In [1]:
```
# Newton method of sqrt
# this is actually Heron's method
# based on http://lomont.org/papers/2003/InvSqrt.pdf
# orignal guess within 3% of target inv_sqrt in that paper
# 0.1% after one newton step

###
# Here i show my init_guess for sqrt is within 25% of target sqrt
# 2% after one Newton step

# it's still much slower than default math.sqrt()
```

In [2]:
```
import math
import random
```

In [3]:
```
def f(x, I):
    # function to use the newton method with
    return x*x - I
```

In [4]:
```
def f_p(x):
    # the derivative of the f function (doesn't include I in this case, but in general might)
    return 2*x
```

In [5]:
```
def init_sqrt_guess(x):
    # returns a guess for sqrt(x)
    # get the mantissa and exponent
    m, e = math.frexp(x)
    # we're looking for a new m and e such that (m*2**e)**2 ~= x
    # for the exponent it's just half
    e = e/2.0
    # for the mantissa which is in [0.5,1[ (because x is positive), the best value is obviously sqrt(m)
    # we approximate that by sqrt(x) ~= x+k
    # k = solve ((2 1^(3/2))/3 - 1^2/2 ) - ((2 0.5^(3/2))/3 - 0.5^2/2)-k/2 ==0
    # I believe this k makes the integral of sqrt(x)-(x+k) in between 0.5,1 to be zero
    # and that this is optimal as a first guess (but doesn't necessarily optimized for the first step, the second step...)
    # approximately
    k = 0.111928812542301634
    m = m+k
    # init guess is
    g = m*(2**e)
    return g
```

In [6]:
```
def newton_sqrt(I, max_steps=1, min_error=1e-200, add_noise=False, verbose=Fal
se):
    # computes the sqrt of I using the Newton method
    # default max_step=1 to mirror fast inverse sqrt paper
    # converges very fast so a very small min_error is possible

    I = float(I)

    # initial guess:
    x = init_sqrt_guess(I)

    step=0
    abs_error = abs(x*x-I)
    rel_error = 100.0*abs_error/I

    if verbose:
        print("Init guess: {}".format(x))
        print("Absolute Error: {}".format(abs_error))
        print("Relative Error: {:.10f} %".format(rel_error))

    # takes steps until max_steps or min_error reached, whichever comes first
    while (abs_error>min_error and step<max_steps):

        # add noise to prevent getting stuck?
        if add_noise: x += random.random()*min_error*2

        # Newton step
        x = x - f(x,I)/f_p(x)

        abs_error = abs(x*x-I)
        rel_error = 100.0*abs_error/I
        step += 1

        if verbose:
            print("After step {}".format(step))
            print("Current x: {}".format(x))
            print("Absolute Error: {}".format(abs_error))
            print("Relative Error: {:.10f} %".format(rel_error))

    return x
```

In [7]:
```
def gen_rand_positive_32_bit_float():
    # random positive 32 bit string
    r = '0b'+''.join([str(random.choice([0,1])) for _ in xrange(31)])
    # convert to float
    r = float(int(r,0))
    return r
```

In [8]:
```
# test some known values
num_to_test = 64
print(newton_sqrt(num_to_test))
```

```
8.08374269822
```

In [9]:
```
# see the computation
print(newton_sqrt(num_to_test, verbose=True))
```

```
Init guess: 6.92318420723
Absolute Error: 16.0695204327
Relative Error: 25.1086256761 %
After step 1
Current x: 8.08374269822
Absolute Error: 1.346896011
Relative Error: 2.1045250172 %
8.08374269822
```

In [10]:
```
# test some random values
num_to_test = gen_rand_positive_32_bit_float()
print("Number to test: {}".format(num_to_test))

# my computation
print("My computation")
my_result = newton_sqrt(num_to_test, max_steps=6, min_error=1e-200, add_noise=
False, verbose=True)

# python's computation
python_result = math.sqrt(num_to_test)

# compare
print("")
print("my results: {0:.100f}".format(my_result))
print("python's  : {0:.100f}".format(python_result))
my_error = abs(my_result*my_result-num_to_test)
print("Newton Absolute Error: {}".format(my_error))
print("Newton Relative Error: {:.5f} %".format(100.0*my_error/num_to_test))

py_error = abs(python_result*python_result-num_to_test)
print("Python Absolute Error: {}".format(py_error))
print("Python Relative Error: {:.5f} %".format(100.0*py_error/num_to_test))
```

```
Number to test: 1125425573.0
My computation
Init guess: 29472.6557683
Absolute Error: 256788134.964
Relative Error: 22.8169806271 %
After step 1
Current x: 33829.0350675
Absolute Error: 18978040.5986
Relative Error: 1.6862990369 %
After step 2
Current x: 33548.5357781
Absolute Error: 78679.8513834
Relative Error: 0.0069911199 %
After step 3
Current x: 33547.3631509
Absolute Error: 1.37505459785
Relative Error: 0.0000001222 %
After step 4
Current x: 33547.3631304
Absolute Error: 0.0
Relative Error: 0.0000000000 %

my results: 33547.363130356461624614894390106201171875000000000000000000000000
0000000000000000000000000000000000000000000000
python's  : 33547.363130356461624614894390106201171875000000000000000000000000
0000000000000000000000000000000000000000000000
Newton Absolute Error: 0.0
Newton Relative Error: 0.00000 %
Python Absolute Error: 0.0
Python Relative Error: 0.00000 %
```

