EGR 103L – Fall 2019

Lab 5: Structured Programming II

Marcus Deans (md374)
***Lab Section 2, Tuesdays 11:45 - 2:35 ***
13 October 2019

I understand and have adhered to all the tenets of the Duke Community Standard in completing every part of this assignment. I understand that a violation of any part of the Standard on any part of this assignment can result in failure of this assignment, failure of this course, and/or suspension from Duke University.

Contents

1 P&E 4.46 **2**

2 P&E 4.49 **2**

3 Chapra 4.25 **2**

4 Finding Roots **2**

A Codes **4**
A.1 swap_letters.py 4
A.2 scrambler.py 5
A.3 cos_series.py 6
A.4 poly_root.py 8

B Figures **9**

List of Figures

1 Figures for Cosine Approximation 9
2 Function and Derivative 10
3 Root Estimates and Iteration Count 11
4 Root Estimates and Iteration Count - As Image 12

1 P&E 4.46

```
Dolore etincidunt 2018 adipisci magnam eius numquam ipsum.
Wloliv vgrmxrwfmg 2018 zwrkrhxr nztmzn vrfh mfnjfzn rkhfn.
Quisquam etincidunt est non (voluptatem) porro aliquam.
Jfrhjfzn vgrmxrwfmg vhg mlm (elofkgzgvn) kliil zorjfzn.
Non ipsum "non" dolore.
Mlm rkhfn "mlm" wloliv.
Amet dolor sit quaerat!
Znvg wloli hrg jfzvizg!
```

2 P&E 4.49

```
Duke Utrisneivy is a cinmoumty ditcdeead to shoiharslcp, lesrdheiap,
and sicreve and to the pcirepinls of hetsony, fniraess, rseecpt, and
alubnitacitocy. Cznteiis of tihs cmonimtuy cmiomt to rlfeect uopn and
uophld tshee prpiinelcs in all aaceimdc and nemdanciaoc evreadnos, and
to prcotet and prtomoe a crultue of ieigrntty.

To ulohpd the Dkue Cotunmimy Stanrdad:

I wlil not lie, ceaht, or stael in my acmeadic evonraeds;
I wlil cunocdt meyslf horonbaly in all my eradonves; and
I will act if the Sdanratd is crmopomiesd.
```

3 Chapra 4.25

The graphs for the cos series approximator were very interesting, particularly in comparing the graphs representing identical angles but at different positions along the function. While these graphs would, in an ideal and properly coded program, be identical, the results were anything but. For the graphs $\pi/4$ and $1.02\pi/2$, the approximation value "balanced" out after several terms to yield similar approximations with each successive term (achieved around the fourth term). By the time the calculations were stopped as a result of the relative error reaching the specified limit, the approximations were near the limit of improving their own accuracy. The values of the approximation themselves were highly similar to the actual value of the cosine value at that position. In sum, the approximation worked well for these two values.

Different results were achieved with the graphing of $9\pi/4$ and $41.02\pi/2$. For the graph of $9\pi/4$, the approximation initially was highly inaccurate and whiplashed to further positive and negative values. The approximation actually increased in distance from the origin for the first three terms. After the third term, however, the approximation increased in accuracy to yield similar values to that of the identical value $\pi/4$. The approximation similarly ceased when the relative error reached the threshold; this was in fact only met with a greater number of terms (22) compared to the preceding functions. The graph of $41.02\pi/2$ was perhaps the most interesting, the approximation actually increased in inaccuracy with each successive term and when the relative error threshold was met, the approximation was not accurate at all. It was also interesting to note that the number of terms required for the approximation to cease (by meeting the thresholds) increased with a greater input value (regardless of it being the same angle as a prior input value). The number of terms in each approximation increased from $\sim$9 to $\sim$12 to $\sim$22 and finally to $\sim$30.

4 Finding Roots

I found the graphs of the roots to be interesting, particularly for a function that, at first glance, appears to be quite simple. The Map to Find Roots graph clearly illustrates the relation between x_k and x_{k+1}. It was very interesting to see the Root Estimate graph; unexpectedly, there were not three distinct domains in which the initial guesses corresponded to the same actual root. These existed only for three sections of the domain; there were also four other points at which a certain initial guess would yield a root that would also be identified using a different initial guess along the domain. Specifically, an initial guess between 0 and 1 would yield a root of 1.0, and an initial guess of 3 would **also** yield a root of 1.0. However, a guess between 1 and 3 could yield either 1.0, 2.0 or 4.0 as the root. This was unexpected based only on the code. I expected three separate segments that would comprise a piecewise

function, with steps between each level. I was not expecting such a degree of variability and fluctuation in the obtained root over very similar initial guess values.

