České vysoké učení technické
v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

Bakalářská práce

Algoritmy pro matice s Toeplitzovou strukturou

Matouš Raisigl

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Matouš Raisigl**

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný
Obor: Kybernetika a měření

Název tématu: **Algoritmy pro matice s Toeplitzovou strukturou**

Pokyny pro vypracování:

Porovnejte různé metody pro řešení základních operací s Toeplitzovými maticemi (násobení, determinant, inverze) a především pak řešení lineární soustavy s Toeplitzovou maticí (Levinson-Durbinův algoritmus a novější super rychlé algoritmy). Aplikace řešení této soustavy je využívána při výpočtu optimálního PWM problému, který je na katedře vyvíjen.
1. Nastudujte metody pro Toeplizovy matice (základní operace, Levinson-Durbinův algoritmus a super rychlé algoritmy).
2. Tyto metody naprogramujte v systému Mathematica.
3. Porovnejte rychlosti výpočtu se standardními metodami.
4. Diskutujte (porovnejte) numerickou stabilitu těchto metod.

Seznam odborné literatury:

[1] Dario Bini, Victor Y. Pan, Polynomial and Matrix Computations: Volume 1: Fundamental Algorithms, Birkhäuser Boston; 1 edition (August 1, 1994).
[2] Victor Y. Pan, Structured Matrices and Polynomials: Unified Superfast Algorithms, Birkhäuser Boston; 1 edition (June 26, 2001)

Vedoucí: Ing. Petr Kujan, Ph.D.

Platnost zadání: do konce letního semestru 2011/2012

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 6. 2. 2012

Prohlašuji, že jsem předloženou práci vypracoval samotatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, dne 25.5.2012 .....

Anotace

Cílem této bakalářské práce bylo nastudovat a naprogramovat v systému Mathematica speciální algoritmy pro operace s Toeplizovými maticemi. Toeplitzova matice je matice se speciální strukturou (konstantní prvky na diagonálách) a proto lze většinu operací s touto maticí provádět efektivněji a různými způsoby. V této práci byly konkrétně popsány a naprogramovány různé algoritmy pro násobení vektorem, výpočet determinantu, inverzní matice a řešení lineární soustavy.

Annotation

The objective of this bachelor thesis was to investigate and implement in Mathematica special algorithms for operations with Toeplitz matrices. Toeplitz matrices have a special structure (constant values along left to right diagonals) enabling matrix operations to be calculated more efficiently and several different ways. This thesis describes and implements various algorithms for vector multiplication, determinant calculation, inverse matrix calculation and linear systems solution.

Obsah

Kapitola 1

Motivace

Tato práce se zabývá řešením vybraných operací s maticemi Toeplitzova typu. Jedná se o matice se speciální strukturu, kterou lze využít pro návrh rychlejších a méně paměťově náročných algoritmů.

Jedna ze základních úloh je výpočet lineární soustavy rovnic s Toeplitzovu maticí. Tato úloha se vyskytuje v mnoha praktických aplikacích jako identifikace, lineární predikce, zpracování signálů, spektrální analýza, samoopravné kódy a další. V matematické teorii se řešení této soustavy uplatňuje při výpočtu Pádeho aproximace.

Řešení lineární soustavy rovnic s Hankelovou maticí, kterou lze snadno převést na Toeplitzovu matici (viz. úvodní kapitola), se vyskytuje v úloze optimálního PWM problému [Kujan et al., 2010], který je studován na katedře řídicí techniky zadavatelem této práce a právě efektivní metody řešení jsou intenzivně zkoumány.

Výpočetní náročnost řešení lineární soustavy rovnic s obecnou maticí je n^3. Pro systém s Toeplitzovu maticí byly navrženy rychlé algoritmy se složitostí n^2. Konkrétně se jedná o Levinsonův algoritmus a jeho modifikace. Tento algoritmus je založen na využití speciální struktury. Pro řešení lineární soustavy s Toeplitzovou maticí byly dokonce navrženy super rychlé algoritmy se složitostí $n \log^2 n$. Tyto algoritmy jsou založeny na rychlém výpočtu největšího společného dělitele dvou polynomů [Brent et al., 1980, Bini and Pan, 1994] (speciální Eukleidův algoritmus) nebo modifikovaném algoritmu pro výpočet autoregresivního filtru [Blahut, 2010] (Berlekamp-Massay algoritmus, též používaný pro bezpečnostní kódování). V obou případech se jedná o velmi komplikované algoritmy.

Kapitola 2

Úvod

V této kapitole je uvedena definice Toeplitzovy matice a matice jí velmi podobné, konkrétně Hankelova matice. Jedná se o druhou matici, která se často používá, a kterou lze převést právě na matici Toeplitzovu.

2.1 Toeplitzovy matice

Definice 2.1 Teoplitzova matice je libovolná strukturováná matice, která má konstnantí hodnoty podél hlavní diagonály. ▶

Příklad 2.1. Příklad Toeplitzovy matice

$$\begin{pmatrix} a & b & c \\ d & a & b \\ e & d & a \\ f & e & d \end{pmatrix}.$$

Díky konstantnosti hodnot na diagonále lze celou matici o rozměrech $m \times n$ zadat jen $m + n - 1$ hodnotami. Při výpočtech nejčastěji pracujeme se čtvercovou maticí rozměru $n \times n$ a zapisujeme ji ve tvaru

$$\begin{pmatrix} t_0 & t_{-1} & \dots & t_{1-n} \\ t_1 & t_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_{-1} \\ t_{n-1} & \dots & t_1 & t_0 \end{pmatrix}.$$

Tuto Toeplitzovu matici lze reprezentovat vektorem

$$\mathbf{t} = [t_{n-2}, t_{n-1}, \dots, t_{-1}, t_0, t_1, \dots, t_{n-1}, t_{n-2}].$$

Hankelovy matice

Definice 2.2 Hankelova matice je matice, která má konstantní hodnoty podél vedlejší diagonály. ▶

Příklad 2.2. Příklad Hankelovy matice

$$\begin{pmatrix} c & b & a \\ b & a & d \\ a & d & e \\ d & e & f \end{pmatrix}.$$

Hankelova matice se od Toeplitzovy matice liší jen ve směru diagonály, v jakém jsou konstantní hodnoty.

Převod mezi Hankelovou a Toeplitzovou maticí

Definice 2.3 Reflexní matice je čtvercová matice, která má prvky na vedlejší diagonále rovny jedné, ostatní jsou nulové. ▶

Příklad 2.3. Příklad reflexní matice rozměru 3×3

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}. \tag{2.1}$$

Reflexní matice má následující vlastnosti

$$\mathbf{J} = \mathbf{J}^{-1} \tag{2.2}$$

a

$$\mathbf{J} = \mathbf{J}^{T}. \tag{2.3}$$

Tedy reflexní matice je sama sobě inverzí a symetrická. Dále reflexní matice obrací pořadí prvků ve vektoru

$$\mathbf{J} \cdot \begin{pmatrix} a_1 \\ \vdots \\ a_k \end{pmatrix} = \begin{pmatrix} a_k \\ \vdots \\ a_1 \end{pmatrix},$$

$$\mathbf{J} \cdot \begin{pmatrix} a_1 & \dots & a_k \end{pmatrix} = \begin{pmatrix} a_k & \dots & a_1 \end{pmatrix}. \tag{2.4}$$

Tvrzení 2.1. *Toeplitzova matice je je persymmetrická*

$$E_k T_k = T_k^T E_k. \tag{2.5}$$

Důkaz tohoto tvrzení je následující.

Důkaz:

$$E_k T_k = \left(E_k \begin{pmatrix} t_0 \\ t_1 \\ \vdots \\ t_{k-1} \end{pmatrix}, \quad E_k \begin{pmatrix} t_{-1} \\ t_0 \\ \vdots \\ t_{k-2} \end{pmatrix}, \quad \ldots, \quad E_k \begin{pmatrix} t_{-k+1} \\ t_{-k+2} \\ \vdots \\ t_0 \end{pmatrix} \right) =$$

$$= \left(\begin{pmatrix} t_{k-1} \\ \vdots \\ t_1 \\ t_0 \end{pmatrix}, \quad \begin{pmatrix} t_{k-2} \\ \vdots \\ t_0 \\ t_{-1} \end{pmatrix}, \quad \ldots, \quad \begin{pmatrix} t_0 \\ \vdots \\ t_{-k+2} \\ t_{-k+1} \end{pmatrix} \right)^{T^T} \cdot E_k^{T^T} E_k =$$

$$= \left(E_k^T \cdot \left(\begin{pmatrix} t_{k-1} \\ t_{k-2} \\ \vdots \\ t_0 \end{pmatrix}, \quad \ldots, \quad \begin{pmatrix} t_1 \\ t_0 \\ \vdots \\ t_{-k+2} \end{pmatrix}, \quad \begin{pmatrix} t_0 \\ t_{-1} \\ \vdots \\ t_{-k+1} \end{pmatrix} \right) \right)^T E_k =$$

$$= \left(\begin{pmatrix} t_0 \\ \vdots \\ t_{k-2} \\ t_{k-1} \end{pmatrix}, \quad \ldots, \quad \begin{pmatrix} t_{-k+2} \\ \vdots \\ t_0 \\ t_1 \end{pmatrix}, \quad \begin{pmatrix} t_{-k+1} \\ \vdots \\ t_{-1} \\ t_0 \end{pmatrix} \right)^T E_k = T_k^T E_k. \qquad \square$$

Pro inverzi platí

$$E_k T_k^{-1} = T_k^{-T} E_k. \tag{2.6}$$

Díky těmto vlastnostem, můžeme převádět mezi Toeplitzovou a Hankelovou maticí. Některé vlastnosti využijeme i v algoritmech uvedených dále.

Tvrzení 2.2. *Mějme Hankelovu matici* $\boldsymbol{H}$*, Toeplitzovu matici* $\boldsymbol{T}$ *a reflexní matici* $\boldsymbol{J}$*. Poté platí následující tvrzení*

$$\boldsymbol{T} = \boldsymbol{HJ}, \boldsymbol{H} = \boldsymbol{TJ},$$
$$\boldsymbol{T} = \boldsymbol{JH}, \boldsymbol{H} = \boldsymbol{JT}.$$

Důkaz: Mějme $\mathbf{T} = \mathbf{HJ}$, dosadíme za matici $\mathbf{H} = \mathbf{TJ}$, čímž dostaneme $\mathbf{T} = \mathbf{TJJ}$. Díky rovnosti $\mathbf{J} = \mathbf{J}^{-1}$ dostaneme $\mathbf{T} = \mathbf{T}$.
Důkaz druhého tvrzení je obdobný. $\square$

Poznámka 2.1. Při převodu matic musíme pamatovat na to z jaké strany jsme násobili Hankelovu/Toeplitzovu matici. Pokud by jsme na to zapomněli a násobili jí z druhé strany rovnost by neplatila. ■

V následujících kapitolách rozebereme různé algoritmy, které využívají Toeplitzovy matice. Pomocí polynomů ukážeme rychlé násobení Toeplitzovy matice vektorem. Dále ukážeme rychlý výpočet determinantu. Budeme se také věnovat výpočtu inverzní matice k Toeplitzově matici a nakonec probereme několik způsobů řešení soustavy lineárních rovnic.

Kapitola 3

Násobení Toeplitzovy matice vektorem

V této kapitole popíšeme rychlý algoritmus pro násobení Toeplitzovy matice vektorem. Jeho náročnost je $n \log(n)$. Tento algoritmus je založen na rychlém násobení polynomů [Pan, 2001, kap. 2.4].

Pokud si postup pro násobení polynomů rozepíšeme pomocí matic, dostaneme násobení matice vektorem. Tato matice bude Toeplitzova. Postup pro násobení polynomů je jednoduchý, každý koeficient levého polynomu násobíme každým koeficientem pravého polynomu. Vektor tvoří koeficienty pravého polynomu. Matici vytvoříme z koeficientů levého polynomu. Počet sloupců matice je dán stupněm výsledného polynomu, jež se rovná součtu stupňů obou polynomů. Do prvního sloupce matice napíšeme koeficienty polynomu pod sebe a spodek doplníme nulami. Do druhého sloupce napíšeme také koeficienty levého polynomu, ale posuneme je o jeden řádek dolů a do prázdného místa napíšeme nulu. Po zapsání všech sloupců by v prvním sloupci měli být koeficienty polynomu nahoře a v posledním sloupci dole. Poznamenejme, že tato matice se nazývá Sylvesterova matice.

