Homework 1

Dan Kolbman

September 19, 2014

1 Break math:

Rounding errors in Python's sqrt() function:

```
import math
a = 25.0
print("Squaring a square-root:")
while ( math.sqrt(a)**2 == a ):
  print('sqrt(a)^2 = ' + str(a) + ' = ' + str(math.sqrt(a)**2))
  a *= 10
# There was a rounding error
print('sqrt(a)^2 = ' + str(a) + ' != ' + str(math.sqrt(a)**2))
# Determine the exponent of the float
expo = math.floor(math.log10(a))-1.0
# Reduce to only significant digits
b = a/(10**expo)
print("Ajusting decimal placement before taking the square-root:")
print('sqrt(a)^2 = ' + str(a) + ' = ' + str((math.sqrt(b)**2)*10**expo))
```

Output:

```
Squaring a square-root:
sqrt(a)^2 = 25.0 = 25.0
sqrt(a)^2 = 250.0 = 250.0
sqrt(a)^2 = 2500.0 = 2500.0
sqrt(a)^2 = 25000.0 != 25000.000000000004
Ajusting decimal placement before taking the square-root:
sqrt(a)^2 = 25000.0 = 25000.0
```

2 Matrix inversion via libraries and research:

Using numpy:

```
# Using numpy matrix module
import numpy as np
import numpy.matlib
# 5x5 random matrix
M = numpy.matlib.rand(3,3)
# The inverse
N = M.I
print("A matrix:")
print(M)
print("The inverse of that matrix:")
print(N)
# Gives the identity matrix
print("The two multiplied together (should be the identity matrix):")
print(M.dot(N))
```

Output:

```
A matrix:
[[ 0.22781309  0.29300554  0.35490115]
 [ 0.63443532  0.34759898  0.13994113]
 [ 0.74373481  0.87841762  0.54915384]]
The inverse of that matrix:
[[ 1.36106978  3.02114005 -1.64949583]
 [-4.89329541 -2.78083689  3.8710275 ]
 [ 5.98390049  0.35656511 -2.1370902 ]]
The two multiplied together (should be the identity matrix):
[[  1.00000000e+00   1.38777878e-16   1.11022302e-16]
 [  1.11022302e-16   1.00000000e+00   2.22044605e-16]
 [  0.00000000e+00   1.38777878e-16   1.00000000e+00]]
```

3 Binary and Floating point notation:

$10.5 = 0|10000000010|01010\overline{0}_2$

$\frac{1}{3} = 0|01111111101|\overline{01}_2$

$\frac{22}{7} = 0|10000000000|\overline{1001}_2$

4 Head-to-Head

Find zero of $f(x) = x^4 - 2$ using Bisection, Secant, False Position, and Fixed Point Iteration.

4.1 Bisection

```
def func(x):
  return x**4-2
def bisection(a, b,thresh, max_step, f):
  diffs = []
  itr = []
  x1 = a
  x2 = b
  x1_val = f(x1)
  x2_val = f(x2)
  if( x1_val < 0):
    x1, x2 = x1_val, x2_val
  else:
    x1, x2, = x2_val, x1_val
  dx = x2 - x1
  for i in range(max_step):
    dx /= 2
    mid = x1 + dx
    # Evaluate the midpoint
    fval = f(mid)
    itr.append(i)
    diffs.append(abs(fval))
    # Get out if our guess is good enough
    if(abs(fval) < thresh):
      return mid, itr, diffs
    # Assign midpoint
    x1 = x1 if (fval > 0) else mid
  return mid, itr, diffs
mid, itr, diffs = bisection(0, 2, 1e-10, 100, func)
```

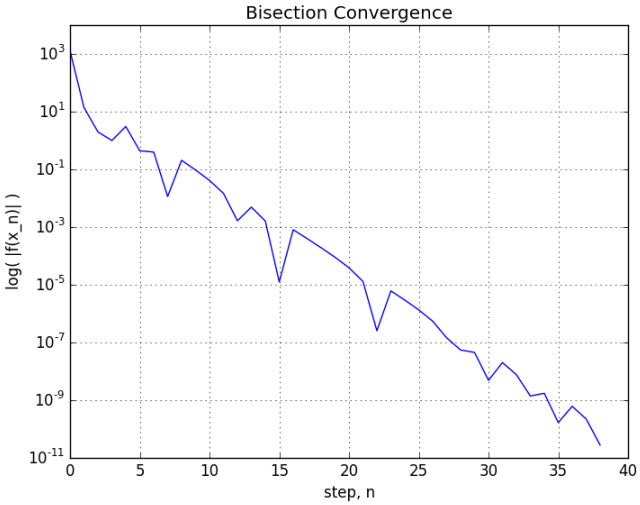

Figure 1: (Approximately) Quadratic Convergence

4.2 Secant Method

```
def func(x):
  return x**4 - 2
def secant(a, b, thresh, max_step, f):
  itr = []
  diffs = []
  x_old = a
  x_new = b
  x_old_val = f(x_old)
  x_new_val = f(x_new)
  for i in range(max_step):
    dx = (x_old - x_new) * x_new_val / ( x_new_val - x_old_val )
    x_old = x_new
    x_old_val = x_new_val
    x_new += dx
    x_new_val = f(x_new)
    itr.append(i)
    diffs.append(abs(x_new_val))
    if( abs(x_new_val) < thresh):
      return x_new, itr, diffs
  return x_new, itr, diffs
x, itr, diffs = secant(0, 2, 1e-10, 100, func)
```