The iteration count graph, while initially confusing, was much more comprehensible when related to the root estimate graph. Around the more complex initial guess values - those for which a greater number of possible actual roots could be obtained from x-values similar to that of the initial guess (1.5, for example) - there was a marked increase in the number of iterations necessary to yield the correct root result. There was also a logical connection between the initial guess being very similar to the actual root value and the (lower) number of iterations necessary to yield that root, considering that the initial guess was relatively accurate.

The images were very interesting to see, particularly as I did not know that the graphs could be visualized in such a way. They seem to more simply communicate the result using a colour scheme, and they also effectively demonstrate the gradient of the results. The drawbacks appear to be in the lack of clarity when trying to find a specific result using such an image.

A Codes

A.1 swap_letters.py

```
# -*- coding: utf-8 -*-
"""
[Swap Letters]
[Marcus Deans]
[1 October 2019]

I understand and have adhered to all the tenets of the Duke Community Standard
in creating this code.
Signed: [md374]
"""

def code_message(old_sent, show=0):
    new=[]
    for z in range(0,len(old_sent)):
        new.append(code_letter(old_sent[z]))
    if(show==1):
        return old_sent + "\n" + ''.join(new)
    else:
        return ''.join(new)

def code_letter(char_in):
    char = ord(char_in)
    lower = []
    if char in ((list(range(32, 65)))+(list(range(91,97)))+(list(range(123,127)))):
       return chr(char)
    if char in range(65,91):
        for x in range(65,91):
            lower.append(x)
            if char == lower[x-65]:
                loc = x-65
        final = (25-loc)
        return chr(lower[final])
    if char in range(97,123):
        for x in range(97,123):
            lower.append(x)
            if char == lower[x-97]:
                loc = x-97
        final = (25-loc)
        return chr(lower[final])

if __name__=="__main__":
    print(code_message("Let's Go Duke EGR 103 Fall 2019!", 1))
```

A.2 scrambler.py

```
# -*- coding: utf-8 -*-
"""
[Scrambler]
[Marcus Deans]
[4 October 2019]

I understand and have adhered to all the tenets of the Duke Community Standard
in creating this code.
Signed: [md374]
"""
import numpy as np
#%%
def scramble_line(combined):

    indi = combined.split()
    divide = []
    for x in range(0, len(indi)):
        divide.append(scramble_word(indi[x]))
    return ' '.join(divide)
#%%
def scramble_word(words):
    length = len(words)
    punc = 0
    if(((words[length-1]).isalpha()))!=1:
        sword = words[1:(length-2)]
        punc = 1
    else:
        sword = words[1:(length-1)]
    lword = list(sword)
    np.random.shuffle(lword)
    first = words[0]
    if(punc==1):
        lword.append(words[length-2])
    last = words[length-1]
    lword.insert(0,first)
    lword.append(last)
#     print(lword)
    if((len(words))==1):
        lword=words
    return ''.join(lword)
#%%
if __name__ == "__main__":
    print(scramble_line("Duke University in Durham North Carolina."))
```

A.3 cos_series.py

```
# -*- coding: utf-8 -*-
"""
[Cos Series]
[Marcus Deans]
[1 October 2019]

Based on: Extended python version of Chapra Figure 4.2
Extended python version of Chapra Figure 4.2
From: Applied Numerical Methods with MATLAB
      for Engineers and Scientists, 4th ed
      Steven C. Chapra
      McGraw-Hill 2018
@author: DukeEgr93
v. 1.0, 9/21/2019

I understand and have adhered to all the tenets of the Duke Community Standard
in creating this code.
Signed: [md374]
"""

import math as m
import numpy as np

def calc_cos(x, es=0.0001, maxit=50):
    '''
    Maclaurin series of exponential function
      fx, ea, it, sollist, ealist = iter_meth(x, es, maxit)
    input:
      x = value at which series evaluated
      es = stopping criterion (default = 0.0001)
      maxit = maximum iterations (default = 50)
    output:
      fx = estimated value
      ea = approximate relative error (%)
      it = number of iterations
      sollist = estimated values at each iteration
      ealist = approcimate relative error (%) at each iteration
    '''

    # initialization
#    maxit+=3
    it = 0
    sol = 0
    ea = 100
    sollist = []
    ealist = []
    pm = -1.0

    # iterative calculation
    while True:
        pm = -pm
        solold = sol
        top = x**(2*it)
        bottom = m.factorial(2*it)
        sol += pm*(top/bottom)
```

```
56
57          it += 1
58          if sol != 0:
59              ea = abs((sol - solold)/sol)*100
60
61          sollist += [sol]
62          ealist += [ea]
63
64          if ea <= es or it >= maxit:
65              break
66
67      fx = sol
68
69      return fx, ea, it, np.array(sollist), np.array(ealist)
70
71  if __name__ == "__main__":
72      print(calc_cos(1,0,4))
```

