PHYS4300 Numerical Methods and Scientific Computing

Roy Forestano

HW 1
11 February 2021

Solution. *Problem 1.* These are the $c++$ files for $a = 0.8$ and $a = 1.2$ for the traditional method. You can see that one needs to start extremely close to 0 and run more steps than in the Newton-Raphson method to arrive at solutions. For $a = 1.2$, using the traditional method, you can see no matter how close to 0 you are, it runs to a stable fixed point.

```
//-----------------------------------
// This program will use the traditional method to find the stable solutions a
// of tanh(ax) = x for values of a=0.8 and a=1.2 I hope to first generate the sequence
// to find when g(x) - x = 0. Then, take this value of x and evaluate f(x) = g'(x) = 'alpha'
// This should  retun the value 'alpha' for the approximate derivative at the fixed point.
// If it is less than one, the fixed point is stable. If the value is greater than one,
// the fixed point is unstable.
//----------------------------------------------------------------
using namespace std;
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <math.h>

const double a=0.8;
const double b=1.2;

double f(double x) { return tanh(a*x); }
double g(double x) { return tanh(b*x); }

int main() {

double x;
int i;

x=0.001;

for(i=0;i<100;i++) {

cout << setw(6) << i << setw(22) << setprecision(12) << x
<< setw(22) << f(x) << setw(22)  << endl; x=f(x);

  }

cout << "There is a stable point at x = " << x
<< " for the function f(x) = tanh(0.8*x) -x.\n";

cout << "We will now show the solutions for a=1.2 in the traditional method.\n";


x=0.55;

for(i=0;i<100;i++) {

cout << setw(6) << i << setw(22) << setprecision(12) << x
<< setw(22) << g(x) << setw(22)  << endl; x=g(x);

  }

cout << "There is a stable point at x = " << x
<< " for the function f(x) = tanh(1.2*x) -x.\n";


x=-0.55;

for(i=0;i<100;i++) {

cout << setw(6) << i << setw(22) << setprecision(12) << x
<< setw(22) << g(x) << setw(22)  << endl; x=g(x);

  }

cout << "There is a stable point at x = " << x
<< " for the function f(x) = tanh(1.2*x) -x.\n";

cout << "Starting from a small positive number we find.\n";

x=0.01;

for(i=0;i<20;i++) {

cout << setw(6) << i << setw(22) << setprecision(12) << x
<< setw(22) << g(x) << setw(22)  << endl; x=g(x);

  }

cout << "Starting from a small negatuve number we find.\n";

x=-0.01;

for(i=0;i<20;i++) {

cout << setw(6) << i << setw(22) << setprecision(12) << x
<< setw(22) << g(x) << setw(22)  << endl; x=g(x);

  }

cout << "There is an unstable point at x = " << "0" << " for the function f(x) = tanh(1.2*x) -x.\n";
cout << "One can see when we start significantly close to 0, "
<< "the system trends away from x=0 to the stable fixed points found above.\n";


}
```

