ივ.ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტის
ზუსტ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტის
მათემატიკის დეპარტამენტი

# პოლინომიალური მატრიც-ფუნქციის სპექტრალური ფაქტორიზაციის ალგორითმის შესახებ

გიორგი რუხაია

ხელმძღვანელი: ასოცირებული პროფესორი ლაშა ეფრემიძე

თბილისი 2014

# სარჩევი

# ანოტაცია

საბაკალავრო ნაშრომში აღწერილია პოლინომიალური მატრიც-ფუნქციის სპექტრალური ფაქტორიზაციის ახალი ალგორითმი. ეს ალგორითმი ეყრდნობა სპექტრალური ფაქტორიზაციის არსებობის თეორემის ახლახანს გამოქვეყნებულ მარტივ დამტკიცებას. გაკეთდა შესაბამისი პროგრამული რეალიზაცია.

პროგრამის შექმნისას საფუძვლიანად დამუშავდა სკალარული ფაქტორიზაციის ვილსონის მეთოდი და მისი გაუმჯობესება, რომელიც არსებით როლს თამაშობს საბოლოო ამოცანის სწრაფ და ზუსტ გადაწყვეტაში. მოძიებულ იქნა პოლინომის კომპლექსური ფესვების პოვნის უახლესი მეთოდები და მათი პროგრამული უზრუნველყოფა. შესწავლილ იქნა დადებითად განსაზღვრული მატრიცის ხოლეცკის ფაქტორიზაციის მეთოდი და ეს მეთოდი გაუმჯობესებული იქნა არაუარყოფითად განსაზღვრული მატრიცებისთვის.



# Abstract

This bachelor thesis describes new algorithm of spectral factorization for polynomial matrix-functions. This algorithm is based on the recently published proof of spectral factorization existence theorem. Software implementation was created.

While creating program, thoroughly was studied Wilsons method for scalar factorization and it’s improvement, which plays essential role for fast and effective solution of final problem. Newest methods for complex polynomial root finding were searched and studied together with their implementations. Cholesky factorization for positively defined matrices was studied and improved to work on positive semidefinite matrices.

# პოლინომიალური მატრიცის ფაქტორიზაცია

პოლინომიალური მატრიცის ფაქტორიზაციის ამოცანა:

განვიხილოთ შემდეგი სახის $m \times m$ რიგის მატრიც-ფუნქცია:

$$S(z) = \sum_{n=-N}^{N} C_n z^n \quad C_n \in \mathbb{C}^{m\times m} \tag{1}$$

რომელიც დადებითად განსაზღვრულია თითქმის ყველგან $\mathbb{T} := \{z \in \mathbb{C} : |z| = 1\}$ სიმრავლეზე. ამ პირობებში არსებობს S-ის დაშლა $S(z) = S^+(z)S^-(z),\ z \in \mathbb{C}\backslash\{0\}$ სახით, სადაც $S^+(z) = \sum_{n=0}^{N} A_n z^n$ არის არასინგულარული წრის შიგნით, ანუ $\det\ S^+(z) \neq 0 \quad |z| < 1$ და $S^-(z) = \overline{S^+(1/\overline{z})}^T = \sum_{n=0}^{N} A_n^* z^{-n}$ მისი შეუღლებულია. $S^+$ ერთადერთია უნიტარულ მატრიცზე მარჯვნიდან გამრავლებამდე სიზუსტით. ეს ამოცანა პირველად დასვა ვინერმა [2] ზოგადად ნებისმიერი ინტეგრებადი მატრიც-ფუნქციისთვის ინტეგრებადი დეტერმინანტის ლოგარითმით და მანვე დაამტკიცა ფაქტორიზაციის არსებობა და ერთადერობა მრავლაგანზომილებიანი სტოქასტური ანალიზის გამოყენებით. მას შემდეგ ამ თეორემის დამტკიცების და $S^+$ -ის დათვლის მრავალი მეთოდი შემუშავდა.

ამ ნაშრომში განიხილება [1]-ში მოყვანილი კონსტრუქციული დამტკიცების საშუალებით $S^+$ -ის სწრაფად და ეფექტურად დათვლის შესაძლებლობა. ეს დამტკიცება ეფუძნება ხოლეცკის ფაქტორიზაციის ალგორითმს და ახდენს მის განზოგადოებას სკალარული მტრიცებიდან მატრიც-ფუნქციებზე.

