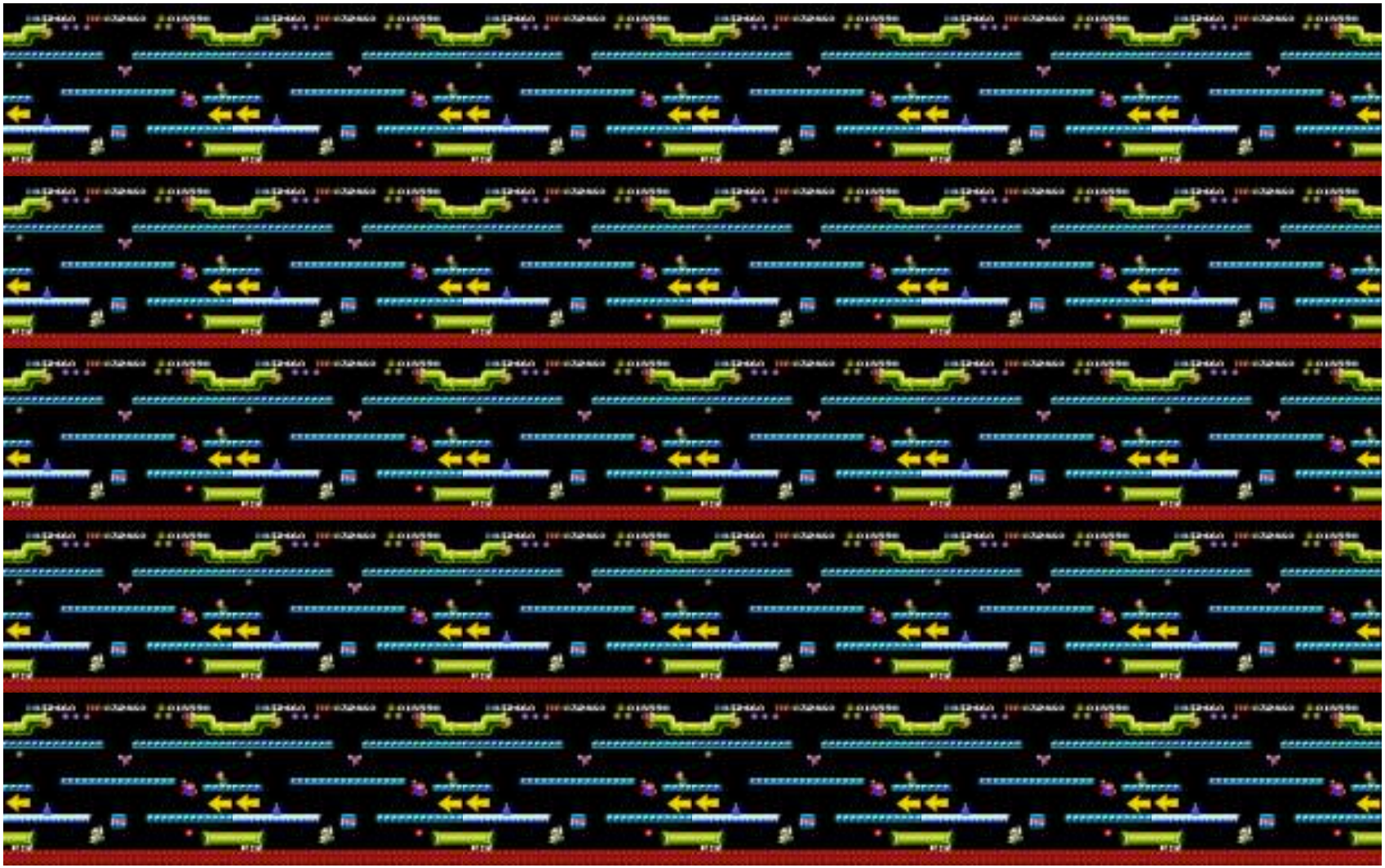
- A velocity Verlet step is a symplectic transformation. Generalization of the idea of an area-preserving map in (x, p) space:



- Systematic study and design: See *Vorlesung "Geometric Numerical Integration" an der TU München (2010)*, `http://www.unige.ch/~hairer/polycop.html`
- E.g.: symplectic Runge-Kutta

Periodic boundary conditions

Periodic boundary conditions



Periodic boundary conditions

MD is powerful and scalable, but cannot tackle the number of molecules that make up a macroscopic phase. Periodic boundary conditions fold the coordinates into the simulation box, so that there are infinite interacting copies of it.

Periodic boundary conditions, in practice

A particle L in the central copy of the simulation box interacts with all the images of each other particle J in all copies of the simulation box. To implement this:

Long-range potentials:

The sum over images is implemented partly in real space and partly in Fourier space (Ewald-like methods) or using the fast multipole method (FMM). Notable example: electrostatic interactions.

Short-range potentials:

Minimum-image convention – only the image of J closer to L is taken into account. Examples: potentials with cutoffs, quickly decreasing functions.

Most potentials can be split into a long-range and a short-range part.

Implementing the minimum-image convention

Completely general case:

- Simulation box: general space-filling shape
- Related to the closest vector problem (class of lattice problem)
- Probably NP-hard, slow, best avoided

Easy, frequent case:

- Simulation box: parallelepiped with sides L_x, L_y and L_z
- Let $\Delta\boldsymbol{x} = \boldsymbol{x}_L - \boldsymbol{x}_J$ be a vector joining L with any arbitrary image of J
- $\Delta\boldsymbol{x} = (\Delta x, \Delta y, \Delta z)$ is reduced to its value in the minimum image convention simply through:

$$\left.\begin{aligned} \Delta x_{\mathrm{m.i.}} &= \Delta x - L_x \left\lfloor \frac{\Delta x}{L_x} \right\rceil \\ \Delta y_{\mathrm{m.i.}} &= \Delta y - L_y \left\lfloor \frac{\Delta y}{L_y} \right\rceil \\ \Delta z_{\mathrm{m.i.}} &= \Delta z - L_z \left\lfloor \frac{\Delta z}{L_z} \right\rceil \end{aligned}\right\}, \text{ where } \lfloor z \rceil \equiv \mathrm{round}(z)$$

Boundary conditions: constant T or p

- Newton's equations $\Rightarrow (N, V, E)$ simulation
- Need modifications for (N, V, T) or (N, p, T)
- E.g.: Thermostats must affect the statistical distribution of velocities. Many deterministic and random approaches exist.

A rescaling thermostat

$$\boldsymbol{v}'_L = \boldsymbol{v}_{\mathrm{CM}} + \alpha\left(\boldsymbol{v}_L - \boldsymbol{v}_{\mathrm{CM}}\right)$$

$$\alpha = \sqrt{1 \pm \frac{\Delta E}{E_{\mathrm{kin},R}}}$$

$$E_{\mathrm{kin},R} = \frac{1}{2}\sum_L m_L \left|\boldsymbol{v}_L\right|^2 - \frac{1}{2}\sum_L m_L \left|\boldsymbol{v}_{\mathrm{CM}}\right|^2$$

- For systems with surfaces, boundary effects have to be considered
- Equilibrium or non-equilibrium? Imposed gradients

Post-processing: Averages

- For theoretical foundations, see e.g. D. Chandler, *Introduction to Modern Statistical Mechanics*, Oxford (1987). ISBN: 9780195042771.
- Underlying assumption: time averages $\frac{1}{t_{\max}} \int_0^{t_{\max}} \bullet \, \mathrm{d}t$ can approximate statistical averages over the state space of the system $\langle \bullet \rangle$.

Examples:

Geometric/mechanical:

$$\rho(\boldsymbol{x}) = \lim_{V \longrightarrow \{\boldsymbol{x}\}} \frac{1}{V} \int\limits_V \sum_L \langle \delta(\boldsymbol{x}' - \boldsymbol{x}_L) \rangle \mathrm{d}^3 \boldsymbol{x}'$$

Thermodynamical:

$$T = \frac{\langle E_{\mathrm{kin}} \rangle}{\frac{3}{2} M k_{\mathrm{B}}}$$

($k_{\mathrm{B}} = 1.380\,649 \times 10^{-23}$ J K^{-1}: Boltzmann constant)

Post-processing: Fluctuations (standard deviations)

Connected with "linear response" / second derivatives of the entropy: specific heats, thermal expansion coefficients, compressibilities...

E.g.: isobaric specific heat

$$c_v = \frac{1}{M}\left(\frac{\partial E}{\partial T}\right)_V = k_\mathrm{B}\frac{\langle E_\mathrm{kin}\rangle^2}{\sigma^2_{E_\mathrm{kin}}}, \text{ where } \sigma^2_{E_\mathrm{kin}} = \left\langle (E_\mathrm{kin} - \langle E_\mathrm{kin}\rangle)^2 \right\rangle$$

Simple calculation strategy: $\sigma_f^2 = \langle f^2\rangle - \langle f\rangle^2$.

Post-processing: Spatial correlations

Example: pair correlation function between atom types a and b

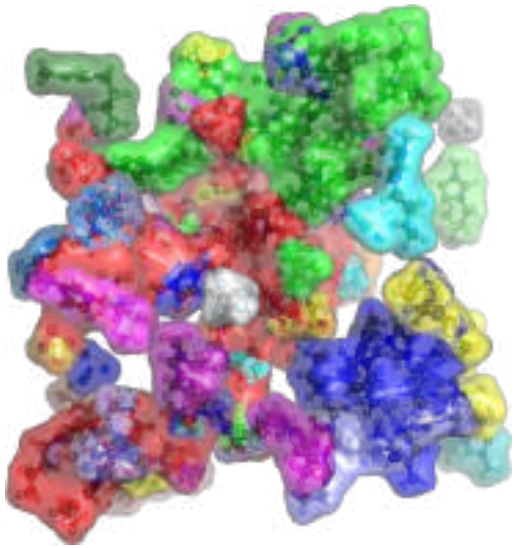
$$g_{ab}(r) = \frac{1}{4\pi r^2} \frac{V}{M_a M_b} \sum_{L=1}^{M_a} \sum_{J=1}^{M_b} \langle \delta(|\boldsymbol{x}_L - \boldsymbol{x}_J| - r) \rangle$$

Interpretation: density of particles of type b at a distance r from a particule of type a, relative to a system with completely random positions.

First order of description of the structure of a fluid:

- Peaks at nearest-neighbor distances
- Can be integrated to measure coordination
- Useful for detecting homo-/heterocoordination

Post-processing: Cluster formation



Method:

- Define a notion of connectivity (distances, angles...)
- Build a graph

Clusters are extracted as connected components (depth-first or breadth-first search). A giant connected component can signal percolation (interesting in e.g. ionic systems).

Post-processing: Time autocorrelation functions

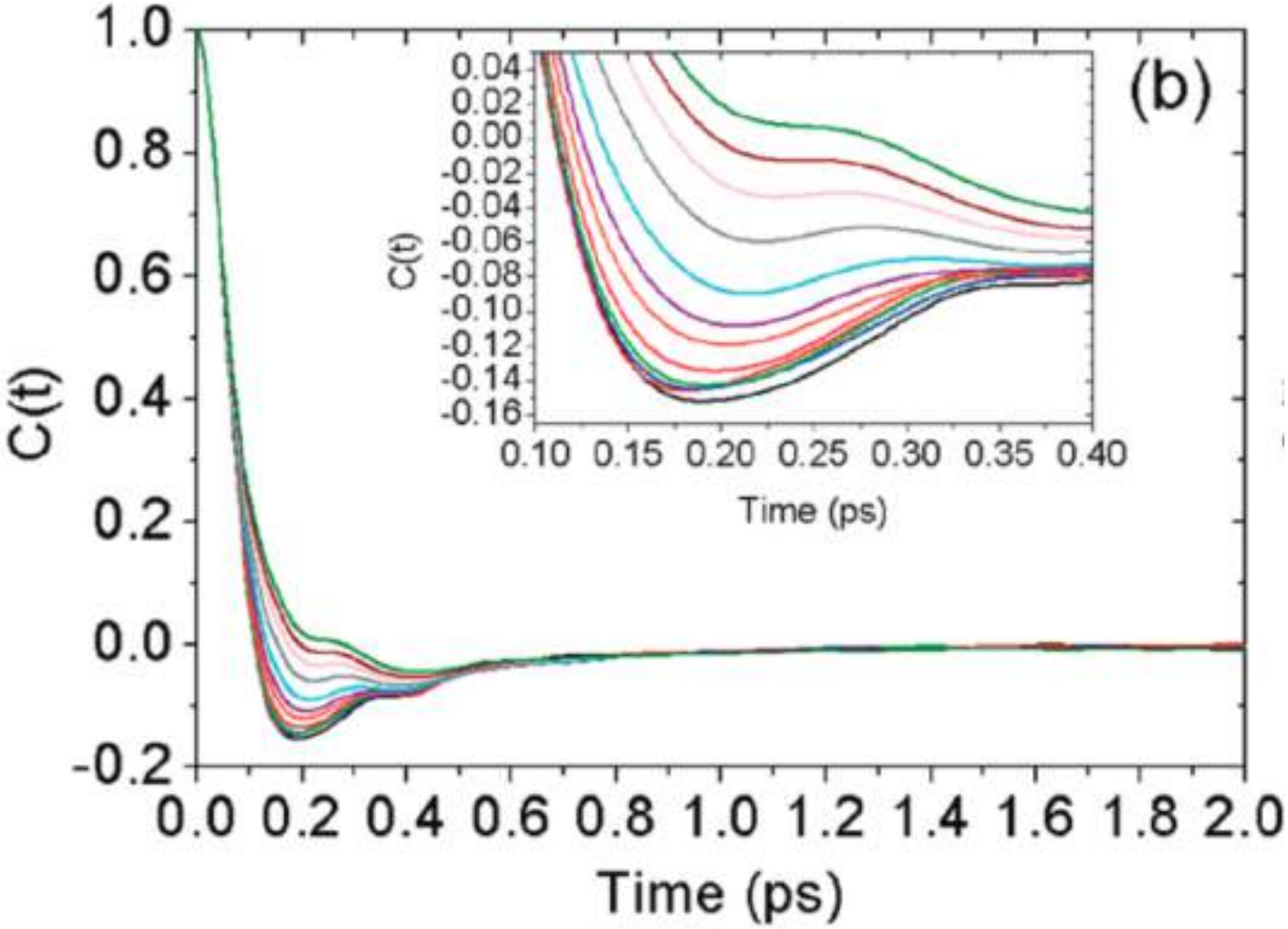
The Green-Kubo relations provide formulas for transport coefficients like the thermal conductivity tensor:

$$\kappa^{(\alpha\beta)} = \frac{V}{k_\mathrm{B} T^2} \int_0^\infty \left\langle j_Q^{(\alpha)}(0) j_Q^{(\beta)}(t) \right\rangle \mathrm{d}t,$$ where $\boldsymbol{j}_Q$ is the heat current

Particularly difficult to converge! Two possibilities:

- Direct calculation (discretization and cutoff in the integral)
- Calculation in frequency space (FFT, correlation theorem, filters to minimize aliasing and noise)

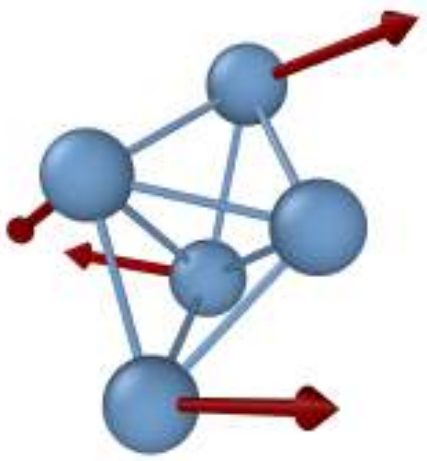
Post-processing: Relaxation times



Connected to confinement, time to first collision...

Post-processing: Normal modes

Low-energy, bounded movements of the system (generalization of molecular vibrations):



Their frequencies create peaks in the power spectrum of the velocity time-autocorrelation function.

Post-processing: Infinite diversity

- Free energy profiles
- Cage correlations
- Hydrogen bonds
- Diffusion
- Dielectric response
- Protein folding ...

A survey of available software

Popular MD software

Ab initio:

- Most DFT software, both commercial (e.g. VASP) and free (e.g. Quantum ESPRESSO)
- CP2K: `http://www.cp2k.org`

Classical and mixed:

- LAMMPS: `http://lammps.sandia.gov`
- OpenMD: `http://www.openmd.org`
- NAMD Scalable Molecular Dynamics: `http://www.ks.uiuc.edu/Research/namd/`
- GROMACS FAST. FLEXIBLE. FREE.: `http://www.gromacs.org`

Steps of a practical MD simulation

At the very least:

1. Particle placement (random algorithms, pre-solvated boxes...)
2. Energy relaxation
3. Stabilization run
4. Production run
5. Post-processing

Possibly: equilibrium and non-equilibrium steps, parameter sweep, ”pull” simulation, particle addition and removal, hybrid quantum-mechanical/molecular-mechanics...

Introduction to Molecular Dynamics

Numerical Simulation and Scientific Computing II

Jesús Carrete Montaña
jesus.carrete.montana@tuwien.ac.at

Lecture 5

TECHNISCHE UNIVERSITÄT WIEN

Interatomic potentials

Numerical Simulation and Scientific Computing II

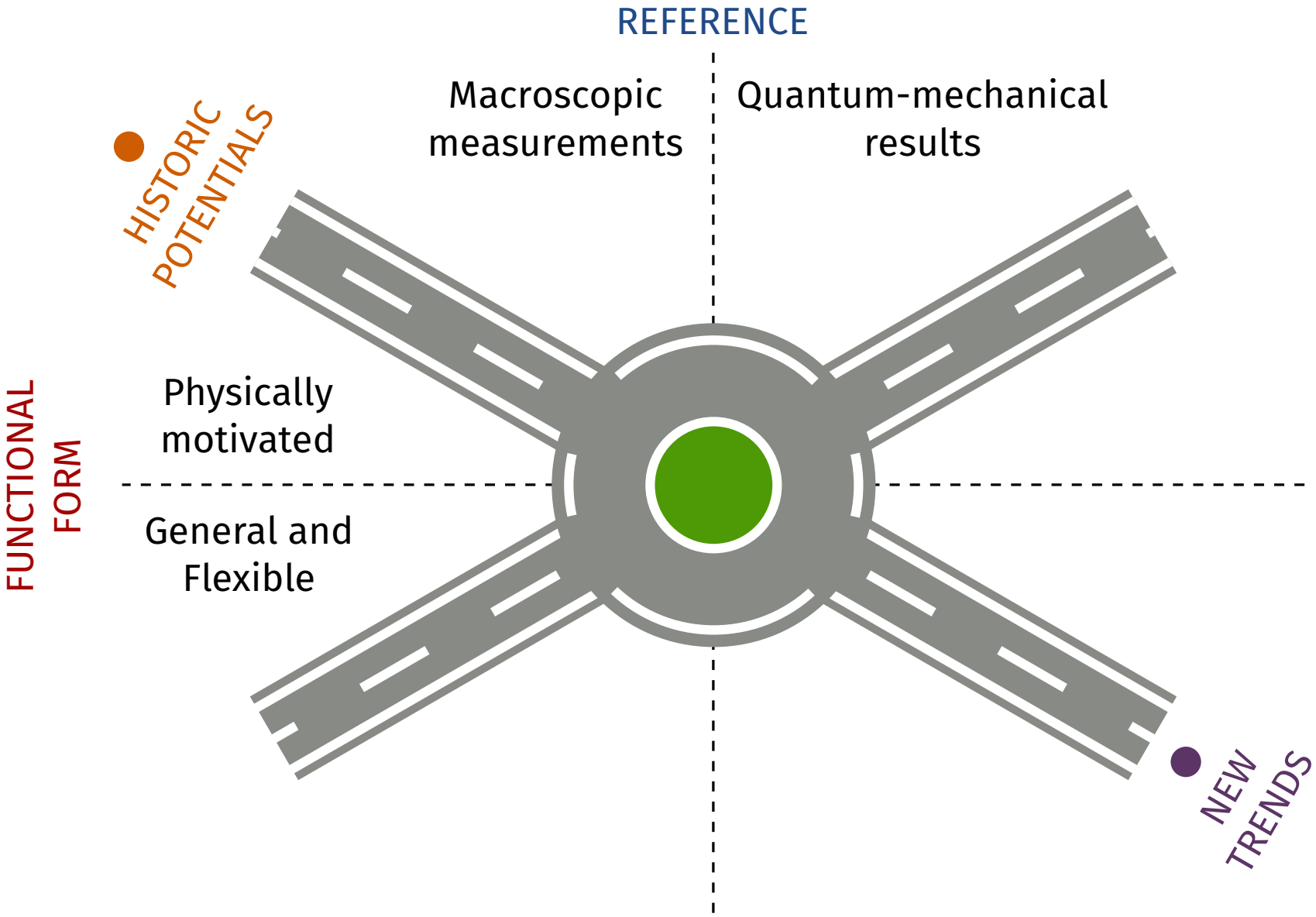
Jesús Carrete Montaña
jesus.carrete.montana@tuwien.ac.at

Lecture 6

TECHNISCHE UNIVERSITÄT WIEN

Introduction

Different approaches to E_{pot}



Physical constraints

Any reasonable potential must reflect the fundamental symmetries of mechanics:

- Translation: $E_{\text{pot}}(\{\boldsymbol{x}_I + \boldsymbol{A}\}) = E_{\text{pot}}(\{\boldsymbol{x}_I\}) \forall \boldsymbol{A} \in \mathbb{R}^3$
- Rotation: $E_{\text{pot}}(\{\mathcal{R}\boldsymbol{x}_I\}) = E_{\text{pot}}(\{\boldsymbol{x}_I\}) \forall \mathcal{R} \in \mathrm{SO}(3)$

...in the absence of external fields.

Not valid potentials:

- $E_{\text{pot}}(\boldsymbol{x}) = -\frac{K}{|\boldsymbol{x}|}$
- $E_{\text{pot}}(\boldsymbol{x}_1, \boldsymbol{x}_2) = K(\boldsymbol{x}_1 + \boldsymbol{x}_2)$
- $E_{\text{pot}}(\boldsymbol{x}_1, \boldsymbol{x}_2) = K(\boldsymbol{x}_1 \cdot \boldsymbol{x}_2)$
- $E_{\text{pot}}(\boldsymbol{x}_1, \boldsymbol{x}_2) = K(y_1 - y_2)$

Furthermore:

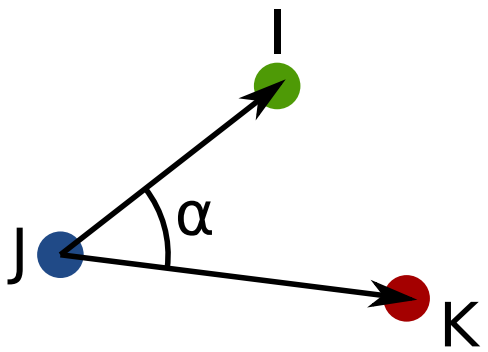
- No arbitrary labeling of identical particles
- No spurious singularities

n-body potentials

A classic, systematic approach

$$E_{\text{pot}}(\{\boldsymbol{x}_I\}) = V_0 + \sum_I V_1(\boldsymbol{x}_I) + \sum_{I,J\neq I} V_2(\boldsymbol{x}_I, \boldsymbol{x}_J) + \sum_{\substack{I,J\neq I \\ K\notin\{I,J\}}} V_3(\boldsymbol{x}_I, \boldsymbol{x}_J, \boldsymbol{x}_K) + \sum_{\substack{I,J\neq I \\ K\notin\{I,J\} \\ L\notin\{I,J,K\}}} V_4(\boldsymbol{x}_I, \boldsymbol{x}_J, \boldsymbol{x}_K, \boldsymbol{x}_L) + \ldots$$

- Rarely extended beyond V_4
- Usually based on explicit scalars like:



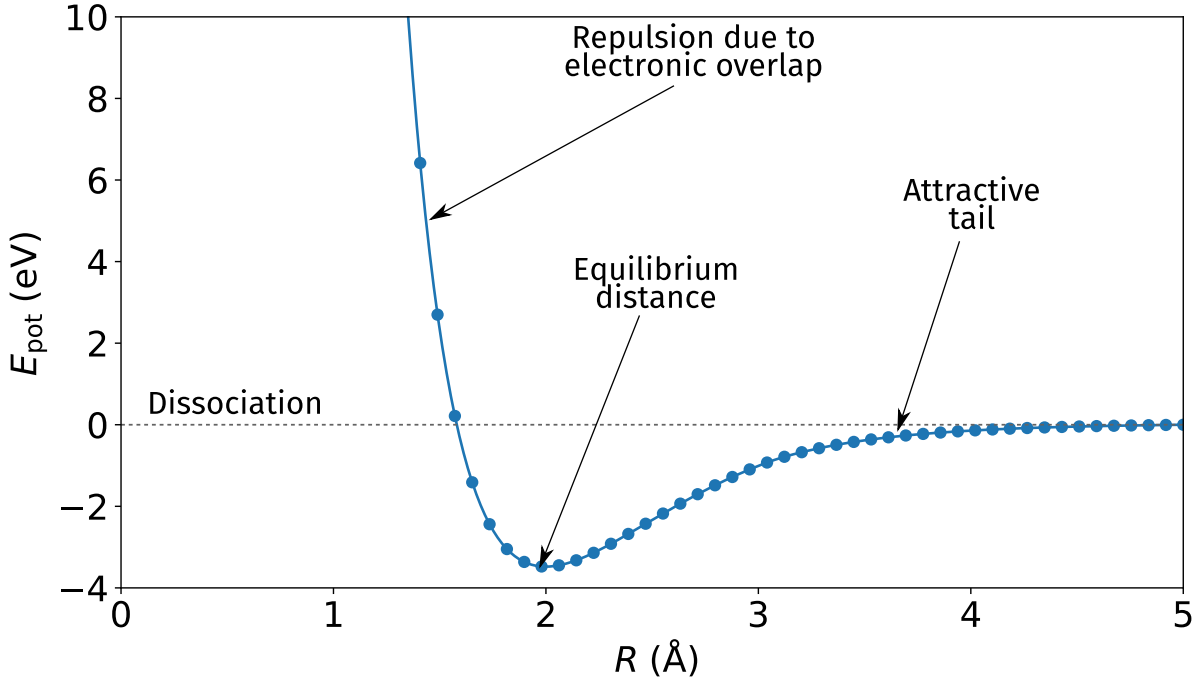
$$R_{IJ} = |\boldsymbol{x}_I - \boldsymbol{x}_J|$$

$$\cos\alpha_{IJK} = \frac{(\boldsymbol{x}_I - \boldsymbol{x}_J)\cdot(\boldsymbol{x}_K - \boldsymbol{x}_J)}{R_{IJ}R_{JK}}$$

Some two-body potentials

Traditional structure

Functional form: $V_2(R)$. Typical shape (DFT energies, Cl_2 dimer):



Some handy Lego blocks:

- Harmonic
- Coulomb
- Dipole-dipole
- Screened Coulomb
- Correlation between dipoles (London)

The Lennard-Jones potential (1924)

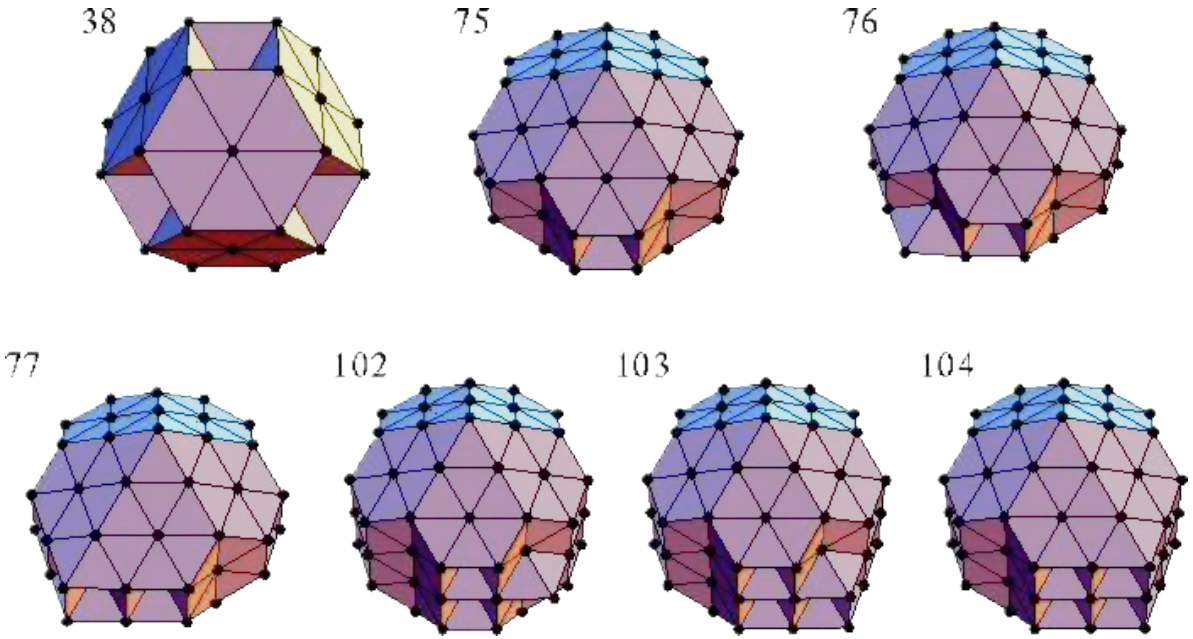
A classic that lives on

$$V_2(R) = 4\epsilon\left[\left(\frac{\sigma}{R}\right)^{12} - \left(\frac{\sigma}{R}\right)^{6}\right]$$

- Exchange interaction + dipole correlation
- Developed for noble gases
- Parameters: ϵ and $\sigma \Rightarrow$ attraction and repulsion interlinked

Rich behaviour from a simple model

- Bulk Lennard-Jones systems adopt close-packing structures
- Finding the ground states of LJ clusters took decades and intense algorithmic development!
- Cambridge energy landscape database: `http://www-wales.ch.cam.ac.uk/CCD.html`



The Buckingham potential (1938)

A much more expensive proposal (at the time)

$$V_2(R) = Ae^{-BR} - \frac{C}{R^6}$$

- Three parameters $\Rightarrow$ more flexible
- Often extended with a Coulomb term

The Morse potential (1929)

A very familiar expression

$$V_2(R) = D_e \left\{ \left[1 - e^{-\alpha(R-R_c)}\right]^2 - 1 \right\}$$

- Quite standard in chemistry (works for many diatomic systems)
- Suitable for analytical treatments of *e.g.* anharmonicity
- D_e: minimum potential energy ($\neq$ dissociation energy!)
- R_c: position of the minimum

The Morse/long-range potential (2009)

A very different story for modern times

$$V_2(R) = D_e \left\{ \left[1 - \frac{u(R)}{u(R_c)} e^{-\beta(R) y(r)} \right]^2 - 1 \right\}$$

where

$$y(R) = \frac{R^p - R_c^p}{R^p + R_c^p}, \text{ with } p \in \mathbb{Z}^+ - \{1\}$$

$$\beta(R) = y(R)\beta_\infty + [1 - y(R)] \sum_{i=0}^{N_{\text{short}} \text{ or } N_{\text{long}}} \beta_i y(R)^i$$

$$u(R) = \sum_i \frac{C_{m_i}}{R^{m_i}}, \text{ with } \{m_i\} \text{ dictated by symmetries}$$

- Standard in modern spectroscopic applications
- *Ab-initio*-like accuracy
- Directly fitted to spectroscopic data: J. Chem. Phys. 131 (2009) 204309

Combination rules for multicomponent systems

When there are several types of atoms $a \in \{1, 2 \dots c\}$:

$$E_{\text{pot}}(\{\boldsymbol{x}_I\}) = \sum_{a,b} \sum_{\substack{I \in a \\ J \in b}} V_2^{(ab)}(R_{IJ})$$

There are $\frac{c(c+1)}{2}$ functions to parameterize $\Rightarrow$ very inconvenient.

Combination rules based on "physical interpretations" of each parameter. For Lennard-Jones:

Lorentz-Berthelot

$$\sigma^{(ab)} = \frac{\sigma^{(a)} + \sigma^{(b)}}{2}$$

$$\epsilon^{(ab)} = \sqrt{\epsilon^{(a)}\epsilon^{(b)}}$$

Waldman-Hagler

$$\sigma^{(ab)} = \left\{ \frac{\left[\left(\sigma^{(a)}\right)^6 + \left(\sigma^{(b)}\right)^6\right]}{2} \right\}^{1/6}$$

$$\epsilon^{(ab)} = \sqrt{\epsilon^{(a)}\epsilon^{(b)}}$$

GROMACS

$$\sigma^{(ab)} = \sqrt{\sigma^{(a)}\sigma^{(b)}}$$

$$\epsilon^{(ab)} = \sqrt{\epsilon^{(a)}\epsilon^{(b)}}$$

Kong

$$\epsilon^{(ab)}\left(\sigma^{(ab)}\right)^6 = \sqrt{\epsilon^{(a)}\left(\sigma^{(a)}\right)^6 \epsilon^{(b)}\left(\sigma^{(b)}\right)^6}$$

$$\epsilon^{(ab)}\left(\sigma^{(ab)}\right)^{12} = \left\{ \frac{\sqrt[13]{\epsilon^{(a)}\left(\sigma^{(a)}\right)^{12}} + \sqrt[13]{\epsilon^{(b)}\left(\sigma^{(b)}\right)^{12}}}{2} \right\}^{13}$$

Uncontrolled approximation: See *e.g.* J. Phys. Chem. B 119 (2015) 14563

Fitting a pair potential

Please see Fitting_a_pair_potential.ipynb

The moral of the story: Transferability

Potentials parameterized for

- Structure
- Energetics
- Vibrations
- Anharmonicity
- Nanostructures
- Surfaces
- Bulk systems
- Solid phases
- Liquids

Often fail to describe

- Structure
- Energetics
- Vibrations
- Anharmonicity
- Nanostructures
- Surfaces
- Bulk systems
- Solid phases
- Liquids

Example: Transferability of water models

Calculated physical properties of the water models

Model	Dipole moment e	Dielectric constant	self-diffusion, 10^{-5} cm^2/s	Average configurational energy, kJ × mol^{-1}	Density maximum, °C	Expansion coefficient, 10^{-4} °C^{-1}
SSD	2.35 [511]	72 [511]	2.13 [511]	-40.2 [511]	-13 [511]	-
SPC	2.27 [181]	65 [185]	3.85 [182]	-41.0 [185]	-45 [983]	7.3 [704] **
SPC/E	2.35 [3]	71 [3]	2.49 [182]	-41.5 [3]	-38 [183]	5.14 [994]
SPC/Fw	2.39 [994]	79.63 [994]	2.32 [994]	-	-	4.98 [994]
PPC	2.52 [3]	77 [3]	2.6 [3]	-43.2 [3]	+4 [184]	-
TIP3P	2.35 [180]	82 [3]	5.19 [182]	-41.1 [180]	-91 [983]	9.2 [180]
TIP3P/Fw	2.57 [994]	193 [994]	3.53 [994]	-	-	7.81 [994]
iAMOEBA	2.78 [2031]	80.7 [2031]	2.54 [2031]	-	4 [2031]	2.5 [2031]
uAMOEBA	2.80 [2401]	76.3 [2401]	2.41 [2401]	-	-	3.38 [2401]
QCT **	1.85 [1251]	-	1.5 [1251]	-42.7 [1251]	+10 [1251]	3.5 [1251]
TIP4P	2.18 [3,180]	53 a [3]	3.29 [182]	-41.8 [180]	-25 [180]	4.4 [180]
TIP4P-Ew	2.32 [649]	62.9 [649]	2.4 [649]	-46.5 [649]	+1[649]	3.1[649]
TIP4P-FQ	2.64[197]	79 [197]	1.93 [197]	-41.4 [201]	+7 [197]	-
TIP4P/2005	2.305 [984]	60 [984]	2.08 [984]	-	+5 [984]	2.8 [984]
TIP4P/2005f	2.319 [1765]	55.3 [1765]	1.93 [1765]	-	+7 [1765]	-
TIP4P/ε	2.4345 [2444]	78.3 [2444]	2.10 [2444]	-	+4 [2444]	-
OPC	2.48 [2168]	78.4 [2168]	2.3 [2168]	-	-1 [2168]	2.7 [2168]
OPC3	2.43 [2722]	78.4 [2722]	2.3 [2722]	-	-13 [2722]	4.3 [2722]
SWFLEX-AI	2.69 [201]	116 [201]	3.66 [201]	-41.7 [201]	-	-
COS/G3 **	2.57 [704]	88 [704]	2.6 [704]	-41.1 [704]	-78[1939]	7.0 [704]
COS/D2	2.55 [1617]	78.9 [1617]	2.2 [1617]	-41.8 [1617]	-	4.9 [1617]
GCPM	2.723 [859]	84.3 [859]	2.26 [859]	-44.8 [859]	-13 [859]	-
SWM4-NDP	2.461 [933]	79 [933]	2.33 [933]	-41.5 [933]	<-53 [1999]	-
BK3	2.644 [2080]	79 [2080]	2.28 [2080]	-43.32 [2080]	+4 [2080]	3.01 [2080]
SWM6	2.431 [1999]	78.1 [1999]	2.14 [1999]	-41.5 [1999]	-48 [1999]	-
TIP5P	2.29 [180]	81.5 [180]	2.62 [182]	-41.3 [180]	+4 [180]	6.3 [180]
TIP5P-Ew	2.29 [619]	92 [619]	2.8 [619]	-	+8 [619]	4.9[619]
TTM2-F	2.67 [1027]	67.2 [1027]	1.4 [1027]	-45.1 [1027]	-	-
POL5/TZ	2.712 [256]	98 [256]	1.81 [256]	-41.5 [256]	+25 [256]	-
Six-site *	1.89 [491]	33 [491]	-	-	+14 [491]	2.4 [491]
MB-pol [3133]	-	68.4	2.3	-	-10	3.5
Experimental	2.95	78.4	2.30	-41.5 [180]	+3.984	2.53

All the data is at 25 °C and 1 atm, except * at 20 °C and ** at 27 °C.

Source: `http://www1.lsbu.ac.uk/water/water_models.html`

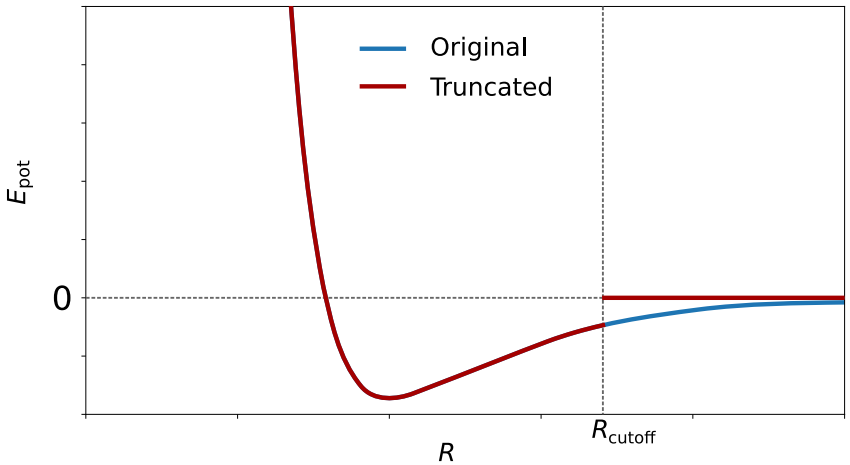
Cutoffs: The problem

Most pair potentials:

- Decay quickly with R (short sighted), but
- Have non-zero value for any finite r (non-bounded support)

$\implies$ number of atoms pairs scales quadratically

If one could ignore atom pairs with $R > R_{\text{cutoff}}$, scaling would be linear. Naive approach:

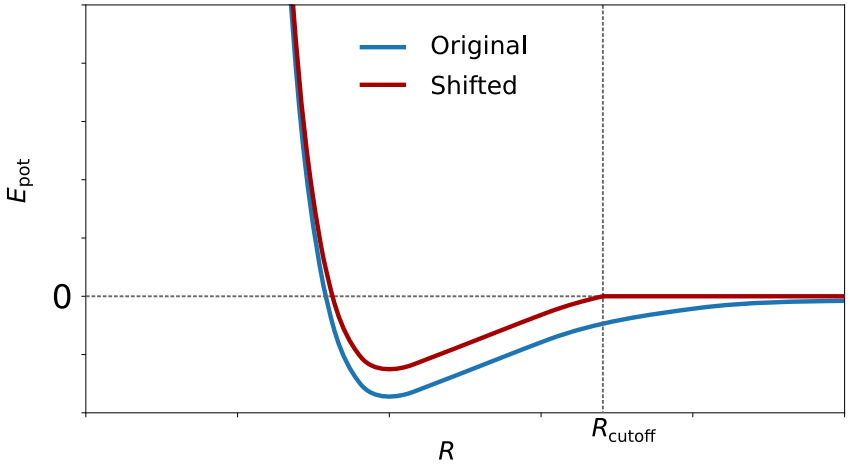


Unphysical consequences:

- Discontinuity of potential energy
- What happens with forces?

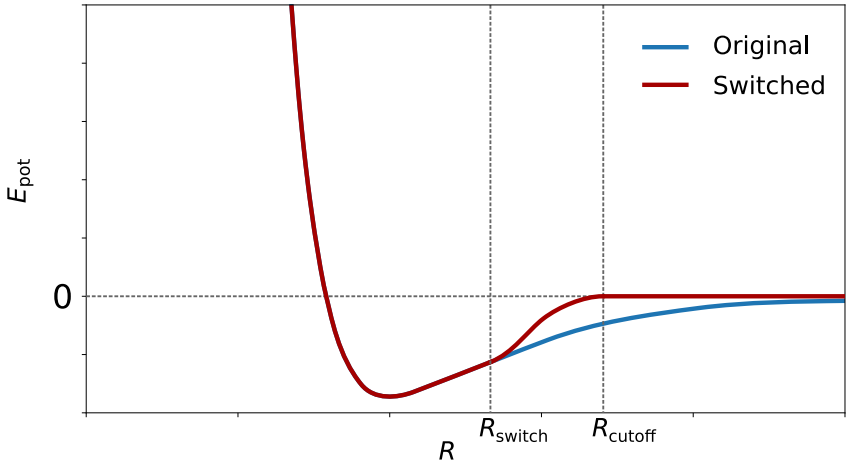
Cutoffs: Smarter approaches

Shifting:



- Rigid displacement
- Affects the whole range
- Forces still undefined

Switching:



- Multiplication by smooth function (1 at R_{switch}, 0 at R_{cutoff})
- Number of continuous derivatives controllable
- Does not affect other regions

A C^∞ bump function

Recipe for a nice cutoff function:

1. Start with the C^∞ function $f(t) = \begin{cases} e^{-1/t} & \text{, if } t > 0 \\ 0 & \text{, otherwise} \end{cases}$
2. Use it to build a smooth step function $g(t) = \frac{f(t)}{f(t)+f(1-t)}$
3. Make it symmetric and ajust the range:

$$\boxed{\rho(x) = 1 - g\left(\frac{x^2 - R_{\text{switch}}^2}{R_{\text{cutoff}}^2 - R_{\text{switch}}^2}\right)}$$

Reference: L. W. Tu, *An Introduction to Manifolds*, 2nd edition, Springer (2011). ISBN: 1441973990

Some potentials for solid-state simulations

The Stillinger-Weber potential (1985)

A paradigmatic three-body potential

$$V_2(R) = A\epsilon\left[B\left(\frac{\sigma}{R}\right)^p - \left(\frac{\sigma}{R}\right)^q\right]e^{\frac{\sigma}{R-a\sigma}}$$

$$V_3(R_{IJ}, R_{IK}, \theta_{IJK}) = \lambda\epsilon(\cos\theta_{IJK} - \cos\theta_0)^2 e^{\frac{\gamma\sigma}{R_{IJ}-a\sigma}} e^{\frac{\gamma\sigma}{R_{IK}-a\sigma}}$$

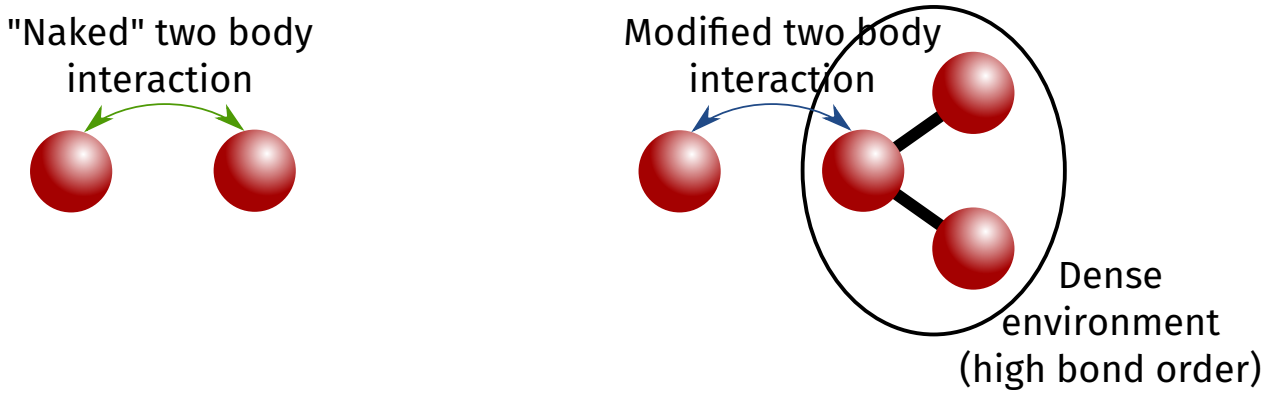
- Parameters: A, ϵ, B, σ, p, q, a, λ, θ_0, γ
- Original success: solid and liquid phases of Si, based on experimental data. $\cos\theta_0 = -1/3$ explains why
- Reparameterized time and time again: GaN, 2D materials, Si (redux)...

Bond-order potentials: Tersoff

Effective two-body term influenced by the environment

$$
\begin{aligned}
V_2(R_{IJ}) =& f_c(R_{IJ})[f_r(R_{IJ}) + b_{IJ} f_a(R_{IJ})] \\
f_r(R_{IJ}) =& A e^{-\lambda_1 R_{IJ}} \\
f_a(R_{IJ}) =& -B e^{-\lambda_2 R_{IJ}} \\
b_{IJ} =& (1 + \beta^n \xi_{IJ}^n)^{-\frac{1}{2n}} \\
\xi_{IJ} =& \sum_{K \notin \{I,J\}} f_c(R_{IK}) g(\theta_{IJK}) e^{\lambda_3^m (R_{IJ} - R_{IK})^m} \\
g(\theta) =& \gamma \left\{ 1 + \frac{c^2}{d^2} - \frac{c^2}{d^2 + (\cos\theta - \cos\theta_0)^2} \right\} \\
f_c(R) =& \begin{cases} 1, & R < C - D \\ \frac{1}{2} - \frac{1}{2}\sin\left(\frac{\pi}{2}\frac{R-C}{D}\right), & C - D < R < C + D \\ 0, & R > C + D \end{cases}
\end{aligned}
$$

The concept behind bond-order potentials



From a fundamental point of view, an atom with saturated bonds will not interact as strongly with others.

Transferability?

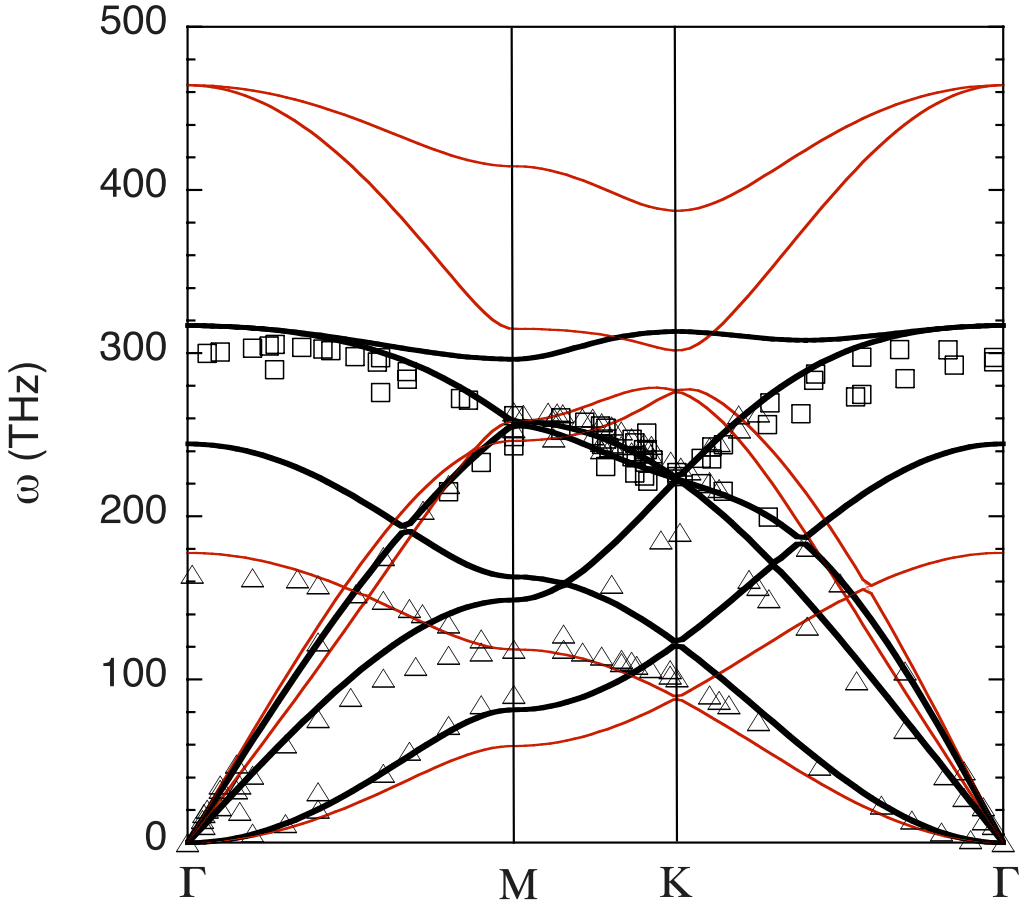


FIG. 1. (Color online) Phonon dispersion for graphene along high-symmetry directions obtained using the Tersoff EIP. Thick black lines correspond to the optimized parameter set (this work); thin red lines correspond to the original parameter set. Squares (triangles) are in-plane experimental data points for graphite from Ref. 43 (Ref. 44).

Source:
Phys. Rev. B 81 (2010) 205441

For a more depressing read, see *Effects of Cutoff Functions of Tersoff Potentials on Molecular Dynamics Simulations of Thermal Transport*, Modelling Simul. Mater. Sci. Eng. 19 (2011) 025004.

Machine-learning (ML) potentials

Mapping the energy landscape

$E_{\text{pot}}\left(\{\boldsymbol{x}_I\}_{i=1}^{M}\right)$ defines a manifold with complex features:

- Number of minima exponential on M
- Multiple transition paths
- Features at different length and energy scales

ML provides powerful general regression techniques to create controlled interpolations on a region of this surface.

