Relatório Final PIBIC - CNPq

Sobre Funções que Geram Números Primos

Período de vigência: Agosto de 2018 a Julho de 2019

Orientador:
Prof. Dr. Fernando Eduardo Torres Orihuela
UNICAMP - IMECC - Dep. Matemática

Aluno:
Antonio Fornari
RA 166628
Granduando em Matemática Aplicada

Instituto de Matemática, Estatística e Computação Científica
Universidade Estadual de Campinas
Campinas, São Paulo, Brasil – 13083-859

12 de Agosto de 2019

1 Introdução

Este projeto tinha como grande objetivo introduzir o autor ao ambiente de pesquisa, garantindo a ele experiência e conhecimento útil para que possa prosseguir pesquisando na área de Teoria de Números ou em áreas que se utilizem da Teoria de Números. Os tópicos selecionados para serem estudados foram:

- Propriedades Elementares dos Números Primos
- Funções Geradoras de Primos
- Teoremas e Conjecturas ligados a números primos, como o Teorema de Dirichlet sobre progressões aritméticas e a Conjectura de Bunyakovsky
- Criptografia

A ideia deste projeto de pesquisa nasceu da leitura de *Existem funções que geram os primos?*[4], e ele como um todo gira em torno da busca por funções, em especial as polinomiais, capazes de gerar primos com uma alta frequência, e por possíveis aplicações destas. Vários textos, que se encontram na parte de Referências, foram usados e ainda mais programas foram escritos durante o último ano com este propósito, alguns presentes neste relatório.

Durante o período de agosto de 2018 a janeiro de 2019, resumidamente, o autor estudou as propriedades elementares dos números primos e as aplicou na análise de Polinômios Geradores de Primos. Vários resultados já obtidos anteriormente por outros pesquisadores, e que podem ser encontrados nos livros de Ribenboim [5, 6, 7], foram verificados e alguns foram até mesmo atualizados, como o de polinômio quadrático mônico com maior geração de primos no intervalo $\mathbb{I} = [0, 1000] \cap \mathbb{Z}$ do domínio.

Na segunda metade do período de vigência deste projeto, de fevereiro de 2019 a julho de 2019, o foco foi o estudo do RSA e o aperfeiçoamento de algoritmos feitos na primeira metade.

Todo o cronograma proposto foi cumprido.

2 Materiais e Métodos

Os materiais utilizados durante a pesquisa foram os livros que se encontram nas Referências Bibliográficas e os programas escritos em C pelo autor que se encontram abaixo, todos utilizam as bibliotecas stdio.h, stdlib.h, math.h e time.h e foram compilados pelo GCC 8.3.

2.1 PrimeLister

O programa abaixo, PrimeLister, cria um vetor dinâmico já contendo 2 e 3 como primeiros elementos e vai se atualizando com cada novo número que não tem divisores já no vetor. O vetor então é transformado em um arquivo list.txt contendo um número MAX de primos a ser usado na maioria dos demais programas.

```
#define MAX 1000

int prime(int number, int* v) {
    int raiz=(int)sqrt(number);
    int k=0;

    while(v[k]<=raiz){
        if(number%v[k++]==0) return 0;
    }

    return 1;
}

int main(){
int counter=2;
    FILE *list;
    list=fopen("list.txt", "w");

    int *v;
    if(MAX>=1000000){
        v=malloc((int)(MAX/10)*sizeof(int));
        if(v==NULL){
            printf("malloc has failed\n");
            system("PAUSE");
            return 1;
        }
    }
    else{
        v=malloc(MAX*sizeof(int));
        if(v==NULL){
            printf("malloc has failed\n");
            system("PAUSE");
            return 1;
        }
    }

    fprintf(list, "2\n3\n");
    v[0]=2;v[1]=3;

   {int i,j;
    for(i=6, j=2; i<MAX; i+=6){
        if(prime(i-1, v)){
            fprintf(list, "%d\n",(i-1));counter++;
```

```
            v[j++]=i-1;
        }
        if(prime(i+1, v)){
            fprintf(list, "%d\n", (i+1));counter++;
            v[j++]=i+1;
        }
    }
   }

    fclose(list);
    printf("%d",counter);
    return 0;
}
```

2.2 PolyFinder1

O programa abaixo, PolyFinder1, foi umas das primeiras versões a ser usada na busca por polinômios que gerassem bastantes primos no intervalo [0,MAX]. Ele ainda tem uma função "prime"que testa a primalidade de cada possível primo, feita com base na propriedade de que todo número primo maior que 3 é congruente a $\pm 1 \mod 6$.