ამ დამტკიცების პირველი ეტაპი არის მატრიცის ხოლეცკის ფაქტორიზაცია:

$$b_{11} = \sqrt{a_{11}},\ \ b_{k1} = a_{k1}/\sqrt{a_{11}}^*,\ k = 2, 3, \ldots, m,$$
$$b_{nn} = \sqrt{a_{nn} - \sum_{j=1}^{n-1} b_{nj} b_{nj}^*},\ \ b_{kn} = \left(a_{kn} - \sum_{j=1}^{n-1} b_{kj} b_{nj}^*\right)/b_{nn}^*,$$
$$n = 2, 3, \ldots, m.\ \ k = n+1, \ldots, m.$$

სადაც ფესვის ამოღებაში იგულისხმება ერთგანზომილებიანი ფაქტორიზაცია.

# ერთგანზომილებიანი ფაქტორიზაცია

განვიხილოთ ნამდვილკოეფიციენტებიანი სიმეტრიული პოლინომი :

$$a(z) = \sum_{j=-m}^{m} a_j z^j,\ a_j = a_{-j} \in \mathbb{R},\ -m \le j \le m$$

თუკი ეს პოლინომი დადებითად განსაზღვრულია ერთეულოვან წრეწირზე, მაშინ არსებობს მისი ერთადერთი დაშლა შემდეგი სახით:

$a(z) = \gamma(z)\gamma(z^{-1}), \quad z \in \mathbb{C} \setminus \{0\}$ სადაც $\gamma(z) = \sum_{j=0}^{m} \gamma_j z^j \quad \gamma_0 > 0$

პოლინომის ყველა ფესვი ერთეულოვანი წრის გარეთაა.

ამ ფაქტორიზაციის კლასიკურ და აპრობირებულ იტერაციულ ალგორითმს წარმოადგენს ვილსონის მეთოდი [4] რომელიც გულისხმობს $\gamma^{(0)}(z) = \sum_{j=0}^{m} \gamma_j^{(0)} z^j$ საწყისი მიახლოების აღებას მოთხოვნილი თვისებებით და გარანტიას იძლევა რომ იტერაციის ყოველ ბიჯზე ახალი მიახლოება ამ თვისებებს შეინახავს. ვილსონის იტერაცია გულიხმობს შემდეგი სისტემის ამოხნას :

$\mathcal{J}^{(k)}(\gamma^{(k+1)} - \gamma^{(k)}) = b^{(k)}$ სადაც $b^{(k)}$ არის $a(z) - \gamma^{(k)}(z)\gamma^{(k)}(z^{-1})$ პოლინომის კოეფიციენტებისაგან შემდგარი ვექტორი, ხოლო $\mathcal{J}^{(k)}$ აქვს სახე:

$$\mathcal{J}^{(k)} = \begin{bmatrix} \gamma_0^{(k)} & \cdots & \cdots & \gamma_m^{(k)} \\ & \ddots & & \vdots \\ & & \ddots & \vdots \\ & & & \gamma_0^{(k)} \end{bmatrix} + \begin{bmatrix} \gamma_0^{(k)} & \cdots & \cdots & \gamma_m^{(k)} \\ \vdots & & \iddots & \\ \vdots & \iddots & & \\ \gamma_m^{(k)} & & & \end{bmatrix}$$

თუმცაღა უკვე არსებობს ვილსონის იტერაციაზე დამყარებული მეთოდები რომლებიც ასწრაფებენ გამოთვლებს და აუმჯობესებენ სიზუსტეს. ნაშრომში განხილული და პროგრამულად განხორციელებულია [3] სტატიაში მოცემული ალგორითმი, რომელიც გარდაქმნის ვილსონის გამოსახულებას და იღებს ახალ სისტემას, რომელიც ეფექტურად იხსნება პრეკონდენცირებული გრადიენტის მეთოდით.

კერძოდ საწყისი სისტემა გადაიწერება პოლინომიალური სახით:

$$\gamma^{(k)}(z)(\gamma^{(k+1)}(z^{-1}) - \gamma^{(k)}(z^{-1})) + \gamma^{(k)}(z^{-1})(\gamma^{(k+1)}(z) - \gamma^{(k)}(z) = a(z) - \gamma^{(k)}(z)\gamma^{(k)}(z^{-1}),$$

საიდანაც ვიღებთ :

$$\gamma^{(k)}(z)\gamma^{(k+1)}(z^{-1}) + \gamma^{(k)}(z^{-1})\gamma^{(k+1)}(z) = a(z) + \gamma^{(k)}(z)\gamma^{(k)}(z^{-1})$$

ანუ ახალი მიახლოების აგება დავიდა შემდეგ ამოცანაზე:

გვაქვს $b(z) = \sum_{j=-m}^{m} b_j z^j,\ b_j = b_{-j} \in \mathbb{R},\ -m \le j \le m$ $\sigma(z) = \sum_{j=0}^{m} \sigma_j z^j,\ \sigma_j \in \mathbb{R},$ სადაც $\sigma_0 > 0$ და $\sigma(z)$ არ ხდება ნული წრის შიგნით. ვიპოვოთ $\alpha(z) = \sum_{j=0}^{m} \alpha_j z^j$ ისეთი რომ

$$\sigma(z)\alpha(z^{-1}) + \sigma(z^{-1})\alpha(z) = b(z) \qquad (2)$$

ამ ამოცანის ამოხსნის მრავალი მიდგომა არსებობს, მაგრამ [3] ში მოყვანილია მიდგომა რომელიც აქტიურად იყენებს წრის შიგნით არანულოვნებას. სახელდობრ (2) გადაიწერება შემდგი სახით:

$$\frac{\alpha(z^{-1})}{\sigma(z^{-1})} + \frac{\alpha(z)}{\sigma(z)} = \frac{b(z)}{\sigma(z)\sigma(z^{-1})} \qquad (3)$$

სადაც $g(z) = \frac{\alpha(z)}{\sigma(z)}$ ანალიზურია $G = \{z \in \mathbb{C} : |z| < |\nu|\}$ -ზე, როცა $\nu$ არის $\sigma(z)$ -ის მოდულით უმცირესი ფესვი. ანუ გვაქვს:

$$g(z) = \frac{g_0}{2} + \sum_{i=1}^{\infty} g_i z^i, \quad \forall z \in G,$$

შესაბამისად (3)-ის მარჯვენა მხარე გაიშლება ლორენცის მწკრივად $\mathcal{G} = \{z \in \mathbb{C} : \frac{1}{|\nu|} < |z| < |\nu|\}$ არეში:

$$g(z) + g(z^{-1}) = \frac{\alpha(z^{-1})}{\sigma(z^{-1})} + \frac{\alpha(z)}{\sigma(z)} = \sum_{i \in \mathbb{Z}} g_{|i|} z^i.$$

და ასევე $h(z) = \frac{1}{\sigma(z)\sigma(z^{-1})} = \sum_{i \in \mathbb{Z}} h_{|i|} z^i, \quad \forall z \in \mathcal{G},$ აქედან ვიღებთ

$$\begin{bmatrix} h_0 & h_1 & \dots & \dots & \dots & h_{2m} \\ h_1 & h_0 & h_1 & & & h_{2m-1} \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ h_{2m-1} & \dots & \dots & h_1 & h_0 & h_{11} \\ h_{2m} & \dots & \dots & \dots & h_1 & h_0 \end{bmatrix} \begin{bmatrix} b_m \\ \vdots \\ b_0 \\ b_1 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} g_m \\ \vdots \\ g_0 \\ g_1 \\ \vdots \\ g_m \end{bmatrix}$$

ანუ დავედით შემდეგ ამოცანაზე:

მოცემულია $\sigma(z) = \sum_{j=0}^{m} \sigma_j z^j,\ \sigma_j \in \mathbb{R},$ ისეთი, რომ არანულოვანია წრის შიგნით, უნდა ვიპოვოთ $h(z) = \frac{1}{\sigma(z)\sigma(z^{-1})}$ ფუნქციის ლორენცის მწკრივის 0 დან 2m მდე კოეფიციენტები.

[3] შივე მტკიცდება რომ ამ მატრიცის შებრუნებულ მატრიცს საზგოადოდ აქვს სახე:

$$H_k^{-1} = (\sigma(Z_k))^T \sigma(Z_k) - \hat{\sigma}(Z_k)(\hat{\sigma}(Z_k))^T \quad \text{სადაც} \qquad \hat{\sigma}(z) = z^k \sigma(z^{-1})$$

ხოლო $$Z_k = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}.$$

ხოლო თუ დავუბრუნდებით (2) ამოცანის პირობებს, უნდა ვიპოვოთ შემდეგი მატრიცის შებრუნებული:

$$\mathcal{P}^{(k)} = (\gamma^{(k)}(Z_{2m+1}))^T \gamma^{(k)}(Z_{2m+1}) - \hat{\gamma}^{(k)}(Z_{2m+1})(\hat{\gamma}^{(k)}(Z_{2m+1}))^T$$

რისთვისაც იხსნება $\mathcal{P}^{(k)} h^{(k)} = e_1$ სისტემა. ანუ იტერაციის ყოველ ბიჯზე ახალი მიახლოებით შედგენილი მატრიცისთვის ვპოულობთ შებრუნებულს და მისი საშუალებით, ზემოთ მოყვანილი აგების საფუძველზე, ვპოულობთ ახალ მიახლოებას, ხოლო $\mathcal{P}^{(k)} h^{(k)} = e_1$ სისტემის ამოხსნისას გამოიყენება პრეკონდენცირებული შეუღლებული გრადიენტის მეთოდი სადაც კონდენსატორად აიღება წინა იტერაციისას მიღებული შებრუნებული.