Back to Cl_2

We have already parameterized a ML potential:

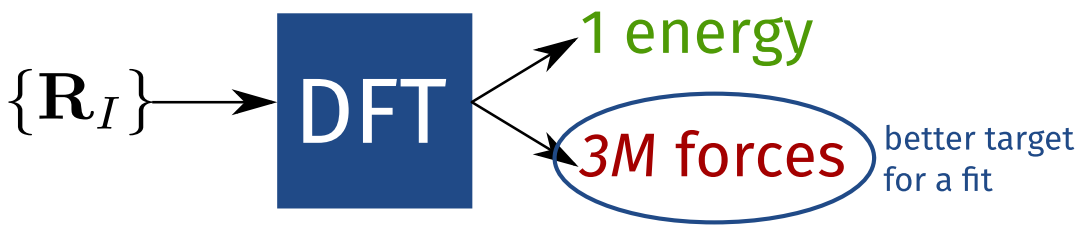
```
data = np.loadtxt("e_vs_l.txt")
r = data[:, 0]
E = data[:, 1] * ase.units.Hartree
newr = np.linspace(r.min(), r.max(), num=1000)
newE = sp.interpolate.interp1d(r, E, kind="cubic")(newr)
```

The interpolating spline:

- \+ Does not rely on “chemical insight”
- \+ Outperforms the Morse potential
- \+ Has clear limits for transferability: `E.min()`, `E.max()`
- \- Analytical calculations mostly out of the question
- \- Not easy to “interpret”

The ML workflow

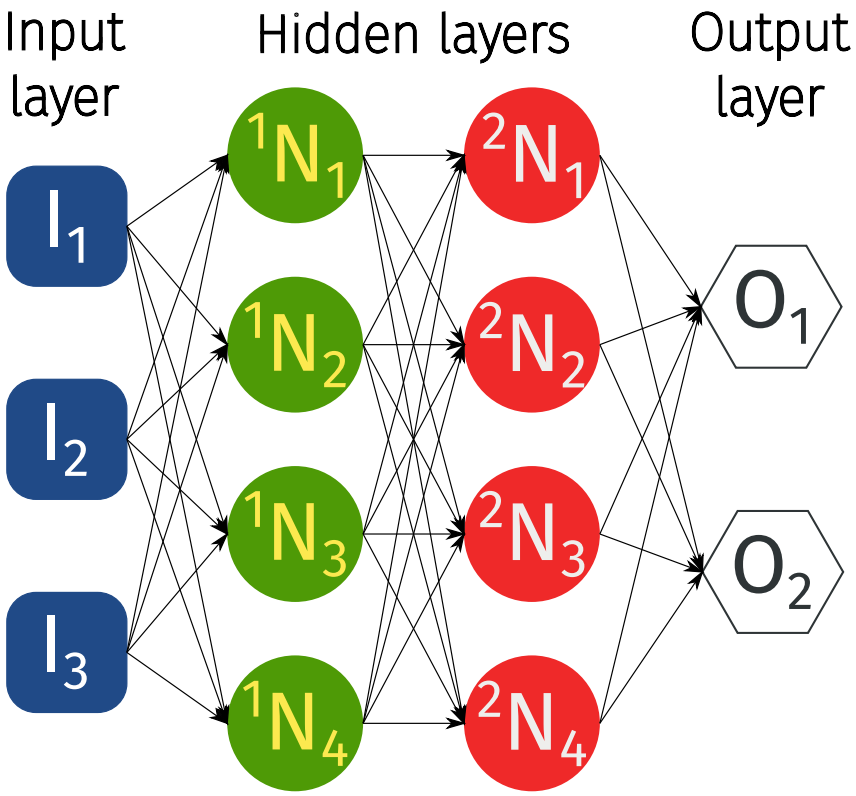
1. Data gathering



2. Choice of regression method
   - Descriptors (inputs)
   - Functional form and target
   - Regularization
3. Training: Parameter fitting
4. Validation: Assessment of transferability and accuracy
5. Application

Not fundamentally different from training a self-driving car, a spam filter or Google Translate

Neural networks



- Each "neuron"/perceptron transforms its inputs $\{J_i\}$:
  1. Linear combination + constant: $L = b + \sum_i a_i J_i$
  2. Nonlinear activation function: $N = f(L)$
- Activation function: $\tanh$, ELU, ReLU...
- Huge number of parameters, but can be adjusted systematically (*e.g.* backpropagation).
- This basic idea has been greatly expanded recently (*e.g.* deep learning).

Other popular approaches: Gaussian process regression, kernel methods

Methods for validation

Overfitting happens when a model is trained to reproduce meaningless minutia of the input data at the expense of transferability.

To assess and avoid overfitting, fits must be validated on out-of-sample data. Several strategies are possible:

Data splitting: Use a given percentage of the data for fitting the model, assess its accuracy using the rest.

Bootstrap: Extract hundreds or thousands of samples from the dataset (with replacement), fit the model each time, evaluate its performance with the remaining data.

k-fold cross-validation: Split the data in k equal parts, fit the model to each subset of $k-1$ in sequence, measuring its quality based on the remaining "fold".

Repeated k-fold cross-validation: Run several k-fold cross-validation steps with random splits.

Leave-one-out cross-validation: Fit the model to all data minus one point, repeatedly, and use that lone point to obtain validation statistics.

Useful software

- `http://scikit-learn.org` : a comprehensive, well documented ML kit for Python, covering the whole workflow
- `https://www.r-project.org` The language of statistics. Check out the Rstudio (`https://www.rstudio.com/`) environment and the caret (`http://caret.r-forge.r-project.org`) package.
- `https://www.tensorflow.org/` : Google's open source software library for numerical computation using data flow graphs.
- `https://pytorch.org/` : Facebook's competitor to TensorFlow

Descriptors

Big challenge for NN potentials. Must be invariant w.r.t.:

- Rotation
- Translation
- Permutation of equivalent atoms

Still a work in progress

Descriptors: Three examples

Atom-centered symmetry functions (ACSFs):

$$
\begin{aligned}
G_I^{\mathrm{rad}} &= \sum_J e^{-\eta(R_{IJ}-Rs)^2} f_{\mathrm{cut}}(R_{IJ}) \\
G_I^{\mathrm{ang}} &= 2^{1-\xi} \sum_{J,K\neq I} (1+\lambda\cos\theta_{IJK})^{\xi} e^{-\eta(R_{IJ}^2+R_{IK}^2+R_{JK}^2)} \times \\
&\quad f_{\mathrm{cut}}(R_{IJ}) f_{\mathrm{cut}}(R_{IK}) f_{\mathrm{cut}}(R_{JK})
\end{aligned}
$$

Smooth overlap of atomic positions:

$$
\rho_{\mathrm{SOAP}}(\boldsymbol{x}) = \sum_J e^{-\alpha|\boldsymbol{x}-\boldsymbol{x}_J|^2}
$$

Eigenvalues of the Coulomb matrix:

$$
M_{IJ} = \begin{cases} 0.5 Z_I^{2.4}, & \text{if } I = J \\ \frac{Z_I Z_J}{|\boldsymbol{x}_I - \boldsymbol{x}_j|}, & \text{if } I \neq J \end{cases}
$$

A Gaussian approximation potential (GAP) for graphene

Phys. Rev. B 97 (2018) 054303

- Sum of atomic contributions based on SOAP
- Training data: energies, forces and stresses from DFT
- Around 1000 configurations in the training set
- DFT-like accuracy
- Great transferability: phonons, finite-T anharmonicity, elasticity...

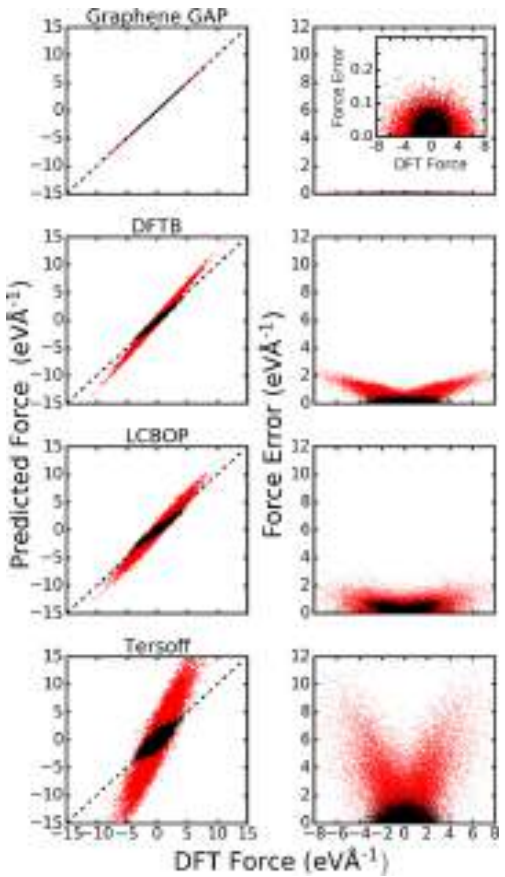
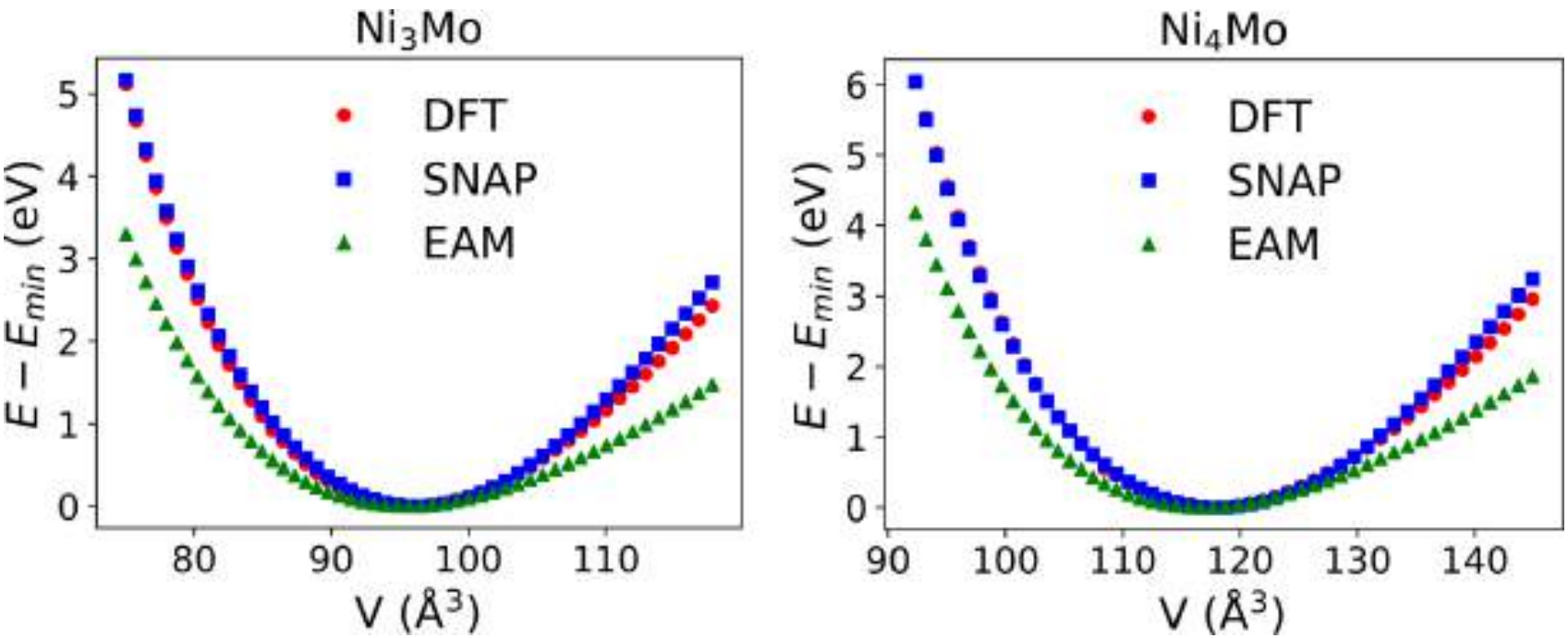


FIG. 1. Force correlations (left) and associated force errors (right) on an independent reference data set of conf gurations for the graphene GAP model, DFTB, LCBOP, and Tersoff potentials as compared to the reference DFT method, the plots for all methods considered can be found in the SM. Black points indicate forces perpendicular to the plane of the graphene sheet (out-of-plane) while red points indicate forces oriented in the plane. The inset in the graphene GAP plot has a different scale on the y axis to show more clearly the distribution of force errors, which are smallest for large forces with a Gaussian distribution.

SNAP (Spectral network analysis potential) vs classical

Equation of state for Ni-Mo alloy:



[EAN: Embedded atom method]

Source: X.-G. Li *et al.*,

`https://lammps.sandia.gov/workshops/Aug19/talk_li.pdf` (2019)

ANI-1: An extensible neural network potential

Chem. Sci. 8 (2017) 3192

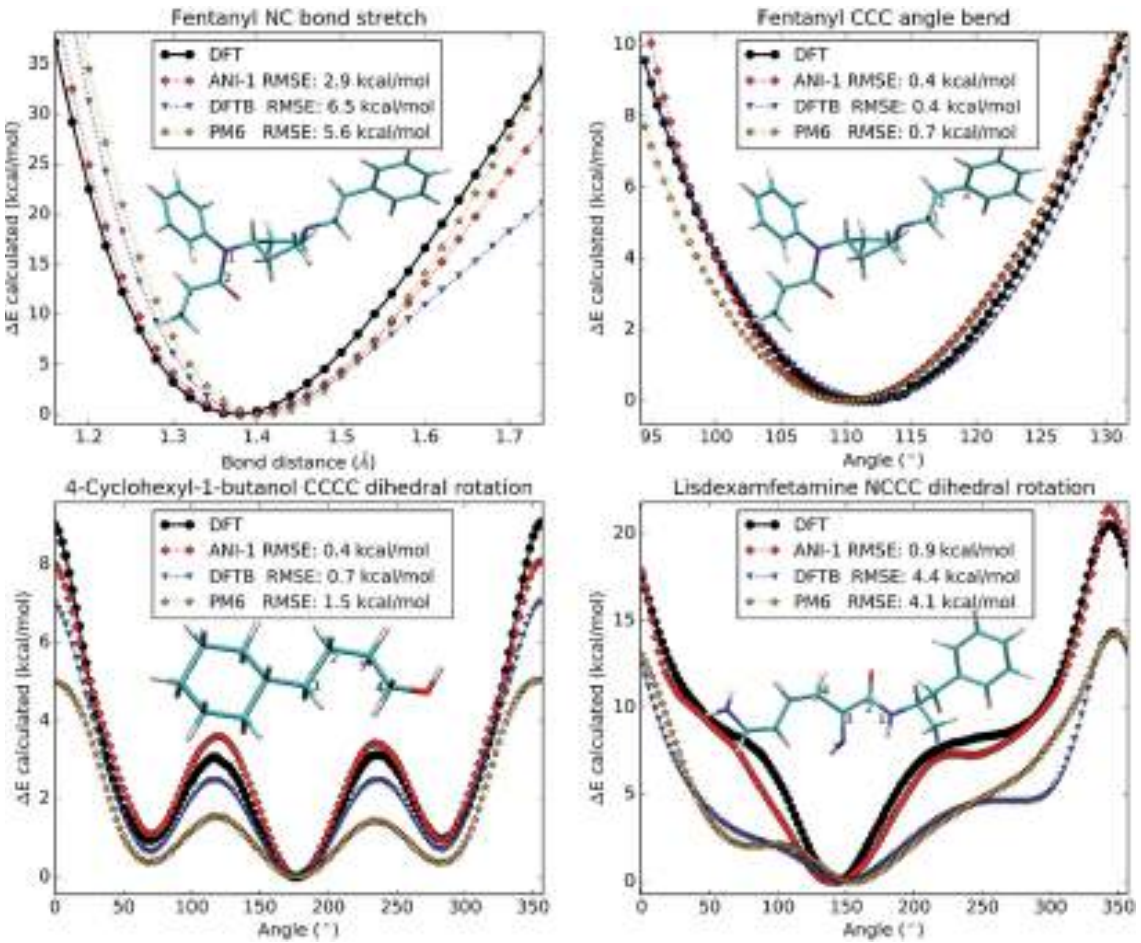


Fig. 7 Each subplot shows a one-dimensional potential surface scan generated from DFT, the ANI-1 potential, and two popular semi-empirical methods, DFTB and PM6. The atoms used to produce the scan coordinate are labeled in the images of the molecules in every sub-plot. Each figure also lists the RMSE, in the legend, for each method compared to the DFT potential surface.

- Sum of atomic contributions based on ACSFs
- Deep-learning NN
- 57951 molecules from the GDB-11 database
- Configuration sampling based on normal modes

What about the long-range part?

No commonly accepted approach yet. Some possibilities:

- Fixed atomic multipoles (charges, dipoles...– see next section)
- Machine-learning method for the charge density, potential energy computed from it [CENT Method, Goedecker and coworkers, Phys. Rev. B 92 (2015) 045131]
- Project long-range part onto local descriptors [A. Grisafi & M. Ceriotti M, J. Chem. Phys. 151 (2019) 204105]

Very relevant for systems with local or global charges

Evaluating the forces

First derivatives of the potential required for:

- Molecular dynamics
- Training of advanced potentials using forces

[And we may need higher orders too!]

By hand: 👍 for Lennard-Jones

😱🤧 for advanced potentials

Computer algebra systems + code generation:

🐌 (redundant calculations)

⚠️ (loss of precision)

Numerical differentiation:

🐌 (lots of evaluations)

⚠️ (fragile)

Automatic differentiation: 👌

Analytic differentiation + code generation (input)

- Let a computer algebra system do most of the work

Code

```python
#!/usr/bin/env python

import sympy
from sympy.utilities.codegen import codegen

x = sympy.Symbol("x")
y = sympy.Symbol("y")

r = sympy.sqrt(x**2 + y**2)
V = sympy.exp(r / 1.5) * sympy.cos((r - 1.) / 3.)
fx = -sympy.diff(V, x)
fy = -sympy.diff(V, y)
result = codegen([("fx", fx), ("fy", fy)], "C")

print(result[0][1])
```

Analytic differentiation + code generation (result)

Code generated with SymPy 1.5.1

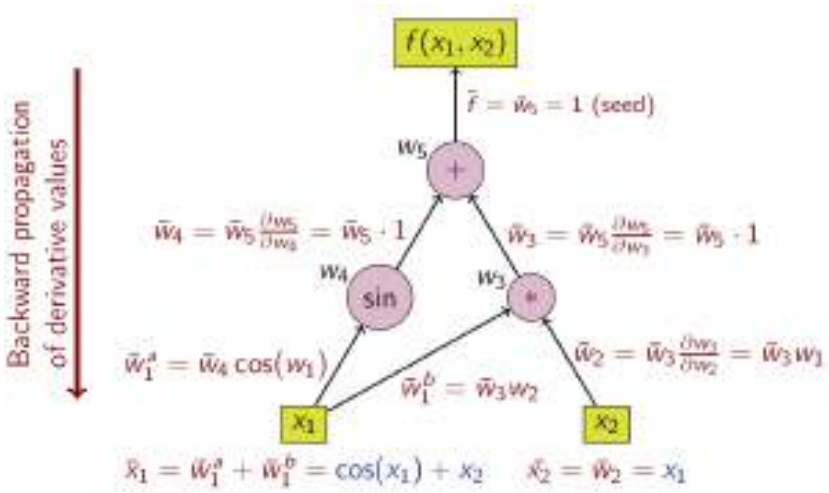
```
#include <math.h>

double fx(double x, double y)
{
    double fx_result;
    fx_result =
        0.33333333333333331 * x *
            exp(0.66666666666666663 * sqrt(pow(x, 2) + pow(y, 2))) *
            sin(0.33333333333333331 * sqrt(pow(x, 2) + pow(y, 2)) -
                0.33333333333333331) / sqrt(pow(x, 2) + pow(y, 2)) -
        0.66666666666666663 * x *
            exp(0.66666666666666663 * sqrt(pow(x, 2) + pow(y, 2))) *
            cos(0.33333333333333331 * sqrt(pow(x, 2) + pow(y, 2)) -
                0.33333333333333331) / sqrt(pow(x, 2) + pow(y, 2));
    return fx_result;
}

double fy(double x, double y)
{
    double fy_result;
    fy_result =
        0.33333333333333331 * y *
            exp(0.66666666666666663 * sqrt(pow(x, 2) + pow(y, 2))) *
            sin(0.33333333333333331 * sqrt(pow(x, 2) + pow(y, 2)) -
                0.33333333333333331) / sqrt(pow(x, 2) + pow(y, 2)) -
        0.66666666666666663 * y *
            exp(0.66666666666666663 * sqrt(pow(x, 2) + pow(y, 2))) *
            cos(0.33333333333333331 * sqrt(pow(x, 2) + pow(y, 2)) -
                0.33333333333333331) / sqrt(pow(x, 2) + pow(y, 2));
    return fy_result;
}
```

Automatic differentiation

- Usually, potentials are built from a limited number of functions (exponentials, trigonometric, sums, multiplications...)
- Recursive, automated, optimised application of the chain rule $\frac{\mathrm{d}f[g(x)]}{\mathrm{d}x} = f'[g(x)]g'(x) \implies$ the function and the derivative can be evaluated at the same time
- Numerical precision/accuracy comparable to function itself



(Source: Berland at the English language Wikipedia)

The JAX framework

Automatic differentiation and JIT compilation for Python and NumPy

- Google-sponsored research project
- Goal: high-performance machine-learning research (great for NNs)
- Targets XLA (Accelerated Linear Algebra), designed to accelerate TensorFlow
- Good performance on CPUs, GPUs and TPUs
- Foundation for new ecosystem of machine learning (Flax), optimization (Optax)...

Home page: `https://github.com/google/jax`

JAX in action

Lennard-Jones with derivatives and test

```python
#!/usr/bin/env python

import numpy.random as random
import jax
import jax.numpy as np

def lj_potential(positions):
    """Lennard-Jones potential in reduced units.

    'positions' must be an Mx3 array.
    """
    # Compute all relative positions between pairs without iterating.
    delta = positions[:, np.newaxis, :] - positions
    # Take only the combinations of two different atoms.
    indices = np.triu_indices(positions.shape[0], k=1)
    delta = delta[indices[0], indices[1], :]
    # Compute the squared distances and evaluate the potential energy.
    r2 = (delta * delta).sum(axis=1)
    rm2 = 1. / r2
    rm6 = rm2 * rm2 * rm2
    rm12 = rm6 * rm6
    return (rm12 - 2. * rm6).sum()

lj_gradient = jax.jit(jax.grad(lj_potential))  # Automatic gradient function
# Put 20 particles in a cube of side 5.
test_positions = 5. * random.random_sample((20, 3))
e_pot = lj_potential(test_positions)
forces = -lj_gradient(test_positions)
print("Potential energy:", e_pot, "Forces:", forces, sep="\n")
```

Potentials for molecular mechanics

Molecular mechanics

Specialized domain of atomistic simulation: molecular systems, with a bent towards organic molecules in fluid phases

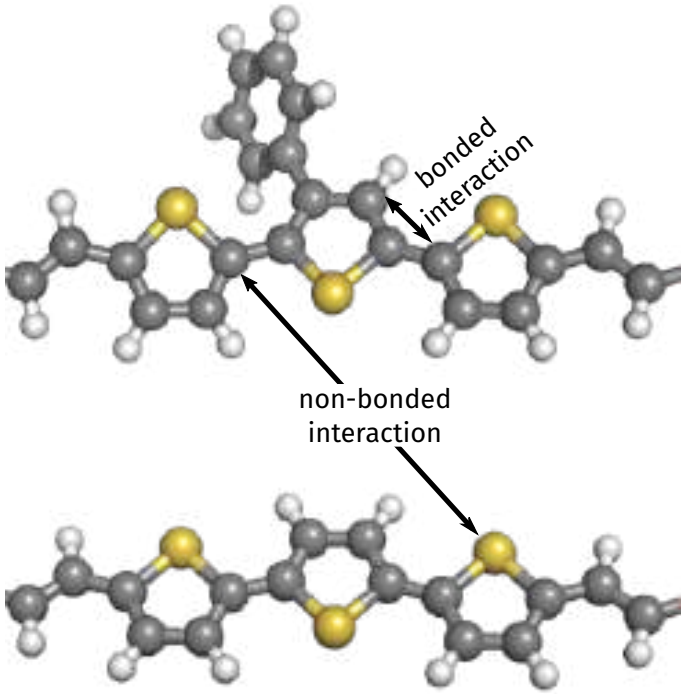
---

Features:

- Molecular geometry + topology
- Bonded vs. non-bonded interactions
- Atoms have vdW radii, partial charges and possibly polarizabilities

Variations:

- All-atom vs. united-atom
- Coarse graining
- Hybrid MM + *ab-initio*



OPLS-AA

Jorgensen *et al.*, J. Am. Chem. Soc. 118 (1996) 11225

A typical molecular-mechanics potential

$$V_{\text{non-bonded}}(a,b) = \sum_{\substack{I\in a\\ J\in b}} f_{IJ}\left\{4\epsilon_{IJ}\left[\left(\frac{\sigma_{IJ}}{R_{IJ}}\right)^{12} - \left(\frac{\sigma_{IJ}}{R_{IJ}}\right)^{6}\right] + \frac{1}{4\pi\varepsilon_0}\frac{q_i q_j}{R_{IJ}}\right\},$$

$$V_{\text{bonded}} = V_{\text{stretch}} + V_{\text{angle}} + V_{\text{dihedrals}}$$

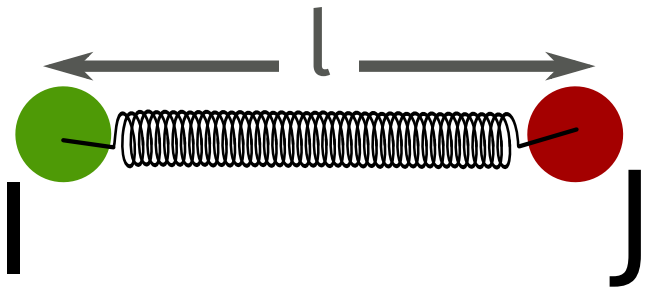
$$V_{\text{stretch}} = \sum_{(I,J)\text{ are bonded}} K_R(I,J)\left[R_{IJ} - R_{0,IJ}\right]^2$$

$$V_{\text{angle}} = \sum_{\substack{(I,J,K)\text{ form}\\ \text{an angle}}} K_\theta(I,J,K)\left[\theta_{IJK} - \theta_{0,IJK}\right]^2$$

$$V_{\text{dihedrals}} = \sum_{\text{dihedrals }\Phi_s} \left\{\frac{V_1(s)}{2}[1+\cos\Phi_s] + \frac{V_2(s)}{2}[1-\cos(2\Phi_s)] + \right.$$
$$\left. + \frac{V_3(s)}{2}[1+\cos(3\Phi_s)] + \frac{V_4(s)}{2}[1-\cos(4\Phi_s)]\right\}$$

Bonds and angles

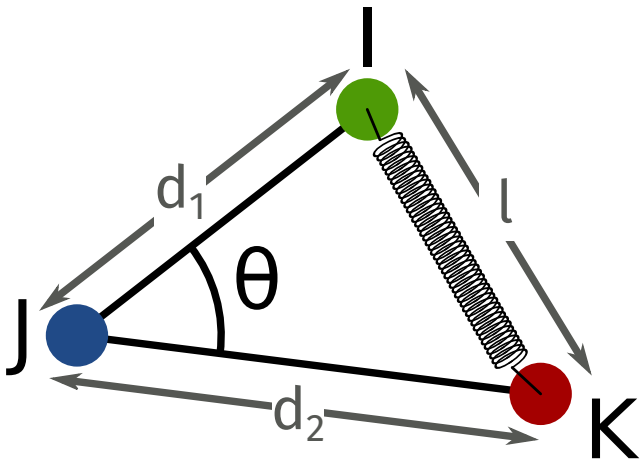
Bond term:



Harmonic potential:

$\Delta E_{\text{pot}} = \frac{1}{2}\tilde{K}(l - l_0)^2$

Angle term:



Approximate harmonic term:

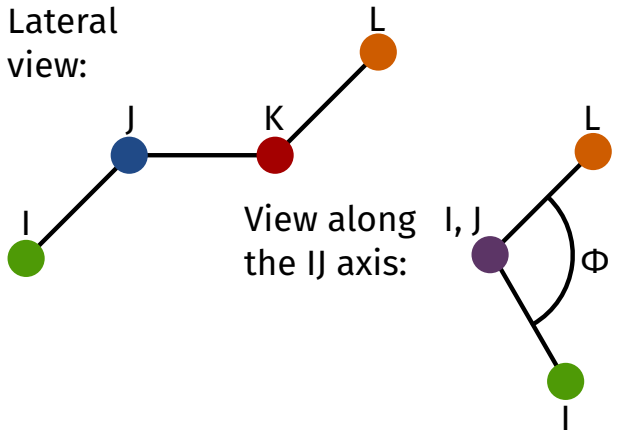
$$\Delta E_{\text{pot}} = \frac{1}{2}\tilde{K}(l - l_0)^2$$

$$l = \sqrt{d_1^2 + d_2^2 - d_1 d_2 \cos\theta}$$

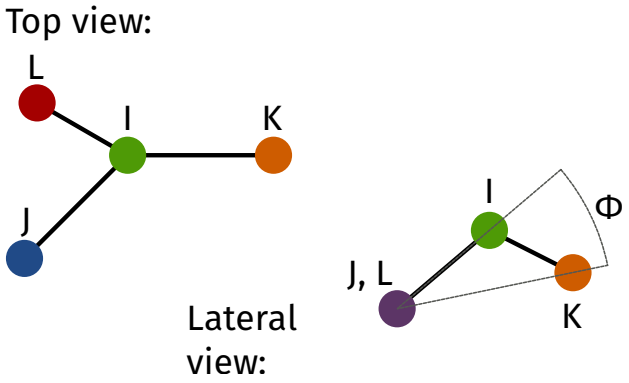
$$\Delta E_{\text{pot}} \simeq \frac{1}{2}\tilde{K}\left(\left.\frac{\mathrm{d}l}{\mathrm{d}\theta}\right|_{\theta=\theta_0}\right)^2 (\theta - \theta_0)^2$$

Dihedrals: A 4-body term

Proper dihedrals:



Improper dihedrals:



Ryckaert-Bellemans functional form:

$$\Delta E_{\text{pot}} = \frac{V_1}{2}[1+\cos\Phi] + \frac{V_2}{2}[1-\cos(2\Phi)] + \frac{V_3}{2}[1+\cos(3\Phi)] + \frac{V_4}{2}[1-\cos(4\Phi)]$$

$\Rightarrow$ different symmetries depending on which constants are zero.

The 1-4 reduction

"Fudge" factor for non-bonded interactions:

$$f_{IJ} = \begin{cases} 0.5, & \text{for atoms up to three bonds apart} \\ 1, & \text{for other pairs of atoms} \end{cases}$$

Goal: Avoid double counting of non-bonded interactions, whose perturbative expansions give rise to bond and angle terms.

Parameterizing proper dihedrals

Please see Parameterizing_dihedrals.ipynb

Schemes for obtaining the charges

No notion of "atomic charges" in quantum mechanics $\Rightarrow$ many different schemes to assign partial charges:

- Formal charges
- Restrained electrostatic potential (RESP)
- Mulliken charges
- Bader charges
- Born charge tensors
- ...(~ 30 schemes listed at `https://en.wikipedia.org/wiki/Partial_charge`)

Bader charges

Comput. Mater. Sci., 36 (2006) 35

- Based on charge density $n(\boldsymbol{x})$
- Space partitioned with zero-flux surfaces (inflection points of n)
- $Q_I =$ integral of n in the Bader volume of I
- Preserves total charge

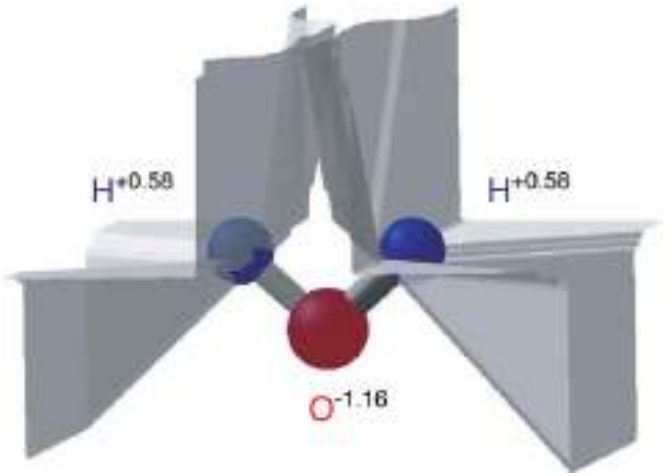


Fig. 2. Dividing surfaces separating the oxygen and hydrogen Bader regions in a water molecule. The integration over the Bader regions indicates that 0.58 e are transferred from each hydrogen atom to the oxygen atom.

RESP

J. Phys. Chem. 97 (1993) 10269

- Charges used as a fitting parameters to reproduce the electrostatic component of DFT results
- Some constraints imposed:
  - Equivalent charges for equivalent atoms
  - Fixed total charge of known charge groups
  - Hyperbolic penalty restrain on the charges avoids spurious values
- Prescriptive for many molecular-mechanics force fields
- Many similar methods exist (see J. Comput. Chem. 19 (1998) 377)

		Mulliken	NPA	AIM*	CHELP	CHELPG	RESP	MK	CHELP-BOW	CHELMO
$(CH_3)_2SO$	C	0.63	0.90	0.17	0.29	0.25	0.18	0.49	0.77	0.92
	H	0.17–0.20	0.24–0.26	0.04–0.06	0.05	0.11	0.09	0.18	0.26	0.30
	S	0.75	1.26	1.27	0.08	0.22	0.17	0.28	0.35	0.40
	O	0.62	0.96	1.25	0.39	0.40	0.39	0.39	0.38	0.38
H_2O	O	0.77	0.93	1.09	0.75	0.75	0.74	0.75	0.74	0.73
	H	0.39	0.47	0.55	0.38	0.37	0.37	0.37	0.37	0.37
$(H_2O)_2$	O_{donor}	0.83	0.97	1.15	0.79	0.81	0.80	0.81	0.81	0.81
	H	0.40	0.48	0.58	0.35	0.37	0.36	0.37	0.37	0.37
	$H_{H\text{-}bond}$	0.38	0.46	0.54	0.39	0.38	0.38	0.38	0.38	0.39
	$O_{acceptor}$	0.75	0.94	1.10	0.76	0.74	0.73	0.74	0.75	0.75
	H	0.41	0.49	0.57	0.41	0.40	0.40	0.40	0.40	0.40

Interatomic potentials

Numerical Simulation and Scientific Computing II

Jesús Carrete Montaña
jesus.carrete.montana@tuwien.ac.at

Lecture 6

TECHNISCHE UNIVERSITÄT WIEN

Numerical Simulation and Scientific Computing in Fluid Mechanics:

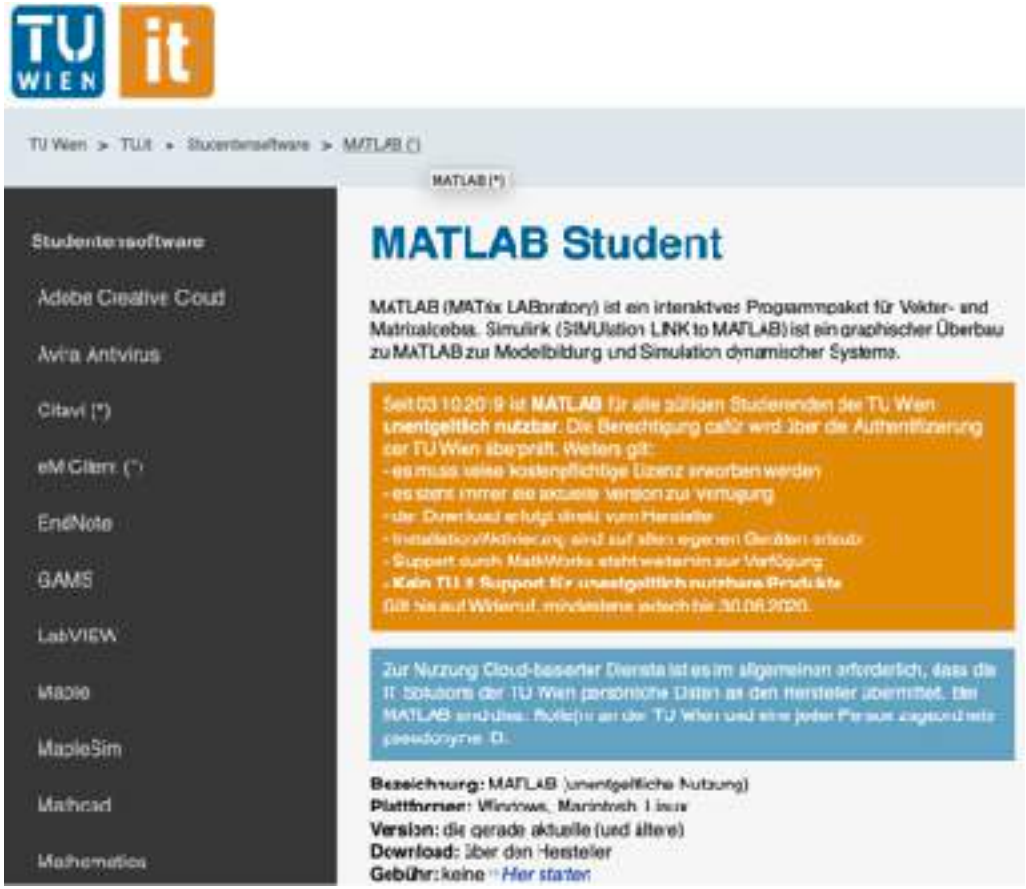
Fundamental aspects and applications

Francesco Zonta

Institute of Fluid Mechanics and Heat Transfer, TU Wien

Before starting

http://www.sss.tuwien.ac.at/sss/mla/



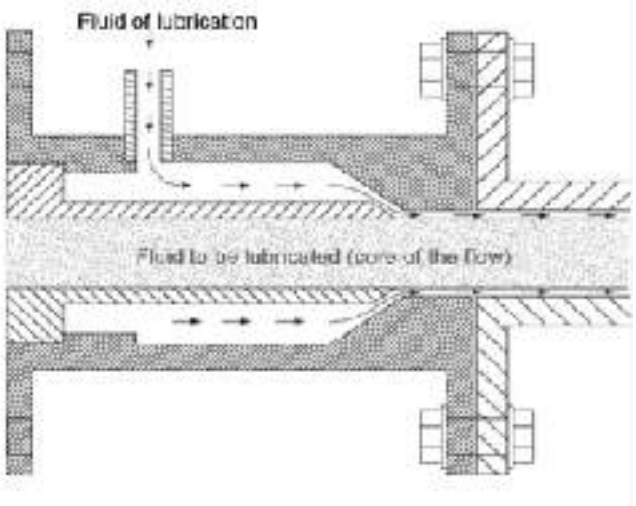
Fluid dynamics: computations we do

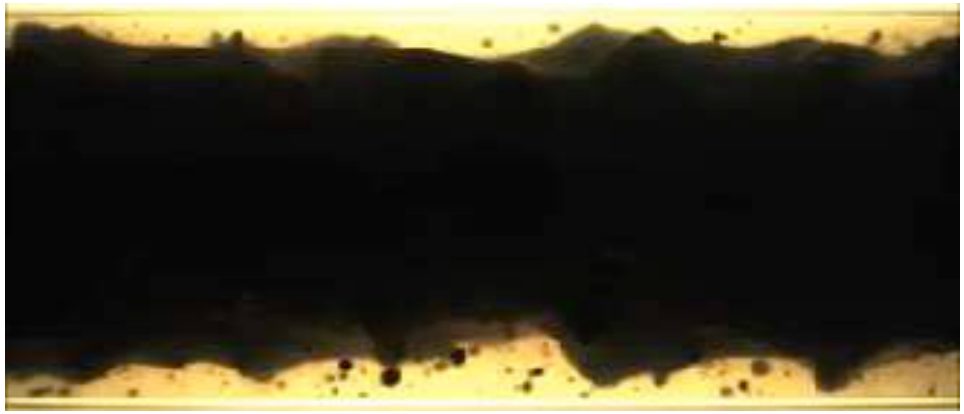
(@Institute of Fluid Mechanics and Heat Transfer, Group of Prof. Soldati)

1. Oil transport
2. Turbulent flow in channels and pipelines: effect of stratification
3. Ocean/atmosphere CO_2 exchange: phytoplankton
4. Ocean/atmosphere CO_2 exchange: effect of waves
5. Oil/water emulsions
6. Oil/water emulsions with surfactants
7. Flows in porous media

Oil Transport

Method and apparatus for measuring characteristics of core-annular flow
US PATENT 20050033545 A1
Abstract
An apparatus and method are disclosed […] core-annular flow (CAF) in a pipe […] the CAF may be developed from a lubricating fluid, such as water, and a fluid to be transported, such as oil, where the fluid to be transported forms the core region and the lubricating fluid forms the annular region.





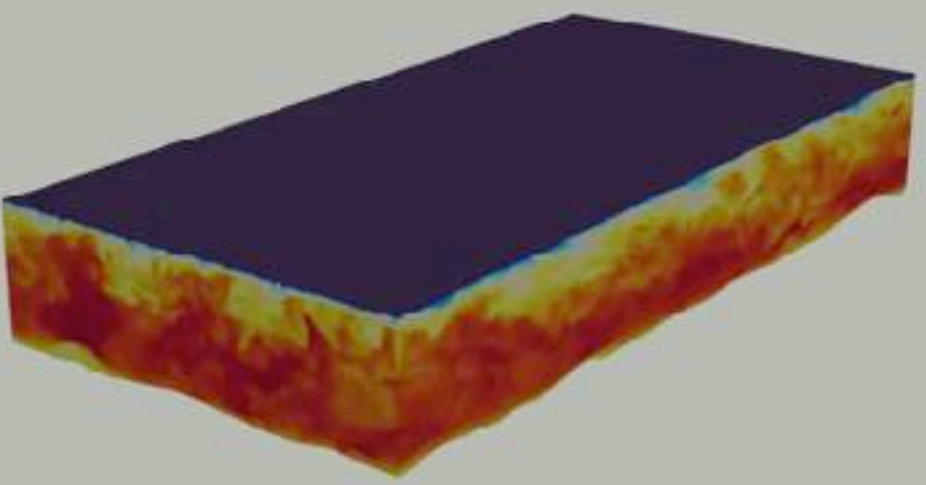
Credit: ALFA Research Group

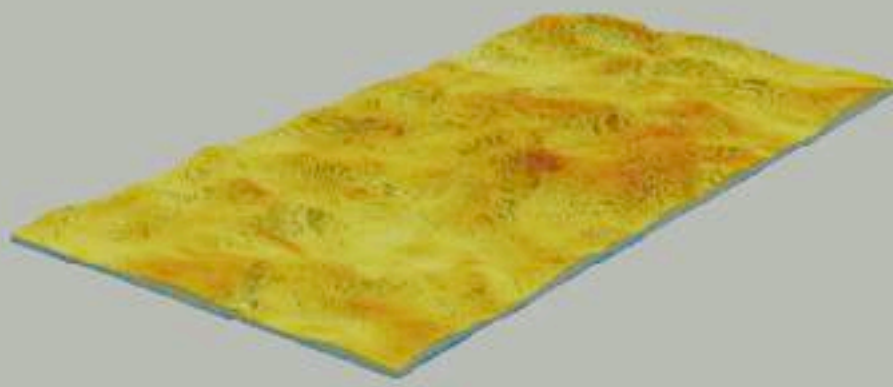
"There is a strong tendency for two fluids to arrange themselves so that the low-viscosity constituent is in the region of high shear.
This gives rise to a kind of a gift of nature in which the lubricated flows are stable, and it opens up very interesting possibilities for technological applications in which one fluid is used to lubricate another "

D:D: Joseph et al, Core-anular flows, ARFM (1997)

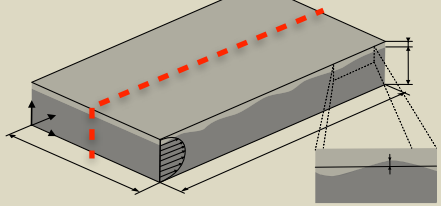
Oil Transport

$$We = \frac{\rho u_\tau^2 h}{\sigma} = 0.5 \qquad Re_\tau = \frac{\rho u_\tau h}{\eta_o} = 300 \qquad \lambda = \frac{\eta_w}{\eta_o} = \frac{\text{Water Viscosity}}{\text{Oil Viscosity}}$$

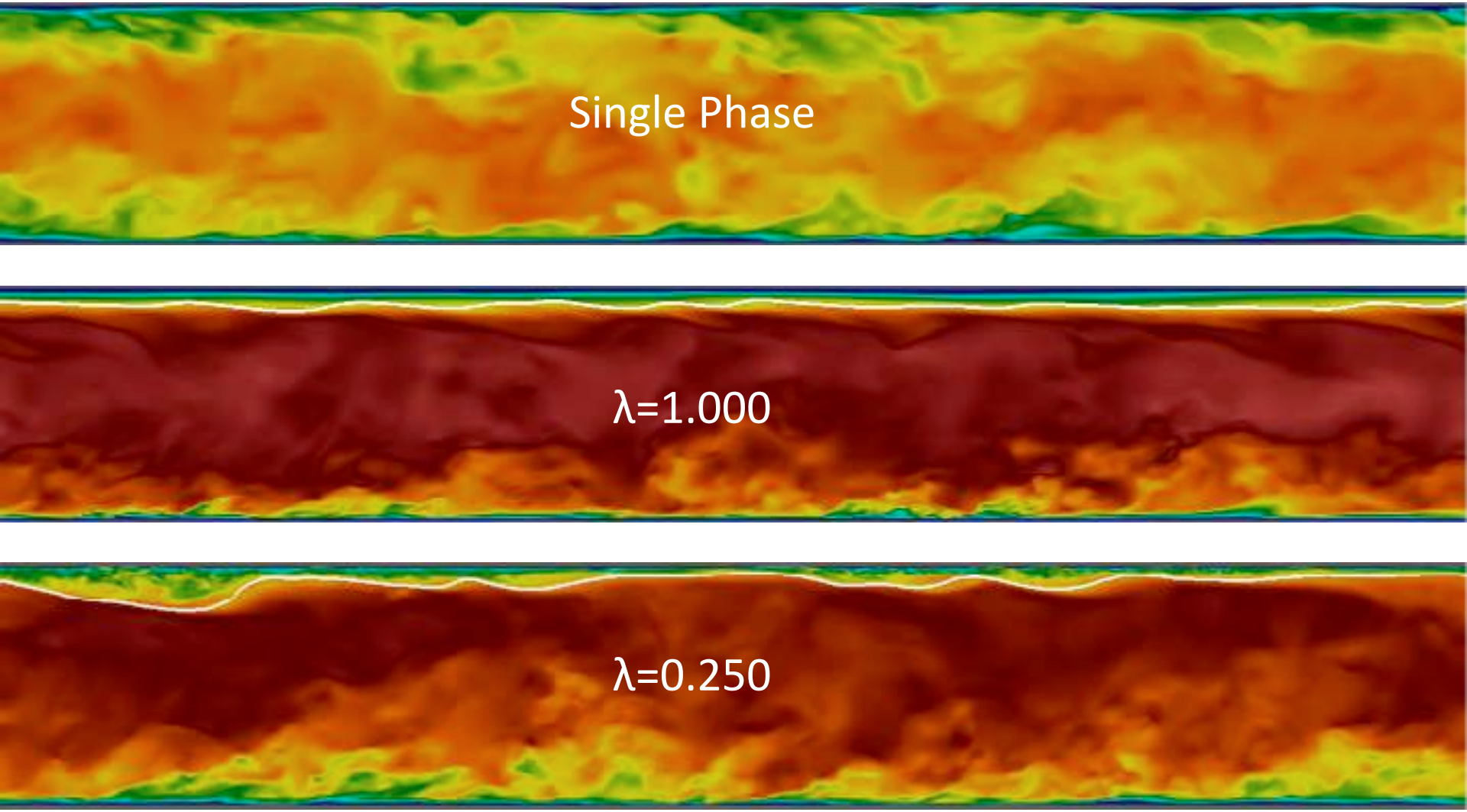


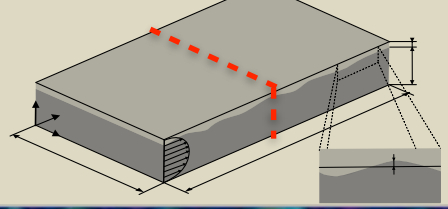


S. Ahmadi et al, Turbulent Drag Reduction by a Near Wall Surface Tension Active Interface, FTAC (2018)
S. Ahmadi et al, Turbulent drag reduction in channel flow with viscosity stratified fluids, CEF (2016)

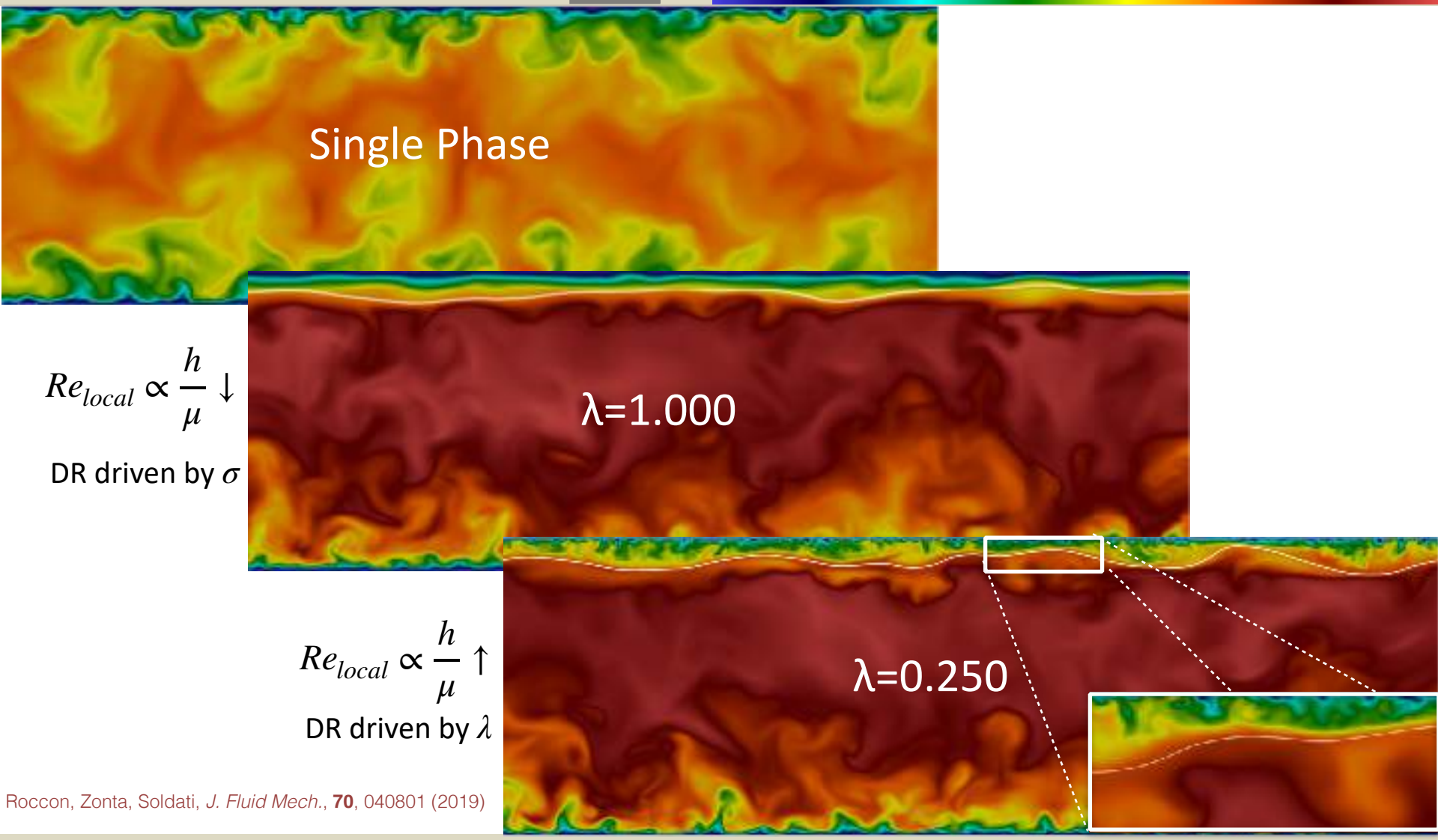


Oil Transport





Explanation: a closer look



$$Re_{local} \propto \frac{h}{\mu} \downarrow$$

DR driven by σ

$$Re_{local} \propto \frac{h}{\mu} \uparrow$$

DR driven by λ

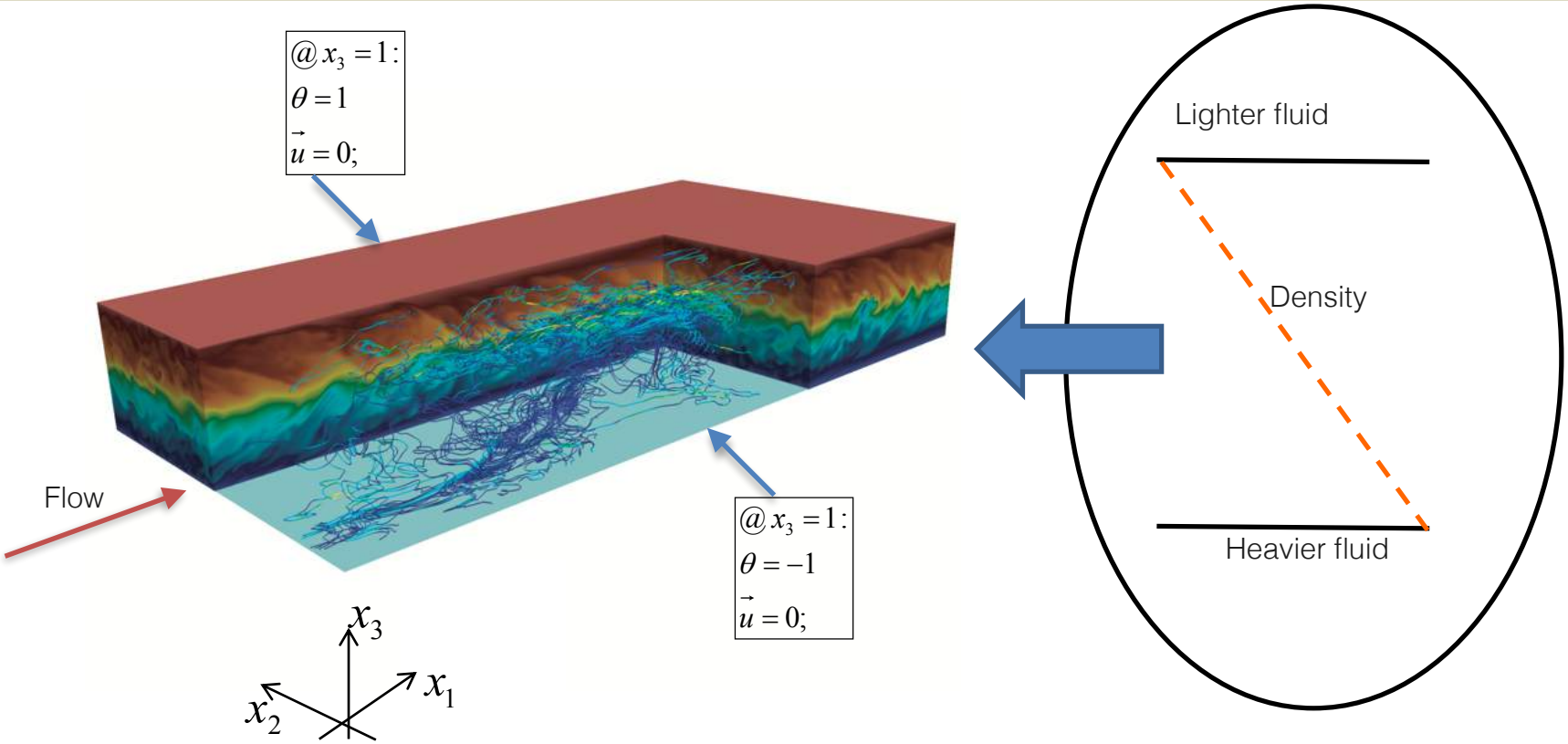
Roccon, Zonta, Soldati, *J. Fluid Mech.*, **70**, 040801 (2019)

Fluid dynamics: computations we do

(@Institute of Fluid Mechanics and Heat Transfer, Group of Prof. Soldati)

1. Oil transport
2. Turbulent flow in channels and pipelines: effect of stratification
3. Ocean/atmosphere CO_2 exchange: phytoplankton
4. Ocean/atmosphere CO_2 exchange: effect of waves
5. Oil/water emulsions
6. Oil/water emulsions with surfactants
7. Flows in porous media

Stratification

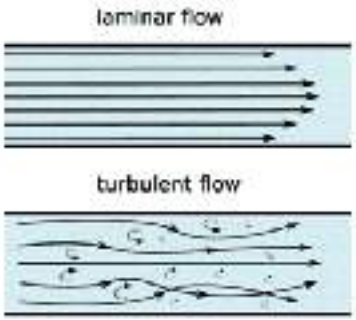


Relevant for:

Industrial applications (heat transfer devices), environmental/geophysical applications (atmospheric boundary layer, ocean)

Flow parameters

Main parameters of the flow:



(Shear) **Reynolds number:** inertial/viscous forces $\dfrac{\rho u^2/L}{\mu u/L^2} = \dfrac{uL}{\nu}$

$$Re_\tau = \frac{u_\tau h}{\nu_0}$$

Prandtl number: momentum diffusivity/thermal diffusivity

$$Pr = \frac{\nu_0}{\kappa_0}$$

Grashof number: buoyancy/viscous forces $\dfrac{g\Delta\rho}{\mu u_\nu/L^2}$

$$Gr = \frac{g\Delta\rho L^3}{\rho_0 \nu_0^2}$$

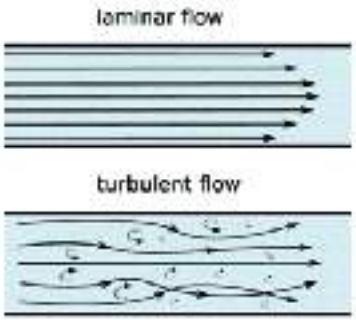
(Shear) **Richardson number:** buoyancy/inertial forces

$$Ri_\tau = \frac{g\Delta\rho h}{\rho_0 u_\tau^2}$$

Note: $Ri_\tau \to 0$, Forced conv.

