Shameless *Mathematica* - Some Fun with Prime Numbers

Hareem Naveed

March 18, 2016

Credit: This project is inspired by some of the work we did with Fibonacci numbers. It mainly draws on the theory listed here: https://primes.utm.edu/prove/merged.html, and relies on “A Classical Introduction to Modern Number Theory” as a reference.

In this project, we implement a test to check for primality of large numbers, specifically, we check the primality of Fermat numbers. We then implement a heuristic search method to find large prime numbers.

Fermat Numbers

A **Fermat Number** is a number of the form $2^{(2^n)} + 1$, where n is a positive integer. First, let us write a short function and output the first few Fermat numbers:

```
fermatnumber[n_] := 2^(2^n) + 1
```

```
fermatnumber[2]
```

```
17
```

```
Grid[
 Prepend[
  Table[{n, fermatnumber[n], PrimeQ[fermatnumber[n]]}, {n, 0, 10}],
  {"n", "F_n", "Prime"}], Frame → All]
```

n	F_n	Prime
0	3	True
1	5	True
2	17	True
3	257	True
4	65 537	True
5	4 294 967 297	False
6	18 446 744 073 709 551 617	False
7	340 282 366 920 938 463 463 374 607 431 768 211 457	False
8	115 792 089 237 316 195 423 570 985 008 687 907 853 269 984 665 640 564 039 457 584 ∖ 007 913 129 639 937	False
9	13 407 807 929 942 597 099 574 024 998 205 846 127 479 365 820 592 393 377 723 561 ∖ 443 721 764 030 073 546 976 801 874 298 166 903 427 690 031 858 186 486 050 853 ∖ 753 882 811 946 569 946 433 649 006 084 097	False
10	179 769 313 486 231 590 772 930 519 078 902 473 361 797 697 894 230 657 273 430 081 ∖ 157 732 675 805 500 963 132 708 477 322 407 536 021 120 113 879 871 393 357 658 ∖ 789 768 814 416 622 492 847 430 639 474 124 377 767 893 424 865 485 276 302 219 ∖ 601 246 094 119 453 082 952 085 005 768 838 150 682 342 462 881 473 913 110 540 ∖ 827 237 163 350 510 684 586 298 239 947 245 938 479 716 304 835 356 329 624 224 ∖ 137 217	False

Only the first four Fermat numbers are prime, and as we can see, Fermat numbers grow very rapidly! At this point, it is noted that using current computational methods, it is unlikely that other prime Fermat numbers will be found. Note from the above table that should another prime Fermat number be found, it would be a very large number and it would be very difficult to check it's primality.

For example, *Mathematica* has trouble computing the factors for even the 10th Fermat number, and times out of the computation. Note that the it only takes 1.06 seconds to compute the factors of the 8th Fermat number:

```
FactorInteger[fermatnumber[8]] // Timing
```

```
{0.59375, {{1 238 926 361 552 897, 1},
  {93 461 639 715 357 977 769 163 558 199 606 896 584 051 237 541 638 188 580 280 321, 1}}}
```

In this project, we will explore both Fermat numbers and work with some large primes.

First, we will implement Pépin's test, which is a test to check the primality of a Fermat number. Second, we will work with *Mathematica*'s inbuilt "Prime[n]" function which returns the n'th prime, and implement a simple methods to find large prime numbers.

