Eclipse統合環境におけるPARI Libraryを使ったCプログラミングとその応用

永田　清

大東文化大学経営学部

nagata@ic.daito.ac.jp

キーワード　PARI/GP，データの完全性，誤り訂正符号，Self-Dual Code

1　はじめに

パーソナルコンピュータが普及し，スマートファンやタブレット端末をほどんどの国民が所有するようになった現在，コンピュータ自体というよりもその処理能力が遍在するUbiquitous Computingが現実となった．また，クラウドコンピューティングを通じたIoT(Internet of Things)の時代には，ハードウェアの存在自体が認識されなくなるのかもしれない．しかし，どこかにはプログラムがあって常に様々な処理を行っているはずであり，AI(artificial intelligence)が再帰的に増殖を繰り返すとしても，それを作成・発展させる必要はあるだろう．

本稿の目的は，発展を遂げてきた数式処理ソフトウェアPARIのライブラリを，既存のプログラミング言語であるC言語に組み込んでプログラム作成を行う手順と，その応用例として筆者らによる誤り訂正符号の生成アルゴリズムの実装例を示すことである．第1，2節では，コンピュータとプログラミング言語の歴史を概観することでさまざまなプログラミング言語の特徴などを把握する．第3節ではプログラミング統合開発環境の一つであるEclipseとPARI/GPについて，その概略と設定方法などを解説する．第4節では，C言語でPARI Libraryを使う場合の留意点や関数仕様を見る．第5節は，誤り訂正符号の生成アルゴリズムであり，第6節で実際のプログラムの一部を紹介する．第7節は結論と今後の課題である．

2　コンピュータとプログラミング言語の歴史

まず，コンピュータの歴史と，プログラミング言語発展の流れなどと観ていくこととする．

2.1　自動計算器としてのコンピュータ

現代のコンピュータの先駆けとなった電子計算は1940年代に開発され，当初はその多くが特定の目的で使用された．英国ではTommy Flowersの設計によって1944年にColossus(Mark II)が完成し，対戦国ドイツの暗号機Lorenz SZ40/42による暗号文を解読するために使われた．米国においてJohn William MauchlyとJohn Adam Presper Eckert Jr.が考案と設計を行い，1946年に完成したENIAC(Electronic Numerical Integrator and Computer)は，陸軍弾道研究所の支援を受け砲撃射表と呼ばれる表の作成を目的として作成された．開発メンバの一人であり，ENIACのプログラムを担当したAdele Goldstineの夫でもあるHerman Goldstineによれば，ENIACはマンハッタン計画で有名なロスアラモス研究所の問題を計算するために使われた([8])．同じく米国では海軍が「Project Whirlwind」として資金援助し，MIT (Massachusetts Institute of Technology)研究所のJay Forrester達が開発したフライトシミュレーター制御用のコンピュータWhirlwindが1951年に動作しており，リアルタイムで動く世界最初のコンピュータといわれている(http://museum.mit.edu/150/21)．Whirlwindは「つむじ風」という意味で，空軍が運用していた旧ソ連の原爆搭載爆撃機を発見，追跡，要撃するために自動化されたコンピュータシステムであるSAGE(Semi-Automatic Ground Environment)に使われたとのことである．

MauchlyとEckertはプログラム内蔵能力のほとんどないENIACに対し，水銀遅延線(delay line memory)によってデータとプログラムを格納するような電子計算機の設計を試み，1951年にEDVAC(Electronic Discrete Variable Automatic Computer)を稼働させる．彼らはマンハッタン計画にも参加していたJohn von Neumannを顧問として迎え，彼がまとめた文書“First Draft of a Report on the EDVAC”がケンブリッジ大学のMaurice Vincent Wilkes達に影響を与え，初期のプログラム内蔵方式による電子計算機となるEDSAC(Electronic Delay Storage Automatic Calculator)を開発した．EDSACでは，Wilkesの書いた1から100までの二乗と，200まで

の素数を求めるプログラムが 1949 年に動いている ([20]).

英国の食品会社 J. Lyons and Co. は，業務上の問題解決の可能性を電子計算機に見出し，EDSAC をベースにしたコンピュータ LEO(Lyons Electronic Office) を開発し，1951 年にはベーカリーの経営を評価する Barkery Valuations をコンピュータ化している．LEO のプログラミング言語 CLEO(Clear Language for Expressing Orders) は，COBOL のような言語といわれている．因みに，1954 年に設立された LEO Computers Ltd. は，いくつかの経緯を辿り 1990 年には Fujitsu に買収された．

ケンブリッジ大学 Computer Lab の資料 ([20]) には，EDSAC が “the world’s first stored-program computer to operate a regular computing service” とあるが，米国マンチェスター大学でやはり von Neumann から影響を受けた Tom Kilburn, Freddie Williams, Geoff Tootill 達によって開発された Manchester Small-Scale Experimental Machine(SSEM, 通称 Manchester Baby) が 1948 年 6 月に稼働していた．SSEM のハードウェアは減算と正負反転だけを実装していたが，そこにプログラムを内蔵して $2^{18} = 262,144$ を割り切る自分自身でない最大の整数，つまり $2^{17} = 131,072$ を求める計算を行った ([13])．その後 SSEM は Manchester Mark I などを経て，世界最初の商用汎用コンピュータといわれている Ferranti Mark I に引き継がれることとなった．Manchester Baby に関しては，User Guide が 2001 年に Warwick 大の David Sharp によって書かれており，その概要を知ることができる (`https\://www.davidsharp.com/baby/babyuserguide.pdf`)．

コンピュータ (電子計算機) の定義要素はさまざまで，電気を動力とする，数値表現が 2 進法か 10 進法か，プログラムを内蔵できる (いわゆるノイマン型コンピュータ) かどうか，Alan Mathison Turing による万能 Turing Machine と同じ計算能力を持つ (Turing-Complete) かどうか ([23])，汎用性があるかどうか，等が挙げられている．

上記のもは全て電動であるが，自動で複雑な計算を行えるものならば 1822 年に Charles Babbage の設計による階差機関があり，その後詩人として有名な George Gordon Byron の娘でもある Augusta Ada King(Ada Lovelace) が解析機関においてベルヌーイ数を求めるプログラムを書き，世界初のプログラマであるといわれている ([6])．

1950 年代にその多くが登場した電動による計算機以外にも，アイオワ州立大学の John V. Atanasoff と Clifford E. Berry(https://jva.cs.iastate.edu/) による ABC (Atanasoff-Berry Computer) が 1939 年に稼働し，ドイツの Konrad Zuse による Z3 は 1941 年に稼働しているが，現在のコンピュータの原型と呼ばれるプログラム内蔵式 (ノイマン型) の登場には，どのようなハードウェアでそのプログラムを保持するのかといった問題の解決を待たなければならなかった．

2.2 コンピュータプログラミングと言語

初期のコンピュータはそれぞれの目的に特化して設計・作成され，各コンピュータに対しさまざまな方法でプログラムがなさており，Ada Lovelace が最初のプログラマだとしても彼女が現在のようなプログラム言語を駆使してプログラムを行ったわけではなく，一定のアルゴリズムを実行するために計算機の仕組みを変更していたものと考えられる．Colossus や ENIAC においても，プログラムはプラグ盤やスイッチの操作によって配線を変更することであった．

一方プログラム言語自体は Hilbert の形式主義に触発されたといわれている Konrad Zuse が，論理回路を表すための記法と構造図式を作り，1943 年から 1945 年頃には “高級” プログラミング言語の先駆けとも言える “Plankalkül” を開発し ([1], [7])，1948 年の論文 “概略的に組み合わされたタスクを定式化する手段としての一般 Plankalkül について” を公表した ([24])．

1940 年代の機械語やアッセンブリ言語は，判別や改変のやり難さ，表現の煩雑さ，個々のコンピュータ特性に偏っている，などさまざまな問題点を持っていたが，1950 年代以降はいわゆる高級プログラミング言語と呼ばれるものが多く開発されてきた．

