Table of Contents

Vectors and Matrices

- Classical definition
  - A vector is a quantity with **magnitude** and **direction**
  - Example: force, speed, etc
- For us it is enough to view vectors as collections of numbers
  - The population in every country could be divided into age brackets
- Vectors are usually denoted by bold letters, and their elements are given with a subscript $x = (x_1, ..., x_n)$
- Similarly **matrices** are simply **two dimensional arrays** of numbers

Vectors and Matrices (2)

- The *size of a matrix* is usually given as the number of rows times the number of columns
- If the number of rows equals the number of columns, the matrix is square
- The elements of a matrix are referred to with a lower case letter with two subscripts:
  - The first of which denotes the row of the element and the second the column
- Note that vectors can be thought of as matrices with one dimension equal to 1
- The special matrix where all elements along the main diagonal are equal to 1 and all other elements are 0 is called the identity matrix

Algebraic operations with vectors and matrices

- Addition and subtraction are defined element-wise for both vectors and matrices
- For multiplication there are several possibilities:
  - The following is known as the *inner product / dot product*

$$a \cdot b = (a_1, ..., a_n) \begin{pmatrix} b_1 \\ ... \\ b_n \end{pmatrix} = \sum_{j=1}^{n} a_j b_j$$

- The length of a vector is given by $\sqrt{a \cdot a}$
- Two matrices **A** and **B** can be multiplied only if their sizes are compatible

Algebraic operations with vectors and matrices (2)

200

- The number of columns in **A** must equal the number of rows in **B**
- If their product matrix is called **C**, the elements of **C** are computed as $$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$
- If the size of **A** is ***p x n*** and the size of **B** is ***n x q***, **C** will be of size ***p x q***
- This definition of multiplication between matrices makes writing systems of linear equations very easy
- New ways of solving systems of linear equations efficiently is one of the most intensive fields in numerical analysis
- There are specialized solvers for almost any special type of system of equations

Algebraic operations with vectors and matrices (3)

201

- For matrices, multiplication is **not commutative**,
  - It is not true in general that $AB = BA$ for arbitrary A & B
- If A is a square matrix and there is another matrix B for which $AB = BA = I$
  - I is the identity matrix
  - A is **invertible** and B is the **inverse** of A, denoted by $B = A^{-1}$
  - If A is invertible, the solution of the linear system of equations $Ax = y$ is $x = A^{-1}y$
- The division operation for matrices is defined as multiplication with an inverse
  - There is a difference between $A^{-1}B \neq BA^{-1}$
- For square matrices one can define
  - The **trace**
  - The **determinant**

Methods for Solving Linear Equations

- Direct Methods
  - Gaussian elimination
  - More efficient versions are based on various *decompositions* (e.g., $A = LU$ or $A = LDU$)
  - The unknowns are solved one at a time
    - The process must be finished in order to have some realistic value for every unknown
  - Accurate in principle – may be too slow for very large systems
  - Often demands lots of memory

Methods for Solving Linear Equations (2)

- Iterative Methods $x_{k+1} = Gx_k$
  - Basic idea: guess some solution and improve it gradually
  - Some methods: Gauss-Seidel, SOR, Conjugate Gradient (CG), GMRES, ...
  - Simple methods may be made faster by using *preconditioners*
  - All unknowns are solved *simultaneously* little by little
    - It is possible to interrupt the solution process whenever sufficient accuracy is achieved
    - In principle, to have an accurate solution a large number of iterations is needed
    - In practice, the iterations can converge remarkably fast
    - Efficient memory usage

Linear Transformations

- In one dimension, a linear transformation A means simply multiplication with a constant:

$$y = A(x) = Ax = ax$$

- In two dimensions, a linear transformation is a function $A: \Re^2 \Rightarrow \Re^2$ which is linear with respect to both arguments:

$$y_1 = a_{11}x_1 + a_{12}x_2$$

$$y_2 = a_{21}x_1 + a_{22}x_2$$

- In matrix notation $y = Ax$

Linear Transformations (2)

