# Table of Contents



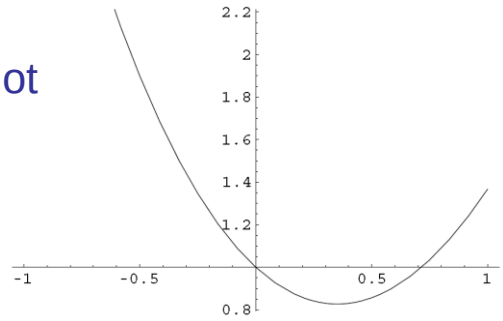


# Solving optimization problems

- What is an optimization problem?
- Examples of optimization problems
- Selected problem types and solution methods

# A simple optimization problem

- The problem $\min_{x \in \Re} f(x) = e^{-x} + x^2$
- Has the following plot




- And the solution: $f(x) = 0.827$ at $x = 0.352$

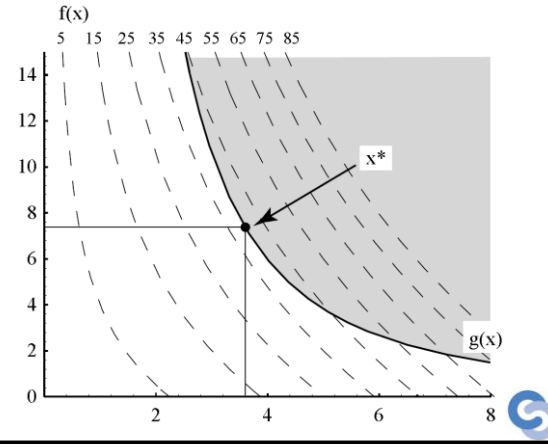


# A 2D optimization problem

- How to manufacture a 0.3 l metal can with as little material as possible?
- The height of the can is *h*, its radius *r*, the volume $\pi \cdot r^2 h$ the surface area $2\pi \cdot r^2 + 2\pi \cdot rh$
- With some variable changes we get the optimization problem:
  - Objective function $\min_{x \in \Re^2} f(x) = x_1^2 + x_1 x_2,$
  - Constraint $g(x) = -x_1 x_2 + 300/\pi \le 0,$
  - Variables $x_i \ge 0,\ i = 1,2.$

# A 2D optimization problem (2)

- The minimizing point $x^*$ satisfies $f(x^*) \leq f(x)$ for all feasible points in $x \in \Re^n$
- The minimum $x^* \approx (3.6, 7.3)^T$
- Leads to the minimum function value $f(x^*) \approx 39$






# A 2D optimization problem (3)

- The gradient of the objective function
$$f(x) = x_1^2 + x_1 x_2$$
- Is: $\nabla f(x) = \begin{pmatrix} 2x_1 + x_2 \\ x_1 \end{pmatrix}$
- The Hessian is: $\nabla^2 f(x) = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}$
- The Jacobian of the constraint function $g(x) = -x_1 x_2 + 300/\pi$ is $J(x) = (-2x_1 x_2 \quad -x_1^2)$

# Types of optimization problems

- Linear programming (LP):

$$\min_x c^T x, Ax = b \text{ or } Ax \le b, x \ge 0$$

- Integer programming (IP):

$$\min_{x,y} c^T x + d^T y, \text{ so that } Ax + Dy \le b$$

with $y_i, i = 1,...,r$ integer variables

- Quadratic programming (QP):

$$\min_x \frac{1}{2} x^T Q x + c^T x, Ax \le b$$



# Types of optimization problems (2)

- (Unconstrained) Nonlinear optimization:

$$\min_x f(x)$$

- Nonlinear least squares problems:

$$\min_x \sum_i f_i(x)^2$$

- Nonlinear optimization, linear constraints:

$$\min_x f(x), Ax = b \text{ or } Ax \le b$$

- Nonlinear optimization, nonlinear constraints:

$$\min_x f(x), g_i(x) \le 0, h_j(x) = 0$$

# Special optimization problems

- Global optimization
- Non-smooth optimization
- Optimal control
- Dynamic programming
- Min-max problems:

