# ECON90055 Computational Economics: Assignment 3

Zelin Chen - 797036

April 26, 2017

## 1 Problem 1

### 1.1 Solutions to part(a) and part(b)

Since when we solve for results of the matrix, all three methods provided consistent convergence to $0_n$. This part will focus on the display of code that solve the problem, and the efficiency of each method, i.e. how many iterations does each method take for the convergence. Each method has similar set up, the difference is at where it iterates to get new updates of x's. For example, Gauss-Seidel use updated value of $x_1$ into calculation of next $x_2$. While, Jacobi's Method does not use updated value into calculation straight away until the current iteration ends. In each iteration, it uses input value of x's to generate a complete vector of updated x's. Successive Over-Relaxation(SOR) Method is similar to Gauss-Seidel as it takes the updated value of each x (e.g.$x_1$) back into the calculation for the next x (e.g.$x_2$). The difference is that when iterate value for a certain x, SOR Method includes a ratio to take into account the effect of old value of $x$ on the new value of to be updated($x'$). The ratio depends on the value of $w$.

The following three figures show my code on each of methods:

Listing 1: Code for Gauss-Seidel Method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Assignment 3 - "Gaussian-Seidel"
%---------------------------
% Coder: Zelin Chen (797036)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Cx = b, C = constant, b = results, err = stopping
    rule
function term = gaussei03(C,b,err)
% create the argument A
A = [C b];
% row of A: column is assumed to be (n+1)
n = size(A,1);
```

```

% define a starting value of x
x_val = zeros(n,1);
for m=1:n
    x_val(m) = 1;
end

% define errors
x_err = x_val;
% define a temporary storage for x's during iteration
x_tem = zeros(n,1);
% iteration count
term = 0;

% iteration continue until hit the stopping rule
while max(abs(x_err))>err
    term = 1 + term;

    % iterate value for each x, given the value of all
        other x.
    for j =1:n  %row
        x_val(j) = A(j,n+1);
        for k =1:n %col
            if k~=j
            x_val(j) = x_val(j) - A(j,k)*x_val(k);
            end
        end
        % solve for updated x'
        x_val(j) = x_val(j)/A(j,j);
        % solve for error btw x' and x
        x_err(j) = x_val(j)-x_tem(j);
    end
    % save temporary results for later error
       calculation
    x_tem = x_val;
end
end
```

Listing 2: Code for Jacobi's Method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Assignment 3 - "Jacobi"
%---------------------------
% Coder: Zelin Chen (797036)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
% Cx = b, C = constant, b = results, err = stopping rule
function term = jacobi(C,b,err)
% create the argument A
A = [C b];
% row of A: column is assumed to be (n+1)
n = size(A,1);

% define x starting values = 1
x_old = zeros(n,1);
for m=1:n
    x_old(m,1) = 1;
end

% space for storing new value of x in iteration
x_new = zeros(n,1);
% define space for storing errors
x_err = x_new - x_old;

% term count
term = 0;

% iteration continue until it hit stopping rule
while max(abs(x_err))>err
    term = 1 + term;

    % iterate value for each x, given the value of all other x.
    for j =1:n  %row
        x_new(j) = A(j,n+1);
        for k =1:n %col
            if k~=j
            x_new(j) = x_new(j) - A(j,k)*x_old(k);
            end
        end
        % solve for updated x'
        x_new(j) = x_new(j)/A(j,j);
        % solve for error btw x' and x
        x_err(j) = x_new(j)-x_old(j);
    end
    % update
    x_old = x_new;
end
end
```

Listing 3: Code for SOR

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Assignment 3 - "SOR Metohod"
%---------------------------
% Coder: Zelin Chen (797036)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Cx = b, C = constant, b = results, err = stopping rule
function term = SOR(C,b,k,err)
% determine value of w.
w = 0.1*k;

% create the argument A
A = [C b];
% row of A: column is assumed to be (n+1)
n = size(A,1);

% define a starting value of x
x_val = zeros(n,1);
for m=1:n
    x_val(m) = 1;
end

