ANÁLISE COMPARATIVA DE SOFTWARES ALGÉBRICOS PARA RESOLUÇÃO DE EQUAÇÕES DIOFANTINAS

Lucas Eduardo da Silva[1], Clécio Luênio Lima Freire de Souza[1], Moésio Morais de Sales[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Ceará – *campus* Crato

{lucasseduhh, cleciolluenio, moesiom }@gmail.com

RESUMO: *Os algoritmos de resolução de equações diofantinas exigem softwares adequados que devem ter ao mesmo tempo flexibilidade no uso de funções e capacidade de processamento. Os CAS (Computer Algebra System) constituem uma importante ferramenta no auxílio à simulação e teste de diversos problemas de teoria dos números. Softwares e bibliotecas como PARI/GP e GiNaC que são otimizados e que contém ao mesmo tempo as funções adequadas para este trabalho são fundamentais. Este trabalho pretende servir como instrumento de apoio para comparação de softwares que auxiliam em pesquisas que exigem grande poder de processamento. Analisou-se o PARI/GP e a biblioteca GiNaC onde se pretende fazer um comparativo das suas principais vantagens e desvantagens.*

Palavras-chave: *CAS. PARI/GP. GiNaC. Poder de Processamento.*

ABSTRCT: *Diophantine equations resolution algorithms require appropriate software that must simultaneously have flexibility in the use of functions and processing capability. The Computer Algebra System (CAS) constitute an important tool in the simulation and test of various number theory problems. Softwares and libraries like PARI/GP and GiNaC that are optimized and that contains at the same time the appropriate functions stop this work are fundamental. This work aims to serve as a support tool for comparison of software that assists in searches that require great processing power. We will analyze PARI/GP and the GiNaC library where we intend to make a comparative of its main advantages and disadvantages.*

Keywords: *CAS. PARI/GP. GiNaC. Processing power.*

1. INTRODUÇÃO

O estudo pela otimização de algoritmos desperta grande interesse na área computacional e matemática, em teoria dos números o estudo das técnicas de solução de uma equação diofantina é considerado um problema muito relevante. Existem vários métodos de resolução de equações diofantinas, no presente trabalho usou-se o método em que tomamos a equação módulo um primo determinado convenientemente, devido ao fato de que neste podemos modelar e aplicar métodos computacionais.

O objetivo do trabalho é comparar os softwares GiNaC e PARI como ferramenta de teste e modelagem de equações diofantinas.

O trabalho inicia com um estudo dos principais teoremas e propriedades de teoria dos números e, em seguida estudaremos as equações diofantinas exponenciais apresentando o algoritmo de Styer onde se fará então uma exposição sucinta sobre as estratégias de resoluções de equações que foram utilizadas nos testes. Há muitas maneiras de resolver tais equações: analisar módulo algum primo conveniente e utilizar o método de Baker. Em seguida, apresentou-se o conceito geral de CAS e, na sequência foi dado as principais características do GiNaC e PARI/GP e os códigos construídos para execução dos testes. Após isso, a seção principal apresenta os resultados onde realizou-se de forma limitada um conjunto padrão de avaliações a partir do qual obteve-se os resultados relativos às soluções e os tempos necessários para obtenção das mesmas utilizando os códigos criados para modelá-las. Ao final, a conclusão apresenta considerações finais sobre os resultados encontrados na implementação dos códigos.

2. A ESTRATÉGIA DE RESOLUÇÃO DE EQUAÇÕES DIOFANTINAS

Para realização dos testes utilizou-se formas modeladas pela primeira parte do algoritmo de Styer, pois dada certa entrada, ela produzirá sempre a mesma saída, passando pela mesma sequência de etapas logo o problema considerado constitui um problema determinístico como foi comprovado por STYER (1993, p. 813).

Antes de enunciarmos o resultado principal, definiremos alguns resultados da teoria dos números que serão apenas enunciados para que o trabalho não fuja aos objetivos principais, enquanto que alguns deles podem ser

utilizados sem menção explícita. Tais resultados podem ser obtidos em MOREIRA (2013, p. 193) e SANTOS (2010, p. 117).

2.1 Teoremas e Definições

Definimos a função de Euler φ por $\varphi(m)=|\{a\in Z;\ 0<a\leq m, mdc(a,m)=1\}|$ ondeR $n\in Z$, $n>0$ e $|X|$ denota o número de elementos de X. Assim, temos que $\varphi(1)=1$, $\varphi(2)=1$, e, para $n>2$, $1<\varphi(m)<m$. Na Tabela 1 temos alguns valores para $\varphi(n)$.