ანუ მივიღეთ მიახლოების დათვლის ეფექტური ალგორითმი, რომლის რეალიზებისათვის საკმარისია მოახლოებით O(n logn) გამოთვლითი ოპერაცია.

# ალგორითმი

**საწყისი მონაცემები:**

$a(\mathrm{z}) = \sum_{j=-m}^{m} a_j z^j$ სიმეტრიული პოლინომის კოეფიციენტები, $\gamma^0(\mathrm{z}) = \sum_{j=0}^{m} \gamma_j^0 z^j$ საწყისი მიახლოების კოეფიციენტები ( $\gamma_0^0 > 0$ და $\gamma^0(\mathrm{z})$ არ ხდება ნული ერთეულოვანი წრის შიგნით), $h = h_0, h_1, ..., h_{2m}$ კოეფიციენტები $H^{(0)}$ მატრიცის შესადგენად (თუ არ გვაქვს პირველ ეტაპზე, შეგვიძია გამოვიყენოთ ერთეულოვანი მატრიცი).

**ალგორითმი:** k=0,1,2...imax (imax მაქსიმალური რაოდენობა იტერაციებისა)

1. გამოვთვალოთ $h^{(k)}$ ვექტორი, როგორც $P^{(\mathrm{k})}$-ს შებრუნებულის პირველი სვეტი, ანუ ამოვხსნათ სისტემა $P^{(\mathrm{k})} h^{(k)} = e_1$, კონდენსირებული გრადიენტის მეთოდით და კონდენსატორი $H^{(\mathrm{k})}$-თი (ანუ $H^{(\mathrm{k})}$ არის $P^{(\mathrm{k})}$-ს შებრუნებულის მიახლოება).

2. გამოვთვალოთ $b^{(\mathrm{k})}(\mathrm{z}) = a(\mathrm{z}) + \gamma^{(\mathrm{k})}(\mathrm{z})\gamma^{(\mathrm{k})}(\mathrm{z}^{-1})$

3. გამოვთვალოთ $g(\mathrm{z})$ როგორც $H^{(\mathrm{k}+1)} b$ ვექტორის პირველი m კოეფიციენტის მქონე პოლინომი ( $g_0$ ანუ პირველი კოეფიციენტი უნდა გავანახევროთ).

4. ავაგოთ ახალი მიახლოება $\gamma^{(\mathrm{k}+1)}(\mathrm{z}) = g(\mathrm{z})\gamma^{(\mathrm{k})}(\mathrm{z})$ და დავტოვოთ ნამრავლის პირველი m+1 კოეფიციენტი.

# ხოლეცკის ფაქტორიზაცია

## ნახევრად დადებითად განსაზღვრული მატრიცისთვის

რამდენადაც ცნობილია, დადებითად განსაზღვული მატრიცისთვის არსებობს და ერთადერთია მისი ხოლეცკის ფაქტორიზაცია: $\mathbf{A} = \mathbf{LDL}^*$ სადაც $\mathbf{L}$ ქვედასამკუთხაა დიაგონალებზე 1-ით ხოლო $\mathbf{L}^*$ მისი ერმიტულად შეუღლებული. მისი ელემენტები გამოითვლება ფორმულით:

$$b_{11} = \sqrt{a_{11}}, \quad b_{k1} = a_{k1}/\sqrt{a_{11}}{}^*, \quad k = 2, 3, \ldots, m,$$
$$b_{nn} = \sqrt{a_{nn} - \sum_{j=1}^{n-1} b_{nj} b_{nj}^*}, \quad b_{kn} = \left(a_{kn} - \sum_{j=1}^{n-1} b_{kj} b_{nj}^*\right) / b_{nn}^*,$$
$$n = 2, 3, \ldots, m. \quad k = n+1, \ldots, m.$$

ხოლეცკის ფაქტორიზაციის არსებობა მტკიცდება ნახევრად დადებითად განსაზღვრული მატრიცებისთვისაც, თუმცა ამ შემთხვევაში იკარგება ერთადერთობა და ამასთან რთულდება მისი გამოთვლაც, რამდენადაც ფაქტორიზაციის წამყვანი ელემენტი შეიძლება გახდეს ნული.

