

Table of Contents

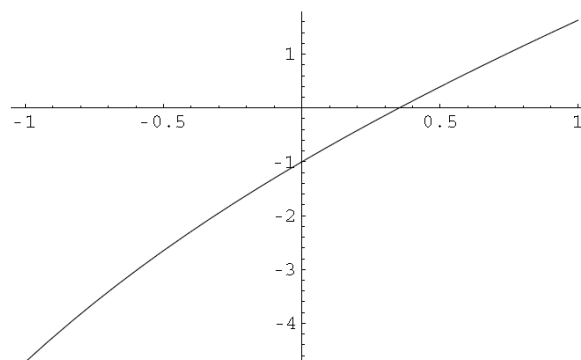
- Motivation & Trends in HPC
- Mathematical Modeling
- **Numerical Methods used in HPSC**
 - Automatic Differentiation
 - Systems of Differential Equations: ODEs & PDEs
 - Solving Optimization Problems
 - **Solving Nonlinear Equations**
 - Basic Linear Algebra, Eigenvalues and Eigenvectors
 - Chaotic systems
- HPSC Program Development/Enhancement: from Prototype to Production
- Visualization, Debugging, Profiling, Performance Analysis & Optimization



152

Nonlinear Equation Example

- The problem $f(x) = -e^{-x} + 2x = 0$



- The solution: $x = 0.351734$



153

Nonlinear equations

- Find the roots of the system of equations:

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^n, \quad f(x) = 0$$

- Numerical methods find an approximate solution point $x^* \in \mathbb{R}^n$
- If $n > 1$, we have the system of equations:

$$\begin{cases} f_1(x_1, \dots, x_n) = 0 \\ f_2(x_1, \dots, x_n) = 0 \\ \dots \\ f_n(x_1, \dots, x_n) = 0 \end{cases}$$

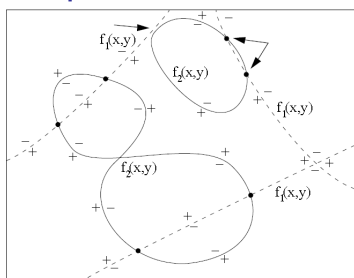
- We have a linear system of equations, if $f(x) = Ax - b$ where A is a matrix and b is a vector in \mathbb{R}^n



154

Finding the solution

- Solving a system of equations is much harder than solving a single equation!



- Numerical methods are iterative:
 - Given a starting point x^1 we compute $x^{k+1} = x^k + s^k, k = 1, 2, \dots$
- A good starting point is essential for speedy convergence: x^1 should be near the solution x^*



155

Guidelines for solving the problem

- Find an area containing the root
- The better the initial guess is – the faster the convergence
- Select a suitable solution method depending on the problem type:
 - Real or complex problem
 - One or several equations
 - State of nonlinearity
- When candidate solution has been found:
 - Verify it



156

Solution methods for one equation

- If the **derivative is easy** to compute:
 - Newton's method is efficient
 - A good initial guess also helps
- If the derivative is **not available**
 - Brent/Dekker method is robust and fast
- Polynomial equations have special solution algorithms
- Solution methods are available in the NAG and IMSL commercial libraries



157

Solution methods for systems of equations

158

- If the derivatives are easy to compute and you have a good initial guess:
 - Newton's method is efficient
- When the derivatives are not available:
 - Quasi-Newton methods are good choices
- It is also possible to use methods for nonlinear least squares problems:

$$f(x) = 0 \rightarrow \min_x \sum_i f_i(x)^2$$

- However, the solutions of the minimization problem may not solve the system of equations



158

Newton's method for one equation

159

- Newton's method can be derived from the Taylor expansion of function $f(x)$:

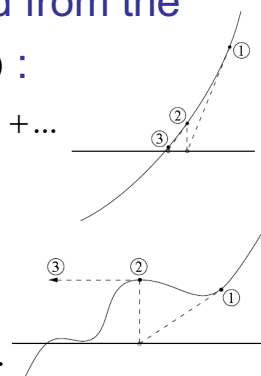
$$f(x+d) \approx f(x) + f'(x)d + \frac{f''(x)}{2}d^2 + \dots$$

- Setting $f(x+d) = 0$ we get $d = -\frac{f(x)}{f'(x)}$

- Newton's method:

1. Guess an initial solution x_1
2. Compute $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, k = 1, 2, 3, \dots$

- If $f'(x_k) = 0$ for some k , the method **fails**
- If $f'(x^*) = 0$ the method is **not efficient**



159

Newton's method for systems of equations

160

- The problem: $f(x) = 0$
- Guess an initial solution $x^1 \in \mathbb{R}^n$
- Solve the linear equations $J(x^k)s_N^k = -f(x^k)$
$$x^{k+1} = x^k + s_N^k$$

with $J(x^k)$ is the Jacobian of the function f , and s_N^k is the Newton step

- The iteration converges quadratically near the solution
 - May fail if the initial guess is poor
- The Jacobian matrix is **expensive** to compute