Příklad 3.1. Zde je rozepsáno násobení dvou polynomů standardním i maticovým zápisem

$$(a_1 x + a_0) \cdot (b_1 x + b_0) = a_1 b_1 x^2 + (a_1 b_0 + a_0 b_1)\, x + a_0 b_0,$$

$$\begin{pmatrix} a_1 & 0 \\ a_0 & a_1 \\ 0 & a_0 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} a_1 b_1 & (x^2) \\ a_0 b_1 + a_1 b_0 & (x) \\ a_0 b_0 & (1) \end{pmatrix}.$$

Počítejme tedy rovnici $\mathbf{w} = \mathbf{U}\mathbf{v}$, kde $\mathbf{U}$ je Sylvesterova matice (speciální Toplitzova matice - nuly v rozích), jejíž vytvoření jsme popsali výše, a $\mathbf{w}$, $\mathbf{v}$

jsou vektory

$$\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ \\ \\ \\ \\ \vdots \\ w_{m+n} \end{pmatrix} = \begin{pmatrix} u_0 & & & 0 \\ u_1 & \ddots & & \\ \vdots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & u_0 \\ \vdots & \ddots & \ddots & u_1 \\ u_m & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \vdots \\ & & \ddots & \vdots \\ 0 & & & u_m \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_n \end{pmatrix}. \tag{3.1}$$

Tato rovnice je ekvivalentní násobení polynomů, jak jsme si již ukázali. Pravému polynomu odpovívá vektor **v** a druhému Sylvesterova matice. Tento maticocý zápis lze využít k výraznému urychlení násobení dvou polynomů, resp. násobení Toeplitzovy matice a vektoru.

Sylvesterova matice v rovnici (3.1) je speciální Toeplitzova matice, protože má nuly v rozích. Střed této matice od $(n+1)$ až po $(2n+1)$ řádek je však obecná Toeplitzova matice. Díky tomu lze tento algoritmus vhodným rozšířením použít pro jakoukoli matici.

Tedy náspobení dvou polynomů je ekvivalentní násobení Toeplitzovy matice vektorem. Tedy pokud umíme efektivně násobit dva polynomy, pak umíme i efektivně násobit Toeplizovu matici vektorem. Jeden efektivní algoritmus pro násobení dvou polynomů je uveden v následující kapitole.

3.1 Násobení polynomů

Z teorie o polynomech známe rychlé metody pro operace s těmito objekty.

Mějme polynomy u a v o stupních m a n a spočtěme výsledný polynom p vzniklý jejich pronásobením. DFT_N a $iDFT_N$ značí diskrétní fourierovu transformaci a inverzní transformaci délky N. Volné místo doplníme nulami.

Tento rychlý algoritmus může být použit pro výpočet násobku několika polynomů, nebo pro násobení polynomů s více proměnnými. V tom případě však nemá žádný vztah s Toeplitzovými maticemi a přepsání tohoto násobení do maticové rovnice vede k obecným maticím.

Polynom stupně n, je pevně určen hodnotami v $n+1$ bodech, [Olšák, 2008, věta 11.11]. Pokud chceme spočítat hodnotu výsledného polynomu v nějakém

bodě, můžeme tuto hodnotu spočíst výpočtem hodnot obou polynomů v daném bodě a jejich pronásobením. Tedy $p(x) = u(x)v(x)$. Pro přesné určení výsledného polynomu tedy musíme spočítat hodnoty v $m + n + 1$ bodech.

Na Diskrétní Fourierovu transformaci se lze dívat jako na výpočet hodnoty polynomu v bodech na jednotkové kružnici, kde koeficienty polynomu jsou transformované hodnoty. Jeden člen transformace se vypočítá takto

$$D(n) = \sum_{k=0}^{N-1} d(k)e^{(in2\pi/N)k} = \sum_{k=0}^{N-1} d(k)\omega^k, \quad \omega = e^{in2\pi/N}.$$

Jedná se tedy o výpočet polynomu v konkrétním bodě.

Na výpočet diskrétní Fourierovy transformace můžeme také hledět jako na výpočet maticové rovnice $\mathbf{P} = \Omega\mathbf{p}$, kde $\mathbf{p}$ a $\mathbf{P}$ jsou vektory obsahující transformované a původní hodnoty a Ω je transformační matice definovaná jako

$$\Omega = \frac{1}{\sqrt{N}} \left(e^{-i2\pi(k)(l)/N}\right),$$

N označuje délku vektoru p, $k = 0, 1, \ldots, n-1$ a $l = 0, 1, \ldots, m-1$ označují pozici v matici. Inverzní Fourierovu transrofmaci můžeme počítat jako $\mathbf{p} = \Omega^{-1}\mathbf{P}$, kde Ω^{-1} je matice inverzní k Ω. Díky tomu můžeme vypočítat koeficienty polynomu z hodnot získaných výpočtem diskrétní Fourierovy transformace. V předchozím odstavci jsme ukázali, že diskrétní Fourierova transformace vypočítá hodnoty polynomu v daných bodech, tak inverzní transformace vypočítá koeficienty tohoto polynomu. Protože platí $\Omega\Omega^{-1} = E, \Omega\mathbf{p} = \mathbf{P}, \Omega^{-1}\Omega\mathbf{p} = \Omega^{-1}\mathbf{P}, \mathbf{p} = \Omega^{-1}\mathbf{P}$.

V tomto algoritmu tedy pomocí Fourierovy transformace vypočítáme hodnoty obou polynomů v $n + 1$ bodech na jednotkové kružnici. Výsledné hodnoty v těchto bodech pronásobíme, čímž dostaneme hodnoty výsledného polynomu. Pomocí inverzní Fourierovy transrofmace dostaneme koeficienty výsledného polynomu z jeho hodnot daných v $n + 1$ bodech.

Jedná se tedy o interpolaci polynomů na jednotkové kružnici. To odpovídá výpočtu DFT. Obrácenému postupu odpovídá IDTF.

Právě popsaný algoritmus je shrnut v Algoritmu 1.

Algoritmus 1 Rychlé násobení polynomů

Vstup: Vektor koeficientů polynomů u a v.

Výstup: Vektor koeficientů výsledného polynomu $p = u \cdot v$.

1: $h = \lceil \log_2(\deg(u) + \deg(v) + 1) \rceil,\ N = 2^h$
2: $U = DFT_N(u), V = DFT_N(v)$
3: $P_j = U_j V_j, j = 0, 1, \ldots, N-1$
4: $p = iDFT_N(P)$

Kapitola 4

Determinant Toeplitzovy matice

Nyní se budeme věnovat výpočtu determinantu obecné Toeplitzovy matice pomocí Levinson-Durbinova algoritmu, jehož počet operací bude $O(n^2)$. Pro více informací viz. [Golub and Van Loan, 1996].

4.1 Levinson-Durbinův algoritmus pro determinant

Toeplitzovu matici si zapíšeme v blokovém zápise

$$T_{k+1} = \begin{pmatrix} T_k & J_k v_k \\ u_k^T J_k & t_0 \end{pmatrix}, \tag{4.1}$$

kde

$$v_k = \begin{pmatrix} t_{-1} \\ \vdots \\ t_{-k} \end{pmatrix}, u_k = \begin{pmatrix} t_1 \\ \vdots \\ t_k \end{pmatrix}.$$

J_k je reflexní matice (2.1) o rozměru $k \times k$

$$\begin{pmatrix} 0 & & 1 \\ & \iddots & \\ 1 & & 0 \end{pmatrix}.$$

4.1.1 Schurův doplněk

V tomto algoritmu se využívá pro výpočet Schurův doplněk. Pro blokovou matici, která je definována jako

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}.$$

Potom Schurův doplněk pro blok A je definován jako

$$D - CA^{-1}B.$$

Pro determinant Schurova doplňku platí následující tvrzení

$$\det \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \det A \cdot \det(D - CA^{-1}B). \tag{4.2}$$

4.1.2 Rekurzivní výpočet determinantu

Aplikováním rovnice (4.2) na matici (4.1) dostaneme

$$\det T_{k+1} = \det T_k \cdot \det(t_0 - u_k^T J_k T_k^{-1} J_k v_k) = \det T_k \cdot (t_0 - v_k^T T_k^{-1} u_k).$$

Poslední úpravu získáme následným aplikováním vlastností reflexní matice

$$u_k^T E_k T_k^{-1} E_k v_k \overset{(2.2)}{=} u_l^T T_k^{-T} E_k E_k v_k \overset{(2.6)}{=} (T_k^{-1} u_k)^T v_k = v_k^T T_k^{-1} u_k.$$

Poslední úprava platí protože výsledkem je skalár a díky tomu můžeme použít následující rovnost

$$\left(T_k^{-1} u_v\right) v_k = \left(\left(T_k^{-1} u_k\right)^T v_k\right)^T = v_k^T T_k^{-1} u_k.$$

Determinant Teoplitzovy matice T_n spočteme rekurzí

$$\det T^{k+1} = \det T_k \alpha_k, \quad k = 1, \ldots, n-1,$$

kde

$$\alpha_k = t_0 - v_k^T T_k^{-1} u_k. \tag{4.3}$$

Budeme rekurzivně počítat determinanty matic, dostaneme
pro $k = 1, 2, 3, \ldots, n-1$

$$\begin{aligned}
\det T_1 &= \det\left(t_0\right) = t_0, \\
\det T_2 &= \det T_1 \alpha_1 = t_0(t_0 - v_1^T T_1^{-1} u_1) = t_0(t_0 - t_{-1} t_0^{-1} t_1), \\
\det T_3 &= \det T_2 \alpha_2 = \det T_2 (t_0 - v_1^T T_1^{-1} u_2) \\
&= t_0 \left(t_0 - t_{-1} t_0^{-1} t_1\right) \left(t_0 - \begin{pmatrix} t_{-1} & t_{-2} \end{pmatrix} \begin{pmatrix} t_0 & t_{-1} \\ t_1 & t_0 \end{pmatrix}^{-1} \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}\right), \\
&\vdots, \\
\det T_n &= \det T_{n-1} \alpha_{n-1} = t_0 \prod_{k=1}^{n-1} \alpha_k.
\end{aligned}$$

Problém algoritmu spočívá ve výpočtu inverzní matice ve výrazu pro $\alpha_k,\ k = 2, \ldots, n-1$, který je složitý při standardním postupu výpočtu.

4.1.3 Výpočet inverze ve výrazu α_k

Pro výraz $\alpha = t_0 - v_k^T T^{-1} u_k$ zavedeme proměnné

$$x_k = T_k^{-1} u_k, \tag{4.4}$$
$$y_k = T_k^{-T} v_k. \tag{4.5}$$

Potom platí

$$x_{k+1} = T_{k+1}^{-1} u_{k+1},$$
$$y_{k+1} = T_{k+1}^{-T} v_{k+1}.$$

Nyní x_{k+1}, y_{k+1} budu hledat ve tvaru

$$x_{k+1} = \begin{pmatrix} x_k - x \\ \gamma_{k+1} \end{pmatrix},\ y_{k+1} = \begin{pmatrix} y_k - y \\ \omega_{k+1} \end{pmatrix}.$$

Tedy pro x_{k+1}

$$\begin{pmatrix} x_k - x \\ \gamma_{k+1} \end{pmatrix} = \begin{pmatrix} T_k & E_k v_k \\ u_k^T E_k & t_0 \end{pmatrix}^{-1} \cdot \begin{pmatrix} u_k \\ t_{k+1} \end{pmatrix},$$
$$\begin{pmatrix} T_k & E_k v_k \\ u_k^T E_k & t_0 \end{pmatrix} \cdot \begin{pmatrix} x_k - x \\ \gamma_{k+1} \end{pmatrix} = \begin{pmatrix} u_k \\ t_{k+1} \end{pmatrix}.$$

Po roznásobení předchozí maticové rovnice dostaneme dvě rovnice o dvou neznámých x a γ_{k+1}

$$T_k x_k - T_k x + E_k v_k \gamma_{k+1} = u_k, \tag{4.6}$$

$$u_k^T E_k x_k - u_k^T E_k x + t_0 \gamma_{k+1} = t_{k+1}. \tag{4.7}$$

Z rovnice (4.6) a úprav, které jsou označeny nad rovností, spočteme

$$x = T_k^{-1} T_k x_k + T_k^{-1} E_k v_k \gamma_{k+1} - T_k^{-1} u_k \overset{(4.4)}{=} x_k + T_k^{-1} E_k v_k \gamma_{k+1} - x_k = T_k^{-1} E_k v_k \gamma_{k+1}.$$

