The Lucas-Lehmer-test for Mersenne-numbers and the number Λ ~1.389910663524…

Introduction

The iterative Lucas-Lehmer-test for testing primality of Mersenne-numbers can be expressed as iteration of some function *fLL* :

$$fLL(x) = -2 + x^2$$

(I use examples in the mathematical software program *Pari/GP* in the following):

```
fLL(x)= -2 + x^2   \\ for Lucas Lehmer-test: Iterate beginning at x0=4
```

and its $(p-2)^{th}$ iteration beginning at $x_0=4$ for the p^{th} Mersenne-number $M_p=2^p-1$

$$fLL^{\circ p-2}(4) \equiv 0 \pmod{2^p-1} \qquad \text{Lucas-Lehmer-test of } M_p = 2^p-1$$

Pari/GP:

```
fLL_iter(x0,n,p) = local(mp = 2^p-1); for(k=1,n,  x0 = fLL(x0) % mp ) ;x0
fLucLeh(p) = fLL_iter(4, p-2, p) == 0   \\ Result: 0: Mp is composite, 1: Mp is prime
```

The motivation for this small article was initially to look at that formula as a problem of ***continuous*** iteration (or: how the discrete iterations of the test could be embedded in a continuous flow), and to see, whether we can make something noteworthy out of this more general view. In the end the fractional iteration was not evaluated further after the formalism for this had given rise to a curious re-definition of the Lucas-Lehmer-test in terms of a single constant.

So we'll reflect here the well known recipe for the continuous iteration of (feasible) functions as developed by Ernst Schröder in the late 19th century[1][2].

This method is best suited for analytic functions without constant term in their power series representation. So we have first to find fixpoints of *fLL* and then accordingly to recenter the polynomial around one of that fixpoints for the forthgoing analysis. Then the "Schröder-function" and its inverse must be de-

[1] [Schroeder] Ernst Schröder (1870): Über iterierte Funktionen ; see reference list

[2] [wikiSchr] Wikipedia: "Schroederfunction", "iterated function"

termined to be able to formulate fractional iterates simply by fractional powers of a constant λ . The formula has in principle the form:

$$x_h = fLL\_iter\_frac(h , x_0) = \sigma^{-1} (\lambda^h * \sigma(x_0 - t_1)) + t_1$$

Here σ is the Schröder-function and σ^{-1} its inverse, t_1 one feasible fixpoint, λ the eigenvalue of the (recentered) original function, and h the iteration "height".

The mechanism expressed by the Schröder-functions is perfectly parallel to the use of diagonalization of the matrix-operator (Carlemanmatrix[3]) associated to the original function fLL_1 and we shall use that formalism here because in general it makes the relations and whereabouts much transparent. Continuous iteration reduces then to using fractional powers of the eigenvalues of the matrixoperator. Obviously that eigenvalues should then be positive and this is actually the case for the second of the recentered polynomials.

I've implemented the core procedures of this in the software *Pari/GP* (for documentation see end of the article) and I'll show the discussion mostly in form of the commented protocol of the software-interactions.

Construction of the matrixoperators for the recentered polynomials

For the implementation of arbitrary iterability we need fixpoints

In $fLL(x) = -2 + x^2$

the condition for fixpoints is:

$fLL(x) = x$

=> solutions: $t_0 = -1$

$t_1 = 2$

```
\\ -2+x² find fixpoints -2 + x² = x or x² = 2 + x
t0 = -1
t1 =  2
```

This gives two different polynomials $fLL_0(x)$ and $fLL_1(x)$ recentered at the two fixpoints. Both have then no constant term:

$$fLL_0(x) = fLL(x+t_0) - t_0 = -2x + x^2$$
$$fLL_1(x) = fLL(x+t_1) - t_1 = 4x + x^2$$

and have thus a form better suited also for the definition of noninteger iteration in terms of power series:

```
\\ recentered functions without constant term
  \\ fLL_0(x) = fLL(x+t0)-t0   and t0 = -1
  \\ and insert iterability to (integer) heights h by a for-loop
fLL_0(x,h=1) = for(k=1,h,  x = -2*x + x^2 );x
  \\ fLL_1(x) = fLL(x+t1)-t1  and t1 = 2
fLL_1(x,h=1) = for(k=1,h,  x =  4*x + x^2 );x
```

[3] [wikiCarleman] Wikipedia: "Carlemanmatrix" (we use the transposed version here)

For the non-integer iteration heights we'll employ the concept of Schröder-functions, implemented by the diagonalization of the matrixoperators/ Carleman-matrices associated with the functions.

