New Algorithm for Polynomial Plus-minus Factorization Based on FFT

Martin Hromčík, Michael Šebek

Department of Control Engineering
Czech Technical University in Prague, CZ

Polynomial plus-minus factorization

Given: Discrete-time polynomial

$$p(z) = p_0 + \ldots + p_n z^n$$

that is nonzero on the unit circle: $|z| = 1 \;\Rightarrow\; p(z) \neq 0$,

Find: stable polynomial $p^+(z)$ and unstable $p^-(z)$ such that

$$p(z) = p^+(z)\,p^-(z)$$

Motivation:
- cancellation of stable zeros in classical control
- time-optimal control (deadbeat, FIFO)
- exact model matching
- quadratic optimal filters
- algebraic approach to l_1 optimal control

Special case - symmetric: $p(z) = p(z^{-1}) \quad \left(p^*(z) = p(z^{-1}) \right)$

- less general but more popular (H_2,LQG
- many algorithms a programs

+/- factorization & polynomial roots

all roots

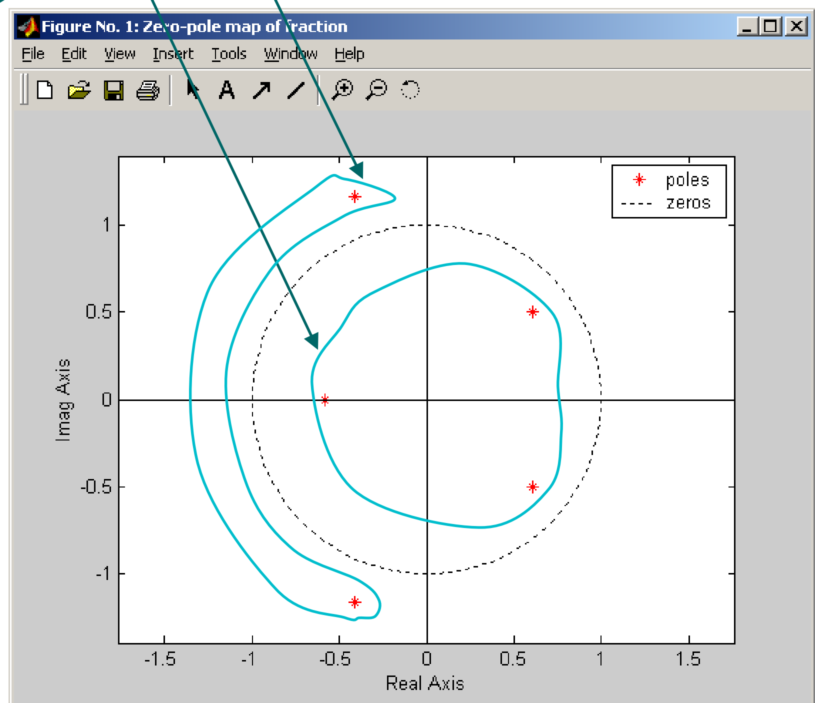
stable & unstable

$$p(z) = p^{+}(z) \cdot p^{-}(z)$$

unstable roots

stable roots

Example:



- **not useful** for high degrees for rounding errors

+/- factorization via symmetric factorization & GCD

Given: $p(z)$

Compute: $P(z) = p(z) \cdot p^{*}(z),\ p^{*}(z) = p(z^{-1})$

Spectral factorization of $P(z) : P(z) = q^{*}(z) q(z),\quad q(z)$ stable

Get $p^{+}(z)$ as greatest common divisor of $q(z)$ and $p(z)$

Weak point:

polynomial greatest common divisor computation - numerically demanding, unreliable for high degrees

Discrete Fourier transform (DFT)

$$P_k = \sum_{i=0}^{n} p_i e^{-j\frac{2\pi k}{n+1}i}$$

$$\begin{bmatrix} p_0 & p_1 & \cdots & p_n \end{bmatrix}$$

direct DFT

$$\begin{bmatrix} P_0 & P_1 & \cdots & P_n \end{bmatrix}$$

inverse DFT

$$p_i = \frac{1}{n+1}\sum_{k=0}^{n} P_k e^{j\frac{2\pi i}{n+1}k}$$

infinite sum => finite sum

Relation to Z-transform:

$$P(e^{j\omega}) = \sum_{i=0}^{\infty} p_i e^{-j\omega i}$$

$$\{p_0, p_1, \ldots, p_n, \ldots\} \quad \longleftrightarrow \quad P(z)$$

$$p_i = \frac{1}{2\pi}\int_0^{2\pi} P(e^{j\omega}) \cdot e^{j\omega} d\omega$$

Discrete Fourier transform (DFT)