$Ri_\tau \uparrow$, Buoyancy controlled/dominated

Main parameters of the flow:

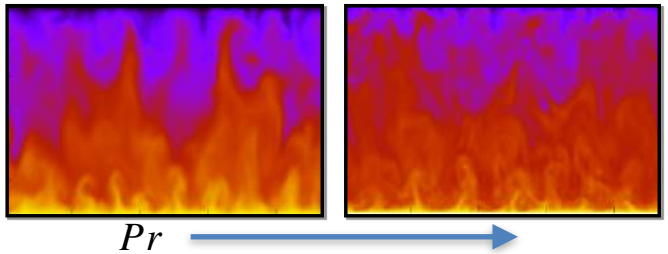


(Shear) **Reynolds number:** inertial/viscous forces $\dfrac{\rho u^2/L}{\mu u/L^2} = \dfrac{uL}{\nu}$

$$Re_\tau = \frac{u_\tau h}{\nu_0}$$

Prandtl number: momentum diffusivity/thermal diffusivity

$$Pr = \frac{\nu_0}{\kappa_0}$$



Grashof number: buoyancy/viscous forces $\dfrac{g\Delta\rho}{\mu u_\nu/L^2}$

$$Gr = \frac{g\Delta\rho L^3}{\rho_0 \nu_0^2}$$

(Shear) **Richardson number:** buoyancy/inertial forces

$$Ri_\tau = \frac{g\Delta\rho h}{\rho_0 u_\tau^2}$$

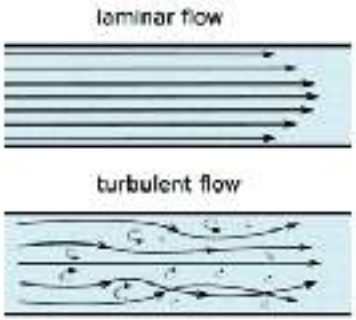
Note: $Ri_\tau \to 0$, Forced conv.

$Ri_\tau \uparrow$, Buoyancy influenced/controlled/dominated

Main parameters of the flow:

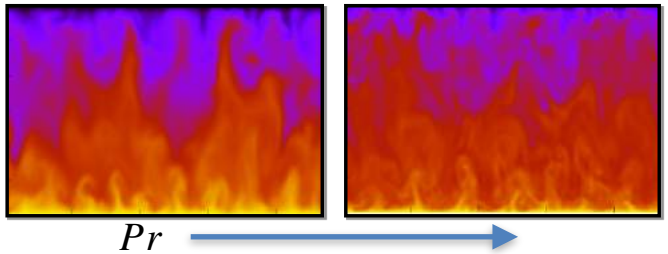
(Shear) **Reynolds number:** inertial/viscous forces $\dfrac{\rho u^2/L}{\mu u/L^2} = \dfrac{uL}{\nu}$

$$Re_\tau = \frac{u_\tau h}{\nu_0}$$



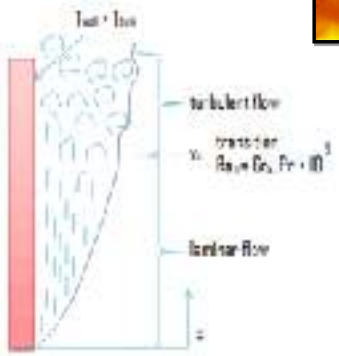
Prandtl number: momentum diffusivity/thermal diffusivity

$$Pr = \frac{\nu_0}{\kappa_0}$$



Grashof number: buoyancy/viscous forces $\dfrac{g\Delta\rho}{\mu u_\nu/L^2}$

$$Gr = \frac{g\Delta\rho L^3}{\rho_0 \nu_0^2}$$



(Shear) **Richardson number:** buoyancy/inertial forces

$$Ri_\tau = \frac{g\Delta\rho h}{\rho_0 u_\tau^2}$$

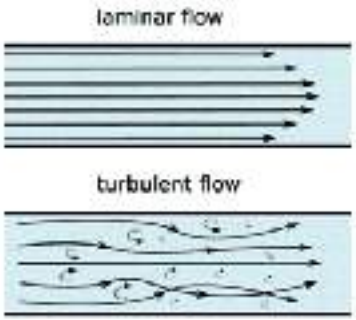
Note: $Ri_\tau \to 0$, Forced conv.

$Ri_\tau \uparrow$, Buoyancy influenced/controlled/dominated

Main parameters of the flow:

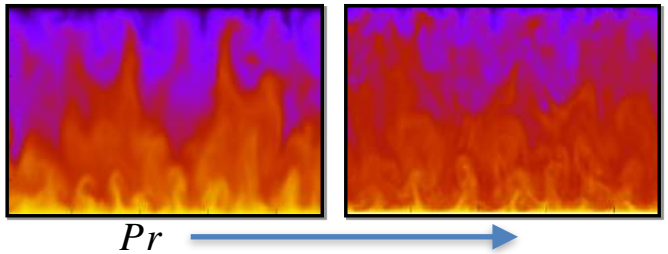
(Shear) **Reynolds number:** inertial/viscous forces $\dfrac{\rho u^2/L}{\mu u/L^2} = \dfrac{uL}{\nu}$

$$Re_\tau = \frac{u_\tau h}{\nu_0}$$



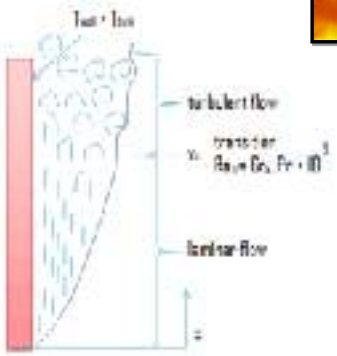
Prandtl number: momentum diffusivity/thermal diffusivity

$$Pr = \frac{\nu_0}{\kappa_0}$$



Grashof number: buoyancy/viscous forces $\dfrac{g\Delta\rho}{\mu u_\nu/L^2}$

$$Gr = \frac{g\Delta\rho L^3}{\rho_0 \nu_0^2}$$



(Shear) **Richardson number:** buoyancy/inertial forces

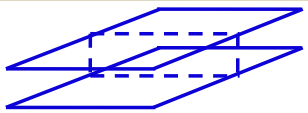
$$Ri_\tau = \frac{g\Delta\rho h}{\rho_0 u_\tau^2}$$

Note: $Ri_\tau \to 0$, Forced conv.

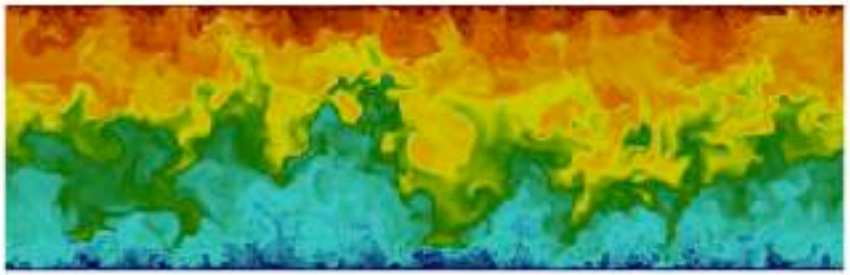
$Ri_\tau \uparrow$, Buoyancy influenced/controlled/dominated

Stratification

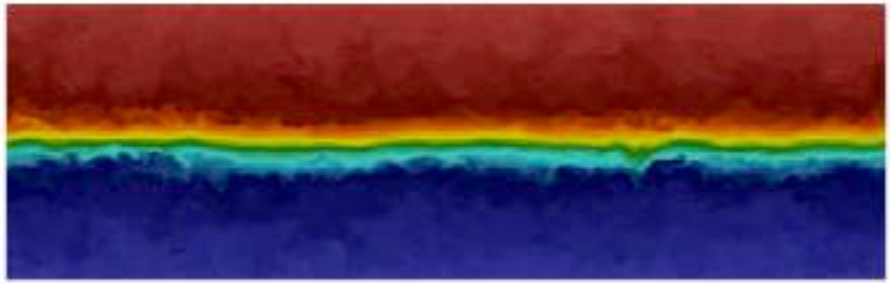
Visualization of temperature on a cross section



$Ri_\tau = 0$



$Ri_\tau = 200$



Neutrally buoyant

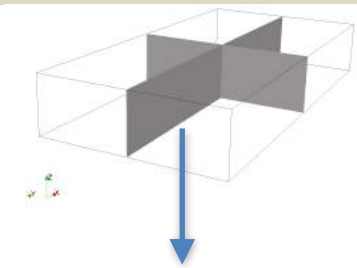
- Higher mixing
- Mixing is driven by turbulence structures

Stably stratified

- Lower mixing
- Internal waves act like barriers

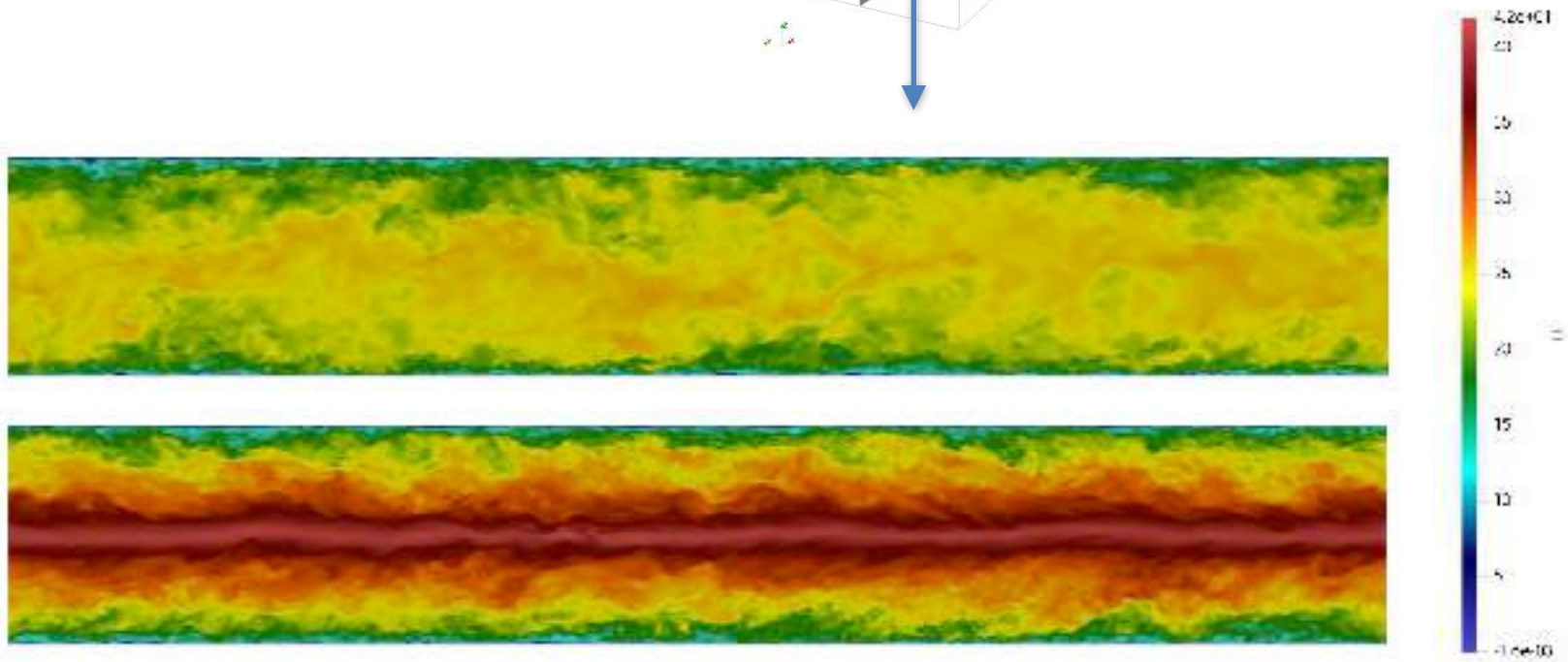
Stratification

The temperature distribution has a remarkable effect on the velocity field



$Ri_\tau = 0$

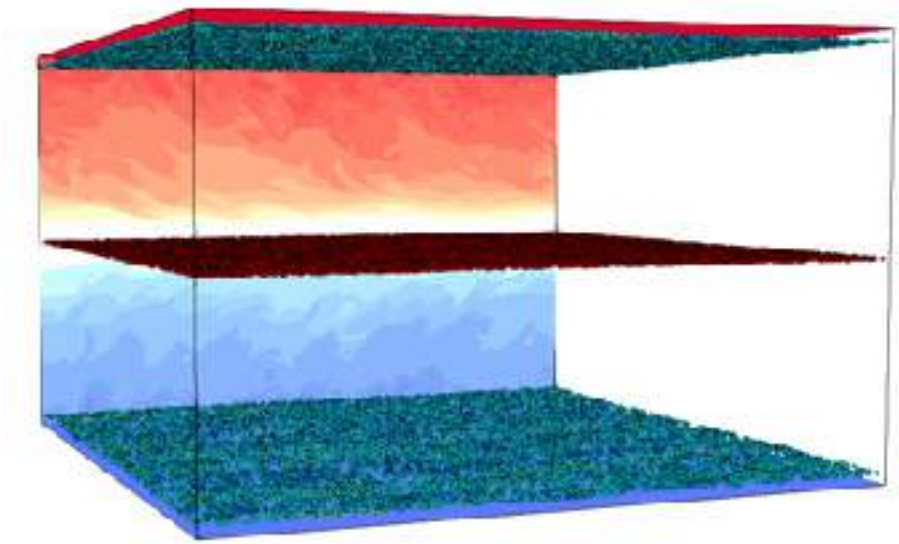
$Ri_\tau = 200$



- Stratified flow is ''faster'' and less homogeneous
- Flow structures in case of stratification are smaller and are ''pushed'' towards the wall

Very important topic: particles dynamics in stratified turbulence

(this is why we have problems with pollution in winter)



Neutrally-buoyant particles in stratified flow

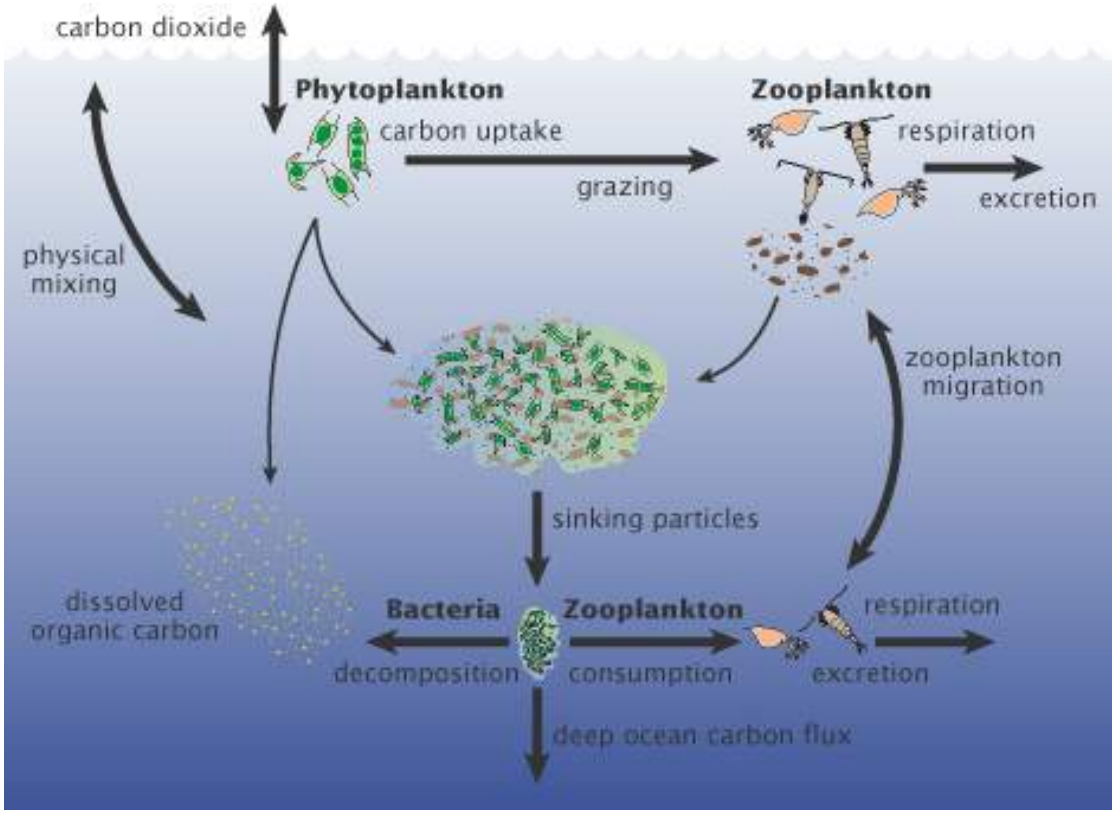
Pollutants are constraint in the proximity of the boundary (in the atmospheric boundary layer)

Fluid dynamics: computations we do

(@Institute of Fluid Mechanics and Heat Transfer, Group of Prof. Soldati)

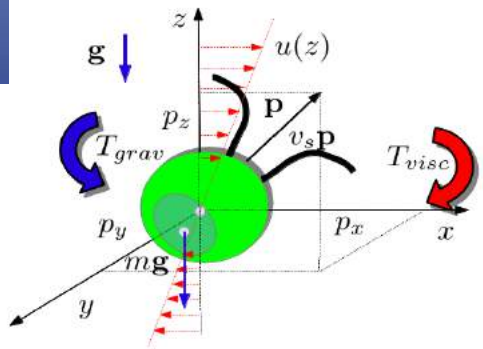
1. Oil transport
2. Turbulent flow in channels and pipelines: effect of stratification
3. Ocean/atmosphere CO_2 exchange: phytoplankton
4. Ocean/atmosphere CO_2 exchange: effect of waves
5. Oil/water emulsions
6. Oil/water emulsions with surfactants
7. Flows in porous media

Phytoplankton

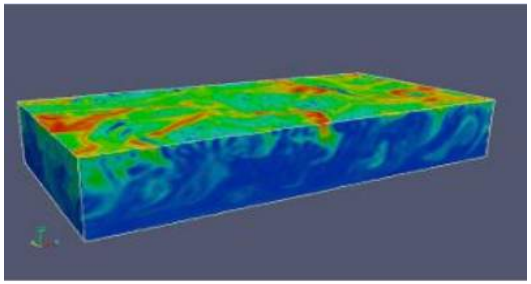


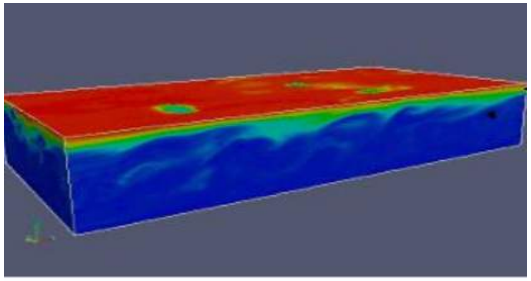
Phytoplankton is the photosynthetic part of plankton

- Primary production: organize compounds from CO2
- Important part of the global carbon cycle
- Provides 50% of the earth's oxygen
- Sustain the aquatic food web

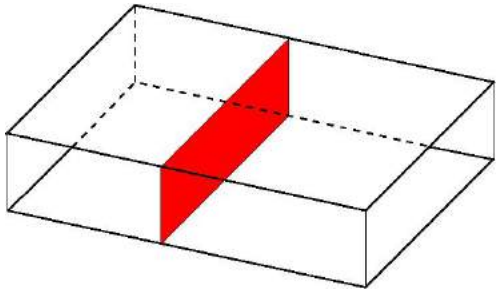


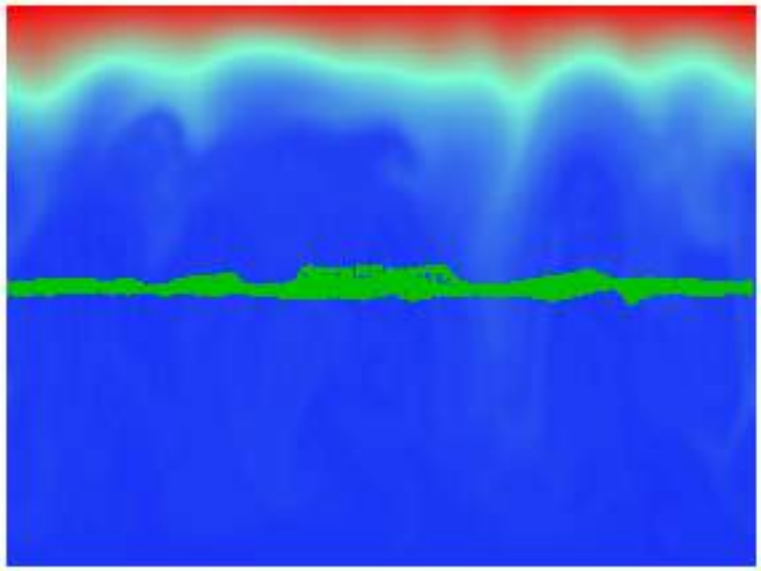
Phytoplankton





Phytoplankton





Fluid dynamics: computations we do

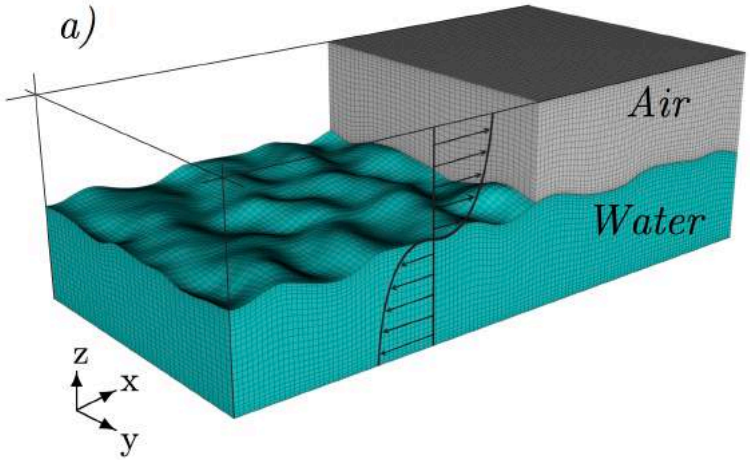
(@Institute of Fluid Mechanics and Heat Transfer, Group of Prof. Soldati)

1. Oil transport
2. Turbulent flow in channels and pipelines: effect of stratification
3. Ocean/atmosphere CO_2 exchange: phytoplankton
4. Ocean/atmosphere CO_2 exchange: effect of waves
5. Oil/water emulsions
6. Oil/water emulsions with surfactants
7. Flows in porous media

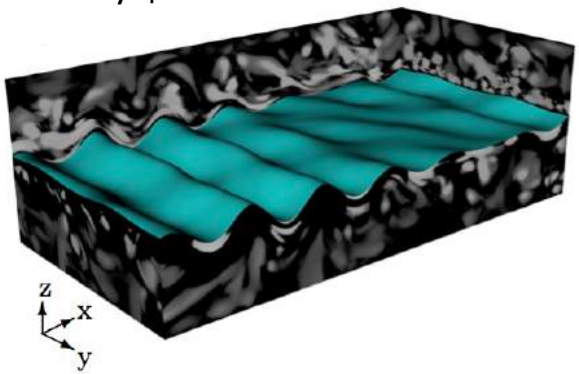
Waves

Transfer rates (for example of CO_2) across an interface depends strongly on the extension of the interface…

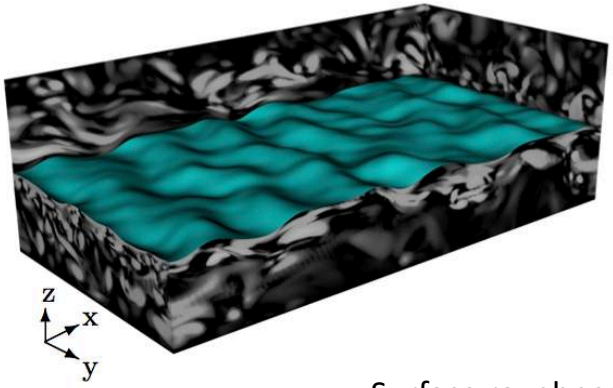
Waves



Gravity ↑

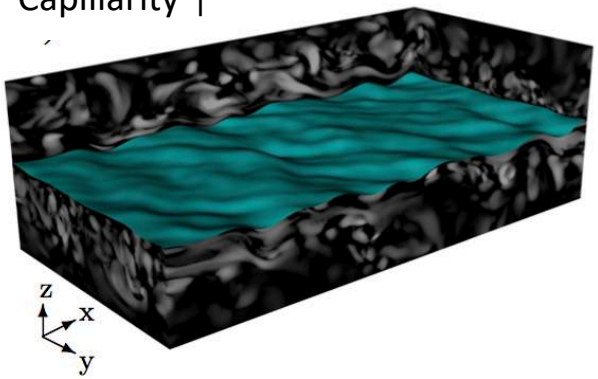


Quasy-sinusoidal wavy interface

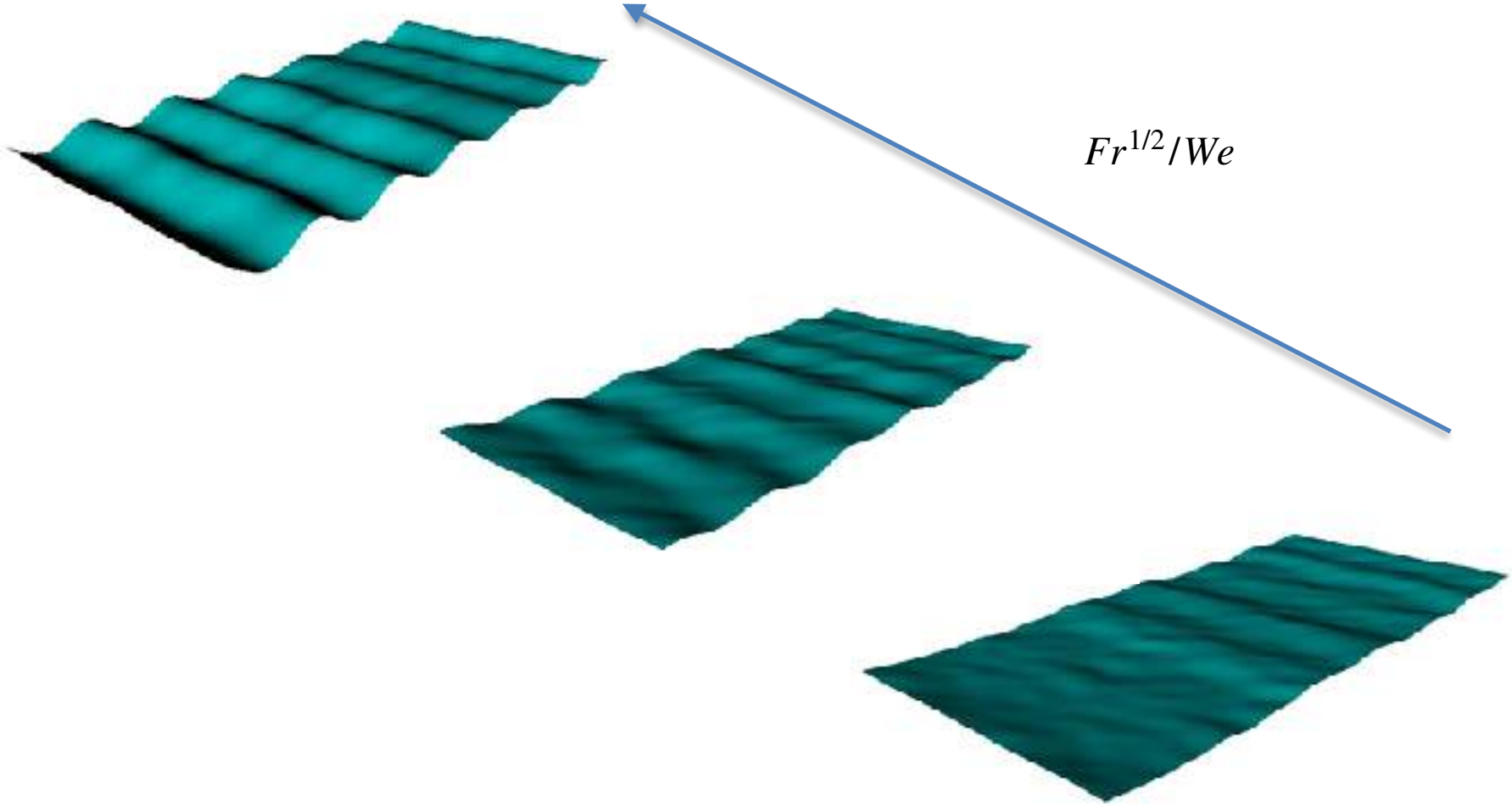


Surface roughness

Capillarity ↑



Waves

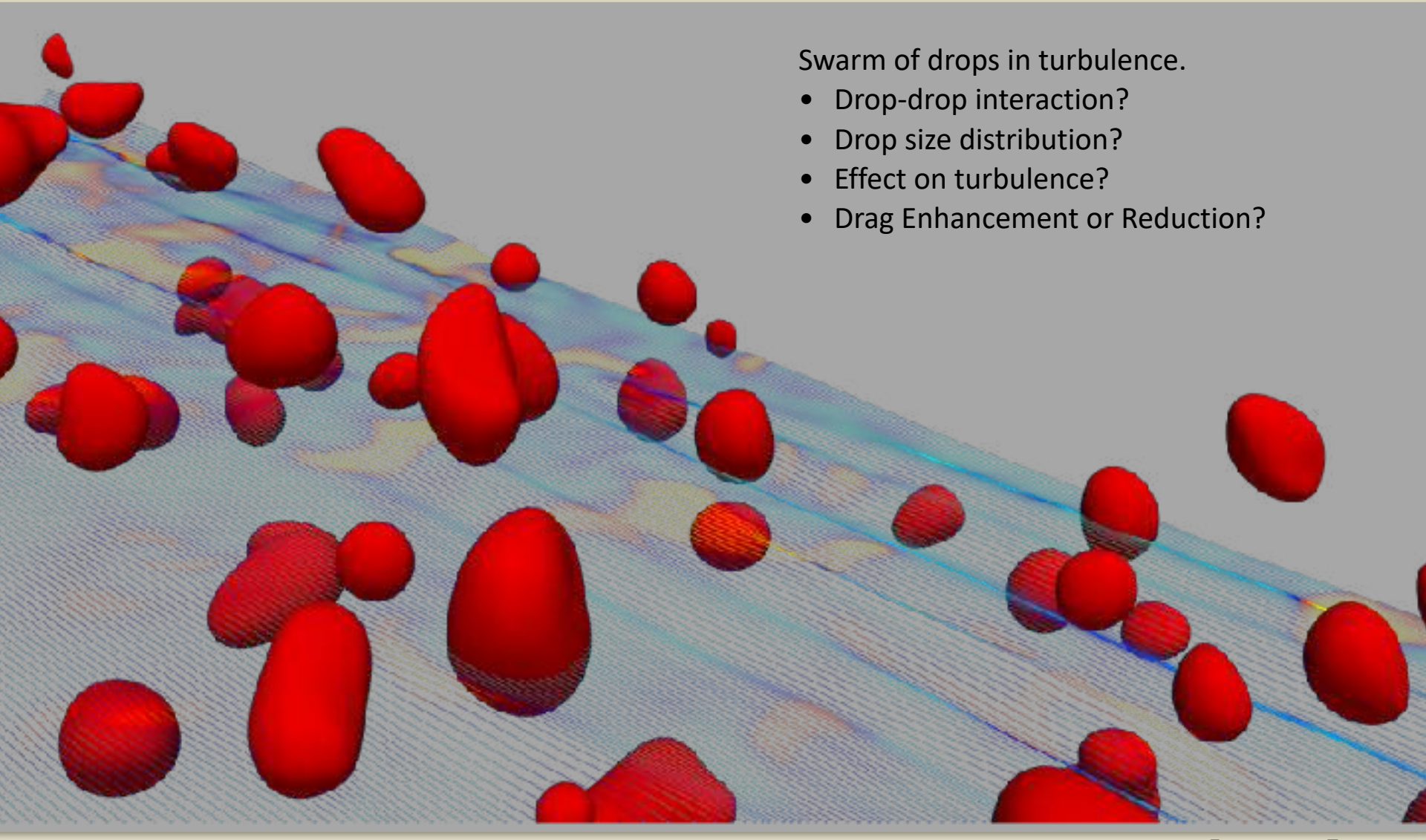


Fluid dynamics: computations we do

(@Institute of Fluid Mechanics and Heat Transfer, Group of Prof. Soldati)

1. Oil transport
2. Turbulent flow in channels and pipelines: effect of stratification
3. Ocean/atmosphere CO_2 exchange: phytoplankton
4. Ocean/atmosphere CO_2 exchange: effect of waves
5. Oil/water emulsions
6. Oil/water emulsions with surfactants
7. Flows in porous media

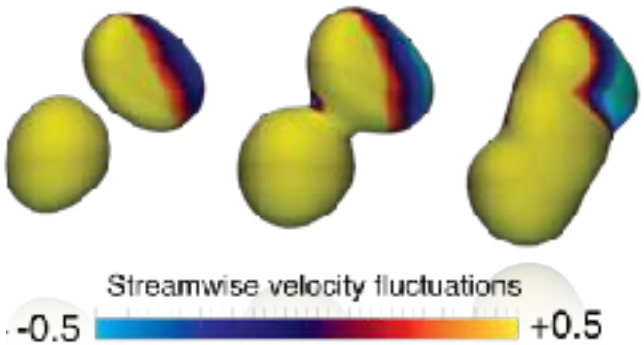
Oil/Water emulsions



Swarm of drops in turbulence.

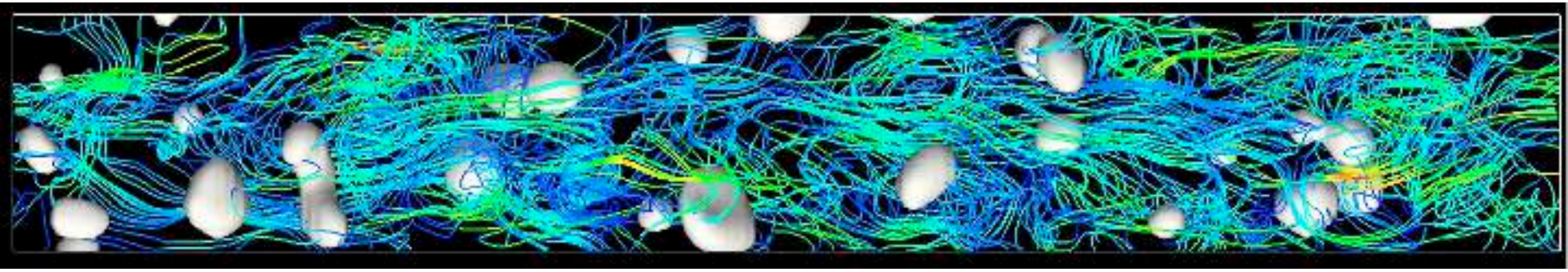
- Drop-drop interaction?
- Drop size distribution?
- Effect on turbulence?
- Drag Enhancement or Reduction?

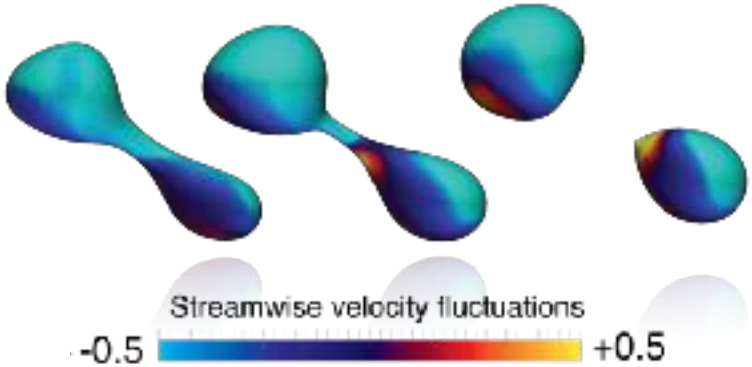
Oil/Water emulsions



COALESCENCE

Two droplets come close and collide due to turbulence fluctuations. During the collision, a small bridge is initially formed; later, surface tension (which tends to reshape the droplet) comes into the picture and complete the coalescence process.





BREAK-UP

A droplet is subjected to a sufficient shear stress, such that it is deformed and stretched until the emerging thin liquid bridge is broken (due to surface tension that acts minimizing the energy stored at the interface).

Fluid dynamics: computations we do

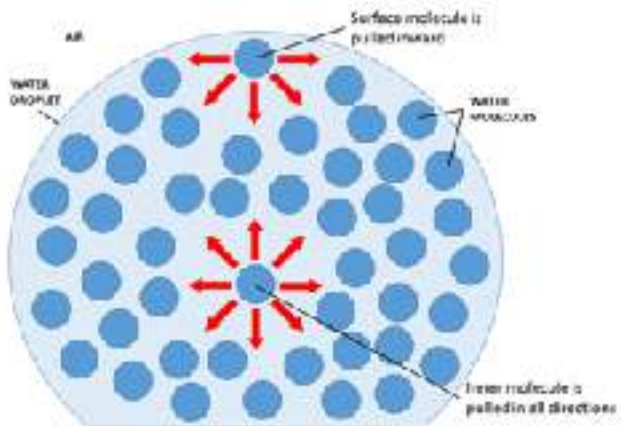
(@Institute of Fluid Mechanics and Heat Transfer, Group of Prof. Soldati)

1. Oil transport
2. Turbulent flow in channels and pipelines: effect of stratification
3. Ocean/atmosphere CO_2 exchange: phytoplankton
4. Ocean/atmosphere CO_2 exchange: effect of waves
5. Oil/water emulsions
6. Oil/water emulsions with surfactants
7. Flows in porous media

Oil/Water emulsions with surfactants

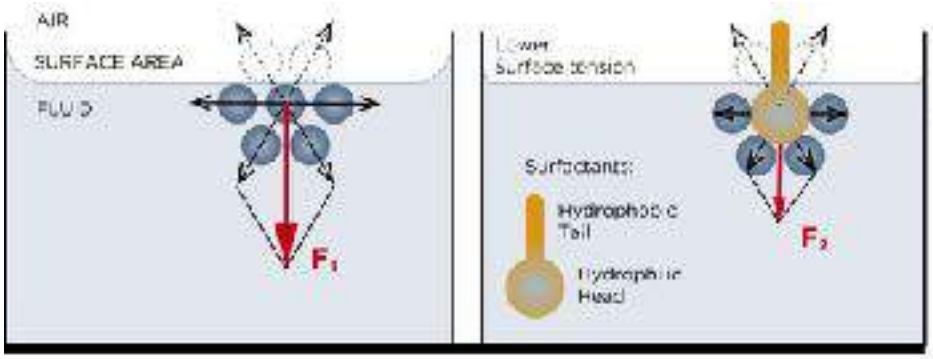
Surface Tension:

The cohesive forces between liquid molecules are the responsible for the phenomenon known as surface tension.



Addition of a surfactant:

Clean system | Surfactant-laden system



Presence of surfactant decreases the surface tension and as a consequence:

-Droplets become more deformable (less surface tension).

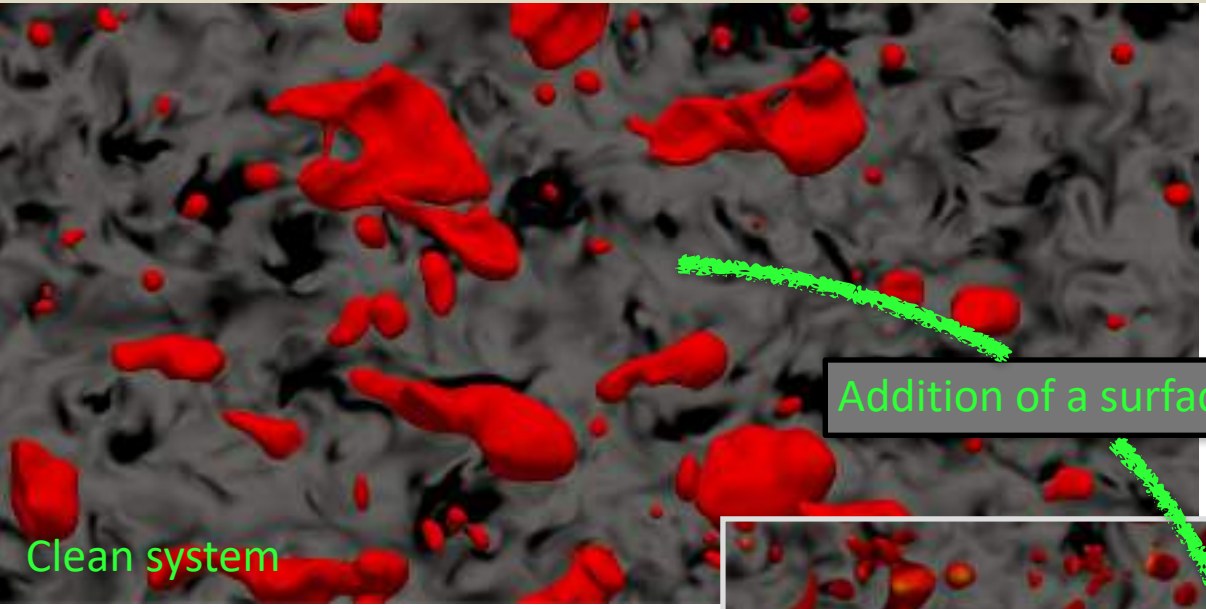
-When non-uniform can give rise to tangential forces (Marangoni).

Oil/Water emulsions with surfactants

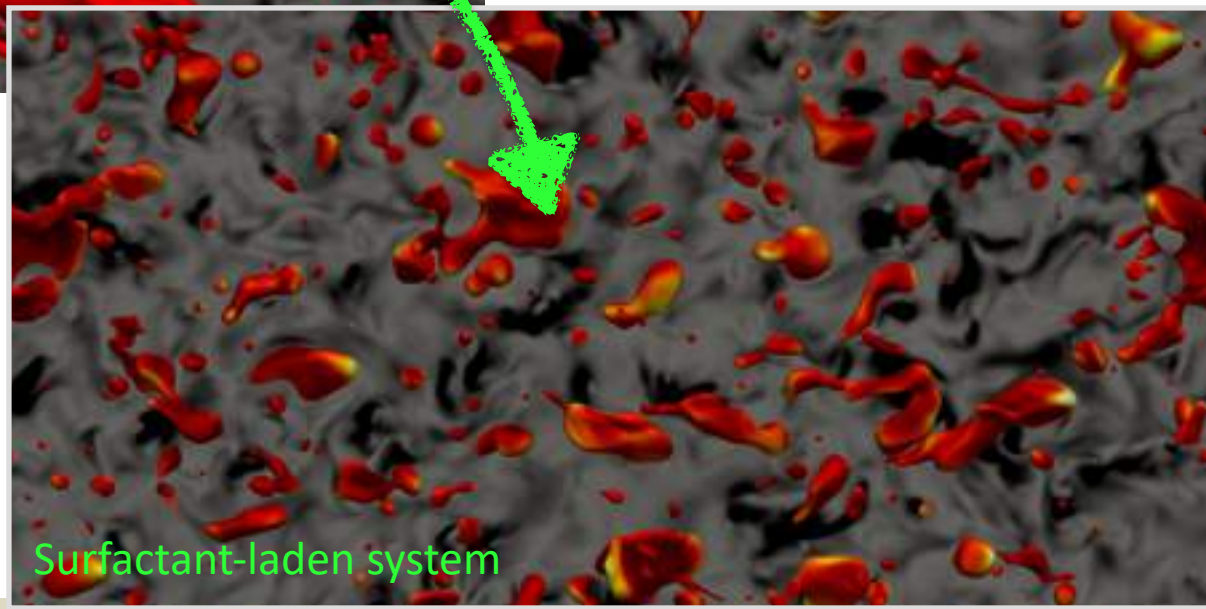
Clean droplets (no surfactant):

Surfactant-laden droplets:

Oil/Water emulsions with surfactants



Addition of a surfactant



Effects of the surfactant:

- Dispersed phase morphology?
- Droplet size distribution?
- Surfactant distribution?
- Turbulence modifications?

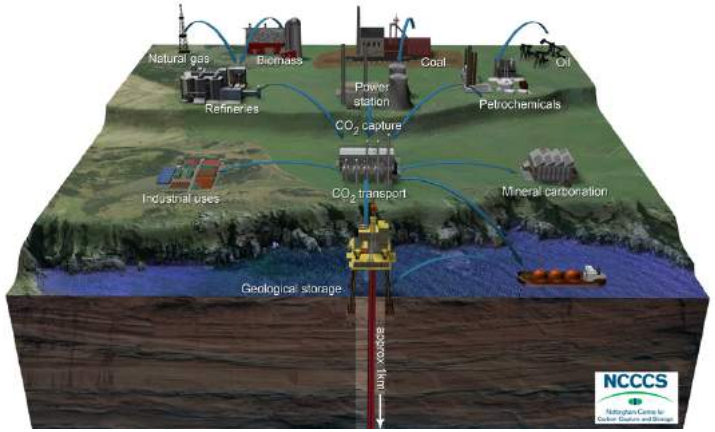
Francesco Zonta

Fluid dynamics: computations we do

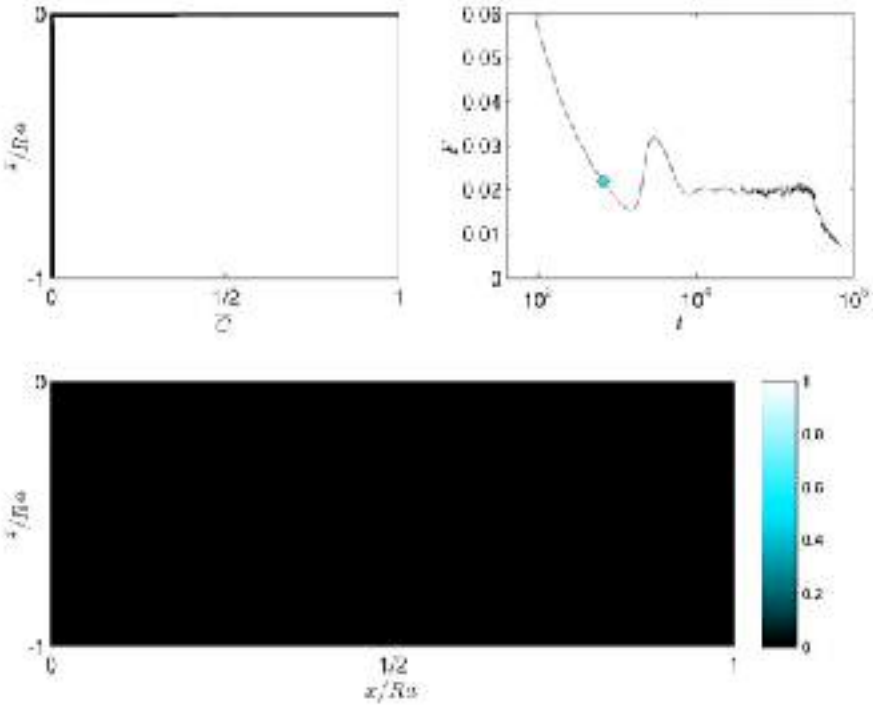
(@Institute of Fluid Mechanics and Heat Transfer, Group of Prof. Soldati)

1. Oil transport
2. Turbulent flow in channels and pipelines: effect of stratification
3. Ocean/atmosphere CO_2 exchange: phytoplankton
4. Ocean/atmosphere CO_2 exchange: effect of waves
5. Oil/water emulsions
6. Oil/water emulsions with surfactants
7. Flows in porous media

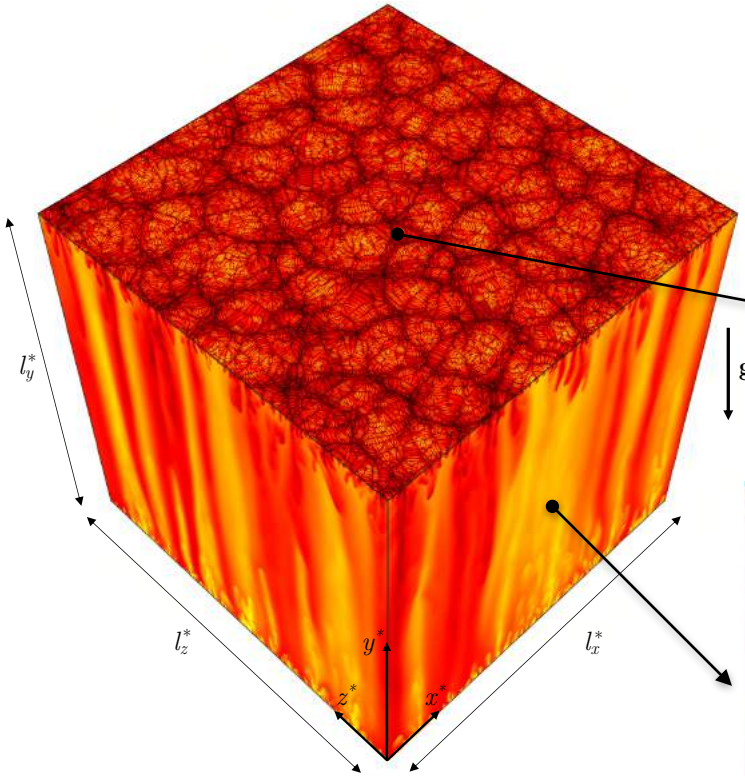
Flows in porous media

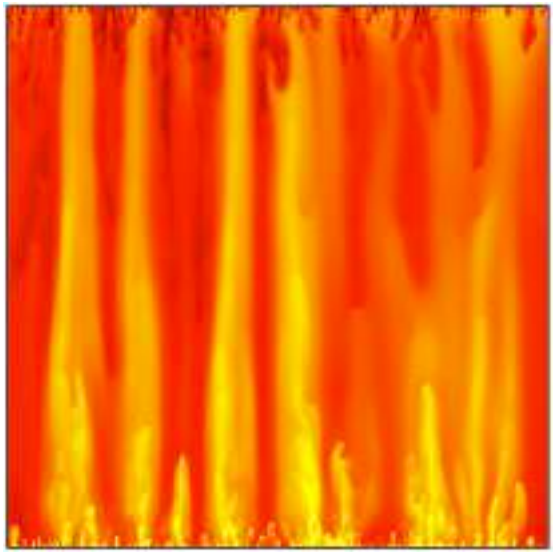


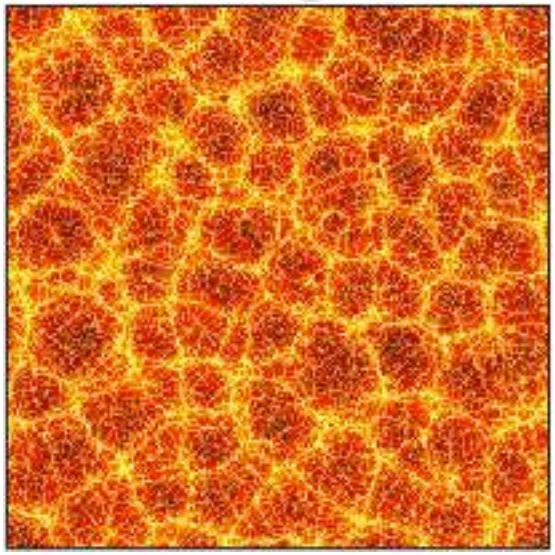
Geological CO2 sequestration



Flows in porous media







A lot of open questions: *role of rock anisotropy & pore scale dynamics*

Study Assistent 1+2 (for 4 months / 5 months)
At the Institute of Visual Computing and Human-Centered Technology, in the Research Unit of Virtual and Augmented Reality, TU Wien is offering a position as student assistant (Studienassistent_in, all genders) for 20 hours/week. Expected start: July 2023.