So we generate matrix-operators (Carleman-matrices, transposed) for preparing the arbitrary height iteration (with *n=128*) rows/colums:

```
M0 = carleman (polcoeffs(fLL_0(x),n));    \\ matrixoperator/Carlemanmatrix for fLL_0
M1 = carleman (polcoeffs(fLL_1(x),n));    \\ for fLL_1
```

In the following we'll look at M_1 only because M_0 has a negative eigenvalue, which is uncomfortable for fractional iteration. Also with a size of *n x n = 128 x 128* a default float-precision of *200* dec digits in the software and a simple convergence-acceleration due to Euler ("Euler summation") we the get intermediate values by the Schröder-function which are then exact to at least *30* digits with some relevant parameters.

Prepare Schröderfunctions at the second fixpoint via matrix-diagonalization

To generate the diagonalization-matrices we solve for the hypothese $M_1 = W\,D\,W^{-1}$ where ***D*** shall be diagonal. This can principally be done by the procedure *"mateigen"* in *Pari/GP*.

```
        W = mateigen(M1); WI = W^-1; D = WI * M1 * W ;
```

Unfortunately this will produce spurious entries which makes the resulting matrices non-triangular. But we can replace this by a specialized procedure for triangular matrices (which also provides exact rational or even symbolic matrixentries depending on the entries of the matrix to be diagonalized):

```
M1K = TriEigen(M1); W = M1K[1]; D = M1K[2]; WI = M1K[3]; \\TriEigen returns
                                                  \\ a vector of 3 matrices
```

This are the resulting (triangular) matrices ***W*** and ***WI***, (the trailing dots indicate infinite size):

$$W=\begin{bmatrix} 1 & . & . & . & . & . & .. \\ . & 1 & . & . & . & . & .. \\ . & -1/12 & 1 & . & . & . & .. \\ . & 1/90 & -1/6 & 1 & . & . & .. \\ . & -1/560 & 7/240 & -1/4 & 1 & . & .. \\ . & 1/3150 & -41/7560 & 13/240 & -1/3 & 1 & .. \\ .. & .. & .. & .. & .. & .. & .. \end{bmatrix} \quad WI=\begin{bmatrix} 1 & . & . & . & . & . & .. \\ . & 1 & . & . & . & . & .. \\ . & 1/12 & 1 & . & . & . & .. \\ . & 1/360 & 1/6 & 1 & . & . & .. \\ . & 1/20160 & 1/80 & 1/4 & 1 & . & .. \\ . & 1/1814400 & 17/30240 & 7/240 & 1/3 & 1 & .. \\ .. & .. & .. & .. & .. & .. & .. \end{bmatrix}$$

The diagonalmatrix ***D*** contains the consecutive powers of *4*:

D=matdiagonal ([1,4,16,64,...])

The matrix ***W*** contains now the coefficients (*[0, 1, -1/12, 1/90,...]*) of the Schröder-function and ***WI*** that (*[0, 1, 1/12, 1/360,...]*) of its inverse in their second columns. (Note, that in *Pari/GP* matrix- and vectorindices begin at *1*), so we have:

$$\sigma_n(x) \sum_{k=0}^{n-1} W_{1+k,1+1} x^k \qquad \sigma_n^{-1}(x) = \sum_{k=0}^{n-1} WI_{1+k,1+1} x^k$$

and in the limit with infinitely many coefficients

$$\sigma(x) = \lim_{n\to\infty} \sigma_n(x) \qquad \sigma^{-1}(x) = \lim_{n\to\infty} \sigma^{-1}{}_n(x)$$

A first shot: the numerical computation of the Schröderfunction

First we compute the value of the Schröderfunction at $x_0=4$. To improve approximation-quality we also use a low-order of Euler-summation/ convergence-acceleration. The resulting value s_1 is then the (approximate) value of the Schröder-function at x_0:

```
x0 = 4
s1 = ESumVec(0.25)*dV(x0 - t1)*W[,2]
       \\    = 1.7343781022726361504  This is unknown to Math'ca and Plouffe
```