Here is the output.

```
[forestan@sirius hw1]$ g++ problem0.C
[forestan@sirius hw1]$ ./a.out
  0                  0.001      0.000799999829333
  1      0.000799999829333      0.000639999776085
  2      0.000639999776085      0.000511999776129
  3      0.000511999776129      0.000409599797997
  4      0.000409599797997      0.000327679826669
  5      0.000327679826669      0.000262143855331
  6      0.000262143855331       0.00020971508119
  7       0.00020971508119      0.000167772063378
  8      0.000167772063378      0.000134217649896
  9      0.000134217649896      0.000107374119505
 10      0.000107374119505      8.58992953923e-05
 11      8.58992953923e-05      6.87194362057e-05
 12      6.87194362057e-05      5.49755489092e-05
 13      5.49755489092e-05       4.3980439099e-05
 14       4.3980439099e-05      3.51843512647e-05
 15      3.51843512647e-05      2.81474810043e-05
 16      2.81474810043e-05      2.25179847996e-05
 17      2.25179847996e-05      1.80143878378e-05
 18      1.80143878378e-05      1.44115102692e-05
 19      1.44115102692e-05      1.15292082149e-05
 20      1.15292082149e-05      9.22336657162e-06
 21      9.22336657162e-06      7.37869325716e-06
 22      7.37869325716e-06      5.90295460566e-06
 23      5.90295460566e-06       4.7223636845e-06
 24       4.7223636845e-06      3.77789094758e-06
 25      3.77789094758e-06      3.02231275805e-06
 26      3.02231275805e-06      2.41785020644e-06
 27      2.41785020644e-06      1.93428016515e-06
 28      1.93428016515e-06      1.54742413212e-06
 29      1.54742413212e-06      1.23793930569e-06
 30      1.23793930569e-06      9.90351444554e-07
 31      9.90351444554e-07      7.92281155643e-07
 32      7.92281155643e-07      6.33824924514e-07
 33      6.33824924514e-07      5.07059939611e-07
 34      5.07059939611e-07      4.05647951689e-07
 35      4.05647951689e-07      3.24518361351e-07
 36      3.24518361351e-07      2.59614689081e-07
 37      2.59614689081e-07      2.07691751265e-07
 38      2.07691751265e-07      1.66153401012e-07
 39      1.66153401012e-07      1.32922720809e-07
 40      1.32922720809e-07      1.06338176648e-07
 41      1.06338176648e-07      8.50705413181e-08
 42      8.50705413181e-08      6.80564330545e-08
 43      6.80564330545e-08      5.44451464436e-08
 44      5.44451464436e-08      4.35561171549e-08
 45      4.35561171549e-08      3.48448937239e-08
 46      3.48448937239e-08      2.78759149791e-08
 47      2.78759149791e-08      2.23007319833e-08
 48      2.23007319833e-08      1.78405855866e-08
 49      1.78405855866e-08      1.42724684693e-08
 50      1.42724684693e-08      1.14179747754e-08
 51      1.14179747754e-08      9.13437982035e-09
 52      9.13437982035e-09      7.30750385628e-09
 53      7.30750385628e-09      5.84600308503e-09
 54      5.84600308503e-09      4.67680246802e-09
 55      4.67680246802e-09      3.74144197442e-09
 56      3.74144197442e-09      2.99315357953e-09
 57      2.99315357953e-09      2.39452286363e-09
 58      2.39452286363e-09       1.9156182909e-09
 59       1.9156182909e-09      1.53249463272e-09
 60      1.53249463272e-09      1.22599570618e-09
 61      1.22599570618e-09      9.80796564942e-10
 62      9.80796564942e-10      7.84637251953e-10
 63      7.84637251953e-10      6.27709801563e-10
 64      6.27709801563e-10       5.0216784125e-10
 65       5.0216784125e-10        4.01734273e-10
 66         4.01734273e-10       3.213874184e-10
 67        3.213874184e-10      2.5710993472e-10
 68       2.5710993472e-10      2.05687947776e-10
 69      2.05687947776e-10      1.64550358221e-10
 70      1.64550358221e-10      1.31640286577e-10
 71      1.31640286577e-10      1.05312229261e-10
 72      1.05312229261e-10      8.42497834091e-11
 73      8.42497834091e-11      6.73998267272e-11
 74      6.73998267272e-11      5.39198613818e-11
 75      5.39198613818e-11      4.31358891054e-11
 76      4.31358891054e-11      3.45087112844e-11
 77      3.45087112844e-11      2.76069690275e-11
 78      2.76069690275e-11       2.2085575222e-11
 79       2.2085575222e-11      1.76684601776e-11
 80      1.76684601776e-11      1.41347681421e-11
 81      1.41347681421e-11      1.13078145137e-11
 82      1.13078145137e-11      9.04625161093e-12
 83      9.04625161093e-12      7.23700128874e-12
 84      7.23700128874e-12      5.78960103099e-12
 85      5.78960103099e-12      4.63168082479e-12
 86      4.63168082479e-12      3.70534465983e-12
 87      3.70534465983e-12      2.96427572787e-12
 88      2.96427572787e-12      2.37142058229e-12
 89      2.37142058229e-12      1.89713646584e-12
 90      1.89713646584e-12      1.51770917267e-12
 91      1.51770917267e-12      1.21416733813e-12
 92      1.21416733813e-12      9.71333870508e-13
 93      9.71333870508e-13      7.77067096406e-13
 94      7.77067096406e-13      6.21653677125e-13
 95      6.21653677125e-13       4.973229417e-13
 96        4.973229417e-13      3.9785835336e-13
 97       3.9785835336e-13      3.18286682688e-13
 98      3.18286682688e-13       2.5462934615e-13
 99       2.5462934615e-13      2.0370347692e-13
There is a stable point at x = 2.0370347692e-13 for the function f(x) = tanh(0.8*x) -x.
We will now show the solutions for a=1.2 in the traditional method.
  0                   0.55      0.578363413045
  1         0.578363413045      0.600568602247
  2         0.600568602247      0.617331771293
  3         0.617331771293      0.629627127894
  4         0.629627127894      0.638449851788
  5         0.638449851788      0.644679217151
  6         0.644679217151      0.649026630346
  7         0.649026630346      0.652035768154
```

```
  6         0.644679217151      0.649026630346
  7         0.649026630346      0.652035768154
  8         0.652035768154       0.65410664444
  9          0.65410664444      0.655526142707
 10         0.655526142707      0.656496481361
 11         0.656496481361      0.657158536604
 12         0.657158536604      0.657609671167
 13         0.657609671167       0.65791681124
 14          0.65791681124      0.658125792276
 15         0.658125792276      0.658267927021
 16         0.658267927021      0.658364570639
 17         0.658364570639       0.65843027046
 18          0.65843027046      0.658474928481
 19         0.658474928481      0.658505281151
 20         0.658505281151      0.658525909695
 21         0.658525909695      0.658539928879
 22         0.658539928879      0.658549456074
 23         0.658549456074       0.65855593047
 24          0.65855593047       0.65856033022
 25          0.65856033022      0.658563320095
 26         0.658563320095      0.658565351869
 27         0.658565351869      0.658566732559
 28         0.658566732559      0.658567670803
 29         0.658567670803      0.658568308382
 30         0.658568308382      0.658568741647
 31         0.658568741647      0.658569036069
 32         0.658569036069      0.658569236142
 33         0.658569236142        0.6585693721
 34           0.6585693721      0.658569464489
 35         0.658569464489      0.658569527272
 36         0.658569527272      0.658569569936
 37         0.658569569936      0.658569598927
 38         0.658569598927      0.658569618629
 39         0.658569618629      0.658569632016
 40         0.658569632016      0.658569641114
 41         0.658569641114      0.658569647296
 42         0.658569647296      0.658569651497
 43         0.658569651497      0.658569654352
 44         0.658569654352      0.658569656292
 45         0.658569656292       0.65856965761
 46          0.65856965761      0.658569658506
 47         0.658569658506      0.658569659115
 48         0.658569659115      0.658569659529
 49         0.658569659529       0.65856965981
 50          0.65856965981      0.658569660001
 51         0.658569660001       0.65856966013
 52          0.65856966013      0.658569660219
 53         0.658569660219      0.658569660279
 54         0.658569660279      0.658569660319
 55         0.658569660319      0.658569660347
 56         0.658569660347      0.658569660366
 57         0.658569660366      0.658569660379
 58         0.658569660379      0.658569660387
 59         0.658569660387      0.658569660393
 60         0.658569660393      0.658569660397
 61         0.658569660397        0.6585696604
 62           0.6585696604      0.658569660402
 63         0.658569660402      0.658569660403
 64         0.658569660403      0.658569660404
 65         0.658569660404      0.658569660405
 66         0.658569660405      0.658569660405
 67         0.658569660405      0.658569660405
 68         0.658569660405      0.658569660405
 69         0.658569660405      0.658569660405
 70         0.658569660405      0.658569660406
 71         0.658569660406      0.658569660406
 72         0.658569660406      0.658569660406
 73         0.658569660406      0.658569660406
 74         0.658569660406      0.658569660406
 75         0.658569660406      0.658569660406
 76         0.658569660406      0.658569660406
 77         0.658569660406      0.658569660406
 78         0.658569660406      0.658569660406
 79         0.658569660406      0.658569660406
 80         0.658569660406      0.658569660406
 81         0.658569660406      0.658569660406
 82         0.658569660406      0.658569660406
 83         0.658569660406      0.658569660406
 84         0.658569660406      0.658569660406
 85         0.658569660406      0.658569660406
 86         0.658569660406      0.658569660406
 87         0.658569660406      0.658569660406
 88         0.658569660406      0.658569660406
 89         0.658569660406      0.658569660406
 90         0.658569660406      0.658569660406
 91         0.658569660406      0.658569660406
 92         0.658569660406      0.658569660406
 93         0.658569660406      0.658569660406
 94         0.658569660406      0.658569660406
 95         0.658569660406      0.658569660406
 96         0.658569660406      0.658569660406
 97         0.658569660406      0.658569660406
 98         0.658569660406      0.658569660406
 99         0.658569660406      0.658569660406
There is a stable point at x = 0.658569660406 for the function f(x) = tanh(1.2*x) -x.
  0                  -0.55     -0.578363413045
  1        -0.578363413045     -0.600568602247
  2        -0.600568602247     -0.617331771293
  3        -0.617331771293     -0.629627127894
  4        -0.629627127894     -0.638449851788
  5        -0.638449851788     -0.644679217151
  6        -0.644679217151     -0.649026630346
  7        -0.649026630346     -0.652035768154
  8        -0.652035768154      -0.65410664444
  9         -0.65410664444     -0.655526142707
 10        -0.655526142707     -0.656496481361
 11        -0.656496481361     -0.657158536604
 12        -0.657158536604     -0.657609671167
 13        -0.657609671167      -0.65791681124
 14         -0.65791681124     -0.658125792276
 15        -0.658125792276     -0.658267927021
 16        -0.658267927021     -0.658364570639
```