Tasks:

- Contributing to a project that brings simulation results from CAE (Computer Aided Engineering) software to stand-alone XR headsets
- Rendering of reconstructed geometries and physical field quantities
- Implementing XR interactions with objects (e.g. geometric transformations, volumetric data (e.g. defining cutting planes) and simulation (e.g. particles that can be released and followed on their way through the flow)
- Implementation of (network) interfaces for communication between CAE software (e.g. Ansys FLUENT or OpenFOAM), the XR application and auxiliary tools (e.g. file parsers for exported data in cases where CAE software does not provide suitable API)

Your profile:

- Studies in computer science, mechanical engineering, physics, mathematics
- Programming skills in Unity, C#
- Experience with XR interaction
- Knowledge of rendering pipelines and shader programming advantageous
- C++ advantageous
- Experience with Android programming advantageous
- Team spirit

Study Assistent 3: (for 6 months)
At the Institute of Fluid Mechanics and Heat Transfer, TU Wien is offering a position as student assistant (Studienassistent_in, all genders) for 20 hours/week. Expected start: July 2023.

Tasks:

- Contributing to a project that brings simulation results from CAE (Computer Aided Engineering) software to stand alone XR headsets
- Implementation of (network) interfaces for communication between CAE software (e.g. Ansys FLUENT, OpenFOAM or Custom Fortran code), the XR application and auxiliary tools (e.g. file parsers for exported data in cases where CAE software does not provide suitable API).
- Implementation of auxiliary tools (e.g. file parsers for exported data in cases where CAE software does not provide suitable API).
- Implementation of software modules that read, interpret, resample (if necessary) and write data (geometries and physical field quantities) from the CAE software in datastructures ready for rendering

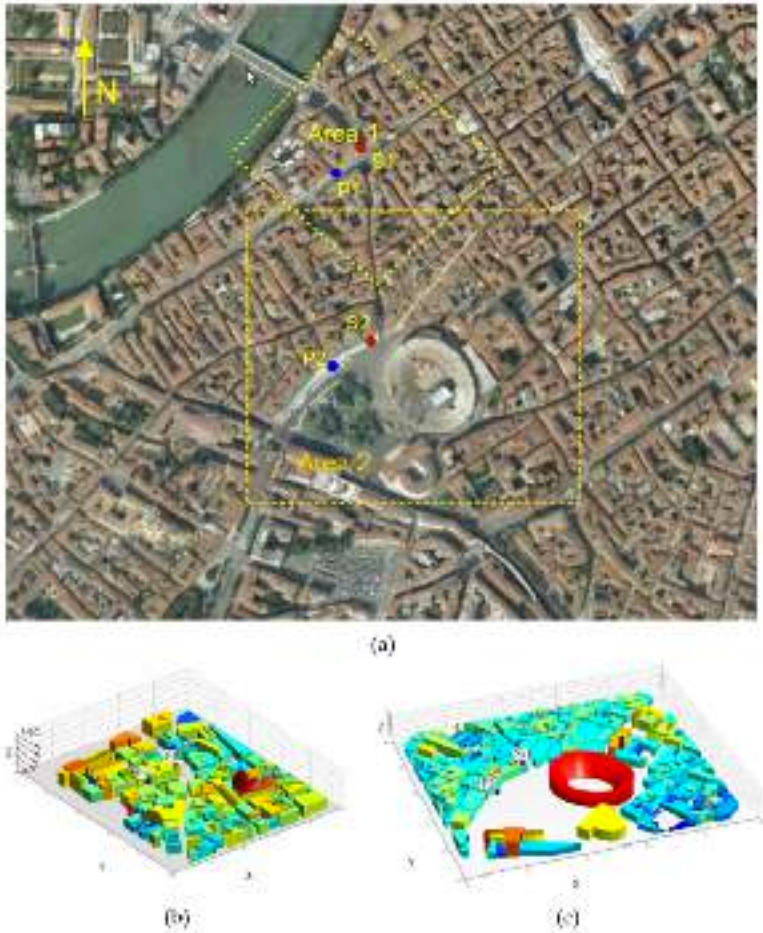
Your profile:

- Studies in computer science, mechanical engineering, physics, mathematics
- Programming skills in C# or C++
- Knowledge in Fortran advantageous
- Experience in parsing of binary file formats beneficial
- Knowledge of Unity advantageous
- Experience with Ansys FLUENT or OpenFOAM beneficial
- Team spirit

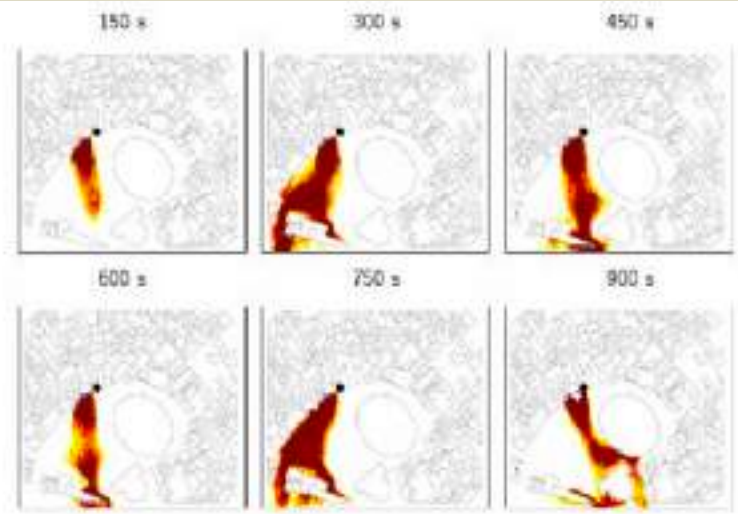
After this survey on applications, we are now ready to start with an in-depth analysis of transport process and equations…

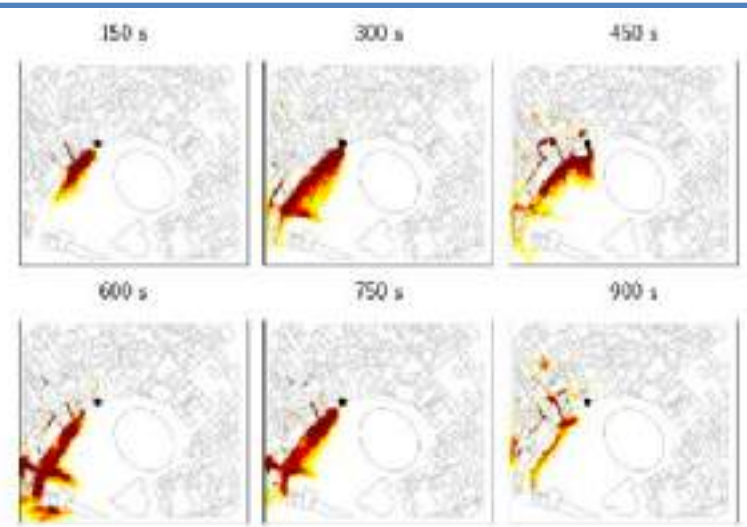
The main character of the story: The transport process/equation! What is it?

Verona Arena



Pettarin et al (2015), *Atmospheric Environment*,**122**, 74-82





We take it as reference!

What do we need to simulate it?

1.**First we need a mathematical model (PDE+BC+IC) of the transport process**: concentration of a chemical species C_i that diffuses and at the same time is advected by the flow. This gives:

$$\boxed{\frac{\partial C_i}{\partial t} + u_j \frac{\partial C_i}{\partial x_j} = D \frac{\partial^2 C_i}{\partial x_j^2}}$$

ADE=Advection Diffusion Equation

Rate of change — Advection — Diffusion

2.**Then we need a discretization strategy** (numerical scheme) to switch from the continuum to the discrete space (Space & Time). The specific method (FD,FE,FV,spectral methods…) is not important here; what matters is the "philosophy"

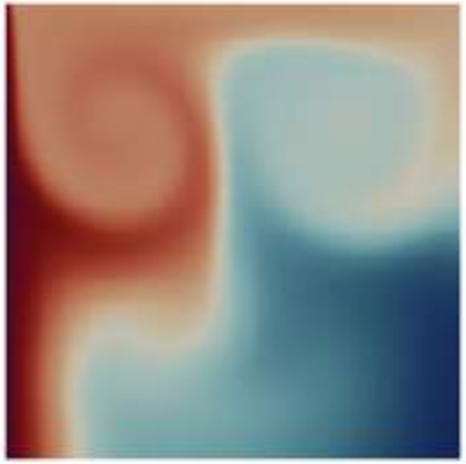
Before starting: What you will learn

1. You will learn the fundamental concepts on how to solve (numerically) a generic transport equation.

2. I will try to do some “surgery” of the equations, analyzing different aspects one by one (and neglecting other aspects).

3. I will skip some details and technicalities (and I would refer you to specific courses for the purpose).

Maybe at the end of the lecture you will learn how to obtain this:



Distribution of concentration (red= high concentration) in a square cavity filled with air

Introduction: Diffusion and advection

Let's consider our transport equation (1D ADE)

$$\boxed{\frac{\partial C^*}{\partial t^*} + u^* \frac{\partial C^*}{\partial x^*} = D \frac{\partial^2 C^*}{\partial x^{*2}}}$$

We have 1st and 2nd derivatives!

In dimensionless form:

$$\begin{cases} C = \dfrac{C^*}{C_w} \\ x = \dfrac{x^*}{L} \\ u = \dfrac{u^*}{u_{ref}} \\ t = \dfrac{t^* D}{L^2} \end{cases} \rightarrow \frac{D C_w}{L^2} \frac{\partial C}{\partial t} + \frac{u_{ref}\, u\, C_w}{L} \frac{\partial C}{\partial x} = \frac{D C_w}{L^2} \frac{\partial^2 C}{\partial x^2}$$

$$\frac{\partial C}{\partial t} + \frac{u_{ref} \cdot \cancel{C_w}}{\cancel{L}} \frac{L^{\cancel{2}}}{D \cancel{C_w}} u \frac{\partial C}{\partial x} = \frac{\partial^2 C}{\partial x^2}$$

$$\frac{\partial C}{\partial t} + \underbrace{\frac{u_{ref} \cdot L}{D}}_{Pe\ (Peclet)} u \frac{\partial C}{\partial x} = \frac{\partial^2 C}{\partial x^2}$$

$$\left[Pe = \frac{\text{diffusion time}}{\text{convection time}} = \frac{L^2/D}{L/u_{ref}} = \frac{L \cdot u_{ref}}{D} \right]$$

$$\boxed{\frac{\partial C}{\partial t} + Pe \cdot u \frac{\partial C}{\partial x} = \frac{\partial^2 C}{\partial x^2}}$$

$Pe \ll 1 \Rightarrow$ Diffusion dominates $\rightarrow \boxed{\frac{\partial C}{\partial t} = \frac{\partial^2 C}{\partial x^2}}$

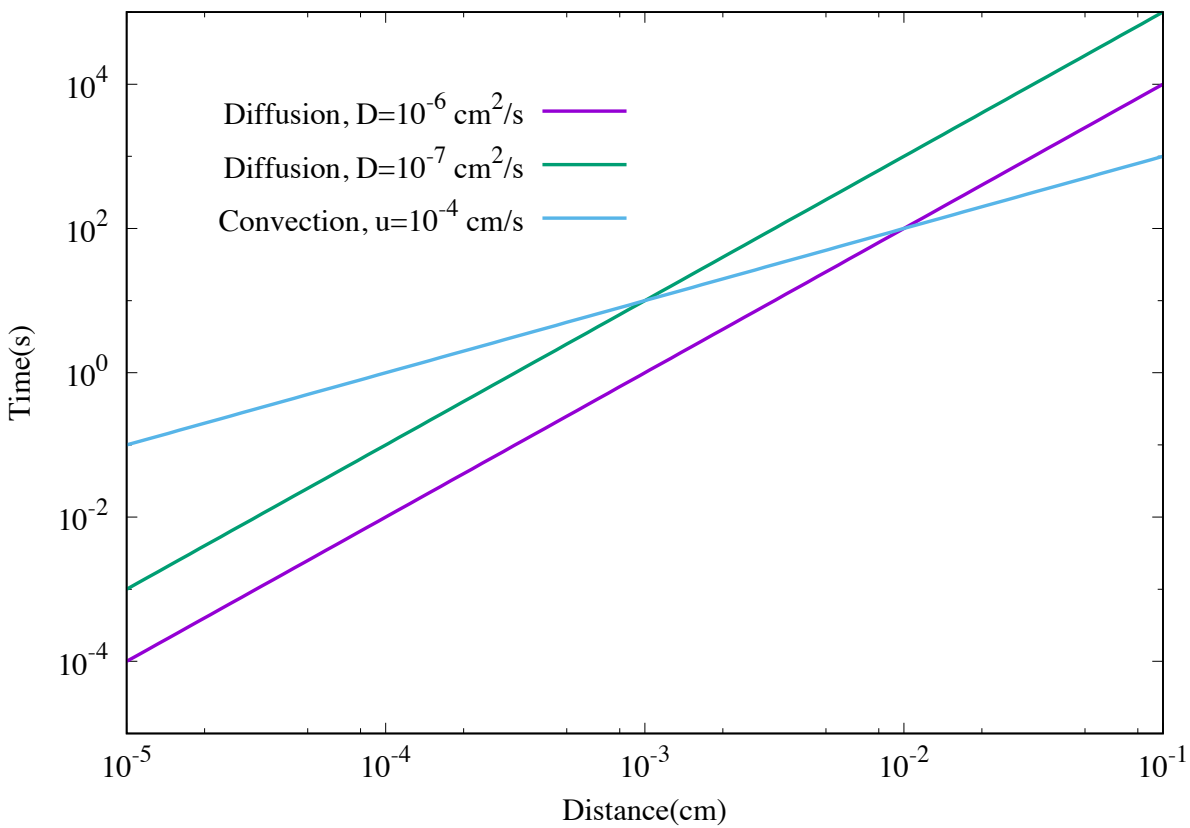
$Pe \gg 1 \Rightarrow$ Advection dominates $\rightarrow \boxed{\frac{\partial C}{\partial t} + Pe \cdot u \frac{\partial C}{\partial x} = 0}$

Introduction: Diffusion and advection

Question: Diffusion or Advection, which one is more efficient?

Introduction: Diffusion and advection

Question: Diffusion or Advection, which one is more efficient?



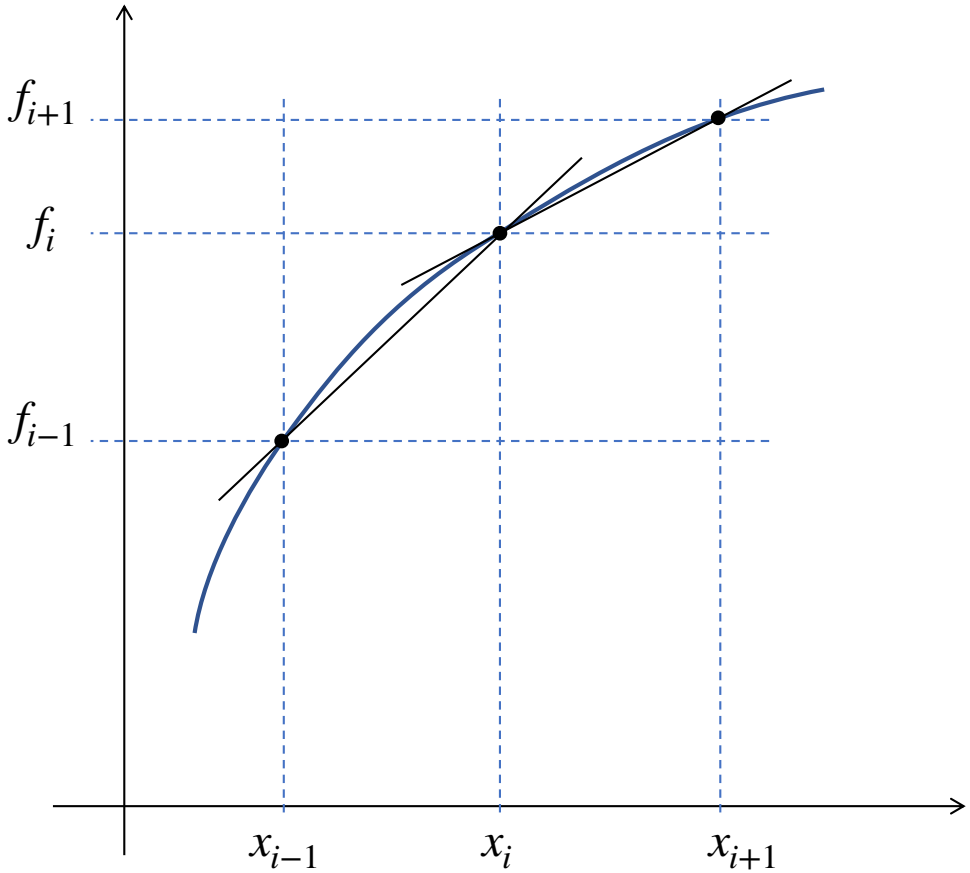
$$T_{diff} = \frac{L^2}{D}$$

$$T_{conv} = \frac{L}{U_{ref}}$$

Now we have a physical/mathematical model (ADE); we need a strategy to discretize and solve it!

Introduction: Discretization strategy

How to approximate (discretize) derivatives? The most intuitive way is the following:



(at a given time instant *n*)

Forward approximation

$$\left[\frac{\partial f}{\partial x}\right]_i^n \simeq \frac{f_{i+1}^n - f_i^n}{\Delta x}$$

Backward approximation

$$\left[\frac{\partial f}{\partial x}\right]_i^n \simeq \frac{f_i^n - f_{i-1}^n}{\Delta x}$$

Are they accurate?

Which one is better?

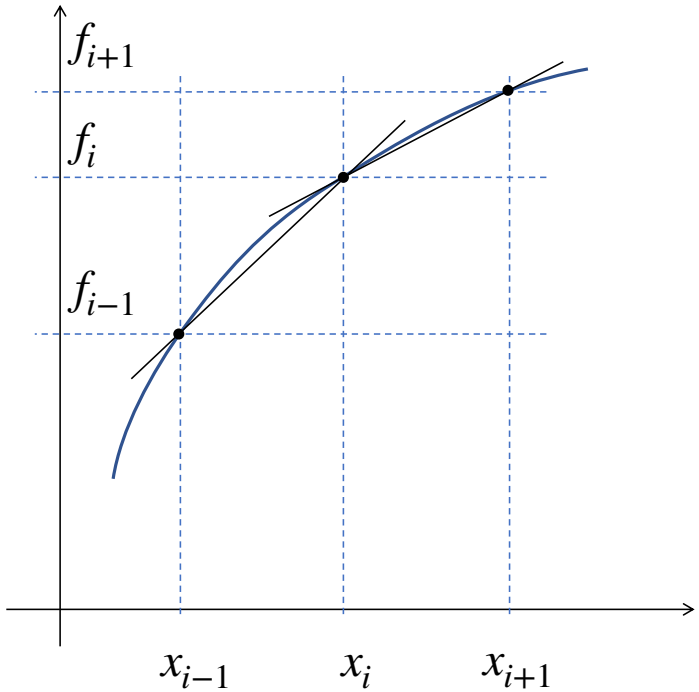
To answer: Taylor series…

Introduction: Discretization strategy

Taylor series

$$f_{i+1}^n = f_i^n + \Delta x \left[\frac{\partial f}{\partial x}\right]_i^n + \frac{\Delta x^2}{2}\left[\frac{\partial^2 f}{\partial x^2}\right]_i^n + \underbrace{O(\Delta x^3)}_{HOT}$$

$$f_{i-1}^n = f_i^n - \Delta x \left[\frac{\partial f}{\partial x}\right]_i^n + \frac{\Delta x^2}{2}\left[\frac{\partial^2 f}{\partial x^2}\right]_i^n + HOT$$



It is straightforward to show that

$$\left[\frac{\partial f}{\partial x}\right]_i^n = \frac{f_{i+1}^n - f_i^n}{\Delta x} - \frac{\Delta x}{2}\left[\frac{\partial^2 f}{\partial x^2}\right]_i^n + HOT$$

$$\left[\frac{\partial f}{\partial x}\right]_i^n = \frac{f_i^n - f_{i-1}^n}{\Delta x} + \frac{\Delta x}{2}\left[\frac{\partial^2 f}{\partial x^2}\right]_i^n + HOT$$

Our approximation — Error

Forward/Backward discretization=1st order!!

"Manipulation" of Taylor series. In a more systematic way…

Introduction: Discretization strategy

We wish to approximate the derivatives as follows

$$\boxed{\left[\frac{\partial f}{\partial x}\right]_i^n = a f_{i-1}^n + b f_i^n + c f_{i+1}^n + O(\Delta x^m)}$$

f_{i-1} and f_{i+1} written as Taylor series

$$f_{i-1}^n = f_i^n - \Delta x \left[\frac{\partial f}{\partial x}\right]_i^n + \frac{\Delta x^2}{2}\left[\frac{\partial^2 f}{\partial x^2}\right] - \frac{\Delta x^3}{3!}\left[\frac{\partial^3 f}{\partial x^3}\right] + \text{HOT}$$

$$f_{i+1} = f_i^n + \Delta x \left[\frac{\partial f}{\partial x}\right]_i^n + \frac{\Delta x^2}{2}\left[\frac{\partial^2 f}{\partial x^2}\right] + \frac{\Delta x^3}{3!}\left[\frac{\partial^3 f}{\partial x^3}\right] + \text{HOT}$$

This finally gives

$$\left[\frac{\partial f}{\partial x}\right]_i^n = a\left[f_i^n - \Delta x\left[\frac{\partial f}{\partial x}\right]_i^n + \frac{\Delta x^2}{2}\left[\frac{\partial^2 f}{\partial x^2}\right] - \frac{\Delta x^3}{3!}\left[\frac{\partial^3 f}{\partial x^3}\right] + O(\Delta x^4)\right] + \quad f_{i-1}$$

$$+ b f_i^n \quad f_i$$

$$+ c\left[f_i^n + \Delta x\left[\frac{\partial f}{\partial x}\right]_i^n + \frac{\Delta x^2}{2}\left[\frac{\partial^2 f}{\partial x^2}\right] + \frac{\Delta x^3}{3!}\left[\frac{\partial^3 f}{\partial x^3}\right] + O(\Delta x^4)\right] \quad f_{i+1}$$

Introduction: Discretization strategy

After some algebra

$$\left[\frac{\partial f}{\partial x}\right]_i^n = (a+b+c)\, f_i^n + \Delta x (c-a)\left[\frac{\partial f}{\partial x}\right]_i^n + \frac{\Delta x^2}{2}(a+c)\left[\frac{\partial^2 f}{\partial x^2}\right]_i^n + (c-a)\frac{\Delta x^3}{3}\left[\frac{\partial^3 f}{\partial x^3}\right]_i^n$$

$$\begin{cases} a+b+c=0 \\ (c-a)\Delta x = 1 \\ (a+c)\frac{\Delta x^2}{2} = 0 \end{cases} \rightarrow \quad c = a + \frac{1}{\Delta x}\,; \quad \boxed{a = -\frac{1}{2\Delta x}\,, \quad c = \frac{1}{2\Delta x}\,, \quad b = 0}$$

Therefore

$$\boxed{\left[\frac{\partial f}{\partial x}\right]_i^n = \frac{f_{i+1}^n - f_{i-1}^n}{2\Delta x} - \overbrace{\frac{\Delta x^2}{6}}^{(c-a)\frac{\Delta x^3}{6}}\left[\frac{\partial^3 f}{\partial x^3}\right]_i^n}$$

Central discretization in space (CS)

2nd order accurate formulation

Introduction: Discretization strategy

Following the same strategy, we can obtain an expression for the second derivative:

$$\left[\frac{\partial^2 f}{\partial x^2}\right]_i^n = a f_{i-1}^n + b f_i^n + c f_{i+1}^n + O(\Delta x^m)$$

And, employing the Taylor series expansion of the function at x_{i-1} and x_{i+1}

$$\left[\frac{\partial^2 f}{\partial x^2}\right]_i^n = a\left(f_i^n - \Delta x\left[\frac{\partial f}{\partial x}\right]_i^n + \frac{\Delta x^2}{2}\left[\frac{\partial^2 f}{\partial x^2}\right]_i^n - \frac{\Delta x^3}{3!}\left[\frac{\partial^3 f}{\partial x^3}\right]_i^n + HOT\right) +$$

$$+ b f_i^n$$

$$+ c\left(f_i^n + \Delta x\left[\frac{\partial f}{\partial x}\right]_i^n + \frac{\Delta x^2}{2}\left[\frac{\partial^2 f}{\partial x^2}\right]_i^n + \frac{\Delta x^3}{3!}\left[\frac{\partial^3 f}{\partial x^3}\right]_i^n + HOT\right)$$

$$\left[\frac{\partial^2 f}{\partial x^2}\right]_i^n = \underbrace{(a+b+c)}_{a+b+c=0} f_i^n + \underbrace{\Delta x(c-a)}_{\Delta x(c-a)=0}\left[\frac{\partial f}{\partial x}\right]_i^n + \underbrace{\frac{\Delta x^2}{2}(a+c)}_{(a+c)\frac{\Delta x^2}{2}=1}\left[\frac{\partial^2 f}{\partial x^2}\right]_i^n + \frac{\Delta x^3}{3!}(c-a)\left[\frac{\partial^3 f}{\partial x^3}\right]_i^n +$$

Introduction: Discretization strategy

$$\begin{cases} a+b+c=0 \\ (c-a)\Delta x=0 \\ (a+c)\frac{\Delta x^2}{2}=1 \end{cases} \rightarrow c=a\ ;\ a=\frac{1}{\Delta x^2},\quad c=\frac{1}{\Delta x^2},\quad b=-\frac{2}{\Delta x^2}$$

Therefore, we get

$$\boxed{\left[\frac{\partial^2 f}{\partial x^2}\right]_i^n = \frac{f_{i-1}^n - 2f_i^n + f_{i+1}^n}{\Delta x^2} + \underbrace{\frac{\Delta x^2}{12}}_{(a+c)\frac{\Delta x^4}{4!}}\left[\frac{\partial^4 f}{\partial x^4}\right]}$$

2nd order accurate

To summarize

$$\boxed{\begin{aligned} \left[\frac{\partial f}{\partial x}\right]_i^n &= \frac{f_{i+1}^n - f_{i-1}^n}{2\Delta x} \\ \left[\frac{\partial^2 f}{\partial x^2}\right]_i^n &= \frac{f_{i-1}^n - 2f_i^n + f_{i+1}^n}{\Delta x^2} \end{aligned}}$$

Central discretization in space (CS)

It's all we need for the moment: a model and a discretization strategy!!

Recall: Diffusion and advection

Let's consider our transport equation (1D ADE)

$$\boxed{\frac{\partial C^*}{\partial t^*} + u^* \frac{\partial C^*}{\partial x^*} = D \frac{\partial^2 C^*}{\partial x^{*2}}}$$

In dimensionless form:

$$\begin{cases} C = \dfrac{C^*}{C_w} \\ x = \dfrac{x^*}{L} \\ u = \dfrac{u^*}{u_{ref}} \\ t = \dfrac{t^* D}{L^2} \end{cases} \rightarrow \frac{D C_w}{L^2} \frac{\partial C}{\partial t} + \frac{u_{ref}\, u\, C_w}{L} \frac{\partial C}{\partial x} = \frac{D C_w}{L^2} \frac{\partial^2 C}{\partial x^2}$$

$$\frac{\partial C}{\partial t} + \frac{u_{ref} \cdot \cancel{C_w}}{\cancel{L}} \frac{L^{\cancel{2}}}{D \cancel{C_w}} u \frac{\partial C}{\partial x} = \frac{\partial^2 C}{\partial x^2}$$

$$\frac{\partial C}{\partial t} + \underbrace{\frac{u_{ref} \cdot L}{D}}_{Pe\ (Peclet)} u \frac{\partial C}{\partial x} = \frac{\partial^2 C}{\partial x^2}$$

$$\left[Pe = \frac{\text{diffusion time}}{\text{convection time}} = \frac{L^2/D}{L/u_{ref}} = \frac{L \cdot u_{ref}}{D} \right]$$

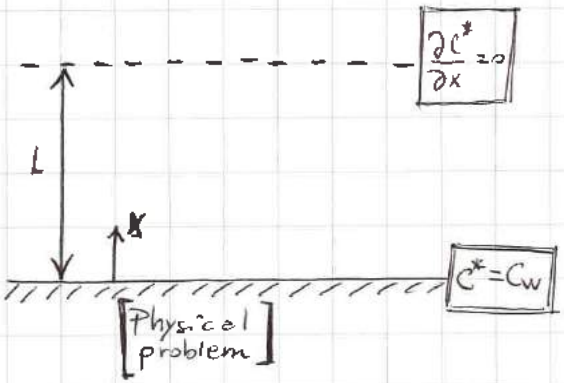
$$\boxed{\frac{\partial C}{\partial t} + Pe \cdot u \frac{\partial C}{\partial x} = \frac{\partial^2 C}{\partial x^2}}$$

$Pe \ll 1 \Rightarrow$ Diffusion dominates $\rightarrow \boxed{\frac{\partial C}{\partial t} = \frac{\partial^2 C}{\partial x^2}}$

$Pe \gg 1 \Rightarrow$ Advection dominates $\rightarrow \boxed{\frac{\partial C}{\partial t} + Pe \cdot u \frac{\partial C}{\partial x} = 0}$

Diffusion: Explicit method

We focus first on diffusion:



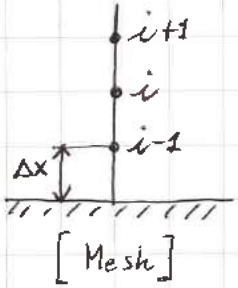
The governing equation is:

$$\boxed{\frac{\partial c}{\partial t} = \frac{\partial^2 c}{\partial x^2}}$$

Note: An analytical solution exists

$$\boxed{c = 1 - 2\sum_{n=0}^{\infty} \frac{(-1)^n}{(n+\frac{1}{2})\pi} \cos\left[(n+\tfrac{1}{2})\pi x\right] \exp\left[-(n+\tfrac{1}{2})^2\pi^2 t\right]}$$

Now we want to solve the problem numerically



Discretization in space $\Rightarrow$ $\left[\frac{\partial^2 c}{\partial x^2}\right]_i^n = \frac{c_{i-1}^n - 2c_i^n + c_{i+1}^n}{\Delta x^2}$

Discretization in time $\Rightarrow$ $\left[\frac{\partial c}{\partial t}\right]_i^n = \frac{c_i^{n+1} - c_i^n}{\Delta t}$

n=known

n+1=unknown

Therefore:

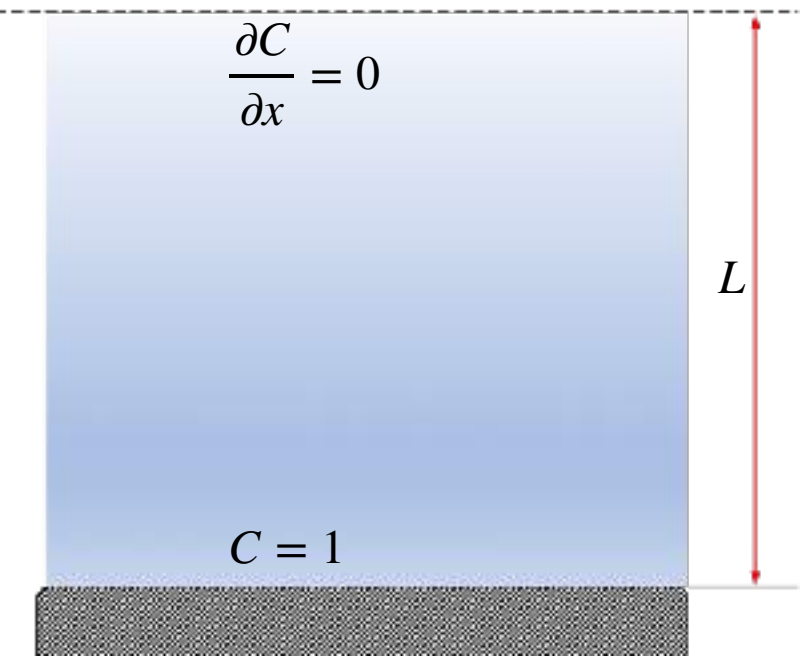
$$\boxed{c_i^{n+1} = c_i^n + \overbrace{\frac{\Delta t}{\Delta x^2}}^{s}\left(c_{i-1}^n - 2c_i^n + c_{i+1}^n\right)}$$

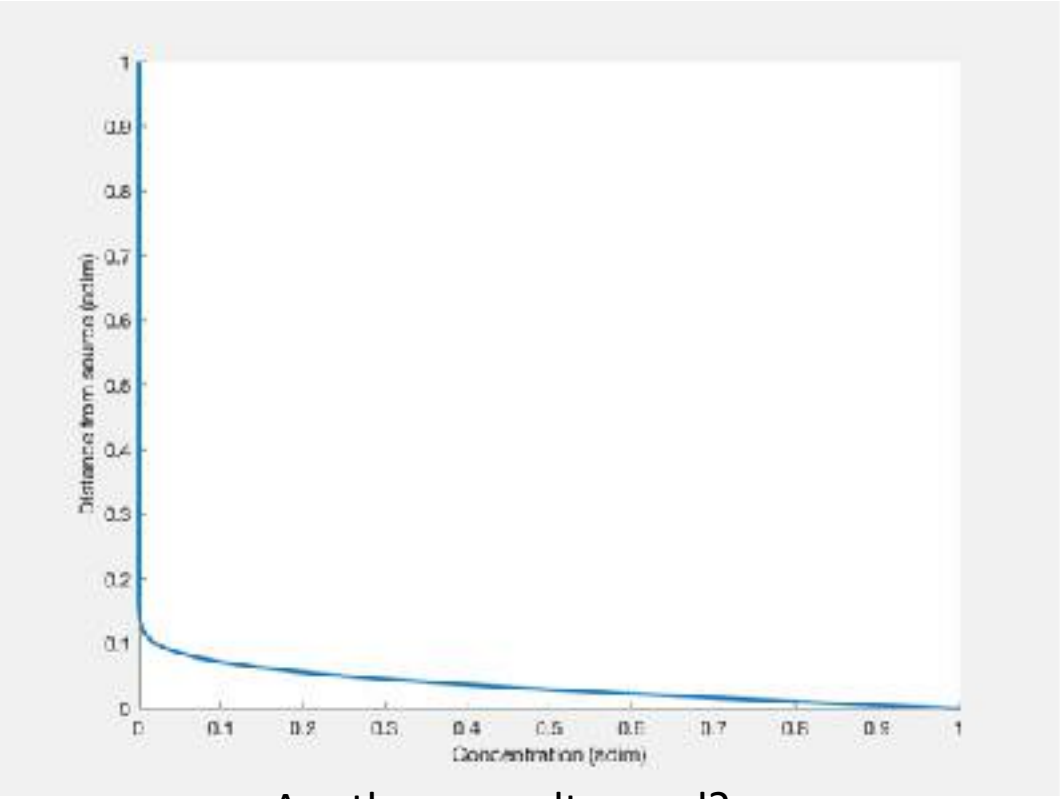
[Explicit method]

Diffusion
Explicit method, Numerical results

$$\Delta x = 4 \cdot 10^{-3}; \Delta t = 8 \cdot 10^{-6}; s = \frac{\Delta t}{\Delta x^2} = 0.5$$

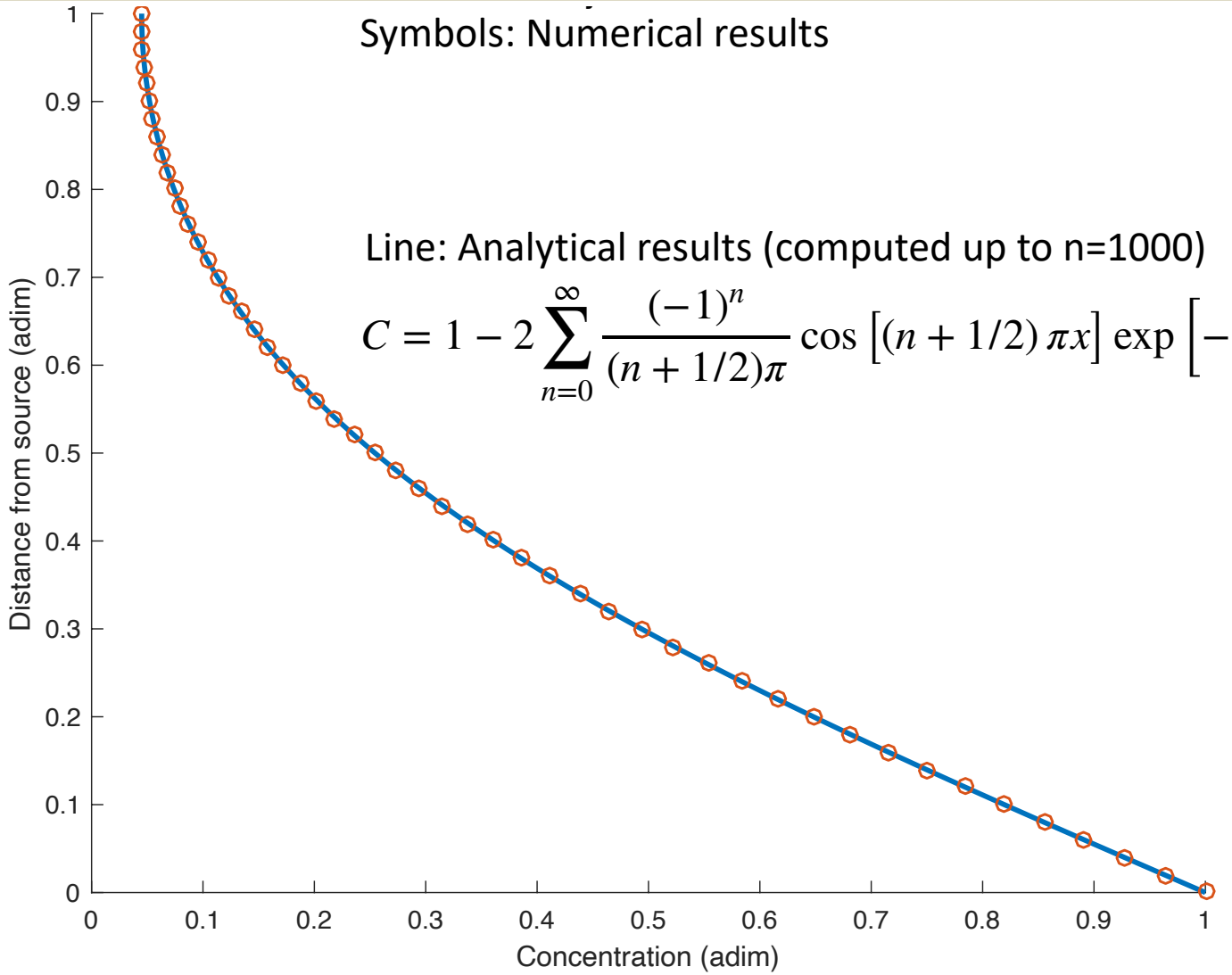
(*Note that: L=1)





Are these results good?

Diffusion
Explicit method: numerical vs analytical results



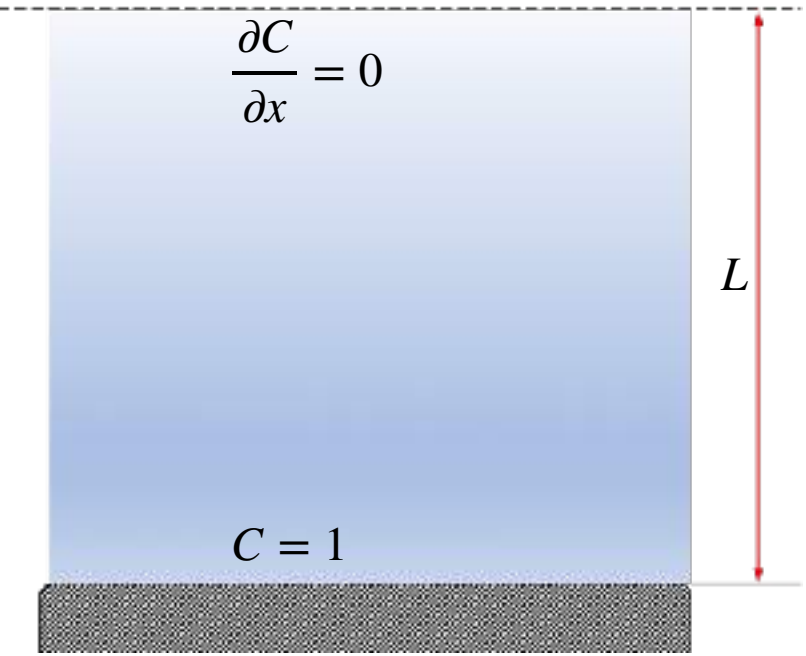
Symbols: Numerical results

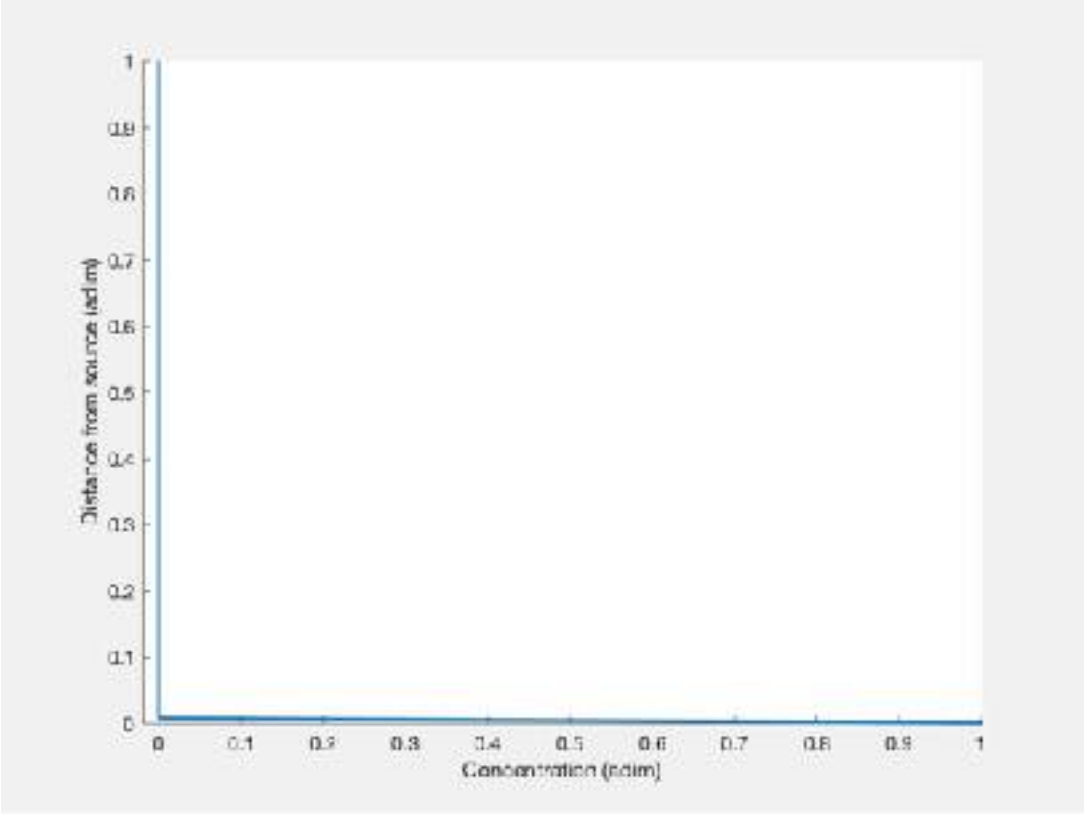
Line: Analytical results (computed up to n=1000)

$$C = 1 - 2\sum_{n=0}^{\infty} \frac{(-1)^n}{(n+1/2)\pi} \cos\left[(n+1/2)\,\pi x\right] \exp\left[-(n+1/2)^2\,\pi^2 t\right]$$

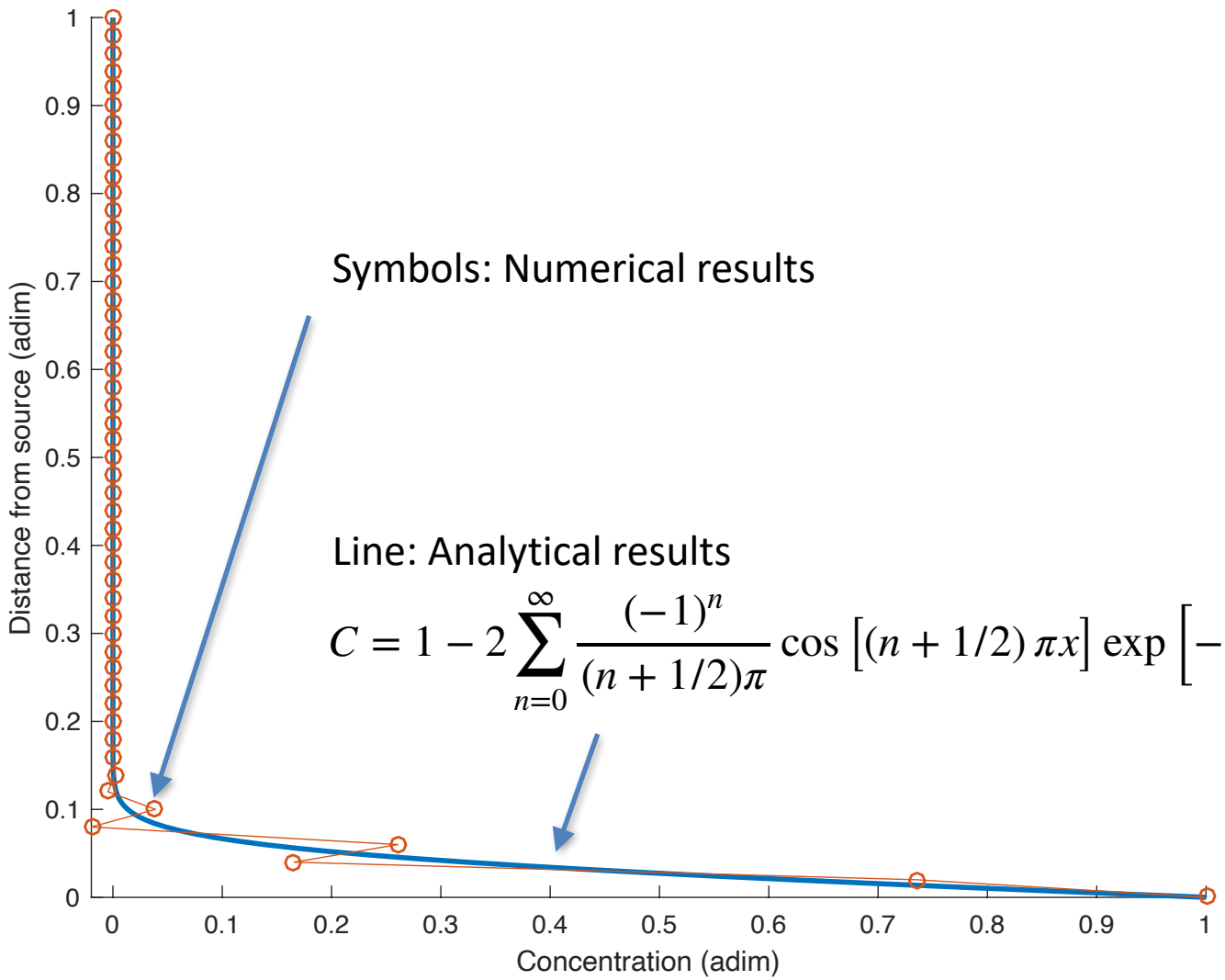
Diffusion
Explicit method: numerical results-2

$$\Delta x = 4 \cdot 10^{-3}; \Delta t = 8.2 \cdot 10^{-6}; s = \frac{\Delta t}{\Delta x^2} = 0.5125$$





Diffusion
Explicit method: numerical vs analytical results-2



Symbols: Numerical results

Line: Analytical results

$$C = 1 - 2\sum_{n=0}^{\infty} \frac{(-1)^n}{(n+1/2)\pi} \cos\left[(n+1/2)\,\pi x\right] \exp\left[-(n+1/2)^2\,\pi^2 t\right]$$

Diffusion
Implicit method

It can be shown that the explicit method is stable only if:

$$\boxed{s = \frac{\Delta t}{\Delta x^2} \leq 0.5}$$

$$\boxed{C_i^{n+1} = C_i^n + \frac{\Delta t}{\Delta x^2}\left(C_{i-1}^n - 2C_i^n + C_{i+1}^n\right)}$$

Otherwise, it is unstable.