Next we find, that the coefficients in ***WI*** (for the inverse-Schröder-function)

```
WI[,2]   \\ show first n coefficients of the inverse of the Schröder-function
```

form an easily interpretable sequence

$$WI_{1+k,2} = 2\frac{1}{(2k+1)!}$$

```
2 * vectorv(n,r, 1/(2*(r-1))! )
```

That coefficients are much familiar: in principle this is the set of coefficients of the function *cosh(x)*:

```
2 * cosh(x)+O(x^16) \\ see that the taylor-series agree in principle
```

However, the sequence of coefficients in ***WI*** is dense but not in *cosh(x)* where each second coefficient vanishes. But the match of coefficients in ***WI*** and *cosh(x)* can be achieved if we write *cosh(sqrt(x))* instead. This way being inserted in the inverse of the Schröder-function gives us then exactness of the coefficients up to rational numbers and allows to leave that (harshly truncated!) example of the ***WI*** behind and to resort to the much more exact expression in terms of the *cosh()* directly.

Arriving at a formula of exp only via cosh and cosh^{-1}

So we adapt the approximated value s_1 to fit conveniently the *2*cosh(sqrt(x))*-formula and call it the "Lucas-Lehmer-number" λ = *LucLeh*:

```
LucLeh=sqrt(s1)*2^-2  \\ adapt the "Lucas-Lehmer"-number s1 to fit the 2*cosh(x)-formula
         \\  0.329239474231204177…
```

and check for a few values of *p*, for instance *p=5*:

```
[p=5 , mp = 2^p - 1 ]
2*cosh(LucLeh * 2^p)                    \\   = 37634.0000000
round(2*cosh(LucLeh * 2^p),&e)          \\   = 37634
round(2*cosh(LucLeh * 2^p),&e) % mp     \\   = 0
```

Since we are involved with iterations and the outer function is *cosh*, we could now try whether λ is the *cosh*$^{-1}$ of some more familiar argument. And bingo! : Math'ica gives a much convincing guess for the value *cosh(λ)*,

$$cosh(\lambda) \approx \frac{\sqrt{2+\sqrt{6}}}{2}$$

so that we guess now the closed-form definition for the new constant λ:

$$\lambda = \cosh^{-1}\left(\frac{\sqrt{2+\sqrt{6}}}{2}\right)$$

with precision to arbitrarily many digits:

```
LucLeh = acosh(sqrt(2+sqrt(2+4))/2)
\\  0.32923947423120417715626158682699211100674549282110608651680 0
\\ check for some p:
    [p=5, mp = 2^p - 1]
2*cosh(LucLeh*2^p)                    \\  = 37634.0000000
round(2*cosh(LucLeh*2^p),&e)          \\  = 37634
round(2*cosh(LucLeh*2^p),&e) % mp     \\  = 0
```

We can even append a final step to put it into one more familiar expression: because we have *2*cosh(x) = exp(x) + exp(-x)* and *exp(-x)* becomes exponentially small for all the *x* in question we can ignore that latter spurious contribution and can reduce to

```
(exp(LucLeh*2^p) + exp(-LucLeh*2^p))  % mp
\\      reduces to
ceil(exp(LucLeh*2^p))   % mp
ceil(exp(LucLeh*2^3))   % (2^3 - 1)
ceil(exp(LucLeh*2^7))   % (2^7 - 1)
```

where all three examples meet the expected results.

