TRANSITUS

Received
December 05, 2025

Revised
January 20, 2026

Accepted
January 25, 2026

Published
January 31, 2026

Keywords
Cryptography, Cryptanalysis, Discrete log problem, Index calculus method, Gaussian Integer Method, Algorithms.

An Analytical Study of the Gaussian Integer Method of Index Calculus for the Discrete Logarithm Problem in a Prime Field

Brij Mohan †

†Department of Mathematics, Hansraj College, University of Delhi, Delhi-110007, India.

Corresponding Author: brijmohan6414@gmail.com

ORCID: https://orcid.org/0000-0002-0400-4186

DOI/url: https://journalmanager.transitus.in/index.php/jamss

1 Abstract

This research investigates application of the Gaussian integer method under the index calculus approach for discrete logarithm problem in a prime field. For a prime field $GF(p)$ of prime order p, and g a primitive element of $GF(p)$, the discrete logarithm base g of an arbitrary non-zero $y \in GF(p)$ is an integer x, $0 \leq x \leq p-2$, such that $g^x = y$ in $GF(p)$. It explored the different cryptographic applications and schemes, and analyzes the index calculus method with appropriate examples. This work showcases different algorithms to simplify the discrete logarithmic problem with their advantages and backdraws. The security of many real-world cryptographic schemes depends on the difficulty of computing discrete logarithms in large finite fields. This project is a study of the discrete logarithm problem in prime field with Gaussian Integer Method of Index Calculus and implementation of this method in NTL(Number Theory Library) with C++ programming language, also comparison with other implemented methods to solve discrete logarithms.

2 Introduction

Let $GF(p)$ be a prime field of order p, where p is a prime. Given g, a primitive element of $GF(p)$, and an arbitrary $y \in GF(p)$, the discrete logarithm of y base g is defined as

$$\log_g y = x \Leftrightarrow g^x = y \in GF(p); \quad 0 \leq x \leq p-2. \tag{1}$$

A fundamental difference between discrete logarithm and real logarithm is that the magnitude of y gives us no information about the magnitude of x. The most obvious method of finding the discrete logarithm of y is to simply keep raising g to different powers until we find the specific exponent x such that $g^x = y$ in $GF(p)$. However, if p is very large, this method is computationally infeasible, and while faster algorithms have been developed, the discrete logarithm problem remains intractable (for almost all $y \in GF(p)$) in very large field $GF(p)$. This intractability makes the discrete logarithm problem useful in cryptographic applications, and many such applications have been developed, and implemented. These real-world cryptographic implementations have raised the stakes on the discrete logarithm problem, making the research of faster algorithms to solve the problem a matter of financial, and even national security-it's now imperative to know in how large a field the problem is solvable.

Discrete logarithms in large finite field F_q come into play in cryptography because they appear to have the attributes of a one-way function. It's inverse function ("discrete exponentiation problem" is to find y such that $y = g^x$ where x is given integer and g is primitive element of finite field F_q) is relatively easy to compute. "square-and-multiply" method can compute $g^x \in F_q$ with at most $2\log_2 q$ multiplications and modular reductions. For example,

$$g^{17} = (((g^2)^2)^2)^2.g \tag{2}$$

and computational blowup can be avoided by taking a modular reduction after each multiplication.

3 Cryptography Applications

The simplest cryptographic application that relies on the difficulty of computing discrete logarithms concerns user authentication – for example, verifying passwords on a multi-user computer.It would be very risky to have a file stored in the computer that contained every user's password, yet the computer needs some way to verify the legitimacy of a password. Instead of storing the password x_i for each user i, we can choose a finite field F_q and a primitive element g, and store the values $g^{x_i} = y_i \in F_q$ for each user i. To authenticate a user's password, the computer first calculates g^{x_i}, then compares the result for a match on the stored file. However, even if someone gains access to the stored file, in order to impersonate user i, they would first have to calculate the discrete logarithm of y_i in F_q.

3.1 Diffie and Hellman Problem

The problem with classical, private-key cryptography has always been that, if two user "A" and "B" wanted to communicate privately over a public, insecure channel, they first needed, in some private and secure way, to agree on a shared secret key, which they would both use to encipher and decipher their message to each other.

Diffie and Hellman [5] have invented a key-exchange based on exponentiation in finite field. In it, a finite field F_q and a primitive element $g \in F_q$ are chosen and made public. User A and B who wish to communicate using some standard encryption method, but who do not have a common key for that system, choose random integer a and b, respectively, with $2 \leq a, b \leq q-2$. Then user A transmits g^a to B over a public channel, while user B transmits g^b to A. The common key is then taken to be g^{ab}, which A can be compute by raising

the received g^b to the a power(which only he knows), and which B forms by raising g^a to the b power.
It is clear that an efficient discrete logarithm algorithm would make this scheme insecure, since the publicly transmitted g^a would enable the cryptanalyst to determine a, and he could then determine the key used by A and B. Diffie and Hellman [5] have even conjectured that breaking their scheme is equivalent in difficulty to computing discrete logarithms. This conjecture remains unproved, and so we cannot exclude the possibility that there might be some way to generate g^{ab} from knowledge of g^a and g^b only, without computing either a or b, although it seems unlikely that such a method exists.