Tabela 1 – Valores para $\varphi(n)$.

n	1	2	3	4	5	6	7	8	9	10
$\varphi(n)$	1	1	2	2	4	2	6	4	6	4

Fonte: Elaborado pelo autor.

Decorre da definição anterior que se p é primo então $\varphi(p)=p-1$. Para potências de primo temos que dentre $1,2,\ldots,p^k$ não são coprimos com p^k aquele que têm p como fator primo, a saber $p,2p,3p,\ldots,p^{k-1}p$, portanto p^k-p^{k-1} são os coprimos, isto é

$$\varphi(p^k)=p^k-p^{k-1}=p^k\left(1-\frac{1}{p}\right)$$

A proposição a seguir garante que dado um inteiro n basta conhecer seus valores em potências de primos, isso devido ao fato que a função φ é multiplicativa.

Proposição 1. Se $n,m\in N$ são coprimos então $\varphi(nm)=\varphi(n)\varphi(m)$.

Seja $n\in N$ um número fixo. Dois números $a,b\in N$ chamam-se congruentes módulo n, se $a-b$ múltiplo de n. Em símbolos: $a\equiv b(modn)$. Assim

$$a\equiv b(modn)\Leftrightarrow n|a-b$$

Proposição 2. Para quaisquer $a,b,c,d,e,n\in Z$ temos:

1. $a\equiv a(modn)$
2. Se $a\equiv b(modn)$, então $b\equiv a(modn)$.
3. Se $a\equiv b(modn)$ e $b\equiv c(modn)$ então $a\equiv c(modn)$.
4. Se $a\equiv b(modn)$ e $c\equiv d(modn)$ então $a+c\equiv b+d(modn)$.
5. Se $a\equiv b(modn)$, então $-a\equiv -b(modn)$.
6. Se $a\equiv b(modn)$ e $c\equiv d(modn)$ então $a.c\equiv b.d(modn)$.
7. Se $mdc(c,n)=1$, então $ac\equiv bc(modn)\Leftrightarrow a\equiv b(modn)$.

MOREIRA (2013, p. 34) afirma que "as propriedades acima mostram que a relação $\equiv(mod\,n)$ ("ser congruente módulo n") tem um comportamento muito similar a relação de igualdade usual". As equações que utilizou-se aqui tem sua base em problemas de divisibilidade e estas propriedades listadas na Proposição 2 é o que tornam as congruências uma ferramenta tão adequada para este tipo de problema.

Teorema 1. (Euler). Sejam $m,a\in N$ com $m>1$ e $mdc(a,m)=1$. Então, $a^{\varphi(m)}\equiv 1(modm)$.

Proposição 3. Dado $a\in N$, existe $h\in N$ tal que $a^h\equiv 1(modm)$ se, e somente se, $mdc(a,m)=1$.

Definição 1. Seja $a,m\in N$, com $m>1$ e $mdc(a,m)=1$. Chamaremos de Ordem de a com respeito à m o número natural $\pi(a,m)=min\{i\in N;a^i\equiv 1(modm)\}$.

A proposição seguinte é o resultado básico mais importante sobre ordem.

Proposição 4. Temos que $a^n\equiv 1(modm)$ se, e somente se, $\pi(a,m)|n$.

Corolário 1. Sejam $a,m\in N$, com $mdc(a,m)=1$. Temos que $\pi(a,m)|\varphi(m)$.

Para MOREIRA (2013, p. 66) a proposição 3 e o corolário 1 nem sempre dá a melhor estimativa para $\pi(a,m)$. De fato, para $m=8$ temos que $\varphi(m)=4$ mas $\pi(a,m)$ divide 2 para todo a ímpar; para $m=9$ temos que $\varphi(m)=6$ mas $\pi(a,m)$ divide 3 para todo a inteiro da forma $3k+1$. Por outro lado, para m primo sempre existe um inteiro a com $\pi(a,m)=\varphi(m)$, neste caso diremos que a raiz primitiva módulo p.

Teorema 2. Sejam $a,m\in N$, com $mdc(a,m)=1$. Temos que $a^j\equiv a^k(modm)$ se, e somente se, $\pi(a,m)|j-k$.