O código é bastante simples: Ele cria um vetor com três elementos, representando coeficientes dos polinômios a serem testados, varia os coeficientes de x^0 e x^1 dentro da coroa circular de raio externo LIM e raio interno LIM2 (métrica do supremo), varia o coeficiente de x^2 de 1 a 2, e testa a primalidade de cada número gerado pelos polinômios no intervalo [0,MAX].

O programa não testa alguns polinômios considerados ruins, aqueles cujo termo independente é um numero composto, e salva em um arquivo de texto PolyList.txt apenas os polinômios que gerarem mais que MINPRIME primos. Também é gravado no arquivo o tempo que a busca demorou.

```
#define LIM 10001
#define LIM2 0
#define MINPRIME 650
#define MAX 1000


int prime(int number) {
    if(number<0){
            number=-number;
    }
    if(number==1){
```

```
            return 0;
    }
    else if (number<=3){
        return 1;
    }
    else if (number%2==0 || number%3==0){
        return 0;
    }
    else{
        unsigned int i,sq;
        sq=sqrt(number);
        for (i=5; i<=sq; i+=6) {
            if (number % i == 0 || number%(i + 2) == 0)
                return 0;
        }
        return 1;
    }
}

int main(){

    clock_t start = clock();
    int poly[3],i,j,x,y,counter;
    FILE *PolyList;
    PolyList=fopen("PolyList.txt","w");

    for(poly[2]=1;poly[2]<=2;poly[2]++){
            for(poly[1]=-LIM;poly[1]<=LIM;poly[1]++){
                    for(poly[0]=-LIM;poly[0]<=LIM;poly[0]+=2){
                       if(poly[0]>-LIM2 && poly[0]<LIM2 && poly[1]>-LIM2 &&
                        ↪  poly[1]<LIM2){
                                poly[0]=LIM2;
                                continue;
                             }
                         if(prime(poly[0])==0 && poly[0]!=1)continue;

                         for(counter=0,x=0;x<=MAX;x++){
                                y=poly[0]+(x*poly[1])+(x*x*poly[2]);
                                if(prime(y))counter++;
                         }
                         if(counter>MINPRIME){
                         fprintf(PolyList,"(%d)+(%dx)+(%dx^2) :
                          ↪  %d\n",poly[0],poly[1],poly[2],counter);
                         }
                    }
            }
    }
    clock_t stop = clock();
    float elapsed = (float)(stop - start) / CLOCKS_PER_SEC;
    fprintf(PolyList, "TIME: %f s", elapsed);
    fclose(PolyList);
```

```
    return 0;
}
```

2.3 PolyFinder2

O programa abaixo, PolyFinder2, é uma versão mais eficiente do programa acima. O grande diferencial é se utilizar de um arquivo gerado pelo PrimeLister para criar um vetor dinâmico que auxiliará nos testes de primalidade. Deve-se antes saber o tamanho (tamanho_lista) desse vetor a ser criado.