Thus we can define an exponential Lucas-Lehmer constant $\Lambda = \exp(\lambda)$

```
ELucLeh = exp(LucLeh)
\\ 1.389910663524147717911548811992210102196089903539205052651820
```

and give the Lucas-Lehmer-formula in the most simplified form (for instance *p=3*)

```
ELucLeh^2^3                 \\   = 13.9282032303
ceil(ELucLeh^2^3)           \\   = 14
ceil(ELucLeh^2^3) % (2^3-1) \\   = 0
\\ thus as far as numerical precision and number of integer-digits allow:
p2 = 2^p  \\ take p a prime
Mp = p2 - 1
if(0 == ceil (ELucLeh^p2) % Mp ,print("Mp is a mersenne prime"))
```

Unfortunately, this way of computation requires exponentially many digits for one iterative Lucas-Lehmer-test and thus is computationally useless on computers already for $p>7$, but however may survive as a curious isolated result:

Let $\Lambda = e^{\lambda} \sim 1.389910663524$,
$p \in \langle primes \rangle$,
$M_p = 2^p - 1$
then

$$\left\lceil \Lambda^{2^p} \right\rceil \equiv 0 \pmod{M_p} \quad \Leftrightarrow \quad M_p \in \langle primes \rangle$$

Gottfried Helms, 25.11.2011 *email: helms@uni-kassel.de*

Remark: the cosh-connection of that quadratic map was already known to E. Schröder (see [Schroeder]) and is meanwhile also widely well known. For instance we find that connection basically in a nice collection of Jörg Arndt [JArndt], pg. 799 and it found its way from there to Chris Caldwell's much interesting and entertaining "prime pages" [Caldwell]. A nice more general discussion about the iterated quadratic map is in Troy Vasiga & Jeffrey Shallit (see [VasShall]) "On the iteration of certain quadratic maps over GF(p)" of 2003.

Literature and Online links

[Schroeder] Über iterirte Functionen
Ernst Schröder
Math. Ann. 3 (2): 296–322

[wikiSchr] wikipedia "Schröder's equation"
http://en.wikipedia.org/wiki/Schr%C3%B6der%27s_equation

[wikiFIter] wikipedia "Function iteration"
http://en.wikipedia.org/wiki/Function_iteration

[wikiCarleman] wikipedia "Carleman Matrix"
http://en.wikipedia.org/wiki/Carleman_matrix

[VasShall] On the iteration of certain quadratic maps over GF(p)
Troy Vasiga & Jeffrey Shallit

[Oeis3010] sequence in "Online encyclopedia of integer sequences" (OEIS)
N.J.A. Sloane (editor)
http://oeis.org/A003010

[wikiLucLeh] wikipedia: Lucas-Lehmer-test
http://en.wikipedia.org/wiki/Lucas%E2%80%93Lehmer_primality_test

[JArndt] Matters Computational – the fxtbook
Jörg Arndt
edited June 2010, pg 799
http://www.jjj.de/fxt/fxtbook.pdf

[Caldwell] "The prime pages": 3.2: *n+1* tests and the Lucas-Lehmer test
Chris Caldwell
http://primes.utm.edu/prove/prove3_2.html

Pari/GP: basic functions used in the text

```
\\   This documentation is for simplicity without error-checks and optimizations
\\ ------------------------------------------------------------------------------
default(floatprecision,200)             \\ default precision for most procedures
n=64 \\ set default vector and matrix-dimension
```

* *Vandermonde vectors for evaluation of power series*

```
 V(x,dim=n) = vector(dim,c,x^(c-1))
dV(x,dim=n) = matdiagonal(V(x,dim))
```

* *Inversion of invertible lower triangular matrices, no error check*

```
TriInv(m) = local(tmp, rs=rows(m), cs=cols(m));
   tmp = matrix(rs,cs);
   for(c = 1, cs,
         tmp[c,c] = 1 / m[c,c];
         for(r = c+1, rs,
               tmp[r,c] = - sum(k=c,r-1, m[r,k] * tmp[k,c]) / m[r,r]
            );
       );
    return(tmp);
```

* *diagonalization of diagonalizable lower triangular matrices, no error check*

```
TriEigen(U) = local( W, WI, D, dim);
    dim = rows(U) ;
    D = vectorv(dim,r, U[r,r]);
    W = matid(dim);
    for(c=1,dim-1,
         for(r=c+1,dim,
              W[r,c] = sum(k=c,r-1, U[r,k] * W[k,c]) / (D[c]-D[r])
        ));
    WI =  matid(dim);
    for(r=2,dim,
         forstep(c = r-1,1,-1,
                  WI[r,c] = sum(k=0,r-1-c,U[r-k,c] * WI[r,r-k])/(D[r]-D[c])
       ));
return([W,D,WI]);
```