$$P_k = \sum_{i=0}^{n} p_i e^{-j\frac{2\pi k}{n+1} i}$$

$$\begin{bmatrix} p_0 & p_1 & \cdots & p_n \end{bmatrix}$$

direct DFT

inverse DFT

$$\begin{bmatrix} P_0 & P_1 & \cdots & P_n \end{bmatrix}$$

$$p_i = \frac{1}{n+1} \sum_{k=0}^{n} P_k e^{j\frac{2\pi i}{n+1} k}$$

Relation to Z-transform

$$P(e^{j\omega}) = \sum_{i=0}^{\infty} p_i e^{-j\omega i}$$

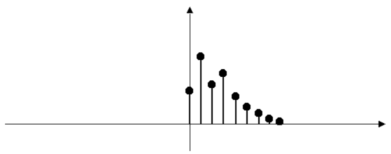
$$\{p_0, p_1, \ldots, p_n, \ldots\}$$

$$P(z)$$

integral =>
=> finite sum

$$p_i = \frac{1}{2\pi} \int_0^{2\pi} P(e^{j\omega}) \cdot e^{j\omega} d\omega$$

Discrete Fourier transform (DFT)



$$\begin{bmatrix} p_0 & p_1 & \cdots & p_n \end{bmatrix}$$

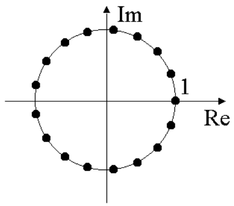
time domain

$$P_k = \sum_{i=0}^{n} p_i e^{-j\frac{2\pi k}{n+1}i}$$

direct DFT

inverse DFT

$$p_i = \frac{1}{n+1}\sum_{k=0}^{n} P_k e^{j\frac{2\pi i}{n+1}k}$$



$$\begin{bmatrix} P_0 & P_1 & \cdots & P_n \end{bmatrix}$$

frequency domain

Relation to Z-transform

For n high enough, DFT approaches Z-Transform ⇒

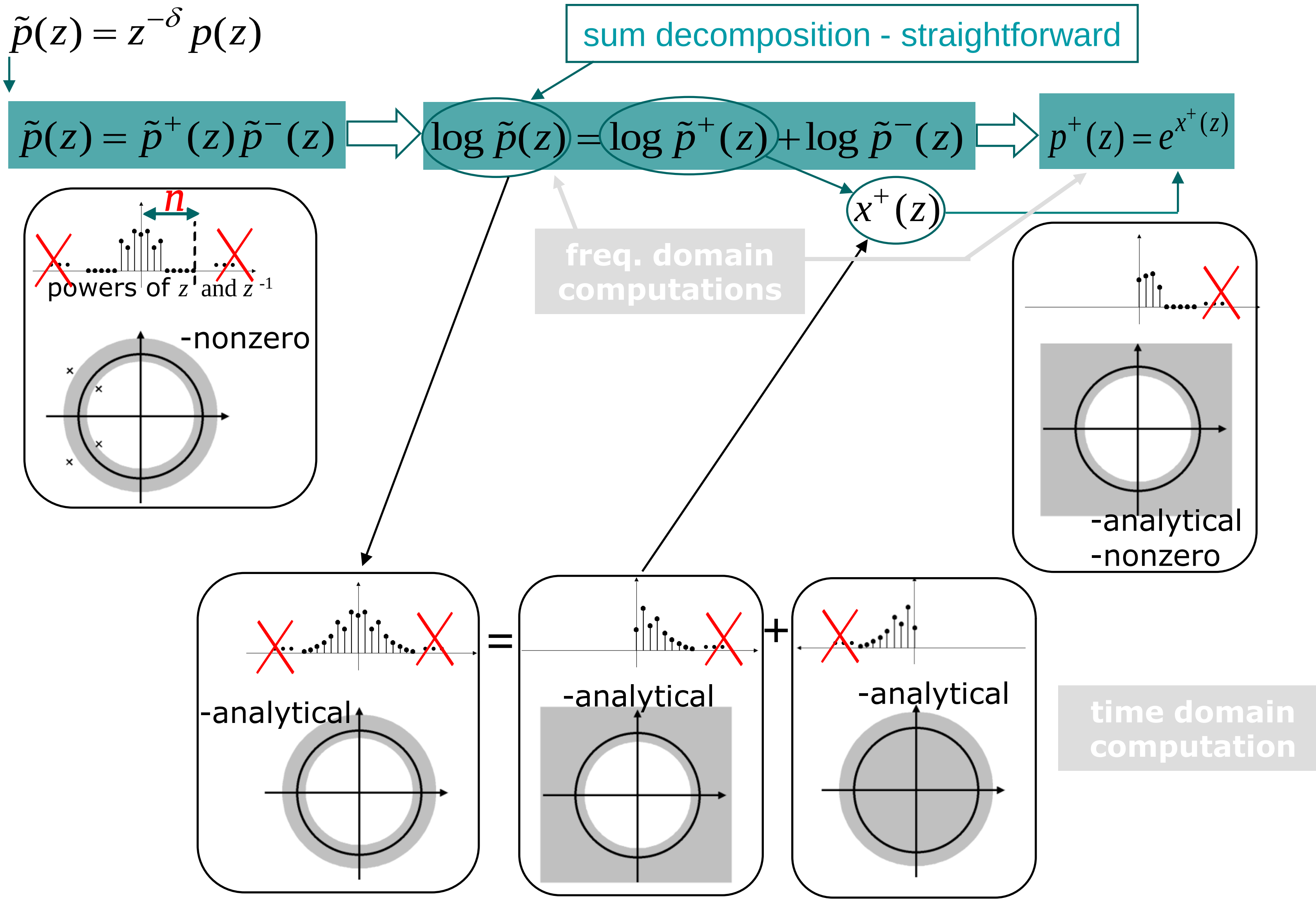
⇒ switch between time and freq.domains by DFT