- **Geometrically** a linear transformation:
  - Stretches or shrinks distances between points
  - Rotates points about the origin
  - Flips the orientation from a right handed system to a left handed system
- In higher dimensions definition is analogous & the action is viewed similarly through geometry
- The **norm** of a linear transformation tells what is the maximum factor by which a linear transformation can stretch a vector

206

Eigenvalues and Eigenvectors

- In general, a linear transformation rotates a given point about the origin
- It may happen that the points lying on some particular lines remain on these lines
  - They may move farther from or closer to the origin
- This can be expressed as $Ax = \lambda x$
- Where λ tells how much is the change in distance with respect to the origin
  - $|\lambda| < 1$ means the points moves closer and it is called an **eigenvalue** of A
  - The corresponding vector x which determines the **neutral** direction is called an **eigenvector**

207

Eigenvalues and Eigenvectors (2)

- A square matrix of size **n x n** can have at most **n** different eigenvalues and eigenvectors
- If the eigenvectors form an n-dimensional basis the matrix is said to be **diagonalizable**
  - Any vector in $\Re^n$ can be expressed as a linear combination of the eigenvectors
- In low-dimensional cases the eigenvalues can be calculated by hand but usually numerical methods are the only way to find them
- If $Ax = \lambda x \Rightarrow (A - \lambda I)x = 0$ and this has a non-trivial solution $(\neq 0)$ only if $\det(A - \lambda I) \neq 0$

Eigenvalues and Eigenvectors (3)

- Given $Ax = \begin{pmatrix} 4 & 3 \\ -2 & -1 \end{pmatrix} \Rightarrow \det(A - \lambda I)$

$$= (4-\lambda)(-1-\lambda) - (-2)(-3) = \lambda^2 - 3\lambda + 2$$

- The zeros of the right hand side of the last equation are $\lambda_1 = 1, \lambda_2 = 2$ Eigenvalues for A
- The corresponding eigenvectors $(1,-1)$ and $(3,-2)$
- Eigenvalues of a real symmetric matrix are real
- $tr(A) = \sum_j \lambda_j$
- $\det(A) = \prod_j \lambda_j$

Analysis of systems of differential equations

- A linear autonomous homogeneous system

$$x'(t) = Ax(t), \quad x(0) = x_0$$

- If $Av = \lambda v$ then $ce^{\lambda t}v$ is a solution of this system
- If A is diagonalizable, the eigenvalues λ_j and eigenvectors v_j of A give all possible solutions as linear combinations

$$x(t) = c_1 e^{\lambda_1 t} v_1 + c_2 e^{\lambda_2 t} v_2 + \ldots + c_n e^{\lambda_n t} v_n$$

- The solution compatible with the initial condition is found once the constants c_j are determined from the equation $c_1 v_1 + c_2 v_2 + \ldots + c_n v_n = x_0$
- If A is not diagonalizable the solution is complicated

Eigenvalues Example

- If $A=\begin{pmatrix}1 & 2\\ 0 & 2\end{pmatrix}\Rightarrow \lambda_1=1,\lambda_2=2$ & the eigenvectors

$$v_1=\begin{pmatrix}1\\0\end{pmatrix}, v_2=\begin{pmatrix}1\\1\end{pmatrix}$$

- Thus, any solution must be of the form

$$x(t)=c_1e^{\lambda_1 t}v_1+c_2e^{\lambda_2 t}v_2$$
$$=c_1e^{t}\begin{pmatrix}1\\0\end{pmatrix}+c_2e^{2t}\begin{pmatrix}1\\1\end{pmatrix}=\begin{pmatrix}c_1e^{t}+c_2e^{2t}\\ c_2e^{2t}\end{pmatrix}$$