ამ პრობლემის გადასაჭრელად საჭიროა შემდეგი

**ლემა:**

$$A = \begin{pmatrix} a_{11} & L^* \\ L & A_0 \end{pmatrix} \geq 0 \rightarrow A_0 - L a_{11} L^* \geq 0$$

**დამტკიცება:**

ერთის მხრივ, ყველა ნახევრად დადებითად განსაზღვრული მატრიცი წარმოდგება $BB^*$ სახით, ხოლო მეორეს მხრივ დიაგონალის ყველა ელემენტი არაუარყოფითი ნამდვილი რიცხვია, ამასთან

$$\begin{pmatrix} 1 & 0 \\ -L a_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} a_{11} & L^* \\ L & A_0 \end{pmatrix} \begin{pmatrix} 1 & -L^* a_{11}^{-1} \\ 0 & I \end{pmatrix} = \begin{pmatrix} a_{11} & 0 \\ 0 & A_0 - L a_{11} L^* \end{pmatrix} \equiv A'$$

ანუ თუ $BB^*$ არის $A$-ს წარმოდგენა, მაშინ $A'$ წარმოდგება $CC^*$ სახით სადაც

$$C = \begin{pmatrix} 1 & 0 \\ -L a_{11}^{-1} & I \end{pmatrix} B$$

ანუ $A'$ ნახევრად დადებითად განსაზღვრულია, რაც თავისთავად გულისხმობს $A_0 - La_{11}L^*$ მატრიცის ნახევრად დადებითად განსაზღვრულობას.

ხოლეცკის ფაქტორიზაციის ყოველი სტრიქონის შევსება შეგვიძლია განვიხილოთ როგორც გარდასახვა $A \to A - La_{11}L^* \to A_0 - L_1 a_{11} L_1^*$ სადაც $L$ შესაბამისი სვეტია, ხოლო $L_1$ მას ჩამოჭრილი პირველი ელემენტი. ამიტომ თუ რაიმე ეტაპზე გადსვლისას წამყვანი ელემენტი გახდა ნული, ეს ნიშნავს რომ $A = BB^*$ წარმოდგენაში $a_{11} =< b_1, b_1^* >= 0 \to b_1 = 0$ ამიტომ $a_{1i} = a_{i1} =< b_1, b_i^* >= 0$ და განსახილველია მომდევნო ეტაპი. დამტკიცებული ლემის თანახმად ყოველ ახალ ეტაპზე მიიღება ნახევრად დადებითად განსზღვრული მატრიცი, ამიტომ ყველა ეტაპზე ეს მსჯელობა კორექტულია, რაც იმას ნიშნავს რომ ნახევრად დადებითად გასნაზღვრული მატრიცის ფაქტორიზაციისას წამყვანი ელემნტის ნულობა ნიშნავს შესაბამისი სვეტის განულებას და შემდეგ ეტაპზე გადასვლას.

# პოლინომიალური მატრიცის ფაქტორიზაციის ამონახსნის აგების ალგორითმი

პოლინომიალური მატრიცის ფაქტორიზაციის ამოცანა:

განვიხილოთ შემდეგი სახის $m \times m$ რიგის მატრიც-ფუნქცია:

$$S(z) = \sum_{n=-N}^{N} C_n z^n \quad C_n \in \mathbb{C}^{m\times m}$$

რომელიც დადებითად განსაზღვრულია თითქმის ყველგან $\mathbb{T} := \{z \in \mathbb{C} : |z| = 1\}$ სიმრავლეზე. ვიპოვოთ S-ის დაშლა $S(z) = S^+(z)S^-(z),\ z \in \mathbb{C}\backslash\{0\}$სახით, სადაც $S^+(z) = \sum_{n=0}^{N} A_n z^n$ არის არასინგულარული წრის შიგნით, ანუ $\det S^+(z) \neq 0$ $|z| < 1$ და $S^-(z) = \overline{S^+(1/\overline{z})}^T = \sum_{n=0}^{N} A_n^* z^{-n}$ მისი შეუღლებულია.

1. ჩავატაროთ ხოლეცკის ფაქტორიზაცია სადაც ფესვის ამოღება განიმარტება როგორც ერთგანზომილებიანი ფაქტორიზაცია.

$$b_{11} = \sqrt{a_{11}},\quad b_{k1} = a_{k1}/\sqrt{a_{11}}^*,\ k = 2, 3, \ldots, m,$$
$$b_{nn} = \sqrt{a_{nn} - \sum_{j=1}^{n-1} b_{nj}b_{nj}^*},\quad b_{kn} = \left(a_{kn} - \sum_{j=1}^{n-1} b_{kj}b_{nj}^*\right)/b_{nn}^*,$$
$$n = 2, 3, \ldots, m.\quad k = n+1, \ldots, m.$$

მივიღებთ $S(z) = S_0^+(z)S_0^-(z)$სადაც $S_0^+(z)$წრის შიგნით ანალიზური, რაციონალური ფუნქციაა.