Fast Fourier transform algorithm

- developed for practical computation of DFT
- Cooley & Tukey, 1970's
- numerically attractive - high effectivity - $n \log n$, numerical stability
- principle: DFT of a vector of length *n* is **recursively** decomposed into 2

  DFT's of half sizes; thanks to periodicity of $s_k = \exp j \frac{2\pi k}{n+1}$, a lot of computations are saved
- frequently used algorithm - DFT defines the spectrum of a finite or periodic discrete-time signal - often required in signal processing
- FFT algorithm(s) naturally available in many computing packages as fast built-in functions (MATLAB, Mathematica, …)

DFT & polynomial spectral factorization



DFT & polynomial spectral factorization

$\tilde{p}(z) = z^{-\delta} p(z)$

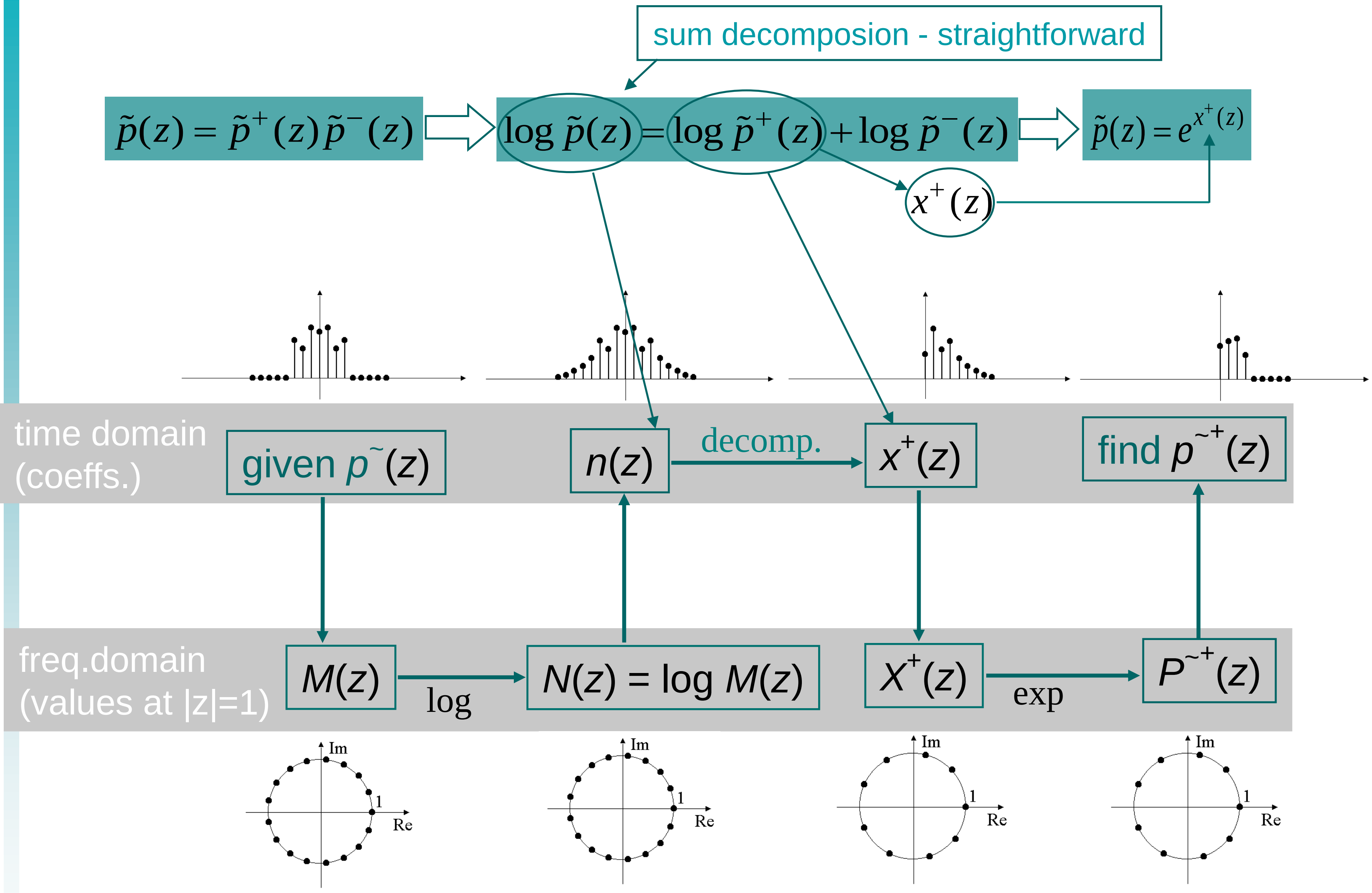
sum decomposition - straightforward

$\tilde{p}(z) = \tilde{p}^{+}(z)\tilde{p}^{-}(z)$ ⇒ $\log \tilde{p}(z) = \log \tilde{p}^{+}(z) + \log \tilde{p}^{-}(z)$ ⇒ $p^{+}(z) = e^{x^{+}(z)}$

