Block Notes Matematico

Forme generatrici di numeri primi

ing. Rosario Turco, prof. Maria Colonnese

Sommario

Nel seguito vengono esaminate varie forme generatrici di numeri primi. Quella della ricerca delle formule generatrici è sempre stata, nei secoli, un tema di sfida, analoga alla sfida di trovare numeri primi con gran numero di cifre. In passato si supponeva che esistessero solo formule lineari per ottenere infiniti numeri primi, mentre gli autori mostrano che è possibile escogitare formule di vario tipo.

Email

mailto:rosario_turco@virgilio.it

INDEX

Dirichlet e le forme generatrici lineari

Nel 1937 Dirichlet in suo lavoro mostrò che: "Se a,b sono due numeri naturali, primi fra loro, allora la funzione lineare F(x) = ax + b con MCD(a,b)=1, contiene infiniti numeri primi, quando l'intero x descrive l'insieme dei numeri naturali".

Durante lo studio della Teoria dei Numeri, si matura subito l'idea che oltre alla forma base generatrice 2k+1 esistono altre forme generatrici (**FGP)**, che generano numeri primi, se si escludono i composti.

La "forma generatrice lineare di numeri primi generalizzata" (**FGLPG)** è del tipo:

(1) $x = m * k + c$

Dove: k, c ϵ N, MCD(c,m)=1 (la costante c è coprima con m), m è un numero primo o un prodotto di numeri primi (esempio 2, 2*3=6, 2*3*5=30 o 2*3*5*7=210 o 2*3*5*7*11=2310 etc.) ed x deve risultare numero primo. Qui la costante c può essere anche negativa (esempio -1).

La **FBG** : x = 2k +1 ad esempio, rientra nella (1), cioè è un caso particolare di FGLPG con m=2, c=1 ed è coprimo con m. La FBG genera tutti i numeri dispari, tra cui si selezionano i numeri primi.

Particolari FGLPG sono, ad esempio, le FGP seguenti:
(2) x=6*k + 1 (Se c=-1 vale anche x=6*k-1)
(3) x=6*k + 5

Vantaggi e velocità di generazione
Il vantaggio di considerare le FGP o le FGLPG rispetto alla FBG è che essi generano una quantità molto minore di numeri interi dispari (da gestire); ad esempio la (2) e la (3) gestiscono 1/3 di tutti gli interi e con una velocità superiore alla FBG. L'uso poi combinato delle due diverse FGP dà facilmente la copertura (l'unione) di tutti i numeri primi ad eccezione del 2,3, che già sappiamo come numeri primi.

Il concetto di velocità, ad esempio, può servire a individuare numeri primi molto grandi più rapidamente.

Esempi di generazione di numeri primi con WinHugs
La (2), (3) si possono esprimere con WinHugs nel seguente modo:

(2a) fgp1 n = [x | k<-[0..p], x<-[1..p], x==6*k+1, isprime x]
se si pone n=1000 (solo i primi fino a n=1000) si ottengono i seguenti numeri primi:

```
Main> fgp1 1000
[7,13,19,31,37,43,61,67,73,79,97,103,109,127,139,151,157,163,181,193,199,211,223
,229,241,271,277,283,307,313,331,337,349,367,373,379,397,409,421,433,439,457,463
,487,499,523,541,547,571,577,601,607,613,619,631,643,661,673,691,709,727,733,739
,751,757,769,787,811,823,829,853,859,877,883,907,919,937,967,991,997]
```

(3a) fgp5 n = [x | k<-[0..p], x<-[1..p], x==6*k+5, isprime x]

```
Main> fgp5 1000
[5,11,17,23,29,41,47,53,59,71,83,89,101,107,113,131,137,149,167,173,179,191,197,
227,233,239,251,257,263,269,281,293,311,317,347,353,359,383,389,401,419,431,443,
449,461,467,479,491,503,509,521,557,563,569,587,593,599,617,641,647,653,659,677,
683,701,719,743,761,773,797,809,821,827,839,857,863,881,887,911,929,941,947,953,
971,977,983]
```

Si osserva che l'insieme P dei numeri primi è dato dall'unione dei due insiemi FG1 e FG2, prodotti dalla (2a) e (3a); in altri termini la (2) e la (3) sono già sufficienti a dare tutti i numeri primi fino a n=1000. In più sono attivabili in parallelo facilmente; non occorre suddividere nessun intervallo. Ogni primo è presente in una sola lista.

Spiegazione
Quali sono i motivi per cui vanno bene la (1) o la (2)? E' possibile spiegarlo con l'aritmetica modulare; difatti se si divide un intero positivo n per 6 si ottiene come resto un valore tra 0 .. 5. Quindi n viene a cadere, per forza, in una delle seguenti successioni: 6k, 6k+1, 6k+2, 6k+3, 6k+4, 6k+5. In altri termini operiamo in modulo 6.