一般に，世界初の高級プログラミング言語と呼ばれている FORTRAN(FORmula TRANslator) は，John Backus によって IBM のメインフレームコンピュータ IBM 704 用に開発され，主に科学・工学分野の計算に貢献し，さまざまな変更を経て現在 Fortran 2018 となっている．

CLEO の流れを汲む COBOL(Common Business Oriented Language) は，CODASYL(Conference on Data Systems Languages) によって 1959 年に開発された共通事務処理用言語であり，自然言語 (英語) による表現が使われている．

IPA が日本国内の IT 企業に対し行った調査によると，2016 年から 2017 年におけるソフトウェア開発 1,237 件に用いられた言語で COBOL は Java の 42.7%に次いで 2 位 (13.3%) となっている．因みに，3 位以降は C#(8.1%)，C 言語 (7.3%)，Visual Basic.NET(6.8%), C++(5.5%) である[1]．2020 年度の基本情報技術者試験から COBOL が

[1]「ソフトウェア開発データ白書 2018-2019」, p.50, `https://www.ipa.go.jp/files/000069381.pdf`

なくなり，過去の言語といった印象もあるが，COBOL で教育を受けたプログラマがまだ多くいることや，使われているシステム自体が COBOL でプログラムされていることなどが影響していると考えられる．実際，それらの 200 件近いソフトウェア開発においても，メインフレームにおけるバッチ処理 (約 60 件) だけでなく，クライアントサーバシステムにおけるオンライントランザクション処理 (約 50 件) にも多く使われていた．

1975 年以降に Bill Gates(William Henry Gates) と Paul Gardner Allen によって，当時販売され出した PC (Personal Computer) に組み込まれたことで有名な BASIC(Beginner's All purpose Symbolic Instruction Code) は，1964 年に米国 Dartmouth College の John George Kemeny と Thomas Eugene Kurtz によって開発された．これら以外にも，1950 年代に Kenneth Eugene Iverson による，配列処理などに特徴を持つ言語 APL(A Programming Language)([11]) や，FORTRAN，COBOL，Algol などの特徴を備えた言語として IBM などにより 1964 年に開発された PL/I(Programming Language One) といった，いわゆる手続き型言語の多くが現れている．

1950 代後半には，構造化プログラミングといった概念が提唱され，主に"go to" 文とその行き先を示すラベルなどによって処理を制御することでプログラムが複雑になるといった，いわゆる "Spagetti Code" への批判が起こった．

Edsger Wybe Dijkstra は，1969 年に "Structured programming" というタイトルの論文 ([4]) において "programs as neckless strung from pearls" といって表現で，構造化プログラミンを譬えている．Dijkstra は構造化プログラミングを一言で表した定義を与えていないようだが，Ole-Johan Dahl，Charles Antony Richard Hoare と共に 1972 年著わした "Structured Programming"([2]) では，1 章において Dijkstra が全体像を，第 2 章で Hoare がデータ構造について，第 3 章では Dahl が構造の継承などに関して記述しており，構造化プログラミングの普及に大きな影響を与えた．

Dijkstra は，1975 年当時に使われていた上記のいくつかのプログラミング言語に対し痛烈な文章を残している ([5])．例えば，FORTRAN を "the infantile disorder"，PL/I を "the fatal disease" と呼び，COBOL は "cripples the mind" であり "be regarded as a criminal offence" とまで述べている．また，BASIC によって教育受けた学生は "as potential programmers they are mentally mutilated beyond hope of regeneration" となり，APL は "the language of the future for the programming techniques of the past" であって "a mistake" との主張である．

実際の構造化プログラミングは,，順次構造 (begin-end)，判別 (選択) 構造 (if-then-else)，繰り返し構造 (for, while-do) を使って行なわれ，結果として "go to" のない (または少ない) プログラムとなるが，Donald Ervin Knuth が指摘しているように Dijkstra の論文 ([4]) には "go to" 文の記述はなく，彼の率直な関心事は "For what program structures can we give correctness proofs without undue labor, even if the programs get larger?" であると述べている [14]．Richard C. Linger，Harlan D. Mills，および Bernard I. Witt による 1979 年の著書 "Structured Programming: Theory and Practice" では，構造化プログラムを以下のように定義している ([16]，p. 118)．

定義 1 *A structured program is a compound program constructed from a fixed basis set of prime programs.*

ここで，"fixed basis set" となるのは，次の定理 ([16]pp.118-119) に示され順次構造，判別 (選択) 構造，繰り返し構造である．

定理 2 (Structure Theorem) *Any proper program is function equivalent to a structured program with basis set {sequence, ifthenelse, whiledo}, using functions and predicates of the original program and assignment and tests on one additional counter.*

構造化プログラミングを念頭において開発されたプログラミング言語の草分け的存在は，上記の 3 人以外にも人工知能や LISP(LIst Processing language) 開発でも有名な John McCarthy，Pascal や Modula-2 開発を行った Niklaus Wirth など多くの優秀な研究者が関わった ALGOL(ALGOrithmic Language) である．ALGOL にはいくつかのバージョンがあり，ALGOL68 は Pascal，C，および Ada などに強い影響を与えたと言われ，またその後 Simula などのオブジェクト指向プログラミング (OOP ; Object Oriented Programming Language) 言語の基礎となっている．

OOP 言語は，1972 年に Smalltalk を開発した Alan Kay が初めて使った言葉だと言われており，データ (フィールド) と処理 (メソッド) を併せ持つ構造設計図であるクラスとその実態となるインスタンス (オブジェクト) を扱う．Kristen Nygaad と共に Simula を開発した Ole-Johan Dahl ([3]) によると，OOP の一般的な定義は存在しないようであるが，C++，Java，C#などのコンパイラ言語が代表格とされるクラスベースの OOP 言語ではカプセル化，抽象化，継承，ポリモーフィズムが重要な要素と

されている．また，1994 年に ver 1.0 がリリースされた Python，1996 年に公表された JavaScript，Ruby などはインスタンスベース (プロトタイプベース) の OOP 言語と呼ばれている．

3　C 言語と統合開発環境

前節でみたように，ハードウェアとしてコンピュータも，コンピュータを動かすソフトウェア開発のためのプログラミング言語も，さまざまな目的，用途，思想などのもとで発展してきた．本論文の目的は，数式処理を汎用プログラミング言語によって行うことであり，またそのための環境を構築しておくことでもある．数式処理のためのパッケージについては後に述べるが，ここでは C 言語とプログラミングをスムーズに行うための統合開発環境の一つである Eclipse について観ておこう．

C 言語の系譜は，1966 年にケンブリッジ大学の Martin Richards によって設計された手続き型かつ命令型の構造化プログラミング言語 BCPL (Basic Combined Programming Language) に遡る．1969 年頃には，AT&T ベル研究所の Ken Thompson による B 言語が登場した．その開発にもかかわった Dennis MacAlistair Ritchie が主体となり，AT&T ベル研究所で UNIX OS 実装のため，1970 年代に開発された言語が C 言語である ([21])．判別 (選択) 構造として “if-else if-else”，“switch”，繰り返し構造 “do/while”，“for”，“while” などの制御構文を持っており，その点では高級言語といえるが，ポインタ変数などによってメモリへ直接アクセスできることから低水準言語の特徴も持っている．Brian W. Kernighan と Dennis M. Ritchie による “The C Programming Language” の初版が 1978 年に出版され，1988 年の第 2 版では ANSI での標準化が反映されている ([12])．これらは石田晴久氏によって日本語訳され，C 言語の教科書的な役割を果たした．

C 言語は，歴史的には古いが現在も使われているプログラミング言語であり，多くの標準ライブラリが提供され，それらをヘッダファイルとして取り込む (#include する) ことで，様々な機能を簡単に使うことができる．

このようなコンパイル型処理系では，プログラミング言語で記述されたソースファイルを，プリプロセス，コンパイル，アセンブル，リンクなどのいくつかの段階で処理して実行可能はファイルを作成するが，プログラム作成時にそれらを特に意識する必要はない．また，プログラム作成には文字を入力するためのエディタソフトウェアを使い，ソースファイルの保存，編集，削除などの処理も必要になる．高機能エディタのいくつかはオープンソースとして無料で提供されており，それに C 言語のコンパイラを結びつけて言語の学習やプログラム作成が行えるが，現在では統合開発環境 (IDE: Integrated Development Environment) として無料のアプリケーションソフトウェアがいくつか提供されている．無料ではあるが，一般的なエディタだけでなく，入力ミスや構文エラー表示，自動補完・修正，実行時の監視など様々な機能が組み込まれており，ここではその一つである Eclipse を使うこととする．