Eigenvalues vs. Equilibrium

- The equilibrium of the system $x'(t)=Ax(t)$ at the origin is **asymptotically stable** if and only if for all eigenvalues of A the $\operatorname{Re}\lambda<0$
- If eigenvalues are real and positive small perturbations always drive the system away from the origin – **source eq**
- If eigenvalues are real and negative – **sink eq**
- If the eigenvalues are real with different signs, the equilibrium is a **saddle point**
- If eigenvalues are complex numbers the solution curves act like spirals around the equilibrium

212

Linear algebra subroutine libraries

- A lot of work is done in the linear algebra part
- It is important that the linear algebra is done as effectively as possible
- **BLAS** (Basic Linear Algebra Subprograms) library contains routines for the basic tasks of linear algebra:
  - BLAS 1 routines: vector-vector operations
  - BLAS 2 routines: matrix-vector operations
  - BLAS 3 routines: matrix-matrix operations

213

Linear algebra subroutine libraries

- **LAPACK** (Linear Algebra PACKage) is a library for
  - Efficient algorithms and BLAS routines to solve systems of linear equations
  - Find eigenvalues
  - Solve least squares problems
- Both BLAS and LAPACK are optimized by the vendors for the specific computer architectures to provide the best performance
- BLAS and LAPACK are superior to user written routines except for small problems
- The cs.pub.ro cluster has installed the **ATLAS** BLAS & **MKL** (BLAS/LAPACK/ScaLAPACK/FFT) libraries

Table of Contents

Terminology

- **Chaos**: a well-defined mathematical property of solutions of some nonlinear differential equations
  - Comparable with randomness
- **Attractor**: A set towards which the solutions eventually drift. Various types available:
  - A fixed point (equilibrium)
  - A limit cycle
  - Torus (doughnut)
  - Strange = 'not one of the above three'
- **Bifurcation**: if the system splits into two branches
  - Usually happens when a control parameter reaches a critical value

How to recognize chaos?

- The difference between **randomness** and **chaos** is not always clear (sometimes a philosophical matter)
- Chaos in the context of dynamical systems means the superficially random behavior of a deterministic system
- If the laws governing the system are known
  - Future states of the system can be predicted for arbitrarily long time periods
  - If the present state is known arbitrarily accurately
- Purely random systems cannot be predicted

How to recognize chaos? (2)

- Signatures of chaos:
  - Patterns in power spectral analysis of time
  - Series
  - Structures in phase space (attractors, projections of attractors)
  - Fractal dimensions of phase space structures
  - Sensitivity to initial conditions (clear sign of chaos)
  - Controllability of time series
- None of these methods is absolutely foolproof

When to expect chaos?

- Some conditions under which chaotic behavior can occur:
  - Nonlinear systems – e.g. with strong feedback
  - Time delays in continuous systems
    - Finite speed of information
    - Different age groups in population models
  - Spatial discretization
  - External forcing functions

Table of Contents

220

Program Development – Prototyping

- How to solve a computational problem?
  - Identify the problem
  - Formulate the problem as a mathematical model
  - Choose a suitable computational method for solving the model
  - Implement the method with the right tools
  - Check the results

221

Program Development - Prototyping (2)

- To understand the problem and the model – start with a simplified version
  - Easier to implement
  - Easier to test
  - Computations take less time
- Move gradually towards the complete model
- Balance the implementation and total computation time

Program Development - Prototyping (3)

- If the problem must be solved only once
  - Performance is probably not critical unless the problem is really big
  - Choose the easiest possible tool to minimize the implementation time
- If the problem needs to be solved repeatedly – various parameter values
  - More time can be spend in optimizing the implementation & parallelizing it
  - Attention should be paid to performance

Tools and implementation

- One may use a tool for the prototype problem that is
  - Easy to use
  - Interactive
  - Less efficient
- Good candidates for prototyping
  - Matlab (numerical)
  - Mathematica or Maple (symbolic)
- The final implementation should be as efficient as possible

Tools and implementation (2)

