Real Time Processing Summary

$$y = a * x + v$$

know a, $s = \mathrm{cov}(v)$, $f = \mathrm{cov}(x)$

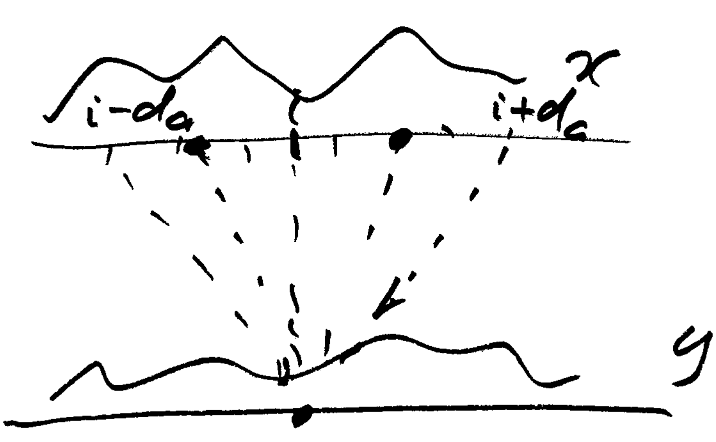
(a, s, f)

r - processing algorithm.

Δ - support of r

$$\left.\begin{array}{ll} p = a * f * a^* + s & \Rightarrow P \\ q = f * a^* & \Rightarrow q_\Delta \end{array}\right\} \quad \begin{array}{l} r_\Delta = P^{-1} q_\Delta \\ r_\Delta \to r \end{array}$$