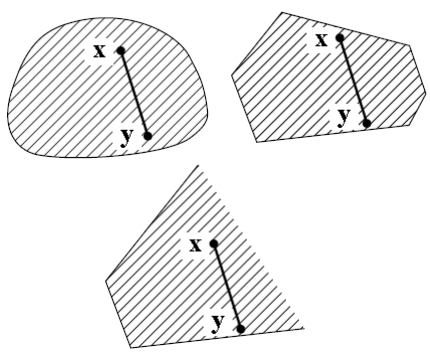
$$\min_{x} \max_{t} \{ f_i(x), i = 1,..., m \} \text{ so that } x \in F$$

- Combinatorial optimization
- Graph problems (e.g. network flow)

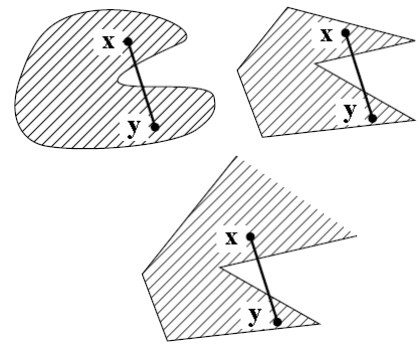


# Convex sets and optimization

- An optimization problem is convex when the objective function and the feasible set are convex



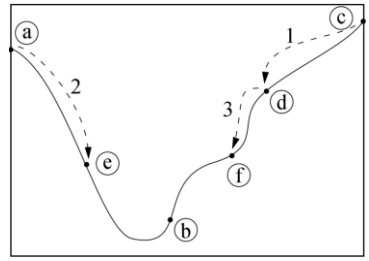

Convex sets



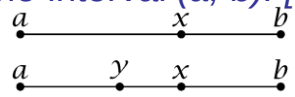

Non-Convex sets

# Scalar functions

- Let $f:\Re \to \Re$ be continuous
- Principle of optimization




- The best choice is the *golden ratio*: lengths *(a, x)* and *(x, b) are* $\frac{3-\sqrt{5}}{2} \approx 0{,}38197$ *and* $\frac{\sqrt{5}-1}{2} \approx 0{,}61803$ compared to the total length of the interval *(a, b): [a, b] is to [a, x] as [a, x] is to [x, b]*



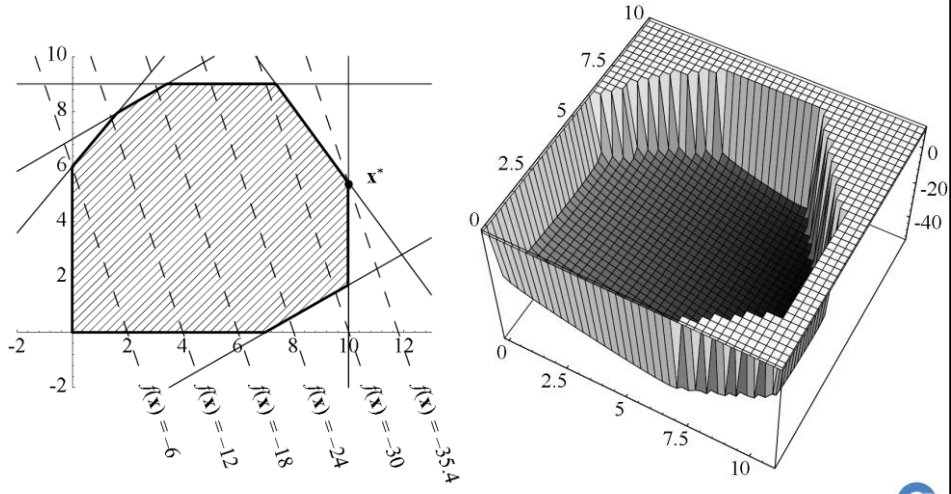



# Linear programming

- Considering $\min_x c^T x, Ax \le b, x \ge 0$
- And the example $\min_x f(x) = -3x_1 - x_2$ so that

$$\begin{cases} -6x_1 + 5x_2 \le 30 \\ -7x_1 + 12x_2 \le 84 \\ x_2 \le 9 \\ 19x_1 + 14x_2 \le 266 \\ x_1 \le 10 \\ 4x_1 - 7x_2 \le 28 \\ x_1 \ge 0, x_2 \ge 0 \end{cases}$$