Here is a continuation of the output.

```
15      -0.658125792276      -0.658267927021
16      -0.658267927021      -0.658364570639
17      -0.658364570639       -0.65843027046
18       -0.65843027046      -0.658474928481
19      -0.658474928481      -0.658505281151
20      -0.658505281151      -0.658525909695
21      -0.658525909695      -0.658539928879
22      -0.658539928879      -0.658549456074
23      -0.658549456074       -0.65855593047
24       -0.65855593047       -0.65856033022
25       -0.65856033022      -0.658563320095
26      -0.658563320095      -0.658565351869
27      -0.658565351869      -0.658566732559
28      -0.658566732559      -0.658567670803
29      -0.658567670803      -0.658568308382
30      -0.658568308382      -0.658568741647
31      -0.658568741647      -0.658569036069
32      -0.658569036069      -0.658569236142
33      -0.658569236142        -0.6585693721
34        -0.6585693721      -0.658569464489
35      -0.658569464489      -0.658569527272
36      -0.658569527272      -0.658569569936
37      -0.658569569936      -0.658569598927
38      -0.658569598927      -0.658569618629
39      -0.658569618629      -0.658569632016
40      -0.658569632016      -0.658569641114
41      -0.658569641114      -0.658569647296
42      -0.658569647296      -0.658569651497
43      -0.658569651497      -0.658569654352
44      -0.658569654352      -0.658569656292
45      -0.658569656292       -0.65856965761
46       -0.65856965761      -0.658569658506
47      -0.658569658506      -0.658569659115
48      -0.658569659115      -0.658569659529
49      -0.658569659529       -0.65856965981
50       -0.65856965981      -0.658569660001
51      -0.658569660001       -0.65856966013
52       -0.65856966013      -0.658569660219
53      -0.658569660219      -0.658569660279
54      -0.658569660279      -0.658569660319
55      -0.658569660319      -0.658569660347
56      -0.658569660347      -0.658569660366
57      -0.658569660366      -0.658569660379
58      -0.658569660379      -0.658569660387
59      -0.658569660387      -0.658569660393
60      -0.658569660393      -0.658569660397
61      -0.658569660397        -0.6585696604
62        -0.6585696604      -0.658569660402
63      -0.658569660402      -0.658569660403
64      -0.658569660403      -0.658569660404
65      -0.658569660404      -0.658569660405
66      -0.658569660405      -0.658569660405
67      -0.658569660405      -0.658569660405
68      -0.658569660405      -0.658569660405
69      -0.658569660405      -0.658569660405
70      -0.658569660405      -0.658569660406
71      -0.658569660406      -0.658569660406
72      -0.658569660406      -0.658569660406
73      -0.658569660406      -0.658569660406
74      -0.658569660406      -0.658569660406
75      -0.658569660406      -0.658569660406
76      -0.658569660406      -0.658569660406
77      -0.658569660406      -0.658569660406
78      -0.658569660406      -0.658569660406
79      -0.658569660406      -0.658569660406
80      -0.658569660406      -0.658569660406
81      -0.658569660406      -0.658569660406
82      -0.658569660406      -0.658569660406
83      -0.658569660406      -0.658569660406
84      -0.658569660406      -0.658569660406
85      -0.658569660406      -0.658569660406
86      -0.658569660406      -0.658569660406
87      -0.658569660406      -0.658569660406
88      -0.658569660406      -0.658569660406
89      -0.658569660406      -0.658569660406
90      -0.658569660406      -0.658569660406
91      -0.658569660406      -0.658569660406
92      -0.658569660406      -0.658569660406
93      -0.658569660406      -0.658569660406
94      -0.658569660406      -0.658569660406
95      -0.658569660406      -0.658569660406
96      -0.658569660406      -0.658569660406
97      -0.658569660406      -0.658569660406
98      -0.658569660406      -0.658569660406
99      -0.658569660406      -0.658569660406
There is a stable point at x = -0.658569660406 for the function f(x) = tanh(1.2*x) -x.
Starting from a small positive number we find.
 0                 0.01      0.0119994240332
 1      0.0119994240332      0.0143983137377
 2      0.0143983137377      0.0172762573678
 3      0.0172762573678      0.0207285392491
 4      0.0207285392491      0.0248691182359
 5      0.0248691182359      0.0298340856515
 6      0.0298340856515       0.035785615225
 7       0.035785615225      0.0429163611292
 8      0.0429163611292      0.0514541522853
 9      0.0514541522853      0.0616666358106
10      0.0616666358106      0.0738651837213
11      0.0738651837213      0.0884068120403
12      0.0884068120403       0.105691961172
13       0.105691961172       0.126154639116
14       0.126154639116        0.15023960674
15        0.15023960674       0.178359263623
16       0.178359263623       0.210821705373
17       0.210821705373       0.247723530759
18       0.247723530759       0.288810690279
19       0.288810690279       0.333332656049
Starting from a small negatuve number we find.
 0                -0.01     -0.0119994240332
 1     -0.0119994240332     -0.0143983137377
 2     -0.0143983137377     -0.0172762573678
 3     -0.0172762573678     -0.0207285392491
```

```
Starting from a small negatuve number we find.
 0                -0.01     -0.0119994240332
 1     -0.0119994240332     -0.0143983137377
 2     -0.0143983137377     -0.0172762573678
 3     -0.0172762573678     -0.0207285392491
 4     -0.0207285392491     -0.0248691182359
 5     -0.0248691182359     -0.0298340856515
 6     -0.0298340856515      -0.035785615225
 7      -0.035785615225     -0.0429163611292
 8     -0.0429163611292     -0.0514541522853
 9     -0.0514541522853     -0.0616666358106
10     -0.0616666358106     -0.0738651837213
11     -0.0738651837213     -0.0884068120403
12     -0.0884068120403      -0.105691961172
13      -0.105691961172      -0.126154639116
14      -0.126154639116       -0.15023960674
15       -0.15023960674      -0.178359263623
16      -0.178359263623      -0.210821705373
17      -0.210821705373      -0.247723530759
18      -0.247723530759      -0.288810690279
19      -0.288810690279      -0.333332656049
There is an unstable point at x = 0 for the function f(x) = tanh(1.2*x) -x.
One can see when we start significantly close to 0, the system trends away from x=0 to t
he stable fixed points found above.
```

