Numerical Approximation

Experimental Report 3

Tutor: Wu Leqin
Student: Zeng Deyi
Student ID: [redacted]
Academy: JBJI
Major: Information and Computer Science

May 28, 2022

Contents

1 How to run the codes and verify the correctness of answer in this project? 2

2 Experimental Purpose and Requirement 5

3 Experimental Principle and Main Content 5

4 Main Equipment 5

5 Minimax Approximation on Discrete Points: Procedure and Codes 5
5.1 Formulate the questions as simultaneous linear equations 6
5.2 Solve the simultaneous linear equations by Gaussian Elimination 6
5.3 Change the Reference 7
5.4 Termination Condition 8
5.5 Handle the basic exceptions 8
5.6 Codes 9

6 Minimax Approximation on Discrete Points: Results, Analysis and Visualization 11
6.1 Case 1: A trivial test 11
6.2 Case 2: Visualization of Results 12
6.3 Case3: Test for Sensitivity to Initial Reference 12
6.4 Analysis 14

7 Minimax Approximation on Continuous Function: Procedure and Codes 14
7.1 Formulate the questions as simultaneous linear equations 15
7.2 Solve the simultaneous linear equations by Gaussian Elimination 15
7.3 Prerequisite: Find a local extremum x_0 such that $|e(x_0)| > |h|$, where e is a specified continuous function and h is a value 16
7.4 Change the Reference 19
7.5 Termination Condition 19
7.6 Handle the basic exceptions 20

7.7 Codes 20

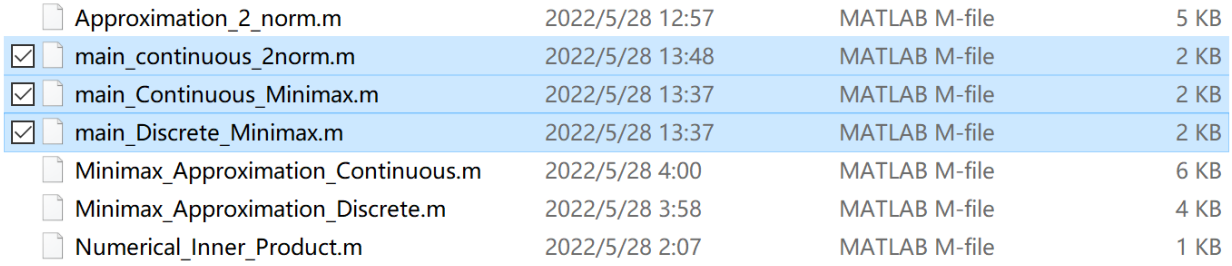
8 Minimax Approximation on Continuous Function: Results, Analysis and Visualization 24
8.1 Case 1: A Trivial Test 24
8.2 Case 2: Visualization of Results 24
8.3 Case3: Test for Sensitivity to Initial Reference 27
8.4 Analysis 27

9 2-norm Approximation on Continuous Function: Procedure and Codes 27
9.1 Numerical Weighted Inner Product 27
9.2 Codes: Numerical weighted Inner Product 28
9.3 Handle the basic exceptions 28
9.4 Procedure of the Algorithm 29
9.5 Codes 29

10 2-norm Approximation on on Continuous Function: Results, Analysis and Visualization 32
10.1 Case1: A normal test on 2-norm 32
10.2 Case2 : Visualization of Results, 2-norm 32
10.3 Case 3: A more general norm, induced by weighted inner product 34
10.4 Analysis 35

1 How to run the codes and verify the correctness of answer in this project?

The files highlighted in blue are main functions, you can run them directly, to obtain the results in experimental report. The others are auxiliary functions, just make sure that the exist, having no need to run them.



The test cases in this experimental report comes from some website or textbook, I have attached the corresponding screenshots of the examples in the files, by comparing the screenshots with the results you can verify the correctness of my solutions.

例2 求函数 $f(x) = e^x$ 在 [-1,1] 上的二次多项式逼近.

解 显然, 最佳逼近多项式具有如下形式:

$$p(x) = a + bx + cx^2.$$

据不同的初值选取, 得表 4.2.

表 4.2

初始点组	系数a, b, c	迭代步数	最佳逼近
$-1, -0.5, 0.5, 1$	0.980398,1.130184,0.5540408	4	0.04501730
$-1, 0, 0.5, 1$	0.9890399,1.1301840,0.5540408	3	0.04501742
$-0.8, -0.3, 0.4, 0.7$	0.9890398,1.1301840,0.5540408	4	0.4501724

8.3 Case3: Test for Sensitivity to Initial Reference

Consider the following example: $[a, b] = [-1, 1]$, $f(x) = e^x$ $p(x) = p_0 + p_1 x + p_2 x^2$. We use 5 groups of initial references. In this example, we want to verify that the continuous exchange algorithm is also not sensitive to the initial reference. We will only provide the final results of each initial reference, rather than the whole procedure.