# Linear programming (2)






# Integer programming

- Mixed-integer programming (MIP):

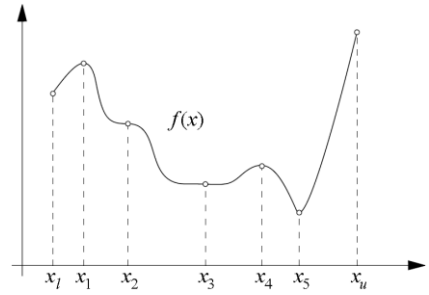
$$\min_{x,y} c^T x + d^T y, \text{ so that } Ax + Dy \le b$$

  where $y_i, i = 1,...,r$ are integer-valued variables, and $x \ge 0, \quad 0 \le y \le w$

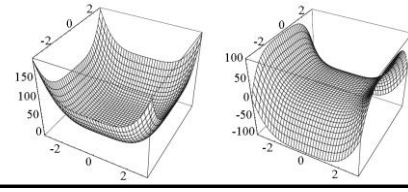
- Integer programming is much harder than LP
- Example:
  - 35 binary variables (0/1)
  - There are $2^{35} \approx 34 \times 10^9$ cases
  - If you can handle 1000 cases in one second, it would take 400 days to solve the problem

# Integer programming (2)

- There are approximate and heuristic solution methods for MIP problems
- Local minimum
  - There is a neighborhood with radius $r, r > 0$ where the function has the minimum value on the center $x^*$

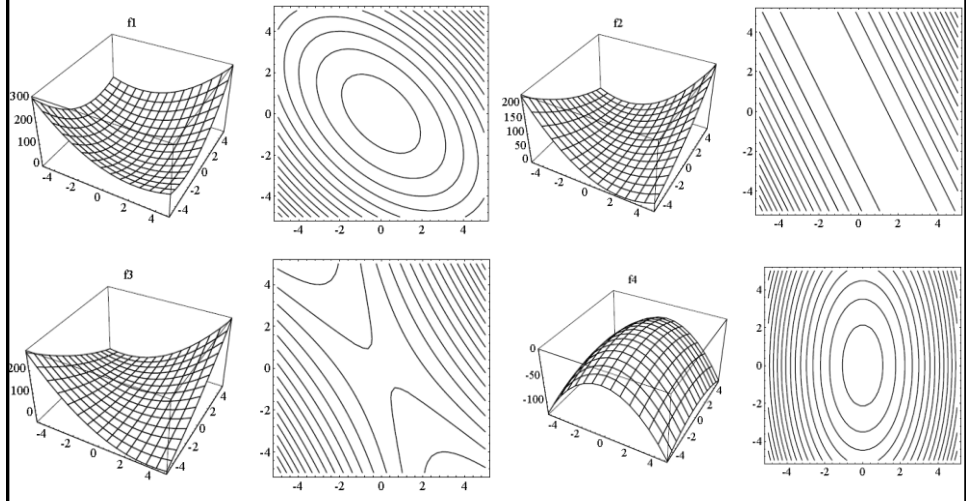



- Global minimum is the smallest of the local minima
- Saddle point



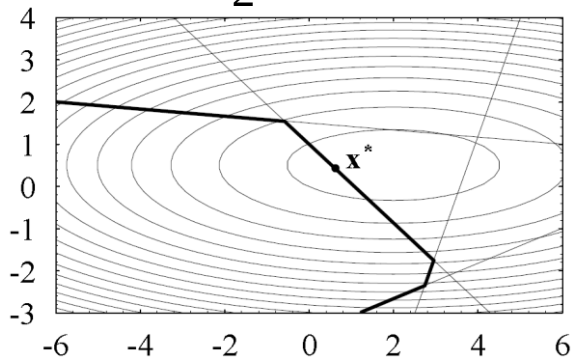


# Quadratic functions



# Quadratic programming

- The problem $\min_x \frac{1}{2} x^T Q x + c^T x, Ax \le b$




- If the matrix Q is positive definite → this is a convex optimization problem = easy