We will now show results for the Newton-Raphson method. These are the $c++$ files for $a = 0.8$, where a stable fixed point is found at $x = 0$.

```
  GNU nano 2.3.1                      File: problem1.C

//------------------------------------
// This program will use the Newton-Raphson method to find the stable solutionsa
// of tanh(ax) - x = 0 for values of a=0.8. I hope to first generate the sequence
// to find when g(x) - x = 0. Then, take this value of x and evaluate f(x) = g'(x) = 'alpha'
// This should  retun the value 'alpha' for the approximate derivative at the fixed point.
// If it is less than one, the fixed point is stable. If the value is greater than one,
// the fixed point is unstable.
//-------------------------------------------------------------------
using namespace std;
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <math.h>

const double a=0.8;

int N=10;

double f(double x) { return tanh(a*x) - x; }

double g(double x) { return (a*x*(1/cosh(a*x)/cosh(a*x)) - tanh(a*x))/(a*(1/cosh(a*x)/cosh(a*x))-1); }

int main() {

double x;
int i;

cout << "Enter an initial x value: \n";
cin >> x;

for(i=0;i<N;i++) {

cout << setw(6) << i << setw(22) << setprecision(12) << x
<< setw(22) << f(x) << setw(22) <<  g(x)-x << endl; x=g(x);

  }

cout << "There is a stable point at x = " << x
<< " for the function f(x) = tanh(0.8*x) -x.\n";


}
```

These are the $c++$ files for $a = 1.2$. The first program finds a stable point at $x_1 = -0.65856966$.

```
  GNU nano 2.3.1                    File: problem1alt.C

//------------------------------------
// This program will use the Newton-Raphson method to find the stable solutionsa
// of tanh(ax) - x = 0 for values of a=1.2. I hope to first generate the sequence
// to find when g(x) - x = 0. Then, take this value of x and evaluate f(x) = g'(x) = 'alpha'
// This should  retun the value 'alpha' for the approximate derivative at the fixed point.
// If it is less than one, the fixed point is stable. If the value is greater than one,
// the fixed point is unstable.
//-------------------------------------------------------------------
using namespace std;
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <math.h>

const double a=1.2;

int N=10;

double f(double x) { return tanh(a*x) - x; }

double g(double x) { return (a*x*(1/cosh(a*x)/cosh(a*x)) - tanh(a*x))/(a*(1/cosh(a*x)/cosh(a*x))-1); }

int main() {

double x;
int i;

x=-2;

for(i=0;i<N;i++) {

cout << setw(6) << i << setw(22) << setprecision(12) << x
<< setw(22) << f(x) << setw(22) <<  g(x)-x << endl; x=g(x);

  }

cout << "There is a stable fixed point at x = " << x
<< " for the function f(x) = tanh(1.2*x) -x.\n";
cout << "Run the next program problem1alt2, to find the next stable fixed point.\n";


}
```

The second program finds a stable point at $x_2 = -x_1$.

```
 GNU nano 2.3.1                     File: problem1alt2.C

//-----------------------------------
// This program will use the Newton-Raphson method to find the stable solutionsa
// of tanh(ax) - x = 0 for values of a=1.2. I hope to first generate the sequence
// to find when g(x) - x = 0. Then, take this value of x and evaluate f(x) = g'(x) = 'alpha'
// This should  retun the value 'alpha' for the approximate derivative at the fixed point.
// If it is less than one, the fixed point is stable. If the value is greater than one,
// the fixed point is unstable.
//-----------------------------------------------------------------
using namespace std;
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <math.h>

const double a=1.2;

int N=10;

double f(double x) { return tanh(a*x) - x; }

double g(double x) { return (a*x*(1/cosh(a*x)/cosh(a*x)) - tanh(a*x))/(a*(1/cosh(a*x)/cosh(a*x))-1); }

int main() {

double x;
int i;

x=2;

for(i=0;i<N;i++) {

cout << setw(6) << i << setw(22) << setprecision(12) << x
<< setw(22) << f(x) << setw(22) <<  g(x)-x << endl; x=g(x);

  }

cout << "There is a stable fixed point at x = " << x
<< " for the function f(x) = tanh(1.2*x) -x.\n";
cout << "Run the next program problem1alt3, to find the next unstable fixed point.\n";


}
```