2. თუ $S_0^+(z)$მატრიცის $s_{ij}$ ელემენტს აქვს l რიგის პოლუსი წრის შიგნით რაიმე a წერტილში, მაშინ მას გავამრავლებთ უნიტარულ მატრიცზე სადაც jj ადგილას არის$u(z) = (z-a)/(1-\overline{a}z)$ხოლო მატრიცია $U(z) = \mathrm{diag}[1, \ldots, u^l(z), \ldots, 1]$ ახალი მატრიცი კვლავ იქნება ფაქტორიზაციის თვისების მქონე. ამ წესით გაქრება ყველა პოლუსი და მივიღებთ ფაქტორიზაციის მატრიცს რომელსაც წრის შიგნით არ აქვს პოლუსი.
3. მიღებულ მატრიცს თუ წრის შგნით აქვს ნულები, ანუ მისი დეტერმინანტი ნულდება რაიმე a წერტილში, მაშინ გავამრავლოთ ისეთ უნიტარულ მატრიცზე რომ ნამრავლის პირველი სტრიქონი განულდეს და შემდეგ გავამრავლოთ $\mathrm{diag}[u(z), 1, \ldots, 1]$ მატრიცზე სადაც $u(z) = (1-\overline{a}z)/(z-a)$. ამ გზით გავაქრობთ ყველა ფესვს და მივიღებთ ახალ მატრიცს რომელსაც კვლავ ექნება ფაქტორიზაციის თვისება და არ ექნება არც პოლუსი არც ფესვი წრის შიგნით.