Eclipse は，1990 年代初頭に IBM によって開発され，プログラミング言語に対応した統合開発環境 VisualAge に由来しているが，2000 年以降は非営利組織である Eclipse Foudation によって無償のオープンソースソフトウェアとして Eclipse Software Development Kit(SKD) がリリースされている．VisualAge は Smalltalk によって実装されたが，Eclipse は Java 言語によっているので，Java 言語によるプログラム作成に適しているが，最近では C/C++，PHP，Python など様々な言語をプラグインによって扱える統合開発環境となっている．基本は英語バージョンであるが，Eclipse Foudation における抱卵的プロジェクトである Babel (`https://www.eclipse.org/babel/`) では，多言語対応を行っている．また，Mergedoc Project (`https://mergedoc.osdn.jp/`) では，さまざまなバージョンの Eclipse があり C/C++，PHP，Python に対応し，Windows 版や Mac 版は All In One(AIO) パッケージとしてダウンロードして使える．ここでは，後者の Windows 版で C/C++パッケージを使うこととし，以下にインストールと C 言語によるプログラム作成手順を簡単に解説する．

3.1　Eclipse のインストールと C 言語によるプログラム作成・実行

Windows に Eclipse の AIO パッケージを解凍するには “7-Zip” を使うので，まずページにあるリンクによって “7-Zip” をダウンロードしておく．次に，Windows64 ビット Full Edition C/C++の “Download”(図 1) をクリックし，表示されるリンクから Zip ファイルをダウンロードし，そのファイルを右クリックして 7-Zip で適当な場所 (MyDocument フォルダなど）に解凍すればよい．

Eclipse の実行ファイル (eclipse.exe) は，“¥Documents ¥pleiades-2020-12-cpp-win-64bit-jre_20201101¥pleiades ¥eclipse” フォルダ (図 2) にあるので，特別な設定などの必要がなくダブルクリックなどによって起動する．最初

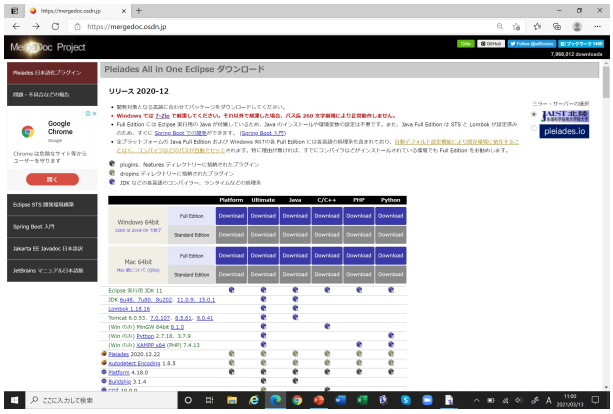

図 1: Mergedoc Project のトップページ

は Work space(ファイルを保存するフォルダ) を聞いてくるので，適当な場所を指定する．

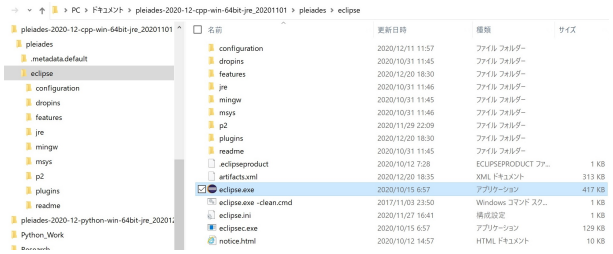

図 2: Eclipse の実行ファイル

Eclipse で C 言語によるプログラムを作成するには，チュートリアルウィンドウを閉じ，[ファイル]-[新規] から “C/C++プロジェクト” を作る (図 3)．

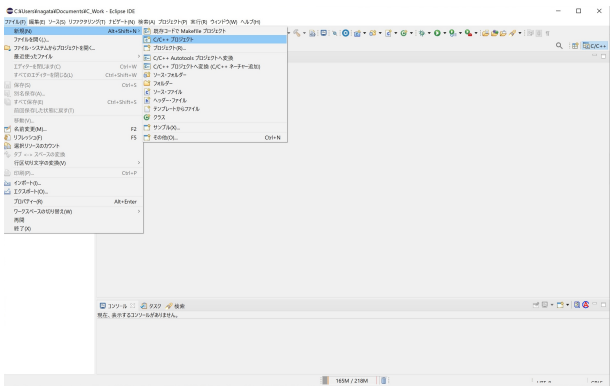

図 3: Eclipse で C/C++プロジェクト作成

次に “C Managed Build” ウィンドウを選び，プロジェクト名を入力すると,「プロジェクト・エクスプローラー」にそのプロジェクトが表示されるので，[ファイル] メニューまたは右クリックによって表示される [新規] から “ソース・ファイル” を選び (図 4)，適当な名前 (.c) を付けて作成するとエディター画面が現れるので C 言語によるプログラムを書いていくことになる．プログラムが書けたら，[保存]-[ビルド]-[実行] の手順で進み，結果は結果は「コンソール」(図 4 では右下) に表示され，実行ファイル自体はプロジェクトフォルダ内の “Debug” フォルダに入る．

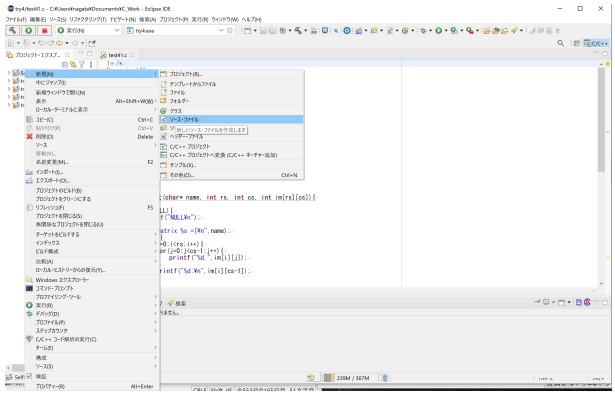

図 4: Eclipse で C 言語のソースファイル作成

3.2 数式処理ソフトウェア PARI/GP

コンピュータを使って数式を処理し，数値計算やグラフィック表示などを行うソフトウェアを数式処理システム (Formula Manipulation System) と呼ぶ．1963 年に Stanford 大学で理論物理学のポスドク研究員だった Anthony C. Hearn が，John McCarthy の助言によってプログラミング言語 LISP を使って開発を始め ([10])，数年後に作成された REDUCE が “portable general-purpose computer algebra system” として有名であり，現在も開発が行われている (`http://www.reduce-algebra.com/index.php`)．

商用版としては Mathematica，Maple，Magma，MATLAB などのかなり高価なものもあるが，RARI/GP，Risa/Asir (`http://jp.asir.org/`) のようなオープンソースのフリーソフトウェアもある．最近では，上記の商用版ソフトウェアの代替となるようなフリーソフトウェアを提供しようといった Sage プロジェクトのようなものも立ち上がっている (`https://www.sagemath.org/`)．

本研究では，RARI Library を使うので，ここで RARI/GP について簡単に観ていくこととする．RARI/GP は，1979 年に Borddeaux 大学の Henri Cohen と François Dress による代数計算インタープリタ ISABELLE に由来し，1986 年に Christian Batut，Henri Cohen，Michel Oliver によって PARI アセンブリカーネルと PARI Library の構想が始まった．1995 年に Karm Belabas が維持者となり，フランスの研究者を中心に開発･改良が行われ，

フリーソフトウェアとして GPL(GNU General Public License) によりライセンス化されている．2021 年 1 月 25 日の時点では最新版 PARI-2.13.1 のソースファイル pari-2.13.1.tar.gz や Windows 版，MacOS 版，Android 版のバイナリーインストールファイルがダウンロードできる (図 5，`http://pari.math.u-bordeaux.fr/download.html`)．