3.2 ElGamal Digital Signature Scheme

In 1985, ElGamal [9] proposed a public-key cryptosystem and digital signature scheme in which the public and private keys are the same as in the Diffie-Hellman system. User B can send a message $m \in F_q$ to user A by choosing an integer k, $2 \leq k \leq q-2$ (and it's important to choose a different k for each message), then sending the pair $(g^k, m{y_A}^k)$ to user A. User A knows $-x_A$ (mod q-1), and can calculate ${y_A}^{-k}$ as $(g^k)^{-x_A}$ to recover the message $m = (m{y_A}^k){y_A}^{-k}$.
To implement the ElGamal digital signature scheme, we must restrict ourselves to field of prime order p (see e.g. the description in [10]).To sign a message m, $1 \leq m \leq p-1$, a user A will provide a pair of integers (r, s), $1 \leq r, s \leq p-1$, that satisfy the following properties: a knowledge of x_A is necessary to produce the signature; anyone who know m can use y_A to verify the signature; and any alteration of the message m after the signature was produced will nullify the signature. To calculate r and s, user A chooses an integer k, $1 \leq k \leq p-2$, such that $(k, p-1)$=1 (again, we must use a different k for each message) and calculates.
$r \equiv g^k \ mod\ p$, and
$s \equiv k^{-1}(m - {x_A}^r) \ mod\ p-1; \quad ((k, p-1) = 1 \Rightarrow \exists\ k^{-1}\ (mod\ p-1))$.
Thus, these r, s satisfy

$$g^m \equiv g^{x_A r + ks} \equiv (g^{x_A})^r (g^k)^s) \equiv (y_A)^r r^s\ (mod\ p) \tag{3}$$

so to verify the signature, anyone can calculate g^m and $(y_A)^r r^s\ (mod\ p)$, and check that they are equal.

4 Index Calculus Method

The fastest method for computing discrete logarithms is known as the index calculus method. The running time of the algorithm has the form

$$L[q, \alpha, c] = exp((c + o(1))(\ln q)^{\alpha}(\ln \ln q)^{1-\alpha}), \quad 0 < \alpha < 1 \tag{4}$$

where c is constant ($o(1) \to 0,\ as\ q \to \infty$), which is known as sub-exponential (if α were 0, the time would be polynomial in $\ln q\ if \alpha$ were 1, it would be fully exponential in $\ln q$).

4.1 Method in Prime Field

For the field F_p, p prime, whose elements are represented by the set {0,1,..,p-1}, with arithmetic performed modulo p: Let S (the factor base), be the set of all prime integers less than or equal some bound b. An element of ${F_p}^* = F_p - \{0\}$ is said to be smooth with respect to b if, all of its factor are contained in S.
The algorithm proceeds in three stage. In the first stage, we take a random integer z in [1, $p-2$], calculate $g^z\ (mod\ p)$, and see if $g^z\ (mod\ p)$ is smooth. if it is smooth, say

$$g^z\ mod\ p = \prod_{i=1}^{|S|} {p_i}^{\alpha_i}, \quad p_i \in S, \tag{5}$$

then we get an equation in the discrete logarithms to the base g:

$$z \equiv \sum_{i=1}^{|S|} \alpha_i \log_g p_i \ (mod\ p-1), \tag{6}$$

where z and all of the α_i are known. We continue this process until we have generated more than $|S|$ equations. In stage two of the algorithm, we solve the system of equations (5.0.1) to find a unique solution for the $\log_g p_i$. Then in stage three, we are able to find the discrete logarithm of any $y \in {F_p}^*$. To do this, we take z'sat random again, until we find a z such that $yg^z \ (mod\ p)$ is smooth. When we find such a z, we get an equation

$$\log_g y \equiv -z + \sum_{i=1}^{|S|} \alpha_i . \log_g p_i (mod\ p-1), \tag{7}$$

where everything on the right hand side is known.

4.2 Example

For example, in F_{37} with $g = 5$, let S=$\{2,3\}$ and suppose we want to find $\log_5 17$.
Stage 1:
Try $z = 7:\ 5^7 \equiv 18\ (mod\ 37) = 2 \cdot 3^2 \Rightarrow \log_5 2 + 2 \cdot \log_5 3 \equiv (mod 36)$.
Try $z = 6:\ 5^6 \equiv 11\ (mod\ 37)$. *Not smooth.*
Try $z = 14:\ 5^{14} \equiv 28\ (mod\ 37)$. *Not smooth.*
Try $z = 31:\ 5^{31} \equiv 24\ (mod\ 37) = 2^3 \cdot 3 \Rightarrow 3 \cdot \log_5 2 + \log_5 3 \equiv 31\ (mod\ 36)$.
Stage 2:
Subtracting 3 times the first equation from the second, and using
$5^{-1} \equiv 29\ (mod\ 36)$,we get the solution $\log_5 2 = 11\ and\ \log_5 3 = 34$.
Stage 3:
We want to find $\log_5 17$.
Try $z = 24:\ 17 \cdot 5^{24} \equiv 35\ (mod\ 37)$. *Not smooth.*
Try $z = 15:\ 15 \cdot 5^{15} \equiv 12\ (mod 37) = 2^2 \cdot 3 \Rightarrow \log_5 17 \equiv -15 + 2 \cdot 11 + 1 \cdot 34 \equiv 5\ (mod\ 36)$.
Result :
$\log_5 17 \equiv 5\ (mod\ 36)$.