The third program finds an unstable fixed point at $x_3 = 0$.

```
  GNU nano 2.3.1                    File: problem1alt3.C

// This program will use the Newton-Raphson method to find the stable solutionsa
// of tanh(ax) - x = 0 for values of a=1.2. I hope to first generate the sequence
// to find when g(x) - x = 0. Then, take this value of x and evaluate f(x) = g'(x) = 'alpha'
// This should  retun the value 'alpha' for the approximate derivative at the fixed point.
// If it is less than one, the fixed point is stable. If the value is greater than one,
// the fixed point is unstable.
//-------------------------------------------------------------------
using namespace std;
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <math.h>

const double a=1.2;

int N=10;

double f(double x) { return tanh(a*x) - x; }

double g(double x) { return (a*x*(1/cosh(a*x)/cosh(a*x)) - tanh(a*x))/(a*(1/cosh(a*x)/cosh(a*x))-1); }

int main() {

double x;
int i;

x=0.2;

for(i=0;i<N;i++) {

cout << setw(6) << i << setw(22) << setprecision(12) << x
<< setw(22) << f(x) << setw(22) <<  g(x)-x << endl; x=g(x);

  }

cout << "There is an unstable fixed point at x = " << x
<< " for the function f(x) = tanh(1.2*x) -x.\n";
cout << "This is the final fixed point for this function.\n";
cout << "We will now use gnuplot to graphically show these results for the unstable point in particular.\n";

}
```

After this, the $logip1.C$ file is a $c{+}{+}$ file that generates the data output for the Newton-Raphson method Logistic map for the unstable point at x_3, $a = 1.2$.

```
[forestan@sirius hw1]$ cat logip1.C
//---------------------
using namespace std;
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <math.h>

const double a=1.2;
const int N=10;

double g(double x) { return (a*x*(1/cosh(a*x)/cosh(a*x))-tanh(a*x))/(a*(1/cosh(a*x)/cosh(a*x))-1.0); }

int main(){

double xn;
int i;

xn=0.3;
cout << xn << " " << 0.0 << endl;
for(i=0;i<N;i++){
cout << xn << " " << g(xn)
<< endl;
xn=g(xn);
cout << xn << " " << xn << endl;
}
}
[forestan@sirius hw1]$ g++ logip1.C
[forestan@sirius hw1]$ ./a.out
0.3 0
0.3 -0.49333
-0.49333 -0.49333
-0.49333 -0.767155
-0.767155 -0.767155
-0.767155 -0.672467
-0.672467 -0.672467
-0.672467 -0.658877
-0.658877 -0.658877
-0.658877 -0.65857
-0.65857 -0.65857
-0.65857 -0.65857
-0.65857 -0.65857
-0.65857 -0.65857
-0.65857 -0.65857
-0.65857 -0.65857
-0.65857 -0.65857
-0.65857 -0.65857
-0.65857 -0.65857
-0.65857 -0.65857
-0.65857 -0.65857
```

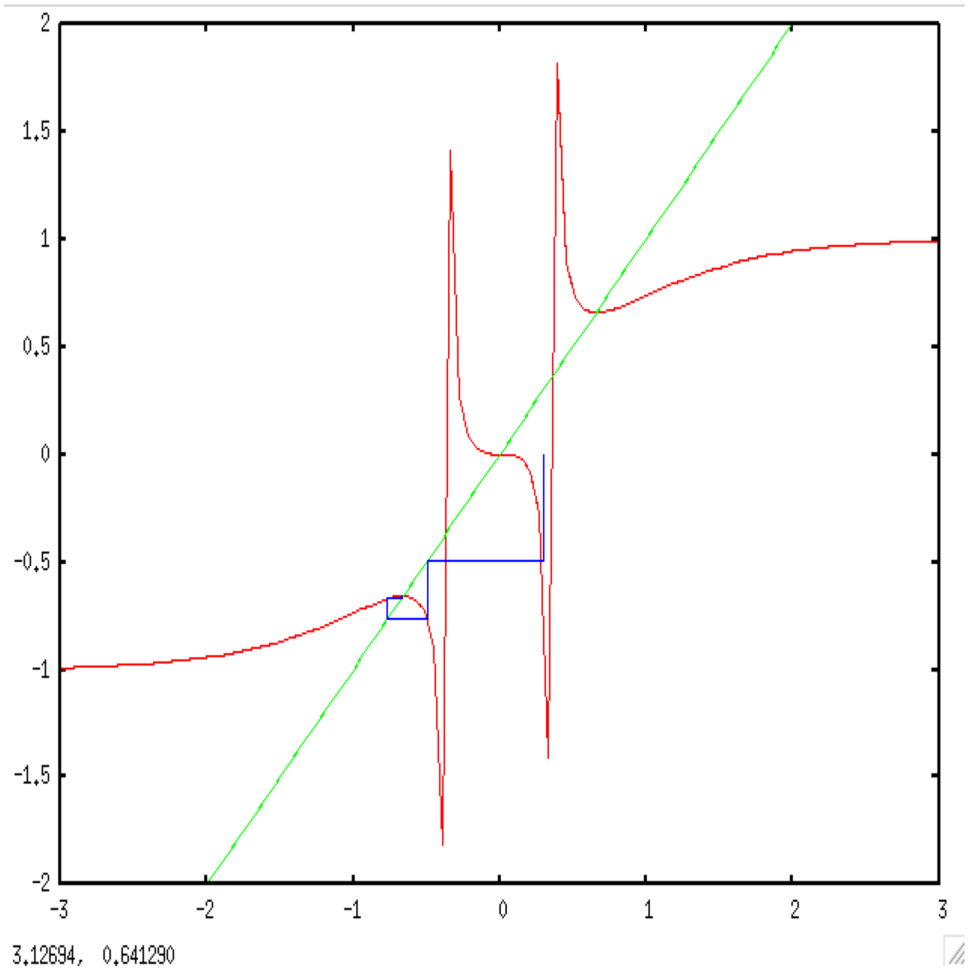
This is followed by the corresponding *gnuplot* file.

```
  GNU nano 2.3.1                 File: logip1buffer.gnu


a=1.2
g(x) = (a*x*(1/cosh(a*x)/cosh(a*x))-tanh(a*x))/(a*(1/cosh(a*x)/cosh(a*x))-1.0)
set xr[-3:3]
set yr[-2:2]
plot g(x), x, "logip1.dat" u 1:2 w l
```