Ad esempio: n=12 n/6=2 resto=0 -> n=6*2 (6k)
n=13 n/6=2 resto=1 -> n=6*2+1 (6k+1)
etc.

Però se n è primo (es: n=13) il resto non potrà essere mai 0,2,3,4; altrimenti n sarebbe divisibile per 6, 2, 3, 4. Il resto sarà 0 solo se divisibile per sé stesso (difatti è primo).

Se dalla (1) si sceglie m=30 (operiamo in modulo 30), allora esistono solo 8 classi con m=30 che danno tutti i numeri primi (esclusi 2,3,5): 30k + 1, 30k + 7, 30k + 11, 30k + 13, 30k + 17, 30k + 19, 30k + 23, 30k + 29

La cui implementazione con WinHugs può essere del tipo:
fgp30-1 n = [x | k<-[0..p], x<-[1..p], x==30*k+1, isprime x]
fgp30-7 n = [x | k<-[0..p], x<-[1..p], x==30*k+7, isprime x]
fgp30-11 n = [x | k<-[0..p], x<-[1..p], x==30*k+11, isprime x]
fgp30-13 n = [x | k<-[0..p], x<-[1..p], x==30*k+13, isprime x]
fgp30-17 n = [x | k<-[0..p], x<-[1..p], x==30*k+17, isprime x]
fgp30-19 n = [x | k<-[0..p], x<-[1..p], x==30*k+19, isprime x]
fgp30-23 n = [x | k<-[0..p], x<-[1..p], x==30*k+23, isprime x]
fgp30-29 n = [x | k<-[0..p], x<-[1..p], x==30*k+29, isprime x]

I risultati ottenibili sono quindi:

```
Main> fgp30-1 1000
[31,61,151,181,211,241,271,331,421,541,571,601,631,661,691,751,811,991]
```

```
Main>fgp30-7 1000
[7,37,67,97,127,157,277,307,337,367,397,457,487,547,577,607,727,757,787,877,907,
937,967,997]

Main>fgp30-11 1000
[11,41,71,101,131,191,251,281,311,401,431,461,491,521,641,701,761,821,881,911,94
1,971]

Main> fgp30-13 1000
[13,43,73,103,163,193,223,283,313,373,433,463,523,613,643,673,733,823,853,883]

Main> fgp30-17 1000
[17,47,107,137,167,197,227,257,317,347,467,557,587,617,647,677,797,827,857,887,9
47,977]

Main> fgp30-19 1000
[19,79,109,139,199,229,349,379,409,439,499,619,709,739,769,829,859,919]

Main> fgp30-23 1000
[23,53,83,113,173,233,263,293,353,383,443,503,563,593,653,683,743,773,863,953,98
3]

Main> fgp30-29 1000
[29,59,89,149,179,239,269,359,389,419,449,479,509,569,599,659,719,809,839,929]
```

L'unione di tutte le otto liste contiene solo i 4/ 15 di tutti i numeri interi (la frequenza è maggiore) e dà l'insieme di tutti i numeri primi fino a n=1000. Inoltre le generazioni sono attivabili facilmente in parallelo.

Forme generatrici lineari per numeri gemelli
Se ci si basa sulla (1) e si pone m=30, allora le possibili coppie di numeri gemelli (p,q) si possono ottenere dall'accoppiamento di sole 6 forme generatrici del tipo:

(30k+1,	30(k+1)+1)	(vedi fgp30-1 e fgp30-29)
(30k+11,	30k +13)	(vedi fgp30-11 e fgp30-13)
(30k+17,	30k+19)	(vedi fgp30-17 e fgp30-19)

Nelle liste precedenti si trovano in rosso i numeri appartenenti ad una coppia di gemelli (numeri primi a distanza 2). Si nota ad esempio, proprio perché si ragiona in modulo 30, che le coppie di gemelli sono ad esempio tra le classi fgp30-1 e fgp30-29 etc.

Forme generatrici lineari per numeri di Polignac
Se anzicchè considerare la distanza 2 come nei numeri gemelli, si considera una distanza multiplo di 2, si otterranno i numeri primi di Polignac (Vedi Appendice)

Forme generatrici lineari per numeri di Sophie Germain
A questa categoria appartengono anche i numeri primi di Sophie Germain, che in linguaggio funzionale sono generati con la seguente regola:

```
--
-- Coppia di numeri di Sophie Germain
--
sophie n = [(s,p)| p<-[2..n], s<-[2..n],s==2*p+1, isprime p, isprime s]
--
-- quantità di numeri di Sophie Germain fino a n
--
```

```
nsophie n = length (sophie n)
```

Forme generatrici esponenziali di numeri primi

Una particolare forma generatrice esponenziale di numeri primi (**FGEP)** è del tipo:

(4) $x = 2^k + 1$,

con x da selezionare tra i numeri primi e con kϵN.