A.4 poly_root.py

```
# -*- coding: utf-8 -*-
"""
[Poly Root]
[Marcus Deans]
[7 October 2019]

Python version of Chapra Figure 4.2 for Newton square roots
From: Applied Numerical Methods with MATLAB
      for Engineers and Scientists, 4th ed
      Steven C. Chapra
      McGraw-Hill 2018
@author: DukeEgr93
v. 1.2, 9/21/2019

I understand and have adhered to all the tenets of the Duke Community Standard
in creating this code.
Signed: [md374]
"""

def calc_root(x, es=0.0001, maxit=50):
    '''
    Newton method for finding square root
      fx, ea, it = iter_meth(x, es, maxit)
    input:
      x = value of which to find square root
      es = stopping criterion (default = 0.0001)
      maxit = maximum iterations (default = 50)
    output:
      fx = estimated value
      ea = approximate relative error (%)
      it = number of iterations
    '''

    # initialization
    it = 1
    sol = 0
    ea = 100

    sol = x
    # iterative calculation
    while True:
        solold = sol
        sol = (((2*(sol**3))-(7*(sol**2))+8)/(3*(sol**2)-(14*sol)+14))
        it += 1
        if sol != 0:
            ea = abs((sol - solold)/sol)*100

        if ea <= es or it >= maxit:
            break

    fx = sol

    return fx, ea, it
if __name__ == "__main__":
    print(calc_root(0))
```