Following this, you can see the results as I run the code.

```
[forestan@sirius hw1]$ g++ problem1.C
[forestan@sirius hw1]$ ./a.out
Enter an initial x value:
90
     0                  90                -89                -89
     1                   1    -0.335963229732    -0.607796769804
     2      0.392203230196   -0.0883470500873     -0.32259594917
     3     0.0696072810262   -0.0139789438312   -0.0690401370534
     4    0.00056714397274 -0.000113428825682 -0.000567143661405
     5   3.11335211135e-10   -6.2267042227e-11  -3.11335211135e-10
     6   2.58493941423e-25  -5.16987882846e-26  -2.58493941423e-25
     7                  -0                  0                  0
     8                  -0                  0                  0
     9                  -0                  0                  0
There is a stable point at x = -0 for the function f(x) = tanh(0.8*x) -x.
[forestan@sirius hw1]$ g++ problem1alt.C
[forestan@sirius hw1]$ ./a.out
     0                  -2      1.01632514231      1.05741692355
     1     -0.942583076449     0.131187782842     0.222339064977
     2     -0.720244011472    0.0217811772884    0.0565127779032
     3     -0.663731233569    0.0016683550843    0.0051177681377
     4     -0.658613465431  1.40386486378e-05  4.38018102284e-05
     5     -0.658569663621  1.03033204191e-09  3.21519788571e-09
     6     -0.658569660406                  0                  0
     7     -0.658569660406                  0                  0
     8     -0.658569660406                  0                  0
     9     -0.658569660406                  0                  0
There is a stable fixed point at x = -0.658569660406 for the function f(x) = tanh(1.2*x) -x.
Run the next program problem1alt2, to find the next stable fixed point.
[forestan@sirius hw1]$ g++ problem1alt2.C
[forestan@sirius hw1]$ ./a.out
     0                   2     -1.01632514231     -1.05741692355
     1      0.942583076449    -0.131187782842    -0.222339064977
     2      0.720244011472   -0.0217811772884   -0.0565127779032
     3      0.663731233569   -0.0016683550843   -0.0051177681377
     4      0.658613465431 -1.40386486378e-05 -4.38018102284e-05
     5      0.658569663621 -1.03033204191e-09 -3.21519788571e-09
     6      0.658569660406                  0                  0
     7      0.658569660406                  0                  0
     8      0.658569660406                  0                  0
     9      0.658569660406                  0                  0
There is a stable fixed point at x = 0.658569660406 for the function f(x) = tanh(1.2*x) -x.
Run the next program problem1alt3, to find the next unstable fixed point.
[forestan@sirius hw1]$ g++ problem1alt3.C
[forestan@sirius hw1]$ ./a.out
     0                 0.2    0.0354957495385    -0.265985180351
     1    -0.0659851803507   -0.0130319638761    0.0676958588737
     2    0.00171067852303  0.00034213282106  -0.00171070735918
     3   -2.8836145569e-08  -5.7672291138e-09    2.8836145569e-08
     4   1.65436122511e-22  3.30872245021e-23  -1.65436122511e-22
     5                   0                  0                  0
     6                   0                  0                  0
     7                   0                  0                  0
     8                   0                  0                  0
     9                   0                  0                  0
There is an unstable fixed point at x = 0 for the function f(x) = tanh(1.2*x) -x.
This is the final fixed point for this function.
We will now use gnuplot to graphically show these results for the unstable point in particular.
```

One can see that in the final graph, the data starts at $x = 0.3$ but runs away from the unstable fixed point, x_3, to the stable fixed point, x_1, for $a = 1.2$.

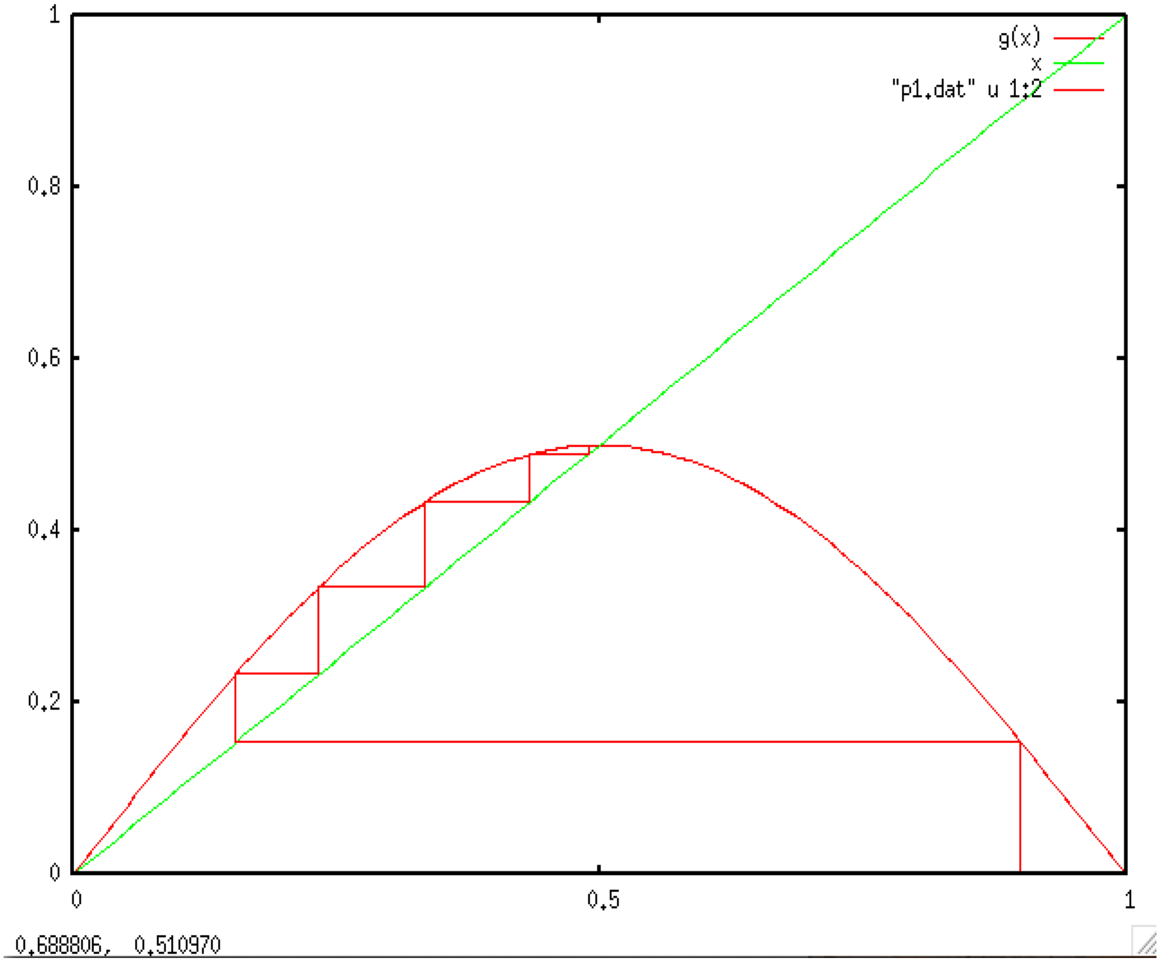


□

Solution. *Problem 2.* One example of a period-1 cycle would be where $r = 0.500$.