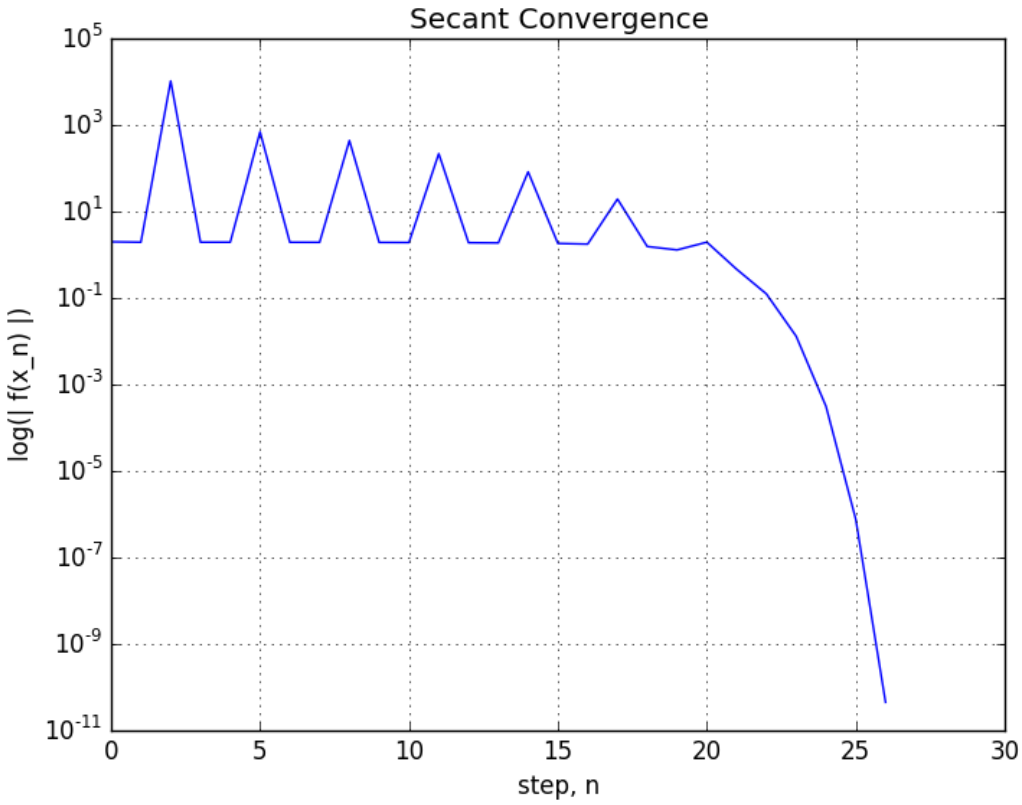

Figure 2: Takes a couple of iterations before the root is properly bracketted

4.3 False Position Method

```
def func(x):
  return x**4 - 2
def false_position(a, b, thresh, max_step, f):
  itr = []
  diffs = []
  x1 = a
  x2 = b
  val1 = f(x1)
  val2 = f(x2)
  # Swap values
  if( val1 > 0 ):
    x1, x2, val1, val2 = x2, x1, val2, val1
  for i in range(max_step):
    x_new = x1 + (x2 - x1)*val1/(val1 - val2)
    val_new = f(x_new)
    itr.append(i)
    diffs.append(abs(val_new))
    if( abs(val_new) < thresh):
      return x_new, itr, diffs
    if( val_new <  0 ):
      x1 = x_new
      val1 = val_new
    else:
      x2 = x_new
      val2 = val_new
  return x_new, itr, diffs
x, itr, diffs = false_position( 0, 2, 1e-10, 300, func)
```

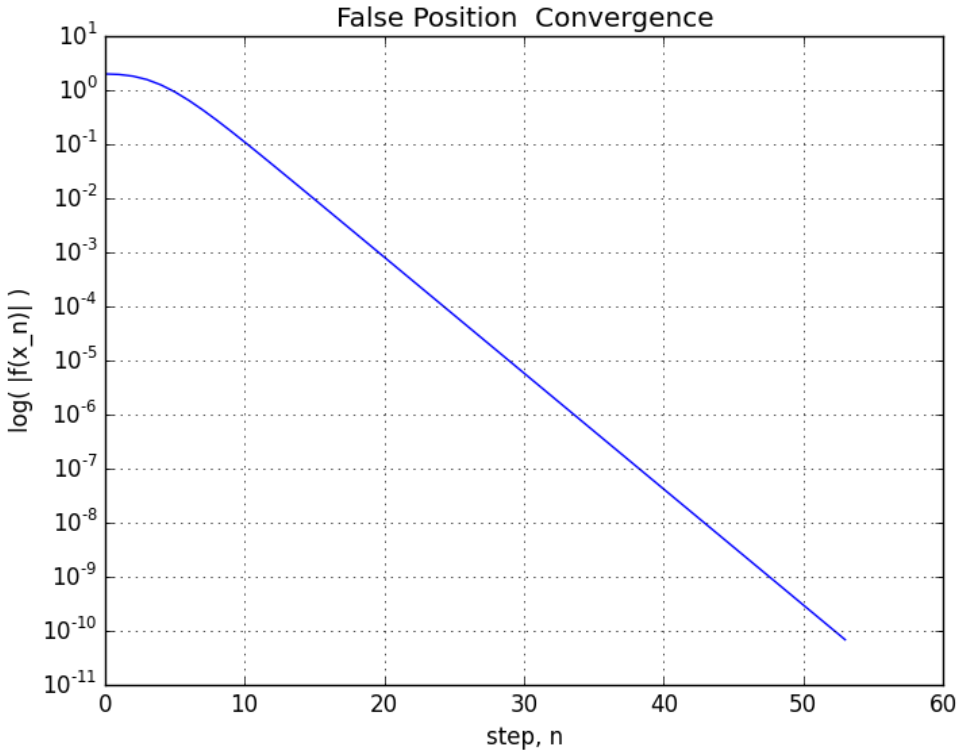

Figure 3: Converges quadratically

4.4 Fixed Point Iteration

```
def func(x):
  return x**4 - 2
def false_position(a, b, thresh, max_step, f):
  itr = []
  diffs = []
  x1 = a
  x2 = b
  val1 = f(x1)
  val2 = f(x2)
  # Swap values
  if( val1 > 0 ):
    x1, x2, val1, val2 = x2, x1, val2, val1
  for i in range(max_step):
    x_new = x1 + (x2 - x1)*val1/(val1 - val2)
    val_new = f(x_new)
    itr.append(i)
    diffs.append(abs(val_new))
    if( abs(val_new) < thresh):
      return x_new, itr, diffs
    if( val_new <  0 ):
      x1 = x_new
      val1 = val_new
    else:
      x2 = x_new
      val2 = val_new
  return x_new, itr, diffs
x, itr, diffs = false_position( 0, 2, 1e-10, 300, func)
```

Using these three differnt forms for $g(x)$:

$$g_1(x) = \frac{x}{2} + \frac{1}{x^3} \tag{1}$$

$$g_2(x) = \frac{2x}{3} + \frac{2}{3x^3} \tag{2}$$

$$g_3(x) = x - \frac{2}{5}(x^4 - 2) \tag{3}$$

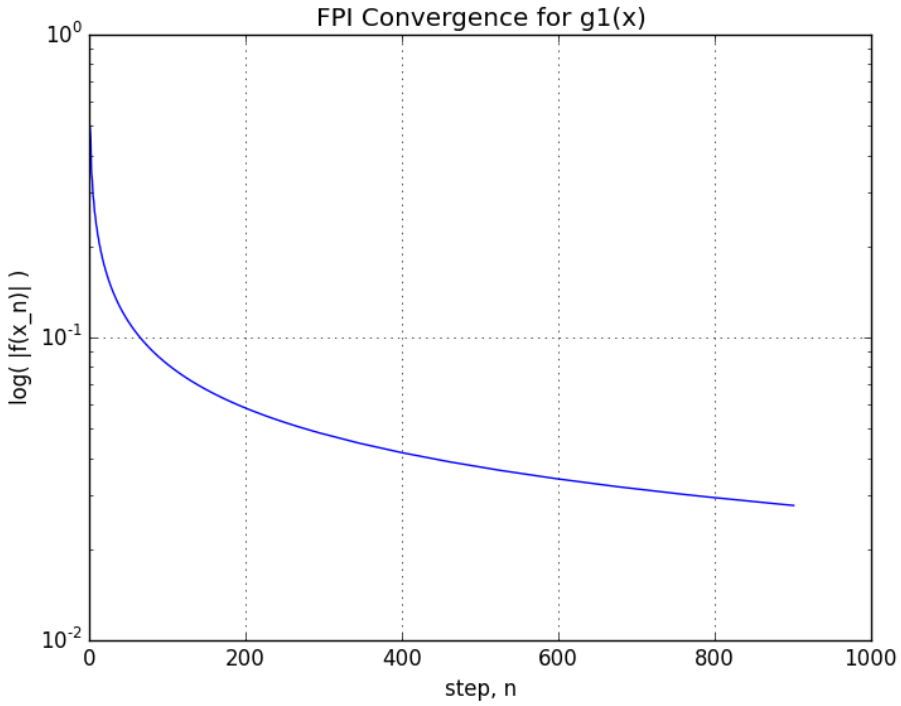

Figure 4: See Eq. 5. Converges to 0.02, not very precise

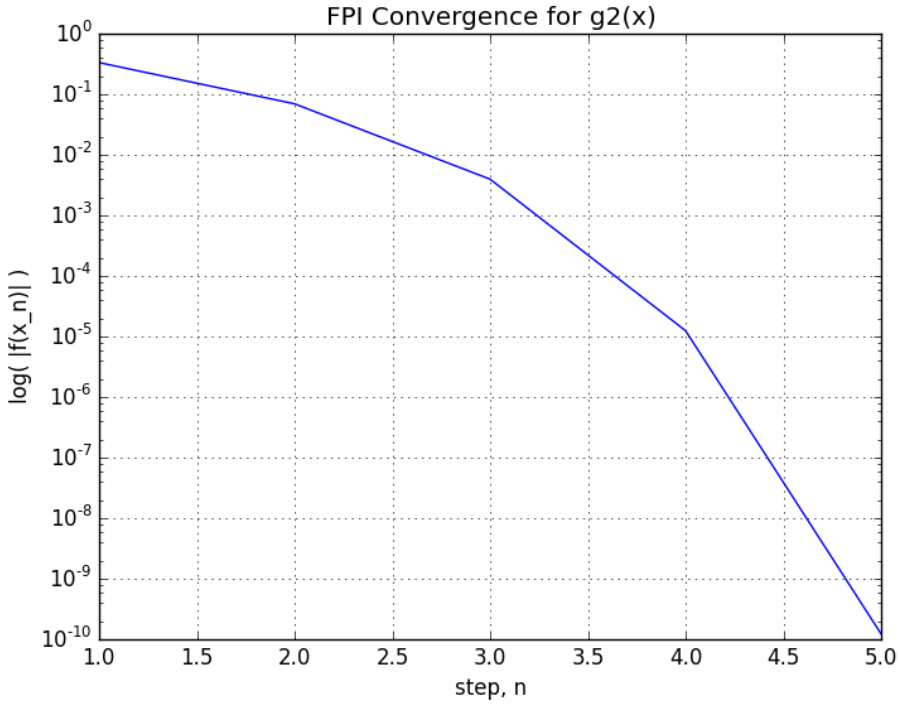

Figure 5: See Eq. 6. Converges increasingly faster and only takes 5 steps to hit desired precision

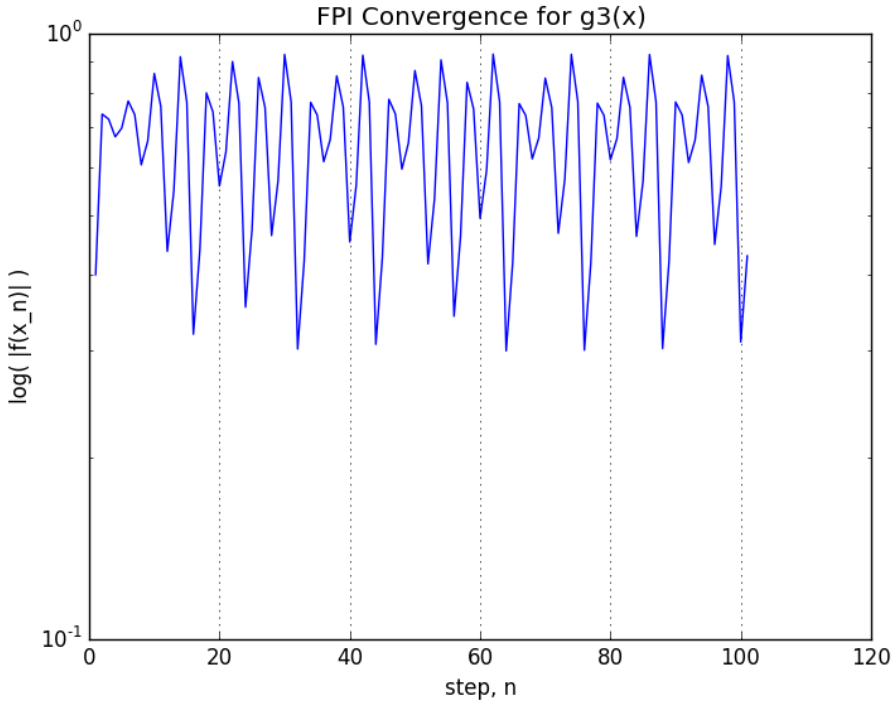

Figure 6: See Eq. 7. Oscillates around 0.5, not accurate or precise

5 More FPI

$$f(x) = 2x^3 - 6x - 1 \tag{4}$$

To find the roots of Eq. 4, we write $f(x)$ as $f(x) = x - g(x) = 0$ where possible forms of $g(x)$ are:

$$g_1(x) = \frac{1}{2x^2 - 6} \tag{5}$$

$$g_2(x) = \frac{3}{x} + \frac{1}{2x^2} \tag{6}$$

$$g_3(x) = \sqrt[3]{3x + \frac{1}{2}} \tag{7}$$

$$g_4(x) = x - \frac{(2x^3 - 6x - 1)}{(x^2 - 6)} \tag{8}$$

Equations Eq. 5, Eq. 6, and Eq. 7 give single roots of Eq. 4. All of theese equations converge quadratically.
$g_1(x)$ converges to -0.16825440
$g_2(x)$ converges to -1.64177875
$g_3(x)$ converges to 1.8100379292