Výraz

$$T_k^{-1} E_k v_k \overset{(2.2)}{=} E_k E_k T_k^{-1} E_k v_k \overset{(2.6)}{=} E_k T_k^{-T} E_k E_k v_k = E_k T_k^{-T} v_k \overset{(4.5)}{=} E_k y_k,$$

tedy

$$x = \gamma_{k+1} E_k y_k. \tag{4.8}$$

Z rovnice (4.7) a po dosazení (4.8) spočteme

$$y_{k+1} = \frac{t_{k+1} - u_k^T E_k x_k}{t_0 - u_k^T y_k} = \frac{t_{k+1} - u_k^T E_k x_k}{\alpha_k}.$$

Protože

$$\begin{aligned} t_0 - u_k^T y_k \overset{(4.5)}{=} t_0 - u_k^T T_k^{-T} v_k &= t_0 - \left(\left(T_k^{-1} u_k \right)^T v_k \right)^{T^T} \\ &= t_0 - (v_k^T T_k^{-1} u_k)^T = t_0 - v_k^T T_k^{-1} u_k = \alpha_k. \end{aligned}$$

Podobně dostaneme pro y_{k+1}

$$\begin{aligned} y &= \omega_{k+1} E_k x_k, \\ \omega_{k+1} &= \frac{t_{-k-1} - v_k^T E_k y_k}{\alpha_k}. \end{aligned}$$

4.1.4 Rekurze α_{k+1}

Nyní spočteme rekurzivní vztah pro α_k, pro které platí (4.3). Po aplikaci (4.4) dostaneme

$$\alpha_k = t_0 - v_k^T x_k.$$

Pro $k+1$ platí

$$\begin{aligned} \alpha_{k+1} &= t_0 - v_{k+1}^T x_{k+1} = t_0 - \begin{pmatrix} v_k^T & t_{-k-1} \end{pmatrix} \begin{pmatrix} x_k - \gamma_{k+1} E_k y_k \\ \gamma_{k+1} \end{pmatrix} = \\ &= t_0 - v_k^T x_k + v_k^T \gamma_{k+1} E_k y_k - t_{-k-1} \gamma_{k+1} = \\ &= t_0 - v_k^T x_k - \gamma_{k+1} \frac{-v_k^T E_k y_k + t_{-k-1}}{\alpha_k} \alpha_k = \\ &= a_k \left(\frac{t_0 - v_k^T x_k}{\alpha_k} - \gamma_{k+1} \omega_{k+1} \right) = \\ &= a_k \left(1 - \gamma_{k+1} \omega_{k+1} \right). \end{aligned}$$

Takto popsaný algoritmus je shrnut v Algoritmu 2.

Algoritmus 2 Výpočet determinantu Toeplitzovy matice

Vstup: Toeplitzova matice T_n o rozměrech $n \times n$.

Výstup: $\det T_n$

1: $v_n = \begin{pmatrix} t_{-1} & \dots & t_{-n} \end{pmatrix}^T$

2: $u_n = \begin{pmatrix} t_1 & \dots & t_n \end{pmatrix}^T$

3: $\alpha_1 = t_0 - v_1 t_0^{-1} u_1$

4: $x_1 = t_1^{-1} u_1$

5: $y_1 = t_1^{-1} v_1$

6: **for** $k = 1, \dots, n-2$ **do**

7: $\quad \gamma_{k+1} = \frac{t_{k+1} - u_k^T E_k x_k}{t_0 - u_k^T y_k} = \frac{t_{k+1} - u_k^T E_k x_k}{\alpha_k}$

8: $\quad \omega_{k+1} = \frac{t_{-k-1} - v_k^T E_k y_k}{\alpha_k}$

9: $\quad a_{k+1} = a_k \left(1 - \gamma_{k+1} \omega_{k+1}\right)$

10: $\quad x_{k+1} = \begin{pmatrix} x_k - \gamma_{k+1} E_k y_k \\ \gamma_{k+1} \end{pmatrix}$

11: $\quad y_{k+1} = \begin{pmatrix} y_k - \omega_{k+1} E_k x_k \\ \omega_{k+1} \end{pmatrix}$

12: **end for**

13: $\det T_n = t_0 \prod_{k=1}^{n-1} \alpha_k$

Kapitola 5

Inverzní matice k Toeplitzově matici

V této části popíšeme nejprve postup pro výpočet inverze dolní trojúhleníkové matice [Pan, 2001, kap. 2.5] a poté algoritmus pro výpočet inverzní matice a současně determinantu obecné Toeplitzovy matice [Pan, 2001, kap. 5.2].

Inverzní matice, k čtvercové matici $\mathbf{A}$ je definována jako $\mathbf{A}\mathbf{A}^{-1} = \mathbf{E}$. Poznamenejme, že v případě Toeplitzových matic, nemusí být (a často není) inverzní matice znovu Toeplitzova matice, ale je vždy persymetrická.

5.1 Inverzní matice k dolní trojúhelníkové Toeplitzově matici

Tento algoritmus patří do kategorie rozděl a panuj. Pokud máme regulární dolní trojúhelníkovou Toeplitzovu matici $n \times n$

$$\mathbf{T} = \begin{pmatrix} t_0 & 0 & \dots & \mathbf{0} \\ t_1 & t_0 & 0 & \vdots \\ \vdots & t_1 & t_0 & 0 \\ t_n & \dots & t_1 & t_0 \end{pmatrix}.$$

Prvek t_0 musí být rozdílný od 0. Pokud by to neplatilo, matice nebude regulární a nebude mít tedy inverzní matici. Pro tuto matici definujeme inverzní matici $\mathbf{Y}$

$$Y = \begin{pmatrix} T & 0 \\ T_0 & T_1 \end{pmatrix}^{-1} = \begin{pmatrix} T^{-1} & 0 \\ -T_1^{-1}T_0T^{-1} & T_1^{-1} \end{pmatrix}.$$

V této matici je submatice T_0 obecná Toeplitzova matice, T a T_1 jsou dolní trojúhleníkové Toeplitzovy matice. Rozměry matice T jsou $\lceil K/2 \rceil \times \lceil K/2 \rceil$ a T_1 má velikost $\lfloor K/2 \rfloor \times \lfloor K/2 \rfloor$. Pro sudé K platí rovnost $T = T_1$. To zjednoduší výpočet inverzní matice, protože $T^{-1} = T_1^{-1}$. Pokud je K liché, tak T_1 je submaticí T a T_1^{-1} je stejnou submaticí T^{-1}. Takže v obou případech stačí spočítat inverzní matici pro T, na tu lze opakovaně použít tento algoritmus dokud nebude matice T velikosti 1×1. Tato matice je rovna prostému číslu a není problém pro ni spočítat inverzní matici, kdy se jedná o prostou obrácenou hodnotu čísla. Zpětně tak mohu dopočítat celou inverzní matici

$$\begin{pmatrix} t_0 & 0 & \dots & \dots & 0 \\ \vdots & t_0 & 0 & \ddots & \vdots \\ t_n & \ddots & t_0 & 0 & \vdots \\ 0 & t_n & \ddots & t_0 & 0 \\ 0 & 0 & t_n & \dots & t_0 \end{pmatrix}.$$

Tento algoritmus je shrnut v Algoritmu 3.

Algoritmus 3 Inverzní matice k dolní trojúhelníkové Toeplitzovy matici

Vstup: Toeplitzova regulární dolní trojúhelníková matice M o rozměrech $n \times n$.

Výstup: Inverzní matice M^{-1}.

1: **if** Rozměr $M = 1 \times 1$ **then**
2: $M^{-1} = \frac{1}{M(0,0)}$
3: **else**
4: $M = \begin{pmatrix} T & 0 \\ T_0 & T_1 \end{pmatrix}$ Tak aby platilo pro rozměr $T\lceil n/2 \rceil \times \lceil n/2 \rceil, T_1 \lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$.
5: Vypočteme inverzní matici pro T, pomocí tohoto algoritmu, dostaneme T^{-1}.
6: Z matice T^{-1} vezmemu submatici o rozměrech T_1 a tím získáme matici T_1^{-1}.
7: $M^{-1} = \begin{pmatrix} T^{-1} & 0 \\ -T_1^{-1} T_0 T^{-1} & T_1^{-1} \end{pmatrix}$
8: **end if**

5.2 Inverze obecné Toeplitzovy matice

Pomocí tohoto algoritmu můžeme najednou spočítat determinant inverzní matice i samotnou inverzní matici.

Obecnou regulární matici $n \times n$ můžeme rozložit na

$$M = \begin{pmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{pmatrix}, \tag{5.1}$$

kde blok M_{00} je regulární matice o rozměru $k \times k$. Tuto submatici nazýváme vyváženou pokud platí $k = \lceil n/2 \rceil$. Můžeme ji rozložit pomocí vzorce

$$M = \begin{pmatrix} I & 0 \\ M_{10}M_{00}^{-1} & I \end{pmatrix} \begin{pmatrix} M_{00} & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I & M_{00}^{-1}M_{01} \\ 0 & I \end{pmatrix}, \tag{5.2}$$

$$M^{-1} = \begin{pmatrix} I & -M_{00}^{-1}M_{01} \\ 0 & I \end{pmatrix} \begin{pmatrix} M_{00}^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -M_{10}M_{00}^{-1} & I \end{pmatrix}, \tag{5.3}$$

kde je matice S definována jako Schurův doplněk (viz. strana 11)

$$S = M_{11} - M_{10}M_{00}^{-1}M_{01}. \tag{5.4}$$

Předchozí vztah lze použít pro výpočet inverzní matice v následujícím Algoritmu 4 a zároveň vztah (4.2) na stráně 12 pro výpočet determinantu inverzní matice.

Algoritmus 4 Inverze Toeplitzovy matice a její determinant

Vstup: Regulární Toeplitzova matice M.

Výstup: Inverzní matice M^{-1} a determinant inverzní matice $\det M^{-1}$.

1: **if** Rozměr $M = 1 \times 1$ **then**
2: $M^{-1} = \frac{1}{M(0,0)}$ a $\det M^{-1} = \frac{1}{M(0,0)}$
3: **else**
4: $M = \begin{pmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{pmatrix}$ Na submatici M_{00} použijeme algoritmus 4 pro výpočet M_{00}^{-1} a determinantu $\det M_{00}^{-1}$
5: $S = M_{11} - M_{10}M_{00}^{-1}M_{01}$
6: Pomocí algoritmu 4 spočteme inverzní matici S^{-1} a její determinant $\det S^{-1}$.
7: Spočteme $M^{-1} = \begin{pmatrix} I & -M_{00}^{-1}M_{01} \\ 0 & I \end{pmatrix} \begin{pmatrix} M_{00}^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -M_{10}M_{00}^{-1} & I \end{pmatrix}$ a determinant $\det M^{-1} = \det S^{-1} \cdot \det M_{00}^{-1}$.
8: **end if**

Kapitola 6

Řešení soustavy rovnic s Toeplitzovou maticí

V této kapitole nejprve popíšene rychlé algoritmy pro speciální případy soustav rovnic [Blahut, 2010, kap. 7.1] a [Blahut, 2010, kap. 7.1]. Poté následuje metoda pro soustavu s obecnou Toeplitzerovou maticí [Golub and Van Loan, 1996]. Na závěr se budu věnovat řešení založeném na Euklidově algoritmu [Blahut, 2010, kap. 7.1].

Jedná se důležitou a často řešenou úlohu, proto je navrhnuto mnoho různých algoritmů řešení. Řeší se rovnice $\mathbf{Ax} = \mathbf{g}$, kde $\mathbf{A}$ je čtvercová matice soustavy, $\mathbf{g}$ jsou pravé strany a $\mathbf{x}$ je vektor hledaných řešení. Možností je spočítat inverzní matici $\mathbf{A}^{-1}$ a vektor pravých řešení spočítat výpočtem $\mathbf{x}{=}\mathbf{A}^{-1}\mathbf{g}$, nebo použít Gaussovu eliminaci a další QR, SVD nebo Choleského faktorizace pro symetrické matice. Tyto řešení mají standardní výpočetní náročnost n^3. Pro některé aplikace, kde matice soustavy má na diagonále 100 a více prvků musíme nalézt efektivnější způsoby řešení.