% define errors
x_err = x_val;
% define a temporary storage for x's during iteration
x_tem = x_val;
% iteration count
term = 0;

% iteration continue until the maxmium element in errors is lower than 0.0001
while max(abs(x_err))>err
    term = 1 + term;

    % iterate value for each x, given the value of all other x.
    for j =1:n  %row
        x_val(j) = A(j,n+1)+ x_tem(j)*((1-w)/w)*A(j,j);

        for k =1:n
            if k~=j
            x_val(j) = x_val(j) - A(j,k)*x_val(k);
```

```
            end
        end
        % solve for updated x'
        x_val(j) = (x_val(j)/A(j,j)) * w;
        % solve for error btw x' and x
        x_err(j) = x_val(j)-x_tem(j);
    end
    % update temporary x
    x_tem = x_val;
end
end
```

I wrote a main script to call the function automatically under sample size n = 5,10,20,50 and stopping error = 0.01, 0.001 and 0.0001. The result is displayed in table 1 in appendix.

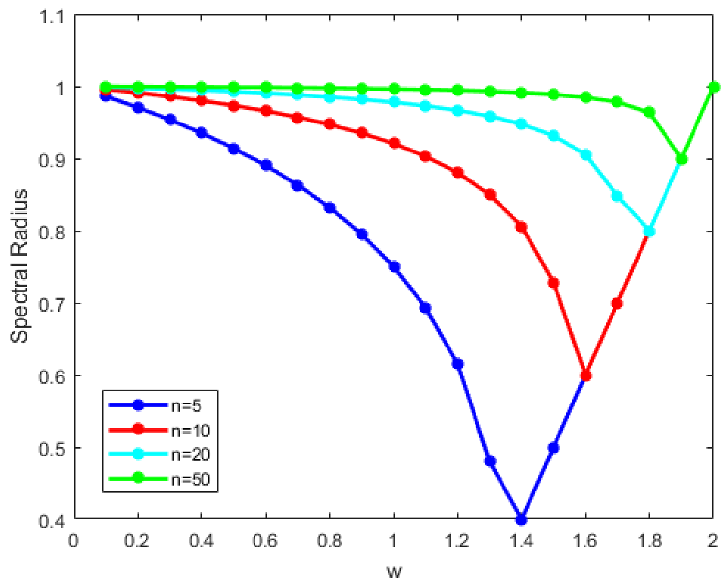
## 1.2 Solutions to part(c)

Spectral radius of matrix change with the size of the matrix, i.e. n = 5, 10, 20, and 50. The matlab function 'eig' solve for eigenvalues of a given matrix. Then the spectral radius equals to the element with largest absolute values. With the change of size of matrix ($n$), figure 1 produces a four different lines indicating the relation between spectral radius and the choice of $w$. Its code for finding and drawing spectal radius are shown below separately:

Listing 4: Code for Finding Spectral Radius

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Assignment 3 - "spec_radius"
%--------------------------
% Coder: Zelin Chen (797036)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function spec = spec_radius(C,k)
% storage space
spec = zeros(1,k);
w_val = zeros(1,k);
% fn LDU decomposite C into L, D and U
[L,D,U] = LDU(C);
for i = 1:k
    w_val(i) = 0.1 * i;
    % spectral radius = maximal eigenvalue of
       expression inside the bracket.
    spec(i) = max(abs(eig(( (1/w_val(i) * D + L)^(-1)
       * ( (1/w_val(i) -1) * D - U )))));
end
end
```

Figure 1: Plot of Spectral Radius




Listing 5: Code for Drawing Spectral Radius

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Assignment 3 - "Q1-c-figure"
%---------------------------
% Coder: Zelin Chen (797036)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;

% err = 0.1 to the power of 2, 3 and 4
err_power = [2,3,4];
% sample size = 5, 10, 20 and 50
n_size = [5,10,20,50];
len_n = length(n_size);

% w = 0.1 * k
k = 20;
w_val = zeros(1,k);
for i = 1:k
    w_val(i) = 0.1 * i;
end