## დანართი (ერთგანზომილებიანი ფაქტორიზაციის პროგრამა)

```
#include <iostream>
#include <fstream>
#include <complex>
#include <valarray>
#include <time.h>
using namespace std;

const double PI = 3.141592653589793238460;
double EPS = 1e-9;
#define M_LOG2E 1.44269504088896340736 //log2(e)

inline long double log2(const long double x){
    return  log(x) * M_LOG2E;
}

double * P;
int PCGI=5;
int WI=5;
typedef valarray< complex<double> > CArray;


void fft(CArray& x);
void ifft(CArray& x);
double *conv(double * a,int n, double *b,int m,int p);
void conv(double * a,int n, double *b,int m,int p,double * ans);
double * sub1 (int n);
double skal(double *a, double *b,int n);
double * sum(double *p,double *q, int n, double a=1,double b=1);
void sum(double *p,double *q, int n,double *ans, double a=1,double b=1);
double sum(double *p,int n);
double * reverse(double *r,int n);
void bewd(double * mas,int n);
void reverse(double *r,int n,double *ans);
void PCG(double * res,double *hk,int n);
double * HK(double *h,double *vec,int n);
double * PK(double *res, double * vec,int n);

int main()
{
	freopen("out.txt","w",stdout);
	int n;
	cin>>n;
	int m=2*n+1;
	double *h,*result,*b,*tmp,*g;
	result = sub1(n);
	h=new double [m];
	b=new double [m];
	bewd(result,n+1);
	for(int i=0;i<m;i++)
	{
		h[i]=0;
	}
	h[0]=1;
	for(int i=0;i<WI;i++)
	{
		cout<<"Wilsonis "<<i<<"uri iteracia"<<endl;
		PCG(result,h,m);
		//cout<<"  h gaumjobesebuli"<<endl;
		//bewd(h,m);
		tmp=reverse(result,n+1);
		g=conv(result,n+1,tmp,n+1,m);
```

```
                //cout<<"  miaxloeba "<<endl;
                //bewd(g,m);
                sum(P,g,m,b);
                //cout<<"  b "<<endl;
                //bewd(b,m);
                delete [] g,tmp;
                g=HK(h,b,m);
                //cout<<"  g "<<endl;
                //bewd(g,m);
                tmp=reverse(g,n+1);
                tmp[0]/=2;
                conv(result,n+1,tmp,n+1,n+1,result);
                delete [] g,tmp;
                cout<<" miaxloeba ";
                bewd(result,n+1);
        }
}



double * HK(double *h,double *vec,int n)
{
        h[0]/=2;
        double *k,*p,*tmp;
        k=conv(h,n,vec,n,n);
        tmp=reverse(vec,n);
        p=conv(h,n,tmp,n,n);
        reverse(p,n,tmp);
        sum(k,tmp,n,p);
        delete [] k,tmp;
        h[0]*=2;
        return p;
}

double * PK(double *res, double * vec,int n)
{
        double *res1,*k,*p,*tmp;
        res1=new double [n];
        for(int i=0;i<=n/2;i++)
                res1[i]=res[i];
        for(int i=n/2+1;i<n;i++)
                res1[i]=0;
        tmp=reverse(vec,n);
        k=conv(res1,n,tmp,n,n);
        reverse(k,n,tmp);
        p=conv(res1,n,tmp,n,n);

        for(int i=0;i<n/2+1;i++)
                res1[i]=0;
        for(int i=n/2+1;i<n;i++)
                res1[i]=res[n-i];
        conv(res1,n,vec,n,n,k);
        reverse(k,n,tmp);
        conv(res1,n,tmp,n,n,k);
        reverse (k,n,tmp);
        sum(p,tmp,n,k,1,-1);
        delete []p,tmp,res1;
        return k;
}

void PCG(double * res,double *hk,int n)
{
        double *r,*d,*q,*s;
        double d0,d1,a,b;
```

```
	r=PK(res,hk,n);
	r[0]=1-r[0];
	for(int i=1;i<n;i++)
	{
		r[i]=-r[i];
	}
	d=HK(hk,r,n);
	d1=skal(r,d,n);
	d0=d1;
	for(int i=0;i<PCGI;i++)
	{
		cout<<i<<"uri PCG iteracia"<<endl;
		q=PK(res,d,n);
		cout<<"PK gamravlebuli veqtorze "<<endl;
		bewd(q,n);
		a=d1/skal(d,q,n);
		cout<<"koeficienti a   "<<a<<endl;
		sum(hk,d,n,hk,1,a);
		cout<<"axali miaxloeba"<<endl;
		bewd(hk,n);
		sum(r,q,n,r,1,-a);
		cout<<"axali nasti"<<endl;
		bewd(r,n);
		s=HK(hk,r,n);
		cout<<"axali prekondicionerze namravli"<<endl;
		bewd(r,n);
		d0=d1;
		d1=skal(r,s,n);
		cout<<"axali skalari d1   "<<d1<<endl;
		b=d1/d0;
		cout<<"koeficienti b   "<<b<<endl;
		sum(s,d,n,d,1,-b);
		cout<<"axali d veqtori"<<endl;
		bewd(d,n);
		delete [] s,q;
	}
	delete [] r,d;
}

void bewd(double * mas,int n)
{
	cout<<endl<<endl;
	for(int i=0;i<n;i++)
	{
		cout<<mas[i]<<endl;
	}
	cout<<endl<<endl;
}

//vectori sebrunebuli komponentebit
//argumentebi: vectori, ganzomileba+ pasuxis vectori void semtxvevasi
double * reverse(double *r,int n)
{
	double * k;
	k=new double [n];
	for(int i=0;i<n;i++)
		k[i]=r[n-1-i];
	return k;
}

void reverse(double *r,int n,double *ans)
{
	for(int i=0;i<n;i++)
```

```
		ans[i]=r[n-1-i];
}

//vectoris elementebis jami
double sum(double *p,int n)
{
	double sum=0;
	for(int i=0;i<n;i++)
	{
		sum+=p[i];
	}
	return sum;