図 5: PARI/GP Download ページ

PARI/GP が扱える桁数は 10 億桁程度までで，整数論に関する多くの関数を持ち，また一般の行列，多項式，冪級数や特殊関数なども扱える．“PARI” は当初 Pascal によって開発されたため “Pascal ARIthmetic” の略であるとか，フランス語で賭けを意味する “pari” に関係していると言われている．現在は C 言語によって開発され，C 言語のライブラリが提供されているが，スクリプト言語 “GP” によってインタラクティブシェル (gp) 上でさまざまな計算やプログラムを作成することができる．

我々は誤り訂正符号の生成プログラムを “GP” によって作成したが ([25])，C 言語に近い構文であっても条件判断や繰り返しの入れ子構造が複雑になってしまうといった問題があり，PARI Library をインクルードして Eclipse において C 言語でプログラム作成を行うこことした．プログラム自体は第 6 節に示すが，ここでは Eclipse における PARI Library の利用設定について説明する．

まず，PARI のライブラリ (.dll) とヘッダーファイルの場所を確認しておく．図 6 では，ライブラリ “libpari.dll” が “C:¥Program Files (x86)¥Pari64-2-13-1” で，ヘッダーファイルホルダーが “C:¥Program Files (x86)¥Pari64-2-13-1¥include¥pari” となっている．

次に，Eciplse で作成したプロジェクトを右クリックして，一番下に表示される “プロパティー” ウィンドウを開き，“C/C++ビルド” の [設定] から表示される “GCC C Compiler” の [インクルード] を選んで，右側に表示され

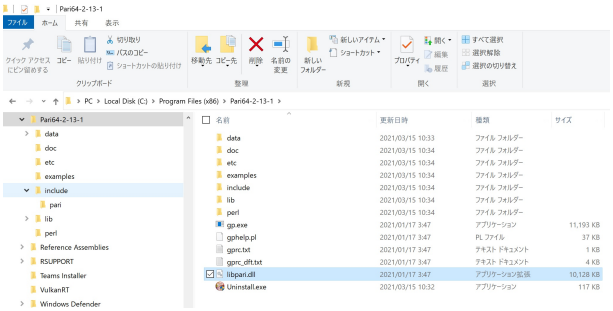

図 6: PARI/GP Download ページ

るウィンドウボックスの “インクルード・パス” にヘッダーファイルホルダーを追加しておく (図 7)．

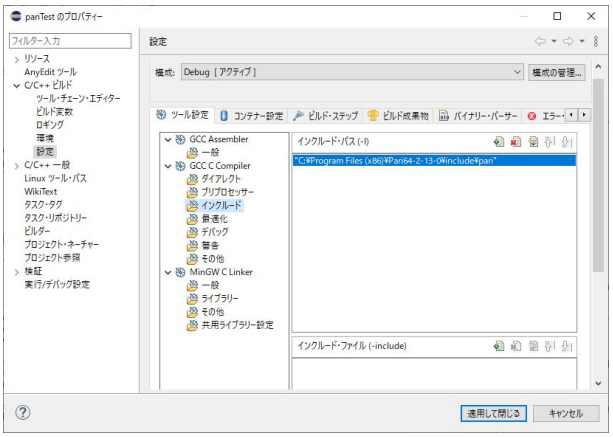

図 7: PARI/GP Download ページ

ライブラリ設定は，“MinGW C Linker” の [ライブラリー] で表示されるウィンドウボックスで行う．ライブラリファイル名 “libpari” と検索パス “C:¥Program Files (x86)¥Pari64-2-13-1” それぞれを追加しておく (図 8)．

4 PARI Library の使用とプログラム作成

Eclipse 上で C 言語による PARI Library を使う準備ができたので，ここでは一般的な注意と実際に使う Library 仕様について述べる．ただし，この節で特に断らずに括弧内に示すページ番号は，“User’s Guide to the PARI library (version 2.11.1)” (`http://pari/math.u-bordeax.fr/`) のものとする．

4.1 一般的な注意事項

PARI Library を使う場合，他のインクルードファイルと共に`#include <pari.h>`として，“pari.h” をインクルードしておく．C 言語のプログラム内で PARI の関数

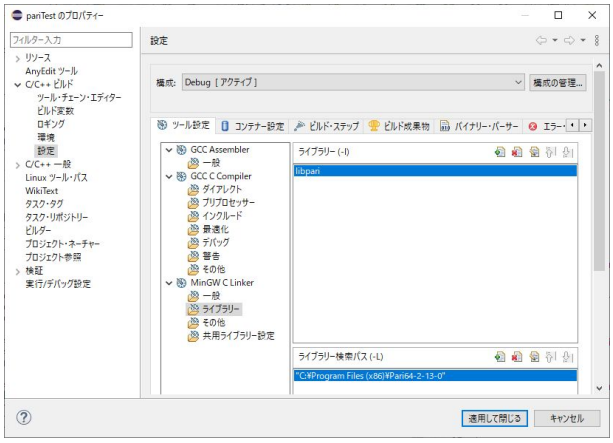

図 8: PARI/GP Download ページ

を使う場合，一定サイズのスタック領域が必要となる．そのために領域を確保しておく関数は，

`void pari_init(size_t size, ulong maxprime)`

であり，第一引数に必要なメモリ値を，第 2 引数には予め求めておく素数の上限を設定する．`size` は，アドレス `bot` から `top` (`bot` < `top`) までに入る `long` の個数，つまり

$$(\texttt{top} - \texttt{bot})/\texttt{sizeof(long)}$$

を設定する．素数を使った計算や関数を活用するのでなければ第 2 引数の値は 0 でもよいが，第 1 引数の値は 500000 を下回らないように設定する (p. 13)．

PARI の関数を使うプログラムでは，引数と戻り値に C 言語による型と PARI の型 (タイプ) が混在するが，PARI 関数の整数値のほとんどが `long` 型となることに注意する．PARI の型宣言は一般的に `GEN` で，オブジェクトへのポインタとなり，表示や書き込みを行うときは書式指定子`%Ps` を使う．

スタックとして確保されたメモリ領域には PARI オブジェクトが入るが，`top` の位置から下に押し込んでいく形になるで，底 `bot` まで達するとエラーとなる．したがって，不要なオブジェクトに対しては“garbage collection”によってそのスタック領域を解放しておく必要がある．現在のスタック位置に対するポインタ変数が `avma` (**av**ailable **m**emory **a**ddress) で，一般にスタック位置に対するポインタ変数は `pari_sp` (**pari** stack **p**ointer) タイプなので下のようなコード書けば，あいだで生成された PARI オブジェクトは，これ以降で上書きされることになる (pp.16–17)．

```
pari_sp av = avma;
 ..............
avma = av;
```

この他にも，関数

`GEN gerepile(pari_sp ltop,pari_sp lbot,GEN q)`

は `lbot` と `ltop` の間のオブジェクトを削除して `avma` の位置を上げ，PARI オブジェクト `q` のアップデートされたアドレスを返す (pp.18–19)．

4.2 基本的処理と線形方程式解法に使う関数

本研究で扱う具体的なプログラムは，与えられたバイナリコードから整数の剰余環 $\mathbb{Z}/8\mathbb{Z}$ 上の Self-Dual Code を生成するものであり，主に使う PARI の関数は行列に関するものと連立一次方程式の解を全て求めるためのものである．ここでそれらを簡単に見ておく．

以下では PARI のオブジェクトとして `gN` で整数，`gY` で列ベクトル，`gM` で行列，`gX` で一般のオブジェクトを表すものとする．また，`n` は C 言語における `long` 型整数である．

整数の変換：

- `stoi(n)`: 整数 n を PARI の整数に変換
- `gmodulo(gX, gN)`: PARI オブジェクトの成分を `gN` を法した値に変換