% store value for each spectral radius
spec = zeros(len_n,k);
```

```
for i=1:len_n
    n = n_size(i);
    C = An(n);
    spec(i,:) = spec_radius(C,k);
end

figure
plot(w_val,spec(1,:),'b*-',w_val,spec(2,:),'r*-',w_val,spec(3,:),'c*-',w_val,spec(4,:),'g*-','LineWidth',2)
legend('n=5','n=10','n=20','n=50')
xlabel('w')
ylabel('Spectral Radius')
saveas(gcf,'spectral_radius_plot.png')
```

### 1.3 Solutions to part(d)

[1] Under stopping rule of 0.01, the choice on best $w$ for SOR is consistent for different $n$. When $w = 0.9$, SOR solve four different sizes of matrix by 6 rounds. When $n = 5$, it produce the same performance as Gauss-Seidel method (SOR($w = 1$)=6). Under the other three sizes of $n$, they are all one term shorter than Gauss-Seidel method (SOR($w = 1$)=7).

When increase preciseness of our answer to 0.001, the result shifted. Under $n = 5$, $w = 1.3$ reach the limit with lowest terms(10), while Guass-Seidel method (SOR($w = 1$)) takes 14 terms to converge. When $n = 10$, $w$ increases to 1.5 with only 18 terms, leave where SOR($w = 1$) takes 20 terms. For $n = 20$, SOR has four values of $w$ produce the shortest terms, including SOR($w = 1$). The result is same for $n = 50$ as $w = 1, 1.1, 1.2$, and 1.3 all have the same number of times (17) to converge.

The advantage of SOR method emergs when limit rises up to 0.0001. For $n = 5$, the quickest SOR ($w = 1.3and1.4$) takes only 13 terms, which is 9 terms quicker than Gauss-Seidel method. For $n = 10$, SOR($w = 1.6$) takes the 22 terms, where Gauss-Seidel method takes more than doubled (48 terms). For $n = 20$, SOR($w = 1.7$) produces the quickest result with only 38 iterations, where it takes Gauss-Seidel method to run up to 57 rounds. Finally $n = 50$, SOR($w = 1.6, 1.7$) produce the quickest convergence with 46 period of times, on the contrast, Gauss-Seidel method takes 50 terms.

[1]please refer to table 1 in appendix

## 2 Problem 2

Table 2[2] displays the result of convergence for different methods: Bisection, False Position, Secant and Newton's method. In order to produce the table, the stopping rule has been set to be equal to *iteration* $< 20$, instead of *error* $>$ 0.0005, because 20 the largest iteration among that four methods to stop at error of 0.00005. From table 2, we see that Bisection method takes the longest iteration to converge to a limit, 20 iterations. False Position method takes only 14 terms to converge. Comparatively, Secant and Newton's methods are more efficient, as Secant method takes 7 terms and Newton's method takes only 6 terms to converge. The code of each method is shown as following. (It also includes a main script to call each function and build the result into a latex table)

Listing 6: Code for Bisection Method

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Assignment 3 - "Bisection Method"
%---------------------------
% Coder: Zelin Chen (797036)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% function: f(x) = x^3 -2
% starting interval for search: [1,2]
% middle point = b - w(b-a), where w assumed to be 0.5.
function [vm,fm] = bisection()
a = 1;
b = 2;
w = 0.5;

% define space to store temporary value of errors and median points
error = 1;
m = 1;

% space to store result for each iteration, first two iterations set as
% defined
vm = zeros(15,1);
vm(1) = a;
vm(2) = b;
fm = zeros(15,1);
fm(1) = f(vm(1));
fm(2) = f(vm(2));
```

[2]please find it in appendix

```
term = 2;

while term<20 %error >0.00005
    % update new point
    term = term + 1;
    m = b - w*(b-a);
    vm(term) = m;
    error = abs(m - vm(term-1));

    % sign: if >0 TRUE=1, if <0 FALSE=0
    fm_sign = (f(m)>0);
    a_sign  = (f(a)>0);

    % update interval
    if fm_sign==a_sign
        a = m;
    else
        b = m;
    end
    % store results
    fm(term) = f(m);
    vm(term) = m;
end
end
```