}

//ori veqtoris jami Sesabamisi koeficientebit
//ori toli ganzomilebis veqtori, mati gnazomileba, danisnulebis veqtori void
semtxvevasi
//pirveli da meore koeficientebi
double * sum(double *p,double *q, int n, double a,double b)
{
	double * k;
	k=new double [n];
	for(int i=0;i<n;i++)
	{
		k[i]=a*p[i]+b*q[i];
	}
	return k;
}

void sum(double *p,double *q, int n,double *ans, double a,double b)
{
	for(int i=0;i<n;i++)
	{
		ans[i]=a*p[i]+b*q[i];
	}
}
//skalaruli namravli
//argumentebi: ori tanabari ganzomilebis veqtori da mati ganzomileba
double skal(double *a, double *b,int n)
{
	double sum=0;
	for(int i=0;i<n;i++)
	{
		sum+=a[i]*b[i];
	}
	return sum;
}

double * sub1 (int n)
{
	// mivigot A+
	double *Ap=new double [n+1];
	double r;
	srand((unsigned)time(NULL));
	for(int i=0; i<n+1; i++)
	{
		r = (double)rand() / (double)RAND_MAX;
		Ap[i]= 2*r-1;
		cout<<Ap[i]<<endl;
	}
	// mivigot A+ inv
	double * Ainv=new double [n+1];
	//fout<<"A+ is inverse"<<endl;
	for(int i=0; i<n+1; i++)
```

```
{
	Ainv[i]=Ap[n-i];
}
//mivigot P polinomi
P=conv(Ap, n+1, Ainv,n+1,2*n+1);
for(int i=0; i<2*n+1; i++)
{
	cout<<"p["<<i<<"]   "<<P[i]<<endl;
}
//shevadginot a
int R= 1 << int( floor(log2(100*n - 1)) + 1);
double *a=new double [2*R];
int i=0;
for(; i<n+1; i++)
{
	a[i]=P[n+i];
}
for(; i<2*R-n; i++)
{
	a[i]=0;
}
for(; i<2*R; i++)
{
	a[i]=a[2*R-i];
}
complex<double> o=(0,0);
CArray X(o,2*R);
for( int i=0; i<2*R; i++)
{
	X[i]=complex<double>(a[i],0);
}
fft(X);
double e=1.0/R;
for(int i=0; i<2*R; i++)
{
	if( abs(X[i]) < e )
	{
		X[i]=log(e);//complex<double>(log(e),imag(X[i]));
	}
	else
	{
		X[i]=log(X[i]);
	}
}
ifft(X);
for(int i=0; i<2*R; i++)
{
	a[i]=real(X[i]);
}
a[0] /=2;
a[R] /=2;
for ( int i=R+1; i<2*R; i++)
{
	a[i]=0;
}
for(int i=0;i<2*R;i++)
{
	X[i]=complex<double>(a[i],0);
}
fft(X);
for(int i=0; i<2*R; i++)
{
	X[i]= exp(X[i]);
}
```

```
	ifft(X);
	cout<<endl<<endl;
	double * b=new double [n+1];
	for(int i=0;i<n+1;i++)
	{
		b[i]=real(X[i]);
		Ainv[i]=real(X[n-i]);
	}
	delete [] a;
	a=conv(b,n+1,Ainv,n+1,2*n+1);
	double maxer=0;
	for(int i=0;i<2*n+1;i++)
	{
		cout<<P[i]<<"    "<<a[i]<<"    "<<P[i]-a[i]<<endl;
		if(abs(P[i]-a[i])>maxer)maxer=abs(P[i]-a[i]);
	}
	cout<<"cdomileba sawyisi miaxloebisTvis" << maxer <<endl;
	delete [] Ainv,a;
	X.resize(0);
	return b;
}

double *conv(double * a,int n, double *b,int m,int p)
{
	complex <double> o=(0,0);
	long long deg=1;
	while(deg<n+m)deg*=2;
	CArray a1(o,deg);
	CArray b1(o,deg);
	for(int i=0;i<n;i++)
		a1[i]= complex<double> (a[i],0);
	for(int i=0;i<m;i++)
		b1[i]=complex<double> (b[i],0);
	fft(a1);
	fft(b1);
	for(int i=0;i<deg;i++)
		a1[i]*=b1[i];
	ifft(a1);
	double *k=new double [p];
	for(int i=0;i<p;i++)
		k[i]=real(a1[i]);
	a1.resize(0);
	b1.resize(0);
	return k;
}

void conv(double * a,int n, double *b,int m,int p,double * ans)
{
	complex <double> o=(0,0);
	long long deg=1;
	while(deg<n+m)deg*=2;
	CArray a1(o,deg);
	CArray b1(o,deg);
	for(int i=0;i<n;i++)
		a1[i]= complex<double> (a[i],0);
	for(int i=0;i<m;i++)
		b1[i]=complex<double> (b[i],0);
	fft(a1);
	fft(b1);
	for(int i=0;i<deg;i++)
		a1[i]*=b1[i];
	ifft(a1);
	for(int i=0;i<p;i++)
		ans[i]=real(a1[i]);
```

```
       a1.resize(0);
       b1.resize(0);
}

void fft(CArray& x)
{
    const int N=x.size();
    if (N<=1) return;
    // divide
    CArray even = x[slice(0,N/2,2)];
    CArray odd = x[slice(1,N/2,2)];
    // conquer
    fft(even);
    fft(odd);
    // combine
    for (int k=0;k<N/2;k++)
    {
       complex <double> t=polar(1.0,-2*PI*k/N)*odd[k];
        x[k]=even[k]+t;
        x[k+N/2]=even[k]-t;
    }
}

// inverse fft (in-place)
void ifft(CArray& x)
{
    x=x.apply(conj);// conjugate the complex numbers
       fft( x );// forward fft
    x=x.apply(conj);// conjugate the complex numbers again
    x/=x.size(); // scale the numbers
}
```

# გამოყენებული ლიტერატურა


[1]. Lasha Ephremidze "An Elementary Proof of the Polynomial Matrix Spectral Factorization Theorem."

[2]. N. Wiener and P. Masani, "The prediction theory of multivariate stochastic processes", Acta Math., vol. 98, pp. 111–150, 1957.

[3]. A. Bacciardi ; Luca Gemignani "Iterative refinement techniques for the spectral factorization of polynomials" Proc. SPIE 4791, Advanced Signal Processing Algorithms, Architectures, and Implementations XII, 150 (December 1, 2002);

[4]. G.T Wilson "Factorization of the covariance generating funqction of a pure moving-average process" SIAM J. Num. Anal. 6, pp. 1-7, 1969