6.1 Levinsonův algoritmus

Tento algoritmus je iterativní a pracuje se symetrickou Toeplitzovu maticí

$$\left(\begin{array}{ccccc|c} a_0 & a_1 & a_2 & \dots & a_{n-2} & a_{n-1} \\ a_1 & a_0 & a_1 & \dots & a_{n-3} & a_{n-2} \\ a_2 & a_1 & a_0 & \dots & a_{n-4} & a_{n-3} \\ \vdots & & & & \vdots & \vdots \\ a_{n-2} & a_{n-3} & a_{n-4} & \dots & a_0 & a_1 \\ \hline a_{n-1} & a_{n-1} & a_{n-3} & \dots & a_1 & a_0 \end{array}\right) \left(\begin{array}{c} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-2} \\ \hline x_{n-1} \end{array}\right) = \left(\begin{array}{c} g_0 \\ g_1 \\ g_2 \\ \vdots \\ g_{n-2} \\ \hline g_{n-1} \end{array}\right).$$

Použitím matice $\mathbf{J}$ rovnici můžeme upravit

$$\mathbf{Ax} = \mathbf{g},$$

$$\mathbf{JAJJx} = \mathbf{Jg}.$$

Protože je matice $\mathbf{A}$ symetrická platí

$$\mathbf{JAJ} = \mathbf{A},$$

$$\mathbf{AJx} = \mathbf{Jg}.$$

Díky tomu můžeme napsat

$$\begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_{n-1} \\ s_1 & a_0 & a_1 & \dots & a_{n-2} \\ a_2 & a_1 & a_0 & \dots & a_{n-3} \\ \vdots & & & & \vdots \\ a_{n-1} & a_{n-2} & & \dots & a_0 \end{pmatrix} \begin{pmatrix} x_{n-1} \\ x_{n-2} \\ \vdots \\ x_1 \\ x_0 \end{pmatrix} = \begin{pmatrix} g_{n-1} \\ g_{n-2} \\ \vdots \\ g_1 \\ g_0 \end{pmatrix}.$$

Algoritmus pracuje se submaticí matice $\mathbf{A}$, která je daná

$$\mathbf{A}^{(r)} = \begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_{r-1} \\ a_1 & a_0 & a_1 & \dots & a_{r-2} \\ a_2 & a_1 & a_0 & \dots & a_{r-3} \\ \vdots & & & & \vdots \\ a_{r-1} & & & \dots & a_0 \end{pmatrix}.$$

Umažeme tedy $n - r$ sloupců z prava a stejný počet řádků od spodu matice.

Levinsonův algoritmus je iterační algoritmus, postup iterace značí proměnná r. V každém kroku r vyřešíme r-tou rozšířenou soustavu

$$\begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_{r-1} \\ a_1 & a_0 & a_1 & \dots & a_{r-2} \\ a_2 & a_1 & a_0 & \dots & a_{r-3} \\ \vdots & & & & \vdots \\ a_{r-1} & & & \dots & a_0 \end{pmatrix} \begin{pmatrix} x_0^{(r)} \\ x_1^{(r)} \\ \vdots \\ x_{r-1}^{(r)} \end{pmatrix} = \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{r-1} \end{pmatrix},$$

kde index r ve vektoru $\mathbf{x}^{(r)}$ označuje řešení r-té rozšířené soustavy. Vektor $\mathbf{x}^{(r)}$ řeší rozšířenou soustavu, neřeší však původní soustavu a je rozdílný od vektoru $\mathbf{x}$, který řeší původní soustavu.

Tento algoritmus zavádí čtyři pracovní proměnné, tři jsou skaláry $\alpha_r, \beta_r, \gamma_r$ a vektor délky r pojmenovaný $\mathbf{t}^{(r)}$. Tyto proměnné, $\alpha_r, \beta_r, \gamma_r, \mathbf{t}^{(r)}$ splňují pro každé r rovnici

$$\begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_{r-1} \\ a_1 & a_0 & a_1 & \dots & a_{r-2} \\ a_2 & a_1 & a_0 & \dots & a_{r-3} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{r-1} & a_{r-2} & a_{r-3} & \dots & a_0 \end{pmatrix} \begin{pmatrix} x_0^{(r)} \\ x_1^{(r)} \\ \vdots \\ \\ x_{r-1}^{(r)} \end{pmatrix} = \begin{pmatrix} \alpha_r \\ 0 \\ \vdots \\ \\ 0 \end{pmatrix}. \tag{6.1}$$

Vektor na pravé straně je nulový, až na jeden prvek α_r. Další krok iterace rozšíří předchozí iteraci tak, aby platilo

$$\left(\begin{array}{ccccc|c} a_0 & a_1 & a_2 & \dots & a_{r-1} & a_r \\ a_1 & a_0 & a_1 & \dots & a_{n-2} & a_{r-1} \\ \vdots & \vdots & & & \vdots & \vdots \\ a_{r-1} & a_{r-2} & & \dots & a_0 & a_1 \\ \hline a_r & a_{r-1} & & \dots & a_1 & a_0 \end{array}\right) \begin{pmatrix} x_0^{(r)} \\ x_1^{(r)} \\ \vdots \\ x_{r-1}^{(r)} \\ 0 \end{pmatrix} = \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{r-1} \\ \gamma_r \end{pmatrix},$$

která definuje γ_r a

$$\left(\begin{array}{ccccc|c} a_0 & a_1 & a_2 & \dots & a_{r-1} & a_r \\ a_1 & a_0 & a_1 & \dots & a_{n-2} & a_{r-1} \\ \vdots & \vdots & & & \vdots & \vdots \\ a_{r-1} & a_{r-2} & & \dots & a_0 & a_1 \\ \hline a_r & a_{r-1} & & \dots & a_1 & a_0 \end{array}\right) \begin{pmatrix} t_0^{(r)} \\ t_1^{(r)} \\ \vdots \\ t_{r-1}^{(r)} \\ 0 \end{pmatrix} = \begin{pmatrix} \alpha_r \\ 0 \\ \vdots \\ 0 \\ \beta_r \end{pmatrix},$$

která zavádí β_r. Když platí $\gamma_r = g_r$ a $\beta_r = 0$ tak $\mathbf{x}^{(r+1)}$ a $\mathbf{t}^{r+1}$ jsou stejné jako $\mathbf{x}^{(r)}$ a $\mathbf{t}^{(r)}$. Pokud toto neplatí musí být $\mathbf{x}^{(r)}$ nebo $\mathbf{t}^{(r)}$ do tvaru $\mathbf{x}^{(r+1)}$ a $\mathbf{t}^{r+1}$.

Nejprve inicializujeme proměnné pro $r = 1$, poté předpokládáme splnění pro $r - 1$ a musíme ukázat způsob, jak rovnice splnit pro r. Jedná se tedy o prostou matematickou indukci.

Pro další výpočty zavedeme tyto rovnice

$$\begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_{r-1} & a_r \\ a_1 & a_0 & a_1 & \dots & a_{r-2} & a_{r-1} \\ \vdots & \vdots & & & \vdots & \vdots \\ a_{r-1} & a_{r-1} & & \dots & a_0 & a_1 \\ a_r & a_{r-1} & & \dots & a_1 & a_0 \end{pmatrix} \begin{pmatrix} 0 \\ x_{r-1}^{(r)} \\ \vdots \\ x_1^{(r)} \\ x_0^{(r)} \end{pmatrix} = \begin{pmatrix} \gamma_r \\ g_{r-1} \\ \vdots \\ g_1 \\ g_0 \end{pmatrix},$$

$$\begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_{r-1} & a_r \\ a_1 & a_0 & a_1 & \dots & a_{r-1} & a_{r-1} \\ \vdots & \vdots & & & \vdots & \vdots \\ a_{r-1} & a_{r-2} & & \dots & a_0 & a_1 \\ a_r & a_{r-1} & & \dots & a_1 & a_0 \end{pmatrix} \begin{pmatrix} 0 \\ t_{r-1}^{(r)} \\ \vdots \\ t_1^{(r)} \\ t_0^{(r)} \end{pmatrix} = \begin{pmatrix} \beta_r \\ 0 \\ \vdots \\ 0 \\ \alpha_r \end{pmatrix}.$$

Z těcho rovnicí určíme další iteraci. Nechť

$$\begin{pmatrix} t_0^{(r+1)} \\ t_1^{(r+1)} \\ \vdots \\ t_{r-1}^{(r+1)} \\ t_r^{(r+1)} \end{pmatrix} = k_1 \begin{pmatrix} t_0^{(r)} \\ t_1^{(r)} \\ \vdots \\ t_{r-1}^{(r)} \\ 0 \end{pmatrix} + k_2 \begin{pmatrix} 0 \\ t_{r-1}^{(r)} \\ \vdots \\ t_1^{(r)} \\ t_0^{(r)} \end{pmatrix},$$

pro nějaké konstanty k_1 a k_2. Tím, že doplníme nulu nejprve na konec vektoru a poté na začáte prodloužíme výsledný vektor na požadovanou délku. Pro tento rožšířený vektor musíme splnit rovnici (6.1), která rozepsaná vypadá

$$\begin{pmatrix} a_0 & a_1 & \dots & a_r \\ a_1 & a_0 & \dots & a_{r-1} \\ \vdots & \vdots & & \vdots \\ a_{r-1} & a_{r-2} & \dots & a_1 \\ a_r & a_{r-1} & \dots & a_0 \end{pmatrix} \begin{pmatrix} t_0^{(r+1)} \\ t_1^{(r+1)} \\ \vdots \\ t_{r-1}^{(r+1)} \\ t_r^{(r+1)} \end{pmatrix} = k_1 \begin{pmatrix} \alpha_r \\ 0 \\ \vdots \\ 0 \\ \beta_r \end{pmatrix} + k_2 \begin{pmatrix} \beta_r \\ 0 \\ \vdots \\ 0 \\ \alpha_r \end{pmatrix} = \begin{pmatrix} \alpha_{r+1} \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix},$$

kde musíme zvolit proměnné k_1 a k_2 tak, aby to splnilo podmínku z rovnice (6.1), která zní

$$0 = k_1\beta_r + k_2\alpha_r.$$

Zvolíme $k_1 = \alpha_r$ a $k_2 = -\beta_r$. Potom $a_{r+1} = k_1\alpha_r + k_2\beta_r = \alpha_r^2 - \beta_r^2$. Tím máme dokončen jeden krok iterace.

Avšak můžeme vybrat rozdílné k_1, které nám dá rozdílnou numerickou přesnost. Konečně mějme

$$\begin{pmatrix} x_0^{(r+1} \\ x_1^{(r+1} \\ \vdots \\ x_{r-1}^{(r+1} \\ x_r^{(r+1} \end{pmatrix} = \begin{pmatrix} x_0^{(r)} \\ x_1^{(r)} \\ \vdots \\ x_{r-1}^{(r)} \\ 0 \end{pmatrix} + k_3 \begin{pmatrix} t_r^{(r+1)} \\ t_{r-1}^{(r+1)} \\ \vdots \\ t_1^{(r+1)} \\ t_0^{(r+1)} \end{pmatrix},$$

kde konstanta k_3 není ještě určená, poté

$$\begin{pmatrix} a_0 & a_1 & \dots & a_r \\ a_1 & a_0 & \dots & a_{r-1} \\ \vdots & & & \vdots \\ a_{r-1} & & \dots & a_1 \\ a_r & & \dots & a_0 \end{pmatrix} \begin{pmatrix} x_0^{(r+1)} \\ x_1^{(r+1)} \\ \vdots \\ x_{r-1}^{(r+1)} \\ x_r^{(r+1)} \end{pmatrix} = \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{r-1} \\ \gamma_r \end{pmatrix} + k_3 \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \alpha_{r+1} \end{pmatrix} = \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{r-1} \\ g_r \end{pmatrix},$$

kde pro získání rovnosti na pravo, zvolíme k_3 tak aby

$$\gamma_r + k_3 \alpha_{r+1} = g_r.$$

To dokončí iteraci.

V samotném výpočtu algoritmu nemusíme vytvářet všechny submatice. Algoritmus je přehledně zapsán v Algoritmu 5. Funkce rev otáčí pořadí prvků ve vektoru.

Algoritmus 5 Levinsonův algoritmus

Vstup: Vektor popisující Toeplitzovu matici $(a_0, \dots, a_{n-1})$ a vektor pravé strany $(g_0, \dots, g_{n-1})$.

Výstup: Vektor řešení $(x_0, \dots, x_{n-1})$.