→ To overcome this problem, we can use an implicit scheme:

$$\left[\frac{\partial C}{\partial t}\right]_i^n = \frac{C_i^{n+1} - C_i^n}{\Delta t}$$

$$\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n = \frac{C_{i-1}^{n+1} - 2C_i^{n+1} + C_{i+1}^{n+1}}{\Delta x^2}$$

[Evaluated at the new time step "n+1"]

$$\boxed{-sC_{i-1}^{n+1} + (1+2s)C_i^{n+1} - sC_{i+1}^{n+1} = C_i^n}$$

← But now we have a linear system of equations

Unknowns

Diffusion
Implicit method

$$\boxed{-s\,C_{i-1}^{n+1} + (1+2s)\,C_i^{n+1} - s\,C_{i+1}^{n+1} = C_i^{n}}$$

$$\underbrace{\begin{pmatrix} 1+2s & -s & & \\ -s & 1+2s & -s & \\ & \ddots & \ddots & \ddots \\ & & -s & 1+2s \end{pmatrix}}_{\text{Tridiagonal Matrix}} \underbrace{\begin{pmatrix} C_1^{n+1} \\ \vdots \\ C_i^{n+1} \\ \vdots \\ C_m^{n+1} \end{pmatrix}}_{\text{unew1}} = \underbrace{\begin{pmatrix} C_1^{n} \\ \vdots \\ C_i^{n} \\ \vdots \\ C_m^{n} \end{pmatrix}}_{\text{D}}$$

[Thomas algorithm]*

Code

```
function [ unew1 ] = tridiag(n,a,b,c,D)

alfa=zeros(n,1);
gamma=zeros(n,1);
v=zeros(n,1);
unew1=zeros(n,1);

%    Tridiagonal solver
    alfa(1)=a(1);
    gamma(1)=c(1)/(a(1));
    v(1)=D(1)/(a(1));

    for i=2:n
      alfa(i)=a(i)-b(i)*gamma(i-1);
      gamma(i)=c(i)/alfa(i);
      v(i)=(D(i)-b(i)*v(i-1))/(alfa(i));
    end

    unew1(n)=v(n);

    for i=n-1:-1:1
        unew1(i)=v(i)-gamma(i)*unew1(i-1);
    end
```

If we now run the simulations...

*Thomas algorithm-details

Thomas algorithm

$$\underbrace{\begin{bmatrix} a_1 & c_1 & 0 & 0 \\ b_2 & a_2 & c_2 & 0 \\ 0 & b_3 & a_3 & c_3 \\ 0 & 0 & b_4 & a_4 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} \alpha_1 & 0 & 0 & 0 \\ \beta_2 & \alpha_2 & 0 & 0 \\ 0 & \beta_2 & \alpha_3 & 0 \\ 0 & 0 & \beta_4 & \alpha_4 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} 1 & \gamma_1 & 0 & 0 \\ 0 & 1 & \gamma_2 & 0 \\ 0 & 0 & 1 & \gamma_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{U}$$

Suppose we want to solve: $\boxed{A \cdot \overset{\text{Variable}}{C} = R}$

$$\begin{cases} A \cdot C = R \\ \uparrow \\ A = L \cdot U \end{cases} \Rightarrow L \underbrace{U \cdot C}_{y} = R \xrightarrow{\text{Decomposed}} \begin{cases} L \cdot y = R & \text{Problem A} \\ U \cdot C = y & \text{Problem B} \end{cases}$$

*Thomas algorithm-details

Problem A

$$\underbrace{\begin{pmatrix} \alpha_1 & 0 & 0 & 0 \\ \beta_2 & \alpha_2 & 0 & 0 \\ 0 & \beta_3 & \alpha_3 & 0 \\ 0 & 0 & \beta_4 & \alpha_4 \end{pmatrix}}_{L} \cdot \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \end{bmatrix}}_{R}$$

Note: $\alpha_i, \beta_i, \gamma_i$ from $A = L \cdot U$

$\alpha_1 = a_1$

$\alpha_1 \gamma_1 = c_1 \rightarrow \gamma_1 = \frac{c_1}{\alpha_1}$

$\beta_2 = b_2$

$a_2 = \beta_2 \gamma_1 + \alpha_2 \rightarrow \alpha_2 = a_2 - b_2 \frac{c_1}{a_1}$

$c_2 = \alpha_2 \gamma_2 \rightarrow \gamma_2 = \frac{c_2}{\alpha_2}$

$$\Downarrow$$

$$\alpha_1 y_1 = R_1 \rightarrow y_1 = \frac{R_1}{\alpha_1} = \frac{R_1}{a_1}$$

$$\beta_2 y_1 + \alpha_2 y_2 = R_2 \rightarrow \begin{cases} y_2 = \frac{R_2 - \beta_2 y_1}{\alpha_2} \\ \alpha_2 = a_2 - b_2 \frac{c_1}{a_1} \end{cases}$$

For a generic node i:

loop over i

$$\begin{cases} \alpha_i = a_i - b_i \gamma_{i-1} \\ \beta_i = b_i \\ \gamma_i = c_i / \alpha_i \\ y_i = (R_i - b_i y_{i-1}) / \alpha_i \end{cases}$$

*Thomas algorithm-details

Problem B

$$\underbrace{\begin{pmatrix} 1 & \gamma_1 & 0 & 0 \\ 0 & 1 & \gamma_2 & 0 \\ 0 & 0 & 1 & \gamma_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{U} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

$$U \cdot C = y$$

Starting from the bottom:

$$c_4 = y_4$$

$$c_3 + \gamma_3 c_4 = y_3 \rightarrow c_3 = y_3 - \gamma_3 c_4$$

$$\vdots$$

In general

$$c_n = y_n$$

loop over i $\Rightarrow$ $\left[c_i = y_i - \gamma_i y_{i+1} \right.$

Note on Neumann BC for the implicit method

Neumann BC (assume it is imposed @x=0) means that:

$$\frac{\partial C}{\partial x}\Big|_{x=0} = c \rightarrow \frac{C_2^{n+1} - C_0^{n+1}}{2\Delta x} = c$$

The equation written @x=0 (i.e. i=1):

$$\begin{cases} (1+2s)\,C_1^{n+1} - s\,C_2^{n+1} - s\,C_0^{n+1} = C_1^{n+1} \\ C_2^{n+1} - C_0^{n+1} = 2\Delta x \cdot c \rightarrow C_0^{n+1} = C_2^{n+1} - 2\Delta x \cdot c \end{cases}$$

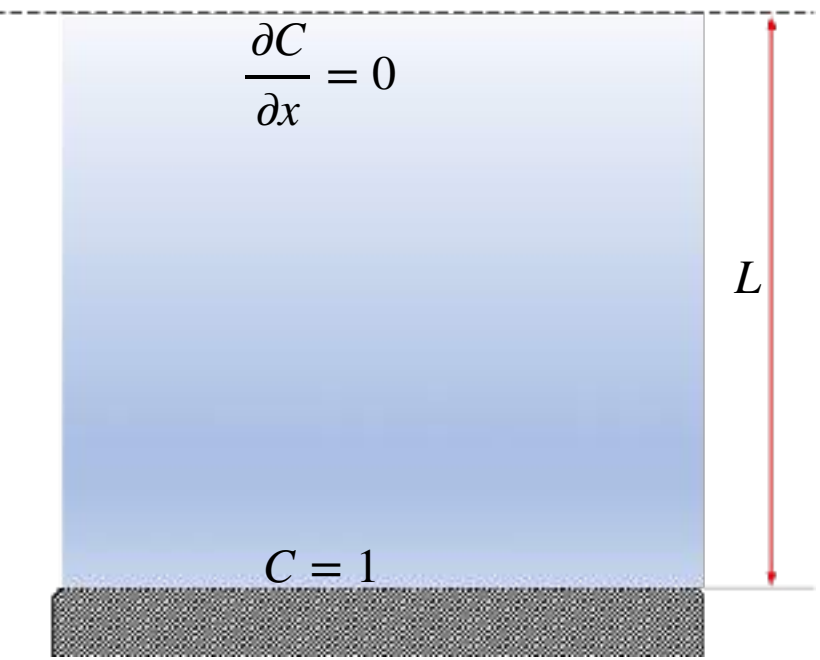
$$(1+2s)\,C_1^{n+1} - s\,C_2^{n+1} - s\,C_2^{n+1} - 2sc\Delta x = C_1^{n}$$

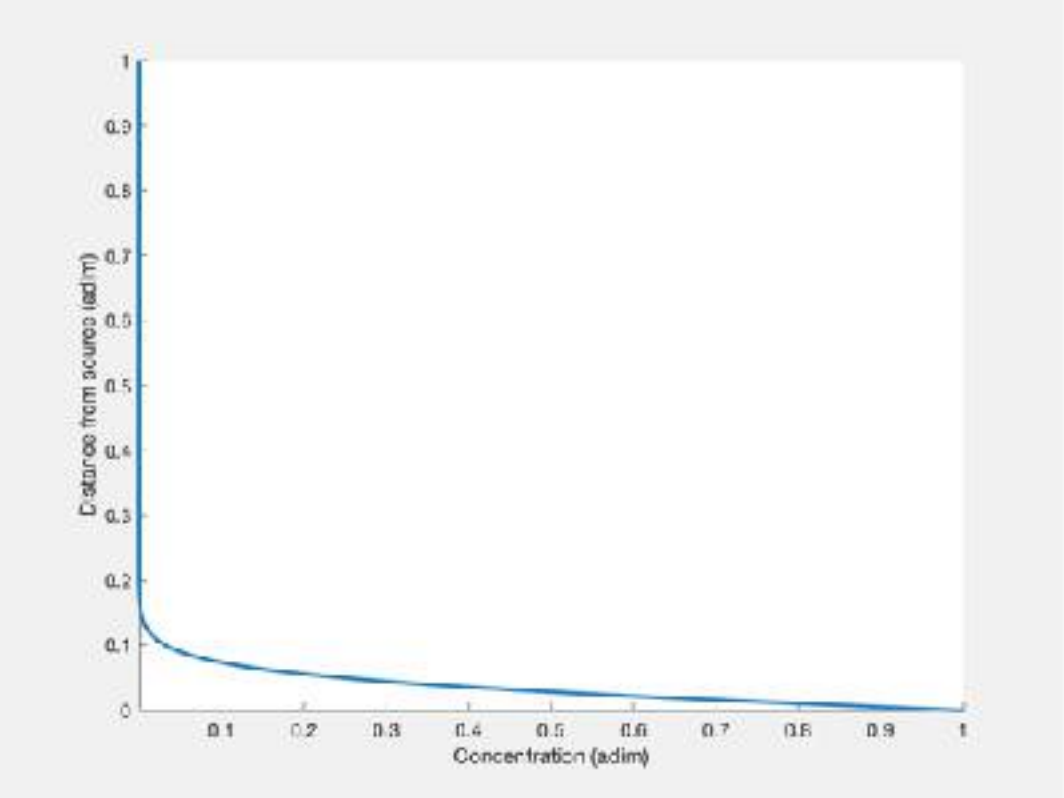
$$\boxed{(1+2s)\,C_1^{n+1} - s\,C_2^{n+1} - s\,C_2^{n+1} = C_1^{n} + 2sc\Delta x}$$

This means that with Dirichlet/Neumann BC we have $n = n_y - 1$ equations, whereas with Dirichlet/Dirichlet we have $n = n_y - 2$ equations

Diffusion
Implicit method, Numerical results

$$\Delta x = 4 \cdot 10^{-3}; \Delta t = 2 \cdot 10^{-4}; s = \frac{\Delta t}{\Delta x^2} = 12.5$$





A further consideration : Accuracy

Previous schemes are 1st order accurate in time. A 2nd order scheme would be desired in many applications

To do that → Crank-Nicolson scheme

$$\frac{\partial C}{\partial t} = F(c,x) \Rightarrow \left[\frac{\partial C}{\partial t}\right]_i^n = \frac{C_i^{n+1} - C_i^n}{\Delta t} = \frac{F^{n+1}(C,x) + F^n(C,x)}{2}$$

For the present case:

$$\frac{C_i^{n+1} - C_i^n}{\Delta t} = \frac{1}{2}\left[\frac{C_{i-1}^{n+1} - 2C_i^{n+1} + C_{i+1}^{n+1}}{\Delta x^2} + \frac{C_{i-1}^n - 2C_i^n + C_{i+1}^n}{\Delta x^2}\right]$$

Note:

$$s = \frac{1}{2}\frac{\Delta t}{\Delta x^2}$$

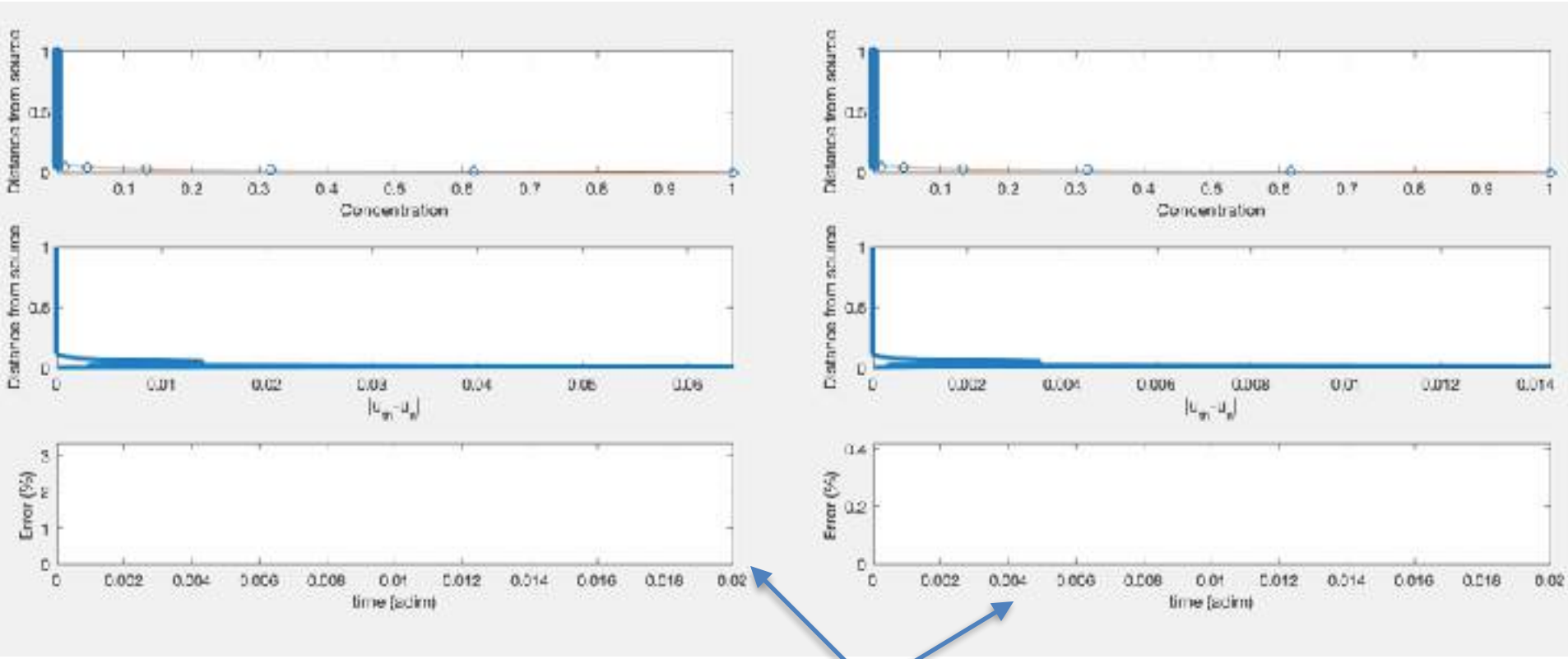
This equation can be rearranged as:

$$\boxed{-sC_{i-1}^{n+1} + (1+2s)C_i^{n+1} - sC_{i+1}^{n+1} = sC_{i-1}^n + (1-2s)C_i^n + sC_{i+1}^n}$$

[Thomas algorithm]

Diffusion
Implicit method: 1st vs 2nd order

$$\Delta x = 1 \cdot 10^{-2}; \Delta t = 1 \cdot 10^{-4}; s = \frac{\Delta t}{\Delta x^2} = 1$$

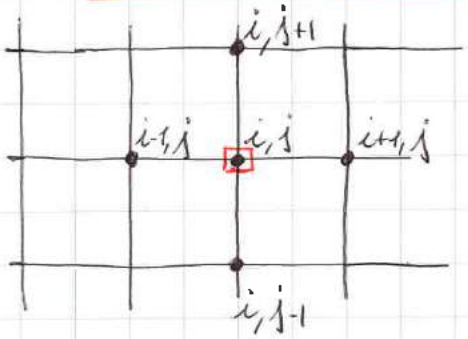


1st Order

2nd Order

(Percentage) Average error $\rightarrow \frac{1}{n_y} \sum |u_{th} - u_n|$

Further considerations: Extension to 2D case (brief discussion)



Now the stencil has five points, not three

With an explicit scheme:

$$\boxed{C_{i,j}^{n+1} - C_{i,j}^{n} = \frac{\Delta t \cdot D}{\Delta x^2}\left[C_{i-1,j}^{n} - 2C_{i,j}^{n} + C_{i+1,j}^{n}\right] + \frac{\Delta t \cdot D}{\Delta y^2}\left[C_{i,j-1}^{n} - 2C_{i,j}^{n} + C_{i,j+1}^{n}\right]}$$

For stability $\rightarrow \left(\frac{D\Delta t}{\Delta x^2} + \frac{D\Delta t}{\Delta y^2}\right) \leq \frac{1}{2}$ [severe condition]

Also, Fully implicit methods would lead to a pentadiagonal system (Time consuming)

↳ Solution 1: Splitting methods

↳ Solution 2: Iterative solvers (Jacobi, Gauss-Seidel, relaxation)

*Diffusion
Further considerations

Implicit formulation

$$\boxed{\frac{\partial C}{\partial t} = F^{n+1}(x, y, t)}$$

$$\frac{C_{i,j}^{n+1} - C_{i,j}^{n}}{\Delta t} = D_{ij}\left[\frac{C_{i+1,j}^{n+1} - 2C_{i,j}^{n+1} + C_{i-1,j}^{n+1}}{\Delta x^2} + \frac{C_{i,j+1}^{n+1} - 2C_{i,j}^{n+1} + C_{i,j-1}^{n+1}}{\Delta y^2}\right]$$

We obtain

$$\boxed{S_x C_{i+1,j}^{n+1} + S_x C_{i-1,j}^{n+1} - (2S_x + 2S_y - 1) C_{i,j}^{n+1} + S_y C_{i,j+1}^{n+1} + S_y C_{i,j-1}^{n+1} = C_{i,j}^{n}}$$

$$S_x = D_{ij}\frac{\Delta t}{\Delta x^2}$$

$$S_y = D_{ij}\frac{\Delta t}{\Delta y^2}$$

(Pentadiagonal system)

*Diffusion
Further considerations

Splitting methods (Alternating Directions Implicit Method, ADI):

Sweep along 1 direction

$$\frac{C_{i,j}^{n+\frac{1}{2}} - C_{i,j}^{n}}{\left(\frac{\Delta t}{2}\right)} = D_{ij}\left[\frac{C_{i+1,j}^{n+\frac{1}{2}} - 2C_{i,j}^{n+\frac{1}{2}} + C_{i-1,j}^{n+\frac{1}{2}}}{\Delta x^2} + \frac{C_{i,j+1}^{n} - 2C_{i,j}^{n} + C_{i,j-1}^{n}}{\Delta y^2}\right]$$

Sweep along the other direction

$$\frac{C_{i,j}^{n+1} - C_{i,j}^{n+\frac{1}{2}}}{\left(\frac{\Delta t}{2}\right)} = D_{i,j}\left[\frac{C_{i+1,j}^{n+\frac{1}{2}} - 2C_{i,j}^{n+\frac{1}{2}} + C_{i-1,j}^{n+\frac{1}{2}}}{\Delta x^2} + \frac{C_{i,j+1}^{n+1} - 2C_{i,j}^{n+1} + C_{i,j-1}^{n+1}}{\Delta x^2}\right]$$

(Two tridiag. systems)

*Diffusion
Further considerations

Iterative methods. Assume a generic 2D diffusion equation (Poisson eq.)

$$\boxed{\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2} = f(x,y)}$$

After discretization:

$$C_{i+1,j} - 2C_{i,j} + 2C_{i-1,j} + \underbrace{\left(\frac{\Delta x^2}{\Delta y^2}\right)}_{\beta^2}\left[C_{i,j-1} - 2C_{i,j} + C_{i,j+1}\right] = f$$

$$\underbrace{\begin{bmatrix} 2 & 1 & 0 & 0 & \beta^2 & 0 \\ 1 & \ddots & \ddots & & & 0 \\ 0 & \ddots & \ddots & \ddots & & \\ 0 & & \ddots & \ddots & \ddots & \\ \beta^2 & & & \ddots & \ddots & \ddots \\ 0 & 0 & & & \ddots & \ddots \end{bmatrix}}_{\text{Pentadiagonal}} \begin{bmatrix} \\ C \\ \\ \end{bmatrix} = \begin{bmatrix} \\ D \\ \\ \end{bmatrix}$$

*Diffusion
Further considerations

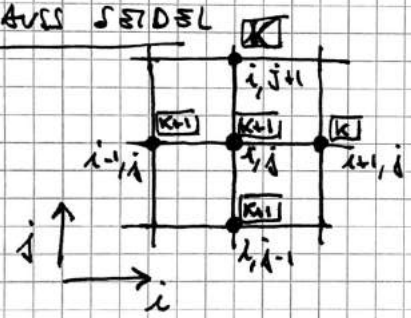
Iterative solvers

• JACOBI

$$-2C_{i,j} - 2\beta^2 C_{i,j} = -C_{i+1,j} - C_{i-1,j} - \beta^2 C_{i,j-1} - \beta^2 C_{i,j+1}$$

$$\boxed{C_{i,j}^{k+1} = \frac{C_{i+1,j}^{k} + C_{i-1,j}^{k} + \beta^2 C_{i,j-1}^{k} + \beta^2 C_{i,j-1}^{k}}{2(1+\beta^2)}}$$

• POINT GAUSS SEIDEL



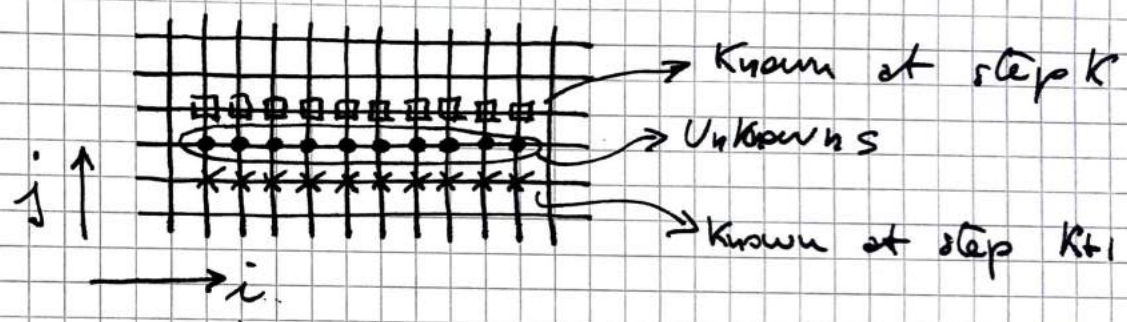
[Assume that the algorithms loops along i and then along j

$$\boxed{C_{i,j}^{k+1} = \frac{1}{2(1+\beta^2)}\left[C_{i+1,j}^{k} + C_{i-1,j}^{k+1} + \beta^2\left(C_{i,j+1}^{k} + C_{i,j-1}^{k+1}\right)\right]}$$

*Diffusion
Further considerations

• **LINE GAUSS-SEIDEL**

Instead of focusing on a single point, we focus on a line:



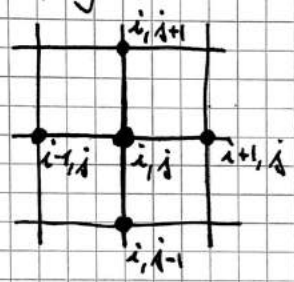
Therefore, we can write a tridiagonal system

$$\boxed{C^{k+1}_{i-1,j} - 2(1+\beta^2)C^{k+1}_{i,j} + C^{k+1}_{i+1,j} = -\beta^2 C^{k}_{i,j+1} - \beta C^{k+1}_{i,j-1}}$$

*Diffusion
Further considerations

• POINT SUCCESSIVE OVER RELAXATION METHODS (PSOR)
(Extrapolation to accelerate convergence)

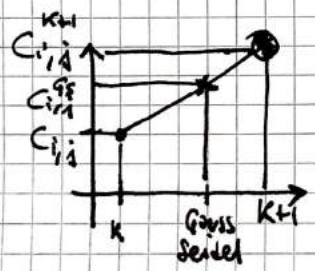
↳ Assume you apply a GAUSS-SEIDEL

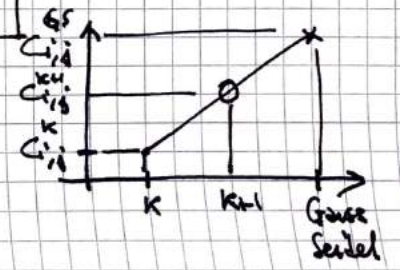


$$C^{calc}_{i,j} = \frac{1}{2(1+\beta^2)}\left[C^{k}_{i+1,j} + C^{k+1}_{i-1,j} + \beta^2\left(C^{k}_{i,j+1} + C^{k+1}_{i,j-1}\right)\right]$$

↳ Relaxation:

$$\boxed{C^{k+1}_{i,j} = C^{calc}_{i,j}\cdot\omega + (1-\omega)\,C^{k}_{i,j}}$$





Recall: Diffusion and advection

Let's consider our transport equation (1D ADE)

$$\boxed{\frac{\partial C^*}{\partial t^*} + u^* \frac{\partial C^*}{\partial x^*} = D \frac{\partial^2 C^*}{\partial x^{*2}}}$$

In dimensionless form:

$$\begin{cases} C = \dfrac{C^*}{C_w} \\ x = \dfrac{x^*}{L} \\ u = \dfrac{u^*}{u_{ref}} \\ t = \dfrac{t^* D}{L^2} \end{cases} \rightarrow \frac{D C_w}{L^2} \frac{\partial C}{\partial t} + \frac{u_{ref}\, u\, C_w}{L} \frac{\partial C}{\partial x} = \frac{D C_w}{L^2} \frac{\partial^2 C}{\partial x^2}$$

$$\frac{\partial C}{\partial t} + \frac{u_{ref} \cdot \cancel{C_w}}{\cancel{L}} \frac{L^{\cancel{2}}}{D \cancel{C_w}} u \frac{\partial C}{\partial x} = \frac{\partial^2 C}{\partial x^2}$$

$$\frac{\partial C}{\partial t} + \underbrace{\frac{u_{ref} \cdot L}{D}}_{Pe\ (\text{Peclet})} u \frac{\partial C}{\partial x} = \frac{\partial^2 C}{\partial x^2}$$

$$\left[Pe = \frac{\text{diffusion time}}{\text{convection time}} = \frac{L^2/D}{L/u_{ref}} = \frac{L \cdot u_{ref}}{D} \right]$$

$$\boxed{\frac{\partial C}{\partial t} + Pe \cdot u \frac{\partial C}{\partial x} = \frac{\partial^2 C}{\partial x^2}}$$

$Pe \ll 1 \Rightarrow$ Diffusion dominates $\rightarrow \boxed{\frac{\partial C}{\partial t} = \frac{\partial^2 C}{\partial x^2}}$

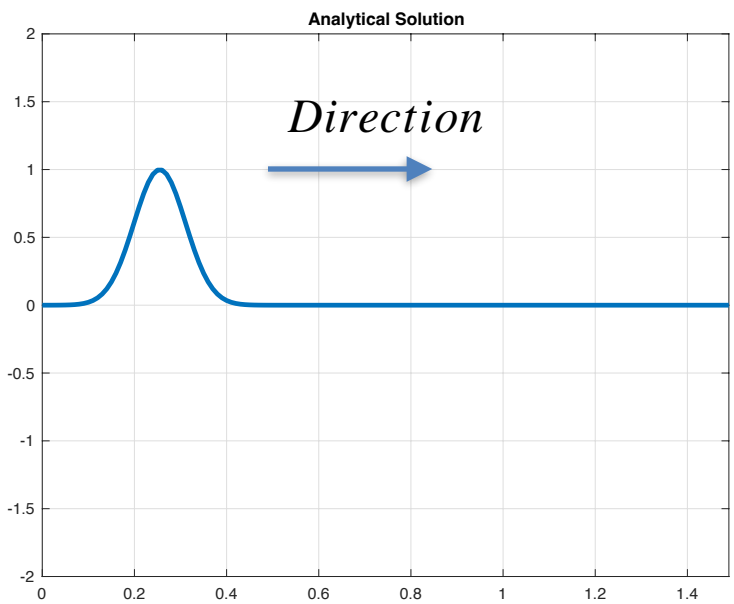
$Pe \gg 1 \Rightarrow$ Advection dominates $\rightarrow \boxed{\frac{\partial C}{\partial t} + Pe \cdot u \frac{\partial C}{\partial x} = 0}$

Advection: Introduction

$$\frac{\partial C}{\partial t} + U\frac{\partial C}{\partial x} = 0$$

Analytical solution exists

$$\begin{cases} C(x,t) = C_0(x - Ut) \\ C(x,0) = C_0(x) \end{cases}$$



Advection: Discretization strategy

In this case we do not use the usual "central" approximation:

$$\left[\frac{\partial C}{\partial x}\right]_i^n = \frac{C_{i+1}^n - C_{i-1}^n}{2\Delta x}$$

It can be proven that this scheme is unstable

We use what is usually called an "upwind" approximation:

$[U>0]$

$$\frac{C_i^{n+1} - C_i^n}{\Delta t} + U\,\frac{C_i^n - C_{i-1}^n}{\Delta x} = 0$$

$[U = Pe \cdot u]$

This gives:

$$C_i^{n+1} = C_i^n - \underbrace{\frac{U\Delta t}{\Delta x}}_{C_0\ (\text{Courant})}\left(C_i^n - C_{i-1}^n\right) \rightarrow \boxed{C_i^{n+1} = C_i^n(1-C_0) + C_0 \cdot C_{i-1}^n}$$

Stability: It can be shown that (Von-Neumann*)

$$G = 1 - C_0(1-\cos\theta) - i\,C_0\sin\theta \rightarrow |G| = 1 + \overbrace{(1-\cos\theta)}^{\geq 0}(C_0 - 1)\overbrace{2C_0}^{\geq 0} \rightarrow \boxed{C_0 \leq 1} \quad \left(\text{small } \Delta t\right)$$

Advection: Accuracy

What about the accuracy (Truncation error)

(Taylor series)

$$\begin{cases} C_i^{n+1} = C_i^n + \Delta t \left[\frac{\partial C}{\partial t}\right]_i^n + \frac{\Delta t^2}{2}\left[\frac{\partial^2 C}{\partial t^2}\right]_i^n + HOT & \text{Time} \\ C_{i-1}^n = C_i^n - \Delta x \left[\frac{\partial C}{\partial x}\right]_i^n + \frac{\Delta x^2}{2}\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n + HOT & \text{Space} \end{cases}$$

Therefore:

$$\boxed{C_i^{n+1} = (1-C_0)C_i^n + C_0 C_{i-1}^n}$$

$$C_i^n + \Delta t \left[\frac{\partial C}{\partial t}\right]_i^n + \frac{\Delta t^2}{2}\left[\frac{\partial^2 C}{\partial t^2}\right]_i^n = (1-C_0)C_i^n + C_0\left[C_i^n - \Delta x\left[\frac{\partial C}{\partial x}\right]_i^n + \frac{\Delta x^2}{2}\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n\right]$$

We get:

$$\cancel{C_i^n} + \Delta t \left[\frac{\partial C}{\partial t}\right]_i^n + \frac{\Delta t^2}{2}\left[\frac{\partial^2 C}{\partial t^2}\right]_i^n = \cancel{C_i^n} - \cancel{C_0 C_i^n} + \cancel{C_0 C_i^n} - C_0 \Delta x \left[\frac{\partial C}{\partial x}\right]_i^n + C_0 \frac{\Delta x^2}{2}\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n$$

$$\left[\frac{\partial C}{\partial t}\right]_i^n + \frac{\Delta t}{2}\left[\frac{\partial^2 C}{\partial t^2}\right]_i^n = -\frac{C_0 \Delta x}{\Delta t}\left[\frac{\partial C}{\partial x}\right]_i^n + C_0 \frac{\Delta x^2}{2\Delta t}\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n$$

Advection: Accuracy

Finally

$$\underbrace{\left[\frac{\partial C}{\partial t}\right]_i^n + \overbrace{\frac{C_0 \Delta x}{\Delta t}}^{U} \left[\frac{\partial C}{\partial x}\right]_i^n}_{\text{Original equation}} = \underbrace{-\frac{\Delta t}{2}\left[\frac{\partial^2 C}{\partial t^2}\right]_i^n + C_0 \frac{\Delta x^2}{2\Delta t}\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n}_{\boxed{\text{Error } (\varepsilon)}}$$

We focus now on the error:

$$\varepsilon = -\frac{\Delta t}{2}\left[\frac{\partial^2 C}{\partial t^2}\right]_i^n + \frac{U \Delta x}{2}\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n$$

$$\hookrightarrow \boxed{\frac{\partial C}{\partial t} = -U\frac{\partial C}{\partial x} \Rightarrow \frac{\partial}{\partial t}\left(\frac{\partial C}{\partial t}\right) = -U\frac{\partial}{\partial x}\left(-U\frac{\partial C}{\partial x}\right) \Rightarrow \boxed{\left[\frac{\partial^2 C}{\partial t^2} = U^2 \frac{\partial^2 C}{\partial x^2}\right]}}$$

$$\varepsilon = -\frac{\Delta t}{2}\left[\frac{\partial^2 C}{\partial t^2}\right]_i^n + \frac{U\Delta x}{2}\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n$$

$$= -\frac{\Delta t}{2}\left(U^2\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n\right) + \frac{U\Delta x}{2}\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n = \frac{U\Delta x}{2}\left\{\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n - \frac{U\Delta t}{\Delta x}\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n\right\}$$

$$= \frac{U\Delta x}{2}\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n (1 - C_0)$$

Advection: Accuracy, numerical diffusion

We have obtained the following expression

$$\boxed{\varepsilon = \underbrace{\frac{U\Delta x}{2}(1-C_0)}_{\alpha' = \text{Numerical viscosity or diffusivity}} \left[\frac{\partial^2 C}{\partial x^2}\right]_i^n}$$

→ We have already seen that $\left[\frac{\partial^2 C}{\partial x^2}\right]$ characterizes diffusion processes

Therefore, the equation we are going to solve is:

$$\boxed{\left[\frac{\partial C}{\partial t}\right]_i^n + U\left[\frac{\partial C}{\partial x}\right]_i^n = \alpha' \left[\frac{\partial^2 C}{\partial x^2}\right]_i^n}$$

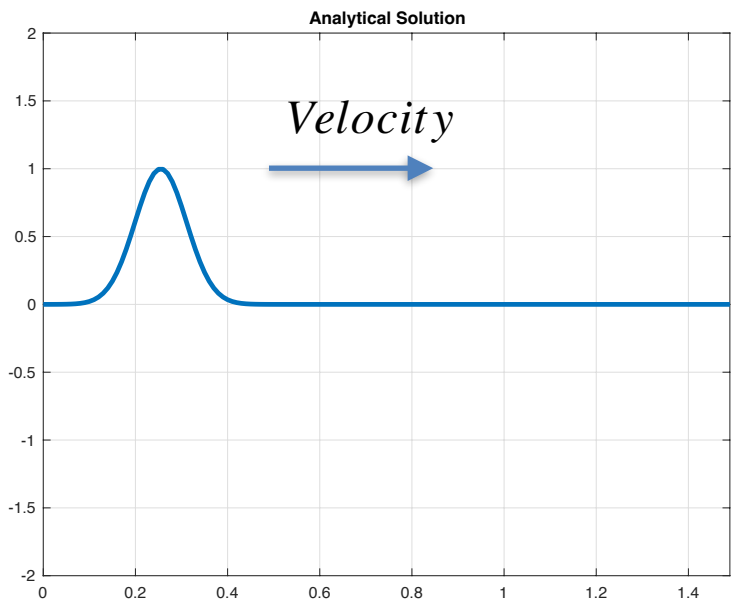
With the additional constraint that $C_0 \leq 1$, obtained from stability considerations (but also expected...)

Advection
Numerical Results

Discretized equation

$$C_i^{n+1} = (1 - Co)\, C_i^n + Co C_{i-1}^n$$

We first see what happens when $Co = 1.1$



As expected…

Advection
Numerical Results

Now we reduce the Courant number

$Co = 0.5$

$Co = 0.1$

Advection
Numerical Results

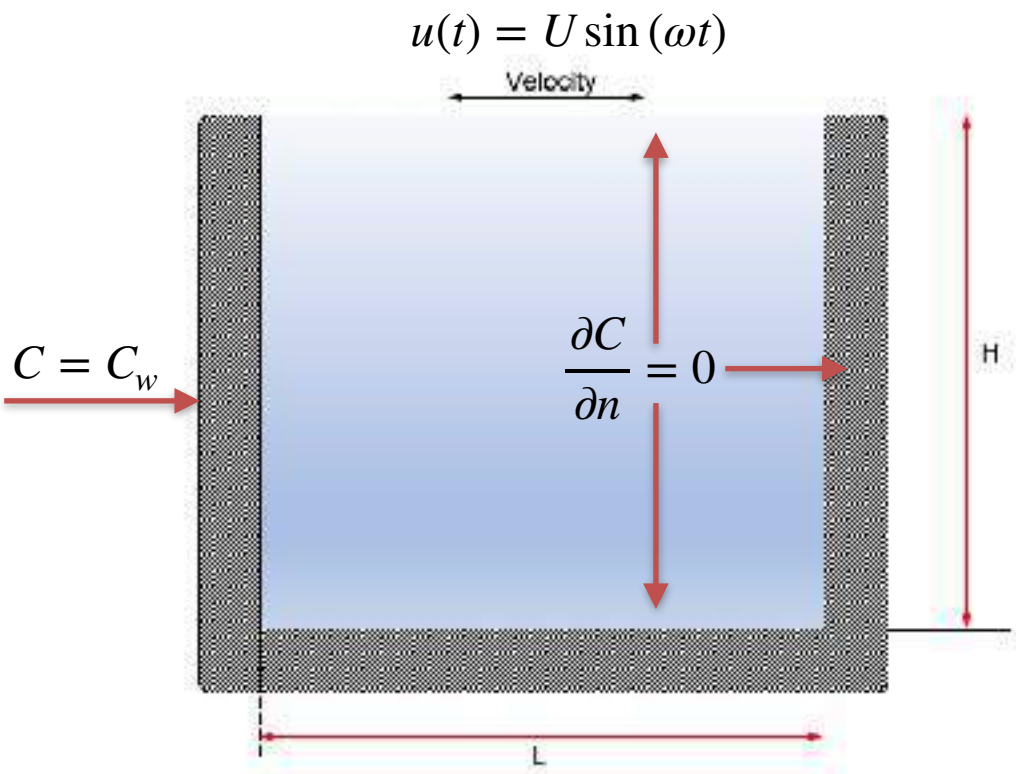
A heavier simulation does not necessarily mean a better simulation…

Looking at the truncation error: $$\epsilon = \frac{U\Delta x}{2}(1 - Co)\left[\frac{\partial^2 C}{\partial x^2}\right]$$

$Co = 1$

Now we have seen the fundamental concepts about diffusion & advection, and how to discretize them. It's now time to combine them and to turn into a more complex problem….

The full problem: Physics



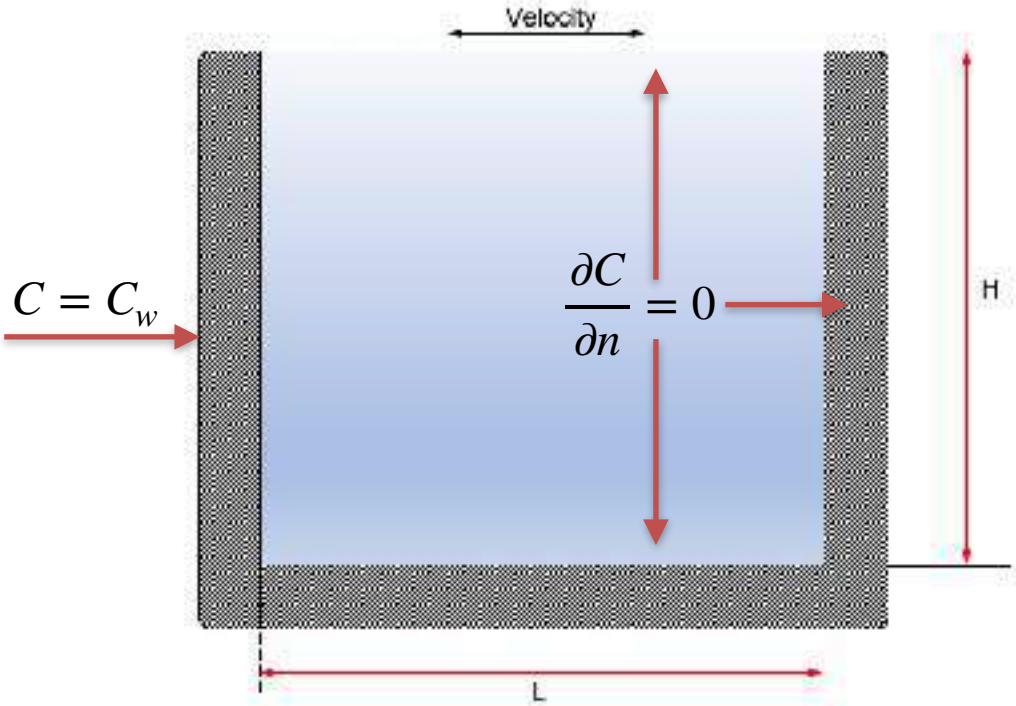
Governing equations

$$\begin{cases} \dfrac{\partial u_i}{\partial x_i} = 0 \\ \dfrac{\partial u_i}{\partial t} + u_j \dfrac{\partial u_i}{\partial x_j} = -\dfrac{\partial p}{\partial x_i} + \dfrac{1}{Re}\dfrac{\partial^2 u_i}{\partial x_j^2} \\ \dfrac{\partial C}{\partial t} + u_j \dfrac{\partial C}{\partial x_j} = \dfrac{1}{RePr}\dfrac{\partial^2 C}{\partial x_j^2} \end{cases}$$

$$\boxed{\frac{1}{Pe}}$$

The full problem: Physics

$$u(t) = U \sin(\omega t)$$



Note: transport equation for ω (*see later)

Note: BC for ω obtained from Taylor series expansion of ψ (**see later)

Alternative formulation (ψ, ω)

$$\begin{cases} u_x = \dfrac{\partial \psi}{\partial y} & \text{Riemann} \\ u_y = -\dfrac{\partial \psi}{\partial x} & \\ \omega = \dfrac{\partial u_y}{\partial x} - \dfrac{\partial u_x}{\partial y} & \text{Vorticity} \end{cases}$$

$$\begin{cases} \omega = -\left(\dfrac{\partial^2 \psi}{\partial x^2} + \dfrac{\partial^2 \psi}{\partial y^2}\right) \\ \dfrac{\partial \omega}{\partial t} + u_j \dfrac{\partial \omega}{\partial x_j} = +\dfrac{1}{Re}\dfrac{\partial^2 \omega}{\partial x_j^2} & \text{*Curl(NS)} \\ \dfrac{\partial C}{\partial t} + u_j \dfrac{\partial C}{\partial x_j} = \dfrac{1}{RePr}\dfrac{\partial^2 C}{\partial x_j^2} \end{cases}$$

The full problem: *A note on the vorticity transport equation

(Note: in this slide, ***u=velocity along x***; ***v=velocity along y***; ***subscripts indicate partial derivatives***). To obtain the Vorticity transport eq.:

$$u_t + uu_x + vu_y = -P_x + \mu \nabla^2 u \qquad (A)$$

$$v_t + uv_x + vv_y = -P_y + \mu \nabla^2 v \qquad (B)$$

Then, $\partial/\partial y(A)$ and $\partial/\partial x(B)$

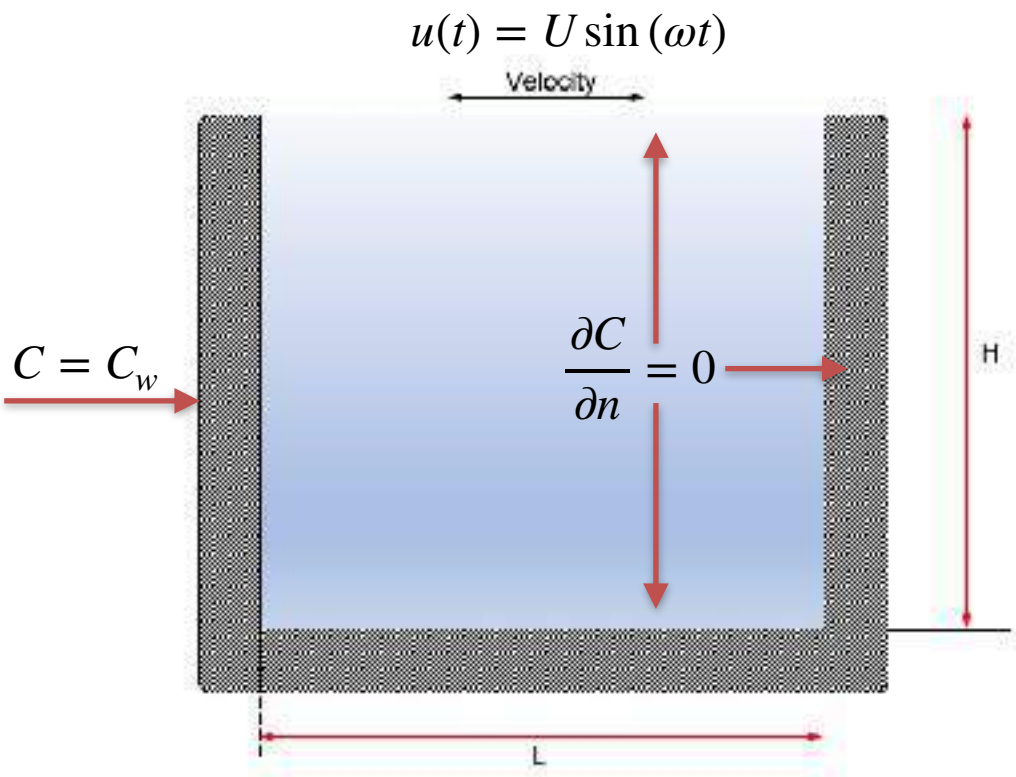
$$u_{ty} + u_y u_x + uu_{xy} + v_y u_y + vu_{yy} = -P_{xy} + \mu \nabla^2 u_y \qquad (C)$$

$$v_{tx} + u_x v_x + uv_{xx} + v_x v_y + vv_{yx} = -P_{xy} + \mu \nabla^2 v_x \qquad (D)$$

Making (D)-(C), and taking into account that:

$$\left.\begin{aligned} &\omega = v_y - u_x \\ &u_x + v_y = 0 \quad (continuity) \end{aligned}\right\} \longrightarrow \omega_t + u\omega_x + v\omega_y = Re^{-1}\nabla^2\omega$$

The full problem: Physics



Central approx.

$$\left[\frac{\partial f_{i,j}}{\partial x}\right]^n_{i,j} = \frac{f^n_{i+1,j} - f^n_{i-1,j}}{2\Delta x}$$

$$\left[\frac{\partial^2 f}{\partial x^2}\right]^n_{i,j} = \frac{f^n_{i-1,j} - 2f^n_{i,j} + f^n_{i+1,j}}{\Delta x^2}$$

$$\left[\frac{\partial f_{i,j}}{\partial y}\right]^n_{i,j} = \frac{f^n_{i,j+1} - f^n_{i,j-1}}{2\Delta y}$$

$$\left[\frac{\partial^2 f}{\partial y^2}\right]^n_{i,j} = \frac{f^n_{i,j-1} - 2f^n_{i,j} + f^n_{i,j+1}}{\Delta y^2}$$

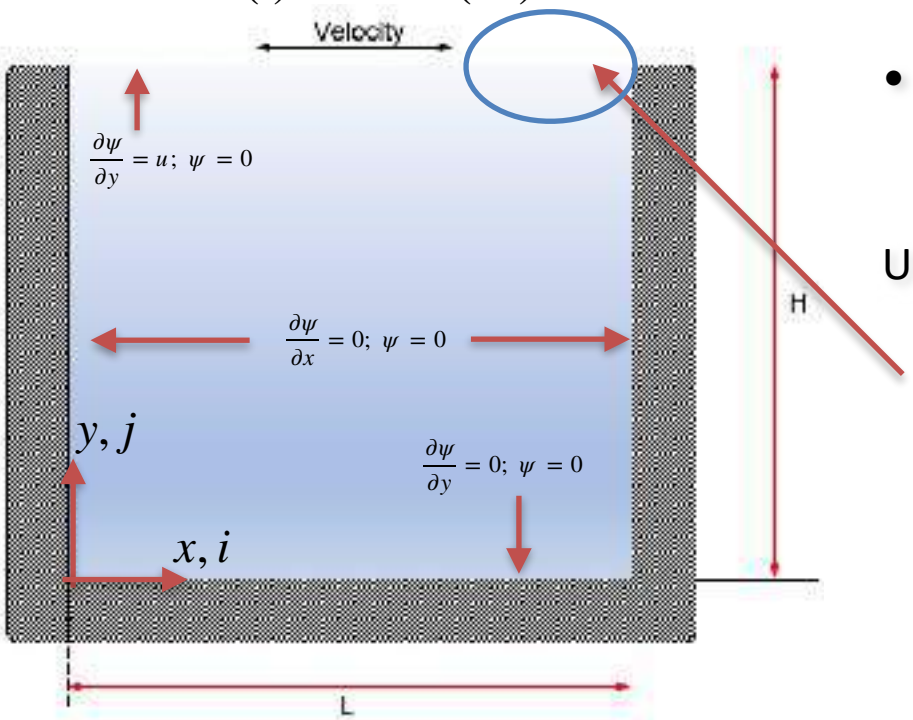
Explicit

$$\left[\frac{\partial f}{\partial t}\right]^n_{i,j} = F^n_{i,j}$$

The full problem: **A note on B.C.

Let's focus on B.C. for ψ, ω:

$$u(t) = U \sin(\omega t)$$



- B.C. $\psi = 0$ is used to solve the Poisson equation
- B.C on the derivative of ψ is used to build B.C. for ω.

Using Taylor series expansion for ψ we have:

$$\psi_{i,j-1} = \psi_{i,j} - \Delta y \left[\frac{\partial \psi}{\partial y}\right]_{i,j} + \frac{\Delta y^2}{2}\left[\frac{\partial^2 \psi}{\partial y^2}\right]_{i,j} + \dots$$

$$\omega_{i,j} = \left[\frac{\partial^2 \psi}{\partial y^2}\right]_{i,j}$$

$$\psi_{i,j} = 0; \quad \left[\frac{\partial \psi}{\partial y}\right]_{i,j} = g$$

See *Fletcher, C.A.J,* ***Computational techniques for fluid dynamics — 2 volume*** (1991), Springer-Verlag Berlin Heidelberg

The full problem: Numerics

```
c       WRITE(*,*) 'CALL POISSON'
        call poisson(dx,dy,vor2,psi,psin)

        DO I=1,NX
         DO J=1,NY
          PSI(I,J)=PSIN(I,J)
         ENDDO
        ENDDO


c       WRITE(*,*) 'CALL BC'
        call boundary_vor(dx,dy,psi,uwall,vor2)

        DO I=1,NX
         DO J=1,NY
          VOR(I,J)=VOR2(I,J)
         ENDDO
        ENDDO


c       WRITE(*,*) 'CALL VELOCITY'
        call velocity(dx,dy,psi,u,v)

c       WRITE(*,*) 'CALL TSCHEME'
        call tscheme(re,dx,dy,dt,u,v,vor,vor2)
```

Find ψ^{n+1} from ω^n $\longrightarrow$ $\omega = -\left(\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2}\right)$

Apply BC for ω^{n+1} from the knowledge of ψ^{n+1}

$$\omega_{1,j} = \frac{1}{2\Delta x^2}\left(8\psi_{2,j} - \psi_{3,j}\right)$$

$$\omega_{N,j} = \frac{1}{2\Delta x^2}\left(8\psi_{N-1,j} - \psi_{N-2,j}\right)$$

$$\omega_{i,1} = \frac{1}{2\Delta y^2}\left(8\psi_{i,2} - \psi_{i,3}\right)$$

$$\omega_{i,M} = \frac{1}{2\Delta x^2}\left(8\psi_{i,M-1} - \psi_{i,M-2}\right) + \frac{3u}{\Delta y}$$

Solve the transport of vorticity $\omega^{n+1} = \omega^n + \Delta t\left(F^n_{i,j}\right)$

The full problem: Numerics

$$Re = 2000; \quad L = H = 1; \quad N_x = N_y = 100; \quad \Delta t = 10^{-3}$$

$$u(t) = U \sin\left(\frac{t}{3}\right)$$

$$u(t) = U \sin\left(\frac{t}{2}\right)$$

$$u(t) = U \sin\left(\frac{3t}{4}\right)$$

Conclusions

1. Advection-Diffusion Equation

2. Discretization strategies: Basic concepts (1st, 2nd derivatives, Taylor series)

3. Diffusion equation: Explicit and implicit methods; stability and accuracy

4. Linear advection equation: “numerical diffusion" and stability

5. Transport of a scalar in a time dependent field: the full problem and its implementation

Navier-Stokes Eqs. (primitive variables)

$$\begin{cases} \dfrac{\partial u_i}{\partial x_i} = 0 \\ \dfrac{\partial u_i}{\partial t} + u_j \dfrac{\partial u_i}{\partial x_j} = -\dfrac{\partial p}{\partial x_i} + \dfrac{1}{Re}\dfrac{\partial^2 u_i}{\partial x_j{}^2} \end{cases}$$

$$Re = \frac{U_T L}{\nu}, \quad \nu = \frac{\mu}{\rho}$$

ν kinematic viscosity;

ρ density

μ dynamics. viscosity

L Reference Length

U_T Reference Velocity

Streamfunction-Vorticity formulation (ψ, ω)

$$\begin{cases} u_x = \dfrac{\partial \psi}{\partial y} \\ u_y = -\dfrac{\partial \psi}{\partial x} \\ \omega = \dfrac{\partial u_y}{\partial x} - \dfrac{\partial u_x}{\partial y} \end{cases}$$

Riemann equations

Vorticity definition

$$\begin{cases} \omega = -\left(\dfrac{\partial^2 \psi}{\partial x^2} + \dfrac{\partial^2 \psi}{\partial y^2}\right) \\ \dfrac{\partial \omega}{\partial t} + u_j \dfrac{\partial \omega}{\partial x_j} = +\dfrac{1}{Re}\dfrac{\partial^2 \omega}{\partial x_j{}^2} \end{cases}$$

*Curl(NS)

TECHNISCHE UNIVERSITÄT WIEN

ILSB

NSSC-II

The Finite Elements

Some Basics in the Context of Solid Mechanics

Pavan Kumar, H. Pettermann

Institute of Lightweight Design and Structural Biomechanics,
TU Wien

Introduction

These lecture notes, and the accompanying classes, aims at a very short introduction into Finite Element Method. This will be done in the context of Solid Mechanics, highlighting the intricate material behaviour in linear elasticity based on 4th order tensors.
The subject will be presented from the point of view of mechanical engineer, but mathematical foundation will be visible.

Motivation

Where can you use FEM?

- Structural mechanics
- Heat Transfer analysis
- Electromagnetic analysis
- Vibrational analysis
- Acoustics
- Coupled analysis etc., (fatigue, creep, corrosion)

Anything that can be represented as system of Partial Differential Equation's can be solved using FEM.

Around 50+ commercial software of FEA are available.

Some examples

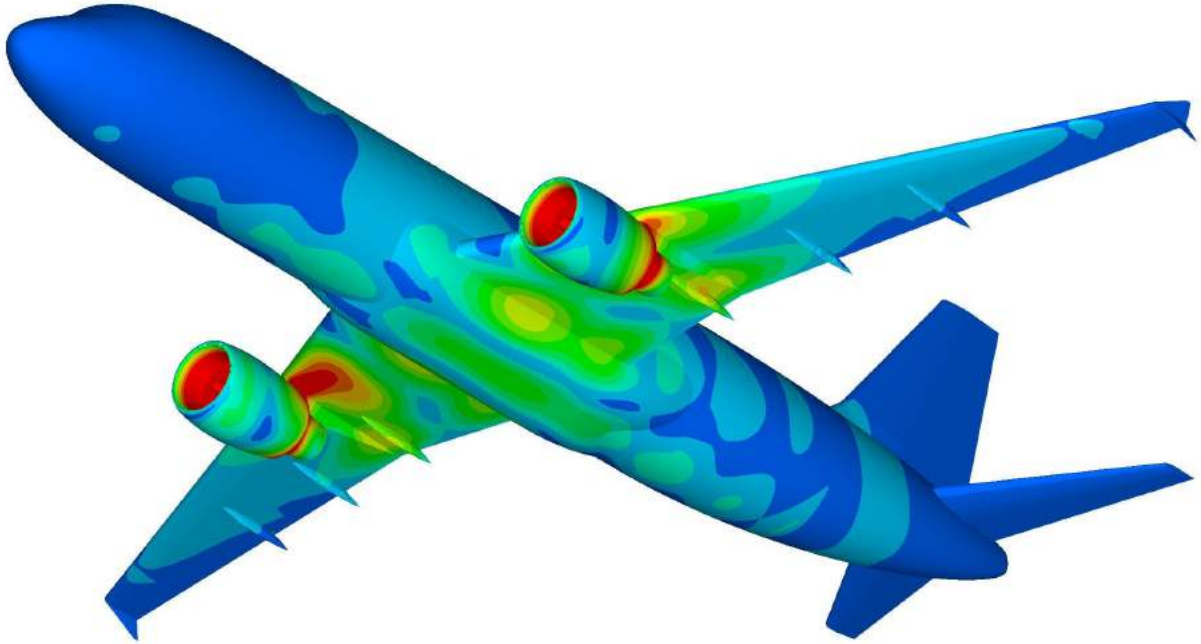


Fig: Stress analysis of Airbus A320

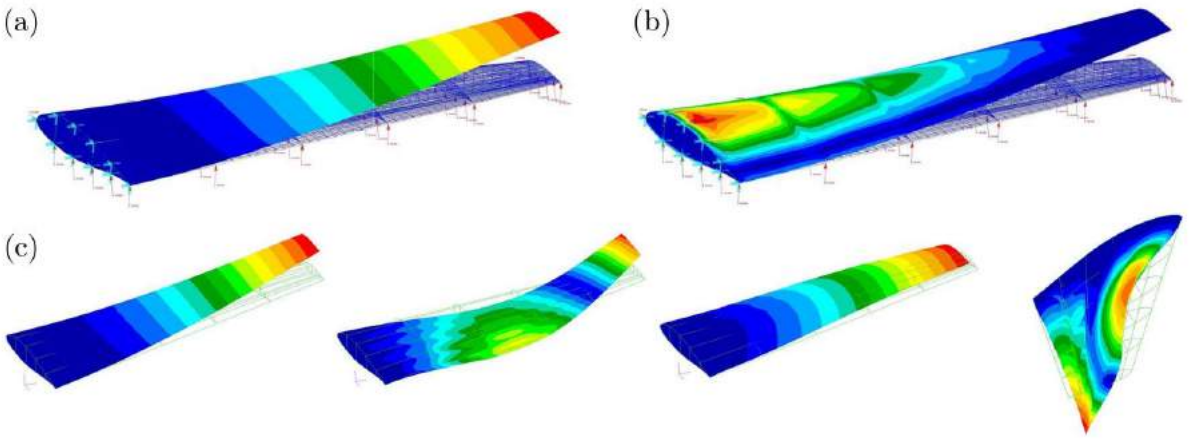


Fig: Aircraft wing structure, a)displacement, b)stress, c) modes [1]

[1]: Abdelkader B et al., DOI: 10.15632/jtam-pl.55.2.447

Some examples

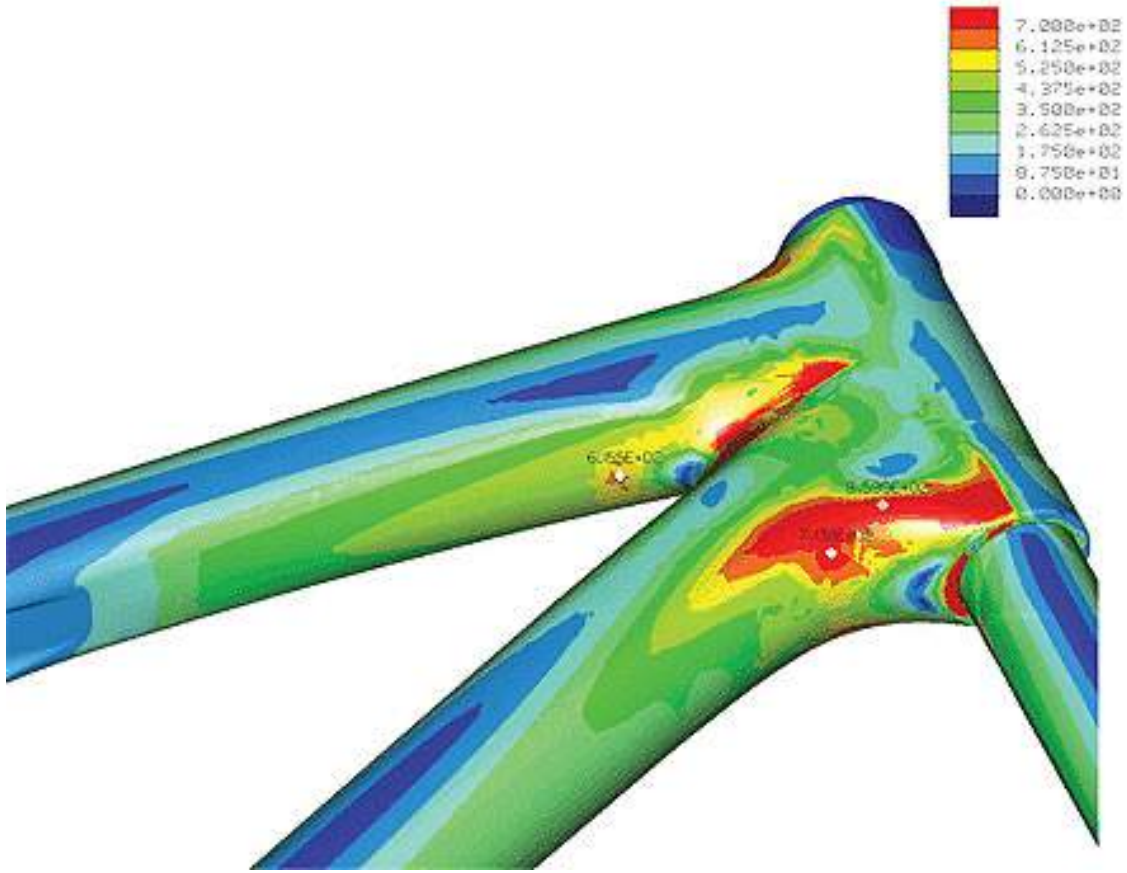


Fig: FEA Images of bicycle frame [1].

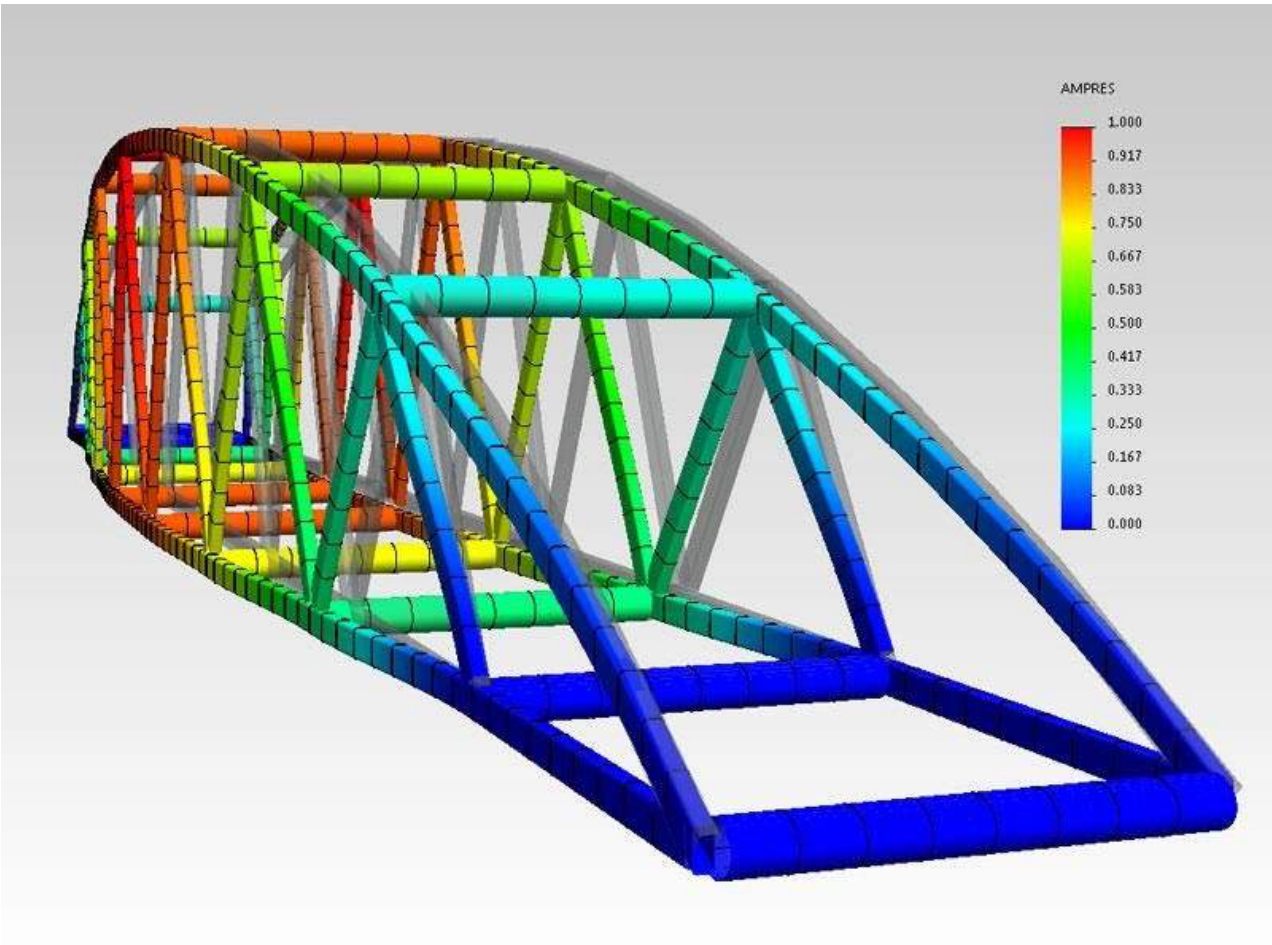


Fig: FEA Model of a sample bridge

[1]: https://www.designworldonline.com/mountain-bike-company-uses-fea-to-design-fast-bikes-faster/

Some examples

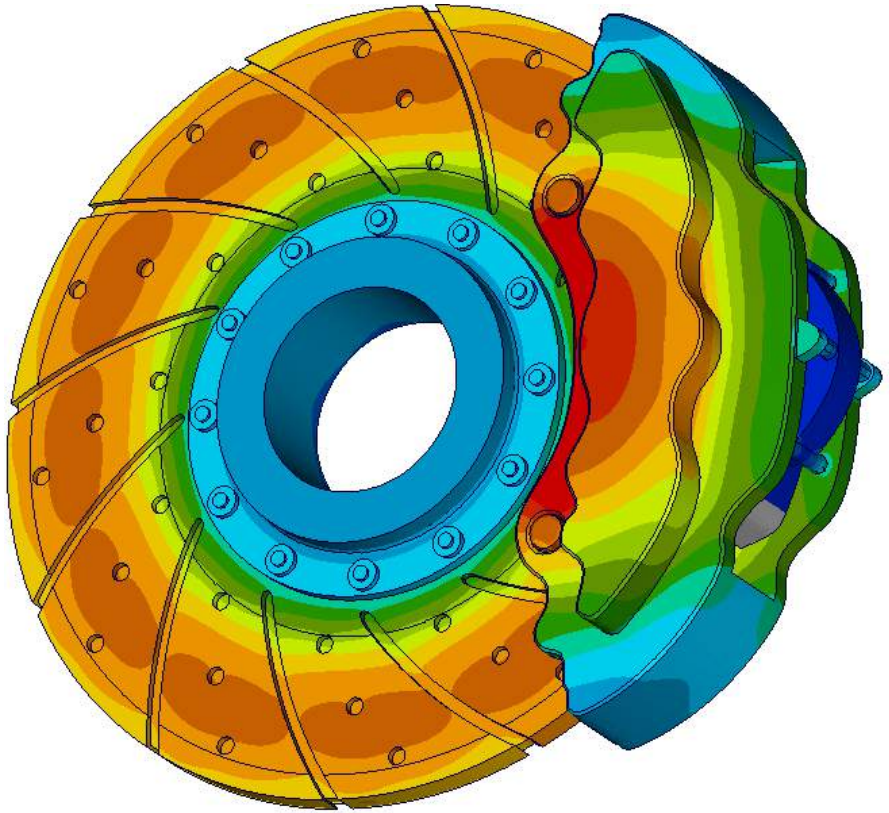


Fig: Heat transfer in a brake disk.

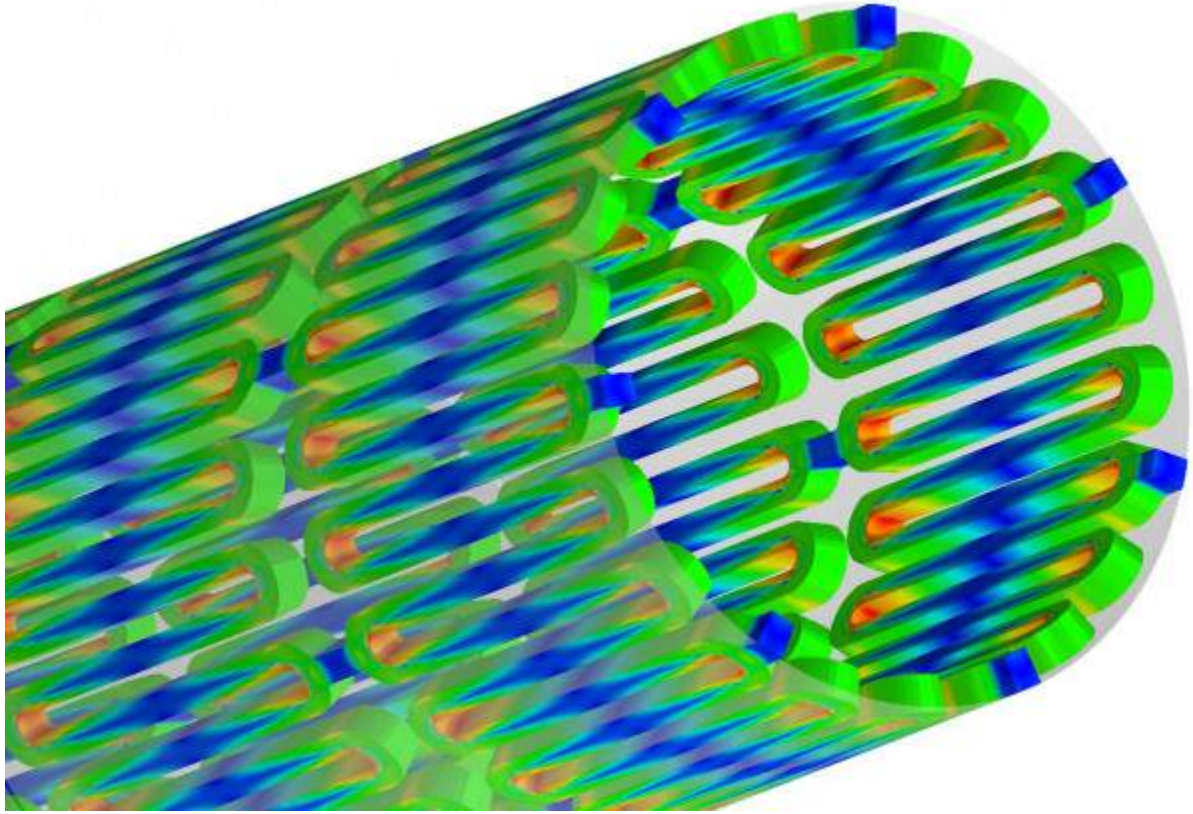


Fig: Heat Transfer in coils [1].

[1] Source: Craig Bonsignore/Flickr

Some examples

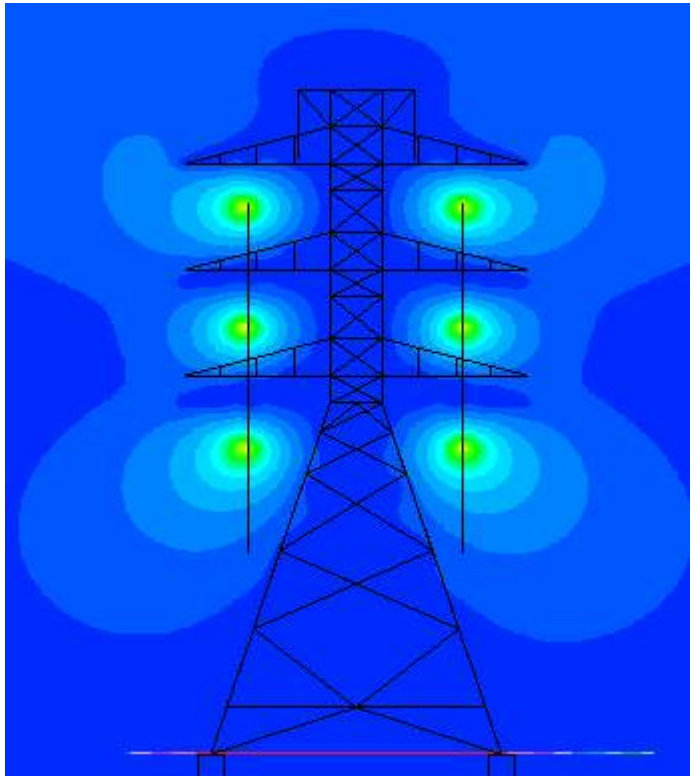
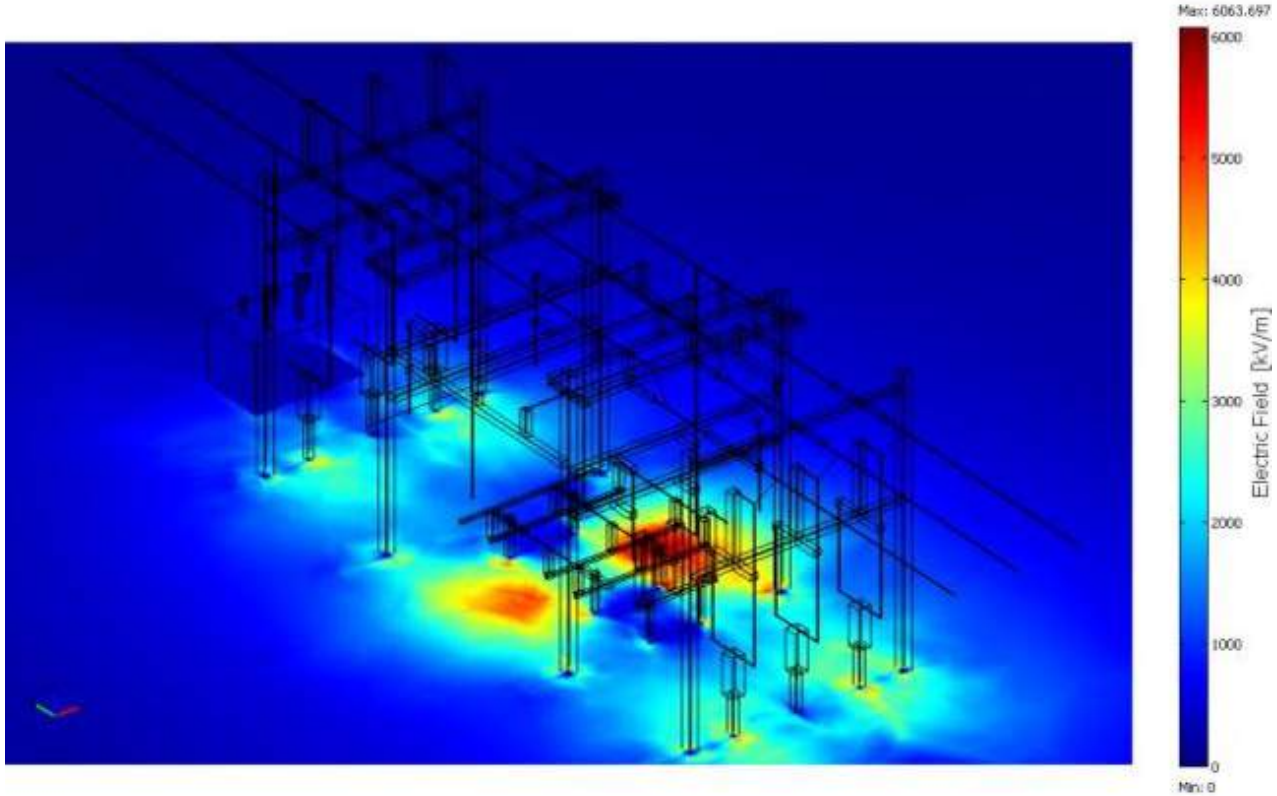


Fig: Electromagnetic interference [1].



(b)

Fig: Distribution of electric field in high voltage substation [2].

[1] https://greymattersglobal.com/finite-element-analysis/
[2] R. Barros et al., Electric field mapping in high voltage substation using the finite elements method

Some examples

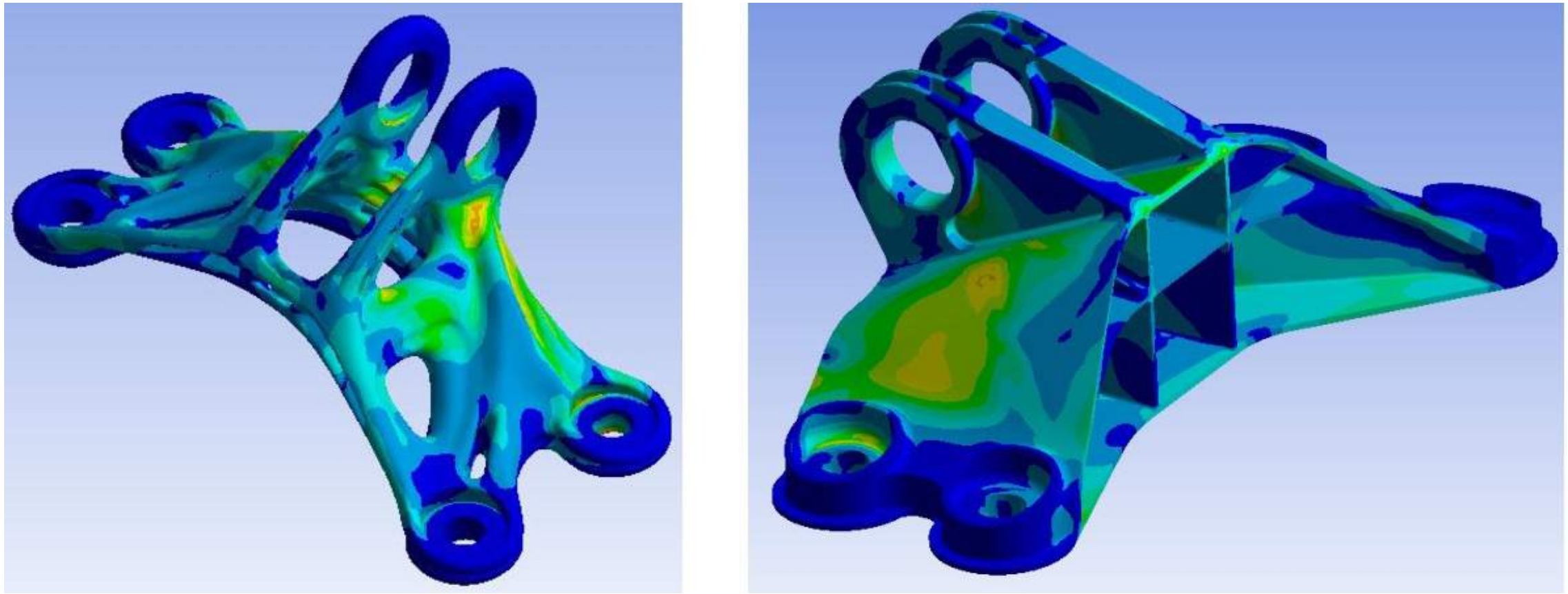


Fig: GE Aircraft engine bracket.

Some examples

Fig: Volumetric trusses optimum design [1].

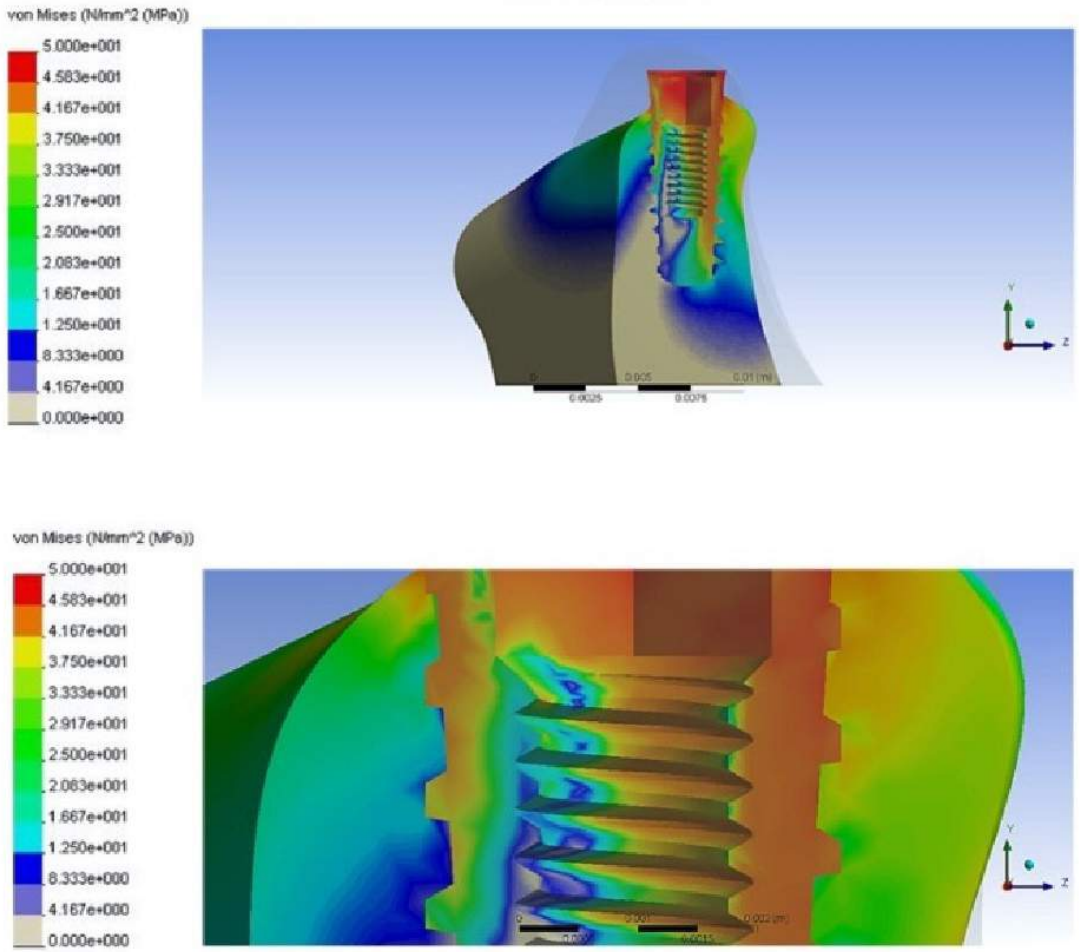
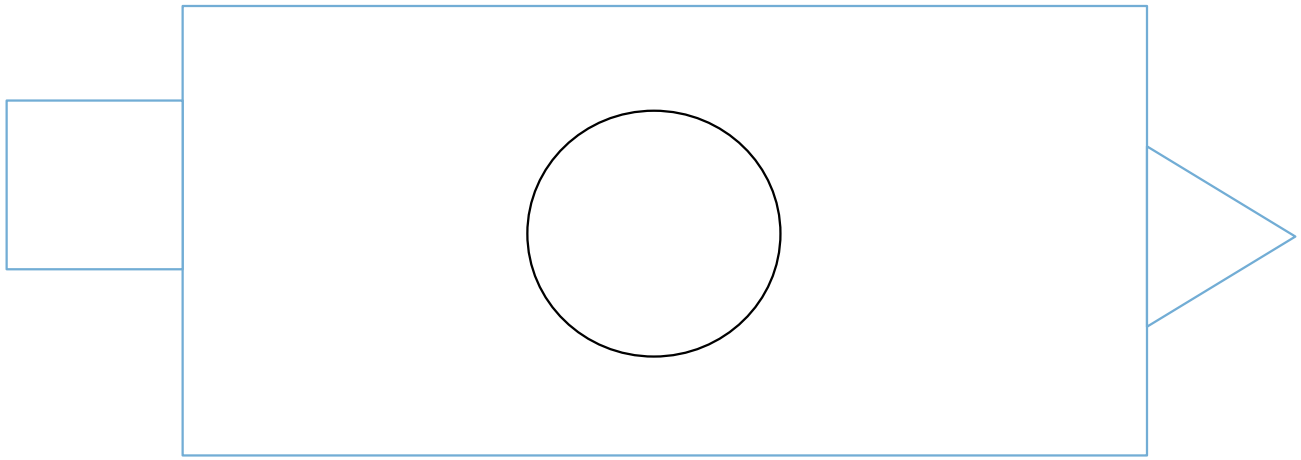


Fig: Stress pattern in cortical bone with implants [2].

[1] https://github.com/topics/topology-optimization
[2] doi:10.3390/prosthesis2030019

Simple Analogy

Find the Area of the following structure.



1. Recognize the area into different known shapes (Numerics).
2. Find Areas of individual shapes (Physics).
3. Assemble them together. (Numerics).

Continuum Solid Mechanics

The basics of Continuum solid mechanics will be presented in matrix notation which is advantageous for software implementation.

Balance Equation

The local balance equation (condition of equilibrium) in differential form reads

$$\nabla\sigma + \vec{\mathbf{q}}^T = 0.$$

In the index notation form reads,

$$\nabla\sigma_{ji} + q_i = 0,$$

$$\sigma_{ji,i} + q_i = 0,\ \forall i, j = 1, 2, 3(x, y, z).$$

(Einstein summation applies)

$\sigma =$ stress

$\sigma_{i,j}$ Are the components of the symmetric stress tensor

$\vec{\mathbf{q}}^T = (q_x, q_y, q_z)$

are the vector components of the volume force density

Balance Equation

TECHNISCHE UNIVERSITÄT WIEN

Fully expanded for *plane stress* state, the balance equation reads

$$\begin{aligned} \sigma_{zz} &= 0, & \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{yx}}{\partial y} &= -q_x, \\ \sigma_{zx} &= 0, & \frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} &= -q_y, \\ \sigma_{zy} &= 0, & \sigma_{xy} &= \sigma_{yx}. \end{aligned}$$

Applying Voigt--Nye matrix notation, a differential operator matrix can be defined as

$$\begin{pmatrix} \frac{\partial}{\partial x} & 0 & \frac{\partial}{\partial y} \\ 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{pmatrix} \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} = - \begin{pmatrix} q_x \\ q_y \end{pmatrix}$$

Balance Equation

TECHNISCHE UNIVERSITÄT WIEN

For general tri-axial stress states the balance equation in Voigt—Nye notation read,

$$\begin{pmatrix} \frac{\partial}{\partial x} & 0 & 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial z} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{pmatrix} \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{pmatrix} = - \begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix} .$$

In short,

$$\underset{\approx}{\boldsymbol{d}}^T \underset{\sim}{\boldsymbol{\sigma}} = - \vec{q}^{\,T} .$$

with the corresponding operator matrix $\underset{\approx}{\boldsymbol{d}}^T$. And the vector of the stress components $\underset{\sim}{\sigma}$ is a vector in the sense of linear algebra.

Displacement-Strain relation

TECHNISCHE UNIVERSITÄT WIEN

The linearized displacement---strain relation reads in index notation,

$$\varepsilon_{ij} = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right),$$

with $\vec{u}^{\mathrm{T}} = (u_1, u_2, u_3) = (u, v, w)$ being the local displacement vector.

For example,

$$\varepsilon_{xx} = \frac{\partial u}{\partial x} \quad , \quad \varepsilon_{xy} = \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) = \frac{\gamma_{xy}}{2}.$$

Applying Voigt--Nye notation, the tri-axial relation reads,

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

In short, $\underset{\approx}{\varepsilon} = \underset{\approx}{\boldsymbol{d}}\vec{u}$. Here, $\gamma_{xy}, \gamma_{yz}, \gamma_{zx}$ are called shear angles.

Constitutive Material Law

The relation between stresses and strains is given by the constitutive law. In linear elasticity it is called Hooke's Law, and reads in index notation,

$$\sigma_{ij} = E_{ijkl}\,\varepsilon_{kl}$$

Note, that the elasticity tensor, E_{ijkl} is a 4th order tensor. Applying Voigt--Nye matrix notation and assuming isotropic (i.e. Direction independent) behaviour, the constitutive law can be given as

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{pmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & \nu & & & \\ \nu & 1-\nu & \nu & & 0 & \\ \nu & \nu & 1-\nu & & & \\ & & & \frac{1-2\nu}{2} & 0 & 0 \\ & 0 & & & \frac{1-2\nu}{2} & 0 \\ & & & & & \frac{1-2\nu}{2} \end{pmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{pmatrix}$$

E -Young's modulus

ν -Poisson ratio

Stress-Strain Curve: https://www.geogebra.org/m/AMjGYhNm
Hook Law: https://www.geogebra.org/m/c5un7end

Constitutive Material Law

Two independent material properties can be measured by experimental testing (Young's Modulus and Poisson ratio). In short, the relation between stress and strain can be written as

$$\underset{\sim}{\boldsymbol{\sigma}} = \underset{\approx}{\mathbf{E}}\underset{\sim}{\boldsymbol{\varepsilon}}$$

with the elasticity matrix $\underset{\approx}{\mathbf{E}}$. The inverse relation can be written as,

$$\underset{\sim}{\boldsymbol{\varepsilon}} = \underset{\approx}{\mathbf{C}}\underset{\sim}{\boldsymbol{\sigma}}$$

with the Compliance Matrix $\underset{\approx}{\mathbf{C}} = \underset{\approx}{\mathbf{E}}^{-1}$.

Strain Energy and Elastic Potential

By use of the above conventions, the strain energy density can be expressed as,

$$\hat{U} = \frac{1}{2}\,\sigma_{ij}\,\varepsilon_{ij} = \frac{1}{2}\,\underset{\sim}{\boldsymbol{\sigma}}^{\mathrm{T}} : \underset{\sim}{\boldsymbol{\varepsilon}},$$

from which follows the strain energy

$$U = \int_V \hat{U}\mathrm{dV} = \int_V \frac{1}{2}\,\sigma_{ij}\,\varepsilon_{ij}\mathrm{dV} = \int_V \frac{1}{2}\,\underset{\sim}{\boldsymbol{\sigma}}^{\mathrm{T}} : \underset{\sim}{\boldsymbol{\varepsilon}}\mathrm{dV},$$

in a elastic body with volume V.

With the constitutive equation, $\underset{\sim}{\boldsymbol{\sigma}} = \underset{\approx}{\mathbf{E}}\underset{\sim}{\boldsymbol{\varepsilon}}$, the symmetry of the elasticity matrix, $\underset{\approx}{\mathbf{E}} = \underset{\approx}{\mathbf{E}}^T$,

$$U = \int_V \left(\frac{1}{2}\,\underset{\sim}{\boldsymbol{\varepsilon}}^{\mathrm{T}} : \underset{\approx}{\boldsymbol{E}} : \underset{\sim}{\boldsymbol{\varepsilon}}\right)\mathrm{dV},$$

is the strain energy stored by the body. Notice that we have integral equation now.

External Energy and Elastic potential

The total potential, V, of an elastic body is the sum of the

- potential of the internal forces, U, (the strain energy) and
- the potential of the external forces, W.

$$V = U + W$$

What could be external forces?

- Concentrated forces, $\overrightarrow{F_i}$
- Surface force, $\overrightarrow{q_O}$
- Volume forces, $\overrightarrow{q_v}$

$$W = -\sum_i \overrightarrow{F_i} \cdot \overrightarrow{u_i} - \int_A \overrightarrow{q_O} \cdot \overrightarrow{u_O}\,\mathrm{dA} - \int_V \overrightarrow{q_v} \cdot \overrightarrow{u}\,\mathrm{dV}$$

Total energy is now

$$\int_V \frac{1}{2}\,\sigma_{ij}\,\varepsilon_{ij}\mathrm{dV} + \sum_i \overrightarrow{F_i} \cdot \overrightarrow{u_i} + \int_A \overrightarrow{q_O} \cdot \overrightarrow{u_O}\,\mathrm{dA} + \int_V \overrightarrow{q_v} \cdot \overrightarrow{u}\,\mathrm{dV} = V$$

Now, I want to solve this, How?

The principle of virtual displacements

I add a small quantity $u = u + e\delta u$ which and see the limit when e goes to zero. This idea is called as principle of virtual displacement.

Then, our total potential can be written as

$$\delta W = \sum_i \overrightarrow{F_i} \cdot \delta \overrightarrow{u_i} + \int_O \overrightarrow{q_O} \cdot \delta \overrightarrow{u_O} \, \mathrm{dO} + \int_V \overrightarrow{q_v} \cdot \delta \overrightarrow{u} \, \mathrm{dV},$$

$$\delta U = \int_V \sigma_{ij} \, \delta \varepsilon_{ij} \mathrm{dV}.$$

For $\delta V = 0 \implies \delta U + \delta W = 0$ leads to

$$\int_V \sigma_{ij} \, \delta \varepsilon_{ij} \mathrm{dV} = \underset{\sim}{\boldsymbol{F}}^T \delta \underset{\sim}{\boldsymbol{U}}.$$

This is the principle of virtual displacement. In mathematical term, it is also called as weak forms.

The RITZ Approach (Direct methods)

For the displacement field $\boldsymbol{u}(x,y,z)$, i.e for their components $u(x,y,z)$, $v(x,y,z)$, $w(x,y,z)$, a RITZ ansatz is introduced of the form (for x component)

$$u(x,y,z) \rightarrow u^* = \sum_{j=1}^{N} a_{u,j}\,\varphi_{u,j} \overset{Einstein}{=} a_{u,j}\,\varphi_{u,j} = \underset{\sim}{\boldsymbol{\varphi}_{\boldsymbol{u}}^{\mathrm{T}}}\,\underset{\sim}{\boldsymbol{a}_{\boldsymbol{u}}}$$

and for the complete displacement vector

$$\overrightarrow{u^*} = \underset{\approx}{\boldsymbol{\varphi}}^T \underset{\sim}{\mathbf{a}}$$

which results in an *approximation* of the displacement field $\overrightarrow{u^*} = u(x,y,z)$

Here, $\varphi_j(x,y,z)$ are predefined *interpolation functions*. And a_j are the free coefficients.

The RITZ Approach

With such a Ritz' Ansatz an approximation of the potential V^* is obtained which is a *function* of the generalized coordinates.

$$V^* = V^*(a_1, a_2, \dots)$$

Applying the equilibrium condition

$$\delta V^* = 0 \Leftrightarrow \frac{\partial V^*}{\partial a_k}\,\delta a_k = 0$$

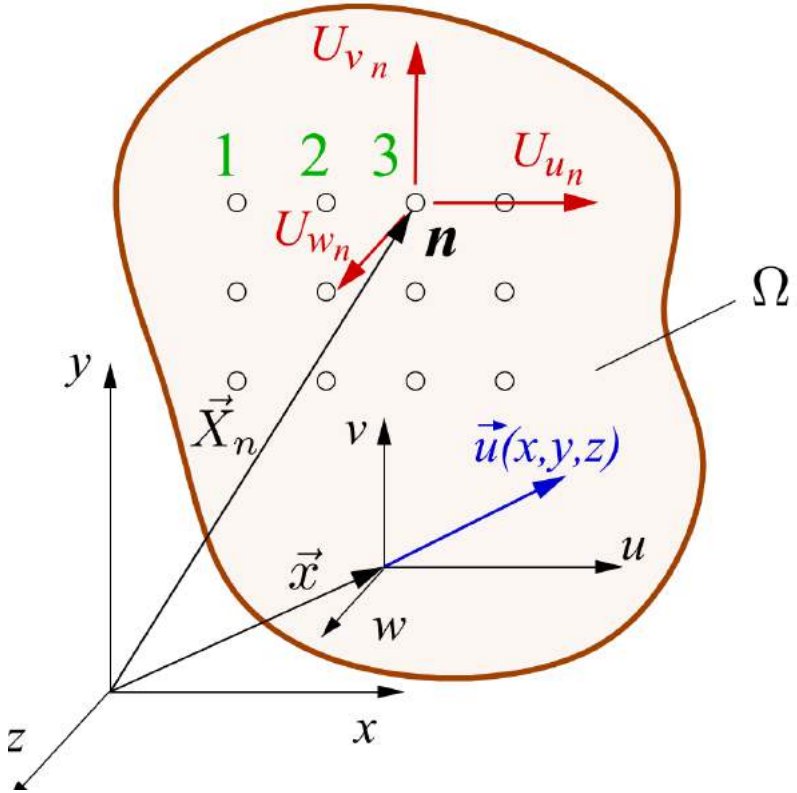
gives rise to

$$\frac{\partial V^*}{\partial a_1} = 0,\ \frac{\partial V^*}{\partial a_2} = 0 \dots \frac{\partial V^*}{\partial a_j} = 0 \quad \forall\, j$$

The FEM (as Ritz Method)

TECHNISCHE UNIVERSITÄT WIEN

We can apply this for linear, conservative, static problems. Before, the coefficients a_j did not carry any meaning, so, we can define properly a_j.



$$u = \sum_i U_{u_i} \varphi_{u_i}(x,y,z)\,; \quad v, w \text{ likewise;}$$

consequently

$$u(x,y,z) = \boldsymbol{\varphi}_{\underset{\sim}{u}}^{\mathbf{T}}(x,y,z)\,\boldsymbol{U}_{\underset{\sim}{u}}$$

$$v(x,y,z) = \boldsymbol{\varphi}_{\underset{\sim}{v}}^{\mathbf{T}}(x,y,z)\,\boldsymbol{U}_{\underset{\sim}{v}}$$

$$w(x,y,z) = \boldsymbol{\varphi}_{\underset{\sim}{w}}^{\mathbf{T}}(x,y,z)\,\boldsymbol{U}_{\underset{\sim}{w}}$$

in short

$$\overrightarrow{\mathbf{u}^*}(x,y,z) = \underset{\approx}{\boldsymbol{\varphi}}^T(x,y,z)\underset{\sim}{\mathbf{U}}$$

FEM (as Ritz Method)

TECHNISCHE UNIVERSITÄT WIEN

with

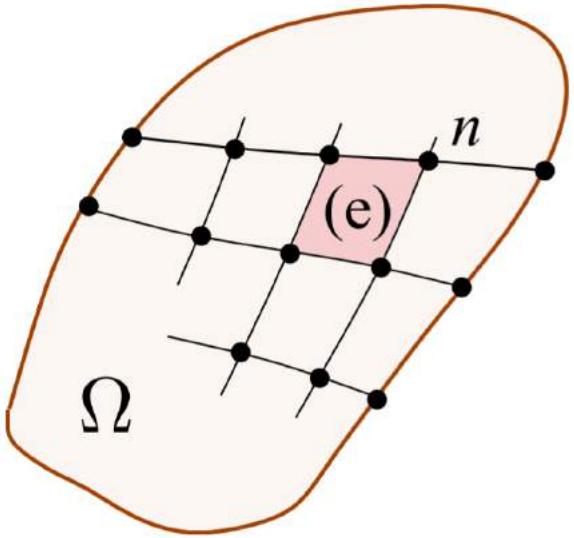
$$\underset{\approx}{\varphi}^{\mathrm{T}} = \begin{pmatrix} \underset{\sim}{\varphi_u}^{\mathrm{T}} & \underset{\sim}{0}^{\mathrm{T}} & \underset{\sim}{0}^{\mathrm{T}} \\ \underset{\sim}{0}^{\mathrm{T}} & \underset{\sim}{\varphi_v}^{\mathrm{T}} & \underset{\sim}{0}^{\mathrm{T}} \\ \underset{\sim}{0}^{\mathrm{T}} & \underset{\sim}{0}^{\mathrm{T}} & \underset{\sim}{\varphi_w}^{\mathrm{T}} \end{pmatrix}, \qquad \underset{\sim}{U} = \begin{pmatrix} \underset{\sim}{U_u} \\ \underset{\sim}{U_v} \\ \underset{\sim}{U_w} \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ v_1 \\ v_2 \\ \vdots \\ w_1 \\ w_2 \\ \vdots \end{pmatrix}$$

with $u_1, u_2, u_3 \ldots. u_n$ being the x −components of the displacement vector of the nodes 1,2,3...n; likewise for v_i and w_i. The displacement in x − direction of any point $(x, y, z) \in \Omega$ can be given as

$$\begin{aligned} u(x,y,z) &= u_1\varphi_{u_1}(x,y,z) + \ \ldots u_n\varphi_{u_n}(x,y,z)\cdots = \\ &= U_{u_1}\varphi_{u_1}(x,y,z) + \ldots U_{u_n}\varphi_{u_n}(x,y,z)\ldots \end{aligned}$$

But, This is still not easy, because φ_u are for entire domain.

Ansatz for Small Sub-Domains- “Finite Elements”



- The domain is divided into small sub-domains, the *Finite Elements* (e) and their boundaries form the *mesh* which is anchored at the n.
- The definition of the interpolation functions is given for the sub-domains, only, but no longer for the entire domain Ω.
- Consequently, the displacement field at any point inside the Finite Element $(x, y, z) \in (e)$ reads

$$\vec{u}(x, y, z) = \underset{\approx}{\varphi}^{(e)^T}(x, y, z)\underset{\sim}{U}^{(e)}$$

e.g.~for the x-component

$$u(x,y,z) = \sum_{j=1}^{N^{(e)}} \varphi_{u,j}^{(e)}(x,y,z) U_{u_j}^{(e)} = \boldsymbol{\varphi}_{\underset{\sim}{u}}^{(e)\,\mathrm{T}}(x,y,z)\, \boldsymbol{U}_{\underset{\sim}{u}}{}^{(e)}$$

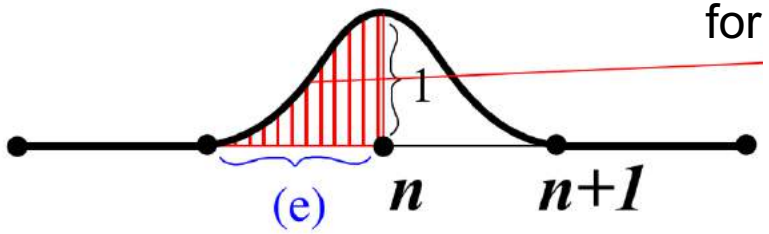
where $j = 1,2, \ldots N^{(e)}$ denote the local node numbers of the element (e).