Eq. 8 converges to all three roots super-quadratically for appropriate approximations:
$g_4(x)$ converges to -0.16825440 for $x_0 = -0.1$
$g_4(x)$ converges to -1.64177875 for $x_0 = -1.1$
$g_4(x)$ converges to 1.8100379292 for $x_0 = 2$

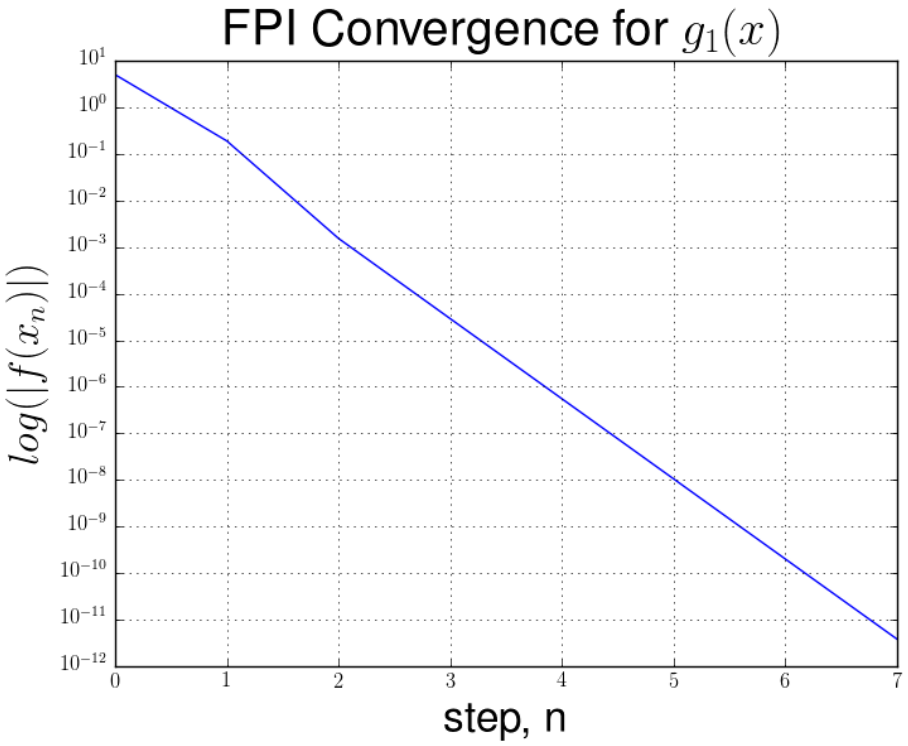

Figure 7: See Eq. 5

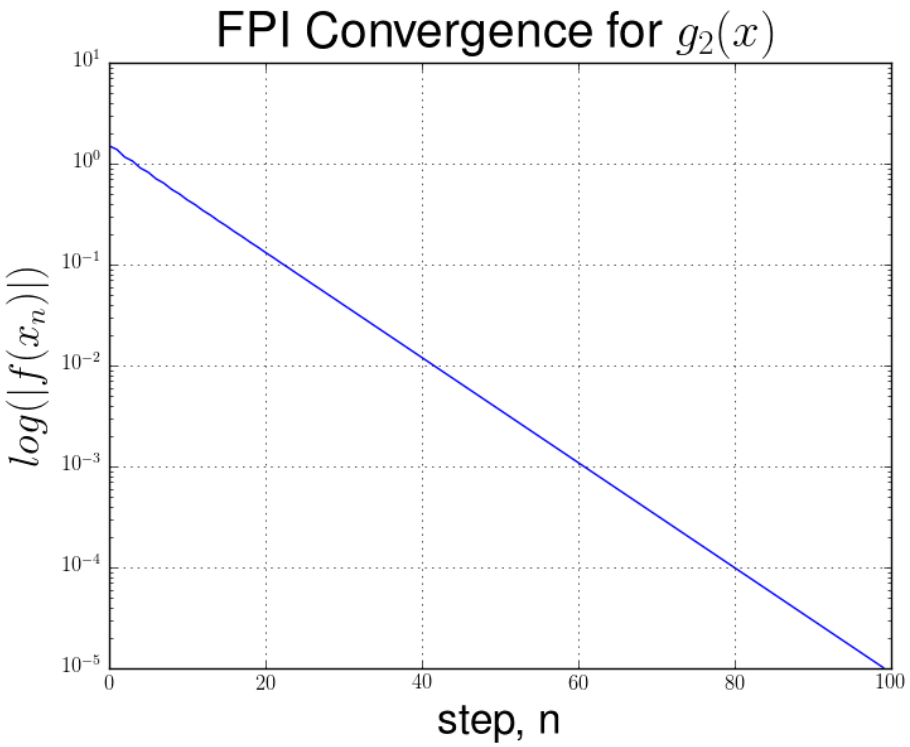

Figure 8: See Eq. 6

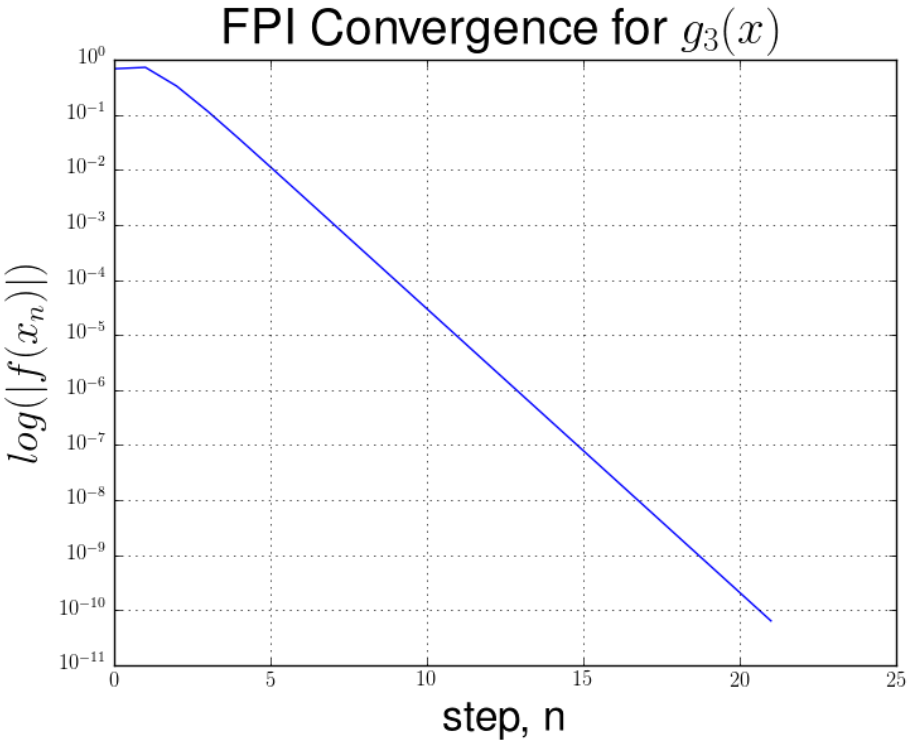

Figure 9: See Eq. 7

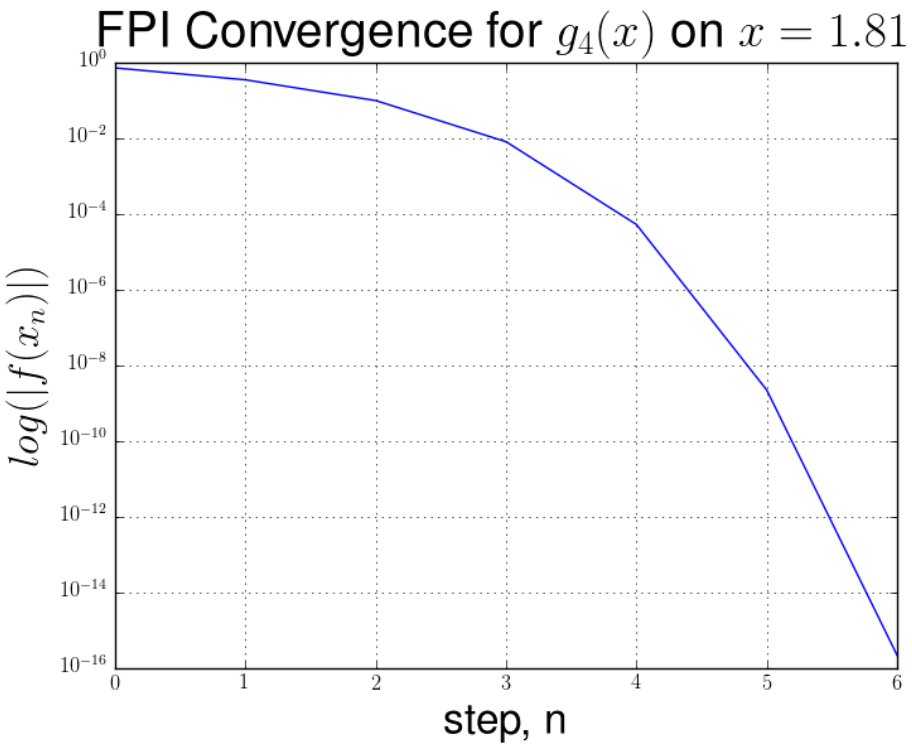

Figure 10: Eq. 8 for $x_0 = 2$

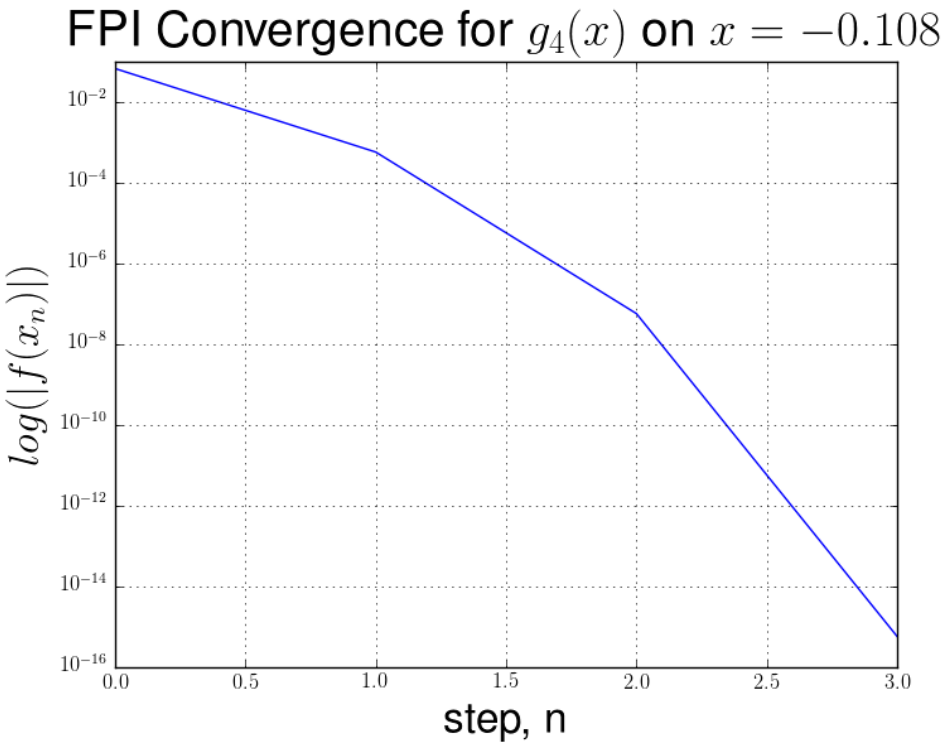