La forma generalizzata **FGEPG è del tipo:**
(5) $x = m^k + c$

Dove: k ϵ N, x primo, con m primo, c è tale che MCD(c,m)=1, ad esempio **c = m -1**

Alcuni esempi: $x=3^k+2$, $x=5^k+4$, $x=7^k+6$, $x=11^k+10$, etc.

Supponiamo : $x=3^k+2$, con WinHugs è:
fgp3potk p = [x | k<-[0..p], x<-[1..3^k+2], x==3^k+2, isprime x]

Si ottiene la successione infinita di numeri primi:

```
Main> fgp3potk 10
[3,5,11,29,83,6563,59051]
```

Lanciare il programma con un valore maggiore di 10 richiederà un tempo di attesa che va crescendo esponenzialmente.

Forme generatrici fattoriali di numeri primi

Una forma generatrice fattoriale di numeri primi (**FGFIP)** è del tipo:
(6) x = k! +1

Su queste forme alcune idee sono legate al Teorema di Wilson.

Forme generatrici a parabola (euleriane e non)di numeri primi

La prima forma generatrice a parabola (**FGPP**) storicamente fu escogitata da Eulero:

(7) $f(x) = x^2 + x + 41$. Tale forma generatrice è denominata la **parabola di Eulero**.

Una forma generalizzata (**FGPPG**) è del tipo:
(8) $f(x) = x^2 + x + p$ con p primo (**parabole euleriane**).

A questa categoria appartengono anche i ***numeri primi di Landau***, generabili dalla forma:

(9) $f(x) = x^2+1$, con n pari ad eccezione di n=1 che produce lo stesso p=2.

La (7) e la (9) sono un caso particolare della (8). La (7) ed è anche caratterizzata dal fatto che la successione infinita di numeri primi generabile ha almeno 40 numeri primi consecutivi.

Tutte le altre forme euleriane, cioè che rispettano la (8), sono con un numero di primi consecutivi inferiori a 40.

Da citare sono anche le **"parabole non euleriane"**:

(10) $f(x) = 103\ x^2 + 31x - 3391$ (Fung 1988)
(11) $f(x) = 47\ x^2 + 9x - 5209$ (Fung 1989)
(12) $f(x) = 36\ x^2 + 18x - 1801$ (Ruby 1989)

La **forma generalizzata per le parabole non euleriane** è del tipo:

(13) $f(x) = a\ x^2 + b\ x - c$

In realtà si può far uso di varie forme polinomiali. Ad esempio le forme generatrici di numeri primi sono le **forme ellittiche**:

$f(x)=x^3+ a\ x +b$

Se si pone ad esempio con a=1, b=1, si nota che f(x) è in grado di generare numeri primi. Se ne possono inventare ancora altre polinomiali del tipo x^4+x+1, etc.

Appendice

In appendice si mostrano sorgenti in PARI/GP per la generazione ed il conteggio dei seguenti numeri primi:

- Primi di Landau
- Numeri primi gemelli
- Numeri primi di Polignac a distanza d>2
- Numeri di Sophie German (2n+1)

```
/*
    LibThN
    R. Turco
*/

/*****************************************************
* ListLandauPrimes(n)
* It returns the Landau's primes up to n on a file

*
* GetLandauPrimes(n)
* It counts the Landau's primes up to n
*
* R. Turco
* PARI/GP
*****************************************************/
{ListLandauPrimes(n) = local(i, j);

  j=0;
  for(i=1,n,
```

```
     if( isprime(i*i + 1), write1("c:\\srcpari\\landau.txt"," ", i*i + 1));
     if( isprime(i*i + 1), j=j+1);
     if( i*i+1 > n, break;);
  );
  print1(" #Landau's prime up to ", n, " : ", j);

}
{GetLandauPrimes(n) = local(i,j);

  j=0;
  for(i=1,n,
     if( isprime(i*i + 1), j=j+1);
     if( i*i+1 > n, break;);
  );
  return(j);
}
{GetLandauPrimesFrom(n,np,L) = local(i,j, a);

  j=L;
  a=floor(np);
  for(i=a, n,
     if( isprime(i*i + 1), j=j+1);
     if( i*i+1 > n, break;);
  );
  return(j);
}


/*******************************************************
* ListTwinPrimes(n)
* It returns the Twin Primes up to n on a file
*
* GetTwinPrimes(n)
* It counts the Twin Primes up to n
*
* R. Turco
*
*******************************************************/
{ListTwinPrimes(n) = local(i, j, k);

  j=0;
  for(i=3,n-2,

     k = i + 2;

     if( isprime(i),
         if( isprime(k), write1("c:\\srcpari\\twinprimes.txt"," ", i, "-", k);
j=j+1);
             i = i + 2;
     );
     if( k > n, break;);

   );
  print1(" #Twin Primes up to ", n, " : ", j);

}

{GetTwinPrimes(n) = local(i,j,k);
  j=0;
  for(i=3,n-2,

     k = i + 2;
```

```
     if( isprime(i),
         if( isprime(k),j=j+1);
             i = i + 2;
     );
     if( k > n, break;);
   );
  return(j);
}

/*******************************************************
* ListPolignacPrimes(n,d)
* It returns the Polignac's Primes up to n with distance d
*
* GetPolignacPrimes(n, d)
* It counts the Polignac's Primes up to n with distance d
*
* R. Turco
*
*******************************************************/
{ListPolignacPrimes(n,d) = local(i, j, k);

  if( d==0, print("help: GetPolignacPrimes(n,d)"); return);
  if( Mod(d,2) != 0, error("You must insert even numbers"));

  j=0;
  for(i=3,n-d,

     k = i + d;

     if( isprime(i),
         if( isprime(k), write1("c:\\srcpari\\poligancprimes.txt"," ", i, "-",
k); j=j+1);
             i = i + d;
     );
     if( k > n, break;);

   );
  print1(" #Polignac Primes  up to ", n, " : ", j);

}

{GetPolignacPrimes(n,d) = local(i,j,k);

  if( d==0, print("help: GetPolignacPrimes(n,d)"); return);
  if( Mod(d,2) != 0, error("You must insert even numbers"));

  j=0;
  for(i=3,n-d,
     k = i + d;

     if( isprime(i),
         if( isprime(k),j=j+1);
             i = i + d;
     );
     if( k > n, break;);
   );
  return(j);
}

/*******************************************************
* ListSophieGermainPrimes(n,d)
* It returns the Sophie Germain's primes up to n
*
```

```
* GetSophieGermainPrimes(n)
* It counts the Sophie Germain's primes up to n
*
* R. Turco
*
*******************************************************/
{ListSophieGermainPrimes(n) = local(i, j, s);

  j=0;
  for(i=2,n,
     if( isprime(i),
       s = 2*i + 1;
       if( isprime(s), write1("c:\\srcpari\\sophiegermain.txt"," ", i,":"
s);j=j+1;);
     );
     if( i > n, break;);
   );
  print1(" #Sophie Germain Primes  up to ", n, " : ", j);
}
{GetSophieGermainPrimes(n) = local(i, j, s);

  j=0;
  for(i=2,n,
     if( isprime(i),
       s = 2*i + 1;
       if( isprime(s),
           j=j+1;
         );
     );
     if( i > n, break;);
   );

   return(j);


}

/*******************************************************
* Goldbach

° I returns the pairs (x,n-x) of prime numbers,
° not repeate, equals to an even number n.

* Rosario Turco

*******************************************************/
{printgold(n) = local ();

  if( n<=4, error("printgold(n) --> with n>4 and even"));

  if( Mod(n,2) != 0, error("prinntgold(n) --> You must insert an even n>4"));

         for(p=3, n, if(
                         isprime(p) & isprime(n-p) & p <= n-p,
                         print1("(", p, "," , n-p, ")", " ");
                        )
          );

}

{gold(n) = local s;

  if( n<=4, error("gold(n) --> n>4 and even"));
```

```
  if( Mod(n,2) != 0, error("gold(n) --> You must insert an even n>4"));
          s=0;

          for(p=3, n, if(
                         isprime(p) & isprime(n-p) & p <= n-p,
                         s++
                        )
          );

          return(s)
}

/*******************************************************
* GetHelp
* Dà l'elenco delle funzionalità create
*
* R. Turco
*
*******************************************************/
{GetHelp() = local();
  print1(" # ListLandauPrimes(n)\n");
  print1(" # GetLandauPrimes(n)\n");
  print1(" # GetLandauPrimesFrom(n,np,L)\n");
  print1(" # ListTwinPrimes(n)\n");
  print1(" # GetTwinPrimes(n)\n");
  print1(" # ListPolignacPrimes(n,d)\n");
  print1(" # GetPolignacPrimes(n,d)\n");
  print1(" # ListSophieGermainPrimes(n)\n");
  print1(" # GetSophieGermainPrimes(n)\n");
  print1(" # printgold(n)\n");
  print1(" # gold(n)\n");

}
```