From this, it gets obvious that the interpolation function $\varphi_i^{(e)}$ is associated to the node i and that it interpolates the contribution of node i over the element. The value at the node must be ``one", generally written as,

$$\varphi_{k_j}^{(e)}\left(\vec{X}_i^{(e)}\right) = \delta_{ij}, \qquad k = u, v, w, \qquad i, j \ldots \text{node numbers}$$

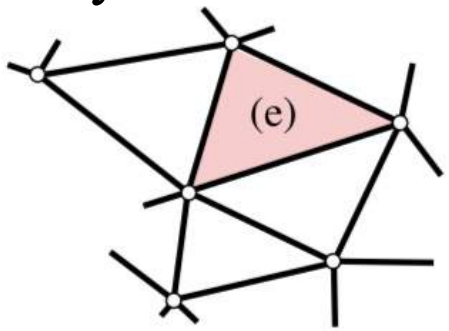
and “zero” at the location of all other nodes in the element. The interpolation function $\varphi_i^{(e)}$ means: contribution of the node i to the displacement field of the element (e)

Interpolation function for n



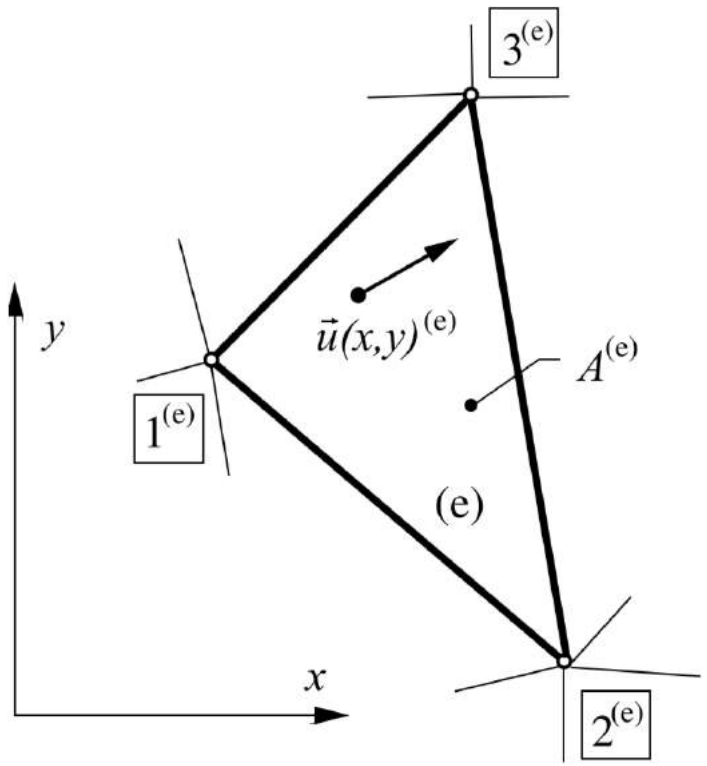
The following advantages arise;

- substantially simpler interpolation functions φ.
- much easier to satisfy the essential boundary conditions.
- topological equivalent elements have interpolation functions of the same type, e.g. triangular elements
- complex shaped geometries can be modeled, i.e. the domain boundary
- suitable for automation (computer implementation)



Example-Constant Strain Triangle

As example a plane problem is shown where nodes have only two components of coordinates and displacement, respectively. For a triangular element the interpolation of the displacements and the strains will be shown.



$$\underset{\sim}{U}^{(e)} = \begin{pmatrix} u_1^{(e)} \\ u_2^{(e)} \\ u_3^{(e)} \\ v_1^{(e)} \\ v_2^{(e)} \\ v_3^{(e)} \end{pmatrix} = \begin{pmatrix} \underset{\sim}{U}_{u^{(e)}} \\ \underset{\sim}{U}_{v^{(e)}} \end{pmatrix}$$

$$\vec{u}(x,y) = \underset{\approx}{\varphi}^{(e)^T}(x,y)\underset{\sim}{U}^{(e)}$$

$$u^{(e)} = \underset{\sim}{\varphi}_u^{(e)^{\mathrm{T}}} \underset{\sim}{U}_{u^{(e)}}$$

$$v^{(e)} = \underset{\sim}{\varphi}_v^{(e)^{\mathrm{T}}} \underset{\sim}{U}_v^{(e)}$$

Example-Constant Strain Triangle

$$\boldsymbol{\varphi}_{\underset{\sim}{u}}^{(\mathbf{e})} = \frac{1}{2A^{(\mathrm{e})}} \begin{pmatrix} X_{x_2} X_{y_3} - X_{x_3} X_{y_2} + (X_{y_2} - X_{y_3})x + (X_{x_3} - X_{x_2})y \\ X_{x_3} X_{y_1} - X_{x_1} X_{y_3} + (X_{y_3} - X_{y_1})x + (X_{x_1} - X_{x_3})y \\ X_{x_1} X_{y_2} - X_{x_2} X_{y_1} + (X_{y_1} - X_{y_2})x + (X_{x_2} - X_{x_1})y \end{pmatrix}^{(\mathrm{e})}$$

$$\boldsymbol{\varphi}_{\underset{\sim}{v}}^{(\mathbf{e})}(x, y) = \boldsymbol{\varphi}_{\underset{\sim}{u}}^{(\mathbf{e})}(x, y)$$

with $A^{(e)}$ being the area of the element which is computed as,

$$2A^{(\mathrm{e})} = X_{x_1}(X_{y_2} - X_{y_3}) + X_{x_2}(X_{y_3} - X_{y_1}) + X_{x_3}(X_{y_1} - X_{y_2})$$

Example-Constant Strain Triangle

TECHNISCHE UNIVERSITÄT WIEN

Consideration of the interpolation function at node $1^{(e)}$

$$\underset{\sim}{\varphi}_u^{(\mathrm{e})}(x_1, y_1) = \frac{1}{2A^{(\mathrm{e})}} \begin{pmatrix} X_{x_2}\, X_{y_3} - X_{x_3}\, X_{y_2} + X_{y_2}\, X_{x_1} - X_{y_3}\, X_{x_1} + X_{x_3}\, X_{y_1} - X_{x_2}\, X_{y_1} \\ X_{x_3}\, X_{y_1} - X_{x_1}\, X_{y_3} + X_{y_3}\, X_{x_1} - X_{y_1}\, X_{x_1} + X_{x_1}\, X_{y_1} - X_{x_3}\, X_{y_1} \\ X_{x_1}\, X_{y_2} - X_{x_2}\, X_{y_1} + X_{y_1}\, X_{x_1} - X_{y_2}\, X_{x_1} + X_{x_2}\, X_{y_1} - X_{x_1}\, X_{y_1} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

shows that it is “1” at the associated node and “0” at the others.

The strain component ε_{xx} follows form

$$\varepsilon_{xx} = \frac{\partial u}{\partial x} \qquad u(x, y, z)^{(\mathrm{e})} = \boldsymbol{\varphi}_{\underset{\sim}{u}}^{(\mathbf{e})^{\mathrm{T}}} \boldsymbol{U}_{\underset{\sim}{u}}^{(\mathbf{e})} \quad \Rightarrow \quad \frac{\partial u}{\partial x} = \frac{\partial}{\partial x}(\boldsymbol{\varphi}_{\underset{\sim}{u}}^{(\mathbf{e})^{\mathrm{T}}})\, \boldsymbol{U}_{\underset{\sim}{u}}^{(\mathbf{e})} = \frac{1}{2A^{(\mathrm{e})}} \begin{pmatrix} X_{y_2} - X_{y_3} \\ X_{y_3} - X_{y_1} \\ X_{y_1} - X_{y_2} \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} u_1^{(\mathrm{e})} \\ u_2^{(\mathrm{e})} \\ u_3^{(\mathrm{e})} \end{pmatrix}$$

$$\Rightarrow \quad \varepsilon_{xx}(x, y)^{(\mathrm{e})} = \mathrm{const.}^{(\mathrm{e})}$$

Example-Constant Strain Triangle

The result shows, that the strain ε_{xx} (as well as ε_{yy} and ε_{xy}) are independent of $(x, y)^{(e)}$, i.e. it is constant in (e), which is the reason for the name *Constant Strain Triangle*.

Note:

- Even if the displacements are continuous at the element boundaries, the strains (and stresses) are dis-continuous.
- The local balance equations are not fulfilled locally, however, the nodal forces and displacements are in equilibrium. Consequently, the total energy balance in the element is fulfilled which is called the "weak form" of the solution.

Derivation of Stiffness Matrices

Next the element stiffness matrix is sought for which relates the nodal forces to the nodal displacements. Here, the case is treated in which the element coordinates are given in the global coordinate system (e.g. corresponding to the Constant Strain Triangle in the previous section). A more general framework based on local element considerations will be presented later.

$$V = \sum_{(\mathrm{e})} V^{(\mathrm{e})} \quad , \quad V^{(\mathrm{e})} = U^{(\mathrm{e})} + W^{(\mathrm{e})},$$

with its contributions given at element level (e).

The strain energy of the element can be expressed as,

$$U^{(e)} = \frac{1}{2}\int_{V^{(e)}} \sigma_{ij}\varepsilon_{ij}\mathrm{dV} = \frac{1}{2}\int_{V^{(e)}} \underset{\sim}{\boldsymbol{\sigma}}^T \underset{\sim}{\boldsymbol{\varepsilon}}\,\mathrm{dV}$$

$$\underset{\sim}{\boldsymbol{\sigma}} = \underset{\approx}{\boldsymbol{E}}^{(e)}\underset{\sim}{\boldsymbol{\varepsilon}} \quad , \quad \underset{\sim}{\boldsymbol{\sigma}}^T = \underset{\sim}{\boldsymbol{\varepsilon}}^T \underset{\approx}{\boldsymbol{E}}^{(e)^T} \quad , \quad \underset{\approx}{\boldsymbol{E}}^{(e)^T} = \underset{\approx}{\boldsymbol{E}}^{(e)}$$

and the definition $\underset{\sim}{\boldsymbol{\varepsilon}} = \underset{\approx}{\boldsymbol{D}}^{(e)}\underset{\sim}{\boldsymbol{U}}^{(e)}$ from $\underset{\sim}{\boldsymbol{\varepsilon}} = \underset{\approx}{\boldsymbol{d}}\overrightarrow{u} = \underset{\approx}{\boldsymbol{d}}\underset{\approx}{\boldsymbol{\varphi}}^{(e)^{\mathrm{T}}}\underset{\sim}{\boldsymbol{U}}^{(e)}$

Derivation of Stiffness Matrices

follows

$$U^{(e)} = \frac{1}{2}\int_{V^{(e)}} \underset{\sim}{\boldsymbol{U}}^{(e)^T} \underset{\approx}{\boldsymbol{D}}^{(e)^T} \underset{\approx}{\boldsymbol{E}}^{(e)} \underset{\approx}{\boldsymbol{D}}^{(e)} \underset{\sim}{\boldsymbol{U}}^{(e)} \mathrm{dV} = \frac{1}{2} \underset{\sim}{\boldsymbol{U}}^{(e)^T} \int_{V^{(e)}} \underset{\approx}{\boldsymbol{D}}^{(e)^T} \underset{\approx}{\boldsymbol{E}}^{(e)} \underset{\approx}{\boldsymbol{D}}^{(e)} \mathrm{dV} \underset{\sim}{\boldsymbol{U}}^{(e)}$$

From solid mechanics, it is known that the energy is a quadratic form of the displacements with the stiffness

$$U^{(e)} = \frac{1}{2} \underset{\sim}{\boldsymbol{U}}^{(e)^T} \underset{\approx}{\boldsymbol{K}}^{(e)} \underset{\sim}{\boldsymbol{U}}^{(e)}$$

which renders the element stiffness matrix

$$\underset{\approx}{\boldsymbol{K}}^{(e)} = \int_{V^{(e)}} \underset{\approx}{\boldsymbol{D}}^{(e)^T} \underset{\approx}{\boldsymbol{E}}^{(e)} \underset{\approx}{\boldsymbol{D}}^{(e)} \mathrm{dV}$$

Assembly of the Total Stiffness Matrix

TECHNISCHE UNIVERSITÄT WIEN

Since the assembly of all elements has to solved for equilibrium, the total stiffness matrix has to be built from all element stiffness matrices. The coincidence transformation matrix, $\underset{\approx}{\bar{T}}^{(e)}$ relates the total displacement vector $\underset{\sim}{U}$ to the element displacements as,

$$\underset{\sim}{U}^{(e)} = \underset{\approx}{\bar{T}}^{(e)} \underset{\sim}{U}$$

Then the strain energy of the element (e) reads

$$U^{(e)} = \frac{1}{2} \underset{\sim}{U}^T \underset{\approx}{\bar{T}}^{(e)^T} \underset{\approx}{K}^{(e)} \underset{\approx}{\bar{T}}^{(e)} \underset{\sim}{U}$$

from which the transformation of the element stiffness matrix follows

$$\underset{\approx}{\bar{K}}^{(e)} = \underset{\approx}{\bar{T}}^{(e)^T} \underset{\approx}{K}^{(e)} \underset{\approx}{\bar{T}}^{(e)}$$

With $\underset{\approx}{\bar{K}}^{(e)}$ being the element stiffness in global (total) consideration.

Assembly of the Total Stiffness Matrix

TECHNISCHE UNIVERSITÄT WIEN

Applying the principle of minimum total potential energy $\frac{\partial V}{\partial U_i} = 0 \; \forall i$ gives

$$\sum_{(e)} \bar{\underset{\approx}{T}}^{(e)^{\mathrm{T}}} \underset{\approx}{K}^{(e)} \bar{\underset{\approx}{T}}^{(e)} \underset{\sim}{U} - \sum_{(e)} \bar{\underset{\approx}{T}}^{(e)^{\mathrm{T}}} \underset{\sim}{F}^{(e)} = \underset{\sim}{0},$$

which can be written as

$$\underset{\approx}{K} \underset{\sim}{U} = \underset{\sim}{F}$$

System of Linear equations $\longrightarrow$ Solve

The expression,

$$\underset{\approx}{K} = \sum_{(e)} \bar{\underset{\approx}{T}}^{(e)^{\mathrm{T}}} \underset{\approx}{K}^{(e)} \bar{\underset{\approx}{T}}^{(e)} = \sum_{(e)} \bar{\underset{\approx}{K}}^{(e)}$$

is the total stiffness matrix, and

$$\underset{\sim}{F} = \sum_{(e)} \bar{\underset{\approx}{T}}^{(e)^{\mathrm{T}}} \underset{\sim}{F}^{(e)}$$

Is the total load vector.

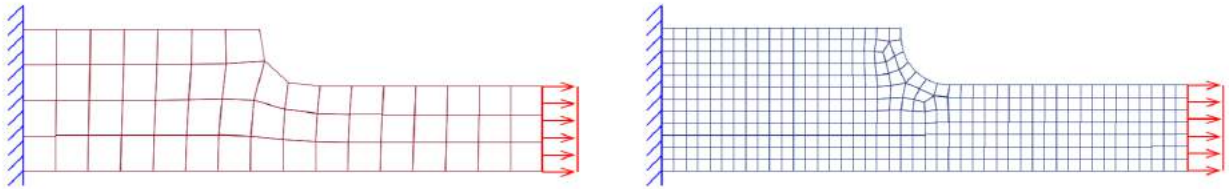
Some properties of the mesh

Consistent Nodal Loads

Since only nodal forces appear in $\underset{\approx}{K}\underset{\sim}{U} = \underset{\sim}{F}$ which take action at the nodes, surface force and volume force densities must be converted to nodal forces. This must be done element wise, such that the work by the nodal loads is the same as the work by the distributed force densities, i.e. in an energetically consistent way.

Discretization and accuracy

The accuracy of the solution depends (amount others) on the element formulation and on the mesh density.



The Finite Element Method also allows for selective mesh refinements in regions of interest or where high accuracy is desired.

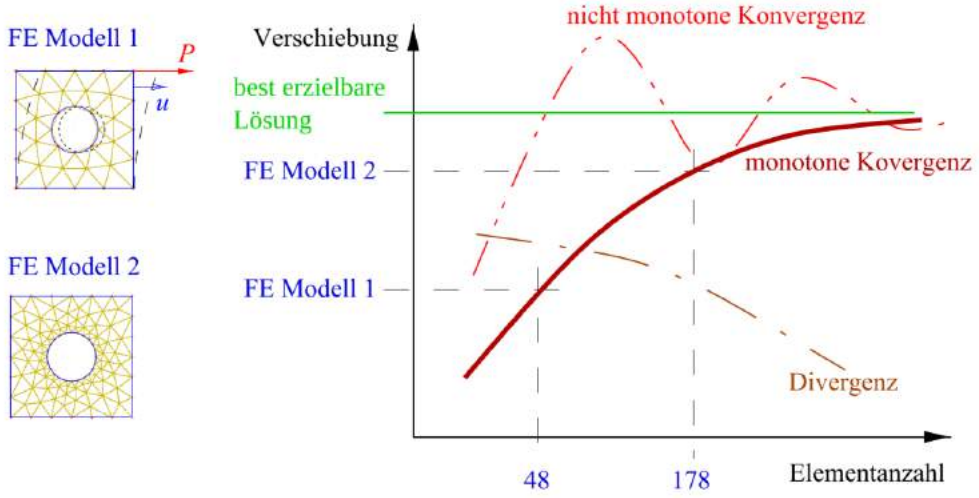
Some properties

Convergence

Monotonous convergence means, that with systematic increase of the mesh density the solution tends to the ``true'' response.

Completeness

Interpolation functions are required to be *complete*, i.e. to be able to describe rigid body movements and to be able describe states of constant strains.



Compatibility

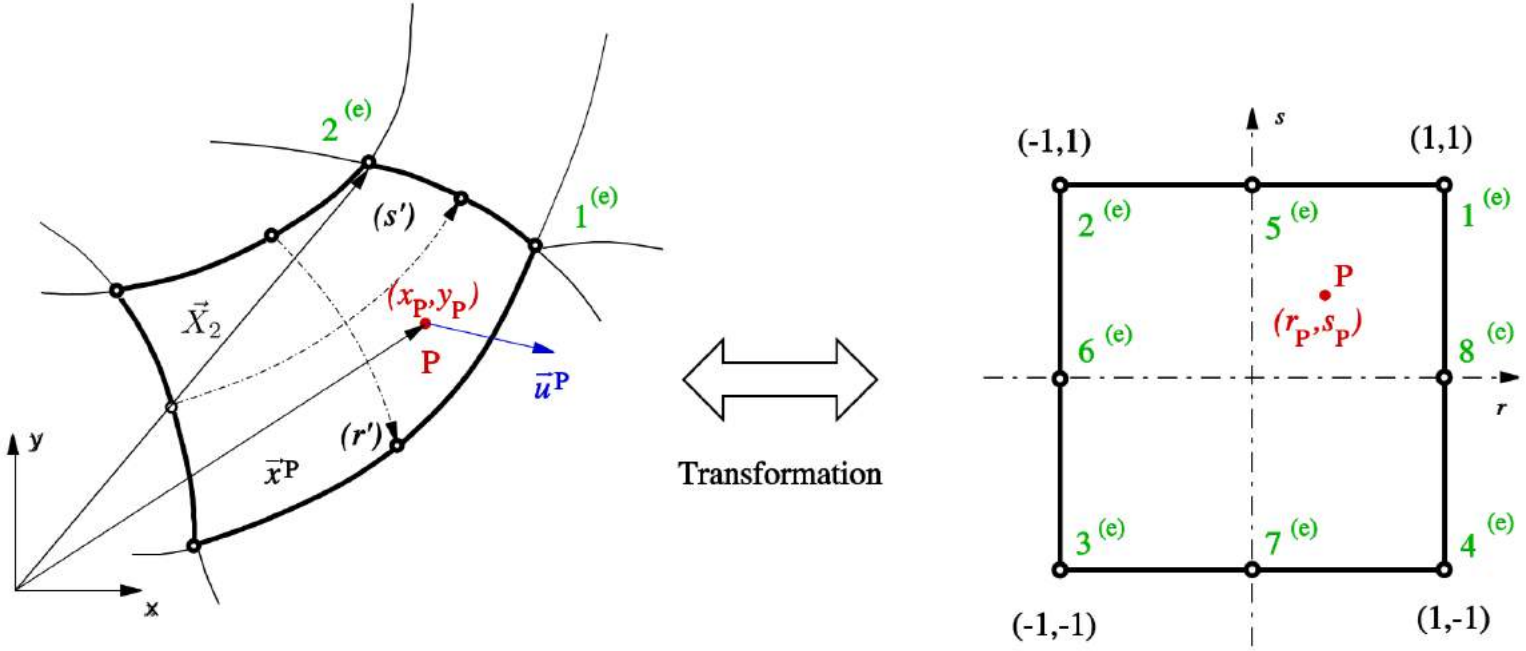
The displacement field at element boundaries must be continuous to prevent overlapping and formation of gaps inside a continuum.

If adjacent elements have different interpolation functions, special measures must be take to achieve compatibility.

Isoparametric Finite Elements

An important step towards easy automation and efficiency has been the introduction of local reference elements.

Interpolation of Geometry and Displacement Field



- transformation to local reference element.
- pre-computations for all elements of the same type.

Isoparametric Finite Elements

TECHNISCHE UNIVERSITÄT WIEN

for a two-dimensional 8-noded quadrilateral element, this translates to

$$x_{\mathrm{P}} = \sum_{j=1}^{n} f_j^{(\mathrm{e})}(r_{\mathrm{P}}, s_{\mathrm{P}})\, X_{x_j} \qquad x = \underset{\sim}{\boldsymbol{f}}^{(\mathrm{e})^T}(r,s)\, \underset{\sim}{\boldsymbol{X}}_x^{(\mathrm{e})},$$

$$y_{\mathrm{P}} = \sum_{j=1}^{n} f_j^{(\mathrm{e})}(r_{\mathrm{P}}, s_{\mathrm{P}})\, X_{y_j} \qquad y = \underset{\sim}{\boldsymbol{f}}^{(\mathrm{e})^T}(r,s)\, \underset{\sim}{\boldsymbol{X}}_y^{(\mathrm{e})},$$

$$\vec{x} = \underset{\approx}{\boldsymbol{f}}^{(\mathrm{e})^{\mathrm{T}}}(r,s)\underset{\sim}{\boldsymbol{X}}^{(\mathrm{e})}.$$

$(x_{\mathrm{P}}, y_{\mathrm{P}})$ Coordinates of point $P \in (e)$

(X_{x_j}, X_{y_j}) Coordinates of node j

$\underset{\sim}{\boldsymbol{X}}_x^{(\mathrm{e})}$ x −coordinates of the node $(X_{x_1}, X_{x_2}, \dots, X_{x_n})^T$

$\underset{\sim}{\boldsymbol{X}}_y^{(\mathrm{e})}$ y −coordinates of the node $(X_{y_1}, X_2, \dots, X_{y_n})^T$

where $\vec{x} = (x, y)^T$ and $\underset{\sim}{\boldsymbol{X}}^{(e)} = (\underset{\sim}{\boldsymbol{X}}_x^{(e)^{\mathrm{T}}}, \underset{\sim}{\boldsymbol{X}}_y^{(e)^{\mathrm{T}}})^{\mathrm{T}}$.

Isoparametric Finite Elements

The interpolation of the displacements are given by

$$u^{\mathrm{P}} = \sum_{j=1}^{n} N_j^{(\mathrm{e})}(r_{\mathrm{P}}, s_{\mathrm{P}}) U_{uj}^{(\mathrm{e})} \qquad u = \underset{\sim}{\boldsymbol{N}}^{(\mathrm{e})^{\mathrm{T}}}(r,s)\underset{\sim}{\boldsymbol{U}_{\boldsymbol{u}}}^{(\mathrm{e})} \qquad \underset{\sim}{\boldsymbol{\varphi}_{\boldsymbol{u}}}^{\mathrm{T}} \longrightarrow \underset{\sim}{\boldsymbol{N}_{\boldsymbol{u}}}^{\mathrm{T}} \longrightarrow \underset{\sim}{\boldsymbol{N}}^{\mathrm{T}},$$

$$v^{\mathrm{P}} = \sum_{j=1}^{n} N_j^{(\mathrm{e})}(r_{\mathrm{P}}, s_{\mathrm{P}})\, U_{vj}^{(\mathrm{e})} \qquad v = \underset{\sim}{\boldsymbol{N}}^{(\mathrm{e})^{\mathrm{T}}}(r,s)\underset{\sim}{\boldsymbol{U}_{\boldsymbol{v}}}^{(\mathrm{e})} \qquad \underset{\sim}{\boldsymbol{N}_{\boldsymbol{v}}} = \underset{\sim}{\boldsymbol{N}_{\boldsymbol{u}}} = \underset{\sim}{\boldsymbol{N}}.$$

Note, that the interpolation functions $f_j^{(e)}(r,s)$ and $N_j^{(e)}(r,s)$, are defined in the local (r,s) space. All elements of the same type have identical interpolation functions, and the superscript (e) can be omitted, giving, $\vec{u} = \underset{\approx}{\boldsymbol{N}}^{\mathrm{T}}(r,s)\underset{\sim}{\boldsymbol{U}}^{(\mathrm{e})}$

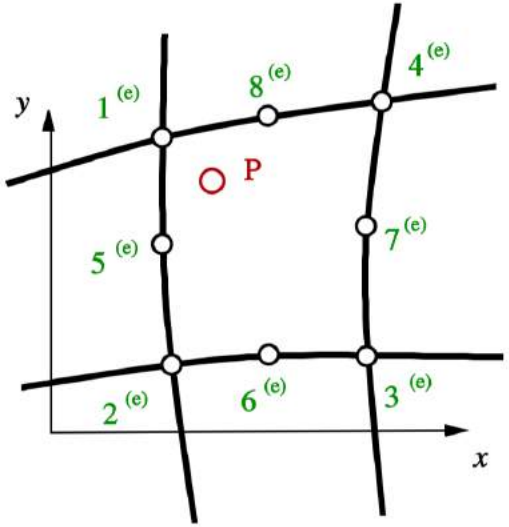
For the case of isoparametric elements $f_j(r,s) \equiv N_j(r,s) \quad j = 1,2,\ldots,n$. Again, completeness is desired

$$f_j\left(r_i, s_i, (t_i)\right) = \delta_{ij}$$
$$N_j\left(r_i, s_i, (t_i)\right) = \delta_{ij}$$

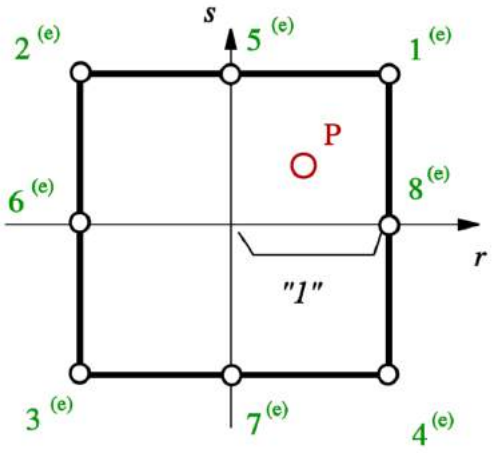
$i, j \ldots$ node number at reference elements

Example

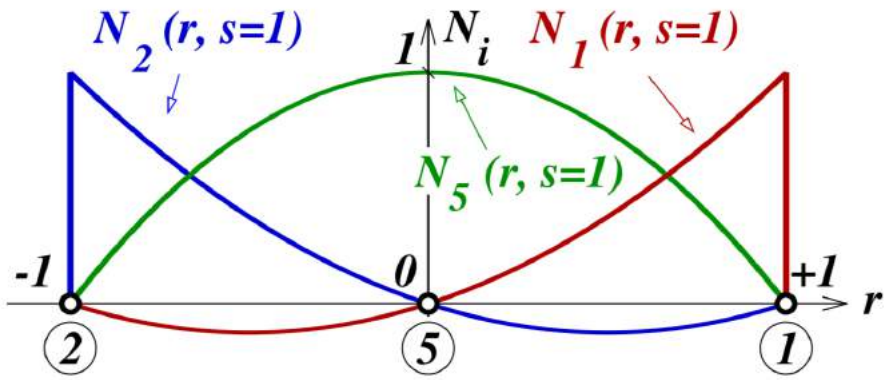
Isoparametric *quadrilateral element* with 8 nodes and bi-quadratic interpolation



Transformation



Discussion for $s = 1$



$$N_i(r,s) = \frac{1}{4}(1+rr_i)(1+ss_i)(rr_i+ss_i-1) \quad \text{for } i=1,2,3,4$$

$$N_i(r,s) = \frac{1}{2}(1-r^2)(1+ss_i) \quad \text{for } i=5,7$$

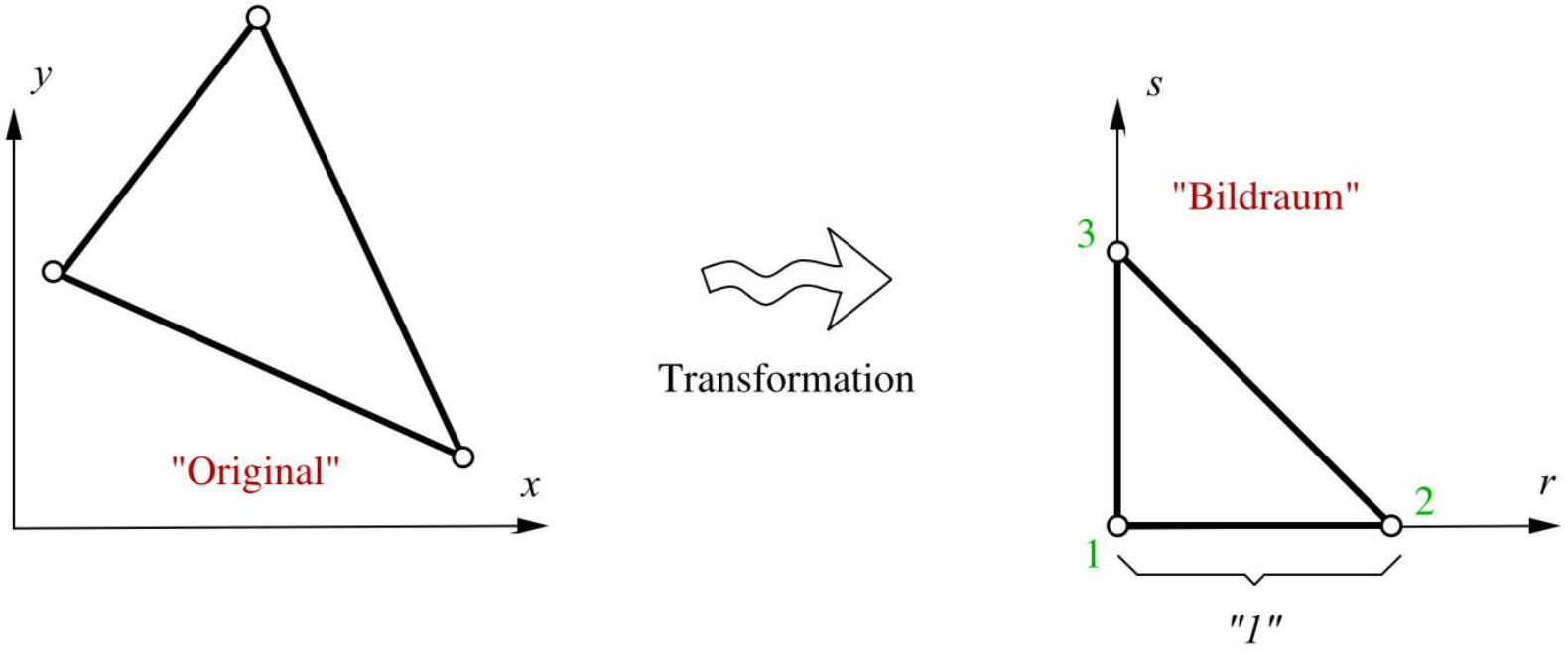
$$N_i(r,s) = \frac{1}{2}(1+rr_i)(1-s^2) \quad \text{for } i=6,8$$

check for completeness,

$$\sum_{i=1}^{8} N_i(r,s) \equiv 1.$$

Example

Isoparametric *Constant Strain Triangle*



$$\underset{\sim}{N}(r,s) = \begin{pmatrix} 1-r-s \\ r \\ s \end{pmatrix}$$

The "Jacobi Matrix" of the Element

The computation of the strains requires the partial derivatives,

$$\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \ldots, \frac{\partial v}{\partial x}, \ldots,$$

$$\text{z.B.} \quad \varepsilon_{xx} = \frac{\partial u}{\partial x}, \ldots ,$$

but they cannot be given straightforward since, $\underset{\sim}{\boldsymbol{u}} = \underset{\approx}{\boldsymbol{N}}^T(r,s,t)\underset{\sim}{\boldsymbol{U}}^{(\mathrm{e})}$

For two-dimensional cases we can write,

$$\left.\begin{aligned} \frac{\partial}{\partial r} &= \frac{\partial}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial}{\partial y}\frac{\partial y}{\partial r} \\ \frac{\partial}{\partial s} &= \frac{\partial}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial}{\partial y}\frac{\partial y}{\partial s} \end{aligned}\right\} \Leftrightarrow \begin{pmatrix} \frac{\partial}{\partial r} \\ \frac{\partial}{\partial s} \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \end{pmatrix} \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix} \Leftrightarrow \begin{pmatrix} \frac{\partial}{\partial r} \\ \frac{\partial}{\partial s} \end{pmatrix} =$$

$$= \underset{\approx}{\boldsymbol{J}}^{(\mathrm{e})}(r,s) \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix} \quad \Longleftrightarrow \quad \underset{\sim}{\boldsymbol{\partial}}_{rs} = \underset{\approx}{\boldsymbol{J}}^{(\mathrm{e})}(r,s)\underset{\sim}{\boldsymbol{\partial}}_{xy}.$$

$\underset{\approx}{\boldsymbol{J}}^{(\mathrm{e})}(r,s)$is the "Jacobi Matrix of element "(e)

The “Jacobi Matrix” of the Element

Inversion gives the desired derivatives,

$$\underset{\sim}{\boldsymbol{\partial}}_{xy} = \underset{\approx}{\boldsymbol{J}}^{(\mathrm{e})^{-1}}(r,s)\underset{\sim}{\boldsymbol{\partial}}_{rs}.$$

Note that, because of $\vec{x} = \underset{\approx}{\boldsymbol{N}}^{\mathrm{T}}(r,s,t)\underset{\sim}{\boldsymbol{X}}$, the jacobi matrix $\underset{\approx}{\boldsymbol{J}}^{(\mathrm{e})}$ depends only on the global coordinates $\underset{\sim}{\boldsymbol{X}}^{(\mathrm{e})}$.

Inversion of the Jacobi matrix is possible only if it is non-singular, i.e $\underset{\approx}{\boldsymbol{J}}^{(\mathrm{e})}(r,s,t) \neq 0 \text{for all}(r,s,t)$
This gives restrictions to element shapes.

The volume of the element is given by $\mathrm{dV} = \mathrm{d}x\mathrm{d}y\mathrm{d}z = \det\underset{\approx}{\boldsymbol{J}}^{(\mathrm{e})}\,\mathrm{d}r\,\mathrm{d}s\,\mathrm{d}t$.

The Stiffness Matrix of Isoparametric Elements

The element stiffness matrix is computed as

$$\underset{\approx}{\boldsymbol{K}}^{(\mathrm{e})} = \int_{V^{(\mathrm{e})}} \underset{\approx}{\boldsymbol{D}}^{(\mathrm{e})^T} \underset{\approx}{\boldsymbol{E}}^{(\mathrm{e})} \underset{\approx}{\boldsymbol{D}}^{(\mathrm{e})} \mathrm{dV} \qquad with \quad \underset{\approx}{\boldsymbol{D}}^{(\mathrm{e})} = \underset{\approx}{\boldsymbol{d}}^{(\mathrm{e})} \underset{\approx}{\boldsymbol{N}}^T \quad .$$

For a two-dimensional element (for example) this reads,

$$\underset{\approx}{\boldsymbol{D}}^{(\mathrm{e})} = \begin{pmatrix} \frac{\partial}{\partial x} \underset{\sim}{\boldsymbol{N}}^{\mathrm{T}} & 0^{\mathrm{T}} \\ 0^{\mathrm{T}} & \frac{\partial}{\partial y} \underset{\sim}{\boldsymbol{N}}^{\mathrm{T}} \\ \frac{\partial}{\partial y} \underset{\sim}{\boldsymbol{N}}^{\mathrm{T}} & \frac{\partial}{\partial x} \underset{\sim}{\boldsymbol{N}}^{\mathrm{T}} \end{pmatrix}$$

The (1,1)-element of can be expressed as $\frac{\partial}{\partial x} \underset{\sim}{\boldsymbol{N}}^{\mathrm{T}} = \frac{1}{\det \underset{\approx}{\boldsymbol{J}}^{(\mathrm{e})}} \left(\frac{\partial y}{\partial s} \frac{\partial}{\partial r} \underset{\sim}{\boldsymbol{N}}^{\mathrm{T}} - \frac{\partial y}{\partial r} \frac{\partial}{\partial s} \underset{\sim}{\boldsymbol{N}}^{\mathrm{T}} \right).$

The Stiffness Matrix of Isoparametric Elements

Introducing the short hand notation, $(\quad)' := \frac{\partial(\,)}{\partial r}\,, \quad (\dot{\ }) := \frac{\partial(\,)}{\partial s}$

The $\underset{\approx}{D}^{(\mathrm{e})}$ matrix can be written as

$$\underset{\approx}{D}^{(\mathrm{e})} = \frac{1}{\det \underset{\approx}{J}^{(\mathrm{e})}} \begin{pmatrix} \dot{y}(\underset{\sim}{N}^{\mathrm{T}})' - y'(\dot{\underset{\sim}{N}}^{\mathrm{T}}) & 0^{\mathrm{T}} \\ 0^{\mathrm{T}} & -\dot{x}(\underset{\sim}{N}^{\mathrm{T}})' + x'(\dot{\underset{\sim}{N}}^{\mathrm{T}}) \\ -\dot{x}(\underset{\sim}{N}^{\mathrm{T}})' + x'(\dot{\underset{\sim}{N}}^{\mathrm{T}}) & \dot{y}(\underset{\sim}{N}^{\mathrm{T}})' - y'(\dot{\underset{\sim}{N}}^{\mathrm{T}}) \end{pmatrix} = \frac{1}{\det \underset{\approx}{J}^{(\mathrm{e})}} \underset{\sim}{Q}^{(\mathrm{e})} \quad .$$

For isoparametric elements further holds,

$$x = \underset{\sim}{N}^{\mathrm{T}} \underset{\sim}{X}_x^{(\mathrm{e})} \quad , \quad y = \underset{\sim}{N}^{\mathrm{T}} \underset{\sim}{X}_y^{(\mathrm{e})}$$

$$\Longrightarrow x' = \frac{\partial x}{\partial r} = \frac{\partial}{\partial r} \underset{\sim}{N}^{\mathrm{T}} \underset{\sim}{X}_x^{(\mathrm{e})} = \left(\underset{\sim}{N}^{\mathrm{T}}\right)' \underset{\sim}{X}_x^{(\mathrm{e})},$$

$$\dot{y} = \frac{\partial y}{\partial s} = \frac{\partial}{\partial s} \underset{\sim}{N}^{\mathrm{T}} \underset{\sim}{X}_y^{(\mathrm{e})} = \left(\dot{\underset{\sim}{N}}^{\mathrm{T}}\right) \underset{\sim}{X}_y^{(\mathrm{e})}.$$

(likewise $\dot{x}, y'$)

The Stiffness Matrix of Isoparametric Elements

Derivatives are required only for $\underset{\sim}{N}^{(\mathrm{e})^{\mathrm{T}}}$, which are equal for all elements of the same type. The global coordinates of the elements, $\underset{\sim}{X}^{(\mathrm{e})}$, appear as simple coefficients.

Applying the short hand notation gives,

$$\underset{\sim}{Q}^{(\mathrm{e})} = \begin{pmatrix} \left[(\dot{\underset{\sim}{N}}^{\mathrm{T}})\underset{\sim}{X}_y\right](\underset{\sim}{N}^{\mathrm{T}})' - \left[(\underset{\sim}{N}^{\mathrm{T}})'\underset{\sim}{X}_y\right](\dot{\underset{\sim}{N}}^{\mathrm{T}}) & 0^{\mathrm{T}} \\ 0^{\mathrm{T}} & \left[(\underset{\sim}{N}^{\mathrm{T}})'\underset{\sim}{X}_x\right](\dot{\underset{\sim}{N}}^{\mathrm{T}}) - \left[(\dot{\underset{\sim}{N}}^{\mathrm{T}})\underset{\sim}{X}_x\right](\underset{\sim}{N}^{\mathrm{T}})' \\ \left[(\underset{\sim}{N}^{\mathrm{T}})'\underset{\sim}{X}_x\right](\dot{\underset{\sim}{N}}^{\mathrm{T}}) - \left[(\dot{\underset{\sim}{N}}^{\mathrm{T}})\underset{\sim}{X}_x\right](\underset{\sim}{N}^{\mathrm{T}})' & \left[(\dot{\underset{\sim}{N}}^{\mathrm{T}})\underset{\sim}{X}_y\right](\underset{\sim}{N}^{\mathrm{T}})' - \left[(\underset{\sim}{N}^{\mathrm{T}})'\underset{\sim}{X}_y\right](\dot{\underset{\sim}{N}}^{\mathrm{T}}) \end{pmatrix}$$

and the $\underset{\approx}{D}^{(\mathrm{e})}$ matrix is a function of the local coordinates, $\underset{\approx}{D}^{(\mathrm{e})}(r,s)$ of the reference element (and the coefficients $\underset{\sim}{X}^{(\mathrm{e})}$. Now the element stiffness matrix can be written as,

The Stiffness Matrix of Isoparametric Elements

$$\int_{V^{(\mathrm{e})}} \underset{\approx}{\boldsymbol{D}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\boldsymbol{E}}^{(\mathrm{e})} \underset{\approx}{\boldsymbol{D}}^{(\mathrm{e})} \mathrm{dV} = \int_{-1}^{+1}\int_{-1}^{+1} \underset{\approx}{\boldsymbol{D}}^{(\mathrm{e})^{\mathrm{T}}}(r,s) \underset{\approx}{\boldsymbol{E}}^{(\mathrm{e})} \underset{\approx}{\boldsymbol{D}}^{(\mathrm{e})}(r,s) h \det \underset{\approx}{\boldsymbol{J}}^{(\mathrm{e})} \mathrm{d}r\mathrm{d}s,$$

$$= \int_{-1}^{+1}\int_{-1}^{+1} \frac{1}{\det \underset{\approx}{\boldsymbol{J}}^{(\mathrm{e})}} \underset{\sim}{\boldsymbol{Q}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\boldsymbol{E}}^{(\mathrm{e})} \underset{\sim}{\boldsymbol{Q}}^{(\mathrm{e})} h \mathrm{d}r\mathrm{d}s.$$

The integration is typically performed by Gauss quadrature, such as

$$\int_{\Omega} \Psi(r,s,t)\,\mathrm{d}r\,\mathrm{d}s\,\mathrm{d}t = \sum_{j=1}^{\mathcal{O}} \Psi(r_j,s_j,t_j)\bar{w}_j.$$

Further advantage of isoparametric elements

- Integration over local coordinates $\Psi(r,s,t)$.
- At reference element.
- Independent of global shape.

Closing Remarks

Various Extensions are readily developed and are widely used for FEM simulations. Among them are

- Transient problems are time varying loads and responses.
- “Dynamic” problems for which first and second time derivatives of the displacements enter the equations.
- Eigenvalue problems to study buckling and resonance.
- Non-linear formulations of the strain-displacement relation, the material repones, etc.
- Dissipative and load history dependent mechanisms.
- Multi-field and multi-physics coupled problems.

  and endlessly more.

Advantages of the FEM from the point of textbook listed in [2]

- It can deal simply with nonhomogeneous and anisotropic situations (particularly when the direction of anisotropy is variable)
- The elements can be graded in shape and size to follow arbitrary boundaries and to allow for regions of rapid variation of the function sought, thus controlling the errors in a most efficient way
- Specified gradient or “radiation” boundary conditions are introduced naturally and with a better accuracy.
- Higher order elements.. Can be readily used to improve accuracy without complicating boundary conditions
- Finally, but of considerable importance in the computer age, standard programs may be used for assembly and solution.

[2] O. C. Zeinkiewicz, R.L. Taylor, J. Zhu, The Finite Element Method: Its basis and Fundamentals, Butterworth Heinenmann, 2013

lecture notes

NSSC–II
The Finite Element Method

Some Basics in the Context of Solid Mechanics

2020 05 15

H.Pettermann
Institute of Lightweight Design and Structural Biomechanics
TU Wien

based on the "Vorlesungsbehelf zur Vorlesung"
Einfhrung in die Finite Elemente Methoden, *LVA-Nr. 317.016*, WS2019
by Em.o.Univ.Prof. DI Dr. F.G. Rammerstorfer
Institut fürLeichtbau und Struktur-Biomechanik,TU Wien

Inhaltsverzeichnis

1 Introduction **1**

2 Continuum Solid Mechanics **2**

2.1 Balance Equation 2

2.2 Displacement—Strain Relation 3

2.3 Constitutive Material Law 4

2.4 Strain Energy and Elastic Potential 5

3 Variational Principles **6**

3.1 The Principle of Virtual Displacements 6

3.2 Principle of Minimum Total Potential Energy 7

4 The RITZ' Approach (for Static Problems) **8**

5 The Finite Element Method (as Ritz' Method) **9**

5.1 A Special Ritz' Ansatz 9

5.2 Ansatz for Small Sub-Domains — "Finite Elements" 10

5.3 Example — Constant Strain Triangle 12

6 Derivation of Stiffness Matrices **13**

7 Discretization and Accuracy **16**

8 Isoparametric Finite Elements **18**

8.1 Interpolation of Geometry and Displacement Field 18

8.2 The "Jacobi Matrix" of the Element 21

8.3 The Element Stiffness Matrix of Isoparametric Elements 22

9 Closing Remarks **25**

1 Introduction

These lecture notes, and the accompanying classes, aim at a very short introduction into the Finite Element Method. This will be done in the context of solid mechanics, highlighting the intricate material behavior in linear elasticity based on 4th order tensors.
The subject is presented from the perspective of an (mechanical) engineer rather than from the point of view of mathematics. Nevertheless, the mathematical foundation shall become visible as well as the mechanical basics.

The user of Finite Element Method simulation software, and even more the developer, has to have knowledge on both the mathematics of the method and the physics of the modeled problem. This is required to set up appropriate models, to correctly interpret the simulation results, to know the limitations of the approach, and — if necessary — to improve the modeling.

Besides these lecture notes, an introduction to the Finite Element Method and to solid mechanics can be found in the first few chapters of [1]. Fundamental treatment is given e.g. in [2, 3]. The tutorial example for this class is based on [4].
All mentioned literature is available on-line for students of the TU Wien via the TU Wien library (with established VPN connection).

2 Continuum Solid Mechanics

The basics of continuum solid mechanics will be introduced and presented in matrix notation which is advantageous for software implementation. The equations pertain to linearized theory of elasticity for static problems.

2.1 Balance Equation

The local balance equation (condition of equilibrium) in differential form and index notation read,

$$\sigma_{ji,j} + q_i = 0, \quad \sigma_{ij} = \sigma_{ji} \qquad \forall\, i, j = 1, 2, 3\ (x, y, z)$$

(Einstein summation applies)

σ_{ji} are the components of the symmetric 2nd order stress tensor and $_{,j}$ denotes the partial derivative. q_i are the vector components of the volume force density, $\vec{q}$, i.e. $\vec{q}^{\,\mathrm{T}} = (q_x, q_y, q_z)$.

Fully expanded for *plane stress* state, the balance equation read,

$$\begin{aligned}
\sigma_{zz} &= 0, & \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{yx}}{\partial y} &= -q_x \\
\sigma_{zx} &= 0, & \frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} &= -q_y \\
\sigma_{zy} &= 0, & \sigma_{xy} &= \sigma_{yx}
\end{aligned}$$

Applying Voigt–Nye matrix notation, a differential operator matrix can be defined as

$$\begin{pmatrix} \frac{\partial}{\partial x} & 0 & \frac{\partial}{\partial y} \\ 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{pmatrix} \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} = - \begin{pmatrix} q_x \\ q_y \end{pmatrix}$$

For general tri-axial stress states the balance equation in Voigt–Nye notation read,

$$\begin{pmatrix} \frac{\partial}{\partial x} & 0 & 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial z} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{pmatrix} \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{pmatrix} = - \begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix}$$

in short,

$$\boxed{\underset{\approx}{\mathbf{d}}^{\mathrm{T}} \underset{\sim}{\sigma} = -\vec{q}}$$

with the corresponding operator matrix $\underset{\approx}{\mathbf{d}}^{\mathrm{T}}$.

The vector of the stress components, $\underset{\sim}{\sigma}$, is a vector in the sense of linear algebra, composed by a particular arrangement of the tensor components; it is not a physical entity. In contrast, the traction vector, $\vec{\sigma}_n$, is a physical entity with describes the force density (per area) with respect to the plane n.

2.2 Displacement—Strain Relation

The linearized displacement—strain relation reads in index notation,

$$\boxed{\varepsilon_{ij} = \tfrac{1}{2}\left(u_{i,j} + u_{j,i}\right)}$$

with $\vec{u}^{\,\mathrm{T}} = (u_1, u_2, u_3) = (u, v, w)$ being the local displacement vector,

For example,

$$\varepsilon_{xx} = \frac{\partial u}{\partial x} \quad , \quad \varepsilon_{xy} = \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) = \frac{\gamma_{xy}}{2}$$

Applying Voigt–Nye notation, the tri-axial relation reads,

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

in short,

$$\boxed{\underset{\approx}{\varepsilon} = \underset{\approx}{\mathbf{d}}\vec{u}}$$

Note 1: The shear contributions in $\underset{\approx}{\varepsilon}$ are expressed by the shear angles $\gamma_{ij} = 2\varepsilon_{ij}$ $(i \neq j)$ (which give rise to certain advantages).

Note 2: The differential operator matrix, $\underset{\approx}{\mathbf{d}}$, is indeed the transposed of the matrix in the balance equations, $\underset{\approx}{\mathbf{d}}^{\mathrm{T}}$, provided the above definition of the vector of strain components, $\underset{\sim}{\varepsilon}$ is used.

2.3 Constitutive Material Law

The relation between stresses and strains is given by the constitutive law. In linear elasticity it is called Hooke's Law, and reads in index notation,

$$\sigma_{ij} = E_{ijkl}\,\varepsilon_{kl}$$

Note, that the elasticity tensor, E_{ijkl}, is a 4th order tensor.

Applying Voigt–Nye matrix notation and assuming isotropic (i.e. direction independent) behavior, the constitutive law can be given as

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{pmatrix} = \underbrace{\frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & \nu & & & \\ \nu & 1-\nu & \nu & & \underset{\approx}{\mathbf{0}} & \\ \nu & \nu & 1-\nu & & & \\ & & & \frac{1-2\nu}{2} & 0 & 0 \\ & \underset{\approx}{\mathbf{0}} & & & \frac{1-2\nu}{2} & 0 \\ & & & & & \frac{1-2\nu}{2} \end{pmatrix}}_{\underset{\approx}{\mathbf{E}}} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{pmatrix}$$

with E being the Young's modulus and ν the Poisson ratio, stating two independent material properties which can be measured by experimental testing.

In short,

$$\boxed{\underset{\sim}{\sigma} = \underset{\approx}{\mathbf{E}}\,\underset{\sim}{\varepsilon}}$$

with the elasticity matrix $\underset{\approx}{\mathbf{E}}$.

The inverse relation can be given as,

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{pmatrix} = \underbrace{\frac{1}{E} \begin{pmatrix} 1 & -\nu & -\nu & & & \\ -\nu & 1 & -\nu & & \underset{\approx}{\mathbf{0}} & \\ -\nu & -\nu & 1 & & & \\ & & & 2(1+\nu) & 0 & 0 \\ & \underset{\approx}{\mathbf{0}} & & 0 & 2(1+\nu) & 0 \\ & & & 0 & 0 & 2(1+\nu) \end{pmatrix}}_{\underset{\approx}{\mathbf{C}}} \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{pmatrix}$$

or

$$\boxed{\underset{\sim}{\varepsilon} = \underset{\approx}{\mathbf{C}}\, \underset{\sim}{\sigma}}$$

with the *Compliance Matrix* $\underset{\approx}{\mathbf{C}} = \underset{\approx}{\mathbf{E}}^{-1}$.

2.4 Strain Energy and Elastic Potential

By use of the above conventions, the strain energy density can be expressed as,

$$\hat{U} = \frac{1}{2}\, \sigma_{ij}\, \varepsilon_{ij} = \frac{1}{2}\, \underset{\sim}{\sigma}^{\mathrm{T}}\, \underset{\sim}{\varepsilon}$$

from which follows the strain energy

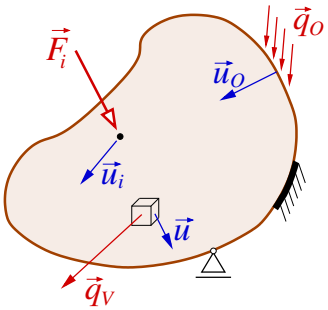
$$U = \int_{Vol} \hat{U}\, \mathrm{d}Vol$$

in a elastic body with volume Vol.

With the constitutive equation, $\underset{\sim}{\sigma} = \underset{\approx}{\mathbf{E}}\underset{\sim}{\varepsilon}$, the symmetry of the elasticity matrix, $\underset{\approx}{\mathbf{E}} = \underset{\approx}{\mathbf{E}}^{\mathrm{T}}$, and the linear algebra rule, $(\underset{\approx}{\mathbf{A}}\underset{\sim}{\mathbf{b}})^{\mathrm{T}} = \underset{\sim}{\mathbf{b}}^{\mathrm{T}}\underset{\approx}{\mathbf{A}}^{\mathrm{T}}$, follows,

$$U = \int_{Vol} \left(\frac{1}{2}\, \underset{\sim}{\varepsilon}^{\mathrm{T}} \underset{\approx}{\mathbf{E}}\, \underset{\sim}{\varepsilon} \right) \mathrm{d}Vol$$

The total potential, V, of an elastic body is the sum of the potential of the internal forces, U, (the strain energy) and the potential of the external forces, W.