行列関連関数：

- `matsize(gM)`: 行列サイズ (行数–>列数の順)
- `mkvecs(n)`: `n` を要素とする 1 次元ベクトル
- `vconcat(gV_1, gV_2)`: 列ベクトルの結合
- `shallowmatconcat(gM_1, gM_2)`: 行列の行方向への結合 (“shallow” は簡易版)
- `gcoeff(gV, n)`: ベクトルの第 `n` 成分 ($1 \leq n$)
- `gcoeff(gM, n_1, n_2)`: 行列の (`n_1`, `n_2`) 成分
- `det(gM)`: 行列式
- `ginv(gM)`: 逆行列
- `gtrans(gM)`: 転置行列

  ベクトルや行列に関しては四則演算や成分に関する演算もいくつかあるが，PARI のオブジェクトとして扱わなくて済む部分はできるだけ C 言語の処理で行うこととした．

連立一次方程式の解：

- `gaussmodulo2(gM, gn, gY)`: 連立一次方程式に対応する式 $gM \times gX = gY \mod gN$ を満たす列ベクトル `gX` を求める．解が存在する時，第 1 成分としてその代表解列ベクトル，第 2 成分として不定解列ベクトルからなる行列を返す．解法は剰余環上で行っているが，成分値は PARI の整数となるので gN の値を超えることがあり，必要に応じて `gmodulo` で変換する．

- `lift_shallow(gMm)`: `gMm` は `gmodulo` で変換された成分を持つ行列で，これにより成分値を PARI の整数に戻す．

4.3 繰り返し処理

PARI には合成数を動く `forcomposite`，素数の動く `forprime`，ベクトルを動く `forvec`，置換を動く `forperm` などの繰り返し処理がある (pp. 42–44)．ここでは，前述の連立一次方程式の基本解に不定解の組み合わせを足し合わせたものすべてを求めたいので，以下のような 1 から n までの集合の部分集合すべてを動く `forsubset` で処理を使う．“garbage collection” も含めた，一般的な処理の流れ次のようになる (p.44)．

```
forsubset_t T;
GEN sub;
pari_sp av = avma, av2;
forallsubset_init(&T, n);
av2 = avma;
while((sub=forsubset_next(&T)){
     .....
     avma = av2;
}
avma = av;
```

5 誤り訂正符号と剰余環上の Self-Dual Code

まず誤り訂正符号の簡単な説明と，その例を示す．続いて，整数の剰余環 $\mathbb{Z}/8\mathbb{Z}$ 上の Self-Dual Code を生成するための拡張アルゴリズムを示す．

5.1 誤り訂正符号

デジタル情報処理における基本的な単位は 0 と 1 で表現されるビット情報であり，それらをいくつか並べたものを符号語 (code word)，いくつかの符号語の集まりを符号 (code) と呼ぶ．

例えば，次のビット行列の 4 つの行ベクトルが 2 進数体 $\mathbb{F}_2$ 上で生成する $2^4 = 16$ 個のベクトル全体を一つの符号 $\mathcal{C}_0$ と考える．

$$\mathcal{C}_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{1}$$

この符号では，16 個の内からどの 2 つの符号語をとっても 4ヵ所以上異なり，Hamming 重みによる最小距離

$$min\{d_H(\vec{c}, \vec{c'}); c, c' \in \mathcal{C}_0\} = min\{wt(\vec{c}); c \in \mathcal{C}_0\} = 4$$

の線形符号となっているので，$[8, 4, 4]$ 型の線形バイナリコードと呼ばれる．

5.2 剰余環 $\mathbb{Z}/8\mathbb{Z}$ 上の Self-Dual Code

1994 年に Hammons 等は，以前から知られていた非線形符号である Kerdock 符号，Preparata 符号，Goethals 符号などが，整数の剰余環 $\mathbb{Z}/4\mathbb{Z}$ 上の Self-Dual Code と呼ばれる符号と対応していることを示し ([9])，それ以降剰余環や有限体の拡大環上の誤り訂正符号の研究が盛んに行われた．我々は 2009 年に任意の素数 p に対しその 3 乗による整数の剰余環 $\mathbb{Z}/p^3\mathbb{Z}$ 上の Self-Dual Code の構成法を示すことで，符号全体の数を求める Mass Formula と呼ばれる式を与えた ([17])．

Self-Dual Code は自分自身がその直交補空間と一致する符号のことであり，この時はベクトル空間としての次元は $n/2$ となる．また任意の符号語 $\vec{c}$ に対し 1 の個数が 4 で割り切れるとき，その符号を doubly even Self-Dual Code と呼ぶ．我々のアルゴリズムは，2 進数体上の doubly even Self-Dual Code を与え，$\mathbb{Z}/8\mathbb{Z}$ 上の Self-Dual Code を構成するものであり，また任意の Self-Dual Code がこの方法で構成されることを示した．

ここでは，与えられた $[n, n/2, d]$ 型バイナリ Self-Dual Code $\mathcal{C}_0$ から，$\mathbb{Z}/8\mathbb{Z}$ 上の Self-Dual Code を生成するための具体的な構成法を示す．まず $\mathcal{C}_0$ の $n/2$ 個の生成ベクトルを doubly even なもの k 個とそれ以外の l 個 ($n = k+l$) のパートに分割し，次のように表す．

$$\begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} I_k & A_2 & A_{30} & A_{40} \\ 0 & I_l & B_3 & B_{40} \end{bmatrix}$$

ただし，I_k，I_l はそれぞれ k 次，l 次の単位行列であり，A_{40} は適当な列変換で逆行列を持つようにする．これから，次のような $\mathbb{Z}/8\mathbb{Z}$ 上の $[n, k+2l]$ 型 Self-Dual Code

$\mathcal{C}$ を生成する．

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} I_k & A_2 & A_{30}+2A_{31} & A_{40}+2A_{41}+4A_{42} \\ 0 & 2I_l & 2B_3 & 2B_{40}+4B_{41} \\ 0 & 0 & 4I_l & 4C_4 \end{bmatrix}$$

ここで，バイナリ行列として求めなければならないのは，A_{31}，A_{41}，A_{42}，C_4，および B_{41} であるが，最後の 2 つは (mod 2) における次式から一意的に求まる．

$$\begin{cases} C_4 \equiv (-A_{40}^{-1}A_{30})^t \\ B_{41}^t \equiv -A_{40}^{-1}(D + A_{31}B_3^t + A_{41}B_{40}^t) \\ \quad \text{ただし} \quad D = \frac{1}{2}(A_2 + A_{30}B_3^t + A_{40}B_{40}^t) \end{cases}$$

A_{31} と A_{41} は，$(f_{ij}) = \frac{1}{2}(I_k + A_2A_2^t + A_{30}A_{30}^t + A_{40}A_{40}^t)$ に対し，次の式を満たすバイナリ行列である．

$$(f_{ij}) + \widetilde{A_{30}A_{31}^t} + \widetilde{A_{40}A_{41}^t} \equiv 0.$$

ただし，ここで $\widetilde{X} = X + X^t$ である．

[ステップ 1]

$A_{30} = (\vec{a}_i^3)$, $A_{31} = (\vec{x}_i)$, $A_{40} = (\vec{a}_i^4)$, および $A_{41} = (\vec{y}_i)$ としたとき，ベクトルの内積 $\vec{x}\cdot\vec{y}$ を使って次の $(l+k)k = \frac{1}{2}nk$ 個の変数を持つ $\frac{1}{2}k(k-1)+k = \frac{1}{2}(k+1)k$ 個の制御条件を満たす連立一次方程式を解くことになる．

$$\begin{cases} (\vec{a}_i^3 + \vec{1}_l)\cdot\vec{x}_i + (\vec{a}_i^4 + \vec{1}_k)\cdot\vec{y}_i & \equiv \frac{1}{2}f_{ii} \\ & (i = 1, \ldots, k), \\ \vec{a}_i^3\cdot\vec{x}_j + \vec{x}_i\cdot\vec{a}_j^3 + \vec{a}_i^4\cdot\vec{y}_j + \vec{y}_i\cdot\vec{a}_i^4 & \equiv f_{ij} \\ & (1 \le i < j \le k). \end{cases} \tag{2}$$

[ステップ 2]