Figure 11: Eq. 7 for $x_0 = -0.1$

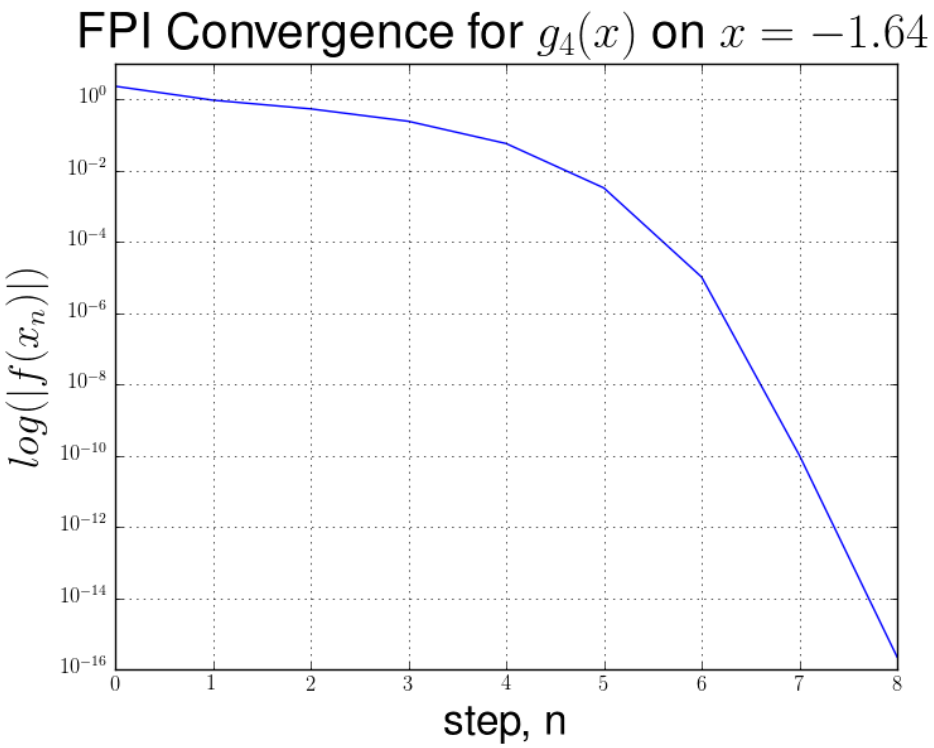

Figure 12: Eq. 7 for $x_0 = -1.1$

6 Secants behaving badly

By attempting to find the root of $\Omega_1 - \Omega_2$ with the secant method, its possible that the brackets may try to converge on one of the two asymptotes. An better method of finding the root of this equation is to take the inverse and solve for its root. The inverse will asymptote on the zero of the function, but the secant method will still converge on this value. See figure 6.