$$\hat{x} = r * y$$



$\Delta_a = [-d_a, d_a]$

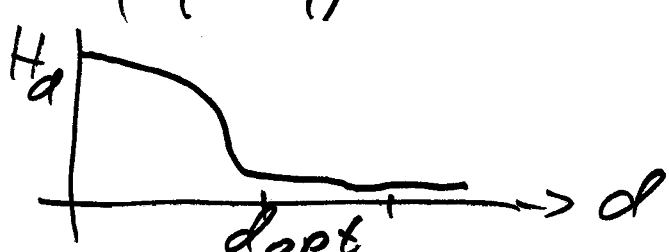
a symmetric if

$$a_i = a_{-i}$$

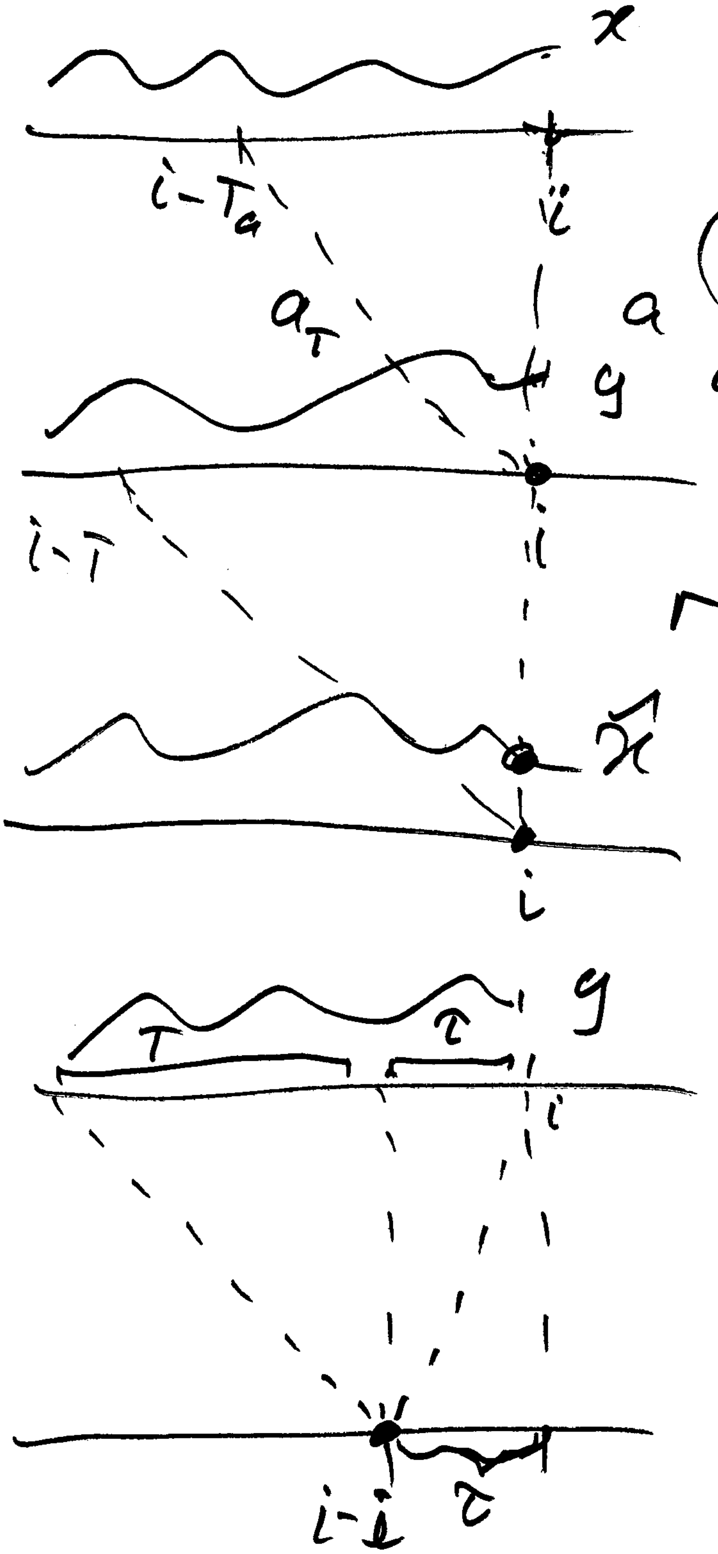
or $a^* = a$

$$\Delta = [-d, d]$$

$$H_d = E(\hat{x}_i - x_i)^2$$

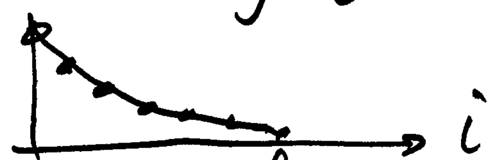


Time series



$$\Delta_a = [0, T_a]$$

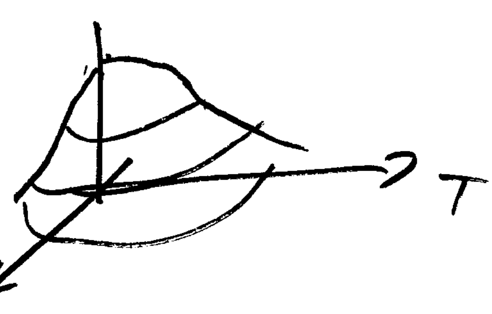
$$(a * x)_i = \sum_{j=0}^{T_a} a_j x_{i-j}$$



$$\Delta = [0, T]$$

$$\Delta = [-\tau, T]$$

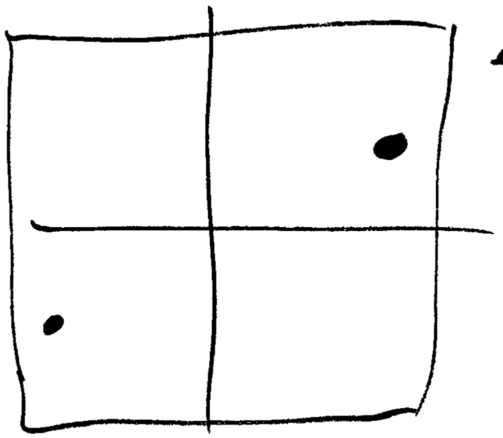
$H(\tau, T)$

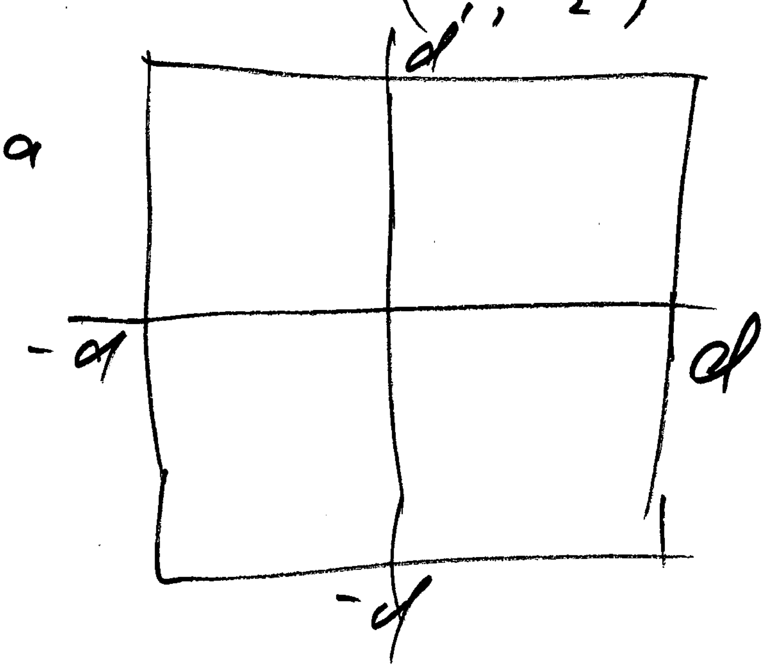


2D case.

Symmetric PSF

$a_i = a_{-i}$ $\qquad$ $i = (i_1, i_2)$





$\Delta = [-d, d]^2$

$H(d)$

In general $\quad y = Ax + v$

$R \Rightarrow \hat{x} = Ry$

$Ry = RAx + Rv$ $\qquad$ $\bar{S} = Var(\bar{v}) = RSR^*$

$\bar{y} = \bar{A}x + \bar{v}$

New "meas. system" $(\bar{A}, \bar{S})$

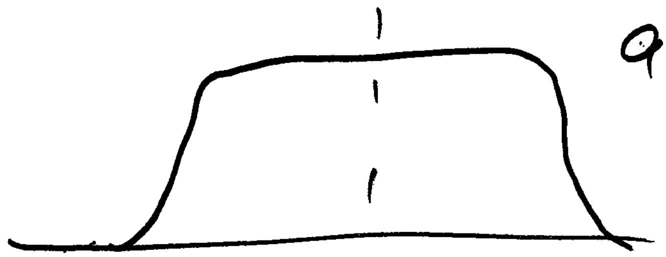
How close is $\bar{A}$ to I ?