5 Gaussian Integer Method

Coppersmith, Odlyzko, and Schroeppel [1] proposed three algorithms for finding discrete logarithms in prime fields F_p, each of which has a run time $L[p, \frac{1}{2}, c = 1]$ for the first and second stages, and a run time of $L[p, \frac{1}{2}, c = \frac{1}{2}]$ for stage three. One of these algorithms called the Gaussian integer method, has proved to be the most practical, and the most important in the way it has spurred the development of other discrete logarithm algorithms.
LaMacchia and Odlyzko [2] successfully implemented the algorithm in a prime field of order $\approx 10^{58}$ (and went halfway through stage two with a prime $\approx 10^{67}$). Although their primary motivation was to obtain empirical results on sparse matrix techniques [8], the implementation with a 58 digit prime also broke an authentication scheme that Sun Microcomputers, Inc. had implemented in that field as part of their Network Field System.
The Gaussian integer method was inspired by a subexponential algorithm that ElGamal [12] invented for finding discrete logarithms in field $F_{p^2},\ p \to \infty$. The idea is to perform the index calculus algorithm in an isomorphic copy of F_{p^2}.

5.1 Basic Method

In the Gaussian integer method, we use a very simple mapping of F_p to a subset of Z×Z. Let r be the small negative integer that is also a quadratic residue modulo p – preferably $r \in \{-1, -2, -3, -7, -11, -19, -43, ..\}$, so we can work in a unique factorization domain, though the algorithm can be modified to work in a non-UFD. (If $p \equiv 1 \ mod\ 4$, we can take $r = -1$.) Let W be an integer such that $W^2 \equiv r \ mod\ p$, and let w represent the complex number $\sqrt{r}$. Find two integer $T, V < \sqrt{p}$ such that $T^2 \equiv rV^2 \ mod\ p$ (these T and V can easily find by the $ExtendedEuclideanalgorithm$), and let $p' = T + Vw$. Then the norm $N(p') = p \Rightarrow (p')$ is a maximal ideal of $Z[w]$ and $\frac{Z[w]}{(p')}$ is isomorphic to F_p. In fact, $\phi : \frac{Z[w]}{(p')} \to F_p$ defined by $\phi(e + fw) = e + fW \ mod\ p$ is an isomorphism.

Choose a complex prime $G = a_1 + a_2 w$ that generates the group of units $(\frac{Z[w]}{(p')})^*$ – this G will be the new base for logarithms. Mapping complex numbers $e + fw$ to real numbers $e + fW$, and the base $G = a_1 + a_2$ to $g = a_1 + a_2 W$, preserves logarithms. Let the smoothness bound $b = L[p, \frac{1}{2}, \frac{1}{2}]$, and we let the factor base S contain the integer V, all real primes $\leq b$, and all complex primes $x + yw \in Z[w]$ whose norm is $\leq b$ (including real primes that factor into two complex primes).

Now to generate logarithm equations in stage one, we will sieve through pairs of small (positive or negative) integers (c_1, c_2) looking for pairs that makes $c_1 V - c_2 T$ smooth with respect to the real primes in factor base S. (A pair (kc_1, kc_2), for constant k, will gives us the same equation as (c_1, c_2) so we should avoid using such multiples, e.g. by using only relatively prime pairs). For each (c_1, c_2) that makes $(c_1 T, c_2 V)$ smooth, we check to see if $(c_1 + c_2 w)$ is smooth with respect to the complex primes in the factor base. (Since the norm of a product quadratic integer is the product of their norms, we can test $(c_1 + c_2 w)$ for smoothness by testing it's norm for smoothness with respect to the norms of complex elements in the factor base.) If $(c_1 + c_2 w)$ is also smooth, then we can get an equation among the discrete logarithms base G of elements in the factor base, because

$$c_1 V - c_2 T = V(c_1 + c_2 w) - c_2(T + Vw) \equiv V(c_1 + c_2 w) \ \ mod\ p'. \tag{8}$$

If $c_1 T - c_2 V$ and $c_1 + c_2 w$ are smooth with respect real and complex prime, respectively in factor base. Then

$$(c_1 T - c_2 V) = \prod_{i=1}^{t} {p_i}^{m_i}, \tag{9}$$

$$(c_1 + c_2 w) = \prod_{i=1}^{t'} (x_i + y_i w)^{n_i}, \tag{10}$$

taking logarithm for both equations with base G

$$\log_G(c_1 T - c_2 V) = \sum_{i=1}^{t} m_i \cdot \log_G p_i, \tag{11}$$

$$\log_G(c_1 + c_2 w) = \log_G V + n_i \sum_{i=1}^{t'} \log_G(x_i + y_i w). \tag{12}$$

6 Lanczos Algorithms

Suppose we have the system

$$Rx = b. \tag{13}$$

The lanczos algorithm over the real field works for a positive definite symmetric matrix. In our case R is not symmetric or even square. Hence we apply Lanczos to normal equations

$$R^T Rx = R^T b, \tag{14}$$

rather than directly to equation (14). Suppose $A = R^T R, \tilde{b} = R^T b$, then the system becomes $Ax = \tilde{b}$.

6.1 Standard Lanczos method

Suppose $A = R^T R$ is a symmetric positive definite matrix over GF(2), $\tilde{b} = R^T b$ and w_i's are vectors, then the stndard lanczos algorithm solves $Ax = \tilde{b}$ by iterating

$$w_i = Aw_{i-1} - \sum_{j=0}^{i-1} c_{ij} w_j; \qquad (i > 0), \tag{15}$$

where

$$c_{ij} = \frac{w_j^T A^2 w_{i-1}}{w_j^t A^2 w_j}, \tag{16}$$

untill $w_i = 0$.
By the symmetry of A, we get the following relation :

$$w_i^T A w_j = 0 \qquad (i \neq j) \tag{17}$$

The vectors $w_0, w_1, w_2 \ldots\ldots w_i$are eventually linearly dependent, namely

$$\sum_{j=0}^{i} a_j w_j = 0 \qquad where \qquad a_{i\neq 0}.$$

Premultiplying by $w_i^T A$ and use(4) to find $a_i w_i^T A w_i = 0$. By positivedefiniteness, $w_i = 0$. Let m denotes the first value of i such that $w_i = 0$. If we define

$$x = \sum_{j=0}^{m-1} \frac{w_j^T \tilde{b}}{w_j^T A w_j} w_j, \tag{18}$$

then based on equation (15) and (16)

$$Ax - b \ \epsilon \ \{Aw_0, Aw_1, \ldots\ldots Aw_{m-1}, \bar{b}\} \subseteq \{w_0, w_1 \ldots\ldots w_{m-1}\}.$$