First, let us update the above table with timing for each of the evaluations:

```
Grid[
 Prepend[Table[{n, fermatnumber[n], Timing[PrimeQ[fermatnumber[n]]]}, {n, 0, 13}],
  {"n", "F_n", "Time, Prime"}], Frame → All]
```

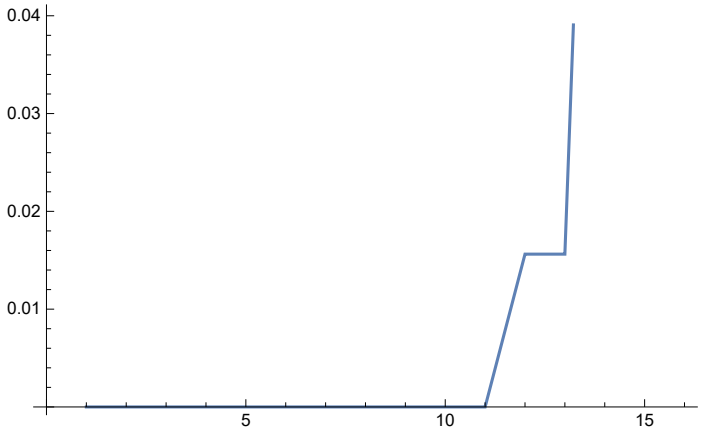
n	F_n	Time, Prime
0	3	{0., True}
1	5	{0., True}
2	17	{0., True}
3	257	{0., True}
4	65 537	{0., True}
5	4 294 967 297	{0., False}
6	18 446 744 073 709 551 617	{0., False}
7	340 282 366 920 938 463 463 374 607 431 768 211 457	{0., False}
8	115 792 089 237 316 195 423 570 985 008 687 907 853 269 984 665 640 ∴ 564 039 457 584 007 913 129 639 937	{0., False}
9	13 407 807 929 942 597 099 574 024 998 205 846 127 479 365 820 592 393 ∴ 377 723 561 443 721 764 030 073 546 976 801 874 298 166 903 427 690 ∴ 031 858 186 486 050 853 753 882 811 946 569 946 433 649 006 084 097	{0., False}
10	179 769 313 486 231 590 772 930 519 078 902 473 361 797 697 894 230 ∴ 657 273 430 081 157 732 675 805 500 963 132 708 477 322 407 536 021 ∴ 120 113 879 871 393 357 658 789 768 814 416 622 492 847 430 639 474 ∴ 124 377 767 893 424 865 485 276 302 219 601 246 094 119 453 082 952 ∴ 085 005 768 838 150 682 342 462 881 473 913 110 540 827 237 163 350 ∴ 510 684 586 298 239 947 245 938 479 716 304 835 356 329 624 224 137 ∴ 217	{0., False}