B Figures

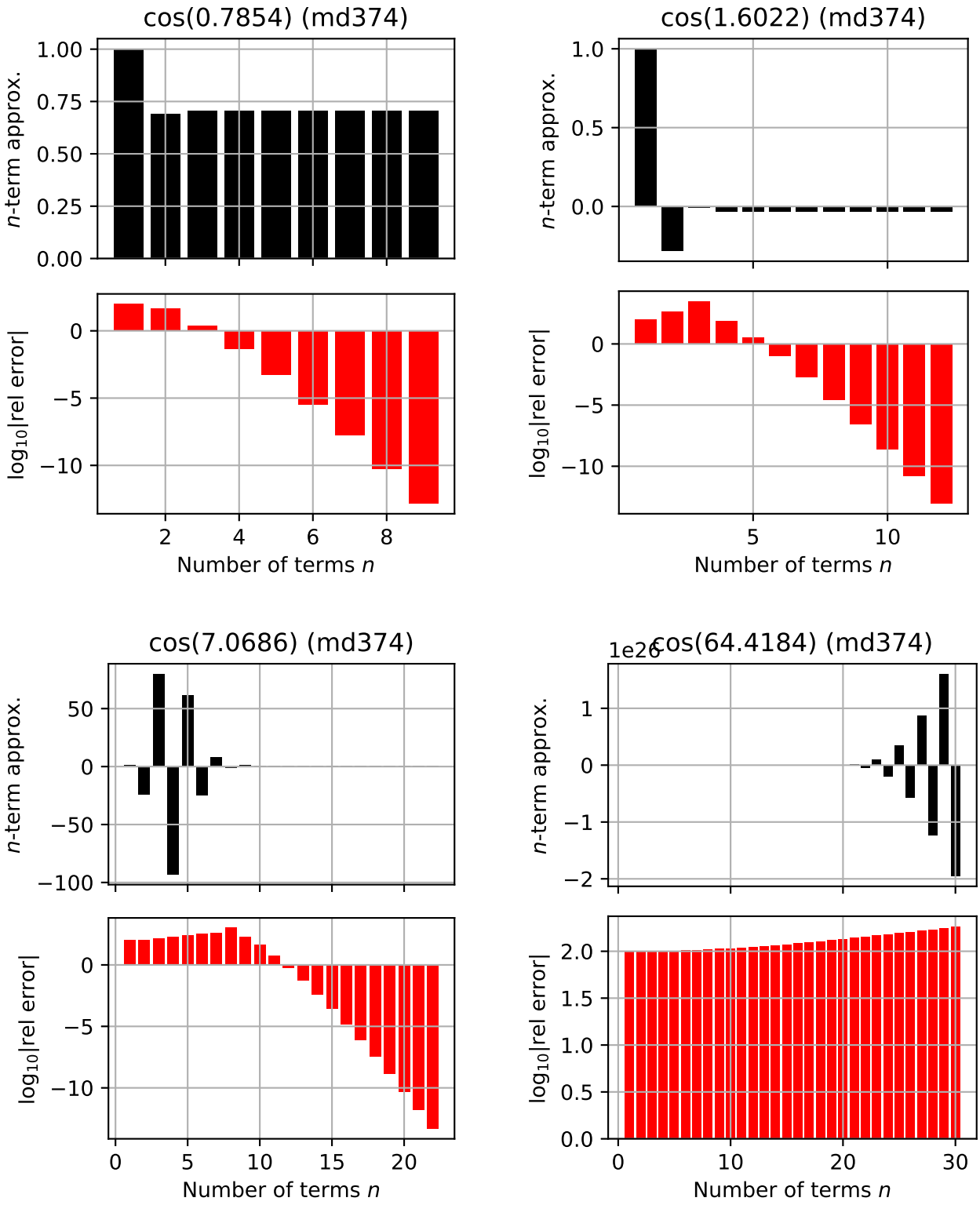

Figure 1: Figures for Cosine Approximation

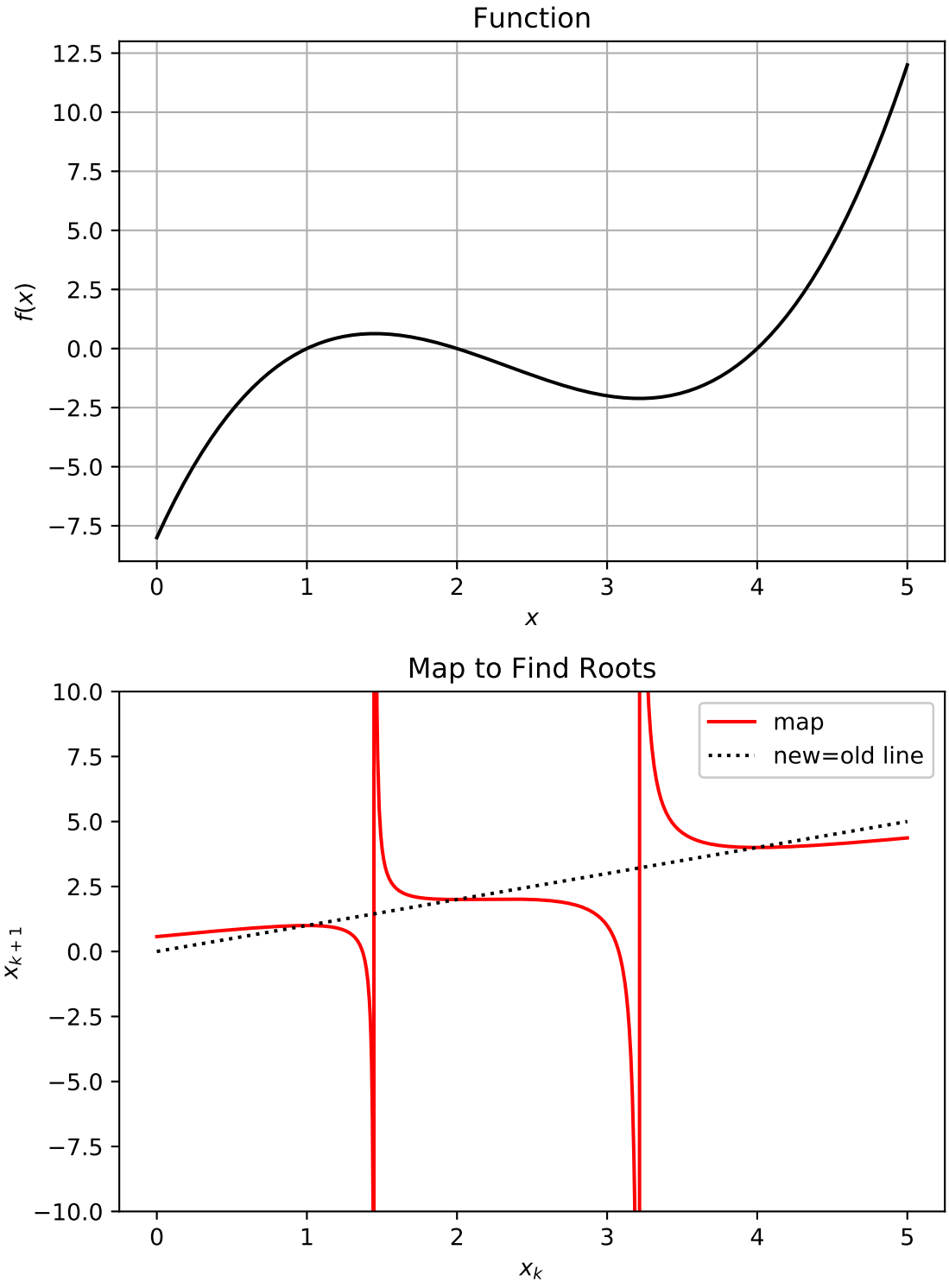