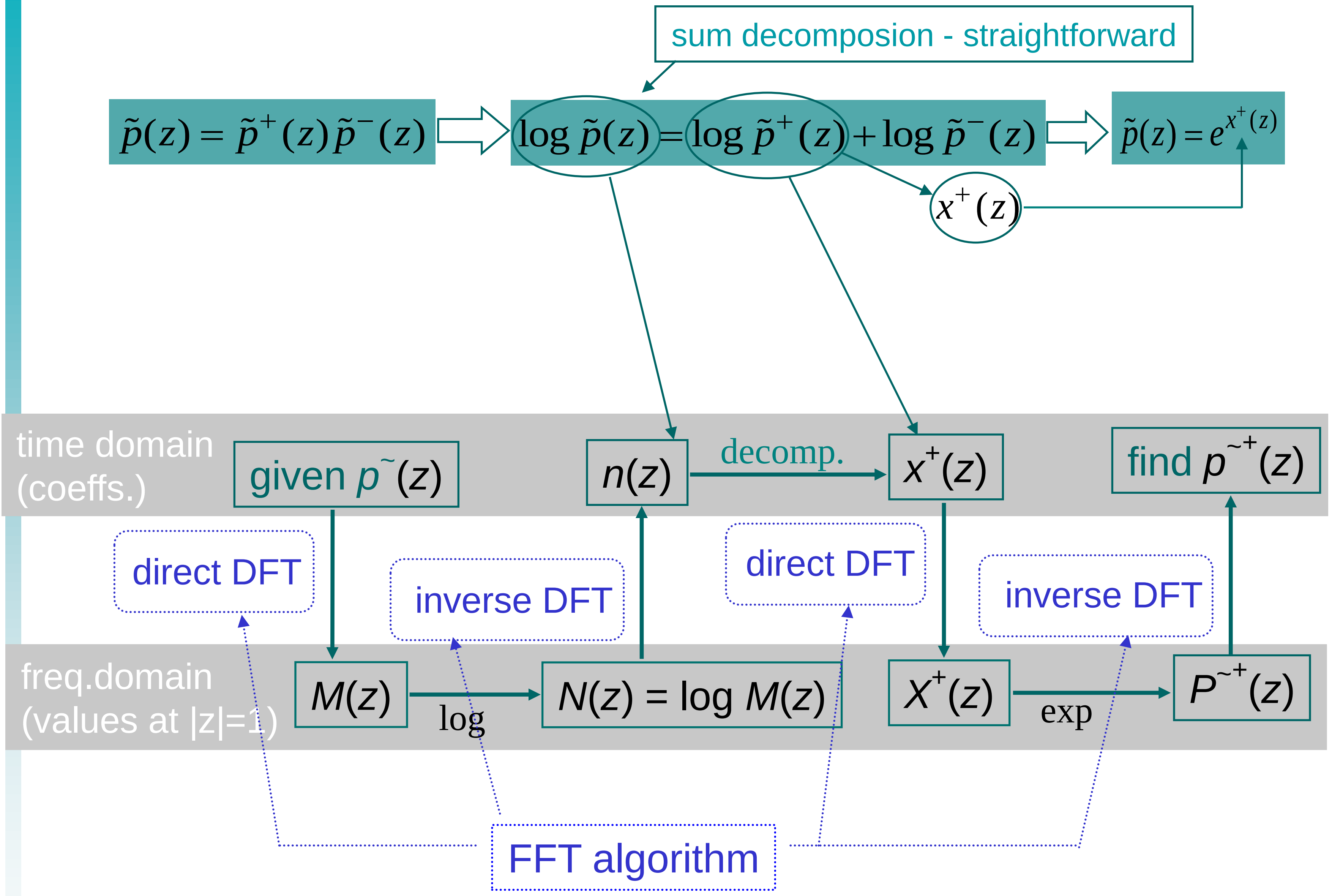
For n high enough, DFT approaches Z-Transform ⇒