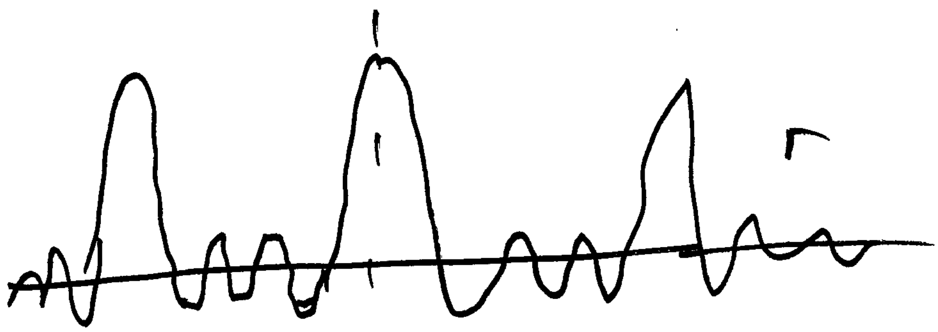
For unbiased est $\bar{A} = RA = I$

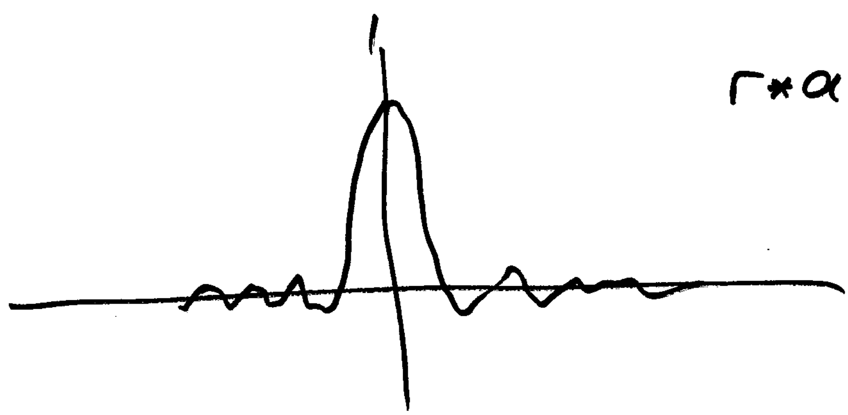
$$y = a * x + v$$

$$\underbrace{r * y} = \underbrace{(r * a)} * x + r * v$$

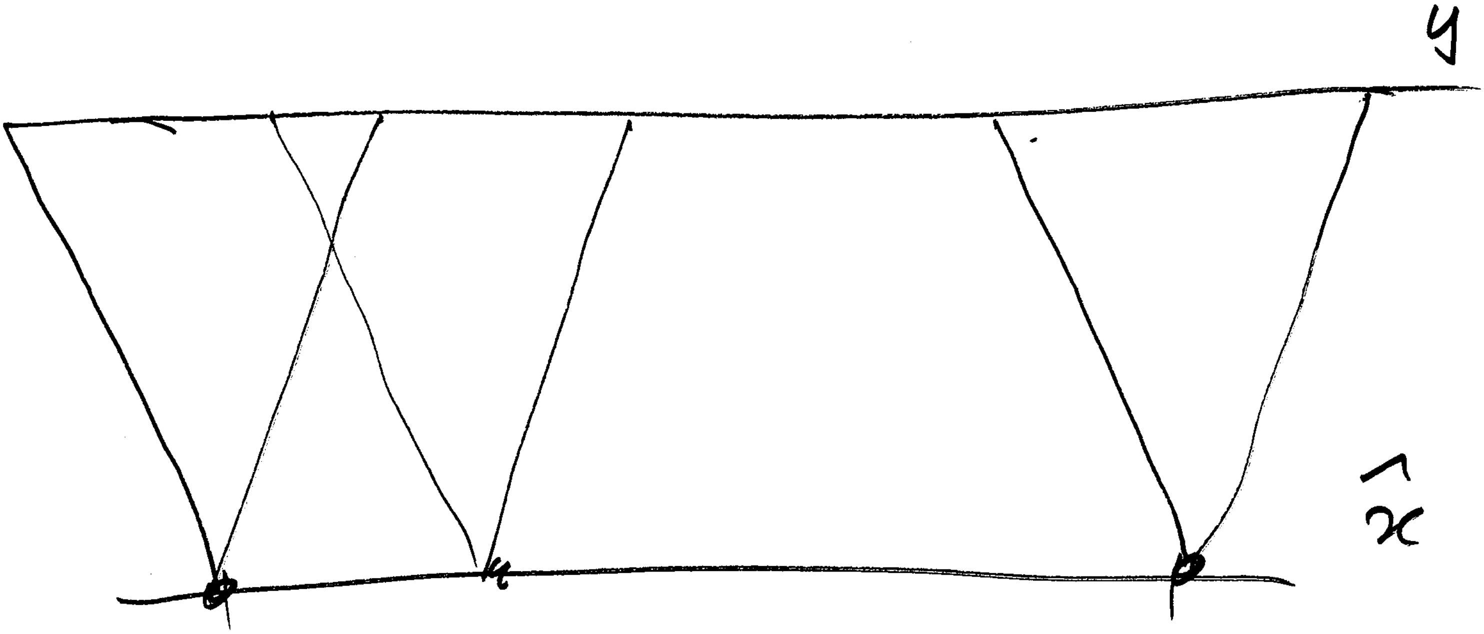
$$\bar{y} = \bar{a} * x + \bar{v}$$

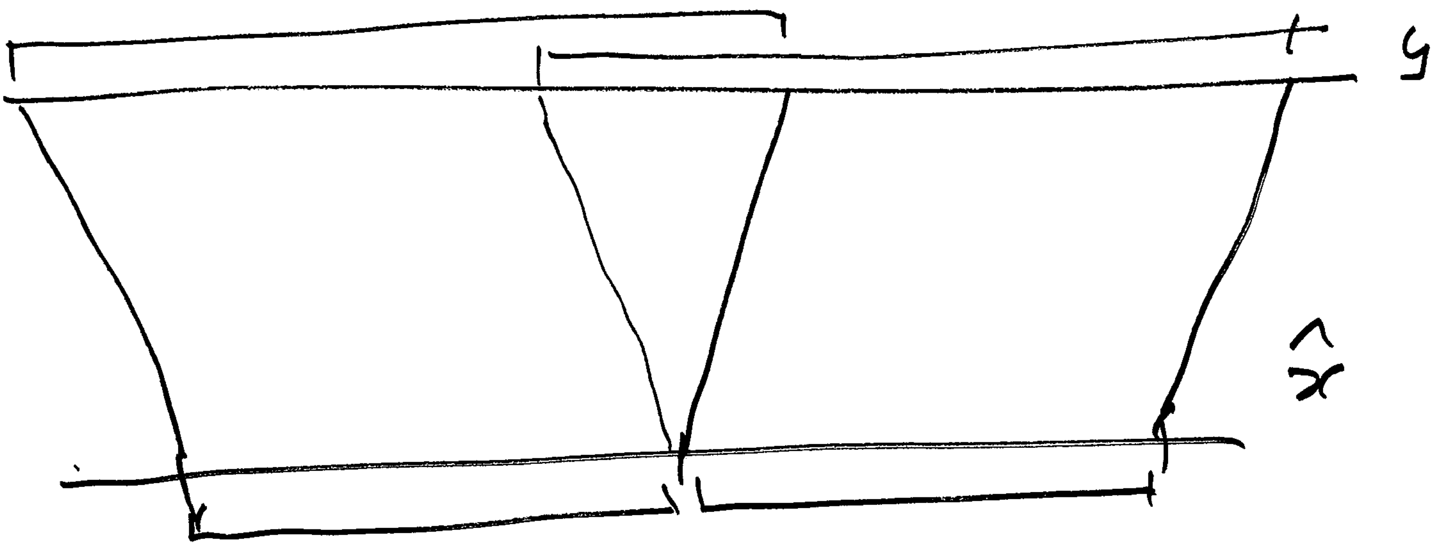


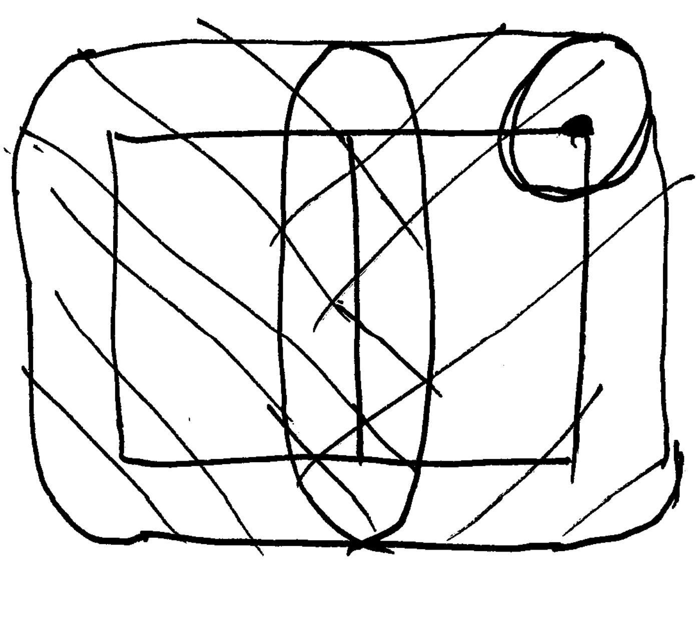




$r * a$ "close" to δ

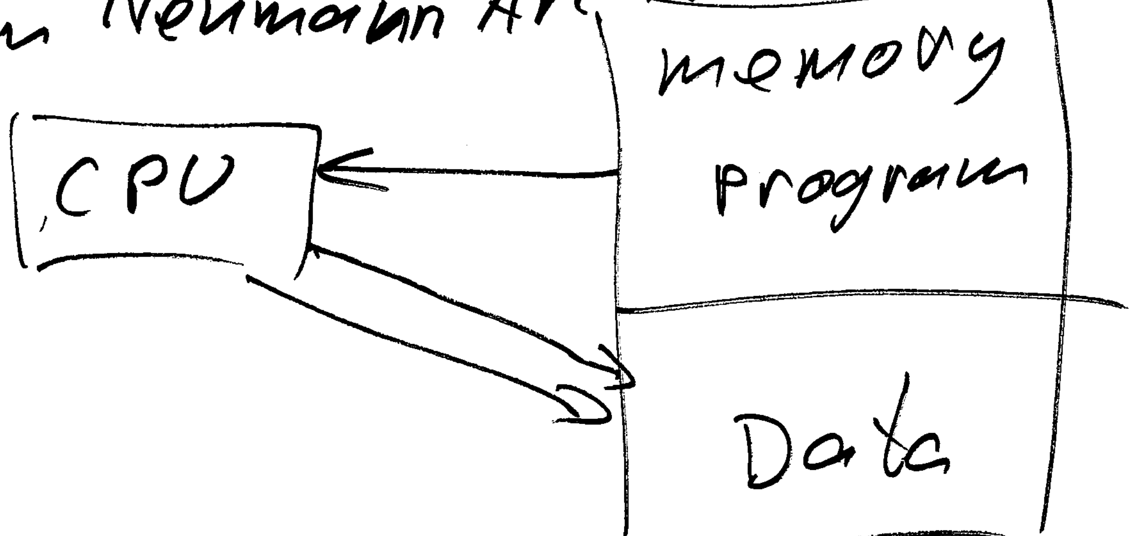

y
$\hat{x}$


y
$\hat{x}$



Arranging parallel computations.

Von Neumann Architecture:



Von Neumann prog. languages

John Backus

step by step instructions.

* Variables
* loops
* Goto

$f \quad g$

$h = f * g$

The 1977 ACM Turing Award was presented to John Backus at the ACM Annual Conference in Seattle, October 17. In introducing the recipient, Jean E. Sammet, Chairman of the Awards Committee, made the following comments and read a portion of the final citation. The full announcement is in the September 1977 issue of *Communications*, page 681.

"Probably there is nobody in the room who has not heard of Fortran and most of you have probably used it at least once, or at least looked over the shoulder of someone who was writing a Fortran program. There are probably almost as many people who have heard the letters BNF but don't necessarily know what they stand for. Well, the B is for Backus, and the other letters are explained in the formal citation. These two contributions, in my opinion, are among the half dozen most important technical contributions to the computer field and both were made by John Backus (which in the Fortran case also involved some colleagues). It is for these contributions that he is receiving this year's Turing award.

The short form of his citation is for 'profound, influential, and lasting contributions to the design of practical high-level programming systems, notably through his work on Fortran, and for seminal publication of formal procedures for the specifications of programming languages.'

The most significant part of the full citation is as follows:

'. . . Backus headed a small IBM group in New York City during the early 1950s. The earliest product of this group's efforts was a high-level language for scientific and technical computations called Fortran. This same group designed the first system to translate Fortran programs into machine language. They employed novel optimizing techniques to generate fast machine-language programs. Many other compilers for the language were developed, first on IBM machines, and later on virtually every make of computer. Fortran was adopted as a U.S. national standard in 1966.