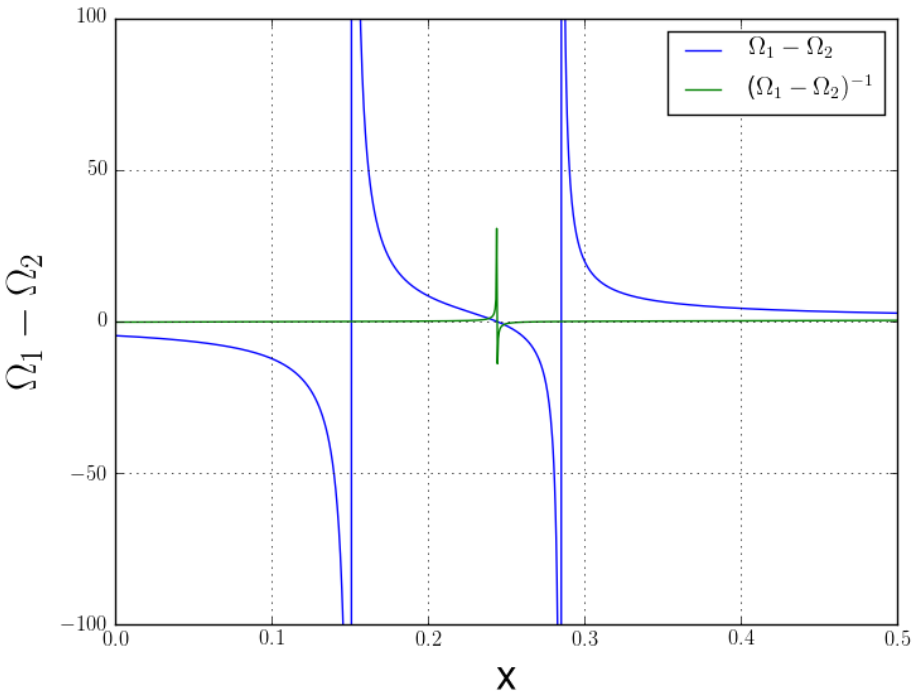

Figure 13: The difference of the two functions results in three possible values to converge on for the secant method where their inverse gives only the correct one.

7 Convergence, or the lack thereof

```
import math
def expand(x,thresh):
  # First term is 1
  approx = 1.0
  last_term = 1.0
  for n in range(0,100):
    new_term = last_term*3*x/(n+1)
    approx += new_term
    #print(abs(approx - math.exp(3*x)))
    #if(abs(approx - math.exp(3*x)) < thresh):
    if(abs(new_term - last_term) < thresh):
      break
    last_term = new_term
  print('terms: '+str(n), 'value: '+str(approx))
```

Running until the error with the known value is less 10^{-9}:

```
Until relative error is less than 10^-9
x = 2
terms: 34 value: 403.4287934927351
x = -2
terms: 99 value: 0.002478752176668117
x = -12
terms: 99 value: 0.05229714454636593
```

Running until the relative error between terms n and $n+1$ is less than 10^{-15}:

```
Until difference between n+1 and n term is less than 10^-15
x = 2
terms: 5 value: 244.60000000000002
x = -2
terms: 38 value: 0.0024787521766681028
x = -12
terms: 99 value: 0.05229714454636593
```

For $x = 2$, the difference in terms quickly diminishes before a reasonable approximation can be made. For $x = -2$, the relative error never reaches the threshold where the difference in terms only makes 39 steps and still produces a close approximation.
For $x = -12$, both methods hit the step limit and produce the same approximation.

```
x = 20
terms: 99 value: 1.1420063979378045e+26
x = -20
terms: 99 value: 2.614878889275148e+19
```

The precision for $x_0 = 20$ is not particularly good, though its accuracy indicates that it will eventually converge to a more precise answer. For larger magnitudes of x_0, many more iterations are required until the factorial in the denominator reaches a large enough number that will result in small terms capable of converging on a precise value. For $x_0 = -20$, this problem is even more apparant as each term will alternate between positive and negative. The terms can also decay in size very quickly resulting in an approximation that decays more and more slowly away from a very innacurate estimate, as is in the case of $x_0 = -20$.

8 Multidimensional Newton's Method

$$B = 0.04,\ S = 0.0011111,\ Q = 0.083333,\ D = 1.29074,\ N = -1.774598$$

$$V(B+W)^2 - S\frac{B+2W}{W^2} = Q \tag{9}$$

$$-\frac{B}{2}(1+V) + \frac{S}{2}W^{-2} - W + \frac{1}{2}\left[(1-V)W - D\sqrt{1-V}\right] = N \tag{10}$$

1D Newtown-Raphson

Solve Eq. 9 for $V(W)$:

$$V = \frac{Q}{(B+W)^2} + S\frac{B+2W}{W^2(B+W)^2} \tag{11}$$

And substitute Eq. 11 into Eq. 10 and set it to 0:

$$-\frac{B}{2}(1 + \frac{Q}{(B+W)^2} + S\frac{B+2W}{W^2(B+W)^2}) + \frac{S}{2}W^{-2} - W$$
$$+\frac{1}{2}\left[(1-V)W - D\sqrt{1 - \frac{Q}{(B+W)^2} + S\frac{B+2W}{W^2(B+W)^2}}\right] - N = 0 \tag{12}$$

Using this technique, we obtain:

$$W = 2.1918669$$

$$V = 0.0169347$$

Where the solution for W converges after only 3 steps due to the near linearity of Eq. 12.