⇒ switch between time and freq.domains by DFT

DFT & polynomial spectral factorization



DFT & polynomial spectral factorization

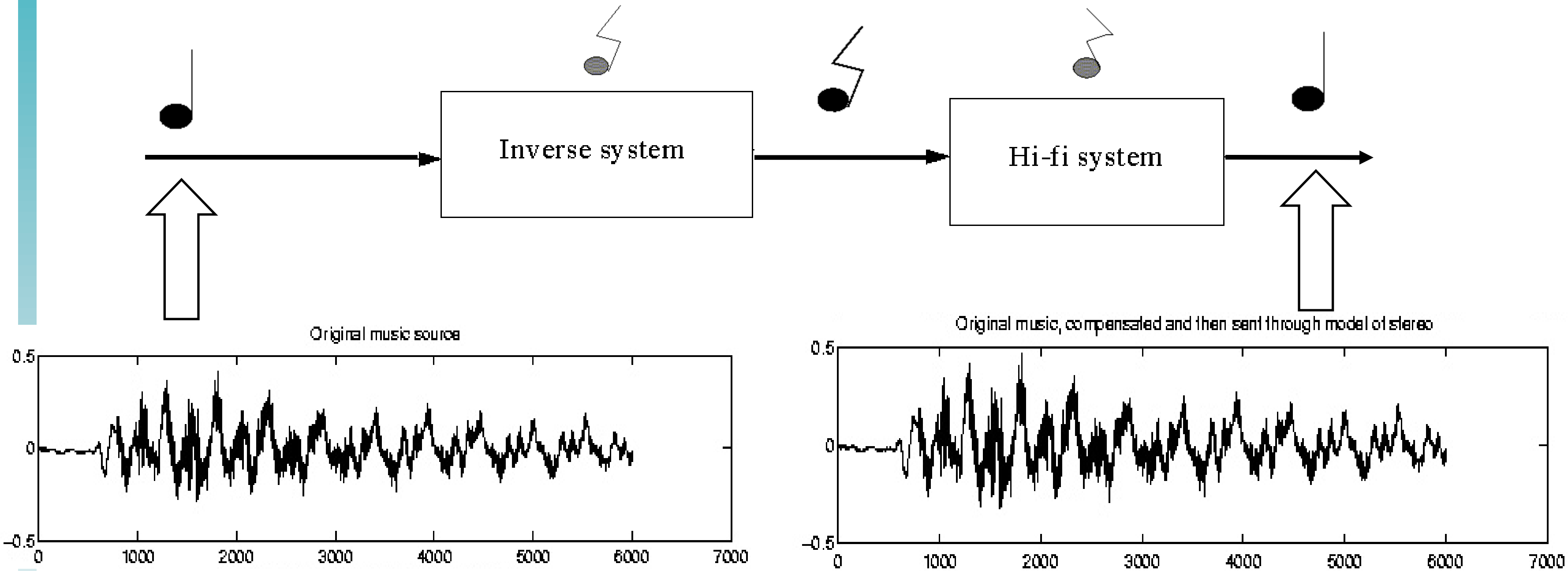


Upgrading loudspeakers dynamics

Inverse dynamics filter for moderate quality loudspeaker

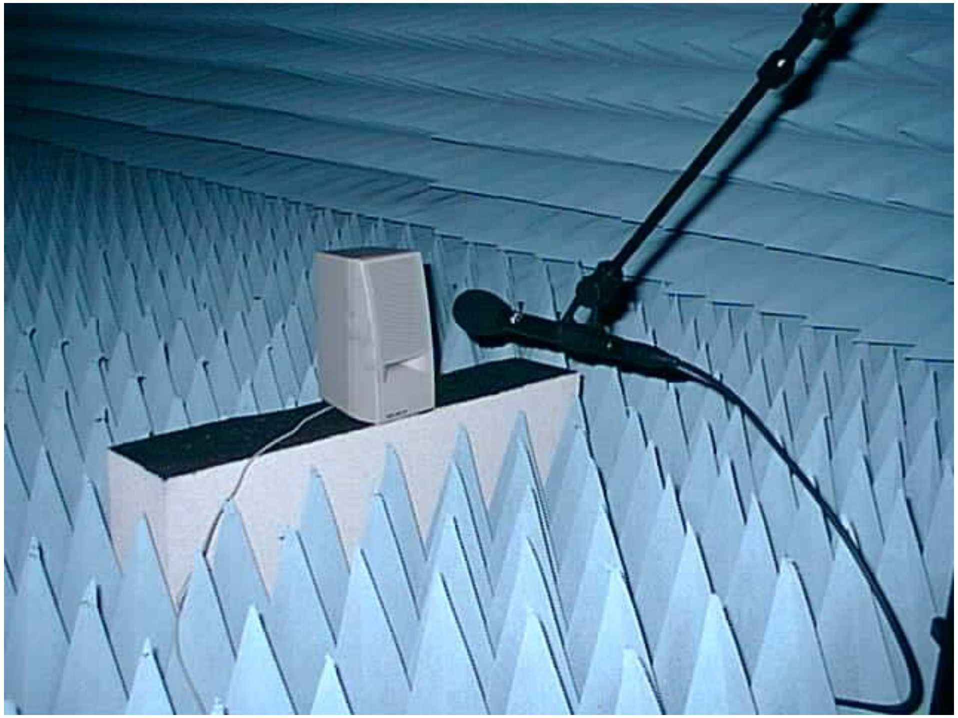
proposed and tested by Mikael Sternad and colleagues (U. of Uppsala)