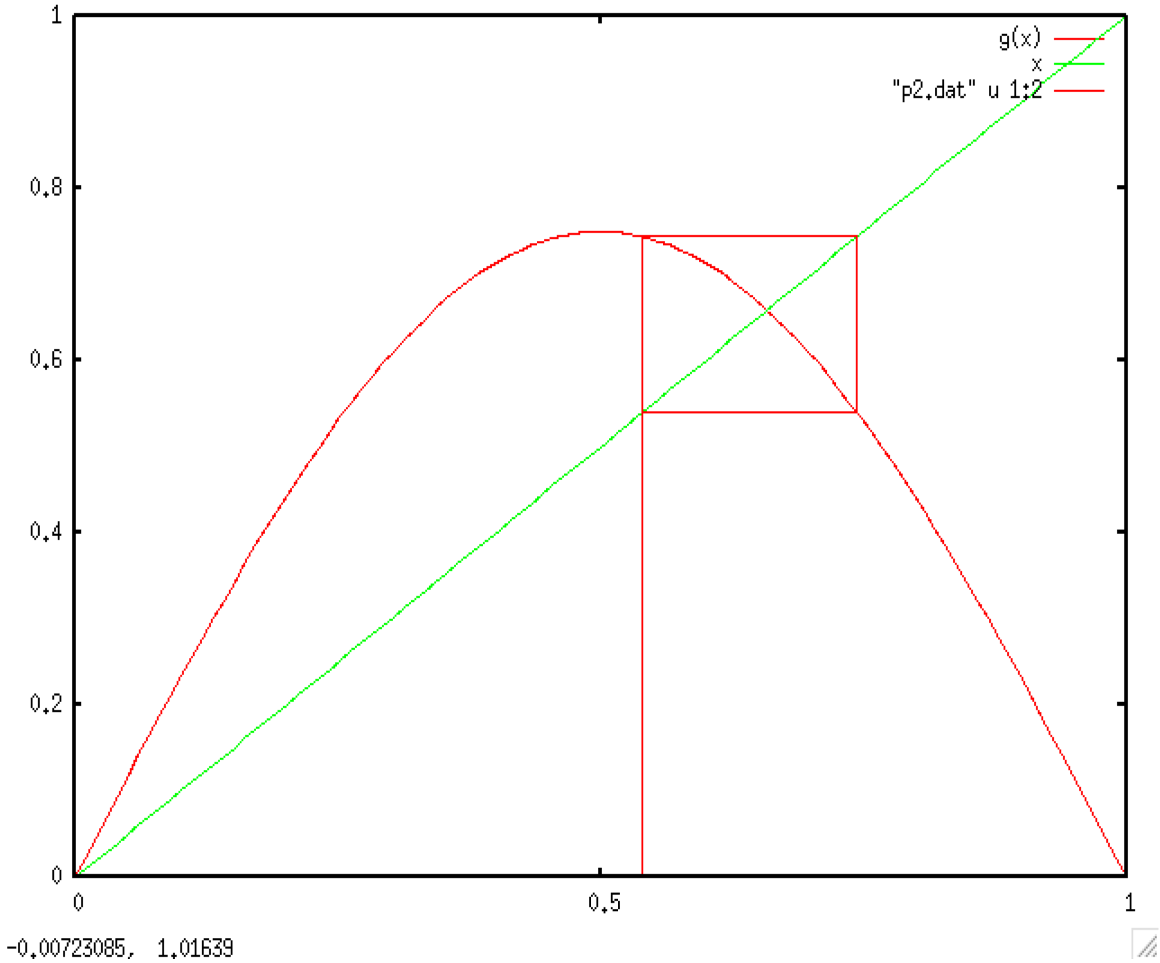
```
[forestan@sirius hw1]$ cat p1.C
//---------------------
using namespace std;
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <math.h>


const double r=0.5; //upper limit is r<0.675
const int N=1000;
const int L=40;

double g(double x) { return r*sin(3.14159*x);}

int main(){

double xn;
int i;
xn=0.9;

cout << xn << " " << 0.0 << endl;
for(i=0;i<L;i++){
cout << xn << " " << g(xn)
<< endl;
xn=g(xn);
cout << xn << " " << xn << endl;
}
}


[forestan@sirius hw1]$ g++ p1.C
[forestan@sirius hw1]$ ./a.out
0.9 0
0.9 0.15451
0.15451 0.15451
0.15451 0.233284
0.233284 0.233284
0.233284 0.334507
0.334507 0.334507
0.334507 0.433931
0.433931 0.433931
0.433931 0.489268
0.489268 0.489268
0.489268 0.499716
0.499716 0.499716
0.499716 0.5
0.5 0.5
0.5 0.5
0.5 0.5
0.5 0.5
0.5 0.5
0.5 0.5
```

where you can see the plot for this as



One example of a period-2 cycle would be where $r = 0.750$.