11	32 317 006 071 311 007 300 714 876 688 669 951 960 444 102 669 715 484 ∖ 032 130 345 427 524 655 138 867 890 893 197 201 411 522 913 463 688 ∖ 717 960 921 898 019 494 119 559 150 490 921 095 088 152 386 448 283 ∖ 120 630 877 367 300 996 091 750 197 750 389 652 106 796 057 638 384 ∖ 067 568 276 792 218 642 619 756 161 838 094 338 476 170 470 581 645 ∖ 852 036 305 042 887 575 891 541 065 808 607 552 399 123 930 385 521 ∖ 914 333 389 668 342 420 684 974 786 564 569 494 856 176 035 326 322 ∖ 058 077 805 659 331 026 192 708 460 314 150 258 592 864 177 116 725 ∖ 943 603 718 461 857 357 598 351 152 301 645 904 403 697 613 233 287 ∖ 231 227 125 684 710 820 209 725 157 101 726 931 323 469 678 542 580 ∖ 656 697 935 045 997 268 352 998 638 215 525 166 389 437 335 543 602 ∖ 135 433 229 604 645 318 478 604 952 148 193 555 853 611 059 596 230 ∖ 657	{0.015625, False}
12	1 044 388 881 413 152 506 691 752 710 716 624 382 579 964 249 047 383 ∖ 780 384 233 483 283 953 907 971 557 456 848 826 811 934 997 558 340 ∖ 890 106 714 439 262 837 987 573 438 185 793 607 263 236 087 851 365 ∖ 277 945 956 976 543 709 998 340 361 590 134 383 718 314 428 070 011 ∖ 855 946 226 376 318 839 397 712 745 672 334 684 344 586 617 496 807 ∖ 908 705 803 704 071 284 048 740 118 609 114 467 977 783 598 029 006 ∖ 686 938 976 881 787 785 946 905 630 190 260 940 599 579 453 432 823 ∖ 469 303 026 696 443 059 025 015 972 399 867 714 215 541 693 835 559 ∖ 885 291 486 318 237 914 434 496 734 087 811 872 639 496 475 100 189 ∖ 041 349 008 417 061 675 093 668 333 850 551 032 972 088 269 550 769 ∖ 983 616 369 411 933 015 213 796 825 837 188 091 833 656 751 221 318 ∖ 492 846 368 125 550 225 998 300 412 344 784 862 595 674 492 194 617 ∖ 023 806 505 913 245 610 825 731 835 380 087 608 622 102 834 270 197 ∖ 698 202 313 169 017 678 006 675 195 485 079 921 636 419 370 285 375 ∖ 124 784 014 907 159 135 459 982 790 513 399 611 551 794 271 106 831 ∖ 134 090 584 272 884 279 791 554 849 782 954 323 534 517 065 223 269 ∖ 061 394 905 987 693 002 122 963 395 687 782 878 948 440 616 007 412 ∖ 945 674 919 823 050 571 642 377 154 816 321 380 631 045 902 916 136 ∖ 926 708 342 856 440 730 447 899 971 901 781 465 763 473 223 850 267 ∖ 253 059 899 795 996 090 799 469 201 774 624 817 718 449 867 455 659 ∖ 250 178 329 070 473 119 433 165 550 807 568 221 846 571 746 373 296 ∖ 884 912 819 520 317 457 002 440 926 616 910 874 148 385 078 411 929 ∖ 804 522 981 857 338 977 648 103 126 085 903 001 302 413 467 189 726 ∖ 673 216 491 511 131 602 920 781 738 033 436 090 243 804 708 340 403 ∖ 154 190 337	{0.015625, False}