http://www.signal.uu.se/Courses/Descr9899/sigproject.html



Upgrading loudspeakers dynamics

Identification: in an anechoic chamber



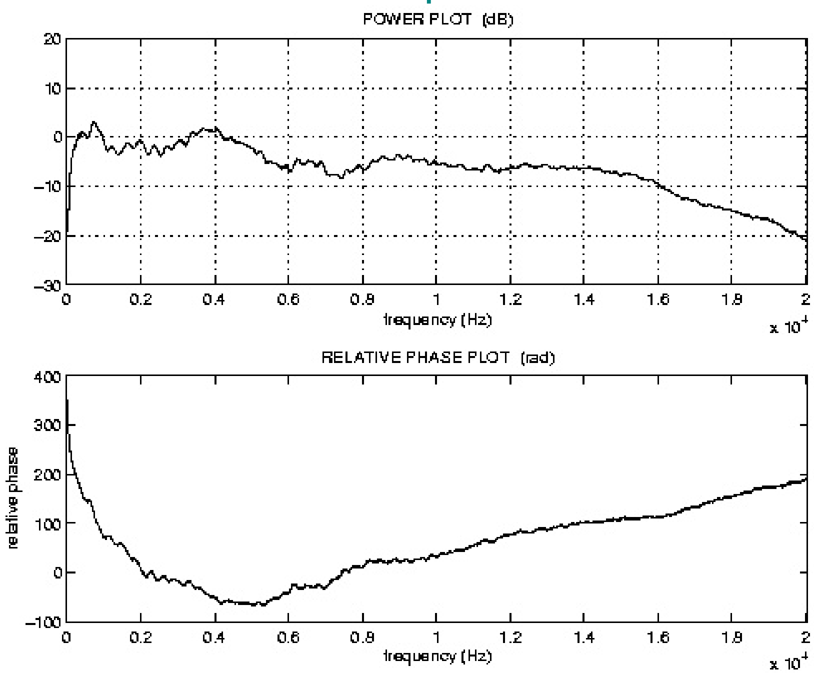
High sampling frequency (44 kHz) ⇨

⇨ **models of high orders** (200, 500, 1000)

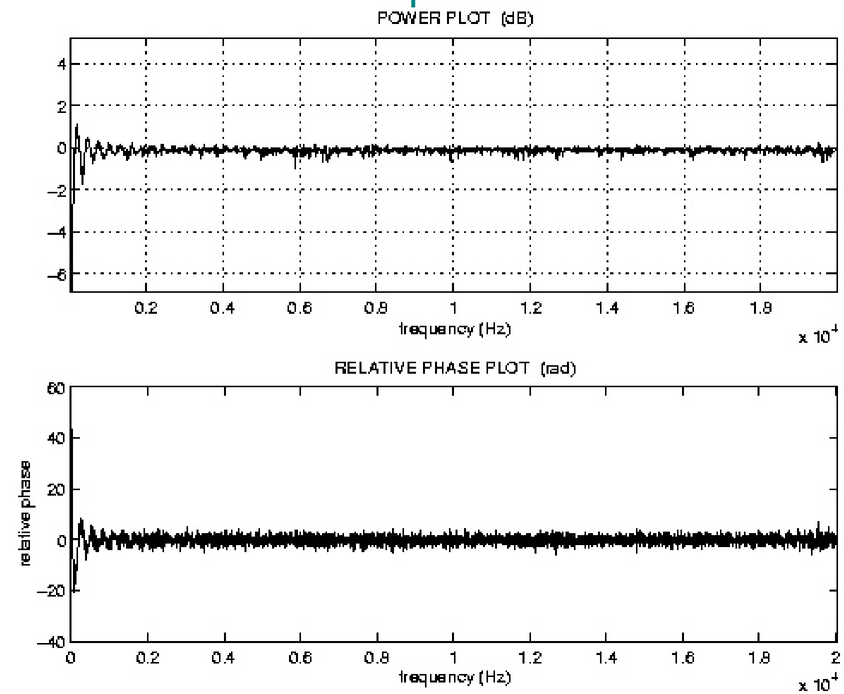
Upgrading loudspeakers dynamics

Bode plots

uncompensated



compensated



Sound examples

http://www.signal.uu.se/Courses/Descr9899/sigproject.html

Bauer-type Algorithm for Polynomial plus-minus Factorization

Martin Hromčík, Michael Šebek

Centre for Applied Cybernetics
Czech Technical University in Prague, CZ

Bauer‛s method for spectral factorisation

$$m(z) = m_{-n} z^{-n} + \ldots + m_0 + \ldots + m_n z^n$$

- symmetric
- positive definite at |z|=1

Cholesky factorization

$$\begin{bmatrix} m_0 & m_1 & \cdots & m_n & & & 0 \\ m_1 & m_0 & \ddots & & \ddots & & \\ \vdots & \ddots & \ddots & m_1 & & m_n & \\ m_n & & m_1 & m_0 & m_1 & & m_n \\ & \ddots & & m_1 & m_0 & m_1 & \vdots \\ & & m_n & & m_1 & m_0 & m_1 \\ 0 & & & m_n & \cdots & m_1 & m_0 \end{bmatrix} = F^T F$$

(n+r) by (n+r), symmetric, pos.def.

$$\begin{bmatrix} f_{11} & f_{12} & \cdots & f_{n1} & & & 0 \\ & f_{21} & \cdots & \vdots & f_{2,n+1} & & \\ & & \ddots & \vdots & \vdots & \ddots & \\ & & & f_{nn} & \vdots & & f_{ij} \\ & & & & f_{n+1,n+1} & & \vdots \\ & & & & & \ddots & \vdots \\ 0 & & & & & & f_{n+1,n+1} \end{bmatrix}$$

computed iteratively

columns converge to the spectral factor

Modification for +/- factorisation

- relax the symmetry condition
- substitute LU non-symmetric factorisation
  for symmetric Cholesky decomposition

Necessary preliminary step:

perform degree shift - construct $\tilde{p}(z) = z^{-\delta} p(z) = p_0 z^{-\delta} + \ldots + p_n z^{n-\delta}$
prior to LU factorisation, where δ is the number of stable roots
(given e.g. by the Schur stability test).