Listing 7: Code for False Position

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Assignment 3 - "False Position"
%---------------------------
% Coder: Zelin Chen (797036)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% function: f(x) = x^3 -2
% starting interval for search: [1,2]
function [vm,fm] = false_position()
a = 1;
b = 2;
error = 1;

% space to store result for each iteration, first two
    iterations set as
% defined
vm = zeros(15,1);
vm(1) = a;
```

```
vm(2) = b;
fm = zeros(15,1);
fm(1) = f(vm(1));
fm(2) = f(vm(2));

term = 2;

while term<20 %error>0.00005
    term = term + 1;
    % update new point
    m = (a*f(b) - b*f(a))/(f(b) - f(a));
    error = abs(m - vm(term-1));
    % update interval
    if f(m)*f(a)<0
        b = m;
    else
        a = m;
    end
    % store result
    vm(term) = m;
    fm(term) = f(m);
end

end
```

Listing 8: Code for Secant Method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Assignment 3 - "Secant Method"
%---------------------------
% Coder: Zelin Chen (797036)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% function: f(x) = x^3 -2
% starting interval for search: [1,2]
function [vm,fm] = secant()
a = 1;
b = 2;

error = 1;
% space to store result for each iteration, first two iterations set as
% defined
vm = zeros(15,1);
vm(1) = a;
vm(2) = b;
```

```
fm = zeros(15,1);
fm(1) = f(vm(1));
fm(2) = f(vm(2));

term = 2;

while term<20 % error > 0.00005
    term = term + 1;
    % update values
    vm(term) = vm(term-2) - ( (vm(term-1)-vm(term-2))
       /(f(vm(term-1))-f(vm(term-2))) ) *f(vm(term-2))
       ;
    fm(term) = f(vm(term));
    error = abs(vm(term) - vm(term-1));
end
end
```

Listing 9: Code for Newton's Method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Assignment 3 - "Newton's Method"
%---------------------------
% Coder: Zelin Chen (797036)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% function: f(x) = x^3 -2
% starting interval for search: [1,2]
% Derivative: f'(x) = 3*x^2
function [vm,fm] = newton()
a = 1;
b = 2;
error = 1;

% space to store result for each iteration, first two
   iterations set as
% defined
vm = zeros(15,1);
vm(1) = a;
vm(2) = b;
fm = zeros(15,1);
fm(1) = f(vm(1));
fm(2) = f(vm(2));

term = 2;

while term < 20 % error > 0.00005
```

```
    term = term + 1;

    % update result given Derivative of funtion
    vm(term) = vm(term-1) - f(vm(term-1)) / df(vm(term-1));
    fm(term) = f(vm(term));
    error = abs(vm(term) - vm(term-1));

end
end
```

Listing 10: Main Script for Question 2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Assignment 3 - "Q2-main script-tables"
%---------------------------
% Coder: Zelin Chen (797036)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% call function
[b_vm,b_fm] = bisection();
[f_vm,f_fm] = false_position();
[s_vm,s_fm] = secant();
[n_vm,n_fm] = newton();

% create table
table.data = [b_vm b_fm f_vm f_fm s_vm s_fm n_vm n_fm];

table.tableColLabels = {'$x_n$','$f(x_n)$','$x_n$','$f(x_n)$','$x_n$','$f(x_n)$','$x_n$','$f(x_n)$'};
table.tableRowLabels ={'1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19','20'};

table.tableCaption = 'Table1: Comparison of Various Methods for Finding Zeros of $f(x)=x^3-2$';
if ~isfield(table,'dataNanString'),table.dataNanString = 'NaN';end

table.dataFormat = {'%.5f'};

table.makeCompleteLatexDocument = 1;

% generate LaTex code
latex = latexTable(table);
```

```matlab

% save LaTex code as file
fid=fopen('Q2_table.tex','w');
[nrows,ncols] = size(latex);
for row = 1:nrows
    fprintf(fid,'%s\n',latex{row,:});
end
fclose(fid);
fprintf('\n... your LaTex code has been saved as ''
   Q2_table.tex'' in your working directory\n');
```