160

Software for nonlinear equations

161

- Mathematica and Matlab offer routines for solving simple nonlinear equations
 - Not too many variables
 - Too much nonlinearity
- IMSL and NAG subroutine libraries offer reliable routines for Fortran & C programmers
- The Netlib archive (<http://www.netlib.org>) contains source code for solution routines



161

Table of Contents

- Motivation & Trends in HPC
- Mathematical Modeling
- **Numerical Methods used in HPSC**
 - Systems of Differential Equations: ODEs & PDEs
 - Automatic Differentiation
 - Solving Optimization Problems
 - Solving Nonlinear Equations
 - **Basic Linear Algebra, Eigenvalues and Eigenvectors**
 - Chaotic systems
- HPSC Program Development/Enhancement: from Prototype to Production
- Visualization, Debugging, Profiling, Performance Analysis & Optimization



162

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 $x = (x_1, \dots, x_n)$
- Similarly **matrices** are simply **two dimensional arrays** of numbers



163

Algebraic operations with vectors and matrices

164

- 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*

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

- 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



164

Algebraic operations with vectors and matrices (2)

165

- 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 $p \times n$ and the size of **B** is $n \times q$, **C** will be of size $p \times q$
- 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



165

Algebraic operations with vectors and matrices (3)

166

- 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**



166

Methods for Solving Linear Equations

167

- 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



167

Methods for Solving Linear Equations (2)

168

- 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



168

Linear Transformations

169

- 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 $A: \mathbb{R}^2 \Rightarrow \mathbb{R}^2$ 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 $y = Ax$



169

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



170

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**



171

Eigenvalues and Eigenvectors (2)

- A square matrix of size $n \times n$ 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 \mathbb{R}^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 $\det(A - \lambda I) \neq 0$



172

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
- $tr(A) = \sum_j \lambda_j$
- $\det(A) = \prod_j \lambda_j$



173

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

$$\begin{aligned} x(t) &= c_1 e^{\lambda_1 t} v_1 + c_2 e^{\lambda_2 t} v_2 \\ &= c_1 e^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} + c_2 e^{2t} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} c_1 e^t + c_2 e^{2t} \\ c_2 e^{2t} \end{pmatrix} \end{aligned}$$



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 $\text{Re } \lambda < 0$
- 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



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



176

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



177

Table of Contents

- Motivation & Trends in HPC
- Mathematical Modeling
- **Numerical Methods used in HPSC**
 - Automatic Differentiation
 - Systems of Differential Equations: ODEs & PDEs
 - Solving Optimization Problems
 - Solving Nonlinear Equations
 - Basic Linear Algebra, Eigenvalues and Eigenvectors
 - **Chaotic systems**
- HPSC Program Development/Enhancement: from Prototype to Production
- Visualization, Debugging, Profiling, Performance Analysis & Optimization



178

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 = 'none of the above three'
- **Bifurcation:** if the system splits into two branches
 - Usually happens when a control parameter reaches a critical value



179

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



180

How to recognize chaos? (2)

- Signatures of chaos:
 - Patterns in power spectral analysis of time series
 - 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



181

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



182

Table of Contents

- Motivation & Trends in HPC
- Mathematical Modeling
- Numerical Methods used in HPSC
 - Automatic Differentiation
 - Systems of Differential Equations: ODEs & PDEs
 - Solving Optimization Problems
 - Solving Nonlinear Equations
 - Basic Linear Algebra, Eigenvalues and Eigenvectors
 - Chaotic systems
- **HPSC Program Development/Enhancement: from Prototype to Production**
- Visualization, Debugging, Profiling, Performance Analysis & Optimization



183

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



184

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



185

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 spent optimizing the implementation & parallelizing it
 - Attention should be paid to performance



186

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



187

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 (Toolboxes)



188

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



189

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:

$$\sum_i^N a_{ki} n_i = b_k; \quad k = 1, 2, \dots, M$$
- Where
 - a_{ki} is the subscript of the k -th 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



190

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:

$$p_i = \frac{n_i}{n_t} p_t$$
 - Where
 - p_t is the total pressure and
 - $n_t = \sum n_i$ is the total molar amount of the species



191

Case study: Simplified Version (2)

- We thus obtain $\mu_i = \mu_i^* + RT \log \frac{n_i}{n_i}$
- Where $\mu_i^* = \mu_i^0 + RT \log p_i$
- 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



192

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: C/C++/Fortran + Numerical libraries
- Choose the suitable numerical methods
 - Method of projected gradient
 - Method of augmented Lagrangian
 - Sequential Quadratic Programming (SQP)



193