Esta versão é em média 35% mais rápida que a anterior de acordo com as buscas realizadas.

```
#define LIM 10001
#define LIM2 0
#define MINPRIME 650
#define tamanho_lista 350


int prime(int number, int* v) {

    if(number<0)number=-number;
    if(number==1)return 0;

    int raiz=sqrt(number);
    for(int i=0;v[i]<=raiz;i++){
        if(number%v[i]==0){
            return 0;
        }
    }

    return 1;
}

int main(){

    clock_t start = clock();
    int poly[3],x,y,counter;

    FILE *PolyList;
    PolyList=fopen("PolyList.txt","w");
    FILE *list;
    list=fopen("list.txt","r");

    int* v;
    v=malloc(sizeof(int)*tamanho_lista);
    if(v==NULL){
        printf("malloc falhou\n");
        return 1;
        }
```

```
    for(int i=0;i<tamanho_lista;i++){
            fscanf(list, "%d", &v[i]);
    }
    fclose(list);

    for(poly[2]=2;poly[2]<=2;poly[2]++){
        for(poly[1]=-LIM;poly[1]<=LIM;poly[1]++){
            for(poly[0]=-LIM;poly[0]<=LIM;poly[0]+=2){
                if(poly[0]>-LIM2&&poly[0]<LIM2){
                    poly[0]=LIM2;
                    continue;}

                if(prime(poly[0], v)==0 && poly[0]!=1)continue;

                for(counter=0,x=0;x<=1000;x++){
                    y=poly[0]+(x*poly[1])+(x*x*poly[2]);
                    if(prime(y, v))counter++;
                }
                if(counter>MINPRIME)fprintf(PolyList, "(%d)+(%dx)+(%dx^2) :
                 ↪  %d\n", poly[0], poly[1], poly[2], counter);

            }
        }
    }

    free(v);
    clock_t stop = clock();
    float elapsed = (float)(stop - start) / CLOCKS_PER_SEC;
    fprintf(PolyList, "TIME: %f s", elapsed);
    fclose(PolyList);
    return 0;
}
```

2.4 PrimeCounter

Há também o PrimeCounter, uma variante mais simplista dos programas acima, que conta quantos são os primos no conjunto imagem $Im = \{P(0), P(1), ..., P(MAX)\}$ de um polinômio quadrático $P(n) = ax^2 + bx + c$. Este não passou por otimizações recentes pois não foi necessário.

```
#define MAX 10000
#define a 1
#define b -1
#define c 41

int main(){
    int primality=1,counter;
    long long int y,n,i;

    for(n=1, counter=0;n<=MAX;primality=1,n++){
        y = (a*n*n + b*n + c);
```

```
    if(y%2==0)primality=0;

    for(i=3;primality==1 && i<=sqrt(y);i+=2){
            if(y%i==0)primality=0;
            }
    if(primality==1)counter++;

    }

printf("There were %d prime values for P(n) from n=0 to n=%d",counter,MAX);

return 0;
}
```

3 Resultados

3.1 Polinômios Geradores de Primos

Os principais resultados deste projeto, sem dúvidas, foram obtidos durante o recorrente estudo dos Polinômios Geradores de Primos. Com o apoio de *"Introdução à Teoria dos Números"*[8], que foi extensamente lido no primeiro mês de vigência deste projeto, foi possível começar a entender mais sobre polinômios como o "Polinômio de Euler", $P(n) = n^2 - n + 41$, que produz uma sequência de 40 primos distintos para $1 < n < 40$ e continua a produz primos com alta frequência mesmo para valores grandes de n.

Teorema 1. *Se* $p(x) \in \mathbb{Z}[x]$ *e* $\{q, d, r\} \in \mathbb{Z}$*, então* $p(qd+r) \equiv p(r) \mod d$.

Analisando os polinômios $p_q(n) = n^2 - n + q$ e o Teorema 1, é fácil ver que $p_q(q)$ é necessariamente um número composto (ou 1), e que, portanto, uma sequência de primos produzida por $p_q(n)$ tem no máximo comprimento $l = q - 1$.

É natural que se deseje saber quais os polinômios com essa propriedade, afinal já se sabia que $p_{41}(n)$, o polinômio de Euler, assim se comportava. Com a ajuda de *"Prime-Producing Quadratics"* [3] foi estudada a caracterização dos polinômios da família $p_q(n) = n^2 - n + q$ com máxima produção ininterrupta de primos, ou seja, com $\{p_q(1), p_q(2), ..., p_q(q-1)\}$ todos primos.