13	1 090 748 135 619 415 929 462 984 244 733 782 862 448 264 161 996 232 × 692 431 832 786 189 721 331 849 119 295 216 264 234 525 201 987 223 × 957 291 796 157 025 273 109 870 820 177 184 063 610 979 765 077 554 × 799 078 906 298 842 192 989 538 609 825 228 048 205 159 696 851 613 × 591 638 196 771 886 542 609 324 560 121 290 553 901 886 301 017 900 × 252 535 799 917 200 010 079 600 026 535 836 800 905 297 805 880 952 × 350 501 630 195 475 653 911 005 312 364 560 014 847 426 035 293 551 × 245 843 928 918 752 768 696 279 344 088 055 617 515 694 349 945 406 × 677 825 140 814 900 616 105 920 256 438 504 578 013 326 493 565 836 × 047 242 407 382 442 812 245 131 517 757 519 164 899 226 365 743 722 × 432 277 368 075 027 627 883 045 206 501 792 761 700 945 699 168 497 × 257 879 683 851 737 049 996 900 961 120 515 655 050 115 561 271 491 × 492 515 342 105 748 966 629 547 032 786 321 505 730 828 430 221 664 × 970 324 396 138 635 251 626 409 516 168 005 427 623 435 996 308 921 × 691 446 181 187 406 395 310 665 404 885 739 434 832 877 428 167 407 × 495 370 993 511 868 756 359 970 390 117 021 823 616 749 458 620 969 × 857 006 263 612 082 706 715 408 157 066 575 137 281 027 022 310 927 × 564 910 276 759 160 520 878 304 632 411 049 364 568 754 920 967 322 × 982 459 184 763 427 383 790 272 448 438 018 526 977 764 941 072 715 × 611 580 434 690 827 459 339 991 961 414 242 741 410 599 117 426 060 × 556 483 763 756 314 527 611 362 658 628 383 368 621 157 993 638 020 × 878 537 675 545 336 789 915 694 234 433 955 666 315 070 087 213 535 × 470 255 670 312 004 130 725 495 834 508 357 439 653 828 936 077 080 × 978 550 578 912 967 907 352 780 054 935 621 561 090 795 845 172 954 × 115 972 927 479 877 527 738 560 008 204 118 558 930 004 777 748 727 × 761 853 813 510 493 840 581 861 598 652 211 605 960 308 356 405 941 × 821 189 714 037 868 726 219 481 498 727 603 653 616 298 856 174 822 × 413 033 485 438 785 324 024 751 419 417 183 012 281 078 209 729 303 × 537 372 804 574 372 095 228 703 622 776 363 945 290 869 806 258 422 × 355 148 507 571 039 619 387 449 629 866 808 188 769 662 815 778 153 × 079 393 179 093 143 648 340 761 738 581 819 563 002 994 422 790 754 × 955 061 288 818 308 430 079 648 693 232 179 158 765 918 035 565 216 × 157 115 402 992 120 276 155 607 873 107 937 477 466 841 528 362 987 × 708 699 450 152 031 231 862 594 203 085 693 838 944 657 061 346 236 × 704 234 026 821 102 958 954 951 197 087 076 546 186 622 796 294 536 × 451 620 756 509 351 018 906 023 773 821 539 532 776 208 676 978 589 × 731 966 330 308 893 304 665 169 436 185 078 350 641 568 336 944 530 × 051 437 491 311 298 834 367 265 238 595 404 904 273 455 928 723 949 × 525 227 184 617 404 367 854 754 610 474 377 019 768 025 576 605 881 × 038 077 270 707 717 942 221 977 090 385 438 585 844 095 492 116 099 × 852 538 903 974 655 703 943 973 086 090 930 596 963 360 767 529 964 × 938 414 598 185 705 963 754 561 497 355 827 813 623 833 288 906 309 × 004 288 017 321 424 808 663 962 671 333 528 009 232 758 350 873 059 ×	{0.109375, False}

In another format, note that when we plot the time for computation, it grows exponentially. We want an algorithm that accomplishes the same task in polynomial time.

```
ListLinePlot[Table[First[Timing[PrimeQ[fermatnumber[n]]]], {n, 0, 15}],
 PlotStyle → PointSize[Large]]
```




```
First[Timing[PrimeQ[fermatnumber[15]]]]
```

```
4.125
```

Pépin's Test:

If we let F_n be the n'th Fermat number, F_n is prime iff $3^{(F_n-1)/2} \equiv -1 \pmod{F_n}$. We can implement this using the repeated-squaring technique in a "Do" function. Note that the base can similarly be 5 or 7.

```
a = 3;
Do[a = Mod[a^2, fermatnumber[3]], {n, 1, 2^3 - 1}]
Print[{a, fermatnumber[3], a - fermatnumber[3]}] // Timing
```

```
{256, 257, -1}
{0., Null}
```

```
PrimeQ[fermatnumber[3]] // Timing
```

```
{0., True}
```

```
a = 7;
Do[a = Mod[a^2, fermatnumber[8]], {n, 1, 2^8 - 1}]
Print[{a, fermatnumber[8], a - fermatnumber[8]}] // Timing
```

```
{106 580 163 108 021 291 013 588 185 217 682 439 217 006 089 516 199 392 945 620 072 055 187 518 961 076,
 115 792 089 237 316 195 423 570 985 008 687 907 853 269 984 665 640 564 039 457 584 007 913 129 639 937,
 -9 211 926 129 294 904 409 982 799 791 005 468 636 263 895 149 441 171 093 837 511 952 725 610 678 861}
{0., Null}
```

```
PrimeQ[fermatnumber[8]] // Timing
```

```
{0., False}
```