1: Inicializace $r = 1, x_0^{(1)} = g_0/a_0, t_0^{(1)} = 1, \alpha_1 = a_0$.
2: **for** $r = n - 1$ **do**
3: Spočteme pracovní konstanty $\beta_r = \sum_{i=0}^{r} a_i t_{r-i}, \gamma_r = \sum_{i=0}^{r} a_i x_{r-i}$.
4: Rozšíříme vektory t a x

$$t^{(r+1)} = \big(\alpha_r t^{(r)}, \quad 0\big) - \big(\beta_r \mathrm{rev}(t^{(r)}), \quad 0\big)$$

$$x^{(r+1)} = \big(x^{(r)}, \quad 0\big) + \frac{g_r - \gamma_r}{\alpha_r^2 - \beta_r^2} \mathrm{rev}(t^{(r+1)}).$$

5: Vypočteme následující $\alpha_{r+1} = \alpha_r^2 - \beta_r^2$.
6: $r = r + 1$
7: **end for**
8: Vrátíme vektor x.

Náročnost toho algoritmu je n^2, protože průchod každou smyčkou je úměrný velikosti r a má n průchodu smyčkou. Nesmíme během výpočtu dělit nulou, to vede k chybě algoritmu. Tato chyba nastane však pouze pokud je nějaká submatice singulární.

V komplexním oboru není symetrycká matice příliš častá. Častější je hermitovsky sdružená matice.

6.2 Durbinův algoritmus

Pokud pravá strana rovnice má speciální tvar $b = (a_1, a_2, \ldots, a_n)^T$, můžeme použít lepší algoritmus, známý jako Durbinův algoritmus. Pro použití toho algoritmu předpokládejme symetrickou matici a vektor pravých stran složenou z částí Toeplitzovy matice v následujícím tvaru

$$\left(\begin{array}{ccccc|c} a_0 & a_1 & a_2 & \ldots & a_{n-2} & a_{n-1} \\ a_1 & a_0 & a_1 & \ldots & a_{n-3} & a_{n-2} \\ a_2 & a_1 & a_0 & \ldots & a_{n-4} & a_{n-3} \\ \vdots & & & & \vdots & \vdots \\ a_{n-2} & a_{n-3} & a_{n-4} & \ldots & a_0 & a_1 \\ \hline a_{n-1} & a_{n-2} & a_{n-3} & \ldots & a_1 & a_0 \end{array}\right) \left(\begin{array}{c} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-2} \\ \hline x_{n-1} \end{array}\right) = - \left(\begin{array}{c} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \\ \hline a_n \end{array}\right).$$

Bloky matice ukazují, postup iterace, na kterém je algoritmus založen. Vektor pravých stan je složen z prvků Toeplitzovy matice. Algoritmus umožňuje získat výsledek za polovinu času, protože se polynom $t(x)$ nemusí počítat. Pravá strana této rovnice by mohla být omezující, ale tato rovnice se používá při spektrální analýze.

V kroku r řešíme zkrácený problém

$$\begin{pmatrix} a_0 & a_1 & a_2 & \ldots & a_{r-1} \\ a_1 & a_0 & a_1 & \ldots & a_{r-2} \\ \vdots & \vdots & & & \vdots \\ a_{r-1} & a_{r-2} & & \ldots & a_0 \end{pmatrix} \begin{pmatrix} x_0^{(r)} \\ x_1^{(r)} \\ \vdots \\ x_{r-1}^{(r)} \end{pmatrix} = - \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_r \end{pmatrix}.$$

Další krok iterace začneme rovnící

$$\left(\begin{array}{ccccc|c} a_0 & a_1 & a_2 & \ldots & a_{r-1} & a_r \\ a_1 & a_0 & a_1 & \ldots & a_{r-2} & a_{r-1} \\ \vdots & \vdots & & & \vdots & \vdots \\ a_{r-1} & a_{r-2} & & \ldots & a_0 & a_1 \\ \hline a_r & a_{r-1} & & \ldots & a_1 & a_0 \end{array}\right) \begin{pmatrix} x_0^{(r)} \\ x_1^{(r)} \\ \vdots \\ x_{r-1}^{(r)} \\ 0 \end{pmatrix} = - \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_r \\ \gamma_r \end{pmatrix},$$

která definuje γ_r. Iterace musí vypočíst $\mathbf{x}^{(r)}$ tak aby γ_r byla rovna a_{r+1}.

Nechť je $\mathbf{x}^{(r+1)}$ dán jako

$$\begin{pmatrix} x_0^{(r+1)} \\ x_1^{(r+1)} \\ \vdots \\ x_{r-1}^{(r+1)} \\ x_r^{(r+1)} \end{pmatrix} = \begin{pmatrix} x_0^{(r)} \\ x_1^{(r)} \\ \vdots \\ x_{r-1}^{(r)} \\ 0 \end{pmatrix} + k_r \begin{pmatrix} x_{r-1}^{(r)} \\ x_{r-2}^{(r)} \\ \vdots \\ x_0^{(r)} \\ 0 \end{pmatrix} + \beta_r \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

Když můžeme vybrat k_r a β_r, tak abychom dostali požadovanou formu, máme dobrý algoritmus. Zvolme

$$\gamma_r = -\sum_{i=1}^{r} x_{r-i}^{(r)} a_i,$$

$$\gamma_r' = -\sum_{i=1}^{r} x_i^{(r)} a_i,$$

potom

$$\left(\begin{array}{ccc|c} a_0 & \dots & a_{r-1} & a_r \\ a_1 & \dots & a_{r-3} & a_r \\ \vdots & & \vdots & \vdots \\ a_{r-1} & \dots & a_0 & a_1 \\ \hline a_r & & a_1 & a_0 \end{array}\right) \begin{pmatrix} x_0^{(r+1)} \\ x_1^{(r+1)} \\ \vdots \\ x_{r-1}^{(r+1)} \\ x_r^{(r+1)} \end{pmatrix} =$$

$$= -\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_r \\ \gamma_r \end{pmatrix} - k_r \begin{pmatrix} a_r \\ a_{r-1} \\ \vdots \\ a_1 \\ \gamma_r' \end{pmatrix} + \beta_r \begin{pmatrix} a_r \\ a_{r-1} \\ \vdots \\ a_1 \\ a_0 \end{pmatrix} = -\begin{pmatrix} a_1 \\ \vdots \\ a_r \\ a_{r+1} \end{pmatrix}.$$

K získání řešení musíme zvolit k_r a β_r tak, aby platilo

$$k_r - \beta_r = 0$$

a

$$-\gamma_r - k_r \gamma_r' + \beta_r a_0 = -a_{r+1}.$$

Tudíž $k_r = \beta_r$ a

$$\beta_r = -\frac{(a_{r+1} - \gamma_r)}{(a_0 - \gamma_r')}.$$

S těmito rovnicemi se řešení r-té iterace změní na řešení $(r+1)$-té rovnice. Proto může být první řešení rekuzivně dopočítáno do n-tého řešení. To je celý postup Durbinova algoritmu, viz. zápis Algoritmus 6.

Algoritmus 6 Durbinův algoritmus

Vstup: Vektor popisující Toeplitzovu matici $(a_0, \ldots, a_{n-1})$ a pravou stranu $(a_1, \ldots, a_n)$.

Výstup: Vektor řešení $(x_1, \ldots, x_n)$.

1: Inicilializujeme $r = 1$ a $x_0^{(1)} = -a_1/a_0$.
2: **for** $r < n$ **do**
3: Vypočítáme pomocné proměnné

$$\gamma_r = -\sum_{i=1}^{r} x_{r-i}^{(r)} a_i$$

$$\gamma_r' = -\sum_{i=1}^{r} x_i^{(r)} a_i$$

4: $\beta_r = -\frac{a_{r-1}-\gamma_r}{a_0-\gamma_r'}$
5: $x^{(r+1)} = \left(x^{(r)} + \beta_r \mathrm{rev}\left(x^{(r)}\right), \quad \beta_r\right)$
6: $r = r + 1$
7: **end for**
8: Vrátíme vektor x.

6.3 Levinson-Durbinův algoritmus

Tyto algoritmy můžeme rozšířit na obecné Toeplitzerovy matice a libovolnou pravou stranu

$$\left(\begin{array}{ccccc|c} a_0 & a_1 & a_2 & \ldots & a_{n-2} & a_{n-1} \\ a_{-1} & a_0 & a_1 & \ldots & a_{n-3} & a_{n-2} \\ a_{-2} & a_{-1} & a_0 & \ldots & a_{n-4} & a_{n-3} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{2-n} & a_{3-n} & a_{4-n} & \ldots & a_0 & a_1 \\ \hline a_{1-n} & a_{2-n} & a_{3-n} & \ldots & a_{-1} & a_0 \end{array}\right) \left(\begin{array}{c} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-2} \\ \hline x_{n-1} \end{array}\right) = \left(\begin{array}{c} g_0 \\ g_1 \\ g_2 \\ \vdots \\ g_{n-2} \\ \hline g_{n-1} \end{array}\right), \tag{6.2}$$

avšak za cenu vyšší výpočetní náročnosti.

6.3.1 Výpočet "forward" a "backward" vektorů

Pro tento algoritmus potřebujeme vypočítat vektory s jejich pomocí spočteme další krok iterace. Vektory f^k ("forward") a b^k ("backward") jsou defi-

novány jako

$$T^{(r)}f^{(r)} = e_1^{(r)} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, T^{(r)}b^{(r)} = e_r^{(r)} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}, \tag{6.3}$$

kde $T^{(r)}$ je submatice Toeplitzovy matice o rozměru $r \times r$

$$f^k = \begin{pmatrix} f_1^{(r)} \\ \vdots \\ f_k^{(r)} \end{pmatrix}, b^k = \begin{pmatrix} b_1^{(r)} \\ \vdots \\ b_k^{(r)} \end{pmatrix}.$$

Mějme "forward" vektor $f^{(r-1)}$, který splňuje rovnici

$$T^{(r-1)}f^{(r-1)} = e_1^{(r-1)}.$$

Nyní tento vektor $f^{(r-1)}$ rozšiřme, tak aby splňoval rovnici

$$T^{(r)}f^{(r)} = e_1^r.$$

Toto rozšíření provedeme rovnicí

$$\begin{aligned} T^{(r)}\begin{pmatrix} f^{(r-1)} \\ 0 \end{pmatrix} &= \begin{pmatrix} & & & a_{r-1} \\ & T^{(r-1)} & & \vdots \\ & & & a_1 \\ a_{1-r} & \dots & a_{-1} & a_0 \end{pmatrix}\begin{pmatrix} f^{(r-1)} \\ 0 \end{pmatrix} = \\ &= \begin{pmatrix} T^{(r-1)}f^{(r-1)} \\ \sum_{i=1}^{r-1} a_{i-1}f_i^{(r-1)} \end{pmatrix} = \begin{pmatrix} e_1^{(r-1)} \\ \epsilon_r^{(r)} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ \epsilon_f^{(r)} \end{pmatrix}. \end{aligned} \tag{6.4}$$

Pro "backward" vektor $b^{(r-1)}$ chceme také splnit podmínku $T^{(r)}b^{(r)} = e_r^{(r)}$

$$T^{(r)}\begin{pmatrix}0\\ b^{(r-1)}\end{pmatrix} = \begin{pmatrix} a_0 & a_1 & \dots & a_{r-1} \\ a_{-1} & & & \\ \vdots & & T^{(r-1)} & \\ a_{1-r} & & & \end{pmatrix}\begin{pmatrix}0\\ b^{(r-1)}\end{pmatrix} =$$

$$= \begin{pmatrix}\sum_{i=1}^{r-1} a_{i-r+1}b_i^{(r-1)} \\ T^{(r-1)}b^{(r-1)}\end{pmatrix} = \begin{pmatrix}\epsilon_b^{(r-1)} \\ e_{r-1}^{(r-1)}\end{pmatrix} = \begin{pmatrix}\epsilon_b^{(r)} \\ 0 \\ \vdots \\ 0 \\ 1\end{pmatrix}. \tag{6.5}$$