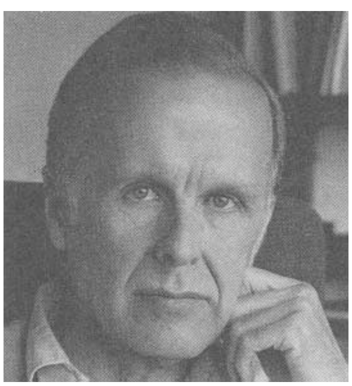
During the latter part of the 1950s, Backus served on the international committees which developed Algol 58 and a later version, Algol 60. The language Algol, and its derivative compilers, received broad acceptance in Europe as a means for developing programs and as a formal means of publishing the algorithms on which the programs are based.

In 1959, Backus presented a paper at the UNESCO conference in Paris on the syntax and semantics of a proposed international algebraic language. In this paper, he was the first to employ a formal technique for specifying the syntax of programming languages. The formal notation became known as BNF—standing for "Backus Normal Form," or "Backus Naur Form" to recognize the further contributions by Peter Naur of Denmark.

Thus, Backus has contributed strongly both to the pragmatic world of problem-solving on computers and to the theoretical world existing at the interface between artificial languages and computational linguistics. Fortran remains one of the most widely used programming languages in the world. Almost all programming languages are now described with some type of formal syntactic definition.' "

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus
IBM Research Laboratory, San Jose



Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak: their primitive word-at-a-time style of programming inherited from their common ancestor—the von Neumann computer, their close coupling of semantics to state transitions, their division of programming into a world of expressions and a world of statements, their inability to effectively use powerful combining forms for building new programs from existing ones, and their lack of useful mathematical properties for reasoning about programs.

An alternative functional style of programming is founded on the use of combining forms for creating programs. Functional programs deal with structured data, are often nonrepetitive and nonrecursive, are hierarchically constructed, do not name their arguments, and do not require the complex machinery of procedure declarations to become generally applicable. Combining forms can use high level programs to build still higher level ones in a style not possible in conventional languages.

Author's address: 91 Saint Germain Ave., San Francisco, CA 94114.

```
function s = SS(x)   % Sum Standard
n = length(x);
i = 1;
s = 0;
while i <= n
    s = s + x(i);
    i = i + 1;
end
end
```

$$X = \begin{bmatrix} \begin{bmatrix} T(x) \end{bmatrix} \\ \begin{bmatrix} B(x) \end{bmatrix} \end{bmatrix}$$

```
function s = S(x)   % Sum
if H(x)==1
    s = x;
else
    s = S(T(x)) + S(B(x));
end
end

function y = T(x)    % Top
y = x(1:floor(H(x)/2));
end

function y = B(x)    % Bottom
n = H(x);
y = x(floor(n/2)+1:n);
end

function n = H(x)   % Height
n = length(x);
end
```

```
function s = PS(x,y)   % Inner Product
                       % Standard
n = length(x);
i = 1;
s = 0;
while i <= n
    s = s + x(i)*y(i);
    i = i + 1;
end
end
```

```
function p = P(x,y)   % Inner Product
if H(x)==1
    p = x*y;
else
    p = P(T(x),T(y)) + P(B(x),B(y));
end
end

function y = T(x)    % Top
y = x(1:floor(H(x)/2));
end

function y = B(x)    % Bottom
n = H(x);
y = x(floor(n/2)+1:n);
end

function n = H(x)   % Height
n = length(x);
end
```

```
function y = Mv(A,x)   % Matrix x Vector
if H(A)==1
    y = P(A',x);
else
    y = [Mv(T(A),x); Mv(B(A),x)];
end
end
```

transp of a Row

T

B

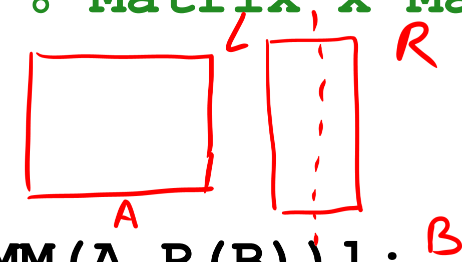
A

x

$[a; b] = \begin{bmatrix} a \\ b \end{bmatrix}$

```
function p = P(x,y)   % Inner Product
if H(x)==1
    p = x*y;
else
    p = P(T(x),T(y)) + P(B(x),B(y));
end
end

function y = T(x)    % Top
y = x(1:floor(H(x)/2), :);
end

function y = B(x)    % Bottom
n = H(x);
y = x(floor(n/2)+1:n, :);
end

function n = H(x)   % Height
n = size(x,1);
end
```

```
function C = MM(A,B)   % Matrix x Matrix
if W(B)==1
     C = Mv(A,B);
else
     C = [MM(A,L(B)), MM(A,R(B))];
end
end
```



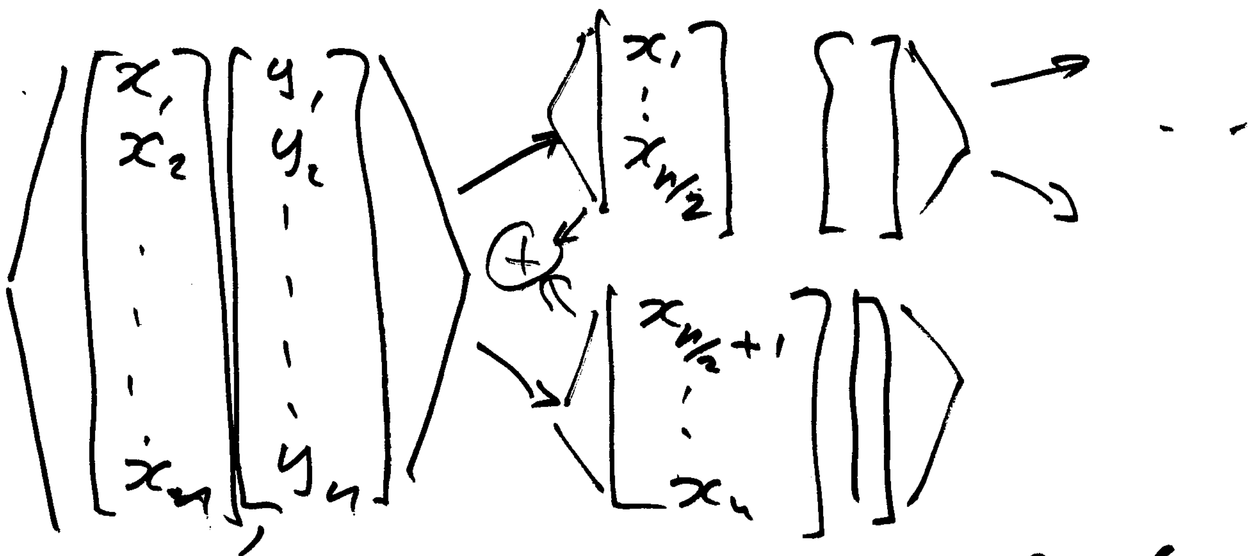

```
function y = Mv(A,x)   % Matrix x Vector
if H(A)==1
     y = P(A',x);
else
     y = [Mv(T(A),x); Mv(B(A),x)];
end
end
```

```
function p = P(x,y)   % Inner Product
if H(x)==1
     p = x*y;
else
     p = P(T(x),T(y)) + P(B(x),B(y));
end
end
```