Note that this method is only faster than the native “PrimeQ” function when working with larger Fermat numbers. Let us plot the output of calling this function for the different Fermat numbers and compare with the “PrimeQ” plot above. To do this, first, let us write a function that implements Pépin’s test.

```
pepinstest[n_] := Module[{m, b},
  m = fermatnumber[n];
  b = 3;
  Do[b = Mod[b^2, m], {m, 1, 2^n - 1}];
  If[Mod[b, m] == m - 1, Return["prime"], Return["composite"]]]
```

```
pepinstest[8] // Timing
```

```
{0., composite}
```

```
pepinstest[10] // Timing
```

```
{0., composite}
```

```
PrimeQ[fermatnumber[10]] // Timing
```

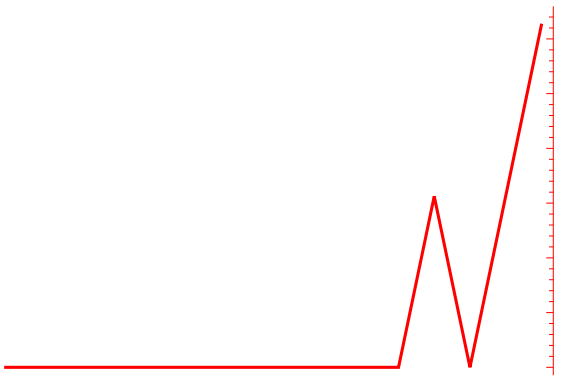
```
{0., False}
```

The new method is *slightly* faster than the in-built “PrimeQ” function up to n = 10, but does much better than the “FactorInteger” function which times out before giving an answer.

```
pepintimes = Table[First[Timing[pepinstest[n]]], {n, 0, 15}]
```

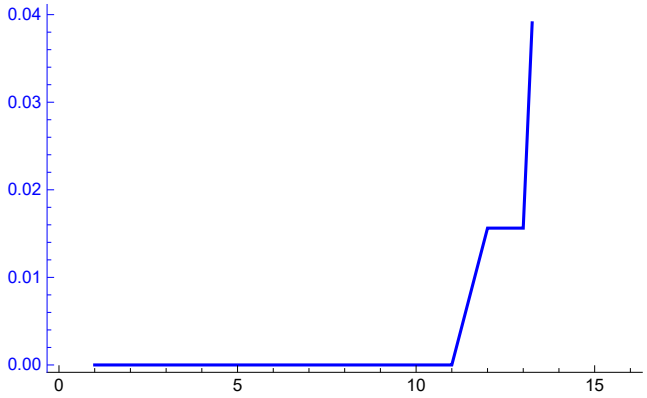
```
{0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.015625, 0., 0.015625, 0.03125}
```

```
plot1 = ListLinePlot[pepintimes, PlotStyle → Red, ImagePadding → 25,
  Frame → {False, False, False, True}, FrameTicks → {None, None, None, All},
  FrameStyle → {Automatic, Automatic, Automatic, Red}]
```

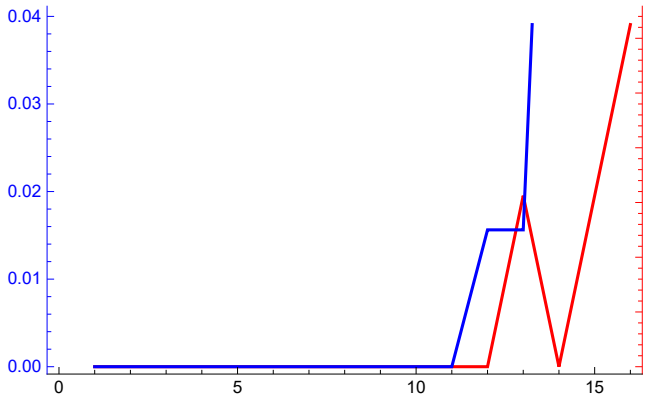


```
primetimes = Table[First[Timing[PrimeQ[fermatnumber[n]]]], {n, 0, 15}];
```

```
plot2 = ListLinePlot[primetimes, PlotStyle → Blue, ImagePadding → 25, Axes → False,
  Frame → {True, True, False, False}, FrameTicks → {All, None, None, None},
  FrameStyle → {Automatic, Blue, Automatic, Automatic}]
```




```
Overlay[{plot1, plot2}]
```




```
pepintimes
```