# Appendices

Listing 11: Code for creating matrix C in Q1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Assignment 3 - "Create Matrix An"
%--------------------------
% Coder: Zelin Chen (797036)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [C,b] = An(n)

% create vector for b
b = zeros(n,1);
% create square matrix n*n
C = zeros(n,n);

% putting values into A_n
    for j = 1:n %row
        for k = 1:n % column
            if k==j
                C(j,k) = 2;
            elseif k==(j-1)
                C(j,k) = 1;
            elseif k==(j+1)
                C(j,k) = 1;
            else
                C(j,k) = 0;
            end
        end
    end
end
```

Listing 12: Code for function $f(x)$ in Q2

```
function y = f(x)
    y = x^3 - 2;
end
```

Listing 13: Code for derivative of function $f(x)$ in Q2

```
function y = df(x)
    y = 3*x^2;
end
```

Table 1: Number of Iterations of three methods for different Stopping Rule and Size n

| Method | n=5 | | | n=10 | | | n=20 | | | n=50 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0.01 | 0.001 | 0.0001 | 0.01 | 0.001 | 0.0001 | 0.01 | 0.001 | 0.0001 | 0.01 | 0.001 | 0.0001 |
| Gauss | 6 | 14 | 22 | 7 | 20 | 48 | 7 | 17 | 57 | 7 | 17 | 50 |
| Jacobi | 39 | 55 | 71 | 135 | 190 | 246 | 494 | 699 | 904 | 2919 | 4132 | 5344 |
| SOR(0.1) | 19 | 42 | 181 | 19 | 44 | 135 | 19 | 43 | 125 | 19 | 43 | 125 |
| SOR(0.2) | 13 | 34 | 111 | 13 | 31 | 90 | 13 | 31 | 91 | 13 | 31 | 91 |
| SOR(0.3) | 10 | 32 | 81 | 10 | 26 | 70 | 10 | 26 | 78 | 10 | 26 | 76 |
| SOR(0.4) | 8 | 28 | 63 | 9 | 23 | 62 | 9 | 23 | 69 | 9 | 23 | 68 |
| SOR(0.5) | 7 | 25 | 51 | 7 | 21 | 62 | 7 | 21 | 63 | 7 | 21 | 62 |
| SOR(0.6) | 7 | 22 | 42 | 7 | 20 | 63 | 7 | 20 | 58 | 7 | 20 | 58 |
| SOR(0.7) | 7 | 20 | 35 | 7 | 19 | 61 | 7 | 19 | 55 | 7 | 19 | 55 |
| SOR(0.8) | 7 | 18 | 30 | 7 | 19 | 57 | 7 | 18 | 54 | 7 | 18 | 53 |
| SOR(0.9) | 6 | 16 | 26 | 6 | 19 | 53 | 6 | 18 | 54 | 6 | 18 | 51 |
| SOR(1.0) | 6 | 14 | 22 | 7 | 20 | 48 | 7 | 17 | 57 | 7 | 17 | 50 |
| SOR(1.1) | 7 | 13 | 19 | 7 | 21 | 44 | 7 | 17 | 61 | 7 | 17 | 48 |
| SOR(1.2) | 7 | 12 | 17 | 9 | 21 | 39 | 9 | 17 | 62 | 9 | 17 | 47 |
| SOR(1.3) | 7 | 10 | 13 | 10 | 20 | 34 | 11 | 17 | 61 | 11 | 17 | 46 |
| SOR(1.4) | 8 | 11 | 13 | 11 | 19 | 29 | 14 | 20 | 58 | 14 | 20 | 46 |
| SOR(1.5) | 11 | 13 | 17 | 11 | 18 | 24 | 18 | 21 | 53 | 18 | 25 | 47 |