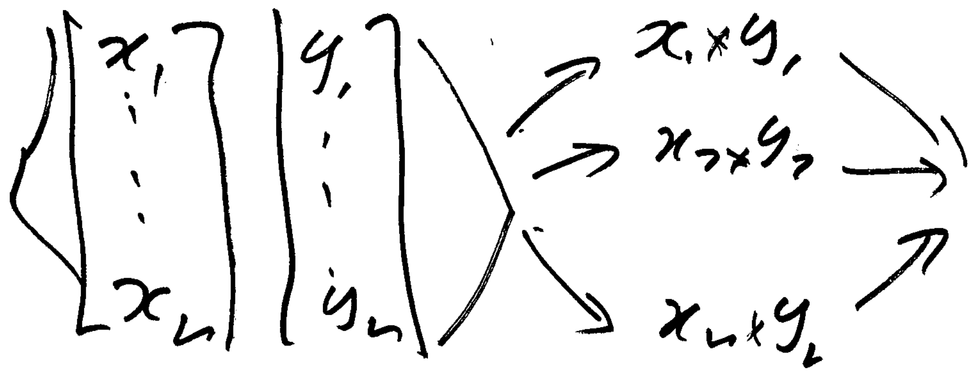
```
function y = T(x)    % Top
y = x(1:floor(H(x)/2), :);
end
```

```
function y = B(x)    % Bottom
n = H(x);
y = x(floor(n/2)+1:n, :);
end

function y = L(x)    % Left
y = x(:, 1:floor(W(x)/2));
end

function y = R(x)    % Right
n = W(x);
y = x(:, floor(n/2)+1:n);
end

function n = H(x)   % Height
n = size(x,1);
end

function n = W(x)   % Width
n = size(x,2);
end
```

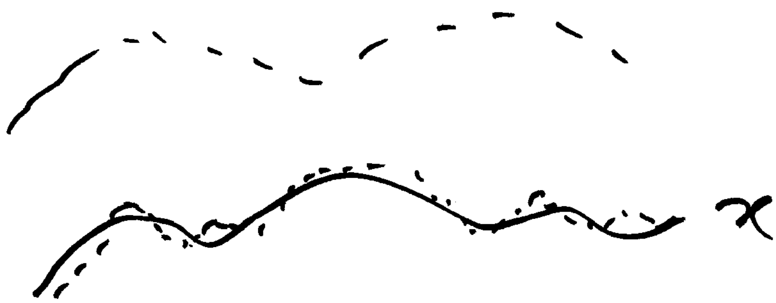
```
function C = MMS(A,B)   % Matrix x Matrix
                        % Standard
[n,m] = size(A);
[m,k] = size(B);
i = 1;
while i <= n
    j = 1;
    while j <= k
        s = 0;
        p = 1;
        while p <= m
            s = s + A(i,p)*B(p,j);
            p = p + 1;
        end
        C(i,j) = s;
        j = j + 1;
    end
    i = i + 1;
end
end
```