Table 6: Threshold :$\delta = 10^{-11}$

Initial Reference				Termination Reference				Iteration Epoch	Coefficient		
ξ_0	ξ_1	ξ_2	ξ_3	ξ_0	ξ_1	ξ_2	ξ_3		p_0	p_1	p_2
-1	-0.5	0.5	1	-1	-0.43695788	0.56005776	1	5	0.98914108	1.13086433	0.55363656
-1	0	0.5	1	-1	-0.43695762	0.56005776	1	6	0.98903973	1.13018381	0.55404091
-0.8	-0.3	0.4	0.7	-1	-0.43693924	0.56005776	1	8	0.98903973	1.13018381	0.55404091
-1	-0.9	-0.8	-0.7	-1	-0.43695063	0.56005776	1	9	0.98903973	1.13018381	0.55404091
0.7	0.8	0.9	1	-1	-0.43695797	0.56005776	1	12	0.98903973	1.13018381	0.55404091

例: 设$f(x)=\sqrt{1+x^2}$, 求[0,1]上得一次最佳平方逼近多项式。

解: 得$d_0=\int_0^1\sqrt{1+x^2}dx=\frac{1}{2}ln(1+\sqrt{2})\approx 1.147$,$d_1=\int_0^1 x\sqrt{1+x^2}dx\approx 0.609$, 得方程组:

$$\begin{bmatrix}1 & 1/2\\ 1/2 & 1/3\end{bmatrix}\begin{bmatrix}a_0\\ a_1\end{bmatrix}=\begin{bmatrix}1.147\\ 0.609\end{bmatrix}$$

解得$a_0=0.934, a_1=0.426$, 故$S_1^*(x)=0.934+0.426x$

10.1 Case1: A normal test on 2-norm

We will use the following example: $[a,b]=[0,1], f(x)=\sqrt{1+x^2}\ w(x)=1, p(x)=p_0+p_1x$

Epoch	i	$\phi_j=\sum_{j=0}^{i}p_j\phi_j$		α	β	Coefficient	
		p_0	p_1			p_0	p_1
1	0	1	\	0.5	\	1.147794	\
2	1	-0.5	1	0.5	0.0833	0.934320	0.426947

2 Experimental Purpose and Requirement

Students should have a better understanding of the exchange algorithm and 2-norm approximation algorithm.

3 Experimental Principle and Main Content

Question 1: Implement a Matlab function which completes the following task: Give the minimax approximation of a given function f on (a)a closed interval; (b)a set of finitely many points.

Solution 1: (a) Section 7 (b) Section 5

Question 2: Implement a Matlab function which completes the following task: Give the best approximation of a given function f on a closed interval with respect to 2-norm (or more generally, a norm induced by an inner product).

Solution 2: Section 9, the weighted inner product is applicable for both 2-norm and general norm induced by inner product.

Question 3: Basic exceptions should be handled.

Solution 3: Section 5.5, Section 7.6, Section 9.3

Question 4: Use some simulation data to test your codes, and visualize your results.

Solution 4: Test the codes: Section 6, Section 8, Section 10.1 and Section 10.2(2-norm), Section 10.3(norm induced by inner product); Visualize results: Section 6.3, Section 8.2, Section 10.2, Section 10.3

4 Main Equipment

Computer and Matlab

5 Minimax Approximation on Discrete Points: Procedure and Codes

This section solves the Question 1(b) of this project. We first restate this problem: Given a set of points $\{x_i; i = 1, 2, ..., m\}$, that are in ascending order $a \leqslant x_1 < x_2 < ... < x_m \leqslant b$, and a function f in $\mathscr{C}[a, b]$. We need to find a p in P_n such that minimizes the discrete maximum error.

We need to be careful here, as we are optimizing on discrete points.

Introduce an important variable called 'indices'. Since we are given a set of discrete point: (x, y), we need to index all pairs according to their ascending order: 1,2,...,m. And the indices corresponding to reference is variable 'indices' = $\{I_0, I_1, ..., I_{n+1}\}$. With this useful variable we have access to all values of x and y. 'reference' is a variable with length (n+2), representing $[\xi_0, \xi_1, \xi_2, ..., \xi_{n+1}]$.Notice that the 'reference' is essentially part of the x.