By construction, $w_j^T Ax = w_j^T \tilde{b}$ for $0 \leq j \leq m-1$. Hence $(Ax - \tilde{b})^T (Ax - \tilde{b}) = 0$ and $Ax = \tilde{b}$.
It can be observed that equations (2) and (3) require adding suitable multiples of all earlier w_j when computing w_i. The terms vanish when $j < i - 2$, because of the following relation:

$$\begin{aligned} w_j^T A^2 w_{i-1} &= (Aw_j)^T A w_{i-1} \\ &= (w_{j+1} + \sum_{k=0}^{j} c_{j+1,k} w_k)^T A w_{i-1} \\ &= 0 \qquad (j < i-2). \end{aligned} \tag{19}$$

Hence equation (15) simplifies to

$$w_{i+1} = Aw_i - c_{i+1,i} w_i - c_{i+1,i-1} w_{i-1} \qquad (i \geq 1), \tag{20}$$

where the coefficient $c_{i+1,i}$ and $c_{i+1,i-1}$ can be computed as follows:

$$\begin{aligned} c_{i+1,i} &= \frac{(Aw_i)^T(Aw_i)}{w_i^T Aw_i}, \\ c_{i+1,i-1} &= \frac{(Aw_{i-1})^T(Aw_i)}{w_{i-1}^T Aw_{i-1}}. \end{aligned} \tag{21}$$

6.2 The improved Lanczos algorithm:

Now we would go back the problems we aim for,namely equation (14). So we make some substitutions. If we substitute $R^T R$ as A into the equations described above, and define

$$\tilde{p_i} = Rw_i \qquad \tilde{q_i} = R^T \tilde{pi},$$

and

$$\alpha_i = \tilde{q_i}^T q_i, \qquad \beta_i = \tilde{q_i}^T w_i, \qquad \theta_i = \tilde{q_i}^T \tilde{q}_{i-1},$$

then the equation (20) can be expressed as follows:

$$w_{i+1} = \tilde{q_{i-1}} - \frac{\alpha_i}{\beta_i} w_i - \frac{\theta_i}{\alpha_{i-1}} w_{i-1} \tag{22}$$

If we define $\tau_i = w_i^T \tilde{b}$ and $\lambda_i = \tilde{q}_i^T \tilde{b}$, we have

$$\tau_{i+1} = w_{i+1}^T \tilde{b} = \lambda_i - \frac{\alpha_i}{\beta_i}\tau_i - \frac{\theta_i}{\alpha_{i-1}}\tau_{i-1}. \tag{23}$$

By substituting equation (23) into the equation (18) and denote $\phi_i = \tilde{p}_i^T \tilde{p}_i$, x_i can be updated from the following:

$$x_i = x_{i-1} + \frac{\tau_i}{\phi_i} w_i.$$

Combining all above equations with complicated mathematical derivations with algorithm reorganization the sketch of the Improved Lanczos algorithm can be depicted in Algorithm 1.

ALGORITHM 1: The Improved Lanczos Algorithm

1 $w_{-1} = w_0 = R^T b$;

2 $\tilde{p}_{-1} = \tilde{p}_0 = Rw_0, \tilde{q}_{-1} = \tilde{q}_0 = R^T \tilde{p}_0$;

3 $\alpha_0 = \tilde{q}_0^T \tilde{q}_0, \beta_0 = \tilde{q}_0^T w_0, \theta_0 = \tilde{q}_0^T \tilde{q}_{-1}$;

4 $\phi_0 = \tilde{p}_0^T \tilde{p}_0, \lambda_0 = \tilde{q}_0^T \tilde{b}, \tau_0 = w_0^T \tilde{b}$;

5 $x_0 = \frac{\tau_0}{\phi_0}$; $w_1 = q_0 - \frac{\alpha_0}{\beta_0} w_0$;

6 for i=1,2,3,.....do

7 $\tilde{p}_i = Rw_i$;

8 $\tilde{q}_i = R^T \tilde{p}_i$;

9 $\tilde{b} = R^T b$;

9 $\alpha_i = \tilde{q}_i^T \tilde{q}_i$;

10 $\beta_i = \tilde{q}_i^T w_i$;

11 $\theta_i = \tilde{q}_i^T \tilde{q}_{i-1}$;

12 $\phi_i = \tilde{p}_i^T \tilde{p}_i$;

13 $\lambda(i) = \tilde{q}_i^T \tilde{b}$;

14 $x_i = x_{i-1} + \frac{\tau_i}{\phi_i} w_i$

15 $w_{i+1} = \tilde{q}_{i-1} - \frac{\alpha_i}{\beta_i} w_i - \frac{\theta_i}{\alpha_{i-1}} w_{i-1}$;

16 $\tau_{i+1} = \lambda_i - \frac{\alpha_i}{\beta_i} \tau_i - \frac{\theta_i}{\alpha_{i-1}} \tau_{i-1}$

17 end for

Parallel Steps :
Under the assumptions, the improved Lanczos method can be efficiently parrallelized as follows:

- The inner product of a single iteration step (9),(10),(11),(12) and (13) are independent(parallel).
- The matrix vector multiplications of a single iteration step (7) and (8) are independent.
- The vector updates (14),(15) and (16) are independent.