- For final implementation and production runs the best choices are:
  - Compiled programming languages:
    - Fortran 77 & 90
    - C/C++
  - Subroutine libraries
    - NAG
    - IMSL
    - BLAS
    - LAPACK
    - Matlab

Case study: The Problem

- We consider chemical reactions in silicon carbide (SiC) production
- The reacting species may be gases, liquids, and solids
- The amounts of species may vary greatly
- We are looking for the equilibrium state
- The equilibrium state is the minimum of the Gibbs free energy under mass conservation constraints

Case study: Mathematical Model

- The Gibbs free energy **G** is given by $G = \sum_{i=1}^{N} n_i \mu_i$
- With
  - **N** is the number of species present
  - n_i is the molar amount of species **i**
  - μ_i is the corresponding chemical potential
- The Gibbs free energy is minimized under the requirement that the amounts of basic elements must be conserved:
- Where $\sum_{i}^{N} a_{ki} n_i = b_k; \quad k = 1,2,...,M$
  - a_{ki} is the subscript of the kth element in the molecular formula of species i,
  - b_k is the initial amount of element **k**
  - **M** is the number of elements
- This is a constrained optimization problem

Case study: Simplified Version

- We consider ideal gases at constant pressure conditions
  - Then the chemical potentials μ_i can be expressed as:
  - With $\mu_i = \mu_i^0 + RT \log p_i$
    - μ_i^0 is the standard chemical potential of species **i**
    - p_i is the partial pressure
    - T is the temperature
    - R is the universal gas constant
  - For ideal gases the partial pressures are given by:
  - Where $p_i = \frac{n_i}{n_t} p_t$
    - p_t is the total pressure and
    - $n_t = \sum n_i$ is the total molar amount of the species

Case study: Simplified Version (2)

- We thus obtain $\mu_i = \mu_i^* + RT \log \frac{n_i}{n_t}$
- Where $\mu_i^* = \mu_i^0 + RT \log p_t$
- We consider only the most important reactants and ignore the ones with low concentrations
  - Decrease the number of unknowns
    - Solve the problem faster
    - Check the results easier
  - Neglecting minor species to avoid numerical problems
- Tools for solving the simplified version
  - Matlab Optimization Toolbox
  - GAMS – General Algebraic Modeling System

Case study: Complete Problem

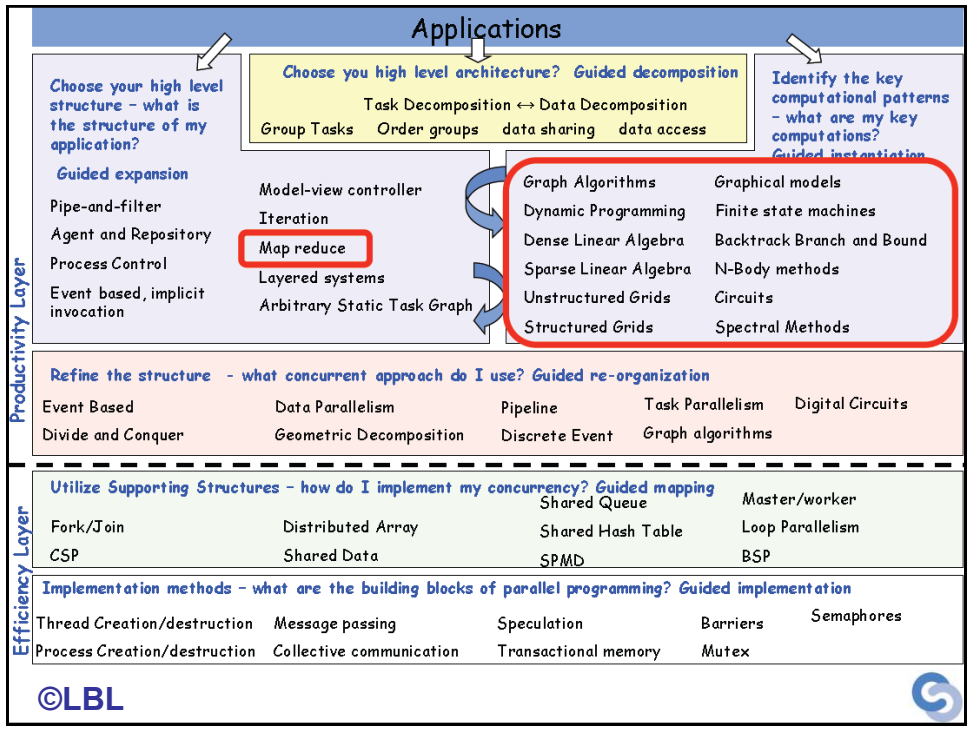
- Include liquids and solids
  - Their chemical potentials require new equations
- Include all reactants
  - Increases the number of unknowns
- The low concentrations of minor species may cause numerical problems
  - Hand tuned solver may be needed: Fortran 90 + NAG/MKL
- Choose the suitable numerical methods
  - Method of projected gradient
  - Method of augmented Lagrangian
  - Sequential Quadratic Programming (SQP)