Figure 2: Function and Derivative

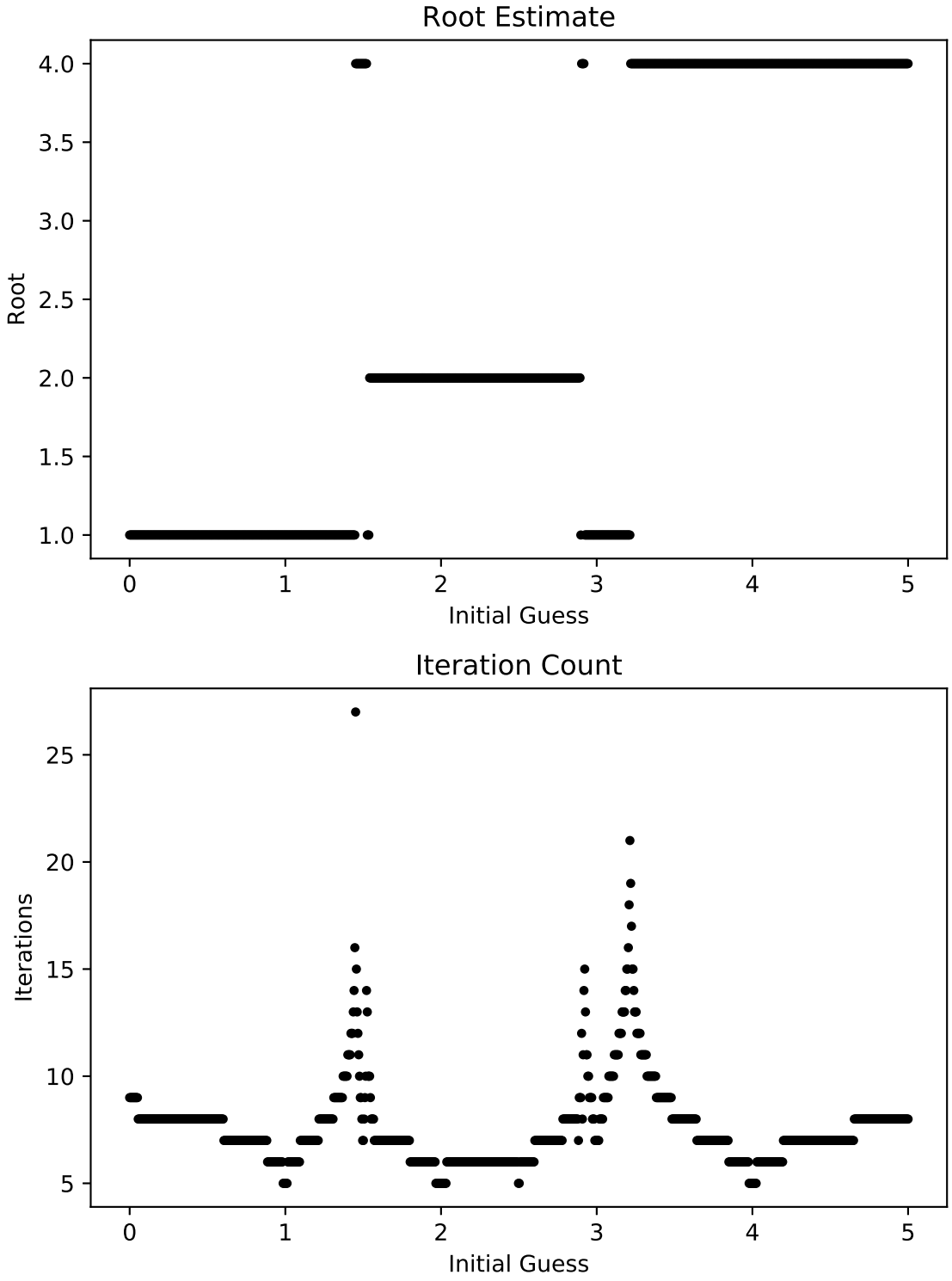

Figure 3: Root Estimates