Claro que $\pi(a,m)|j-k$ é equivalente, por definição de congruência, a $j\equiv k(mod\pi(a,m))$ ou $j\equiv k(mod\varphi(m))$.

2.2 O algoritmo de Styer

A primeira parte do algoritmo contém os seguintes passos:

Fixar os primos b e d . Começamos com um pré-processamento que envolve apenas b e d . Então se escolhe os valores para os parâmetros a , c , e e . Considerando equação $ab^x=cd^y+e$. Suponha que estamos interessados apenas em equações com soluções $x\geq h$. Tomando a equação $ab^x=cd^y+e$ módulo b^h . Seja β a ordem de d módulo b^h , então $d^\beta\equiv 1(mod\ b^h)$. (Nós encontramos β no nosso pré-processamento.) (STYER, 1993, p. 813).

Conforme STYER (1993, p. 812), nessa primeira etapa os primos são escolhidos entre os primos 2, 3, 5, 7, 11, 13. A seguir calculam-se as ordens em relação aos módulos considerados, digamos b^h e fatoramos $d^\beta-1$, para encontrarmos os primos que são utilizados no segundo passo do algoritmo mas, não está no escopo deste texto analisarmos a segunda etapa do algoritmo.

A seguir utilizou-se os dados do pré-processamento onde temos que:

Encontrar um valor de γ , digamos γ_0 , tal que $cd^{y_0}+e\equiv 0(modb^h)$. Muitas vezes não existe tal γ , o que contradiz a suposição de que a equação tem uma solução com $x\geq h$. Se houver uma solução com $x\geq h$, então γ deve satisfazer $y\equiv y_0(mod\beta)$. Considere qualquer fator primo p que divide $d^\beta-1$ com $mdc(ab,p)=1$. (Em nosso passo de pré-processamento, encontramos este conjunto de primos p .) Então $d^\beta\equiv 1(modp)$ (STYER, 1993, p. 813).

O que reduziu nossa equação inicial a $ab^x\equiv cd^{y_0}+e(modp)$.

Para STYER (1993, p.814), a prática, nos leva a crer que necessitamos de muito pouco cálculo para podemos verificar grandes faixas de valores para a , c e e . De fato, o algoritmo proposto é aplicado para valores de $a\leq 50$, $c\leq 50$ e $|e|\leq 1000$.

3. OS CAS A SEREM IMPLEMENTADAS NOS TESTES

Um sistema algébrico computacional é um tipo de software que é usado no manuseamento de fórmulas matemáticas. O foco principal de um sistema algébrico computacional é automatizar tarefas de manipulação algébrica tediosas e às vezes difíceis. A diferença primária entre um sistema algébrico computacional e uma calculadora tradicional é a eficácia em lidar com equações simbolicamente em vez de numericamente. As utilizações e capacidades específicas desses softwares variam muito de um sistema para outro, mas a finalidade permanece a mesma: manipulação de equações simbólicas. Os sistemas algébricos computacionais geralmente incluem pacotes instaláveis para representar equações gráficas e fornecer uma linguagem de programação para o usuário definir seus próprios procedimentos.

3.1 Sobre o GiNaC

GiNaC é uma biblioteca do C++. Elaborado para permitir a criação de sistemas integrados que incorporam manipulações simbólicas junto com áreas mais estabelecidas da ciência da computação (como computação-aplicações numéricas intensas, interfaces gráficas, etc.) sob um mesmo teto. É distribuído nos termos e condições da licença pública geral GNU (GPL). GiNaC é uma sigla iterada e recursiva para GiNaC is Not a CAS, onde CAS significa Computer Algebra System.

O GiNaC portanto, permitir a resolução de problemas em que os cálculos envolvidos exigem uma abordagem combinada simbólica e numérica.

O site oficial (https://www.ginac.de), lista abaixo como as principais vantagens da linguagem:

1. Linguagem familiar: Com o GiNaC você pode escrever seu programa em C ++ comum, que é padronizado.
2. Tipos de dados estruturados: você pode criar tipos de dados estruturados usando estruturas ou classes junto com o STL.
3. Fortemente tipado: no CAS, você normalmente tem apenas um tipo de variáveis que podem conter conteúdo de um tipo arbitrário.
4. Ferramentas de desenvolvimento: ferramentas de desenvolvimento poderosas existem para C ++, como editores sofisticados, depuradores, ferramentas de visualização, documentação.
5. Modularização: os programas C ++ podem ser facilmente divididos em módulos, separando a interface e a implementação.
6. Preço: GiNaC é distribuído sob a Licença Pública GNU, o que significa que é gratuito e está disponível com código fonte. E há excelentes C ++ - compiladores de graça também.

7. Extensível: você pode adicionar suas próprias classes ao GiNaC, estendendo-o em um nível muito baixo.
8. Integração perfeita: é algo entre difícil e impossível de chamar as funções do CAS dentro de um programa escrito em C ++ ou qualquer outra linguagem de programação e vice-versa. Com o GiNaC, suas rotinas simbólicas fazem parte do seu programa.
9. Eficiência: muitas vezes grandes partes de um programa não precisam de cálculos simbólicos. Para aplicações simbólicas puras, o GiNaC é comparável em velocidade com outro CAS.

A justificativa por trás do GiNaC vem da análise de que a maioria dos sistemas algébricos computacionais (CAS) atuais são exaustivos linguisticamente e semanticamente. Ainda que sejam ferramentas extremamente robustas para aprender matemática e resolver impasses específicos, estes são desprovidos de estruturas linguísticas modernas que permitam a criação de projetos em grande escala. O GiNaC é uma tentativa de contornar esta situação, expandindo uma linguagem computacional bem estabelecida e padronizada (C++) por algumas capacidades simbólicas primordiais, possibilitando sistemas integrados que englobam manipulações simbólicas junto com áreas mais estabelecidas da ciência da computação (como aplicações computacionais numéricas intensas, interfaces gráficas, etc.) sob um mesmo teto.

3.2. Sobre o PARI/GP

O PARI/GP ou PARI é um sistema algébrico computacional amplamente utilizado, projetado para cálculos rápidos em teoria dos números (fatorações, teoria dos números algébricos, curvas elípticas ...), mas também contém um grande número de outras funções úteis para calcular com entidades matemáticas, como matrizes, polinômios, séries de poder, números algébricos, etc., e muitas funções transcendentais. O PARI também está disponível como uma biblioteca C para permitir cálculos mais rápidos.

O site oficial (https://pari.math.u-bordeaux.fr) lista como principais vantagens:

- Sua velocidade (que pode ser entre 5 e 100 vezes melhor em muitos cálculos);
- A possibilidade de usar diretamente tipos de dados que são familiar aos matemáticos;
- Um extenso módulo de teoria dos números algébricos que não tem equivalente nos sistemas acima mencionados.

É possível usar o PARI de duas maneiras diferentes:

1. como uma biblioteca, que pode ser chamada de um aplicativo de linguagem de nível superior (por exemplo escrito em C, C ++, Pascal ou Fortran);
2. como uma sofisticada calculadora programável, denominada GP, que contém a maioria das instruções de controle de uma linguagem padrão como C.

A Tabela 1 apresenta os comandos construídos para as etapas adotadas no cálculo da aplicação.

No algoritmo descrito na primeira linha da Tabela 1 o comando $znorder(Mod(b,x))$ determina ordem de b em relação a x, ou seja, o menor expoente inteiro k de b tal que x divide b^k-1. A linha dois da Tabela 1 apresenta um código onde foi um criado um laço que percorre para cada valor de m no intervalo de 0 até $\pi(b,x)-1$, os valores de n percorrem o intervalo de 0 até $\pi(d,x)-1$. A velocidade do processo de determinar essas soluções pelo algoritmo dependerá de encontrar um primo p pequeno e especialmente encontrar um pequeno π(b, p) com grandes fatores comuns (STYER, 1993, p. 814).

4. RESULTADOS

Executou-se de forma limitada um conjunto de avaliações sobre as aplicações PARI e a biblioteca GiNaC, de modo a obter o desempenho relativo de cada um destes programas. Essas avaliações foram efetuadas através de um conjunto padrão de testes, mais precisamente o algoritmo de Styer, com os resultados obtidos, realizamos uma análise que nos permitiu uma comparação entre as mesmas.

Tabela 1 – Comandos construídos.

$\pi(b,x)$	r=znorder(Mod(b,x)) e s=znorder(Mod(d,x))
$a\,b^x = c\,d^y + e \pmod x$	for(m=0,s-1,for(n=0,r-1,if(a*Mod(b,x)^m==c*Mod(d,x)^n+e,print([m,n]))); [s,r]

Fonte: Elaborado pelo autor.