Na $\epsilon_f^{(r)}$ a $\epsilon_b^{(r)}$ se lze dívat jako na chybu, kterou musíme odstranit, aby platila rovnost (6.3). Jestli budou obě hodnoty rovny 0, tak jsme našli $f^{(r)}$ a $b^{(r)}$. Budeme je hledat ve tvaru

$$f^{(r)} = \alpha_f\begin{pmatrix}f^{(r-1)}\\ 0\end{pmatrix} + \beta_f\begin{pmatrix}0\\ b^{(r-1)}\end{pmatrix},\ b^{(r)} = \alpha_b\begin{pmatrix}f^{(k-1)}\\ 0\end{pmatrix} + \beta_b\begin{pmatrix}0\\ b^{(r-1)}\end{pmatrix}, \tag{6.6}$$

kde α_f, β_f a α_b, β_b jsou neznámé parametry. Z definice (6.3) plyne rovnost

$$T^{(r)}f^{(r)} = T^{(r)}\left(\alpha_f\begin{pmatrix}f^{(r-1)}\\ 0\end{pmatrix} + \beta_f\begin{pmatrix}0\\ b^{(r-1)}\end{pmatrix}\right) =$$

$$= \alpha_f T^{(r)}\begin{pmatrix}f^{(r-1)}\\ 0\end{pmatrix} + \beta_f\begin{pmatrix}0\\ b^{(r-1)}\end{pmatrix} = e_1^{(r)}, \tag{6.7}$$

$$T^{(r)}b^{(r)} = T^{(r)}\left(\alpha_b\begin{pmatrix}f^{(r-1)}\\ 0\end{pmatrix} + \beta_b\begin{pmatrix}0\\ b^{(r-1)}\end{pmatrix}\right) =$$

$$= \alpha_b T^{(r)}\begin{pmatrix}f^{(r-1)}\\ 0\end{pmatrix} + \beta_b\begin{pmatrix}0\\ b^{(r-1)}\end{pmatrix} = e_r^{(r)}. \tag{6.8}$$

Podle rovnic (6.4), (6.5) může rovnice (6.7), (6.8) dále upravovat

$$T^{(r)}f^{(r)} = \alpha_f\begin{pmatrix}e_1^{(r-1)}\\ \epsilon_f^{(r)}\end{pmatrix} + \beta_f\begin{pmatrix}\epsilon_b^{(r-1)}\\ e_{r-1}^{(r-1)}\end{pmatrix} = e_1^{(r)},$$

$$T^{(r)}b^{(r)} = \alpha_b\begin{pmatrix}e_1^{(r-1)}\\ \epsilon_x^{(r)}\end{pmatrix} + \beta_b\begin{pmatrix}\epsilon_b^{(r-1)}\\ e_{r-1}^{(r-1)}\end{pmatrix} = e_r^{(r)}.$$

Tyto rovnice můžeme maticově zapsat jako

$$\begin{pmatrix} e_1^{(r-1)} & \epsilon_b^{(r)} \\ \epsilon_f^{(r)} & e_{r-1}^{(r-1)} \end{pmatrix} \begin{pmatrix} \alpha_f & \alpha_b \\ \beta_f & \alpha_b \end{pmatrix} = \begin{pmatrix} e_1^{(r)} & e_r^{(r)} \end{pmatrix}$$

a pokud tento výraz rozepíšeme, dostaneme

$$\begin{pmatrix} 1 & \epsilon_b^{(r)} \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ \epsilon_f^{(r)} & 1 \end{pmatrix} \begin{pmatrix} \alpha_f & \alpha_b \\ \beta_f & \beta_b \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \Leftrightarrow \begin{pmatrix} 1 & \epsilon_b^{(r)} \\ \epsilon_x^{(r)} & 1 \end{pmatrix} \begin{pmatrix} \alpha_f & \alpha_b \\ \beta_f & \beta_b \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Tato rovnice se dá vyřešit

$$\begin{pmatrix} \alpha_f & \alpha_b \\ \beta_f & \beta_b \end{pmatrix} = \begin{pmatrix} 1 & \epsilon_b^{(r)} \\ \epsilon_f^{(r)} & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \frac{1}{1 - \epsilon_x^{(r)} \epsilon_b^{(r)}}.$$

Nyní můžeme spočítat $f^{(r)}$ a $b^{(r)}$, tak aby platilo 6.3. Pro $r = 1$ platí

$$T^(1) f^{(1)} = 1 \rightarrow a_0 x^{(1)} = 1 \rightarrow x^{(1)} = \frac{1}{a_0},$$

$$T^(1) b^{(1)} = 1 \rightarrow a_0 b^{(1)} = 1 \rightarrow x^{(1)} = \frac{1}{a_0}.$$

6.3.2 Řešení lineárních rovnic

Řešíme soustavu rovnic definovanou v (6.2). Jedná se opět o iterativní algoritmus. Začneme výpočtem prvního prvku pro $r = 1$, dostaneme

$$T^{(1)} x^{(1)} = y^{(1)} \rightarrow t_0 x_1^{(1)} = y_1 \rightarrow x^{(1)} = \frac{y_1}{t_0}.$$

Tím máme zajištěn počátek, nyní musíme najít způsob, jak řešení získat z kroku $r - 1$ pro krok r. Dostaneme rovnici

$$T^{(r)} \begin{pmatrix} x^{(r-1)} \\ 0 \end{pmatrix} = \begin{pmatrix} & T^{(r-1)} & & a_{r-1} \\ & & & a_{r-2} \\ & & & \vdots \\ a_{1-r} & a_{2-r} & \dots & a_0 \end{pmatrix} \begin{pmatrix} x^{(r-1)} \\ 0 \end{pmatrix} =$$

$$= \begin{pmatrix} T^{(r-1)} x^{(r-1)} \\ \begin{pmatrix} a_{1-r} & \dots & a_{-1} \end{pmatrix} x^{(r-1)} \end{pmatrix} = \begin{pmatrix} y^{(r-1)} \\ \epsilon_x^{(r)} \end{pmatrix}, \quad (6.9)$$

kde $\epsilon_x^{(r)} = \begin{pmatrix} a_{1-r} & \dots & a_{-1} \end{pmatrix} x^{(r-1)} = \sum_{i=1}^{r} a_{r-i} x_i^{(r-1)}$ je nějaká chyba, kterou musíme odstranit. Použijeme k tomu "forward" a "backward" vektory, které jsme spočítali v části (6.3.1). Z definice (6.3) pro vektor $b^{(r)}$ platí

$$T^{(r)} b^{(r)} = e_r^{(r)} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

Když tuto rovnici přenásobíme číslem $(y_r - \epsilon_x^{(r)})$ a sečteme s rovnicí (6.9), dostaneme

$$T^{(r)} \begin{pmatrix} x^{(r-1)} \\ 0 \end{pmatrix} + (y_r + \epsilon_x^{(r)}) T^{(r)} b^{(r)} = \begin{pmatrix} y^{(r-1)} \\ \epsilon_x^{(r)} \end{pmatrix} + (y_r + \epsilon_x^{(r)}) T^{(r)} b^{(r)} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

Sečteme a upravíme obě strany a výsledek je

$$T^{(r)} \left(\begin{pmatrix} x^{(r-1)} \\ 0 \end{pmatrix} + (y_r + \epsilon_x^{(r)}) b^{(r)} \right) = \begin{pmatrix} y^{(r-1)} \\ y_r \end{pmatrix}. \tag{6.10}$$

Závorka na levé straně odpovídá řešení $x^{(r)}$, předpis řešení tedy je

$$x^{(r)} = \begin{pmatrix} x^{(r-1)} \\ 0 \end{pmatrix} + (y_r + \epsilon_x^{(r)}) b^{(r)}.$$

Celý Levinson-Durbinův algoritmus je shrnut v Algoritmu 7.

Poznamenejme, jestliže je matice symetrická, tak "forward" i "backward" vektor jsou identické.

Algoritmus 7 Výpočet řešení x soustavy $Tx = y$

Vstup: Toeplitzova matice T^n a vektor pravých stran y.

Výstup: Vektor řešení x.

1: $f^{(1)} = \frac{1}{a_0}$

2: $b^{(1)} = \frac{1}{a_0}$

3: $x^{(1)} = \frac{y_1}{a_0}$

4: **for** $r = 2$ do n **do**

5: $\epsilon_x^{(r)} = \sum_{i=1}^{r-1} a_{i-1} f_i^{(r-1)} = \begin{pmatrix} a_{r-1} & a_{r-2} & \vdots & a_1 \end{pmatrix} f^{(r-1)}$

6: $\epsilon_x^{(r)} = \sum_{i=1}^{r-1} a_{i-r+1} b_i^{(r-1)} = \begin{pmatrix} a_{-1} & a_{-2} & \vdots & a_{-r+1} \end{pmatrix} b^{(r-1)}$

7: $\epsilon_x^{(r)} = \sum_{i=1}^{r-1} a_{i-1} x_i^{(r-1)} = \begin{pmatrix} a_{r-1} & a_{r-2} & \vdots & a_1 \end{pmatrix} x^{(r-1)}$

8: $f^{(r)} = \frac{1}{1-\epsilon_f^{(r)}\epsilon_b^{(r)}} \begin{pmatrix} f^{(r-1)} \\ 0 \end{pmatrix} + \frac{\epsilon_f^{(r)}}{1-\epsilon_f^{(r)}\epsilon_b^{(r)}} \begin{pmatrix} 0 \\ b^{(r-1)} \end{pmatrix}$

9: $b^{(r)} = \frac{-\epsilon_f^{(r)}}{1-\epsilon_f^{(r)}\epsilon_b^{(r)}} \begin{pmatrix} f^{(r-1)} \\ 0 \end{pmatrix} + \frac{1}{1-\epsilon_f^{(r)}\epsilon_b^{(r)}} \begin{pmatrix} 0 \\ b^{(r-1)} \end{pmatrix}$

10: $f^{(r)} = \begin{pmatrix} f^{(r-1)} \\ 0 \end{pmatrix} + (y_r - \epsilon_f^{(r)} b^{(r)}$

11: **end for**

12: Vrátíme vektor x.

6.4 Algoritmy založené na Euklidovu algoritmu

Na základě Euklidova algoritmu můžeme řešit soustavu ve speciálním tvaru (stejně jako v případě Durbinova algoritmu, viz (6.2))

$$\begin{pmatrix} a_{n-1} & a_{n-2} & a_{n-3} & \dots & a_0 \\ a_n & a_{n-1} & a_{n-2} & \dots & a_1 \\ a_{n+1} & a_n & a_{n-1} & \dots & a_2 \\ \vdots & \vdots & \vdots & & \vdots \\ a_{2n-2} & a_{2n-3} & a_{2n-4} & \dots & a_{n-1} \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{pmatrix} = \begin{pmatrix} -a_n \\ -a_{n-1} \\ -a_{n-2} \\ \vdots \\ -a_{2n-1} \end{pmatrix}.$$

Mějme dva polynomy

$$a(x) = \sum_{i=0}^{2n-1} a_i x^i,$$

$$f(x) = 1 + \sum_{i=1}^{n} f_i x^i.$$

Spočtěme jejich součin

$$g(x) = f(x)a(x).$$

V souladu s maticovým násobením vidíme

$$g_i = 0,\ i = n, \ldots, 2n-1.$$

Pro i větší $2n$, g_i může nabýt jakékoliv hodnoty. Chceme přeformulovat hledání inverzní matice na problém nalezení polynomů $f(x)$ a $g(x)$ takových, aby splňovali podmínku $\deg f(x) \leq n$, $\deg g(x) \leq n-1$ a

$$g(x) = f(x)a(x) \pmod{x^{2n}}.$$

Tuto rovnici můžu vyřešit pomocí Euklidova algoritmu.

Nyní uvedu důkaz jak Euklidův algoritmus může vypočítat polynomy $f(x)$ a $g(x)$ tak aby platili podmínky uvedené výše. Euklidův algoritmus vypočítá rovnice

$$\begin{pmatrix} s^{(r)}(x) \\ t^{(r)}(x) \end{pmatrix} = \begin{pmatrix} A_{11}^{(r)}(x) & A_{12}^{(r)}(x) \\ A_{21}^{(r)}(x) & A_{22}^{(r)}(x) \end{pmatrix} \begin{pmatrix} s(x) \\ t(x) \end{pmatrix},$$

tak aby platilo

$$t^{(r)}(x) = A_{22}^{(r)} t(x) \pmod{s(x)}.$$

Tyto rovnice budou vyřešeny, když dostaneme $t(x) = a(x)$ a $s(x) = x^{2n}$. To platí pro každou hodnotu r. Pokud můžeme nalézt r pro které platí $\deg A_{22}^{(r)}(x) \leq n$ a $\deg t^{(r)}(x) \leq n-1$, potom můžou být tyto polynomy řešením rovnice. Jestliže takové r existuje, potom polynomy $A_{22}^{(r)}(x)$ a $t^{(r)}$ musí být rovna požadovaným $f(x)$ a $g(x)$. Na závěr vybereme hodnotu r splňující

$$\deg t^{(r-1)}(x) \geq n,$$
$$\deg t^{(r)} \leq n-1.$$

To definuje unikátní hodnotu r, protože $\deg t^{(0)}(x) = 2n$ a stupeň $t^{(r)}(x)$ postupně klesá se zvyšujícím se r. Z definice r splníme první podmínku

$$\deg t^{(r)}(x) \leq n-1.$$

Jak roste r stejně roste stupeň $A_{22}^{(r)}(x)$. Nyní musíme jen ukázat, že platí

$$\deg A_{22}^{(r)}(x) \leq n.$$

To dokážeme pomocí inverzní matice $A^{(r)}(x)$. Ukažme

$$A^{(r)}(x) = \prod_{l=1}^{r} \begin{pmatrix} 0 & 1 \\ 1 & -Q^{(l)}(x) \end{pmatrix}. \tag{6.11}$$

Z této rovnice (6.11) je jasné, že platí $\deg A_{22}^{(r)} > \deg A_{12}^{(r)}(x)$. Dále připomeňme že $\deg s^{(r)}(x) > \deg t^{(r)}(x)$. Z těchto nerovností a maticové rovnice

$$\begin{pmatrix} s(x) \\ t(x) \end{pmatrix} = (-1)^r \begin{pmatrix} A_{22}^{(r)}(x) & -A_{12}^{(r)} \\ -A_{21}^{(r)}(x) & A_{11}^{(r)}(x) \end{pmatrix} \begin{pmatrix} s^{(r)}(x) \\ t^{(r)}(x) \end{pmatrix}$$

může být vyvozeno, že $\deg s(x) = \deg A_{22}^{(r)}(x) + \deg s^{(r)}(x)$ a protože $s^{(r)}(x) = t^{(r-1)}(x)$. Tím dostaneme

$$\deg A_{22}^{(r)}(x) = \deg s(x) - \deg t^{(r-1)}(x) \leq 2n - n = n,$$

kde nerovnost vychází z defenice r.

