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 12-13% of target sqrt
# 0.6-0.7% 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 v
alue 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 optim
ized 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=False):
    # 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
    # after 3 steps, 0% error

    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]:

```python
def gen_rand_positive_32_bit_float():
    # hopefully a rand value over all the positive 32 bit floats?

    exponent_bits = [random.choice([0,1]) for _ in xrange(8)]
    mantissa_bits = [random.choice([0,1]) for _ in xrange(23)]

    # exponent, random int between -127 and 127
    exponent = 0
    for e,b in enumerate(exponent_bits[1:]):
        exponent += b*(2**e)
    # first bit defines exponent sign
    exponent *= (2*exponent_bits[0]-1)

    # mantissa
    mantissa = 1
    for e,b in enumerate(mantissa_bits):
        mantissa += b*(2**-e)

#     print(mantissa_bits)
#     print(exponent_bits)
#     print(mantissa, exponent)

    # always positive
    r = mantissa*(2**exponent)

    return r
```

In [21]:

```python
# test some known values
num_to_test = 100
print(newton_sqrt(num_to_test))
```

```
10.0005472261
```

In [9]:

```python
# 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: 3.96595994124e-24
My computation
Init guess: 1.82964175434e-12
Absolute Error: 6.18370992004e-25
Relative Error: 15.5919626311 %
After step 1
Current x: 1.99862865862e-12
Absolute Error: 2.8556573817e-26
Relative Error: 0.7200419127 %
After step 2
Current x: 1.99148461671e-12
Absolute Error: 5.10373348747e-29
Relative Error: 0.0012868848 %
After step 3
Current x: 1.99147180282e-12
Absolute Error: 1.64195989659e-34
Relative Error: 0.0000000041 %
After step 4
Current x: 1.99147180277e-12
Absolute Error: 7.34683969264e-40
Relative Error: 0.0000000000 %
After step 5
Current x: 1.99147180277e-12
Absolute Error: 7.34683969264e-40
Relative Error: 0.0000000000 %
After step 6
Current x: 1.99147180277e-12
Absolute Error: 7.34683969264e-40
Relative Error: 0.0000000000 %

my results: 0.0000000000019914718027742026441318161048662081671727086540357731
79202806204557418823242187500000000000
python's  : 0.0000000000019914718027742030480285995780242525379777340788223227
7188450098037719726562500000000000000
Newton Absolute Error: 7.34683969264e-40
Newton Relative Error: 0.00000%
Python Absolute Error: 7.34683969264e-40
Python Relative Error: 0.00000%
```

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

```
100000 loops, best of 5: 91.2 ns per loop
```

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

```
100000 loops, best of 5: 1.11 µs per loop
```

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

```
100000 loops, best of 5: 1.73 µs per loop
```

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

```
100000 loops, best of 5: 2.8 µs per loop
```

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

```
100000 loops, best of 5: 3.31 µs per loop
```

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

```
100000 loops, best of 5: 4.39 µs per loop
```

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

```
100000 loops, best of 5: 5.59 µs per loop
```

In [18]:
```
# seems about 20 times slower with maxstep=1
```

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

num_samples = 1000000

for max_steps in range(6):
    print("Testing with max_steps={}".format(max_steps))

    max_abs_error = -1
    max_abs_err_num = -1
    max_abs_err_res = -1

    max_rel_error = -1
    max_rel_err_num = -1
    max_rel_err_res =-1

    max_sqrt_abs_error = -1
    max_sqrt_abs_err_num = -1
    max_sqrt_abs_err_res = -1

    max_sqrt_rel_error = -1
    max_sqrt_abs_err_num = -1
    max_sqrt_abs_err_res  -1

    average_abs_err = 0
    average_rel_err = 0

    for _ in xrange(num_samples):
        # 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

        sqrt_abs_error = abs(my_result-math.sqrt(num_to_test))
        sqrt_rel_error = 100.0*sqrt_abs_error/math.sqrt(num_to_test)

        average_abs_err += my_abs_error
        average_rel_err += my_rel_error

        if (max_abs_error == -1) or (my_abs_error>max_abs_error):
            max_abs_error = my_abs_error
            max_abs_err_num = num_to_test
            max_abs_err_res = my_result*my_result

        if (max_rel_error == -1) or (my_rel_error>max_rel_error):
            max_rel_error = my_rel_error
            max_rel_err_num = num_to_test
            max_rel_err_res = my_result*my_result

        if (max_sqrt_abs_error == -1) or (sqrt_abs_error>max_sqrt_abs_error):
            max_sqrt_abs_error = sqrt_abs_error
            max_sqrt_abs_err_num = math.sqrt(num_to_test)
```

```
            max_sqrt_abs_err_res = my_result

        if (max_sqrt_rel_error == -1) or (sqrt_rel_error>max_sqrt_rel_error):
            max_sqrt_rel_error = sqrt_rel_error
            max_sqrt_rel_err_num = math.sqrt(num_to_test)
            max_sqrt_rel_err_res = my_result

    average_abs_err /= num_samples
    average_rel_err /= num_samples

    print("Max absolute error: {} (result: {}  target: {})".format(max_abs_err
or, max_abs_err_res, max_abs_err_num))
    print("Max relative error: {}% (result: {}  target: {})".format(max_rel_er
ror, max_rel_err_res, max_rel_err_num))
    print("Max sqrt absolute error: {} (result: {}  target: {})".format(max_sq
rt_abs_error, max_sqrt_abs_err_res, max_sqrt_abs_err_num))
    print("Max sqrt relative error: {}% (result: {}  target: {})".format(max_s
qrt_rel_error, max_sqrt_rel_err_res, max_sqrt_rel_err_num))
    print("Average absolute error: {}".format(average_abs_err))
    print("Average relative error: {}%".format(average_rel_err))
    print('')
```

```
Testing with max_steps=0
Max absolute error: 8.54318421654e+37 (result: 2.54887966084e+38  target: 3.4
0319808249e+38)
Max relative error: 25.1085237642% (result: 4.25709668749e-14  target: 5.6843
5408336e-14)
Max sqrt absolute error: 2.48254777573e+18 (result: 1.59652111193e+19  targe
t: 1.8447758895e+19)
Max sqrt relative error: 13.4601385281% (result: 2.06327329443e-07  target:
2.38418834897e-07)
Average absolute error: 3.2547308125e+35
Average relative error: 12.6443008656%

Testing with max_steps=1
Max absolute error: 7.15771615647e+36 (result: 3.47490302324e+38  target: 3.4
0332586167e+38)
Max relative error: 2.10452501718% (result: 0.510522625086  target: 0.5)
Max sqrt absolute error: 1.92986549665e+17 (result: 1.86410917686e+19  targe
t: 1.84481052189e+19)
Max sqrt relative error: 1.04678372773% (result: 0.71450865991  target: 0.707
106781187)
Average absolute error: 1.51072690122e+34
Average relative error: 0.605191753169%

Testing with max_steps=2
Max absolute error: 3.68135084371e+34 (result: 3.40423918793e+38  target: 3.4
0387105284e+38)
Max relative error: 0.0108442515549% (result: 64.0070013628  target: 64.00006
10352)
Max sqrt absolute error: 9.97651641741e+14 (result: 1.8450580446e+19  target:
1.84495827943e+19)
Max sqrt relative error: 0.00542197878817% (result: 8.00043757321  target: 8.
0000038147)
Average absolute error: 3.9841555815e+31
Average relative error: 0.0017124450769%

Testing with max_steps=3
Max absolute error: 9.96085135454e+29 (result: 3.40369906797e+38  target: 3.4
0369905801e+38)
Max relative error: 2.9396692684e-07% (result: 17179871282.5  target: 1717987
1232.0)
Max sqrt absolute error: 26995470336.0 (result: 1.84491166942e+19  target: 1.
84491166672e+19)
Max sqrt relative error: 1.46983461079e-07% (result: 131072.008005  target: 1
31072.007812)
Average absolute error: 5.53729386455e+26
Average relative error: 2.36442946349e-08%

Testing with max_steps=4
Max absolute error: 1.51115727452e+23 (result: 4.32647040487e+38  target: 4.3
2647040487e+38)
Max relative error: 4.42161718457e-14% (result: 4.52323192288e+15  target: 4.
52323192288e+15)
Max sqrt absolute error: 4096.0 (result: 2.07109436902e+19  target: 2.0710943
6902e+19)
Max sqrt relative error: 2.22042275621e-14% (result: 3.05178982655e-05  targe
t: 3.05178982655e-05)
Average absolute error: 2.16277722264e+20
```

```
Average relative error: 8.15315896066e-15%

Testing with max_steps=5
Max absolute error: 7.55578637259e+22 (result: 4.8917930835e+38  target: 4.89
17930835e+38)
Max relative error: 2.22044393167e-14% (result: 2.35099094372e-38  target: 2.
35099094372e-38)
Max sqrt absolute error: 4096.0 (result: 2.02234731734e+19  target: 2.0223473
1734e+19)
Max sqrt relative error: 2.22042725595e-14% (result: 0.062500528989  target:
0.062500528989)
Average absolute error: 2.10425059926e+20
Average relative error: 7.84093481805e-15%
```