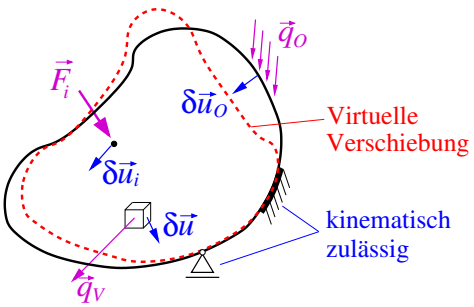
$$
\begin{aligned}
V &= U + W \\
W &= -\sum_i \vec{F}_i \cdot \vec{u}_i - \int_O \vec{q}_O \cdot \vec{u}_O \, \mathrm{d}O - \int_{Vol} \vec{q}_V \cdot \vec{u} \, \mathrm{d}Vol
\end{aligned}
$$

The potential of the external forces comprise the effects of concentrated forces, $\vec{F}_i$, surface force densities, $\vec{q}_O$, and volume force densities, $\vec{q}_V$.

3 Variational Principles

One way to find equilibrium is the application of variational principles. Here, in the context of "displacement formulated Finite Elements" the principle of virtual displacements and the principle of minimum total potential energy will be shown.
Alternatively, the method of the weighted residuals can be applied in the context of the Finite Element Method (which is not followed here).

3.1 The Principle of Virtual Displacements



The virtual displacement is

- infinitesimally small
- kinematic admissible

A body is in equilibrium, if the virtual work is zero,

$$\boxed{\delta A = \delta A^{(\mathrm{a})} + \delta A^{(\mathrm{i})} = 0}$$

with the contribution of the external (a) and internal (i) work caused by virtual displace-

ments given by,

$$\delta A^{(\mathrm{a})} = \sum_i \vec{F}_i \cdot \delta\vec{u}_i + \int_O \vec{q}_O \cdot \delta\vec{u}_O \,\mathrm{d}O + \int_{Vol} \vec{q}_V \cdot \delta\vec{u} \,\mathrm{d}Vol$$

$$\delta A^{(\mathrm{i})} = -\int_{Vol} \sigma_{ij}\, \delta\varepsilon_{ij} \,\mathrm{d}Vol = -\int_{Vol} \underset{\sim}{\sigma}^{\mathrm{T}}\, \delta\underset{\sim}{\varepsilon} \,\mathrm{d}Vol \qquad \delta\underset{\sim}{\varepsilon} = \mathfrak{F}(\delta\vec{u})$$

Note: In the Finite Element Method all surface and volume force densities are converted to "consistent" nodal forces. Consequently,

$$\delta A^{(\mathrm{a})} = \underset{\sim}{F}^{\mathrm{T}}\, \delta\underset{\sim}{U}$$

with $\underset{\sim}{F}$ ... vector of nodal forces
$\underset{\sim}{U}$ ... vector nodal displacements

Then the principle of virtual displacements reads,

$$\int_{Vol} \underset{\sim}{\sigma}^{\mathrm{T}} \delta\underset{\sim}{\varepsilon} \mathrm{d}Vol = \underset{\sim}{F}^{\mathrm{T}}\, \delta\underset{\sim}{U}$$

3.2 Principle of Minimum Total Potential Energy

The principle of minimum total potential energy can be applied for conservative systems where all forces have a potential.

Since the potential is the negative work,

$$\delta A^{(\mathrm{a})} = -\delta W, \qquad \delta A^{(\mathrm{i})} = -\delta U$$

the principle of virtual displacement, $\delta A^{(\mathrm{a})} + \delta A^{(\mathrm{i})} = 0$, can be written as $\delta W + \delta U = 0$, and with $V = U + W$ equilibrium is expressed as

$$\boxed{\delta V = 0}$$

The condition for a minimum (i.e. stable equilibrium) follows from the second variation,

$$\delta^2 V > 0$$

V in general is a *functional* of the displacements $\vec{u}(x, y, z)$ and their derivatives.

If a Ritz' Ansatz (see later) is used, the problem simplifies to find the stationary value of a *function* of the displacements.

4 The RITZ' Approach (for Static Problems)

For the displacement field $\vec{u}(x, y, z)$, i.e. for their components $u(x, y, z)$, $v(x, y, z)$, $w(x, y, z)$ a Ritz' Ansatz is introduced of the form (e.g. for the x-component)

$$u(x, y, z) \rightarrow u^* = \sum_{j=1}^{N} a_{u,j}\, \varphi_{u,j} \overset{Einstein}{=} a_{u,j}\, \varphi_{u,j} = \underset{\sim}{\varphi}_u^{\mathrm{T}}\, \underset{\sim}{a}_u$$

and for the complete displacement vector

$$\vec{u}^* = \underset{\approx}{\boldsymbol{\varphi}}^{\mathrm{T}} \underset{\sim}{a}$$

which results in an *approximation of the displacement field* $\vec{u}^*(x, y, z)$. Here, $\varphi_j(x, y, z)$ are predefined *interpolation functions* which have to satisfy the kinematic, also called essential or Dirichlet boundary conditions (among other conditions). The *generalized coordinates* a_j are free coefficients. With such a Ritz' Ansatz an approximation of the potential V^* is obtained which is a *function* of the generalized coordinates.

$$V^* = V^*(a_1, a_2, \dots)$$

Applying the equilibrium condition

$$\delta V^* = 0 \Leftrightarrow \frac{\partial V^*}{\partial a_k}\, \delta a_k = 0$$

gives rise to

$$\frac{\partial V^*}{\partial a_1} = 0,\ \frac{\partial V^*}{\partial a_2} = 0 \dots \frac{\partial V^*}{\partial a_j} = 0 \quad \forall\, j$$

5 The Finite Element Method (as Ritz' Method)

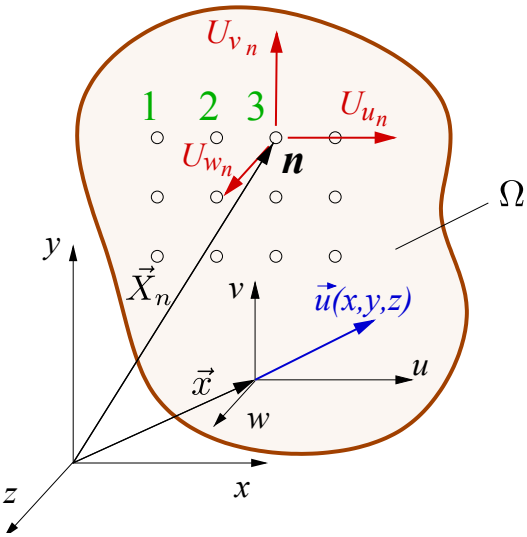
5.1 A Special Ritz' Ansatz

Before, the Ansatz was of the form

$$u(x,y,z) = a_{u_j}\,\varphi_{u_j}(x,y,z) = \underset{\sim}{\varphi}_u^{\mathrm{T}}\,\underset{\sim}{a}_u$$

where the coefficients a_j did not carry a mechanical "meaning".

Now the Ansatz is chosen such, that the coefficients are equal to the displacements of given points — the Finite Element *nodes* — in the domain. Each discrete node with number n is associated a set of interpolation functions $\varphi_{u_n}(x,y,z)$, $\varphi_{v_n}(x,y,z)$, $\varphi_{w_n}(x,y,z)$ valid over the entire domain Ω. Now, at any location $(x,y,z) \in \Omega$ the displacement vector can be expressed as



$$u = \sum_i U_{u_i}\varphi_{u_i}(x,y,z)\,; \quad v, w \text{ likewise;}$$

consequently

$$u(x,y,z) = \underset{\sim}{\varphi}_u^{\mathrm{T}}(x,y,z)\,\underset{\sim}{U}_u$$

$$v(x,y,z) = \underset{\sim}{\varphi}_v^{\mathrm{T}}(x,y,z)\,\underset{\sim}{U}_v$$

$$w(x,y,z) = \underset{\sim}{\varphi}_w^{\mathrm{T}}(x,y,z)\,\underset{\sim}{U}_w$$

in short

$$\Rightarrow \boxed{\vec{u}(x,y,z) = \underset{\approx}{\boldsymbol{\varphi}}^{\mathrm{T}}(x,y,z)\,\underset{\sim}{U}}$$

with

$$\underset{\approx}{\boldsymbol{\varphi}}^{\mathrm{T}} = \begin{pmatrix} \underset{\sim}{\varphi_u}^{\mathrm{T}} & \underset{\sim}{0}^{\mathrm{T}} & \underset{\sim}{0}^{\mathrm{T}} \\ \underset{\sim}{0}^{\mathrm{T}} & \underset{\sim}{\varphi_v}^{\mathrm{T}} & \underset{\sim}{0}^{\mathrm{T}} \\ \underset{\sim}{0}^{\mathrm{T}} & \underset{\sim}{0}^{\mathrm{T}} & \underset{\sim}{\varphi_w}^{\mathrm{T}} \end{pmatrix}, \qquad \underset{\sim}{U} = \begin{pmatrix} \underset{\sim}{U_u} \\ \underset{\sim}{U_v} \\ \underset{\sim}{U_w} \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ v_1 \\ v_2 \\ \vdots \\ w_1 \\ w_2 \\ \vdots \end{pmatrix}$$

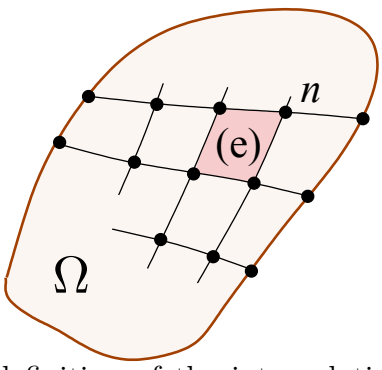
with u_1,u_2,$u_3 \ldots u_n$ being the x–components of the displacement vector of the nodes 1,2,3...n; likewise for v_i and w_i.

The displacement in x–direction of any point $(x, y, z) \in \Omega$ can be given as

$$\begin{aligned} u(x,y,z) &= u_1\varphi_{u_1}(x,y,z) + \ \ldots u_n\varphi_{u_n}(x,y,z)\cdots = \\ &= U_{u_1}\varphi_{u_1}(x,y,z) + \ldots U_{u_n}\varphi_{u_n}(x,y,z)\ldots \end{aligned}$$

For general case it is difficult (impossible) to find appropriate interpolation functions for the entire domain.

5.2 Ansatz for Small Sub-Domains — "Finite Elements"



The domain is divided into small sub-domains, the *Finite Elements* (e) and their boundaries form the *mesh* which is anchored at the *nodes* n.

The definition of the interpolation functions is given for the sub-domains, only, but no longer for the entire domain Ω (obeying some compatibility conditions, see later). Consequently, the displacement field at any point inside the Finite Element $(x, y, z) \in$ (e) reads

$$\boxed{\vec{u}(x,y,z) = \underset{\approx}{\boldsymbol{\varphi}}^{(\mathrm{e})^{\mathrm{T}}}(x,y,z)\,\underset{\sim}{U}^{(\mathrm{e})}}$$

e.g. for the x–component

$$u(x,y,z) = \sum_{i=1}^{N^{(\mathrm{e})}} \varphi_{u_i}^{(\mathrm{e})}(x,y,z)\,U_{u_i}^{(\mathrm{e})} = \underset{\sim}{\varphi}_u^{(\mathrm{e})^{\mathrm{T}}}\,\underset{\sim}{U}_u^{(\mathrm{e})}$$

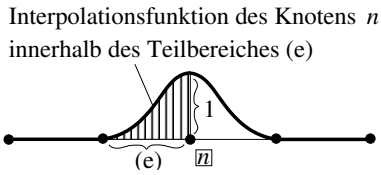
where $i = 1, 2....N^{(\mathrm{e})}$ denote the local node numbers of the element (e). From this, it gets obvious that the interpolation function $\varphi_i^{(\mathrm{e})}$ is associated to the node i and that it interpolates the contribution of node i over the element. The value at the node must be "one", generally written as,

$$\varphi_{k_j}^{(e)}\left(\vec{X}_i^{(e)}\right) = \delta_{ij}, \qquad k = u, v, w, \qquad i, j \ldots \text{node numbers}$$

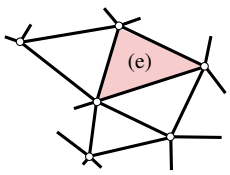
and "zero" at the location of all other nodes in the element.

The following advantages arise;

* substantially simpler interpolation functions φ



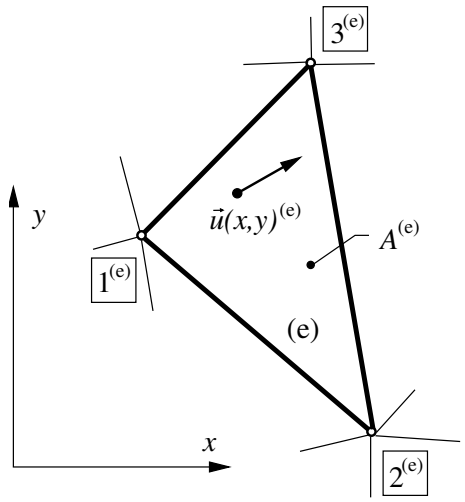
* much easier to satisfy the essential boundary conditions
* topological equivalent elements have interpolation functions of the same type, e.g. triangular elements



* complex shaped geometries can be modeled, i.e. the domain boundary
* suitable for automation (computer implementation)

5.3 Example — Constant Strain Triangle

As example a plane problem is shown where nodes have only two components of coordinates and displacement, respectively. For a triangular element the interpolation of the displacements and the strains will be shown.



$$\underset{\sim}{U}^{(e)} = \begin{pmatrix} u_1^{(e)} \\ u_2^{(e)} \\ u_3^{(e)} \\ v_1^{(e)} \\ v_2^{(e)} \\ v_3^{(e)} \end{pmatrix} = \begin{pmatrix} \underset{\sim}{U}_u^{(e)} \\ \underset{\sim}{U}_v^{(e)} \end{pmatrix}$$

$$\vec{u}(x,y)^{(e)} = \underset{\approx}{\boldsymbol{\varphi}}^{(e)^{\mathrm{T}}} \underset{\sim}{U}^{(e)}$$

$$u^{(e)} = \underset{\sim}{\varphi}_u^{(e)^{\mathrm{T}}} \underset{\sim}{U}_u^{(e)}$$

$$v^{(e)} = \underset{\sim}{\varphi}_v^{(e)^{\mathrm{T}}} \underset{\sim}{U}_v^{(e)}$$

$$\underset{\sim}{\varphi}_u^{(e)} = \frac{1}{2A^{(e)}} \begin{pmatrix} X_{x_2} X_{y_3} - X_{x_3} X_{y_2} + (X_{y_2} - X_{y_3})x + (X_{x_3} - X_{x_2})y \\ X_{x_3} X_{y_1} - X_{x_1} X_{y_3} + (X_{y_3} - X_{y_1})x + (X_{x_1} - X_{x_3})y \\ X_{x_1} X_{y_2} - X_{x_2} X_{y_1} + (X_{y_1} - X_{y_2})x + (X_{x_2} - X_{x_1})y \end{pmatrix}^{(e)}$$

$$\underset{\sim}{\varphi}_v^{(e)}(x,y) = \underset{\sim}{\varphi}_u^{(e)}(x,y)$$

with $A^{(e)}$ being the area of the element which is computed as,

$$2A^{(e)} = X_{x_1}(X_{y_2} - X_{y_3}) + X_{x_2}(X_{y_3} - X_{y_1}) + X_{x_3}(X_{y_1} - X_{y_2})$$

Consideration of the interpolation function at node $1^{(e)}$

$$\underset{\sim}{\varphi}_u^{(e)}(x_1,y_1) = \frac{1}{2A^{(e)}} \begin{pmatrix} X_{x_2} X_{y_3} - X_{x_3} X_{y_2} + X_{y_2} X_{x_1} - X_{y_3} X_{x_1} + X_{x_3} X_{y_1} - X_{x_2} X_{y_1} \\ X_{x_3} X_{y_1} - X_{x_1} X_{y_3} + X_{y_3} X_{x_1} - X_{y_1} X_{x_1} + X_{x_1} X_{y_1} - X_{x_3} X_{y_1} \\ X_{x_1} X_{y_2} - X_{x_2} X_{y_1} + X_{y_1} X_{x_1} - X_{y_2} X_{x_1} + X_{x_2} X_{y_1} - X_{x_1} X_{y_1} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

shows that it is "1" at the associated node and "0" at the others.
Likewise for $\underset{\sim}{\varphi}_v^{(e)}$ and the other nodes.

The strain component ε_{xx} follows form

$$\varepsilon_{xx} = \frac{\partial u}{\partial x} \qquad u(x,y,z)^{(\mathrm{e})} = \underset{\sim}{\varphi}_u^{(\mathrm{e})\mathrm{T}} \underset{\sim}{U}_u^{(\mathrm{e})} \quad \Rightarrow \quad \frac{\partial u}{\partial x} = \frac{\partial}{\partial x}(\underset{\sim}{\varphi}_u^{(\mathrm{e})\mathrm{T}})\, \underset{\sim}{U}_u^{(\mathrm{e})} =$$

$$= \frac{1}{2A^{(\mathrm{e})}} \begin{pmatrix} X_{y_2} - X_{y_3} \\ X_{y_3} - X_{y_1} \\ X_{y_1} - X_{y_2} \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} u_1^{(\mathrm{e})} \\ u_2^{(\mathrm{e})} \\ u_3^{(\mathrm{e})} \end{pmatrix} \quad \Rightarrow \quad \varepsilon_{xx}(x,y)^{(\mathrm{e})} = \text{const.}^{(\mathrm{e})}$$

The result shows, that the strain ε_{xx} (as well as ε_{yy} and ε_{xy}) are independent of $(x,y)^{(\mathrm{e})}$, i.e. it is constant in (e), which is the reason for the name *Constant Strain Triangle*.

Note: Even if the displacements are continuous at the element boundaries, the strains (and stresses) are dis-continuous. The local balance equations are not fulfilled locally, however, the nodal forces and displacements are in equilibrium. Consequently, the total energy balance in the element is fulfilled which is called the "weak form" of the solution.

6 Derivation of Stiffness Matrices

Next the element stiffness matrix is sought for which relates the nodal forces to the nodal displacements. Here, the case is treated in which the element coordinates are given in the global coordinate system (e.g. corresponding to the Constant Strain Triangle in the previous section). A more general framework based on local element considerations will be presented later.

Element Stiffness Matrix from the Potential

The total potential energy reads

$$V = \sum_{(\mathrm{e})} V^{(\mathrm{e})} \qquad , \qquad V^{(\mathrm{e})} = U^{(\mathrm{e})} + W^{(\mathrm{e})},$$

with its contributions given at element level (e).

The strain energy of the element can be expressed as,

$$U^{(\mathrm{e})} = \frac{1}{2} \int_{\mathit{Vol}^{(\mathrm{e})}} \sigma_{ij}\varepsilon_{ij} \mathrm{d}\mathit{Vol} = \frac{1}{2} \int_{\mathit{Vol}^{(\mathrm{e})}} \underset{\sim}{\sigma}^{\mathrm{T}} \underset{\sim}{\varepsilon}\, \mathrm{d}\mathit{Vol}$$

with

$$\underset{\sim}{\sigma} = \underset{\approx}{\mathbf{E}}^{(\mathrm{e})} \underset{\sim}{\varepsilon} \quad , \quad \underset{\sim}{\sigma}^{\mathrm{T}} = \underset{\sim}{\varepsilon}^{\mathrm{T}} \underset{\approx}{\mathbf{E}}^{(\mathrm{e})^{\mathrm{T}}} \quad , \quad \underset{\approx}{\mathbf{E}}^{(\mathrm{e})^{\mathrm{T}}} = \underset{\approx}{\mathbf{E}}^{(\mathrm{e})}$$

and the definition

$$\underset{\sim}{\varepsilon} = \underset{\approx}{\mathbf{D}}^{(\mathrm{e})} \underset{\sim}{U}^{(\mathrm{e})} \qquad \text{from} \quad \underset{\sim}{\varepsilon} = \underset{\approx}{\mathbf{d}} \vec{u} = \underset{\approx}{\mathbf{d}} \underset{\approx}{\boldsymbol{\varphi}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\sim}{U}^{(\mathrm{e})}$$

follows

$$U^{(\mathrm{e})} = \frac{1}{2} \int_{Vol^{(\mathrm{e})}} \underset{\sim}{U}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\mathbf{D}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\mathbf{E}}^{(\mathrm{e})} \underset{\approx}{\mathbf{D}}^{(\mathrm{e})} \underset{\sim}{U}^{(\mathrm{e})} \mathrm{d}Vol = \frac{1}{2} \underset{\sim}{U}^{(\mathrm{e})^{\mathrm{T}}} \int_{Vol^{(\mathrm{e})}} \underset{\approx}{\mathbf{D}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\mathbf{E}} \underset{\approx}{\mathbf{D}}^{(\mathrm{e})} \, \mathrm{d}Vol \, \underset{\sim}{U}^{(\mathrm{e})}$$

From solid mechanics, it is known that the energy is a quadratic form of the displacements with the stiffness

$$U^{(\mathrm{e})} = \frac{1}{2} \underset{\sim}{U}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\mathbf{K}}^{(\mathrm{e})} \underset{\sim}{U}^{(\mathrm{e})}$$

which renders the element stiffness matrix

$$\underset{\approx}{\mathbf{K}}^{(\mathrm{e})} = \int_{Vol^{(\mathrm{e})}} \underset{\approx}{\mathbf{D}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\mathbf{E}}^{(\mathrm{e})} \underset{\approx}{\mathbf{D}}^{(\mathrm{e})} \mathrm{d}Vol$$

Assembly of the Total Stiffness Matrix

Since the assembly of all elements has to solved for equilibrium, the total stiffness matrix has to be built from all element stiffness matrices.

The coincidence transformation matrix, $\underset{\approx}{\widetilde{\mathbf{T}}}^{(\mathrm{e})}$, relates the total displacement vector $\underset{\sim}{U}$ to the element displacements as,

$$\underset{\sim}{U}^{(\mathrm{e})} = \underset{\approx}{\widetilde{\mathbf{T}}}^{(\mathrm{e})} \underset{\sim}{U}$$

Then the strain energy of the element (e) reads

$$U^{(\mathrm{e})} = \frac{1}{2} \underset{\sim}{U}^{\mathrm{T}} \underset{\approx}{\widetilde{\mathbf{T}}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\mathbf{K}}^{(\mathrm{e})} \underset{\approx}{\widetilde{\mathbf{T}}}^{(\mathrm{e})} \underset{\sim}{U},$$

from which the transformation of the element stiffness matrix follows

$$\underset{\approx}{\widetilde{\mathbf{K}}}^{(\mathrm{e})} = \underset{\approx}{\widetilde{\mathbf{T}}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\mathbf{K}}^{(\mathrm{e})} \underset{\approx}{\widetilde{\mathbf{T}}}^{(\mathrm{e})}$$

with $\widetilde{\underset{\approx}{\mathbf{K}}}^{(\mathrm{e})}$ being the element stiffness in global (total) consideration.

The potential of the external forces read

$$W^{(\mathrm{e})} = \sum_{i=1,N^{(\mathrm{e})}} -F_i^{(\mathrm{e})} U_i^{(\mathrm{e})} = -\underset{\sim}{F}^{(\mathrm{e})^{\mathrm{T}}} \underset{\sim}{U}^{(\mathrm{e})} = -\underset{\sim}{F}^{(\mathrm{e})^{\mathrm{T}}} \widetilde{\underset{\approx}{\mathbf{T}}}^{(\mathrm{e})} \underset{\sim}{U}$$

and the total potential can be given as the sum over all elements

$$V = \sum_{(\mathrm{e})} \left(\frac{1}{2} \underset{\sim}{U}^{\mathrm{T}} \widetilde{\underset{\approx}{\mathbf{T}}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\mathbf{K}}^{(\mathrm{e})} \widetilde{\underset{\approx}{\mathbf{T}}}^{(\mathrm{e})} \underset{\sim}{U} - \underset{\sim}{F}^{(\mathrm{e})^{\mathrm{T}}} \widetilde{\underset{\approx}{\mathbf{T}}}^{(\mathrm{e})} \underset{\sim}{U} \right)$$

Applying the principle of minimum total potential energy, i.e. $\frac{\partial V}{\partial U_i} = 0 \ \forall i$, gives

$$\sum_{(\mathrm{e})} \widetilde{\underset{\approx}{\mathbf{T}}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\mathbf{K}}^{(\mathrm{e})} \widetilde{\underset{\approx}{\mathbf{T}}}^{(\mathrm{e})} \underset{\sim}{U} - \sum_{(\mathrm{e})} \widetilde{\underset{\approx}{\mathbf{T}}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\sim}{F}^{(\mathrm{e})} = \underset{\sim}{0},$$

which can be written as

$$\boxed{\underset{\approx}{\mathbf{K}} \underset{\sim}{U} = \underset{\sim}{F}} \quad .$$

The expression,

$$\underset{\approx}{\mathbf{K}} = \sum_{(\mathrm{e})} \widetilde{\underset{\approx}{\mathbf{T}}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\mathbf{K}}^{(\mathrm{e})} \widetilde{\underset{\approx}{\mathbf{T}}}^{(\mathrm{e})} = \sum_{(\mathrm{e})} \widetilde{\underset{\approx}{\mathbf{K}}}^{(\mathrm{e})}$$

is the total stiffness matrix, and

$$\underset{\sim}{F} = \sum_{(\mathrm{e})} \widetilde{\underset{\approx}{\mathbf{T}}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\sim}{F}^{(\mathrm{e})} = \sum_{(\mathrm{e})} \widetilde{\underset{\sim}{F}}^{(\mathrm{e})} = \sum_{(\mathrm{e})} \widetilde{\underset{\sim}{F}}^{(\mathrm{e})}_{(a)} + \underbrace{\sum_{(\mathrm{e})} \widetilde{\underset{\sim}{F}}^{(\mathrm{e})}_{(i)}}_{\underset{\sim}{0}} = \sum_{(\mathrm{e})} \widetilde{\underset{\sim}{F}}^{(\mathrm{e})}_{(a)}$$

is the total load vector.

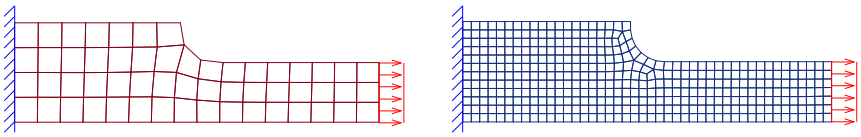
The sum over the internal forces $\underset{\sim}{F}^{(\mathrm{e})}_{(i)}$ is zero, since they are self equilibrated (if no external force is applied).

Consistent Nodal Loads

Since only nodal forces appear in $\underset{\approx}{\mathbf{K}}\,\underset{\sim}{U} = \underset{\sim}{F}$ which take action at the nodes, surface force and volume force densities must be converted to nodal forces. This must be done element wise, such that the work by the nodal loads is the same as the work by the distributed force densities, i.e. in an energetically consistent way.

7 Discretization and Accuracy

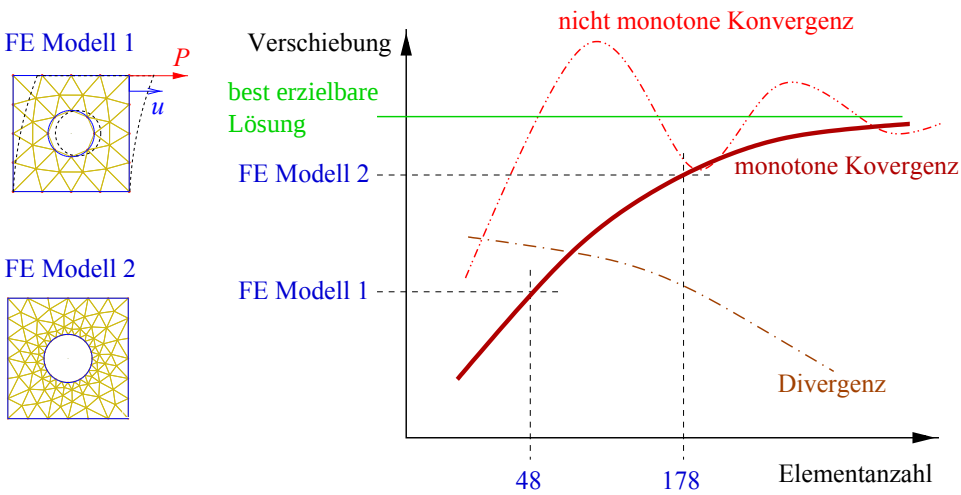
The accuracy of the solution depends (amount others) on the element formulation and on the mesh density.



The Finite Element Method also allows for selective mesh refinements in regions of interest or where high accuracy is desired.

Convergence

Monotonous convergence means, that with systematic increase of the mesh density the solution tends to the "true" response.

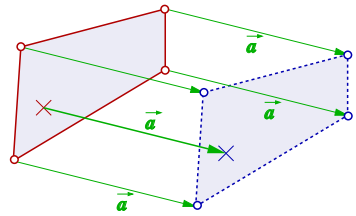


Completeness and Compatibility

Interpolation functions are required to be *complete*, i.e. to be able to describe rigid body movements and to be able describe states of constant strains.

This implies,

$$\sum_{i=1}^{N^{(e)}} \varphi_i^{(e)}(x, y, z) \equiv 1 \qquad \forall (x, y, z) \in (e)$$



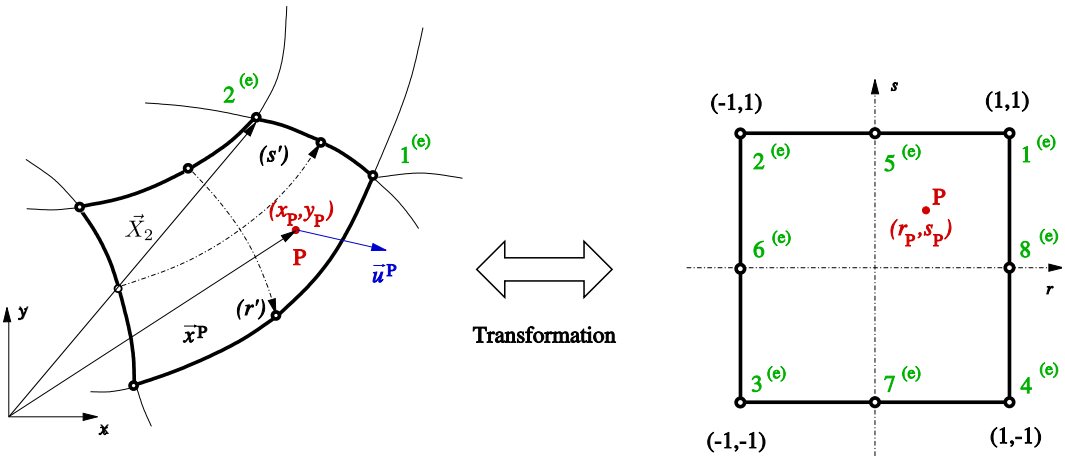
„Compatibility“ means, that the displacement field at element boundaries must be continuous to prevent overlapping and formation of gaps inside a continuum.

If adjacent elements have different interpolation functions, special measures must be take to achieve compatibility.

8 Isoparametric Finite Elements

An important step towards easy automation and efficiency has been the introduction of local reference elements.

8.1 Interpolation of Geometry and Displacement Field



Elements in the global domain can be transformed to a local reference element of the same type. For all elements of the same type, many pre-computations can be performed for the reference element once.

In the following, this is shown for a two-dimensional 8-noded quadrilateral element (with bi-quadratic interpolation functions).

$$x_{\mathrm{P}} = \sum_{j=1}^{n} f_j^{(\mathrm{e})}(r_{\mathrm{P}}, s_{\mathrm{P}})\, X_{x_j} \qquad x = \underset{\sim}{f}^{(\mathrm{e})^{\mathrm{T}}}(r,s)\, \underset{\sim}{X}_x{}^{(\mathrm{e})}$$

$$y_{\mathrm{P}} = \sum_{j=1}^{n} f_j^{(\mathrm{e})}(r_{\mathrm{P}}, s_{\mathrm{P}})\, X_{y_j} \qquad y = \underset{\sim}{f}^{(\mathrm{e})^{\mathrm{T}}}(r,s)\, \underset{\sim}{X}_y{}^{(\mathrm{e})}$$

$$\boxed{\vec{x} = \underset{\approx}{\mathbf{f}}^{(\mathrm{e})^{\mathrm{T}}}(r,s)\underset{\sim}{X}^{(\mathrm{e})}}$$

$(x_{\mathrm{P}}, y_{\mathrm{P}})$	coordinates of point $P \in$ (e)
(X_{x_j}, X_{y_j})	coordinates of node j
$\underset{\sim}{X_x}^{(\mathrm{e})}$	x-coordinates of the nodes $(X_{x_1}, X_{x_2}, \ldots, X_{x_n})^{\mathrm{T}}$
$\underset{\sim}{X_y}^{(\mathrm{e})}$	y-coordinates of the nodes $(X_{y_1}, X_{y_2}, \ldots, X_{y_n})^{\mathrm{T}}$
$\vec{x} = (x, y)^{\mathrm{T}}$	$\underset{\sim}{X}^{(\mathrm{e})} = (\underset{\sim}{X_x}^{(\mathrm{e})^{\mathrm{T}}}, \underset{\sim}{X_y}^{(\mathrm{e})^{\mathrm{T}}})^{\mathrm{T}}$

The interpolation of the displacements are given by

$$u^{\mathrm{P}} = \sum_{j=1}^{n} N_j^{(\mathrm{e})}(r_{\mathrm{P}}, s_{\mathrm{P}})\, U_{uj}^{(\mathrm{e})} \qquad u = \underset{\sim}{N}^{(\mathrm{e})^{\mathrm{T}}}(r, s)\, \underset{\sim}{U_u}^{(\mathrm{e})} \qquad \underset{\sim}{\varphi_u}^{\mathrm{T}} \longrightarrow \underset{\sim}{N_u}^{\mathrm{T}} \longrightarrow \underset{\sim}{N}^{\mathrm{T}}$$

$$v^{\mathrm{P}} = \sum_{j=1}^{n} N_j^{(\mathrm{e})}(r_{\mathrm{P}}, s_{\mathrm{P}})\, U_{vj}^{(\mathrm{e})} \qquad v = \underset{\sim}{N}^{(\mathrm{e})^{\mathrm{T}}}(r, s)\, \underset{\sim}{U_v}^{(\mathrm{e})} \qquad \underset{\sim}{N_v} = \underset{\sim}{N_u} = \underset{\sim}{N}$$

Note, that the interpolation functions $f_j^{(\mathrm{e})}(r, s)$ and $N_j{}^{(\mathrm{e})}(r, s)$, are defined in the local (r, s) space. All elements of the same type have identical interpolation functions, and the superscript (e) can be omitted, giving,

$$\boxed{\vec{u} = \underset{\approx}{\mathbf{N}}^{\mathrm{T}}(r, s)\, \underset{\sim}{U}^{(\mathrm{e})}}$$

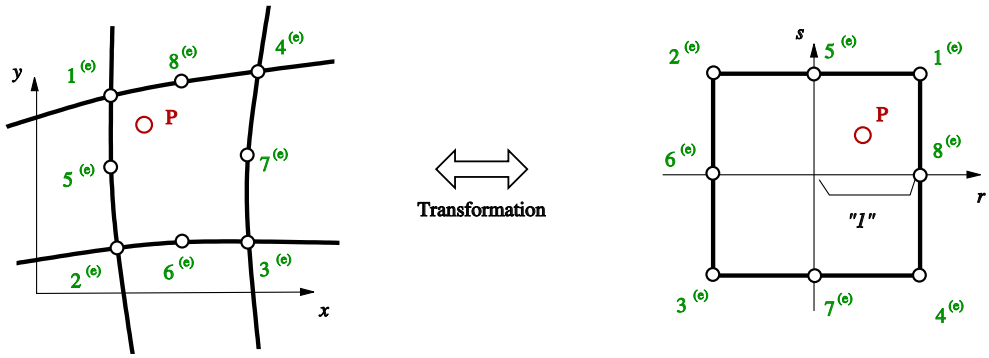
For the case of isoparametric elements $f_j(r, s) \equiv N_j(r, s) \qquad j = 1, 2 \ldots n$

Again, completeness is desired;

$$\begin{aligned} f_j\,(r_i, s_i, (t_i)) &= \delta_{ij} \\ N_j\,(r_i, s_i, (t_i)) &= \delta_{ij} \end{aligned} \qquad i, j \ldots \text{node number at reference elements}$$

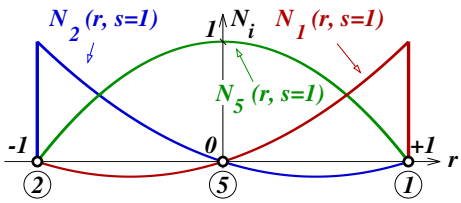
Examples

Isoparametric quadrilateral element with 8 nodes and bi-quadratic interpolation



$$
\begin{aligned}
N_i(r,s) &= \frac{1}{4}(1+rr_i)(1+ss_i)(rr_i+ss_i-1) \quad \text{for } i=1,2,3,4 \\
N_i(r,s) &= \frac{1}{2}(1-r^2)(1+ss_i) \quad \text{for } i=5,7 \\
N_i(r,s) &= \frac{1}{2}(1+rr_i)(1-s^2) \quad \text{for } i=6,8
\end{aligned}
$$

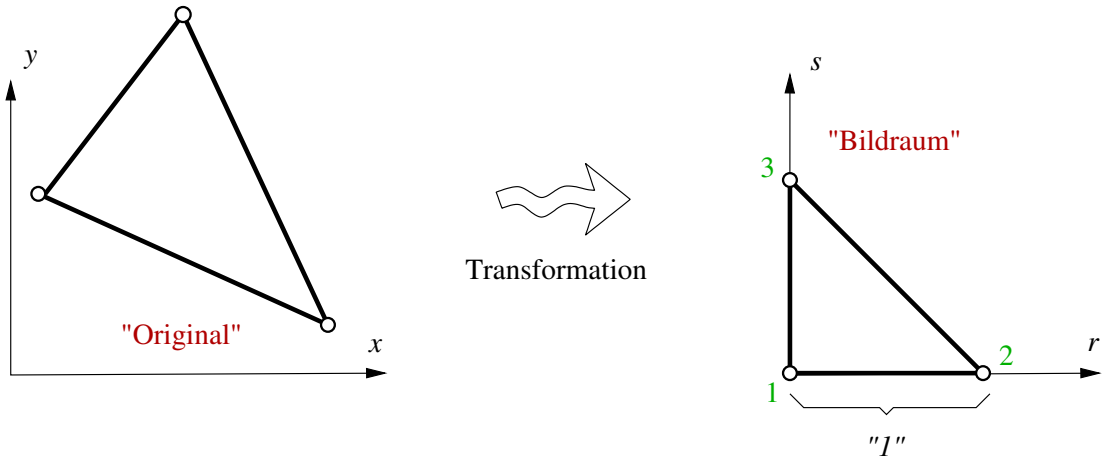
discussion for $s=1$



check for completeness, e.g. rigid body translation

$$\sum_{i=1}^{8} N_i(r,s) \equiv 1$$

Isoparametric Constant Strain Triangle



$$\underset{\sim}{N}(r,s) = \begin{pmatrix} 1-r-s \\ r \\ s \end{pmatrix}$$

8.2 The "Jacobi Matrix" of the Element

The computation of the strains requires the partial derivatives,

$$\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \ldots, \frac{\partial v}{\partial x}, \ldots,$$

$$\text{z.B.} \quad \varepsilon_{xx} = \frac{\partial u}{\partial x}, \ldots ,$$

but they cannot be given straightforward since, $\vec{u} = \underset{\approx}{\mathbf{N}}^{\mathrm{T}}(r,s,t)\, \underset{\sim}{U}^{(\mathrm{e})}$.

For two-dimensional cases we can write,

$$\left.\begin{aligned} \frac{\partial}{\partial r} &= \frac{\partial}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial}{\partial y}\frac{\partial y}{\partial r} \\ \frac{\partial}{\partial s} &= \frac{\partial}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial}{\partial y}\frac{\partial y}{\partial s} \end{aligned}\right\} \Leftrightarrow \begin{pmatrix} \frac{\partial}{\partial r} \\ \frac{\partial}{\partial s} \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \end{pmatrix} \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix} \Leftrightarrow \begin{pmatrix} \frac{\partial}{\partial r} \\ \frac{\partial}{\partial s} \end{pmatrix} = \underset{\approx}{\mathbf{J}}^{(\mathrm{e})}(r,s) \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix}$$

$$\boxed{\underset{\sim}{\partial}_{rs} = \underset{\approx}{\mathbf{J}}^{(\mathrm{e})}(r,s)\ \underset{\sim}{\partial}_{xy}}$$

with $\underset{\approx}{\mathbf{J}}^{(\mathrm{e})}$ being the "Jacobi Matrix" of element (e).

Inversion gives the desired derivatives,

$$\boxed{\underset{\sim}{\partial}_{xy} = \underset{\approx}{\mathbf{J}}^{(\mathrm{e})^{-1}}(r,s)\ \underset{\sim}{\partial}_{rs}}$$

$$\begin{pmatrix} \frac{\partial()}{\partial x} \\ \frac{\partial()}{\partial y} \end{pmatrix} = \frac{1}{\det \underset{\approx}{\mathbf{J}}^{(\mathrm{e})}} \begin{pmatrix} \frac{\partial y}{\partial s} & -\frac{\partial y}{\partial r} \\ -\frac{\partial x}{\partial s} & \frac{\partial x}{\partial r} \end{pmatrix} \begin{pmatrix} \frac{\partial()}{\partial r} \\ \frac{\partial()}{\partial s} \end{pmatrix}$$

Note that, because of $\vec{x} = \underset{\approx}{\mathbf{N}}^{\mathrm{T}}(r,s,t)\,\underset{\sim}{X}$, the Jacobi Matrix $\underset{\approx}{\mathbf{J}}^{(\mathrm{e})}$ depends only on the global coordinates $\underset{\sim}{X}^{(\mathrm{e})}$ of element (e).

Inversion of the Jacobi matrix is possible only if it is non-singular, i.e. $\underset{\approx}{\mathbf{J}}^{(\mathrm{e})}(r,s,t) \neq 0\ \forall(r,s,t)$. This gives restrictions to element shapes.

The volume of the element is given by $\mathrm{d}Vol = \mathrm{d}x\mathrm{d}y\mathrm{d}z = \det\underset{\approx}{\mathbf{J}}^{(\mathrm{e})}\,\mathrm{d}r\,\mathrm{d}s\,\mathrm{d}t$.

8.3 The Element Stiffness Matrix of Isoparametric Elements

The element stiffness matrix is computed as

$$\underset{\approx}{\mathbf{K}}^{(\mathrm{e})} = \int_{Vol^{(\mathrm{e})}} \underset{\approx}{\mathbf{D}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\mathbf{E}}^{(\mathrm{e})} \underset{\approx}{\mathbf{D}}^{(\mathrm{e})} \mathrm{d}Vol \qquad \text{with} \quad \underset{\approx}{\mathbf{D}}^{(\mathrm{e})} = \underset{\approx}{\mathbf{d}}^{(\mathrm{e})} \underset{\approx}{\mathbf{N}}^{\mathrm{T}} \quad .$$

For a two-dimensional element (for example) this reads,

$$\underset{\approx}{\mathbf{D}}^{(\mathrm{e})} = \begin{pmatrix} \frac{\partial}{\partial x}\underset{\sim}{N}^{\mathrm{T}} & \underset{\sim}{0}^{\mathrm{T}} \\ \underset{\sim}{0}^{\mathrm{T}} & \frac{\partial}{\partial y}\underset{\sim}{N}^{\mathrm{T}} \\ \frac{\partial}{\partial y}\underset{\sim}{N}^{\mathrm{T}} & \frac{\partial}{\partial x}\underset{\sim}{N}^{\mathrm{T}} \end{pmatrix}$$

The (1,1)-element of $\underset{\approx}{\mathbf{D}}^{(\mathrm{e})}$ can be expressed as

$$\frac{\partial}{\partial x}\underset{\sim}{N}^{\mathrm{T}} = \frac{1}{\det\underset{\approx}{\mathbf{J}}^{(\mathrm{e})}} \left(\frac{\partial y}{\partial s}\frac{\partial}{\partial r}\underset{\sim}{N}^{\mathrm{T}} - \frac{\partial y}{\partial r}\frac{\partial}{\partial s}\underset{\sim}{N}^{\mathrm{T}} \right)$$

Introducing the short hand notation, $(\quad)' := \frac{\partial(\,)}{\partial r}$, $(\dot{\quad}) := \frac{\partial(\,)}{\partial s}$ the $\underset{\approx}{\mathbf{D}}^{(\mathrm{e})}$ matrix can be written as

$$\underset{\approx}{\mathbf{D}}^{(\mathrm{e})} = \frac{1}{\det \underset{\approx}{\mathbf{J}}^{(\mathrm{e})}} \underbrace{\begin{pmatrix} \dot{y}(\underset{\sim}{N}^{\mathrm{T}})' - y'(\dot{\underset{\sim}{N}}^{\mathrm{T}}) & \underset{\sim}{0}^{\mathrm{T}} \\ \underset{\sim}{0}^{\mathrm{T}} & -\dot{x}(\underset{\sim}{N}^{\mathrm{T}})' + x'(\dot{\underset{\sim}{N}}^{\mathrm{T}}) \\ -\dot{x}(\underset{\sim}{N}^{\mathrm{T}})' + x'(\dot{\underset{\sim}{N}}^{\mathrm{T}}) & \dot{y}(\underset{\sim}{N}^{\mathrm{T}})' - y'(\dot{\underset{\sim}{N}}^{\mathrm{T}}) \end{pmatrix}}_{\underset{\approx}{\mathbf{Q}}^{(\mathrm{e})}} = \frac{1}{\det \underset{\approx}{\mathbf{J}}^{(\mathrm{e})}} \underset{\approx}{\mathbf{Q}}^{(\mathrm{e})} \quad .$$

For isoparametric elements further holds,

$$x = \underset{\sim}{N}^{\mathrm{T}} \underset{\sim}{X}_x{}^{(\mathrm{e})} \quad , \quad y = \underset{\sim}{N}^{\mathrm{T}} \underset{\sim}{X}_y{}^{(\mathrm{e})}$$

$$\Longrightarrow x' = \frac{\partial x}{\partial r} = \frac{\partial}{\partial r} \underset{\sim}{N}^{\mathrm{T}} \underset{\sim}{X}_x{}^{(\mathrm{e})} = \left(\underset{\sim}{N}^{\mathrm{T}}\right)' \underset{\sim}{X}_x{}^{(\mathrm{e})},$$

$$\dot{y} = \frac{\partial y}{\partial s} = \frac{\partial}{\partial s} \underset{\sim}{N}^{\mathrm{T}} \underset{\sim}{X}_y{}^{(\mathrm{e})} = \left(\dot{\underset{\sim}{N}}^{\mathrm{T}}\right) \underset{\sim}{X}_y{}^{(\mathrm{e})}$$

$$(\text{ likewise } \dot{x}, y' \,)$$

Derivatives are required only for $\underset{\sim}{N}^{(\mathrm{e})^{\mathrm{T}}}$, which are equal for all elements of the same type. The global coordinates of the elements, $\underset{\sim}{X}^{(\mathrm{e})}$, appear as simple coefficients.

Applying the short hand notation gives,

$$\underset{\approx}{\mathbf{Q}}^{(\mathrm{e})} = \begin{pmatrix} [(\dot{\underset{\sim}{N}}^{\mathrm{T}})\underset{\sim}{X}_y](\underset{\sim}{N}^{\mathrm{T}})' - [(\underset{\sim}{N}^{\mathrm{T}})'\underset{\sim}{X}_y](\dot{\underset{\sim}{N}}^{\mathrm{T}}) & \underset{\sim}{0}^{\mathrm{T}} \\ \underset{\sim}{0}^{\mathrm{T}} & [(\underset{\sim}{N}^{\mathrm{T}})'\underset{\sim}{X}_x](\dot{\underset{\sim}{N}}^{\mathrm{T}}) - [(\dot{\underset{\sim}{N}}^{\mathrm{T}})\underset{\sim}{X}_x](\underset{\sim}{N}^{\mathrm{T}})' \\ [(\underset{\sim}{N}^{\mathrm{T}})'\underset{\sim}{X}_x](\dot{\underset{\sim}{N}}^{\mathrm{T}}) - [(\dot{\underset{\sim}{N}}^{\mathrm{T}})\underset{\sim}{X}_x](\underset{\sim}{N}^{\mathrm{T}})' & [(\dot{\underset{\sim}{N}}^{\mathrm{T}})\underset{\sim}{X}_y](\underset{\sim}{N}^{\mathrm{T}})' - [(\underset{\sim}{N}^{\mathrm{T}})'\underset{\sim}{X}_y](\dot{\underset{\sim}{N}}^{\mathrm{T}}) \end{pmatrix}$$

and the $\underset{\approx}{\mathbf{D}}^{(\mathrm{e})}$ matrix is a function of the local coordinates, $\underset{\approx}{\mathbf{D}}^{(\mathrm{e})}(r,s)$, of the reference element (and the coefficients $\underset{\sim}{X}^{(\mathrm{e})}$).

Now the element stiffness matrix can be written as,

$$\begin{aligned} \int_{Vol^{(\mathrm{e})}} \underset{\approx}{\mathbf{D}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\mathbf{E}}^{(\mathrm{e})} \underset{\approx}{\mathbf{D}}^{(\mathrm{e})} \mathrm{d}Vol \quad &\rightarrow \quad \int_{-1}^{+1} \int_{-1}^{+1} \underset{\approx}{\mathbf{D}}^{(\mathrm{e})^{\mathrm{T}}}(r,s) \underset{\approx}{\mathbf{E}}^{(\mathrm{e})} \underset{\approx}{\mathbf{D}}^{(\mathrm{e})}(r,s) h \det \underset{\approx}{\mathbf{J}}^{(\mathrm{e})} \mathrm{d}r\mathrm{d}s \\ &= \quad \int_{-1}^{+1} \int_{-1}^{+1} \frac{1}{\det \underset{\approx}{\mathbf{J}}^{(\mathrm{e})}} \underset{\approx}{\mathbf{Q}}^{(\mathrm{e})^{\mathrm{T}}} \underset{\approx}{\mathbf{E}}^{(\mathrm{e})} \underset{\approx}{\mathbf{Q}}^{(\mathrm{e})} h \mathrm{d}r\mathrm{d}s \end{aligned}$$

with the element thickness h.

The integration is typically performed by Gauss quadrature, such as

$$\int_{\Omega} \Psi(r,s,t)\,\mathrm{d}r\,\mathrm{d}s\,\mathrm{d}t = \sum_{j=1}^{\mathcal{O}} \Psi(r_j,s_j,t_j)\bar{w}_j$$

A further advantage of isoparametric elements is, that the integration involves the function $\Psi(r,s,t)$, given in local coordinates. Also this integration needs to be computed only once for the reference element.

9 Closing Remarks

Various extensions are readily developed and are widely used for Finite Element Method simulations. Among them are

- Transient problems with time-varying loads and responses
- "Dynamic" problems for which first and second time derivatives of the displacements enter the equations
- Eigenvalue problems to study buckling and resonance
- Non-linear formulations of the strain–displacement relation, the material response, etc.
- Dissipative and load history dependent mechanisms
- Multi-field and multi-physics coupled problems
- ...

Advantages of the Finite Element Method from the point of view of an Finite Element textbook as listed in [2].

- It can deal simply with nonhomogeneous and anisotropic situations (particularly when the direction of anisotropy is variable).
- The elements can be graded in shape and size to follow arbitrary boundaries and to allow for regions of rapid variation of the function sought, thus controlling the errors in a most efficient way ...
- Specified gradient or "radiation" boundary conditions are introduced naturally and with a better accuracy than in standard finite difference procedures.
- Higher order elements ... can be readily used to improve accuracy without complicating boundary conditions — a difficulty always arising with finite difference approximations of a higher order.
- Finally, but of considerable importance in the computer age, standard programs may be used for assembly and solution.

Literatur

[1] M. Kaltenbacher, Numerical simulation of mechatronic sensors and actuators, Vol. 2, Springer, 2007.

[2] O. C. Zienkiewicz, R. L. Taylor, J. Zhu, The Finite Element Method: Its Basis and Fundamentals, Butterworth-Heinemann, 2013.

[3] K.-J. Bathe, Finite element procedures, Klaus-Jurgen Bathe, 2006.

[4] O. Zienkiewicz, R. Taylor, J. Zhu, Chapter 5 - field problems: A multidimensional finite element method, in: O. Zienkiewicz, R. Taylor, J. Zhu (Eds.), The Finite Element Method: its Basis and Fundamentals (Seventh Edition), seventh edition Edition, Butterworth-Heinemann, Oxford, 2013, pp. 115 – 149.
URL `http://www.sciencedirect.com/science/article/pii/B9781856176330000058`