Teorema 2. [3] $p_q(n) = n^2 - n + q$ *tem máxima produção de primos ininterrupta se, e somente se,* $q \in \{1, 2, 3, 5, 11, 17, 41\}$

O livro *"Números Primos, Velhos Mistérios e Novos Recordes"*[7] traz várias informações sobre a produção de primos de $p_{41}(n)$, como $\pi^*_{n^2+n+41}(10^7) = 2208197$. Abaixo definições úteis.

Definição 1. *A função* $\pi(n) : \mathbb{N} \to \mathbb{N}_0$ *retorna o número de primos em* $I_n = \{1, 2, ..., n\}$.

Definição 2. *A função* $\pi^*_{p(x)}(n) : \mathbb{Z}[x] \times \mathbb{N} \to \mathbb{N}_0$ *retorna o número de primos no conjunto* $Im_n(|p(x)|) := \{|p(1)|, |p(2)|, \ldots, |p(n)|\}$.

Com o objetivo de verificar essas informações e adquirir algumas mais foi feito o PrimeCounter (2.4). Algumas das informações obtidas:

Tabela 1: Número de primos em I_n e em $Im^n_1(|p_q(x)|)$

n	$\pi(n)$	$\pi^*_{p_1(x)}(n)$	$\pi^*_{p_3(x)}(n)$	$\pi^*_{p_5(x)}(n)$	$\pi^*_{p_{11}(x)}(n)$	$\pi^*_{p_{17}(x)}(n)$	$\pi^*_{p_{41}(x)}(n)$
10^2	25	32	14	30	48	60	86
10^3	168	189	93	165	288	365	581
10^4	1229	1410	629	1208	2057	2627	4149
10^5	9592	10751	4899	9086	15661	20127	31985
10^6	78498	88118	40037	74058	128171	164220	261081
10^7	664579	745582	339824	626458	1083223	1388247	2208197

Tabela 2: Proporção $\pi^*_{p_q(x)}(n)/\pi(n)$ de primos, arredondada nos milésimos

n	$q = 1$	$q = 2$	$q = 3$	$q = 5$	$q = 11$	$q = 17$	$q = 41$
10^2	1.280	0.040	0.560	1.200	1.920	2.400	3.440
10^3	1.125	0.006	0.554	0.982	1.714	2.173	3.458
10^4	1.147	0.001	0.512	0.983	1.674	2.138	3.376
10^5	1.121	0.000	0.511	0.947	1.633	2.098	3.335
10^6	1.123	0.000	0.510	0.943	1.633	2.092	3.326
10^7	1.122	0.000	0.511	0.943	1.630	2.089	3.323

Para entender esse comportamento bastam os teoremas 1 e 2, que nos dizem que os números gerados por $p_q(n)$ não são divisíveis por nenhum primo menor que q.

Com o fim de seguir adiante e descobrir novos formatos de Polinômios Geradores de Primos foi feito o PolyFinder1 (2.2), que encontrou $p(n) = n^2 - 839n + 6047$, um polinômio tal que $\pi^*_{n^2-839n+6047}(10^3) = 693$. Segundo Ribenboim em seus livros *"The Little Book of Bigger Primes"*[5] e *"Die Welt der Primzahlen, Geheimnisse und Rekorde"*[6] o recorde de produção

de primos para um polinômio quadrático mônico no intervalor [0,1000] é de $\pi^*_{n^2-999n+277441}(10^3) = 669$.

Na tentativa de encontrar polinômios ainda melhores, o PolyFinder1 foi aperfeiçoado e tornou-se PolyFinder2 (2.3). A partir dele, descobriu-se

$$\pi^*_{n^2-997n+78569}(10^3) = 704$$

. Vale notar que:

$$n^2 - 997n + 78569 = (n - \frac{997}{2})^2 - \frac{679733}{4}$$

$$n^2 - 839n + 6047 = (n - \frac{839}{2})^2 - \frac{679733}{4}$$

Os dois são o mesmo polinômio, deslocado no eixo horizontal.