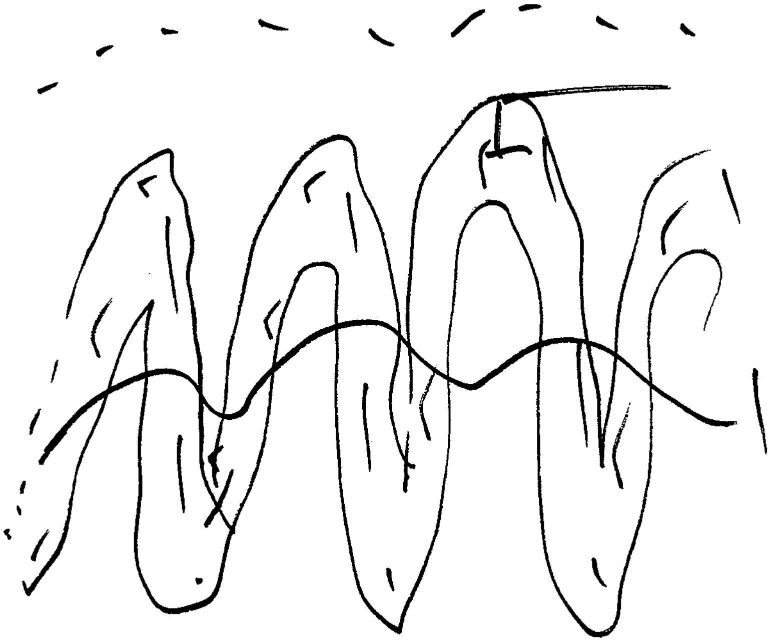


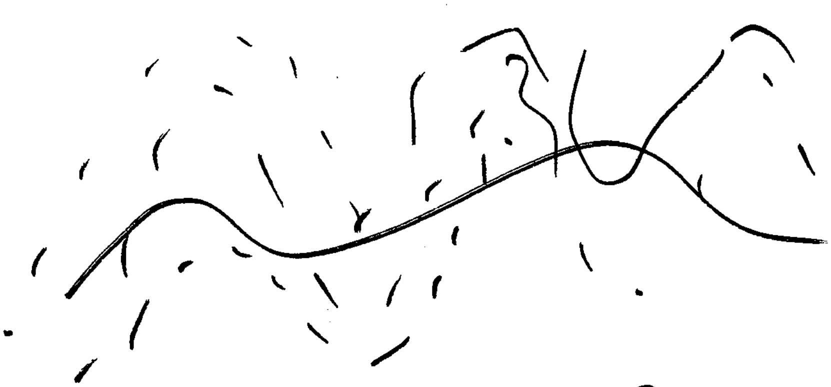
map - Reduce



Projects Tech. comments.



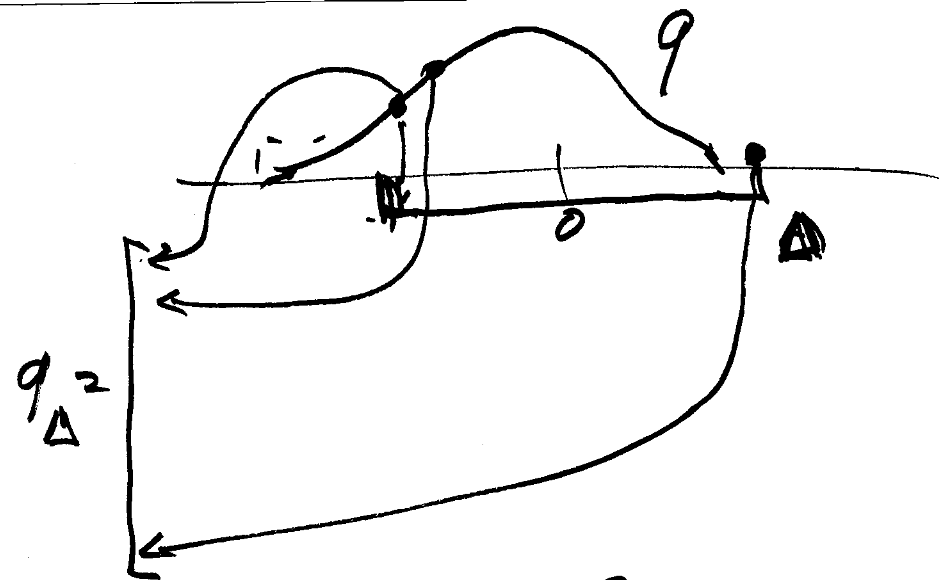


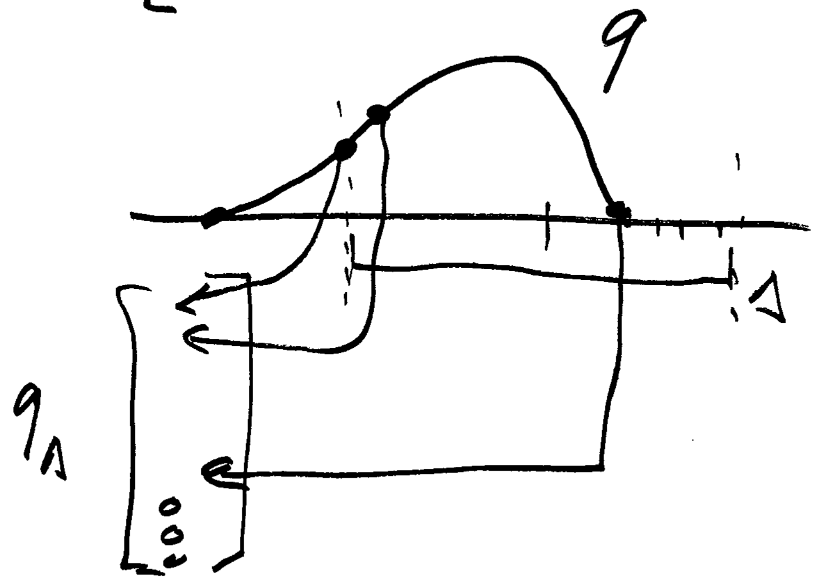


$$H = E\|\hat{x} - x\|^2 = \mathrm{tr}\, Q^{\,2}$$

$$Q = \mathrm{Var}(\hat{x} - x) \quad \text{matrix}$$

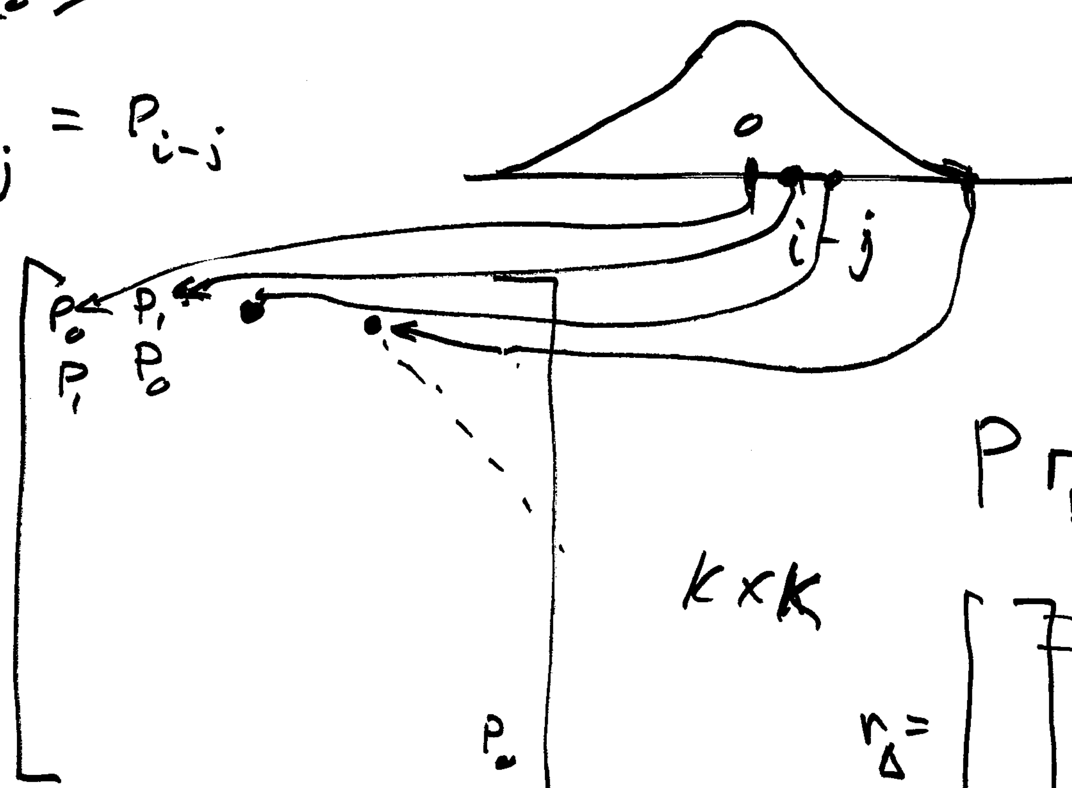
$$H = \underline{E}\sum(\hat{x}_i - x_i)^2 = \sum_{i=1}^{m} \underbrace{\underline{E}(\hat{x}_i - x_i)^2}$$