Como descrito na secção 2, determinou-se as soluções para a equação diofantina $ab^x = cd^y + e$ para valores de $a \leq 50$, $c \leq 50$ e $|e| \leq 1000$. No método considerado pode ser determinado, de fato, as soluções da equação $ab^x \equiv cd^y + e \pmod{p}$. Note que se a equação diofantina $ab^x = cd^y + e$ dada tem uma solução inicial (x_0, y_0) então, para todo inteiro $p > 1$ a equação $ab^x \equiv cd^y + e \pmod{p}$ tem ao menos uma solução, pois, temos que $ab^{x_0} = cd^{y_0} + e$ e p divide zero. Porém isto é irrelevante, pois, executou-se o teste para o primeiro passo do algoritmo de Styer, e a partir disso pôde ser testado todos os valores para x inteiro variando no intervalo de 1 até ordem de b módulo p e todo os valores para y inteiro variando no intervalo de 1 até ordem de d módulo p.

Para realização dos testes considerou-se a equação:

$$7.5^x = 11.13^y + 32 \qquad (01)$$

Note que a Equação 01 cumpre que $a \leq 50$, $c \leq 50$ e $|e| \leq 1000$ e tem como solução particular $(2,1)$. Em uma análise preliminar utilizando Styer mostrou que o primo 3571 é uma escolha adequada para o módulo, pois o $mdc(5,3571) = mdc(13,3571) = 1$ e utilizando o PARI obteve-se que $\pi(13,3571) = \pi(5,3571) = 1785$ o que implica pela discussão acima que o intervalo ser testado para m e n é de 0 a 1784, portanto se aplicará os testes sobre a equação:

$$7.5^m \equiv 11.13^n + 32 \pmod{3571} \qquad (02)$$

Fazendo uma segunda análise utilizando Styer mostrou-se que o primo 4871 também é uma escolha adequada para o módulo, pois o $mdc(5,4871) = mdc(13,4871) = 1$ e utilizando o PARI obtemos que $\pi(13,3571) = \pi(5,3571) = 2435$ o que implica pela discussão acima que o intervalo ser testado para m e n é de 0 a 2434, neste caso se aplicará os testes sobre a equação:

$$7.5^m \equiv 11.13^n + 32 \pmod{4871} \qquad (03)$$

Para equação 01 criou-se um algoritmo e este foi implementado no GiNaC e PARI, então determinou-se, as soluções da equação módulo um número primo qualquer. Na Figura 1, temos o prompt inicial de comando do PARI e o referido algoritmo.

Na Figura 2, temos a implementação feita na biblioteca GiNaC para os valores da equação $7.5^m \equiv 11.13^n + 32 \pmod{x}$ e o algoritmo descrito para a linguagem da biblioteca.

Na Figura 3, temos a execução e alguns valores da saída da implementação feita na biblioteca GiNaC para o algoritmo descrito na Figura 2.

Figura 1 – Tela inicial do PARI.

```
Reading GPRC: /etc/gprc ...Done.

                GP/PARI CALCULATOR Version 2.7.5 (released)
        amd64 running linux (x86-64/GMP-6.0.0 kernel) 64-bit version
     compiled: Nov 10 2015, gcc version 5.2.1 20151028 (Ubuntu 5.2.1-23ubuntu1)
                        threading engine: pthread
              (readline v6.3 enabled, extended help enabled)

                   Copyright (C) 2000-2015 The PARI Group

PARI/GP is free software, covered by the GNU General Public License, and comes WITHOUT ANY
WARRANTY WHATSOEVER.

Type ? for help, \q to quit.
Type ?12 for how to get moral (and possibly technical) support.

parisize = 8000000, primelimit = 500000
? x=3571
%1 = 3571
? #
   timer = 1 (on)
? r=znorder(Mod(13,x));s=znorder(Mod(5,x));for(m=0,s-1,for(n=0,r-1,if(7*Mod(5^m,x)==11*Mod(13^n,x
)+32,print([m,n]))));[s,r]
```

Fonte: Elaborado pelo autor (2017).