In [11]:
```
%%timeit -n 100000 -r 5
math.sqrt(num_to_test)
```

```
100000 loops, best of 5: 90.9 ns per loop
```

In [12]:
```
%%timeit -n 100000 -r 5
newton_sqrt(num_to_test, max_steps=0)
```

```
100000 loops, best of 5: 1.27 µs per loop
```

In [13]:
```
%%timeit -n 100000 -r 5
newton_sqrt(num_to_test, max_steps=1)
```

```
100000 loops, best of 5: 1.81 µs per loop
```

In [14]:
```
%%timeit -n 100000 -r 5
newton_sqrt(num_to_test, max_steps=2)
```

```
100000 loops, best of 5: 2.5 µs per loop
```

In [15]:
```
%%timeit -n 100000 -r 5
newton_sqrt(num_to_test, max_steps=3)
```

```
100000 loops, best of 5: 3.59 µs per loop
```

In [16]:
```
%%timeit -n 100000 -r 5
newton_sqrt(num_to_test, max_steps=4)
```

```
100000 loops, best of 5: 4.2 µs per loop
```

In [17]:
```
%%timeit -n 100000 -r 5
newton_sqrt(num_to_test, max_steps=5)
```

```
100000 loops, best of 5: 4.81 µs per loop
```

In [18]:
```
# seems about 10 times slower with maxstep=0
```

In [19]:
```
# test over a bunch of floats, to evaluate the quality of my init_guess

for max_steps in range(10):
    print("Testing with max_steps={}".format(max_steps))

    max_abs_error = -1
    max_rel_error = -1

    for _ in xrange(100000):
        # generate a new random value
        num_to_test = gen_rand_positive_32_bit_float()

        # compute my sqrt
        my_result = newton_sqrt(num_to_test, max_steps=max_steps, min_error=1e
-100, add_noise=False, verbose=False)

        my_abs_error = abs(my_result*my_result-num_to_test)
        my_rel_error = 100.0*my_abs_error/num_to_test

        if (max_abs_error == -1) or (my_abs_error>max_abs_error):
            max_abs_error = my_abs_error
        if (max_rel_error == -1) or (my_rel_error>max_rel_error):
            max_rel_error = my_rel_error

    print("Max absolute error: {}".format(max_abs_error))
    print("Max relative error: {:.30f} %".format(max_rel_error))
    print('')
```

```
Testing with max_steps=0
Max absolute error: 507629828.824
Max relative error: 25.107617879520933712456098874100 %

Testing with max_steps=1
Max absolute error: 24263318.7397
Max relative error: 2.104142690993499709151137722074 %

Testing with max_steps=2
Max absolute error: 116437.211519
Max relative error: 0.010844023993725148552469406127 %

Testing with max_steps=3
Max absolute error: 3.15295863152
Max relative error: 0.000000293941048060200630241061 %

Testing with max_steps=4
Max absolute error: 4.76837158203e-07
Max relative error: 0.000000000000004010323603699587 8 %

Testing with max_steps=5
Max absolute error: 2.38418579102e-07
Max relative error: 0.000000000000022202247074449589 %

Testing with max_steps=6
Max absolute error: 2.38418579102e-07
Max relative error: 0.000000000000022203882535773568 %

Testing with max_steps=7
Max absolute error: 2.38418579102e-07
Max relative error: 0.000000000000022203929620678561 %

Testing with max_steps=8
Max absolute error: 2.38418579102e-07
Max relative error: 0.000000000000022204176380277577 %

Testing with max_steps=9
Max absolute error: 2.38418579102e-07
Max relative error: 0.000000000000022204416941529298 %
```

In [ ]: