In our example, the result is exact.

But, what if the numbers are functions (e.g. representations of π)?

Precision is lost.

This can be a serious problem!

If possible, reformulate problem so that this situation does not occur.

Example:

Solving $\boxed{ax^2 + bx + c = 0}$

Solution: $x_\pm = \frac{1}{2a}\left\{-b \pm \sqrt{b^2 - 4ac}\right\}$

Problem when $4ac \ll b^2$.

$$x_+ = \frac{1}{2a}\left\{ -b + \sqrt{b^2 - 4ac} \right\}$$

Subtraction of two almost equal numbers.

Reformulating solution:

Define

$$q = -\frac{1}{2}\left(b + \mathrm{sgn}(b)\sqrt{b^2 - 4ac}\right)$$

We then have that

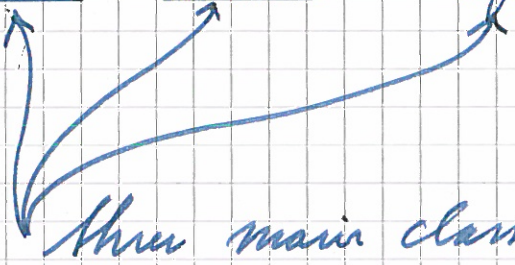
$$x_+ = q/a$$

$$x_- = c/q$$

— and the problem has gone away!

Some words about computer architecture.

Scalar, vector and parallel machines



three main classes.

They define how the computer handles loops:

```
do i = 1, N
a(i) = b(i) * c(i)
enddo
```

In order to perform the multiplication, the computer must for each term

$$a(i) = b(i) * c(i)$$

1. Recover b(i) from memory
2. —"— c(i) —"—
3. Perform multiplication
4. Store a(i) in memory.

Scalar computation:

Steps 1-4 are performed for element i.

When finished, steps 1-4 are performed for element i+1

....

and so forth.

Vector computation:

When step 1 has been performed for element 1, step 1 can be started for element $i+1$ while step 2 is performed for element i.

...

and so on

Key concept: **assembly line.**

Parallel computing

Do each element i in calculation simultaneously on different processing units.

The Basics of Computational Physics.

1. Linear algebra.

Solving sets of linear equations.

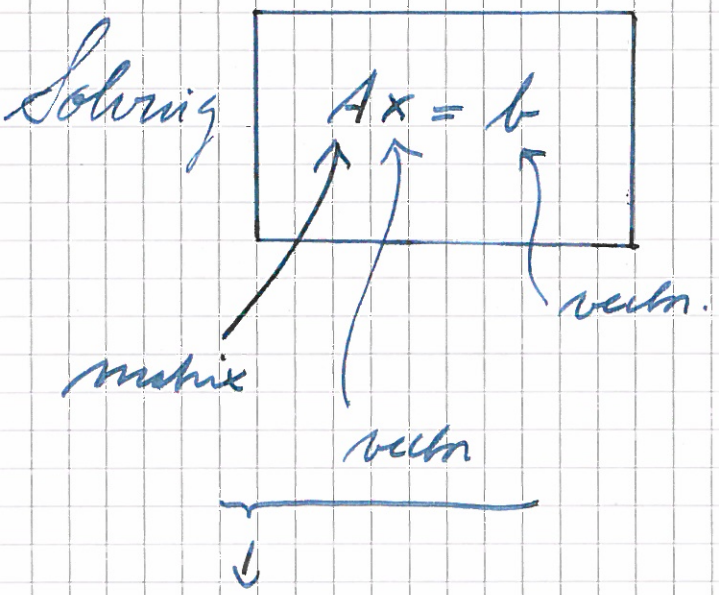
2. Representing differentials.

Finite differences as approximations to derivatives.

3. Stochastics.

Given probabilities, generate numbers; the opposite of statistics.

1. Linear algebra



$$A_{11} x_1 + A_{12} x_2 + \cdots + A_{1N} x_N = G_1$$
$$A_{21} x_1 + A_{22} x_2 + \cdots + A_{2N} x_N = G_2$$
$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$
$$A_{M1} x_1 + A_{M2} x_2 + \cdots + A_{MN} x_N = G_M$$

N unknowns.
M equations.

If $\boxed{N = M}$ two possible situations:

1. Unique solution exist.

2. Some of the equations are linear combinations of others ⇒

Degenerate problem.

If this is the case, A is singular.

Cannot be inverted.

The solution of $Ax = b$ then consists of a particular solution x_p ($Ax_p = b$) plus any linear combination of zero-vectors x_i^0 such that $Ax_i^0 = 0$:

$$x = x_p + \sum_{i=1}^{D} c_i x_i^0$$

D is the dimension of null-space.

If $N > M$ equation set is also degenerate

— more variables than equations.

If $N < M$

The system is underdetermined

and probably

no solution

exists.

This is typically caused by

the physical assumptions

underlying the equations not being fully compatible.

However, it is still possible to find <u>a best solution</u> given the circumstances.

We will take as primary algorithms for handling degenerate and non-existant solutions.

> We now assume $M = N$ and A^{-1} does exist.

We now assume $M = N$ and A^{-1} does exist.

Two classes of approaches:

1. Direct methods

e.g.