Resulting algorithm for +/- factorisation

$$\tilde{p}(z) = z^{-\delta} p(z) = p_0 z^{-\delta} + \cdots + p_\delta + \cdots + p_n z^{n-\delta}$$

$$\begin{bmatrix} p_\delta & p_{\delta-1} & \cdots & p_0 & & & 0 \\ p_{\delta+1} & p_\delta & \ddots & & \ddots & & \\ \vdots & \ddots & \ddots & p_{\delta-1} & & p_0 & \\ p_n & & p_{\delta+1} & p_\delta & p_{\delta-1} & & p_{p_0} \\ & \ddots & & p_{\delta+1} & p_\delta & p_{\delta-1} & \vdots \\ & & p_n & & p_{\delta+1} & p_\delta & p_{\delta-1} \\ 0 & & & p_n & \cdots & p_{\delta+1} & p_\delta \end{bmatrix} = LU$$

~~Cholesky~~ LU factorization

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} & & & 0 \\ & u_{22} & \cdots & \vdots & u_{2,n+1} & & \\ & & \ddots & \vdots & \vdots & \ddots & \\ & & & u_{nn} & \vdots & & u_{ij} \\ & & & & u_{n+1,n+1} & & \vdots \\ & & & & & \ddots & \vdots \\ 0 & & & & & & u_{kl} \end{bmatrix}$$

columns converge to the plus factor

Resulting algorithm - example

Command Window

File Edit View Web Window Help

```
>> p
p =
     12 + 31z + 15z^2 + 2z^3
>> T = toeplitz([31,15,2,0,0,0,0,0], [31,12,0,0,0,0,0,0])
T =
    31    12     0     0     0     0     0     0
    15    31    12     0     0     0     0     0
     2    15    31    12     0     0     0     0
     0     2    15    31    12     0     0     0
     0     0     2    15    31    12     0     0
     0     0     0     2    15    31    12     0
     0     0     0     0     2    15    31    12
     0     0     0     0     0     2    15    31
>> [L,U] = lu(T)
L =
```

$\delta = 1$

Ready

… to be continued

Resulting algorithm - example

Command Window

File Edit View Web Window Help

```
L =

  Columns 1 through 5

    1.0000         0         0         0         0
    0.4839    1.0000         0         0         0
    0.0645    0.5647    1.0000         0         0
         0    0.0794    0.5799    1.0000         0
         0         0    0.0826    0.5827    1.0000
         0         0         0    0.0832    0.5832
         0         0         0         0    0.0833
         0         0         0         0         0

  Columns 6 through 8
```

Ready

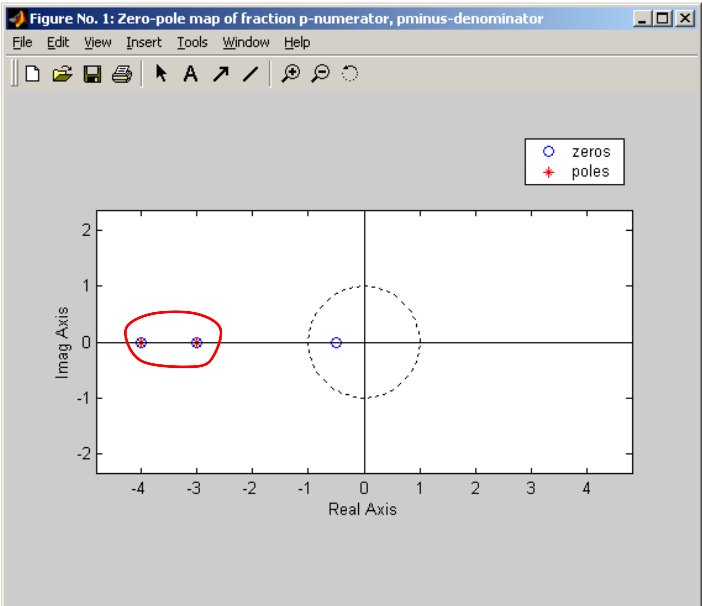
$$p^{-}(z) = 1 + 0.5832z + 0.083z^{2}$$

Resulting algorithm - example

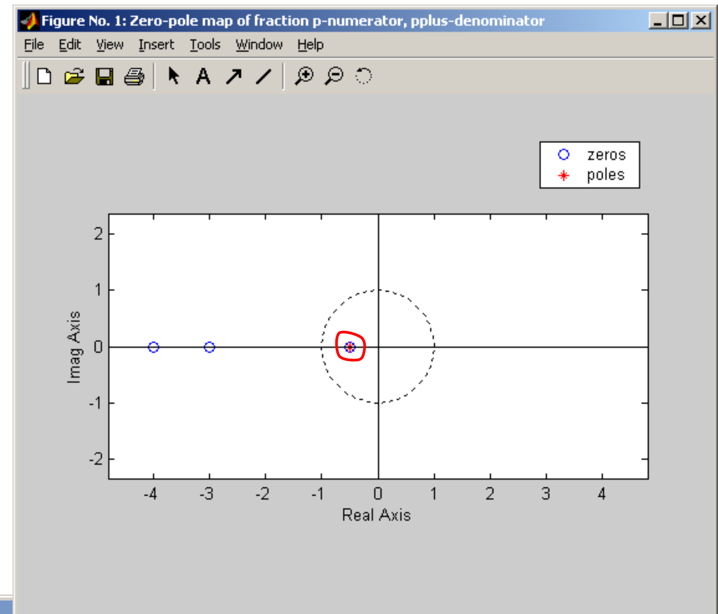
```
U =

  Columns 1 through 5

   31.0000   12.0000         0         0         0
         0   25.1935   12.0000         0         0
         0         0   24.2241   12.0000         0
         0         0         0   24.0413   12.0000
         0         0         0         0   24.0074
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0

  Columns 6 through 8
```