Nyní jsme ukázali většinu z následující věty.

Věta 6.1 *Je dáno* $s^{(0)}(x) = x^{2n}$ *a* $t^{(0)}(x) = a(x)$

$$A^{(0)}(x) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Řešme tyto rekuzivní rovnice dokud $\deg t^{(r)}(x) \leq n - 1$

$$Q^{(r)}(x) = \left\lfloor \frac{s^{(r-1)}(x)}{t^{(r-1)}(x)} \right\rfloor,$$

$$A^{(r)}(x) = \begin{pmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{pmatrix} A^{(r-1)}(x),$$

$$\begin{pmatrix} s^{(r)}(x) \\ t^{(r)}(x) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{pmatrix} \begin{pmatrix} s^{(r-1)}(x) \\ t^{(r-1)}(x) \end{pmatrix}$$

a nechť

$$g(x) = \Delta^{-1} t^{(r)}(x),$$
$$f(x) = \Delta^{-1} A_{22}^{(r)}(x),$$

kde $\Delta = A_{22}^{(r)}(0)$. *Za předpokladu, že* Δ *je nenulová, splňuje rovnici*

$$g(x) = f(x)a(x) \pmod{x^{2n}}$$

s $\deg f(x) \leq n$, $\deg g(x) \leq n - 1$, *and* $f_0 = 1$

Důkaz: Dělení Δ zaručuje $f_0 = 1$. Splnění ostatních podmínek vyplývá z předchozích rovnic. □

Pokud by Toeplitzova matice nebyla invertibilní, tak Euklidův algoritmus bude produkovat $A_{22}^{(r)}(x)$ menšího stupně než kterou pažaduje definice, ale Δ bude rovna nule a $f(x)$ bude nedefinována. Pokud bude Δ rovno nule, tak algoritmus místo řešení bude vracet rovnici

$$\begin{pmatrix} a_{n-1} & a_{n-2} & a_{n-3} & \dots & a_0 \\ a_n & a_{n-1} & a_{n-2} & \dots & a_1 \\ a_{n+1} & a_n & a_{n-2} & \dots & a_2 \\ \vdots & \vdots & \vdots & & \vdots \\ a_{2n-2} & a_{2n-3} & a_{2n-4} & \dots & a_{n-1} \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ \vdots \\ h_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

6.5 Berlekampův-Masseyův algoritmus

Tento algoritmus řeší Toeplitzův systém rovnic v následujícím tvaru

$$\begin{pmatrix} a_{n-1} & a_{n-2} & a_{n-3} & \dots & a_0 \\ a_n & a_{n-1} & a_{n-2} & \dots & a_1 \\ a_{n+1} & a_n & a_{n-1} & \dots & a_2 \\ \vdots & \vdots & \vdots & & \vdots \\ a_{2n-2} & a_{2n-3} & a_{2n-4} & \dots & a_{n-1} \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{pmatrix} = \begin{pmatrix} -a_n \\ -a_{n+1} \\ -a_{n+2} \\ \vdots \\ -a_{2n-1} \end{pmatrix}.$$

Toeplitzova matice může být v libovolném tvaru. Rovnice má však pevně danou pravou stranu, která je tvořena prvky Toeplitzovy matice.

Berlekampův-Masssayův algoritmus lze lehce vysvětlit pomocí autoregresivního filtru. Předpokládejme, že vektor f je znám. První řádek matice, tedy prvky $a_0, \dots, a_{n-1}$ definují prvek a_n. Druhý řádek definuje a_{n+1} pomocí prvků $a_1, \dots, a_n$. Celý tento proces se dá zapsat

$$a_j = -\sum_{i=1}^{n} f_i a_{j-i}, \ j = n, \dots, 2n-1.$$

Pro pevný vektor f tato rovnice popisuje autoregresivní filtr produkující sekvenci a_j s počátečnímy podmínkami pořadí.

V tomto případě je výpočet řešení soustavy rovnic shodný s nalezením odpovídajícího autoregresivního filtru, který bude produkovat danou sekvenci a_j. Pokud je matice regulární, existuje pouze jeden filtr, který má na výstupu odpovídající posloupnost a_j. Jestliže matice není invertabilní, může být více řešení, nebo také žádné. Tento algoritmus nalezne autoregresivní filtr nejnižšího řádu produkujícího danou posloupnost. Pokud bude mít systém rovnic více než jedno řešení, bude výsledek roven odpovídajícímu autoregresivnímu

filtru s nejnižším řádem. Pokud rovnice nemají žádné řešení získáme výsledek s nulovými váhami f_i pro i větší než n.

Postup pro nalezení autoregresivního filtru je také algoritmus pro vyřešení maticové rovnice pro vektor f. Nyní odvodíme postup pro nalezení autoregresivního filtru. Na řadu $a_0, \ldots, a_{2n-1}$ nemáme žádné specální požadavky.

Lineární zpětnovazební posuvný registr může být popsán délkou L posuvného registru a zpětnovazebního polynomu $f(x)$

$$f(x) = f_n x^n + f_{n-1} x^{n-1} + \cdots + f_1 x + 1.$$

Délka posuvného registru může být větší než stupeň $f(x)$, protože některé členy $f(x)$ mohou být nastaveny na nulu a tedy nemusí být použity.

Pro nalezení požadovaného posuvného registru musíme určit dvě věci, délku registru L a zpětnovazební polynom $f(x)$, kde $\deg f(x) \leq L$. Budeme tuto dvojici značit jako $(L, f(x))$. Musíme nalézt zpětnovazební posuvný registr který bude produkovat řadu $a_0, \ldots, a_{2n-1}$ pokud je správně inicializovaná a zároveň je nejkratší filtr s touto vlastností.

Algoritmus návrhu je rekurzivní. Pro každé r, které začíná $r = 1$, budeme navrhovat zpětnovazební posuvný registr produkující posloupnost $a_0, \ldots, a_r$. Dvojice $(L_r, f^{(r)}(x))$ označuje posuvný regist s minimální délkou produkující $a_0, \ldots, a_r$. Tento filtr nemusí být unikátní, může existovat více filtrů délky L_r splňující tyto podmínky. Použijeme však pouze jeden z možných filtrů. Pomocí iterace vytvoříme seznam posuvných registrů

$$\begin{gathered}(L_1, f^{(1)}(x)),\\ (L_2, f(2)(x)),\\ \vdots\\ (L_{r-1}, f^{(r-1)}(x)),\end{gathered}$$

kde je délka filtru neklesající. Berlekampův-Masseyův algoritmus vypočítá nový nejkratší posuvný registr $(L_r, f^{(r)}(x))$, který tvoří posloupnost

$$a_0, \ldots, a_{r-1}, a_r.$$

Při návrhu použijeme současný posuvný registr, pokud to bude možné, jinak změníme délku a váhové koeficienty, tak abychom získali požadovaný výsledek.

Během iterace r nejprve spočteme další výstupní člen $(r-1)$-tého výstupního registru

$$\hat{a_r} = -\sum_{j=1}^{L_{r-1}} f_j^{(r-1)} a_{r-j}.$$

Jelikož může být L_{r-1} větší než stupeň $f^{(r-1)}$, nekteré výrazy v sumě můžou být nulové. Nechť je Δ_r rozdíl mezi předpokládaným výsledkem a_r a aktuálním výsledkem $\hat{a_r}$ z posuvného registru

$$\Delta_r = a_r - \hat{a_r} = a_r + \sum_{j=1}^{L_{r-1}} f_j^{(r-1)} a_{r-j}.$$

Ve zkrácené verzi to může být zapsáno jako

$$\Delta_r = \sum_{j=0}^{L_{r-1}} f_j^{r-1} a_{r-j}.$$

Pokud je Δ_r nulová, pak platí $(L_r, f^{(r)}(x)) = (L_{r-1}, f^{(r-1)}(x))$ a r-tá iterace je hotova. Jinak upravíme váhy filtru upravíme polynom jako

$$f^{(r)}(x) = f^{(r-1)}(x) + Ax^{\ell} f^{m-1} a_{r-j-\ell},$$

kde A je pole prvků, ℓ je celé číslo a $f^{(m-1)}(x)$ je jeden z polynomů uvedených dříve v seznamu iterací. S novým polynomem dostaneme

$$\Delta'_r = \sum_{j=0}^{L_{r-1}} f_j^{(r)} a_{r-j} = \sum_{j=0}^{L_{r-1}} f_j^{(r-1)} a_{r-j} + A \sum_{j=0}^{L_{r-1}} f_j^{(m-1)} a_{r-j-\ell}.$$

Nyní vybereme m, ℓ a A tak aby Δ'_r byla rovna 0. Vybereme m menší než r pro který platí $\Delta_m \neq 0$, vyberme $\ell = r - m$ a vyberme $A = -\Delta_m^{-1}\Delta_r$. Potom

$$\Delta'_r = \Delta_r - \frac{\Delta_r}{\Delta_m}\Delta_m = 0.$$

Což znamená, že upravený posuvný registr bude mít na výstupu posloupnost $a_1, \ldots, a_{r-1}, a_r$. Nyní se musíme ujistit, že se jedná o filtr s nejmenším posuvným registrem. Zatím jsme neurčili jaké m z možných m pro které platí $\Delta_m \neq 0$. Pokud vybereme m jako poslední iteraci pro kterou platí $L_m > L_{m-1}$, dostaneme nejkratší posuvný registr v každé iteraci. Poslední tvrzení však musíme dokázat.

Věta 6.2 *Předpokládejme, že $(L_{r-1}, f^{r-1}(x))$ je zpětnovazebný posuvný registr nejkratší délky produkující posloupnost $a_1, \ldots, a_{r-1}$ a $(L_r, f^{(r)}(x))$ je nejkratší lineární zpětnovazebný posuvný registr, který má na výstupu posloupnost $a_1, \ldots a_{r-1}, a_r$. Když $f^{(r)}(x) \neq f^{(r-1)}(x)$, potom $L_r \geq \max[L_{r-1}, r - L_{r-1}]$.*

Důkaz: Nerovnost pro důkaz se skládá ze dvou nerovností

$$L_r \leq L_{r-1}$$

a

$$L_r \leq r - L_{r-1}.$$

První nerovnost je zřejmá, pokud filtr generuje posloupnost, generuje i jakoukoli podposloupnost dané posloupnosti. Druhá nerovnost je zřejmá pokud $L_{r-1} \leq r$. Nyní předpokládejme $L_{r-1} < r$. Předpokládejme, že druhá nerovnice není splněna a nalezněme spor. Potom $L_r \geq r - 1 - L_{r-1}$. Nechť $c(x) = f^{(r-1)}(x)$, $b(x) = f^{(r)}(x)$, $L = L_{r-1}$ a $L' = L_r$. Z předpokladu máme $r \leq L + L' + 1$ a $L < r$. Z věty dostaneme

$$a_r \neq -\sum_{i=1}^{L} c_i a_{r-i},$$

$$a_j = -\sum_{i=1}^{L} c_i a_{j-i} \; j = L+1, \ldots, r-1$$

a

$$a_j = -\sum_{k=1}^{L'} b_k a_{r-k} \; j = L'+1, \ldots, r.$$

Nyní vytvořme spor

$$a_r = -\sum_{k=1}^{L'} b_k a_{r-k} = \sum_{k=1}^{L'} b_k \sum_{i=1}^{L} c_i a_{r-k-i},$$

kde lze expandovat a_{r-k} do další sumy díky tomu, že $r-k$ jde od $r-1$ do $r-L'$, která je v rozsahu $L+1, \ldots, r-1$ díky předpokladu $r \geq L + L' + 1$. Zadruhé

$$a_r \neq -\sum_{i=1}^{L} c_i a_{r-i} = \sum_{i=1}^{L} c_i \sum_{k=1}^{L'} b_k a_{r-i-k},$$

kde rozšíření a_{r-i} do další sumy protože $r-i$ je od $r-1$ do $r-L$, což je znovu v rozsahu $L'+1, \ldots, r-1$. Sumace napravé strany by měla být zaměnitelná s pravou stranou předchozí rovnice. Díky tomu dostáváme spor $a_r \neq a_r$. Spor dokazuje pravdiovst tvrzení. □

Jestliže můžeme navrhnout posuvný regisr, který splňuje rovnici ve větě (6.2) tak, že se rovná, musí mít nejkratší délku. Důkaz věty (6.3) nám ukáže způsob vytvoření posuvného registru.