Gauss elimination
LU decomposition
Tridiagonal matrices
Toeplitz matrices

Convert matrix with symmetries that may be exploited to speed up the inversion tremendously.

2. Iterative methods

Jacobi
Gauss - Seidel
SOR
} One family

Steepest descent
Conjugate gradient
Powell
} another family.

Preconditioning:

Reformulating the problem to have easier matrices to deal with.

Guess a matrix C such that

CAC^{-1} is simpler (to be defined) than A.

Then solve the problem

$$(CAC^{-1})\underbrace{(Cx)} = (Cb)$$

$$\Downarrow$$

$$Cx$$

Then find $\underline{x = C^{-1}(Cx)}$

Examples:

Fourier acceleration

Multigrid methods.

---

1. Direct methods

Then are used when A is dense.

Gauss elimination and backwards substitution.

$$\begin{pmatrix} A_{11} & A_{12} & \cdots & & \\ A_{21} & A_{22} & & & \\ \vdots & & & & \\ & & & \ddots & \\ & & & \cdots & A_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \\ \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \\ \\ b_N \end{pmatrix}$$

Transform $\longrightarrow$

$$\begin{pmatrix} A_{11}' & A_{12}' & A_{13}' & \cdots & & \\ 0 & A_{22}' & A_{23}' & & & \\ 0 & 0 & A_{33}' & & & \\ 0 & 0 & 0 & & & \\ \vdots & & & & & \\ & & & & \ddots & \\ & & & & \cdots & A_{NN}' \end{pmatrix} \begin{pmatrix} x_1' \\ x_2' \\ x_3' \\ \vdots \\ \\ \\ x_N' \end{pmatrix} = \begin{pmatrix} b_1' \\ b_2' \\ b_3' \\ \\ \\ \\ b_N' \end{pmatrix}$$

1. Find the largest (in absolute value) element in 1st column. This comes in row i.

   Switch 1st and ith row in A, x and b.

   (Partial pivoting.)

2. Subtract from rows 2 to N in A and b row 1 multiplied by a suitable factor such that $A_{i,1} = 0$ for $i = 2, \ldots, N$.

3. Repeat for 2. column and rows up to $N-1$.

   and so on.

After A' is in upper triangular form.

Then iterate

$$x_0 \to x_1 \to x_2 \to \cdots \to x_k \to \cdots \to x_\infty$$

If now

$$\| A x_{k+1} - b \| < \| A x_k - b \|,$$

then, we must have that

$$x_\infty = x,$$

which solves the equation

$$Ax = b.$$

Each iteration will contain a few multiplications of vectors and A.

It is thus important that the matrix A is sparse.

$\Rightarrow$ that elements

$$A_{ij} = 0.$$

Typically, a physics problem will be formulated by the interactions between - say - N objects. Each object i is characterized by a variable x_i.

Each object depends on the rest of N other objects.

Hence, we have

$$x_1 = f_1(x_{1(1)}, x_{2(1)}, \ldots, x_{N(1)})$$

$$x_2 = f_2(x_{1(2)}, x_{2(2)}, \ldots, x_{N(2)})$$

$$\vdots$$

$$x_N = f_N(x_{1(N)}, x_{2(N)}, \ldots, x_{N(N)})$$

(e.g. through interactions of finite range.)

Next: Backwards substitution:

$$\begin{pmatrix} A'_{11} & \cdots & & \\ \vdots & \ddots & & \\ & & A'_{N-1\,N-1} & A'_{N-1\,N} \\ & & 0 & A'_{NN} \end{pmatrix} \begin{pmatrix} x'_1 \\ \vdots \\ x'_{N-1} \\ x'_N \end{pmatrix} = \begin{pmatrix} b'_1 \\ \vdots \\ b'_{N-1} \\ b'_N \end{pmatrix}$$

$$x'_N = \frac{1}{A'_{NN}} b'_N$$

$$x'_{N-1} = \frac{1}{A'_{N-1\,N-1}} \left\{ b'_{N-1} - A'_{N-1\,N} x'_N \right\}$$

$$\vdots$$

$$x'_i = \frac{1}{A'_{ii}} \left\{ b'_i - \sum_{j=i+1}^{N} A'_{ij} x'_j \right\}$$

---

2. Iterative methods

Basic idea:

- Start with some initial x_0.

If this is a linear problem, it is on the form

$$x_i = \sum_{j \in N(i)}' a_{ij} x_j + c_i$$

This can be formulated as $Ax = c$.

When $n \ll N$, the corresponding A is <u>sparse</u>.

Iterative algorithms

2 classes: { "Jacobi" / "Steepest descent"

(A) Jacobi algorithm

$$Ax = c$$

$x = x' + \delta x$ ← error.

Solution ↑ guess ↑

$$Ax = b$$

$$A(x' + \delta x) = b$$

$$\boxed{A\,\delta x = b - Ax'}$$

This simple equation is the starting point for an iterative algorithm:

$$\boxed{A(x_{k+1} - x_k) = b - Ax_k}$$

δx $\qquad$ x'

This is not practical since we must invert A to find x_{k+1} when we know x_k.

But, in reality, all we need is that

$$\|x_{k+1} - x_k\| \to 0 \quad \text{as} \quad k \to \infty.$$