$$\underline{P}_{ij} = p_{i-j}$$

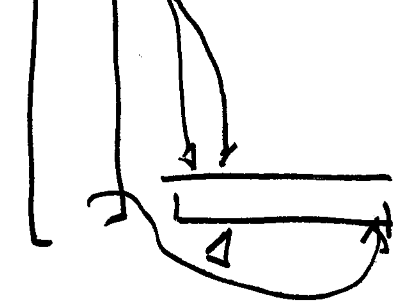
$\underline{P} =$

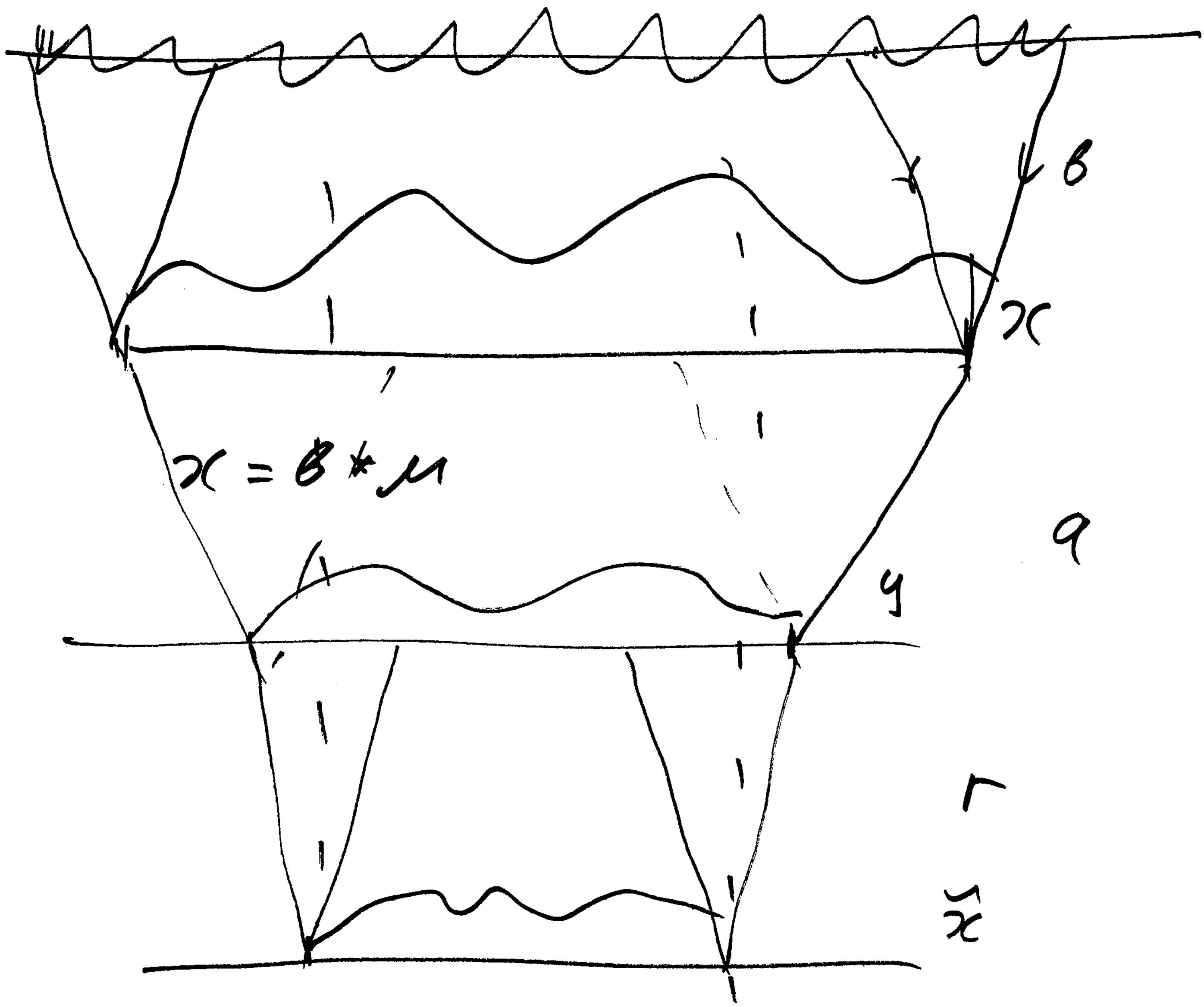


$$P\, r_\Delta = q_\Delta$$

$k \times k$

$r_\Delta =$




в
x
x = в * u
a
y
r
x̃