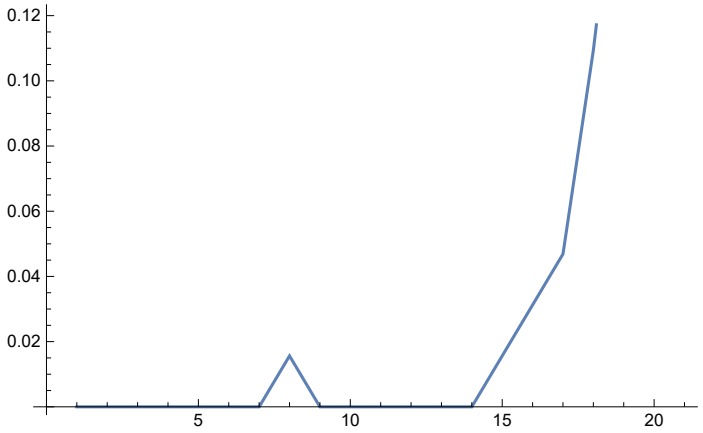
```
{0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.015625, 0., 0.015625, 0.03125}
```

```
primetimes
```

```
{0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0.015625, 0.015625, 0.109375, 0.6875, 4.07813}
```

Note from the above computations that after F_{12}, the native "PrimeQ" function is much slower than Pépin's test for evaluting the primality of the Fermat Number. For F_{15}, "PrimeQ" is 200 times slower than Pépin's test. We can even evaluate the primality of F_{20} relatively quickly using this method!

```
plot3 = ListLinePlot[Table[First[Timing[pepinstest[n]]], {n, 0, 20}]]
```




```
pepinstest[20]
```

```
composite
```

Of course, we know that it will be a composite number, but the purpose of this exercise is to demonstrate the implementation of a trick (Pépin's test) that is faster than the inbuilt computational methods in *Mathematica* for determining the primality of a large number.

Finding Large Prime Numbers

In *Mathematica,* you can specify the nth prime number with a simple command:

```
Prime[100]
```

```
541
```

```
Prime[10^8] // Timing
```

```
{0., 2 038 074 743}
```

The list of largest known primes includes Mersenne primes, as the method used for testing primality is based on factorization of n+1 or n-1. For Mersenne primes, the factorization of n+1 is trivial as it is simply a power of 2.

One way to find a large prime number is to start with a random prime number, then multiply it by another large random integer and add 1.

```
a = Prime[k = Random[Integer, {1, 10^9}]]
```

```
3 380 164 871
```

```
b = Random[Integer, {1, 10^8}]
```

```
31 782 359
```

```
possibleprime = a * b + 1;
```

```
possibleprime
```

```
107 429 613 409 310 690
```

```
PowerMod[2, possibleprime - 1, possibleprime]
```

```
106 950 889 608 339 942
```

This number is clearly not prime. We can write a loop that will check for the different values of b that will give us a prime number.

```
While[PowerMod[2, possibleprime - 1, possibleprime] ≠ 1 &&
  PowerMod[3, possibleprime - 1, possibleprime] ≠ 1 &&
  PowerMod[5, possibleprime - 1, possibleprime] ≠ 1,
 b = Random[Integer, {1, 10^15}];
 possibleprime = a * b + 1]
```

```
possibleprime
```

```
2 865 524 916 037 539 800 098 423
```

```
PrimeQ[possibleprime]
```

```
True
```

By changing the parameters of a and b, and the search space for the “Random” functions, we can specify larger and larger primes.