Computational Patterns

- Productive parallel computing depends on recognizing and exploiting known patterns
  - Design, computational, and mathematical
- Optimizing (some of) the following 7 Motifs
  - To minimize time, minimize communication
- Between levels of the memory hierarchy
- Between processors over a network
  - Autotuning to explore large design spaces
- Too hard (tedious) to write by hand, let machine do it

"7 Motifs" of High Performance Computing

- Phil Colella (LBL) identified 7 kernels of which most simulation and data-analysis programs are composed:
  1. Dense Linear Algebra
     - Ex: Solve Ax=b or Ax = λx where A is a dense matrix
  2. Sparse Linear Algebra
     - Ex: Solve Ax=b or Ax = λx where A is a sparse matrix (mostly zero)
  3. Operations on Structured Grids
     - Ex: Anew(i,j) = 4*A(i,j) – A(i-1,j) – A(i+1,j) – A(i,j-1) – A(i,j+1)
  4. Operations on Unstructured Grids
     - Ex: Similar, but list of neighbors varies from entry to entry
  5. Spectral Methods
     - Ex: Fast Fourier Transform (FFT)
  6. Particle Methods
     - Ex: Compute electrostatic forces on n par2cles
  7. Monte Carlo
     - Ex: Many independent simulations using different inputs


Applications
Choose your high level structure – what is the structure of my application?
Guided expansion
Pipe-and-filter
Agent and Repository
Process Control
Event based, implicit invocation
Choose you high level architecture? Guided decomposition
Task Decomposition ↔ Data Decomposition
Group Tasks
Order groups
data sharing
data access
Model-view controller
Iteration
Map reduce
Layered systems
Arbitrary Static Task Graph
Identify the key computational patterns – what are my key computations?
Graph Algorithms
Dynamic Programming
Dense Linear Algebra
Sparse Linear Algebra
Unstructured Grids
Structured Grids
Graphical models
Finite state machines
Backtrack Branch and Bound
N-Body methods
Circuits
Spectral Methods
Productivity Layer
Refine the structure - what concurrent approach do I use? Guided re-organization
Event Based
Divide and Conquer
Data Parallelism
Geometric Decomposition
Pipeline
Discrete Event
Task Parallelism
Graph algorithms
Digital Circuits
Efficiency Layer
Utilize Supporting Structures – how do I implement my concurrency? Guided mapping
Fork/Join
CSP
Distributed Array
Shared Data
Shared Queue
Shared Hash Table
SPMD
Master/worker
Loop Parallelism
BSP
Implementation methods – what are the building blocks of parallel programming? Guided implementation
Thread Creation/destruction
Process Creation/destruction
Message passing
Collective communication
Speculation
Transactional memory
Barriers
Mutex
Semaphores
©LBL