6.3 Drawbacks of Lanczos algorithm

- The time complexity of the standarnd Lanczos algorithm is O(n^3).
- It is possible to have $w_i^T A w_i = 0$ during the iteration. In this case, the algorithm can not continue and the algorithm fails.

Assume we apply this algorithm to a field K, if $k >> n$ then we can ignore this risk. But we want to apply this algorithm to GF(2), which only has two elements and half of the vectors are A-orthogonal to themselves, so we need to modify our algorithms. So we turn our attention to the block-lanczos algorithm, which iterates on block vectors instead of single vectors.

7 Block Lanczos Algorithms

Let A be a symmetric matrix over a field K. Block Lanczos algorithms modify Lanczos algorithm to produce a sequence of subspaces $\{\omega_i\}_{i=0}^{m-1}$ of K^n which are pairwise A-orthogonal. The condition $w_i^T A w_i \neq 0$ in Lanczos algorithm is replaced by a requirement that no nonzero vector in ω_i be A-orthogonal to all of ω_i.

7.1 Montgomery's block-Lanczos algorithm

Montgomery's block-Lanczos algorithm was proposed by P.L. Montgomery's in 1995. It is an extension of the standard Lanczos algorithm over GF(2). There are some good properties over GF(2), for example, we can apply matrix to N vectors at a time (N is the length of computer world, typically equals to 32 or 64), and also do bitwise operations. It uses explicit symmetrization. In order to decrease the possibility of breakdown, first we build set of vectorspaces instead of set of vectors.

Notation:-
If ω is a subspace of K^n, then $\Theta(\omega)$ represents a vector in ω or a matrix with column vectors in ω.

Definition 7.1. *If A denotes a symmetric $n \times n$ matrix over a field K. Two vectors v,w $\in K^n$ is said to be A-orthogonal if $v^T Aw = 0$.*

Definition 7.2. *A subspace $\omega \subseteq K^n$ is said to be A-invertible if it has a basis W of column vectors such that $W^T AW$ is invertible.*

The property of being A-invertible is independent of the choice of basis, since any two bases for ω are related by an invertible transformation. If ω is A-invertible, then any u$\in K^n$ can be uniquely written as v+w where w$\in \omega$ and ω^TAv=0. Indeed, if the columns of W are a basis for ω, then w=$W(W^T AW)^{-1}W^T Au$. Here we are generalizing the subspaces as ω_i is A-invertible,

$$\omega_j^T A\omega_i = \{0\} \qquad (i \neq j),$$

$$A\omega \subseteq \omega; \omega = \omega_0 + \omega_1 + \cdots \omega_{m-1}. \tag{24}$$

Given $b \in \omega$, we can construct an $x \in \omega$ such that Ax=b. Let X=$\sum_{j=0}^{m-1} w_j$, where $w_j \in \omega_j$ is chosen so that $Aw_j - b$ is orthogonal to all of w_j. If the columns of W_j form a basis for ω_j, then

$$x = \sum_{j=0}^{m-1} W_j (W_j^T AW_j)^{-1} W_j^T b, \tag{25}$$

generalizes (18).
Fix $N > 0$. At each step i, we will have an $n \times N$ matrix V_i which is A-orthogonal to all earlier W_j. The initial V_0 is arbitrary. We select W_i using as many columns of V_i as we can, subject to the requirement that W_i be A-invertible. More precisely, we try to replace the Lanczos iterations by

$$\begin{aligned} W_i &= V_i S_i, \\ V_{i+1} &= AW_i S_i^T + V_i - \sum_{j=0}^{i} W_j C_{i+1,j} \qquad (i \geq 0), \\ \omega_i &= \langle W_i \rangle. \end{aligned} \tag{26}$$

Stop iterating if $V_i^T AV_i = 0$, say for i=m.
Here S_i is an $N \times N_i$ projection matrix chosen so that $W_i^T AW_i$ is invertible while making $N_i \leq N$ as large as possible. The matrix S_i should be zero except for exactly one per column and atmost one per row. These ensure that $S_i^T S_i = I_{N_i}$ and that $S_i^T S_i$ is a submatrix of I_N reflecting the vectors selected from V_i.

Equation (26) for V_{i+1} tries to generalize (15) and (16) while ensuring $W_j^T AV_{i+1} = \{0\}$, for $j \leq i$ if the earlier W_j exhibit the desired A-orthogonality. We use

$$C_{i+1,j} = (W_j^T AW_j)^{-1} W_j^T A(AW_i S_i^T + V_i). \tag{27}$$

Theorem 7.1. *Equations* (26) *and* (27) *imply* (24) *if* $V_m = 0$.
Furthermore,

$$W_j^T A V_i = 0 \qquad (0 \le j < i \le m). \tag{28}$$

Proof:- *The selection of* S_i *ensures* $\omega_i = \langle W_i \rangle$ *is* A*-invertible. The equation* $W_j^T A V_i = 0$ *implies* $W_j^T A W_i = 0$ *since* $W_i = V_i S_i$. *It also implies* $W_i^T A W_j = 0$ *since* A *is symmetric.*