| SOR(1.6) | 12 | 18 | 23 | 14 | 21 | 22 | 21 | 23 | 46 | 24 | 34 | 51 |
| SOR(1.7) | 18 | 24 | 30 | 21 | 25 | 33 | 22 | 26 | 38 | 34 | 47 | 58 |
| SOR(1.8) | 29 | 39 | 48 | 32 | 42 | 52 | 33 | 42 | 54 | 51 | 52 | 68 |
| SOR(1.9) | 59 | 78 | 102 | 56 | 78 | 99 | 63 | 84 | 105 | 63 | 101 | 103 |
| SOR(2.0) | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

Table 2: Table1: Comparison of Various Methods for Finding Zeros of $f(x) = x^3 - 2$

| - | Bisection | | False Position | | Secant | | Newton | |
|---|---|---|---|---|---|---|---|---|
| n | $x_n$ | $f(x_n)$ | $x_n$ | $f(x_n)$ | $x_n$ | $f(x_n)$ | $x_n$ | $f(x_n)$ |
| 1 | 1.00000 | -1.00000 | 1.00000 | -1.00000 | 1.00000 | -1.00000 | 1.00000 | -1.00000 |
| 2 | 2.00000 | 6.00000 | 2.00000 | 6.00000 | 2.00000 | 6.00000 | 2.00000 | 6.00000 |
| 3 | 1.50000 | 1.37500 | 1.14286 | -0.50729 | 1.14286 | -0.50729 | 1.50000 | 1.37500 |
| 4 | 1.25000 | -0.04688 | 1.20968 | -0.22986 | 1.20968 | -0.22986 | 1.29630 | 0.17828 |
| 5 | 1.37500 | 0.59961 | 1.23884 | -0.09874 | 1.26504 | 0.02447 | 1.26093 | 0.00482 |
| 6 | 1.31250 | 0.26099 | 1.25116 | -0.04143 | 1.25971 | -0.00100 | 1.25992 | 0.00000 |
| 7 | 1.28125 | 0.10330 | 1.25630 | -0.01722 | 1.25992 | -0.00000 | 1.25992 | 0.00000 |
| 8 | 1.26563 | 0.02729 | 1.25842 | -0.00712 | 1.25992 | 0.00000 | 1.25992 | 0.00000 |
| 9 | 1.25781 | -0.01002 | 1.25930 | -0.00294 | 1.25992 | 0.00000 | 1.25992 | 0.00000 |
| 10 | 1.26172 | 0.00857 | 1.25967 | -0.00121 | 1.25992 | 0.00000 | 1.25992 | 0.00000 |
| 11 | 1.25977 | -0.00074 | 1.25982 | -0.00050 | NaN | NaN | 1.25992 | 0.00000 |
| 12 | 1.26074 | 0.00391 | 1.25988 | -0.00021 | NaN | NaN | 1.25992 | 0.00000 |
| 13 | 1.26025 | 0.00159 | 1.25990 | -0.00009 | NaN | NaN | 1.25992 | 0.00000 |
| 14 | 1.26001 | 0.00042 | 1.25991 | -0.00004 | NaN | NaN | 1.25992 | 0.00000 |
| 15 | 1.25989 | -0.00016 | 1.25992 | -0.00001 | NaN | NaN | 1.25992 | 0.00000 |
| 16 | 1.25995 | 0.00013 | 1.25992 | -0.00001 | NaN | NaN | 1.25992 | 0.00000 |
| 17 | 1.25992 | -0.00001 | 1.25992 | -0.00000 | NaN | NaN | 1.25992 | 0.00000 |
| 18 | 1.25993 | 0.00006 | 1.25992 | -0.00000 | NaN | NaN | 1.25992 | 0.00000 |
| 19 | 1.25993 | 0.00002 | 1.25992 | -0.00000 | NaN | NaN | 1.25992 | 0.00000 |
| 20 | 1.25992 | 0.00000 | 1.25992 | -0.00000 | NaN | NaN | 1.25992 | 0.00000 |