$(h_{ij}) = \frac{1}{2}((f_{ij}) + \widetilde{A_{30}A_{31}^t} + \widetilde{A_{40}A_{41}^t})$ に対し，$A_{42} = (z_{ij})$ と置いたときに k^2 個の変数を持つ，次のような $\frac{1}{2}k(k-1)$ 個の制御条件を満たす連立一次方程式を解くことである．

$$\begin{aligned} \vec{a}_i^4\cdot\vec{z}_j + \vec{z}_i\cdot\vec{a}_i^4 &\equiv h_{ij} + \vec{x}_i\cdot\vec{x}_j + \vec{y}_i\cdot\vec{y}_j \\ &\quad (1 \le i < j \le k). \end{aligned} \tag{3}$$

上記の方法によって，$\mathcal{C}_0$ から派生するの $\mathbb{Z}/8\mathbb{Z}$ 上の Self-Dual Code がすべて求まることになるが，ここで最小距離 d を大きくすることを考えたい．2 進数体上では，符号語の重さは 1 の個数であったが，ここでは $\mathbb{Z}/8\mathbb{Z} = \{0,1,2,3,4,5,6,7\}$ の各要素の重みを定義する必要がある．その為に各要素に 3 ビットの Gray Code

$$\{(000),(001),(011),(010),(110),(111),(101),(100)\},$$

を対応させることで，重さを与える ([18])．つまり，偶数 $2,4,6$ は重さ“2”，$1,3,7$ は重さ“1”，5 は重さ“3”である．

ここでは，基底となる符号ベクトルの重さをできるだけ大きくするように符号を構成したい．そのためには A_{40} の (i,j) 成分が 1 ならば，A_{41}，A_{42} の (i,j) 成分をそれぞれ 0，1 となるようにすればよい．A_{41} を求めるための連立一次方程式 (2) は，変数が $\frac{1}{2}nk$ 個あり，方程式の数 $\frac{1}{2}k(k+1)$ との差 $\frac{1}{2}(n-k-1)k$ が自由度となるので，それを使って 0 の位置を増やすことを考える．

$A_{42} = (z_{ij})$ においては連立一次方程式 (3) に $\frac{1}{2}k(k+1)k$ の自由度があるので，対応する部分が 1 となるように連立一次方程式を設定すればよい．

A_{30} における (i,j) 成分が 0 ならば A_{31} の対応する成分を 1 としたいが，その分制約式が増えてしまい，場合によると連立一次方程式 (2) が解を持たなくなるので，今回はその部分の条件は設定しないこととした．

6 PARI Library をインクルードした C 言語によるプログラム

プログラム作成の基本的なコンセプトはできる限り C 言語による処理を行い，特に `main` 内では PARI Library の使用を最小限にとどめることである．そのため，行列の基本的な演算用に関数を作成し，PARI の行列やベクトルとの変換も関数を呼び出して行う．最初に与える行列 $\mathcal{C}_0$ は (1) で固定し，分割を `main` 内で与えることとした．

6.1 PARI Library を使ういくつかの関数

C 言語による行列を PARI 行列へ変換する関数：

```
GEN toPariMat(int r_size,int c_size,+
                        nt m[][c_size]){
  GEN gm;
  gm=gtovec(stoi(m[0][0]));
  int i,j;
  for(i=1;i<c_size;i++){
    gm=shallowconcat(gm,mkvecs(m[0][i]));
  }
  gm=gtomat(gm);
  for(i=1;i<r_size;i++){
       GEN mm1;
       mm1=gtovec(stoi(m[i][0]));
       for(j=1;j<c_size;j++){
          mm1=shallowconcat(mm1,
                              mkvecs(m[i][j]));
       }
       gm=vconcat(gm,gtomat(mm1));
     }
  return gm;
}
```

PARI 行列を C 言語による行列変換する関数：

```
void toIntMat(int rs,int cs,GEN m,+
                    int im[rs][cs]){
  int i,j;
  for(i=0;i<rs;i++){
    for(j=0;j<cs;j++){
      im[i][j]=(int)itos(gcoeff(m,i+1,j+1));
    }
  }
}
```

特定解と不定解集合を与えてすべての解を求める関数：

```
int genAllas(int countF,int cs,int ix[],+
                int imF[][cs],int as[][cs]){
  int i;
  pari_sp av=avma,av2;
  forsubset_t T;
  GEN subsmallM;
  forallsubset_init(&T,(long)countF);
  av2=avma;
  int count=0;
  while((subsmallM=forsubset_next(&T))){
    GEN subM=gtovec(subsmallM);
    int l=(int)itos(matsize(subM)[2]);
    int subS[l];
    toIntVec(l,subM,subS);
    for(i=0;i<l;i++){
      int si=subS[i]-1;
      ppVec(cs,imF[si],as[count]);
    }
    ppVec(cs,ix,as[count]);
    count++;
    avma=av2;
  }
  avma=av;
  return count;
}
```

連立一次方程式の解を求める関数：

```
int solveEquation(int rs,int cs,int iy[rs],+
  int im[rs][cs],int ix[cs],int imF[][cs]){
  pari_sp av=avma,av2;
  GEN x,y,x1,x2;
  GEN m;
  av2=avma;
  y=lift_shallow(gmodulo(toPariVec(rs,iy),+
            stoi(2)));
  y=gtrans(y);
  m=lift_shallow(gmodulo(toPariMat(rs,cs,+
            im),stoi(2)));
  x=gaussmodulo2(m,stoi(iMod),y);
  x1=lift_shallow(gmodulo(x[1],stoi(2)));
  toIntVec(cs,x1,ix);
  x2=lift_shallow(gmodulo(x[2],stoi(2)));
  int matRS=(int)itos(matsize(x2)[2]);
  int i,j;
  int cF=0;
  int ix1[cs];
  int imm[(K+L)*K][cs];
  toIntVec(cs,gtrans(x1),ix1);
  toIntMat(matRS,cs,gtrans(x2),imm);
  int im0[cs];
  for(i=0;i<cs;i++){
    im0[i]=0;
  }
  avma=av2;
  avma=av;
  if(matRS>0){
    for(i=0;i<matRS;i++){
      if(isSameVec(cs,im0,imm[i])!=0){
        for(j=0;j<cs;j++){
          imF[cF][j]=imm[i][j];
        }
    cF++;
    }
  }
  return cF;
  }else if(matRS==0){
    for(i=0;i<cs;i++){
      imF[0][i]=0;
    }
    return 0;
  }else{
    return 0;
  }
}
```

6.2 インクルード，定義，および main プログラム

インクルードするヘッダファイルと，定数の定義は以下のようにして，定義の K と L は分割の仕方によって変えることになる．

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pari.h>

#define N 8
#define iMod 8
#define maxW 3

//#define K 4
//#define L 0

//#define K 3
//#define L 1

//#define K 2
//#define L 2
```

```
#define K 1
#define L 3
```

main において，分割に合わせて行列を定義しており，ここでは K=1，L=3 のものが使われる設定となっている．生成されたコードはその Weight Distribution 多項式を求めて，ファイルに書き込むこととした．なお，以下のプログラムにおいて “,” や “(” の後に “+” がある場合は，次行に続くことを示している．