We prove (28) *by induction on* i. *Let* $0 \le k < m$ *and assume* (28) *holds for* $0 \le j < i \le k$. *This assumption is vacuously true when* $k = 0$. *If* $0 \le j \le k$, *then*

$$W_j^T A V_{k+1} = W_j^T A(A W_k S_k^T + V_k) - \sum_{i=0}^{k} W_j^T A W_i C_{k+1,i}$$

$$= W_j^T A(A W_k S_k^T + V_k) - W_j^T A W_j C_{k+1,j} = 0$$

by induction and choice (27) *of* $C_{k+1,j}$.
A corollary to (28) *is* $\omega_j^T A \omega_i = \{0\}$ *if* $i \ne j$.
Post-multiply the defining equation (26) *for* V_{i+1} *by* S_i.

$$V_{i+1} S_i = A W_i S_i^T S_i + V_i S_i - \sum_{j=0}^{i} W_j C_{i+1,j} S_i$$

$$= A W_i + W_i - \sum_{j=0}^{i} W_j C_{i+1,j} S_i$$

This and equation (26) *give*

$$A W_i = V_{i+1} S_i - W_i + \sum_{j=0}^{i} W_j C_{i+1,j} S_i = V_{i+1} S_i + \Theta(\omega), \tag{29}$$

$$V_i = V_{i+1} - A W_i S_i^T + \sum_{j=0}^{i} W_j C_{i+1,j} = V_{i+1} - A W_i S_i^T + \Theta(\omega).$$

By hypothesis,$V_m = 0 = \Theta(\omega)$. *By backward induction and equation* (29), $A W_i = \Theta(\omega)$ *for* $0 \le i \le m-1$. *Hence* $A\omega \subseteq \omega$.
The subspaces generated ω_i *from equation* (26) *have dimension atmost* N. *This is immediate from equation* (26) *since* $\omega_i = \langle V_i S_i \rangle$ *and* V_i *is an* $n \times N$ *matrix.*

Simplifying the Block Lanczos Recurrence

Here we would like to optimize the computation of V_{i+1} in (26) using the invariant (28). We have some freedom in the choice of S_i since $\langle V_i \rangle$ may have multiple bases.
If $j < i$, then the term $W_j^T A V_i$ in (27) vanishes by (28). We attemt to simplify $W_j^T A^2 W_i$, using (26) and (28).

$$\begin{aligned} W_j^T A^2 W_i &= (S_j^T S_j) W_j^T A^2 W_i \\ &= S_j^T (A W_j S_j^T)^T A W_i \\ &= S_j^T (V_{j+1} - V_j + \Theta(\omega_0 + \omega_1 + ...\omega_j)^T A W_i \qquad (j < i) \\ &= S_j^T V_{j+1}^T A W_i - W_j^T A W_i = S_j^T V_{j+1}^T A W_i \end{aligned} \tag{30}$$

If $S_{j+1} = I_N$(so that $V_{j+1} = W_{j+1}$) and if $(j < i-1)$,then (30) vanishes since $W_{j+1}^T A W_i = 0$.

In the other cases equation (30) does not similarly satisfy. We may be unable to force $C_{i+1}^j = 0$ for $j \leq i-3$. Then the recurrence (26) will simplify to

$$V_{i+1} = AW_iS_i^T + V_i - W_iC_{i+1,i} - W_{i-1}C_{i+1,i-1} - W_{i-2}C_{i+1,i-2} \quad (i \geq 2). \tag{31}$$

Equation (31) remains valid for i=0 and i=1 if we define $V_j = 0$ and $W_j = 0$ for $j < 0$.
To achieve equation (31), we require that the equation (30) vanish whenever $j \leq i-3$. That is, we require V_{j+1} to be A-orthogonal to W_{j+3} through W_m. We achieve this by requiring that all vectors in V_{j+1} be used either in W_{j+1} or in W_{j+2}. More precisely we require

$$\langle V_{j+1} \rangle \subseteq \omega_0 + \omega_1 + ...\omega_{j+2} \qquad (j \geq -1). \tag{32}$$

Assuming the equation (32) we try to simplify the matrix equation (31). Denote

$$W_i^{inv} = S_i(W_i^T A W_i)^{-1} S_i^T = S_i(S_i^T V_i^T A V_i S_i)^{-1} S_i^T \tag{33}$$

Each W_i^{inv} is a symmetric $N \times N$ matrix. Eliminating all references to $W_i = V_iS_i$, we get

$$\begin{aligned} V_{i+1} &= AV_iS_iS_i^T + V_i - V_iS_iC_{i+1,i} - V_{i-1}S_{i-1}C_{i+1,i-1} - V_{i-2}S_{i-2}C_{i+1,i-2} \\ &= AV_iS_iS_i^T + V_i - V_iW_i^{inv}V_i^T A(AV_iS_iS_i^T + V_i) \\ &- V_{i-1}W_{i-1}^{inv}V_{i-1}^T A^2 V_iS_iS_i^T - V_{i-2}W_{i-2}^{inv}V_{i-2}^T A^2 V_iS_iS_i^T. \end{aligned} \tag{34}$$

This equation appears to require four inner products: $V_i^T AV_i$, $V_i^T A^2 V_i$, $V_{i-1}^T A^2 V_i$ and $V_{i-2}^T A^2 V_i$. We can express the latter two inner products in terms of first two using equations (27), (28), (29), and (31).

$$\begin{aligned} S_{i-1}^T V_{i-1}^T A^2 V_i &= (AW_{i-1})^T AV_i = (V_iS_{i-1} + \Theta(\omega_0 + \omega_1 + ...\omega_{i-1}))^T AV_i = S_{i-1}^T V_i^T AV_i \\ S_{i-2}^T V_{i-2}^T A^2 V_i &= (AW_{i-2})^T AV_i = (V_{i-1}S_{i-2} + \Theta(\omega_0 + \omega_1 + ...\omega_{i-2}))^T AV_i = S_{i-2}^T V_{i-1}^T AV_i \\ &= S_{i-2}^T V_{i-1}^T A(AW_{i-1}S_{i-1}^T + V_{i-1} - W_{i-1}C_{i,i-1} + \Theta(\omega_{i-2} + \omega_{i-3})) \\ &= S_{i-2}^T V_{i-1}^T A(I_N - V_{i-1}W_{i-1}^{inv}V_{i-1}^T A)(AW_{i-1}S_{i-1}^T + V_{i-1}) \\ &= S_{i-2}^T (I_N - V_{i-1}^T AV_{i-1}W_{i-1}^{inv})(V_{i-1}^T A^2 V_{i-1}S_{i-1}S_{i-1}^T + V_{i-1}^T AV_{i-1}) \end{aligned}$$

Hence equation (34) simplifies to

$$V_{i+1} = AV_iS_iS_i^T + V_iD_{i+1} + V_{i-1}E_{i+1} + V_{i-1}F_{i+1}, \tag{35}$$

for $i \geq 0$, where

$$\begin{aligned} D_{i+1} &= I_N - W_i^{inv}(V_i^T A^2 V_iS_iS_i^T + V_i^T AV_i); \\ E_{i+1} &= -W_{i-1}^{inv}V_i^T AV_iS_iS_i^T; \\ F_{i+1} &= -W_{i-2}^{inv}(I_N - V_{i-1}^T AV_{i-1}W_{i-1}^{inv})(V_{i-1}^T A^2 V_{i-1}S_{i-1}S_{i-1}^T + V_{i-1}^T AV_{i-1})S_iS_i^T \end{aligned} \tag{36}$$

We define W_j^{inv} and V_j to be zero and S_j to be I_N for $j < 0$. For mathematical purpose sketch of the Montgomery's Block-Lanczos method is described in Algorithm 2.

Selecting S_i and W_i
S_i and $W_i = V_iS_i$ should be selected in the way such that the following conditions should satisfy.

- $W_i^T AW_i$ must be invertible.

- rank W_i must be as large as possible.
- Any column of V_{i-1} which was not used in W_{i-1} must be used now.

Algorithm(3) describes the procedure for selecting S_i and W_i.

ALGORITHM 2: Montgomery's block-Lanczos algorithm

Inputs:A,b and V_0, where A is symmetric,invertible and $V_0$is arbitrary.
Cmt.S_i is an $N \times N_i$ projection matrix ($N_i < N$) to make $W_i^T A W_i$ invertible.
Cmt. S_i has exactly 1 per column and atmost 1 per row. Each W_i^{inv} is a symmetric $N \times N$ matrix. we define W_i^{inv} and $V_j$to be 0 and S_j to be I_N for $j < 0$.
Cmt. How to select $S_i S_i^T$ and W_i^{inv} can be found in Algorithm 3.
Outputs: The solution vector x.
While($i = 0$ to m) and($V_i^T A V_i \neq 0$) do
$D_{i+1} = I_N - W_i^{inv}(V_i^T A^2 V_i S_i S_i^T + V_i^T A V_i)$;
$E_{i+1} = -W_{i-1}^{inv} V_i^T A V_i S_i S_i^T$;
$F_{i+1} = -W_{i-2}^{inv}(I_N - V_{i-1}^T A V_{i-1} W_{i-1}^{inv})(V_{i-1}^T A^2 V_{i-1} S_{i-1} S_{i-1}^T + V_{i-1}^T A V_{i-1}) S_i S_i^T$
$V_{i+1} = A V_i S_i S_i^T + V_i D_{i+1} + V_{i-1} E_{i+1} + V_{i-1} F_{i+1}$
end while
if $i == m$ then
Calculate x.
$x = \sum_{j=0}^{j=m-1} V_j W_j^{inv} V_j^T$;
end if
if $V_i^T A V_i \neq 0$ then
Program failure exit.
end if

ALGORITHM 3: Select S_i and W_i^{inv}

Inputs: $T = V_i^T A V_i$ and S_{i-1}, (where A and hence T)is symmetric.
Outputs: Set S for diagonal of $S_i S_i^T$, and $W_i^{inv} = S_i (S_i^T T S_i)^{-1} S_i^T$.
Construct an $N \times 2N$ block matrix M , with T on the left and I_N on the right.
Cmt. Algorithm performs row operations on M. It may zero an entire row.
Number columns of T as c1,c2,.....cN, with columns in S_{i-1} coming last.
Initialize S=0.
Cmt. S has the columns selected from V_i for W_i.

for j=1 to N do
for $k = j; k <= N$ and $M[cj, cj] = 0; k++$ do
if $M[ck, cj] \neq 0$ then
Exchange rows cj and ck of M.
end if
end for
if $M[cj, cj] \neq 0$ then
$S = S \cup \{cj\}$

Divide row cj of M by $M[cj, cj]$.
Add multiples of row cj to other rows of M, to zero rest of column cj.
else
for $k = j; k <= N$ and $M[cj, cj + N] = 0; k++$ do
if $M[ck, cj + N] \neq 0$ then
Exchange rows cj and ck of M.
end if
end for
assert$(M[cj, cj + N] \neq 0)$
Add multiples of row cj to other rows, to zero rest of column cj+N.
end if
end for
copy right half of M into W_i^{inv}.

7.2 Improved Montgomery's block-Lanczos algorithm

It is the extension of Montgomery's block-Lanczos algorithm. Original Montgomery's algorithm uses an explicit method to symmetrize the input with the complexity of O(n^3). The improved Montgomery's algorithm uses an implicit symmetrization process to avoid the explicit symmetrization process with the complexity of O(n^3). It also proposes some initial strategies to find the solutions.