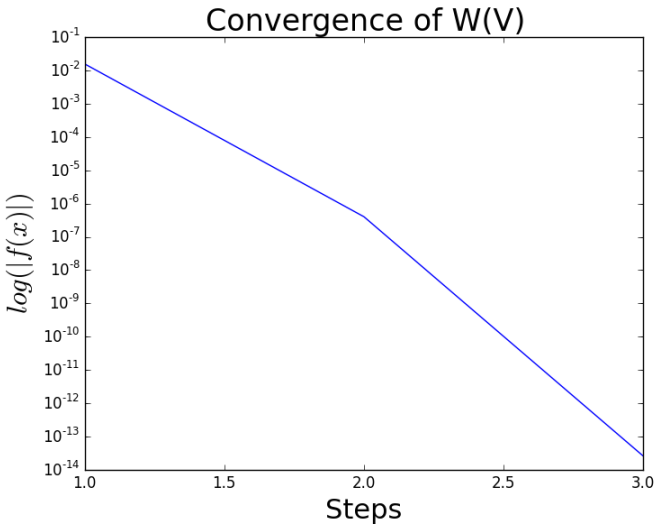

Figure 14: Very fast convergence because of the linearity of W(V)

2d Newton-Raphson

First, define the Jacobian:

$$
\begin{aligned}
J_{11} &= (B+W)^2) \\
J_{12} &= \frac{2(B+W)(S+VW^3)}{W^3} \\
J_{21} &= \frac{1}{4}\left[-2B + \frac{D}{\sqrt{1-V}} - 2W\right] \\
J_{22} &= -\frac{S}{W^3} - \frac{V}{2} - \frac{1}{2}
\end{aligned}
$$

9 Blackbody Radiation

$$
\begin{aligned}
e^{h\nu/k_BT} &= 1 + \frac{h\nu}{k_BT} + \cdots \\
u(\nu, T) &= \frac{dU}{d\nu} = \frac{8\pi h\nu^3}{c^3}\frac{1}{e^{h\nu/k_BT} - 1} \\
u(\nu, T) &= \frac{dU}{d\nu} = \frac{8\pi h\nu^3}{c^3}\frac{k_BT}{h\nu} \\
u(\nu, T) &= \frac{dU}{d\nu} = \frac{8\pi\nu^2 k_BT}{c^3}
\end{aligned} \tag{13}
$$

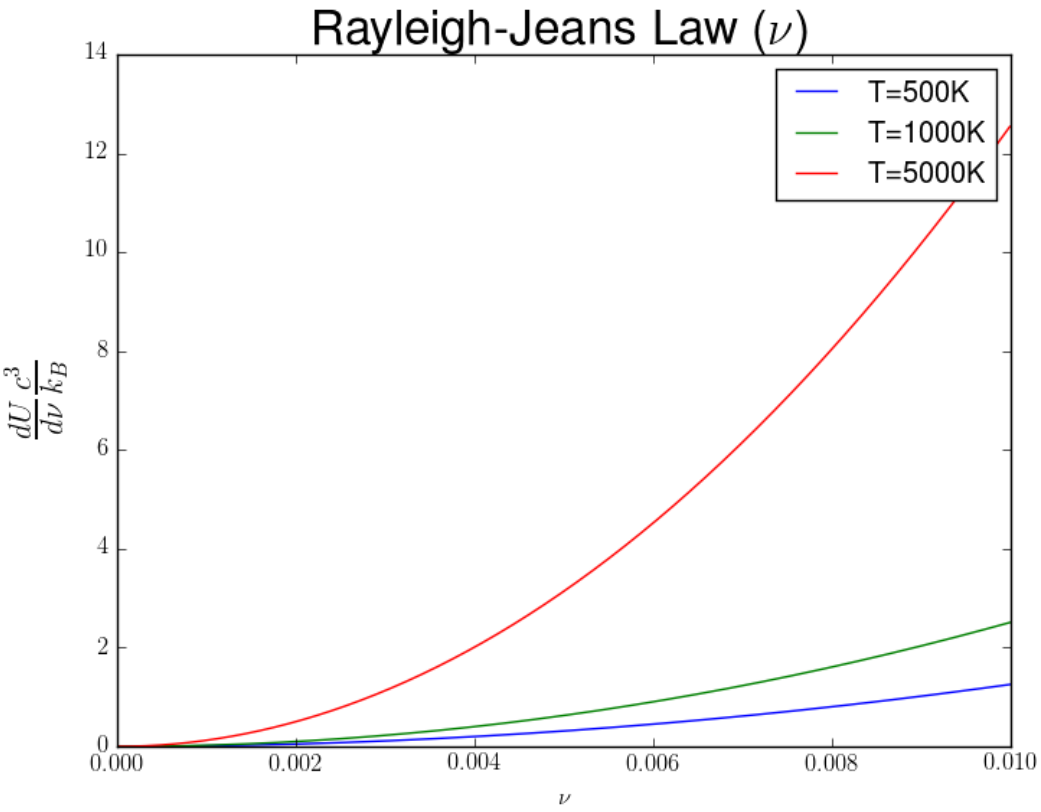

Figure 15: See Eq 13

Or in terms of λ, where $\lambda \equiv \frac{c}{\nu}$

$$\begin{aligned}
\frac{dU}{d\nu}\frac{d\nu}{d\lambda} &= \frac{d\nu}{d\lambda}\left[\frac{8\pi h\nu^3}{c^3}\frac{1}{e^{h\nu/k_BT}-1}\right] \\
\frac{d\nu}{d\lambda} &= \frac{d}{d\lambda}\left(\frac{c}{\lambda}\right) = -\frac{c}{\lambda^2} \\
\frac{dU}{d\lambda} &= \left(-\frac{c}{\lambda^2}\right)\left[\frac{8\pi h}{\lambda^3}\frac{1}{e^{hc/k_BT\lambda}-1}\right] \\
\frac{dU}{d\lambda} &= \left(-\frac{c}{\lambda^2}\right)\left[\frac{8\pi h}{\lambda^3}\frac{k_BT\lambda}{hc}\right] \\
\frac{dU}{d\lambda} &= \frac{8\pi k_BT}{\lambda^4}
\end{aligned} \tag{14}$$