Figura 2 – Implementação na biblioteca GiNaC .

```
#include <iostream>
#include <ginac/ginac.h>
#include <time.h>
using namespace std;
using namespace GiNaC;

int main(){
/* Código  */
clock_t c2, c1;
float tempo;
int v1;

cin >> v1;

c1 = clock();
    const numeric f=5;
    const numeric g=13;
    ex e,d;

    for (int i = 1; i < 1785; ++i)
    {
        const numeric k=i;
        const numeric p1 = pow(f,k);
        e = mod(7*mod(p1,v1),v1);
        for (int t = 1; t < 1785; ++t)
        {
        const numeric j=t;
        const numeric p2 = pow(g,j);
        d= mod(11*mod(p2,v1)+32,v1);

            if (e == d)
            {
              cout << "[" << k << "," << j << "]" << endl;
            }
        }
    }

c2 = clock();
tempo = (c2 - c1)*1000/CLOCKS_PER_SEC;
cout << tempo << endl;
return 0;
}
```

Fonte: Elaborado pelo autor (2017).

Figura 3 – Compilação e Execução do código na biblioteca GiNaC .

```
┌─[[Bib]]--[GiNaC]:~$
└─>>g++ cod_gin.cpp -o cod_gin -lcln -lginac
┌─[[Bib]]--[GiNaC]:~$
└─>>./cod_gin
3571
[2,1]
[3,1033]
[4,1169]
[5,753]
[7,1436]
[8,888]
[10,1599]
[16,340]
[21,272]
[23,708]
[25,1592]
[26,1765]
[29,1667]
[30,52]
[31,1550]
```

Fonte: Elaborado pelo autor (2017).

A fim de termos um grau de otimização algébrica fez-se também um teste de expansão do seguinte polinômio $(x+y)^{500}$

Inicialmente, os resultados encontrados pelo algoritmo no PARI foram comparados com os resultados encontrados pelo algoritmo na biblioteca GiNaC, com o objetivo de validação dos resultados do algoritmo. Os tempos de execução também foram comparados. Em seguida, os algoritmos PARI e GiNaC são comparados quanto aos resultados e ao tempo de execução.

Todos os experimentos deste trabalho foram executados dez vezes e os resultados apresentados correspondem à média dos tempos de execução, sendo muito pequenos os valores dos desvios padrão.

A Tabela 2 mostra os resultados dos testes.

Tabela 2 – Os códigos a seguir foram testados em um computador Intel(R) Core(TM) i5, 1.80GHz, 6094 MB em ambiente LINUX.

Função	GiNaC	PARI
$(x+y)^{500}$	0,017	0,452
$Mod(7*5^{x}=11*13^{y}+32,3571)$	17,188	9,496
$Mod(7*5^{x}=11*13^{y}+32,3733)$	14,704	8,868

Fonte: Elaborada pelo autor.

5. CONCLUSÃO

O estudo feito evidenciou a eficiência tanto do GiNaC quanto do PARI sendo que este último mostrou-se muito mais eficiente para cálculos elaborados, ou seja que exigem estruturas mais complexas, e levando em conta ser o PARI uma CAS, logo muito mais geral, contendo assim muito mais recursos. Assim, não basta somente verificar se o desempenho de uma ferramenta é melhor que o da outra, mas também considerar a qualidade das ferramentas avaliadas. Destaca-se, portanto, o grande potencial do GiNaC e do PARI na exploração e levantamentos de informações sobre vários aspectos de contorno dos problemas que envolvem equações diofantinas que necessitam de maior capacidade de processamento. Podemos concluir que apesar do GiNaC ter sido concebido para fazer cálculos algébricos mais acoplados a linguagem C++, o PARI mostrou-se muito mais rápido e ainda com a vantagem de ter uma maior facilidade de uso e um conjunto muito grande de funções.

REFERÊNCIAS BIBLIOGRÁFICAS

GINAC IS NOT A CAS. Disponível em: <https://www.ginac.de/> Acesso em: 15 de set. 2017.

MOREIRA, C, G e et al. **Teoria dos números – Um passeio com primos e outros números familiares pelo mundo inteiro**, Coleção Projeto Euclides – IMPA: Rio de Janeiro, 2013.

PARI/GP DEVELOPMENT. Disponível em: <https://pari.math.u-bordeaux.fr/> Acesso em: 20 de set. 2017.

GINSBURG, D.; GROOSE, B.; TAYLOR, J.; VERNESCU, B. **The History of the Calculus and the Development of Computer Algebra Systems.** Disponível em: <http://www.math.wpi.edu/IQP/BVCalcHist/index.html> Acesso em: 21 de set. 2017.

SANTOS, J. O. **Introdução à Teoria dos Números.** Rio de Janeiro: IMPA, 2010.

STYER, R. **Small two-variable exponential Diophantine equations. Mathematics of Computation**, 1993.