and Iteration Count

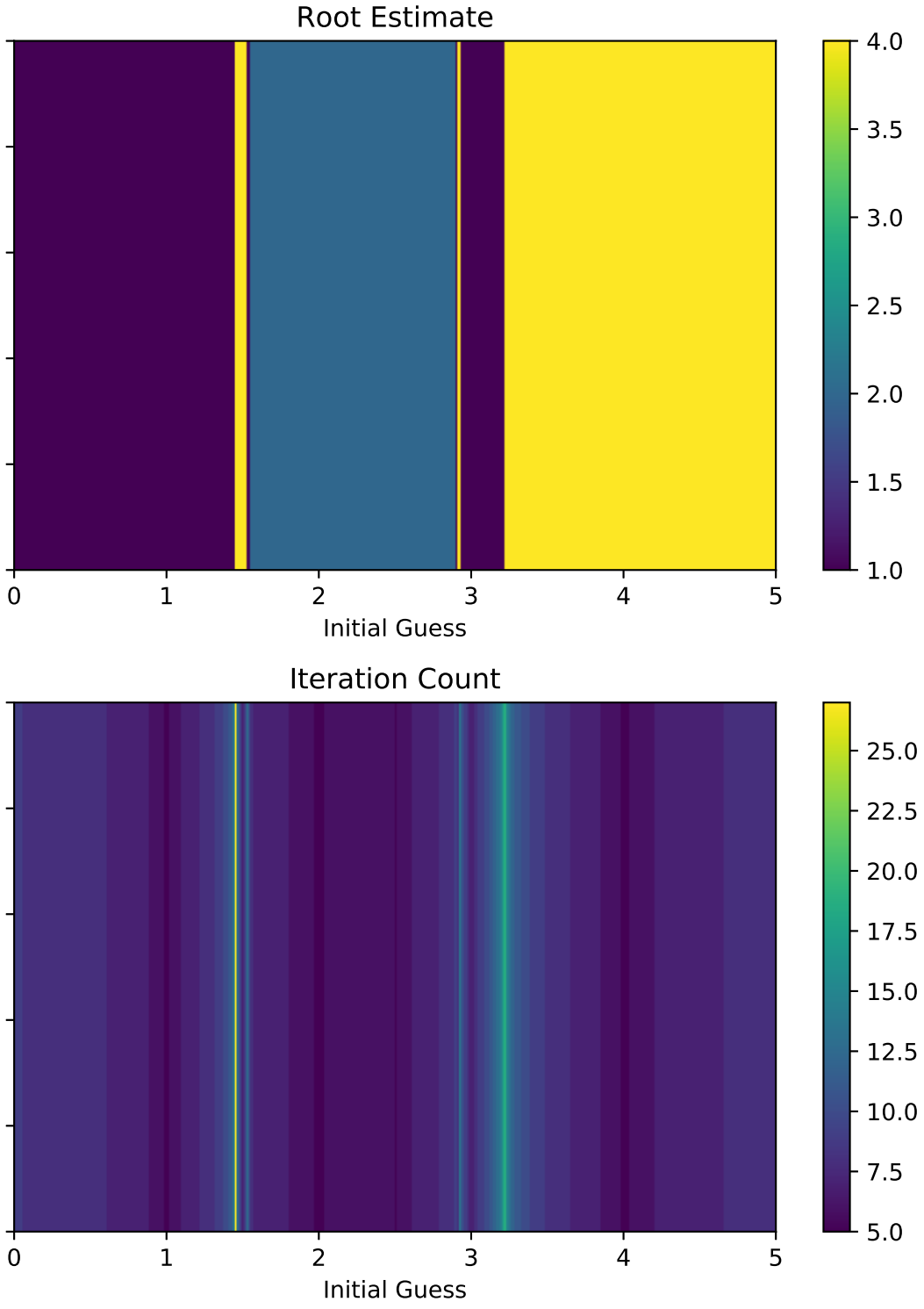

Figure 4: Root Estimates and Iteration Count - As Image