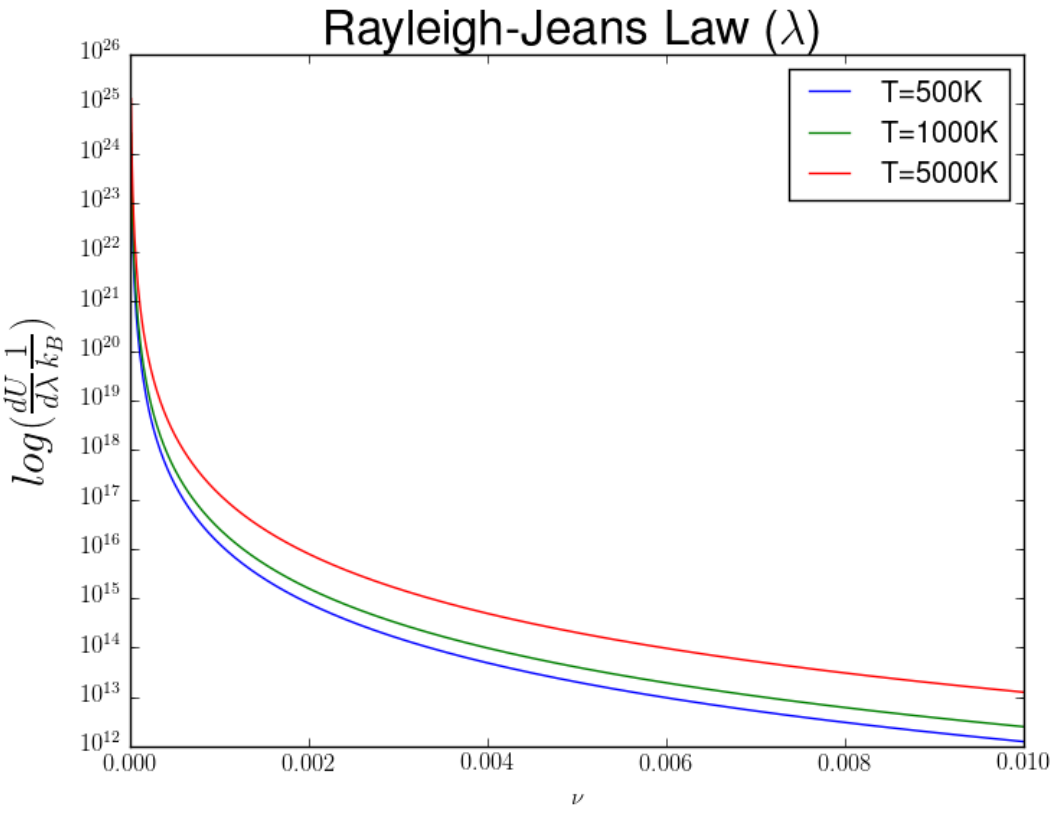

Figure 16: See Eq 14

$$u(x,\hat{T}) = \frac{8\pi h\nu_0^3}{c^3}\hat{T}^3\frac{x^3}{e^x-1} \tag{15}$$

Where $\hat{T} \equiv \frac{T}{T_0}$, $\hat{\nu} \equiv \frac{\nu}{\nu_0}$, $\hat{\lambda} \equiv \frac{\lambda}{\lambda_0}$ and $x \equiv \frac{\hat{\nu}}{\hat{T}} = \frac{h\nu}{k_B T}$

$$\frac{du}{dx} = \frac{8\pi h\nu_0^3}{c^3}\hat{T}^3\frac{x^2(e^x(x-3)+3)}{(e^x-1)^2} \tag{16}$$

$$\frac{du}{dx} = kT^3\frac{x^2(e^x(x-3)+3)}{(e^x-1)^2} \tag{17}$$

$$k = \frac{8\pi h\nu_0^3}{c^3T_0^3} = 5.585\times10^{-27}$$

To find the maximum of Eq. 15, we set Eq. 17 to zero and use the secant method to find the root. To approximate boundaries, the function is sketched to show that 1 and 4 and good positions for brackets. The secant method converges on the root after 25 iterations for these brackets.

```
itr:  0 x:  3.160258422268502  f(x):  0.13281574018461156
itr:  1 x:  2.5669465967777123 f(x):  -0.12032394272508262
itr:  2 x:  2.8489632971539063 f(x):  0.01200787006354061
itr:  3 x:  2.8233729283491273 f(x):  0.0008503145597232478
itr:  4 x:  2.82142269263881   f(x):  -7.339514025156768e-06
itr:  5 x:  2.821439382097633  f(x):  4.389544911587157e-09
itr:  6 x:  2.8214393721221303 f(x):  2.2598234809539056e-14
```

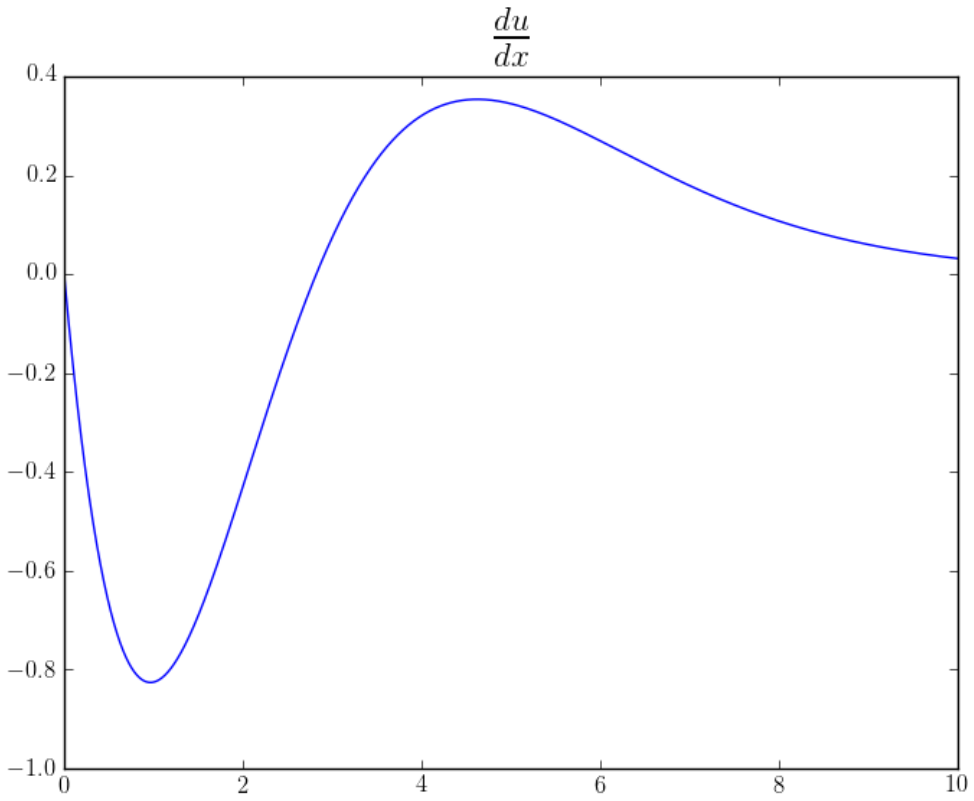

Figure 17: See Eq 17

Thus Eq. 15 is at a maximum for $x = 2.821439$.
To find the values of x at FWHM, x_h, we solve for $u(x_h; T)$ and set the equation to zero. The two roots can then be found as the values of x_h.

$$\frac{u(x_h;T)}{u(x_{max};T)} = \frac{1}{2}$$

$$\frac{u(x_h;T)}{u(x_{max};T)} - \frac{1}{2} = 0 \tag{18}$$

$$\tag{19}$$

Using the secant method to find roots between the brackets $(1, 3)$ and $3, 6$, the points for the FWHM where found to be:

$$x_h = \{1.157464, 5.405507\}$$