ALGORITHM 4: Improved Montgomery's block-Lanczos algorithm

Inputs: A,b and V_0,where A is not necessarily symmetric and $V_0$is arbitrary.
Cmt.S_i is an $N \times N_i$ projection matrix ($N_i < N$) to make $W_i^T A W_i$invertible.
Cmt. S_i has exactly 1 per row.Each W_i^{inv} is a symmetric $N \times N$ matrix .we define W_i^{inv} and $V_j$to be 0 and S_j to be$I_N$for $j < 0$.
Cmt. How to select $S_i S_i^T$ and W_i^{inv} can be found in previous Algorithm 3.
Outputs: The solution vector x.
While($i = 0$ to m) and$(V_i^T A^T A V_i \neq 0)$ do
$D_{i+1} = I_N - W_i^{inv}(V_i^T A^T(A(A^T(AV_i S_i S_i^T))) + V_i^T A^T A(V_i))$;
$E_{i+1} = -W_{i-1}^{inv} V_i^T A^T(AV_i S_i S_i^T)$;
$F_{i+1} = -W_{i-2}^{inv}(I_N - V_{i-1}^T A^T(AV_{i-1} W_{i-1}^{inv}))$ $(V_{i-1}^T A^T(A(A^T(AV_{i-1} S_{i-1} S_{i-1}^T))) + V_{i-1}^T A^T(AV_{i-1}) S_i S_i^T)$
$V_{i+1} = A^T(AV_i S_i S_i^T) + V_i D_{i+1} + V_{i-1} E_{i+1} + V_{i-1} F_{i+1}$ end while
if $i == m$ then
Calculate x.

$$x = \sum_{j=0}^{j=m-1} V_j W_j^{inv} V_j^T;$$

end if
if $V_i^T A V_i \neq 0$ then
Program failure exit.
end if

7.3 Drawbacks of Montgomery's block-Lanczos algorithm

No matter which symmetrizing methods we use in Montgomery's Block-Lanczos algorithm, implicit or explicit, the symmetrization process significantly reduces the solutions, since over GF(2) the rank of the

product A^TA in general much less than that of A. This study implemented the following two algorithms:

Standard Lanczos algorithm:
This is not efficient algorithm, because this method does not ensure about the solution in all the cases. Time complexity of this algorithm is O(n^3)

Montgomery's block-Lanczos algorithm:
This is most reliable and efficient algorithm. Time complexity of this algorithm is $O(n^3)$. This algorithm is for symmetric matric. If the matrix is not symmetric, we first make it symmetric but for that matrix should be invertible. If matrix is neither symmetric nor invertible, then we can't find solution by this method.

8 Conclusions and Future Works

This research investigated three algorithms. Firstly, it have studied standard Lanczos algorithm that doesn't ensures about the solution. Then, It have studied Montgomery's block-Lanczos algorithm that finds the solution in most of the cases but still fails in some cases. After that, it studied improved Montgomery's block-Lanczos algorithm that works in some more cases in comparison to Montgomery's block-Lanczos algorithm.
So these are the successful attempts and improved Montgomery's block-Lanczos algorithm is the most efficient and reliable algorithm among all the algorithms that studied in this work. Future Works can be seen as

- Improved Montgomery's algorithm can be analyzed in more details alongwith its implementation.
- researchers can explore and develop some new algorithm which doesn't have the drawbacks like previous algorithms and can work for all the cases.
- Solving linear system of equations over an arbitrary field can be challenging and have the area to explore.

Declarations

Ethics approval and consent to participate

Not applicable.

Conflict of interest

The author claims that there are no conflicts of interest.

Data availability statement

No datasets have been generated or analyzed during the current investigation.

References

[1] D. Coppersmith, A.M. Odlyzko, and Schroeppel : "Discrete logarithms in GF(p)" Algorithmica 1: 1-1.5(1986).

[2] B.A. LaMacchia and A.M. Odlyzko : "Computation of discrete logarithms in prime fields", Designs, Codes and Cryptography, 1, 47-62 (1991).

[3] K. McCurley : "The discrete logarithm problem", Cryptology and Computational Number Theory, Proceedings of Symposia in Applied Mathematics, American Mathematical Society, 1990.

[4] A. M. Odlyzko : "Discrete logarithms in finite fields and their cryptographic significance" to appear, Proceedings of Eurocrypt' 84, Springer Lecture Notes in Computer Science.

[5] W. Diffie and M. E. Hellman, "New directions in cryptography",IEEE Trans. Inform. Theory, IT-22, 644-654(1976).

[6] John Brillhart : "Note on representing a prime as a sum of two squares", Mathematics of computation, vol 26, 1972.

[7] Dean Phillip Reiff : "Discrete logarithm in finite field",thesis, University of Colorado at Denver, 1996.

[8] LaMaecbla, B.A., and Odlyzko, A.M. (forthcoming). "Solving large sparse linear systems over finite fields", Advances in Cryptology: Proceedings of Crypto '90, (A. Menezes, S. Vanstone, eds.), Lecture Notes in Computer Science, New York: Springer-Vedag.

[9] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", IEEE Trans. Info. Theory 31 (1985), 469-472.

[10] P. Van Oorschot, "A comparison of practical public key cryptosystems based on integer factorization and discrete logarithms", pp. 289-322 in Contemporary Cryptology: The Science of Integrity, G.J. Simmons. ed., IEEE Press, New York, 1992.

[11] D.Coppersmith, "Fast evaluation of logarithms in fields of characteristic two", IEEE Transactions on Information Theory 30 (1984), 587-594.