```
int main(void) {
  pari_init(1000000,1000);
  int wt[(iMod)] = {0,1,2,1,2,3,2,1};
  int iK2 = K*K;
  int iL2 = L*L;
  int iKL = K*L;
  int iKpL = K+L;
  int iKpLpL = K+2*L;
  int iKL2 = iL2+iK2;
/* int iA20;
  int iA30;
  int iA40[K][K]={{0,1,1,1},{1,0,1,1},+
        {1,1,0,1},{1,1,1,0}};
  int iB30;
  int iB40;
  int iA20[K][L]={{0},{0},{0}};
  int iA30[K][L]={{0},{1},{1}};
  int iA40[K][K]={{1,1,1},{0,1,1},{1,0,1}};
  int iB30[L][L]={{1}};
  int iB40[L][K]={{1,1,0}};
  int iA20[K][L]={{0,0},{0,0}};
  int iA30[K][L]={{1,1},{1,1}};
  int iA40[K][K]={{0,1},{1,0}};
  int iB30[L][L]={{0,1},{1,0}};
  int iB40[L][K]={{1,1},{1,1}};
*/
// 分割として使われる値は以下の 5 行
  int iA20[K][L]={{0,0,0}};
  int iA30[K][L]={{0,1,1}};
  int iA40[K][K]={{1}};
  int iB30[L][L]={{1,0,1},{1,1,0},{1,1,1}};
  int iB40[L][K]={{1},{1},{0}};

  int iC41[L][K];
  int iCT41[K][L];
  int i,j;
  FILE *fp;// 結果書き込みファイル
  int iC0[iKpL][N];
  int invA40[K][K];
  // A40 の逆行列計算 (ここだけ PARI 関数使用)
  GEN m = lift_shallow(gmodulo(+
          toPariMat(K,K,iA40),stoi(2)));
  GEN invm = lift_shallow(+
          ginv(gmodulo(m,stoi(2))));
  if((int)itos(det(m))!=0){
    printf("\n determinate is not zero\n");
    toIntMat(K,K,invm,invA40);
  }else{
    printf("\n determinate of A40 is zero\n");
  }
 //  L が正のときに C4 を計算
  if(L>0){
    mulMatrix(K,K,L,invA40,iA30,iCT41);
    transMatrix(K,L,iCT41,iC41);
    iMmod2(L,K,iC41);
    genC0(iKpL,iKpLpL,iA20,iA30,+
            iA40,iB30,iB40,iC0);
  }else if(L==0){
    genC00(iA40,iC0);
  }

  iMmod2(iKpL,iKpLpL,iC0);
  int zeroN = 0;
  int iY1[zeroN+K*(K+1)/2];
  int iM1[zeroN+K*(K+1)/2][(L+K)*K];
// ステップ 1 の右辺列ベクトル生成
  genY1(zeroN,iC0,iY1);
  iVmod2(zeroN+K*(K+1)/2,iY1);
// ステップ 1 の制約行列生成
  if(L>0){
    genM1(zeroN,iA30,iA40,iM1);
  }else if(L==0){
    genM10(iA40,iM1);
  }

  iMmod2(zeroN+K*(K+1)/2,iKpL*K,iM1);
  fp = fopen("y1_a1s.txt","a");

  int ttE;
  int imF[iKpL*K][iKpL*K];
  int countF;
  int ix[iKpL*K];
// ステップ 1 の式 (2) の特定解 (ix) と不定解
(imF)
  countF = solveEquation(zeroN+K*(K+1)/2,+
              iKpL*K,iY1,iM1,ix,imF);
  fprintVec(fp,"ix",iKpL*K,ix);
  fclose(fp);
  ttE = twoPowers(countF);

  int iA1s[ttE][iK2+iKL];
  for(i=0;i<ttE;i++){
    for(j=0;j<iK2+iKL;j++){
      iA1s[i][j] = 0;
    }
  }
// ステップ 1 における解ベクトル全体の生成
  int count = genAllas(countF,iK2+iKL,ix,+
                imF,iA1s);
  for(i=0;i<count;i++){
    iVmod2(iK2+iKL,iA1s[i]);
  }
```

```
  int iA31s[count][iKL];
  int iA41s[count][iK2];
  if(L>0){
    devideAs(count,iKpL,iKL,iA1s,iA31s,+
             iA41s);
  }else if(L==0){
    for(i=0;i<count;i++){
      for(j=0;j<K*K;j++){
        iA41s[i][j]=iA1s[i][j];
      }
    }
  }
  fp=fopen("y1_a1s.txt","a");
  fprintf(fp,"countF=%d, count=+
             %d \n",countF,count);
  fprintMat(fp,"imFree",countF,iKpL*K,imF);
  for(i=0;i<count;i++){
    if(L>0){
      fprintf(fp,"iA31s[%d]=",i);
      fprintVec(fp,"",iKL,iA31s[i]);
    }
    fprintf(fp,"iA41s[%d]=",i);
    fprintVec(fp,"",iK2,iA41s[i]);
  }
  fclose(fp);

  int number;
  int iA31[K][L];
  int iA41[K][K];
  int iB41[L][K];
  int imF2[iK2][iK2];
  int wtdist[(maxW)*N];
  for(i=0;i<iK2;i++){
    for(j=0;j<iK2;j++){
      imF2[i][j]=0;
    }
  }
  int count2;
  int ix2[iK2];
  int c[iKpLpL][N];

  for(number=0;number<count;number++){
    int istr = 100*K+number;
    char str2[10];
    char str3[]=".txt";
    itoa(istr,str2,10);
    strcat(str2,str3);
    fp = fopen(str2,"w");
    fprintf(fp,"test number %d \n",number);
// A31, A41 を固定
    vecToMat(K,L,iA31s[number],iA31);
    vecToMat(K,K,iA41s[number],iA41);
    fprintVec(fp,"iA31s",K*L,iA31s[number]);
    fprintMat(fp,"iA31",K,L,iA31);
    fprintMat(fp,"iA41",K,K,iA41);
    fprintMat(fp,"iC41",L,K,iC41);
    fclose(fp);
// ステップ 2 における結果書出し用ファイル
    fp = fopen(str2,"a");
    zeroN = 0;
    for(i=0;i<K;i++){
      for(j=0;j<K;j++){
        if(i!=j){
          if(iA41[i][j]==0){
            zeroN++;
          }
        }
      }
    }
// A31, A41 から B41 を計算
    genB41(iKpL,iKpLpL,invA40,iC0,iA31,+
                 iA41,iB41);
    iMmod2(K,L,iB41);
    fprintf(fp,"zeroNumber is %d \n",+
              zeroN);

    int iY2[zeroN+K*(K-1)/2];
    int iM2[zeroN+K*(K-1)/2][iK2];
// ステップ 2 の右辺列ベクトル生成
    genY2(iKpL,iKpLpL,zeroN,iC0,iA31,+
              iA41,iY2);
// ステップ 2 の制約行列生成
    genM2(zeroN,iA40,iA41,iM2);
    iVmod2(zeroN+K*(K-1)/2,iY2);
    iMmod2(zeroN+K*(K-1)/2,iK2,iM2);
    fprintVec(fp,"Y2",zeroN+K*(K-1)/2,+
           iY2);
    fprintMat(fp,"M2",+
           zeroN+K*(K-1)/2, iK2,iM2);
// ステップ 2 における解ベクトル全体の生成
    countF=solveEquation(+
          zeroN+K*(K-1)/2,iK2,+
          iY2,iM2,ix2,imF2);
    fprintf(fp,"countF=%d \n",countF);
    fprintVec(fp,"x2",iK2,ix2);
    fprintMat(fp,"mF2",countF,iK2,imF2);

    ttE = twoPowers(countF);
    int a42s[ttE][iK2];
    for(i=0;i<ttE;i++){
      for(j=0;j<iK2;j++){
        a42s[i][j] = 0;
      }
    }
// ステップ 2 における解ベクトル全体の生成
    count2 = genAllas(countF,iK2,ix2,+
           imF2,a42s);
    fprintf(fp,"count2=%d \n",count2);
    for(i=0;i<count2;i++){
      iVmod2(iK2,a42s[i]);
     fprintf(fp,"a2s[%d]=",i);
```