Baseando-se nisso foram feitos outros programas similares aos PolyFinders, que tentavam explorar a simetria vertical das parábolas para encontrar melhores geradores de primos. A ideia era encontrar um ótimo Polinômio Gerador quadrático para o intervalo [0,500], que tivesse vértice próximo à origem, para que então fosse deslocado 500 unidades para a direita e se tornasse um ótimo Polinômio Gerador quadrático para o intervalo [0,1000]. A busca no intervalo [0,500] é menos custosa que no intervalo [0,1000], afinal. Esta busca não encontrou novos bons geradores, mas encontrou vários já conhecidos, como o $n^2 - 997n + 78569$.

Os livros de Ribenboim também falam sobre polinômios recordistas não mônicos. A maior produção de primos no intervalo [0,1000] por um polinômio se deve a $\pi_{2x^2-1584x+98621} = 706$. O PolyFinder2 o pôde encontrar em suas buscas.

3.2 Criptografia RSA

O autor estudou o funcionamento da criptografia RSA com a leitura do livro *"Números Inteiros e Criptografia RSA"* [9]. O conteúdo visto será mostrado através de seu uso:

A chave pública será

$$(n, e) = (6028859, 5)$$

Sabe-se que $n = 997 * 6047$, números que já apareceram como coeficientes de polinômios relevantes neste relatório. Além do mais, $\phi(n) = \phi(997)\phi(6047) = 996 * 6046 = 6021816$ e $\gcd(\phi(n), e) = \gcd(6021816, 5) = 1$.

A tabela de conversão a ser usada será

Tabela 3: Tabela de Conversão

A	B	C	D	...	X	Y	Z
10	11	12	12	...	33	34	35

E a mensagem a ser transmitida será IMECC.

No processo de pré-codificação, IMECC torna-se 1822141212 e então 18221 41212, pois precisa-se de blocos de informação menores que $n = 6028859$.

Na codificação:

$$18221^5 \equiv 1309416 \mod 6028859$$

$$41212^5 \equiv 435471 \mod 6028859$$

Mensagem codificada: 1309416 435471.

Para decodificar:

$$\phi(\phi(n)) = \phi(6021816) = \phi(2^3 * 3 * 83 * 3023) = 4 * 2 * 82 * 3022 = 1982432$$

Disso temos que

$$5^{-1} \equiv 5^{\phi(\phi(n))-1} \equiv 5^{1982431} \equiv 4817453 \mod 6021816$$

.

Para decodificar:

$$1309416^{4817453} \equiv 18221^{5*4817453} \equiv 18221^{k*\phi(n)+1} \equiv 18221 \mod 6028859$$

$$435471^{4817453} \equiv 41212^{5*4817453} \equiv 41212^{k*\phi(n)+1} \equiv 41212 \mod 6028859$$

graças ao Teorema de Euler.

O processo de codificação sempre pode ser feito por qualquer um que conheça a chave (n, e) e tenha um computador suficiente para fazer as contas, mas a decodificação, quando n bem escolhido, só é possível para quem conhece a fatoração de n, pois permite calcular $\phi(n)$.

3.3 Demais resultados

Foram estudados vários teoremas importantes para a Teoria de Números, como o Teorema de Dirichlet sobre Progressões Aritméticas, o Teorema de Korselt e o Teorema de Green-Tao; e conjecturas como a de Bunyakovsky e o décimo problema de Hilbert, todos com o objetivo de conhecê-los e formar repertório matemático e histórico.

4 Conclusões

O aluno, durante este projeto, como esperado, adquiriu experiência e conhecimento na área de Teoria de Números: Ao estudar os Polinômios Geradores de Primos, a Criptografia RSA e demais assuntos relacionados ganhou boa base para prosseguir na área e ganhou cultura matemática.