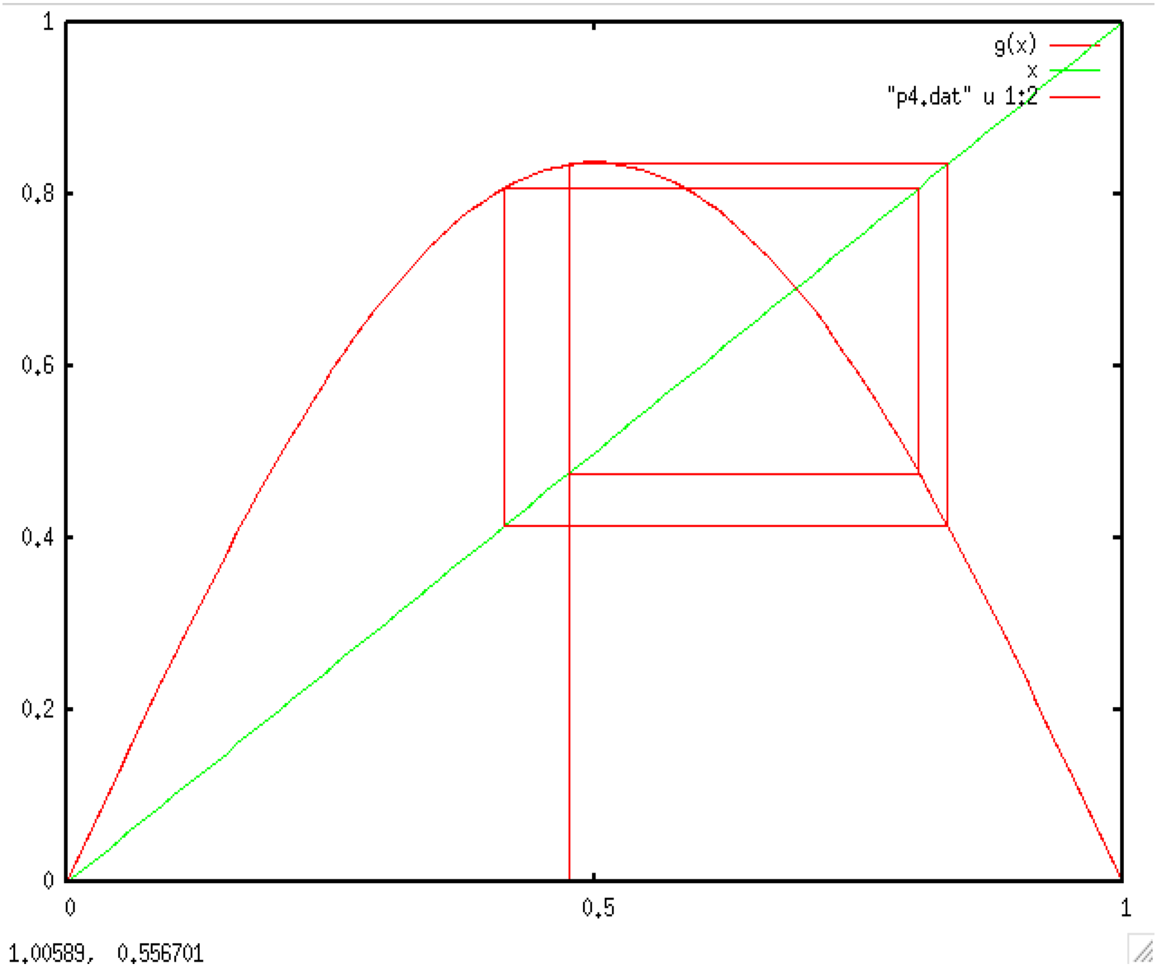
```
[forestan@sirius hw1]$ cat p2.C

//---------------------
using namespace std;
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <math.h>


const double r=0.750;
const int N=1000;
const int L=40;

double g(double x) { return r*sin(3.14159*x);}

int main(){

double xn;
int i;
xn=0.9;

for(i=0;i<N;i++){xn=g(xn);}


cout << xn << " " << 0.0 << endl;
for(i=0;i<L;i++){
cout << xn << " " << g(xn)
<< endl;
xn=g(xn);
cout << xn << " " << xn << endl;
}
}


[forestan@sirius hw1]$ g++ p2.C
[forestan@sirius hw1]$ ./a.out
0.540178 0
0.540178 0.744033
0.744033 0.744033
0.744033 0.540178
0.540178 0.540178
0.540178 0.744033
0.744033 0.744033
0.744033 0.540178
0.540178 0.540178
0.540178 0.744033
0.744033 0.744033
0.744033 0.540178
0.540178 0.540178
0.540178 0.744033
0.744033 0.744033
0.744033 0.540178
0.540178 0.540178
0.540178 0.744033
0.744033 0.744033
0.744033 0.540178
0.540178 0.540178
0.540178 0.744033
```



g(x)
x
"p2.dat" u 1:2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
-0.00723085, 1.01639

A period-4 cycle can be found using $r = 0.8375$.

```
[forestan@sirius hw1]$ cat p4.C
//-----------------------------------
//
//
//
//-----------------------------------
using namespace std;
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <math.h>


const double r=0.8375;
const int N=1000;
const int L=20;

double g(double x) { return r*sin(3.14159*x);}

int main(){

double xn;
int i;
xn=0.9;

for(i=0;i<N;i++){xn=g(xn);}


cout << xn << " " << 0.0 << endl;
for(i=0;i<L;i++){
cout << xn << " " << g(xn)
<< endl;
xn=g(xn);
cout << xn << " " << xn << endl;
}
}



[forestan@sirius hw1]$ g++ p4.C
[forestan@sirius hw1]$ ./a.out
0.475935 0
0.475935 0.835108
0.835108 0.835108
0.835108 0.414703
0.414703 0.414703
0.414703 0.80761
0.80761 0.80761
0.80761 0.475935
0.475935 0.475935
0.475935 0.835108
0.835108 0.835108
0.835108 0.414703
0.414703 0.414703
0.414703 0.80761
0.80761 0.80761
0.80761 0.475935
0.475935 0.475935
0.475935 0.835108
0.835108 0.835108
0.835108 0.414703
```


1
0.8
0.6
0.4
0.2
0
0
0.5
1
g(x)
x
"p4.dat" u 1:2
1.00589, 0.556701

The gnuplot files for the above solutions are similar.

```
gnuplot> !cat p1.gnu
set xtics 0.5
r=0.5
g(x)=r*sin(pi*x)
set xr[0:1]
set yr[0:1]
plot g(x), x, "p1.dat" u 1:2 w l ls 1
```

```
gnuplot> loadgnuplot>
gnuplot> !cat p2.gnu
set xtics 0.5
r=0.750
g(x)=r*sin(pi*x)
set xr[0:1]
set yr[0:1]
plot g(x), x, "p2.dat" u 1:2 w l ls 1
```

```
gnuplot> !cat p4.gnu
set xtics 0.5
r=0.8375
g(x)=r*sin(pi*x)
set xr[0:1]
set yr[0:1]
plot g(x), x, "p4.dat" u 1:2 w l ls 1
```

A final note on the range of r's: for a period 1 cycle the range is about $0.400 \leq r \leq 0.675$, for a period 2 cycle about $0.735 \leq r \leq 0.815$, for a period 4 cycle about $0.8375 \leq r < 0.8425$. $\square$