# Nonlinear optimization

- Optimizing continuous function $f : \Re^n \to \Re$

$$\min_x f(x), g_i(x) \le 0, \mathrm{i} = 1,...,\mathrm{p},$$

$$h_j(x) = 0,\ j = 1,...,1.$$

- Function *f(x)* is the objective function
- Functions $g_i(x)$ and $h_j(x)$ are constraints
- **There is no general method for solving nonlinear optimization problems**
- Therefore we will first look at unconstrained optimization

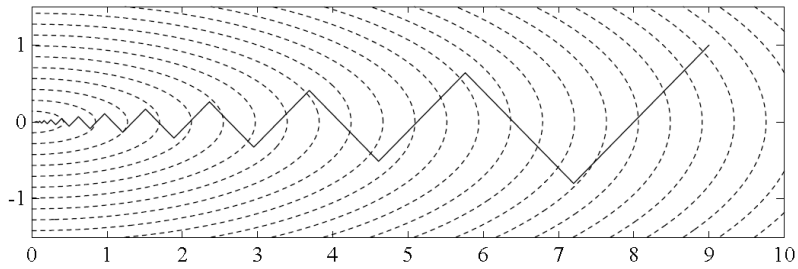
## Unconstrained optimization: steepest descent (bad method!)

- Select the direction, where the gradient is steepest:

$$x_{k+1} = x_k - \lambda_k \nabla f(x_k)$$

- Optimizing a quadratic function

$$f(x) = \tfrac{1}{2} x_1^2 + \tfrac{9}{2} x_2^2$$

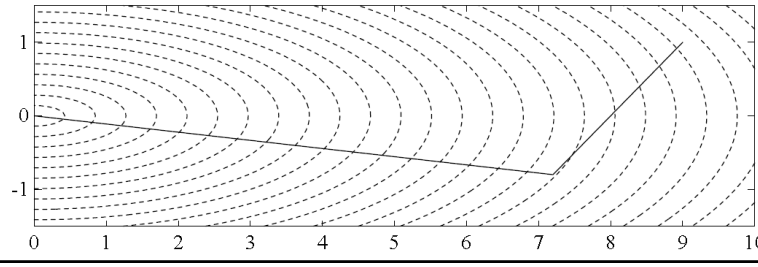



- Converges only linearly, and may be very slow!



## Conjugate gradient method

- Modest memory requirements
- Convergence is (super)linear
- Finds the minimum of an *n* dimensional quadratic function in *n* steps
- Optimizing a quadratic function $f(x) = \tfrac{1}{2} x_1^2 + \tfrac{9}{2} x_2^2$

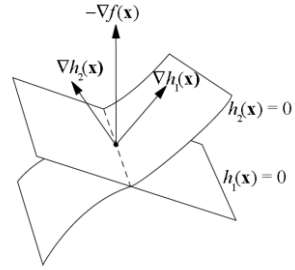



# Constrained nonlinear optimization

- The problem: $\min_{x \in \Re^n} f(x), g_i(x) \le 0, \mathrm{i} = 1,...,\mathrm{p},$

$$h_j(x) = 0,\ j = 1,...,1.$$

- The constraints may be linear or nonlinear




- Sequential quadratic programming (SQP) may be the most **used** and most **robust** method