Vários algoritmos foram desenvolvidos para ajudar nesta pesquisa, cada um tendo passado por diversos aprimoramentos, e certamente todos serão aprimorados mais diversas vezes quando necessário. Tais algoritmos encontraram o polinômio $p(n) = n^2 - 997n + 78569$, possivelmente o polinômio quadrático mônico com maior densidade de primos no intervalo [0,1000] até o momento. A busca por polinômios mônicos que gerem mais primos em [0,1000] que este $p(n)$ não é fácil, pois teria-se que buscar entre polinômios com coeficientes cada vez maiores, que geram números também cada vez maiores. Além de aumentar o custo computacional, sabe-se que a probabilidade de encontrar primos fica cada vez menor à medida se busca entre números maiores.

5 Perspectivas de continuidade

O autor tem bastante interesse em continuar a pesquisar sobre Criptografia, em aprender mais sobre novos métodos criptográficos e poder fazer comparações com o RSA. Um novo projeto com objetivo de estudar Criptografia de Curva Elíptica estava previsto para o período de agosto de 2019 a julho de 2020, foi selecionado em primeira chamada, mas o autor não o pôde aceitar pois estará estudando fora do país com uma bolsa provida pela UNICAMP no edital $n^o54/2019$ da DERI, logo dará seguimento a estes estudos sozinho no momento.

6 Outras informações

Foram realizadas pelo autor deste relatório duas palestras essencialmente sobre Funções Geradoras de Primos. A primeira, em 14 de setembro de 2018 no ciclo de palestras de Curvas Algébricas e temas afins no IMECC, sob o nome "Algumas Propriedades da Sequência de Números Primos". A segunda, com título "Polinômios Geradores de Primos", em 22 de outubro de 2018 pelo DivulgaMat IMECC, um ciclo de palestras voltado para alunos da universidade.

7 Agradecimentos

Agradeço imensamente ao meu orientador, o Professor Doutor Fernando Eduardo Torres Orihuela, pela disposição e pela celeridade com as quais sempre me ajudou nas adversidades da pesquisa, e também por ter me apresentado formalmente à Teoria de Números.

Agradeço à UNICAMP e ao CNPq pela oportunidade de ter realizado este projeto de pesquisa, que certamente me tornou muito mais capaz.

Por fim, agradeço também aos meu amigos, que estiveram por perto para discutir os assuntos aqui tratados, e foram responsáveis pela minha motivação em alta mesmo nos momentos de maior stress.

Referências

[1] APOSTOL, T.M. "Introduction to Analytic Number Theory", Undergraduate Texts in Mathematics, Springer, 1976.

[2] KERNIGHAN, B.W; RITCHIE, D. "C Programming Language", Software Series, Prentice Hall, 1988.

[3] MOLLIN, R.A. "Prime-Producing Quadratics", The American Mathematical Monthly, Vol. 104, No. 6 (Jun. - Jul., 1997), pp. 529-544
Disponível em: https://www.jstor.org/stable/2975080
Acesso em: 12 ago. 2019

[4] RIBENBOIM, P. "Existem funções que geram os primos?", Matemática Universitaria **12** (1993), 1–12.

[5] RIBENBOIM, P. "The Little Book of Bigger Primes", Springer, 2004.

[6] RIBENBOIM, P. "Die Welt der Primzahlen, Geheimnisse und Rekorde", Springer, 2011.

[7] RIBENBOIM, P. "Números Primos, Velhos Mistérios e Novos Recordes", Coleção Matemática Universitária, IMPA, 2014.

[8] SANTOS, J.P.O. "Introdução à Teoria dos Números", Coleção Matemática Universitária, IMPA, 1998.

[9] COUTINHO, S. C. "Números Inteiros e Criptografia RSA", 2ª edição, IMPA, 2014.

[10] TAO, T.; GREEN, B. *"The primes contain arbitrarily long arithmetic progressions"*
Disponível em: https://arxiv.org/abs/math/0404188
Acesso em: 12 ago. 2019

[11] TAO, T.; ZIEGLER, T. *"Polynomial patterns in the primes"*
Disponível em: https://arxiv.org/abs/1603.07817
Acesso em: 12 ago. 2019