```
      fprintVec(fp,"",iK2,a42s[i]);
    }

    for(i=0;i<count2;i++){
// 各 A42 に対し符号生成
      genC(iKpL,iKpLpL,iC0,iA31,iA41,+
           iB41,iC41,a42s[i],c);
      iMmod8(iKpLpL,N,c);
      fprintf(fp,"%d:",i);
      fprintMat(fp,"c",iKpLpL,N,c);
      for(j=0;j<iKpLpL;j++){
        fprintf(fp,"wt[c%d]=%d, ",+
                j+1,wtOfVec(wt,N,c[j]));
      }
      fprintf(fp,"\n");
      wtDist(wt,iKpLpL,N,c,wtdist);

      int minWV = calMinWeight((maxW)*N,+
                  wtdist);
      int maxWV = calMaxWeight((maxW)*N,+
                  wtdist);
// 生成された符号語の最小，最大重み
      fprintf(fp,"Minimun Weight=%d,+
            Maximum Weight=%d,+
            \n Weight Distribution:\n",+
            minWV,maxWV);
// Weight Distribution の書込み
      for(j=0;j<maxW*N;j++){
        if(wtdist[j]!=0){
          fprintf(fp,"%d*X^%d+",+
                  wtdist[j],j);
        }
      }
      fprintf(fp,"\n");
    }
    fclose(fp);
  }
  int end;
  scanf("%d",&end);
  pari_close();
  return EXIT_SUCCESS;
}
```

7　結果と今後の課題

$[8,4,4]$ 型バイナリ線形符号 (1) に対し，4 通りの分割 $(k,l)=(4,0),(3,1),(2,2),(1,3)$ を行い，それぞれに対して $\mathbb{Z}/8\mathbb{Z}$ 上の Self-Dual Code で符号語生成ベクトルの重みができるだけ大きくなるようなものを計算した．ただし，$(k,l)=(2,2)$ の場合は，A_{40} を正則とするため，7，8 列と 5，6 列の入れ替えを行った．

$(k,l)=(4,0)$ の場合は，ステップ 1 で 64 個の A_{41} を生成し，それぞれに対して $2\sim 128=2^7$ 個の $\mathbb{Z}/8\mathbb{Z}$ 上 Self-Dual Code を得た．また，すべてで，最小重みは 6 となっていた．

$(k,l)=(3,1)$ の場合は，ステップ 1 で 64 個の A_{31} と A_{41} の組を生成し，それぞれに対して $2\sim 32=2^5$ 個の $\mathbb{Z}/8\mathbb{Z}$ 上 Self-Dual Code を得た．また，すべてで最小重みは 2，次が 6 となっていた．

$(k,l)=(2,2)$ の場合は，ステップ 1 で 32 個の A_{31} と A_{41} の組を生成し，それぞれに対して $2\sim 8=2^3$ 個の $\mathbb{Z}/8\mathbb{Z}$ 上 Self-Dual Code を得た．また，すべてで最小重みは 2，次が 4 となっていた．

$(k,l)=(1,3)$ の場合は，ステップ 1 で 8 個の A_{31} と A_{41} の組を生成し，それぞれに対して 1 か 2 個の $\mathbb{Z}/8\mathbb{Z}$ 上 Self-Dual Code を得た．また，すべてで最小重みは 2，次が 4 となっていた．

$(k,l)=(4,0)$ の場合が一番時間がかかっているが数分程度であり，それほどストレスを感じることがなく終了する．扱った連立一次方程式に対応する行列は，行数 10 前後，列数 16 であり，この程度ならば PARI の関数は問題なくすべての解を出力してくれることがわかった．

今回は，特定の $[8,4,4]$ 型バイナリ線形符号から出発したものであったが，他の符号に対しても適応できることを確認する必要がある．

参考文献

[1] Behr, Bernhard, Seminar Report Plankalkül, 31 July 2015, `cl-informatik.uibk.ac.at/teaching/ss15/bob/reports/ss15-BB.pdf`(2021/03/12 参照)

[2] Dahl, O. -J., Dijkstra, E. W., and Hoare, C. A. R., *Structured Programming*, A. P. I. C. Studies in Data Processing, No. 8, ACADEMIC PRESS Inc. (London), (1972)

[3] Dahl, O. -J., The Birth of Object Orientation: The Simula Languages, *Object-Orientation to Formal Methods*, Lecture Notes in Computer Science, 2635, (2004), pp. 15–25.

[4] Dijkstra, E. W., Structured programming, *Report on a conference sponsored by NATO SCIENCE COMMITTEE*, (1969), pp. 65–68

[5] Dijkstra, E. W., EWD 498: How do we tell truths that might hurt?, *Selected Writings on Computing: A Personal Perspective*, Springer-Verlag, (1982), pp. 129-131

[6] Fuegi, J. and Francis, J., Lovelace & Babbage and the Creation of the 1843 ‘Notes’, *IEEE Annals of the History of Computing*, Vol. 25, Issue 4, (2003) pp. 16–26. `https://pdfs.semanticscholar.org/81bb/\f32d2642a7a8c6b0a867379a4e9e99d872bc.pdf` (2021/03/12 参照)

[7] Giloi, Wolfgang, K., Konrad Zuse's Plankalkül: The First High-Level, "non von Neumann" Programming

Language, *IEEE Annals of the History of Computing*, Vol. 19, (1997), pp. 17–24.

[8] Goldstine, Herman H., *The Computer: from Pascal to von Neumann*, Princeton University Press. ISBN 0-691-02367-0, (1972)

[9] Hammons, A.R., Jr., Kumar, P.V., Calderbank, A.R., Sloane, N.J.A.and Solé, P., The $\mathbb{Z}/4\mathbb{Z}$ linearity of Kerdock, Preparata, Goethals and related codes, *IEEE Trans. Inform. Theory* 40 (1994) pp. 301–319.

[10] Hearn, A. C., REDUCE: The First Forty Years, *Invited paper presented at the A3L Conference in Honor of the 60th Birthday of Volker Weispfenning*, (2005). `http://www.reduce-algebra.com/reduce40.pdf`, (2021/03/12 参照)

[11] Iverson, K. E., *A Programming Language*, JOHN WILEY AND SONS, INC, (1967).

[12] Kernighan, B. W and Ritchie, D. M., *The C Programming Language (2nd ed.)*, Englewood Cliffs, NJ: Prentice Hall. ISBN 0-13-110362-8., (1988)

[13] Kilburn, T., Tootill, G. C. and Williams, F. C., Universal High-Speed Digital Computers: A Small-Scale Experimental Machine. *J.l.E.E.* 98 Part II, No. 61 (1951). `http://curation.cs.manchester.ac.uk/computer50/www.computer50.org/kgill/mark1/ssem.html`, (2021/03/12 参照)

[14] Knuth, D. E., Structured Programming with go to Statements, *Computing Serveys*, Vol. 6, No. 4, (1974), 261–301.

[15] Knuth, D. E., and Pardo, L. T., The Early Development of Programming Languages, *A History of Computing in the Twentieth Century*, Academic Press, Elsevier, (1980), pp. 197–273

[16] Linger, R. C., Mills, H. D., and Witt, B. I., *Structured Programming: Theory and Practice*, Addison-Wesley Publication, (1979)

[17] Nagata, K., Nemenzo, F., and Wada, H., The number of self-dual codes over $\mathbb{Z}_{p^3}$, *Designs, Codes and Cryptography* 50 (2009) pp. 291–303.

[18] Nagata, K. and Nemenzo, F., Some Properties of Binary Gray Code, *Proceedings of International Conference on Computer Application Technologies*, Aug. 31-Sep. 2 (2015) pp. 72–75.

[19] Richards, M., The portability of the BCPL compiler, *Software:Practice and Experience*, Vol. 1, Issue 2, https://doi.org/10.1002/spe.4380010204(1971), pp.135–146.

[20] Richards, M., EDSAC Initial Orders and Squares Program, *University of Cambridge Computer Laboratory*, `https://www.cl.cam.ac.uk/~mr10/Edsac/edsacposter.pdf`, (2021/03/12 参照)

[21] Ritchie, D. M., The Development of the C Language, *Bell Labs/Lucent Technologies*, Murray Hill, NJ 07974 USA, `https://www.bell-labs.com/usr/dmr/www/chist.pdf`, (2021/03/12 参照)

[22] Stern, Nancy, *From ENIAC to UNIVAC : an appraisal of the Eckert-Mauchly computers*, Digital Press history of computing series, Digital Press, 1981

[23] Turing, A. M., On Computable Numbers, with an Application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society*, 2 42: pp. 230–265, (1936) `https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf`, (2021/03/12 参照)

[24] Zuse, Konrad, Über den allgemeinen Plankalkül als Mittel zur Formulierung schematisch-kombinativer Aufgaben, Arch. Math. 1, (1948/49), pp. 441–449.

[25] 永田清，PARI/GP による Self-Dual Code の生成，第 *5* 回国際 *ICT* 利用研究学会全国大会 *IIARS2020* 講演予稿集，2020 年 12 月 6 日，pp. 40-43