# Nonlinear least squares

- The problem: $\min_x F(x) = \sum_{i=1}^{n} (f_i(x))^2 = f(x)^T f(x)$

where $x \in \Re^m$ and $f : \Re^m \to \Re^n$

- Newton's iteration for the problem

$$(J(x^k)^T J_k + S_k)s_k) = -J(x^k)^T f^k$$

$$x^{k+1} = x^k + s^k$$

- This is a costly solution method, because you need $\frac{1}{2} mn(n+1)$ second derivatives to calculate $S_k$
- The most used methods are Gauss-Newton (simpler) and Levenberg-Marquardt (more robust)

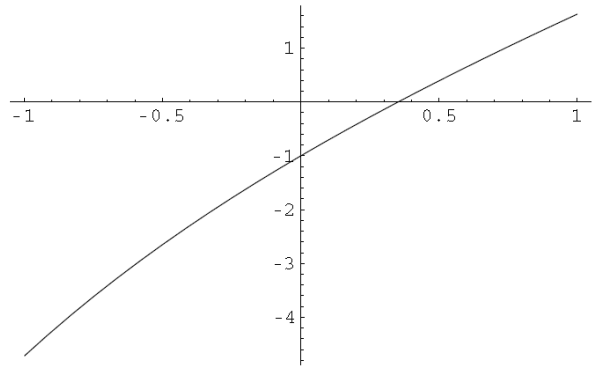
# Table of Contents





# Nonlinear Equation Example

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




- The solution: $x = 0.351734$

# Nonlinear equations

- Find the roots of the system of equations:

$$f : \Re^n \to \Re^n, \quad f(x) = 0$$

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

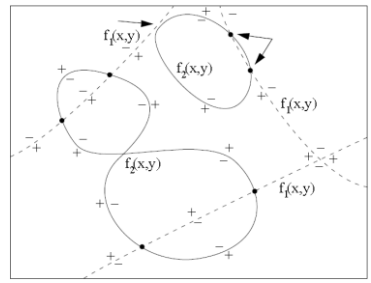
$$\begin{cases} f_1(x_1,...,x_n) = 0 \\ f_2(x_1,...,x_n) = 0 \\ ... \\ f_n(x_1,...,x_n) = 0 \end{cases}$$

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



# 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 $x^{k+1} = x^k + s^k, k = 1,2,...$
- A good starting point is essential for speedy convergence: $x^1$ should be near the solution $x^*$

# 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



# 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's method is robust and fast
- Polynomial equations have special solution algorithms
- Solution methods are available in the NAG and IMSL commercial libraries

# Solution methods for systems of equations

- 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:

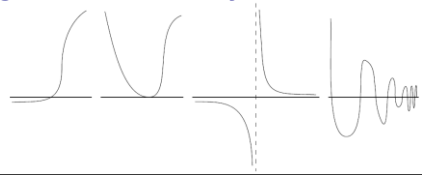
$$f(x) = 0 \rightarrow \min_x \sum_i f_i(x)^2$$

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



# Problems with finding a root

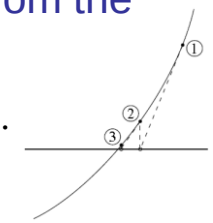
- If the function is continuous on the interval $[a,b]$ and $f(a)f(b) < 0$ there is at least one root
- Compare with the equation: $f(x) = (x-a)^2 = 0$
- Singularities: function $\sin(1/x)$ has infinitely many roots near the point $x = 0$
- Singular function $1/(x-a)$ changes sign near $x = a$ although there is no root
- Bad scaling: solution may lie in $x \in [1, 1+10^{-100}]$



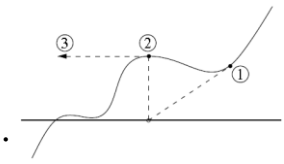
# Newton's method for one equation

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

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




- 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 $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, k = 1,2,3,\ldots$




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



# Newton's method for systems of equations

- The problem: $f(x)=0$
- Guess an initial solution $x^1 \in \Re^n$
- 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

# Software for nonlinear equations

- 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



# 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)



- 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)



- 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



# 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 $\lambda$ 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**

# 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 $\lambda_j$ 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

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

194

# 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 $\mathrm{Re}\,\lambda < 0$
- 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

195

# 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

# 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