Věta 6.3 *Předpokládejme že* $(L_i, f^{(i)}(x))$, $i = 1, \ldots, r$ *je posloupnost lineárních zpětnovazebních posuvných registrů s minimální délkou, které produkují* $a_1, \ldots, a_i$. *Když* $f^{(r)}(x) \neq f^{(r-1)}(x)$ *potom*

$$L_r = \max[L_{r-1}, r - L_{r-1}]$$

a jakýkoli posuvný registr který produkuje $a_1, \ldots, a_r$ *a má délku rovnou pravé straně, tak se jedná o posuvný registr s minimální délkou.*

Důkaz: Z věty (6.2) nemůže být L_r menší než pravá strana. Nyní vytvoříme posuvný registr, který tvoří požadovanou sekvenci a jehož délka je rovna pravé staně a tudíž se musí jednat o minimální registr. Důkaz je pomocí matematické indukce. Předpokládejme, že máme takové registry pro všechny $k \leq r - 1$. Pro $k = 1, \ldots, r - 1$ nechť $(L_k, f^{(k)}(x))$ je minimální posuvný registr, který generuje $a_1, \ldots, a_k$. Pro indukční důkaz předpokládejme

$$L_k = \max[L_{k-1}, k - L_{k-1}], k = 1, \ldots, n - 1,$$

kdykoli $f^{(r)}(x) \neq f^{(k-1)}(x)$. To je jasné pro $k = 0$, když $L_0 = 0$ a $L_1 = 1$. Více obecně, když a_i je první nenulový člen dané sekvence, pak $L_{i-1} = 0$ a $L_i = i$. Indukční krok začíná v $k = 1$.

Nechť m označuje největší hodnotu k pro kterou je nutno změnit délku posuvného registru. Potom m je takové celé číslo, pro které platí

$$L_{r-1} = L_m > L_{m-1}.$$

Nyní máme

$$a_j + \sum_{i=1}^{L_{r-1}} f_i^{(r-1)} a_{j-i} = \sum_{i=1}^{L_{r-1}} f_i^{(r-1)} a_{j-i} = \begin{cases} 0, & j = L_{r-1}, \ldots, r-1 \\ \Delta_r, & k = r \end{cases}.$$

Když $\Delta_r = 0$, potom posuvný registr $(L_{r-1}, f^{(r-1)}(x))$ také generuje r-tou posloupnost, poté $(L_r, f^{(r)}(x)) = (L_{r-1}, f^{(r.1)}(x))$. Když $\Delta_r \neq 0$, potom musí být vytvořen nový posuvný registr. Připomeňmě, že délku registru měníme při $k = m$. Tak

$$a_j + \sum_{i=1}^{L_{m-1}} f_i^{(m-1)} a_{j-i} = \begin{cases} 0, & j = L_{m-1}, \ldots, m-1 \\ \Delta_m \neq 0, & j = m \end{cases}$$

a z úvodní věty

$$L_{r-1} = L_m = \max[L_{m-1}, m - L_{m-1}] = m - L_{m-1},$$

protože $L_m > L_{m-1}$. Vyberem nový polynom

$$f^{(r)}(x) = f^{(r-1)}(x) - \Delta_r \Delta_m^{-1} x^{r-m} f^{(r-1)}(x)$$

a potom $L_r = \deg f^{(r)}(x)$. Protože

$$\deg f^{(r-1)}(x) \leq L_{r-1} \text{ a } \deg[x^{r-m} f^{(m-1)}(x)] \leq r - m + L_{m-1},$$

tím dostáváme

$$L_r \leq \max[L_{r-1}, r - m + L_{m-1}] \leq \max[L_{r-1}, r - L_{r-1}].$$

Nyní připomeneme větu (6.2), když $f^{(r)}(x)$ produkuje $a_1, \ldots, a_r$, potom $L_r = \max[L_{r-1}, r - L_{r-1}]$. Nyní musíme dokázat, že posuvný registr $(L_r, f^{(r)}(x))$ tvoří požadovanou posloupnost. To můžeme doloži přímím výpočtem rozdílů mezi požadovaným a_j a výstupem filtru

$$\begin{aligned} a_j - \left(-\sum_{i=1}^{L_r} f_i^{(r)} a_{j-i} \right) = \\ = a_j + \sum_{i=1}^{L_{r-1}} f_i^{(r-1)} a_{j-i} - \Delta_r \Delta_m^{-1} \left[a_{j-r+m} + \sum_{i=1}^{L_{m-1}} f_i^{(m-1)} a_{j-r+m-i} \right] = \\ = \begin{cases} 0 & j = L_r, L_{r+1}, \ldots, r-1 \\ \Delta_r - \Delta_r \Delta_m^{-1} \Delta_m = 0, & j = r \end{cases}. \end{aligned}$$

Tedy posuvný registr $(L_r, f^{(r)}(x))$ generuje $a_1, \ldots, a_r$. A zejména $(L_n, f^{(n)}(x))$ má a výstupu posloupnost $a_1, \ldots, a_n$ což dokazuje větu (6.3). □

Věta 6.4 *Nechť je v jakémkoli číselném oboru daná posloupnost $a_1, \ldots, a_n$. Počáteční podmínky $f^{(0)}(x) = 1$, $t^{(0)}(x) = 1$ a $L_0 = 0$ nechť jsou následující rovnice použity pro výpočet $f^{(n)}(x)$*

$$\Delta_r = \sum_{j=0}^{n-1} f_j^{(r-1)} a_{r-j},$$

$$L_r = \delta_r (r - L_{r-1}) + (1 - \delta_r) L_{r-1},$$

$$\begin{pmatrix} f^{(r)}(x) \\ t^{(r)}(x) \end{pmatrix} = \begin{pmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1} \delta_r & (1 - \delta_r) x \end{pmatrix} \begin{pmatrix} f^{(r-1)}(x) \\ t^{(r-1)}(x) \end{pmatrix}.$$

pro $r = 1, \ldots, n$ kde $\delta_r = 1$ když obě podmínky $\Delta_r \neq 0$ a $2L_{r-1} \leq r - 1$ platí, jinak $\delta_r = 0$. Potom $f^{(2t)}(x)$ je polynom s nejmenším stupněm s vlastnostmi $f_0^{(2t)} = 1$ a

$$a_r + \sum_{j=1}^{n-1} f_j^{(2t)} a_{r-j} = 0, \; r = L_{2t} + 1, \ldots, 2t.$$

Celý Berlekampův-Masseyův algoritmus je shrnut v Algoritmu 8.

Algoritmus 8 Berlekampův-Masseyův algoritmus

Vstup: Vektor popisující Toeplitzovu matici $(a_0, \ldots, a_n, \ldots, a_{2n-2})$ a vektor pravých stran $(-a_n, \ldots, -a_{2n-1})$.

Výstup: Vektor řešení stran $(f_1, \ldots, f_n)$.

1: Inicializujme $f(x) = t(x) = 1, L = 0, r = 1$.
2: **for** $r = 1, \ldots, n$ **do**
3: $\quad \Delta_r = \sum_{j=0}^{n-1} f_j a_{r-j}$
4: $\quad$ **if** $\Delta_r \neq 0$ a zároveň $2L \leq r - 1$ **then**
5: $\quad\quad \delta = 1$
6: $\quad\quad L \leftarrow r - L$
7: $\quad$ **else**
8: $\quad\quad \delta = 0$
9: $\quad$ **end if**
10: $\quad \begin{pmatrix} f(x) \\ t(x) \end{pmatrix} \leftarrow \begin{pmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1}\Delta_r & (1-\delta_r)x \end{pmatrix} \begin{pmatrix} f(x) \\ t(x) \end{pmatrix}$
11: **end for**
12: Vrátíme vektor konstant polynomu $f(x)$.

Kapitola 7

Závěr

Podařilo se nám nastudovat a naprogramovat rychlé algoritmy, které využívají vlastnostní Toeplitzovy matice. Tyto algoritmy umožňují řešit různé úlohy s menší asymptotickou složitostí a paměťovou náročností. Díky struktuře Toeplitzovy matice lze celou matici reprezentovat vektorem a tím výrazně snížit množství paměti nutné na výpočet. Vektor má lineární náročnost na paměť, kdežto obecná matice kvadratickou. Struktura Toeplitzovy matice nám umožňuje efektivněji využít standardní postupy pro řešení úloh a tím snížit asymptotickou složitost. Také umožňuje použití metody rozděl a panuj, kdy je problém dělen na menší problémy až zbudou jen triviální problémy. Tyto problémy poté jednoduše vyřešíme a složíme z nich řešení celé úlohy.

Problém těchto algoritmů je obtížné využití paralelizace. Na současných počítačích, které mají více výpočetních jednotek, tak dochází k neoptimálnímu využití jejich výkonu. Implementace těchto standardních úloh (řešení lineárních rovnic - LU, QR faktorizace) v systému Mathematica je velmi rychlá. Protože námi naprogramované algoritmy nebyly speciálně optimalizovány pro jazyk Mathematica, tak i když mají asymptotickou složitost menší, jsou, až na vyjímku, standardní postupy řešení v systému Mathematica rychlejší. Násobení Toeplitzovy matice vektorem, realizované pomocí rychlého násobení polynomů, je rychlejší, než násobení matice vektorem v systému Mathematica.

Přesnost řešení (numerickou stabilitu) jsme ověřovali na jednoduchém příkladu. Námi implementované algoritmy měli v nejhorším případě o dva řády menší přesnost. Nejlepší výsledek byl přesný na stejný počet desetiných míst, ale norma chyby byla větší, než u algoritmu ze systému Mathematica. Tato chyba se v jiných případech může lišit. Nevěnovali jsme se odvození přesného výrazu pro určení numerické chyby, protože tato úloha přesahuje rozměry bakalářské práce.

Dodatek A

Implementace v systému Mathematica

Popisované algoritmy s testy jsou implementované v systému Mathematica a jsou obsaženy na přiloženém CD.

Literatura

Dario Bini and Victor Pan. *Polynomial and Matrix Computations: Volume 1: Fundamental Algorithms*. Springer, August 1994. ISBN 0817637869.

Richard E. Blahut. *Fast Algorithms for Signal Processing*. Cambridge, 2010. ISBN 9780521190497.

Richard P. Brent, Fred G. Gustavson, and David Y. Yun. Fast solution of toeplitz systems of equations and computation of Padé approximants. *Journal of Algorithms*, 1(3):259–295, 1980. ISSN 0196-6774. doi: 10.1016/0196-6774(80)90013-9.

Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. ISBN 0801854148.

P. Kujan, M. Hromčík, and Šebek M. Complete fast analytical solution of the optimal odd single-phase multilevel problem. *IEEE Transactions on Industrial Electronics*, 57(7):2382–2397, jul. 2010. ISSN 0278-0046. doi: 10.1109/TIE.2009.2034677.

Petr Olšák. *Úvod do algebry, zejména lineární*. Fakulta elektrotechnická ČVUT, 2008. ISBN 978-80-01-03775-1.

Victor Y. Pan. *Structured Matrices and Polynomials, Unified Superfast Algorithms*. Birkhäuser/Springer, 2001. ISBN 0-8176-4240-4.