$$p^{+}(z) = 12 + 24.0074z$$

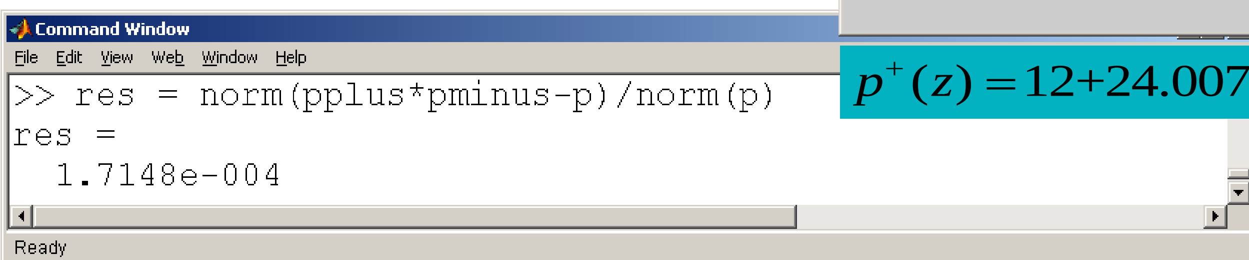
Resulting algorithm - example



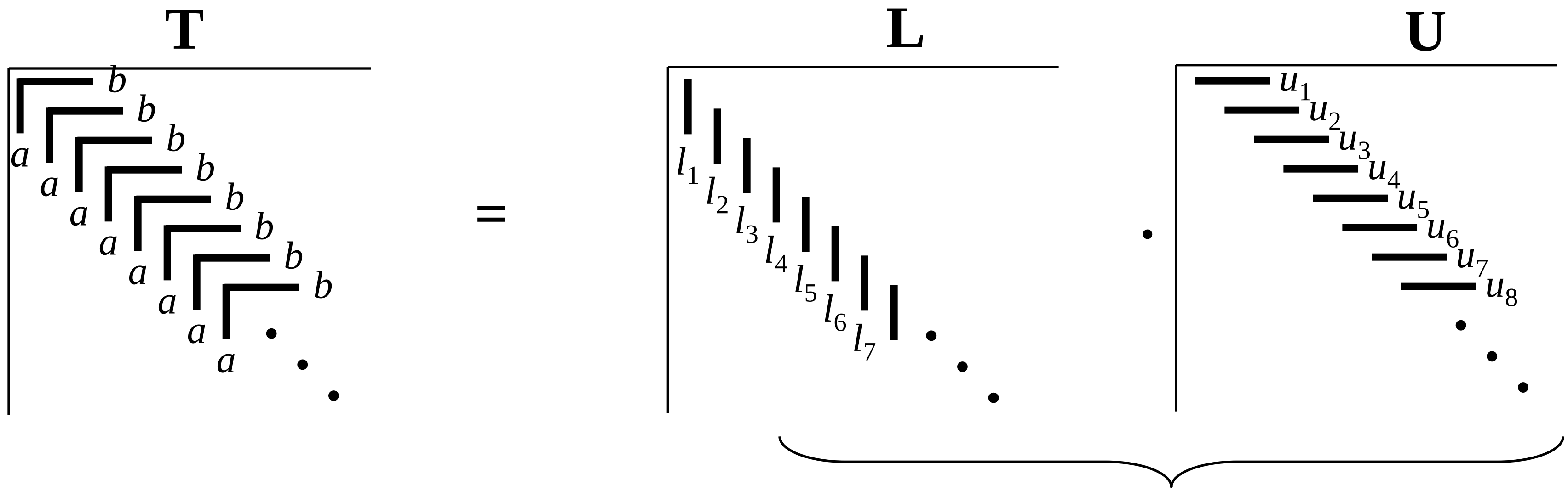
$$p^-(z) = 1 + 0.5832z + 0.083z^2$$



$$p^+(z) = 12 + 24.0074z$$



LU iterative scheme for indefinite Toeplitz matrix




```
m = length(b);
n = length(a);

X = toeplitz(a,b);
lastCol = X(1:n-1,m);
lastRow = X(n,:);

for i=1:iter_number,
    l = [1;a(2:end)./a(1)];
    u = b;

    X = [X(2:n,2:m) lastCol;lastRow];

    l_krat_u = l*u;
    X(1:n-1,1:m-1)=X(1:n-1,1:m-1)-l_krat_u(2:n,2:m);

    a = X(:,1);
    b = X(1,:);
end;
```