5.1 Formulate the questions as simultaneous linear equations

We will formulate the problem in the following form:

$$\begin{cases} f(\xi_0) - p_0 - p_1\xi_0 - p_2\xi_0^2 - \cdots - p_n\xi_0^n &= -h \\ f(\xi_1) - p_0 - p_1\xi_1 - p_2\xi_1^2 - \cdots - p_n\xi_1^n &= h \\ \vdots & \vdots \\ f(\xi_{n+1}) - p_0 - p_1\xi_{n+1} - p_2\xi_{n+1}^2 - \cdots - p_n\xi_{n+1}^n &= (-1)^n h \end{cases}$$

$(\Rightarrow)$

$$\begin{cases} -h + p_0 + p_1\xi_0 + p_2\xi_0^2 + \cdots + p_n\xi_0^n &= f(\xi_0) \\ h + p_0 + p_1\xi_1 + p_2\xi_1^2 + \cdots + p_n\xi_1^n &= f(\xi_1) \\ \vdots & \vdots \\ (-1)^n h + p_0 + p_1\xi_{n+1} + p_2\xi_{n+1}^2 + \cdots + p_n\xi_{n+1}^n &= f(\xi_{n+1}) \end{cases}$$

$(\Rightarrow)$

$$\begin{cases} -h + p_0 + \xi_0 p_1 + \xi_0^2 p_2 + \cdots + \xi_0^n p_n &= f(\xi_0) \\ h + p_0 + \xi_1 p_1 + \xi_1^2 p_2 + \cdots + \xi_1^n p_n &= f(\xi_1) \\ \vdots & \vdots \\ (-1)^n h + p_0 + \xi_{n+1} p_1 + \xi_{n+1}^2 p_2 + \cdots + \xi_{n+1}^n p_n &= f(\xi_{n+1}) \end{cases}$$

This simultaneous linear equations can be solved by Gaussian Elimination, the readers could refer to codes for details.

5.2 Solve the simultaneous linear equations by Gaussian Elimination

Firstly, we convert the simultaneous linear equations to be coefficient matrix:

$$A = \begin{pmatrix} -1 & \xi_0 & \xi_0^2 & \dots & \xi_0^n \\ 1 & \xi_1 & \xi_1^2 & \dots & \xi_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (-1)^n & \xi_{n+1} & \xi_{n+1}^2 & \dots & \xi_{n+1}^n \end{pmatrix} \quad \beta = \begin{pmatrix} f(\xi_0) \\ f(\xi_1) \\ \vdots \\ f(\xi_{n+1}) \end{pmatrix}$$

We do not discuss the theory of Gaussian Elimination in this section. In the programming, we will combine matrix A and b to an augmented matrix, and perform Gaussian Elimination to render the diagonal elements to be 1 and the other elements except the last column to be 0(which is exactly Gaussian Elimination). Then we extract the elements of the last columns to be h and p.

5.3 Change the Reference

There are many choices of new reference, while we use 'one-point exchange algorithm' in this project. In other words, only one component of reference will be changed in the process of each iteration. The chosen criteria is quite simple: we choose a new $x_i \in \{x_1, x_2, ...x_m\}$ with the largest magnitude (or say absolute value) of error $e(x) = f(x) - p(x)$, to be the new component of reference, denoted as x_{enter}. Since there are finitely many points, we can go through the value of $e(x)$ on each point of the given x: $\{x_1, x_2, ...x_m\}$ and obtain that one with the largest absolute value. And we will remove one component of the old reference, denoted as x_{leave}, such that the size of new reference remains to be $(n+1)$:

- If $x_{enter} < \xi_0$ and the sign of $e(x_{enter})$ is the same as the sign of $e(\xi_0)$, the new reference is constructed as: $\{x_{enter}, \xi_1, \xi_2 ...\xi_{n+1}\}$
- If $x_{enter} < \xi_0$ and the sign of $e(x_{enter})$ is the same as the sign of $-e(\xi_0)$, the new reference is constructed as: $\{x_{enter}, \xi_0, \xi_1 ...\xi_n\}$
- If $x_{enter} > \xi_{n+1}$ and the sign of $e(x_{enter})$ is the same as the sign of $e(\xi_{n+1})$, the new reference is constructed as: $\{\xi_0, \xi_1 ...\xi_n, x_{enter}\}$
- If $x_{enter} > \xi_{n+1}$ and the sign of $e(x_{enter})$ is the same as the sign of $-e(\xi_{n+1})$, the new reference is constructed as: $\{\xi_1, \xi_2 ...\xi_{n+1}, x_{enter}\}$
- If $\xi_i < x_{enter} < \xi_{i+1}$ for an $i \in \{0, 1, ..., n\}$ and the sign of $e(x_{enter})$ is the same as the sign of $e(\xi_i)$, the new reference is constructed as: $\{\xi_0, \xi_1, ...\xi_{i-1}, x_{enter}, \xi_{i+1}, ..., \xi_{n+1}\}$
- If $\xi_i < x_{enter} < \xi_{i+1}$ for an $i \in \{0, 1, ..., n\}$ and the sign of $e(x_{enter})$ is the same as the sign of $e(\xi_{i+1})$, the new reference is constructed as: $\{\xi_0, \xi_1, ...\xi_i, x_{enter}, \xi_{i+2}, ..., \xi_{n+1}\}$

Since the $e(\xi_i) = (-1)^{i-1}h$ for $i \in \{0, 1, ..., n+1\}$. We know that the sign of $e(\xi_i)$ is the same as the sign of $(-1)^{i-1}h$. Additionally, since each x is monotonically one-to-one mapped to an index, and ξ is essentially part of x. We can replace the ξ_i with I_i, and x_{enter} with its corresponding index, denoted as I_{enter}, where $x_{enter} = x(I_{enter})$:

- If $I_{enter} < I_0$ and the sign of $e(x(I_{enter}))$ is the same as the sign of $-h$, the new indices is constructed as: $\{I_{enter}, I_1, I_2, ..., I_{n+1}\}$
- If $I_{enter} < I_0$ and the sign of $e(x(I_{enter}))$ is the same as the sign of h, the new indices is constructed as: $\{I_{enter}, I_0, I_1, ..., I_n\}$

- If $I_{enter} > I_{n+1}$ and the sign of $e(x(I_{enter}))$ is the same as the sign of $(-1)^n h$, the new indices is constructed as: $\{I_0, I_1, ..., I_n, I_{enter}\}$
- If $I_{enter} > I_{n+1}$ and the sign of $e(x(I_{enter}))$ is the same as the sign of $(-1)^{n-1} h$, the new indices is constructed as: $\{I_1, I_2, ..., I_{n+1}, I_{enter}\}$
- If $I_i < I_{enter} < I_{i+1}$ for an $i \in \{0, 1, ..., n\}$ and the sign of $e(x(I_{enter}))$ is the same as the sign of $(-1)^{i-1} h$, the new indices is constructed as: $\{I_0, I_1, ..., I_{i-1}, I_{enter}, I_{i+1}, ..., I_{n+1}\}$
- If $I_i < I_{enter} < I_{i+1}$ for an $i \in \{0, 1, ..., n\}$ and the sign of $e(x(I_{enter}))$ is the same as the sign of $(-1)^i h$, the new indices is constructed as: $\{I_0, I_1, ..., I_i, I_{enter}, I_{i+2}, ..., I_{n+1}\}$

5.4 Termination Condition

There is only finite references, and the h can strictly increase in every iteration, the largest h will be achieved in finitely many iterations. However, because of the computational rounding error when performing the Gaussian Elimination. The rounding error results in that the $|h|$ can never be strictly equal to the absolute extremum of the error function, which holds theoretically. Therefore, a threshold should be established to control the error tolerance. However, sometimes the error tolerance can never be achieved also because of the rounding error. We should state a maximum iteration epochs. The termination condition can also be:

$$\max_{i=1,2,..,m} |x_i - p(x_i)| \ - |h| < \delta$$

5.5 Handle the basic exceptions

```
if length(x) ~= length(y)
    error('The size of x and y is not the same!')
end

if length(x) < 0
    error('Invalid size x!')
end

if length(y) < 0
    error('Invalid size y!')
end

if n < 0
    error('Invalid n!')
end

if length(indices) ~= n+2
    error('Invalid size of initial indices and reference!')
end
```

```
20
21 if threshold < 0
22     error('Invalid threshold value, please choose a non-zero threshold
           value!')
23 end
```

5.6 Codes

```
1  function [p,p_table, indices_table , reference_table ,h_table ,extremum_table] =
           Minimax_Approximation_Discrete(x,y,n,indices,threshold,epoch_max)
2
3  % Input
4  % x: Abscissa of discrete points
5  % y: Ordinate of discrete points
6  % n: Order of Approximation polynomial
7  % indices: The initial indices of iterations
8  % threshold: When the inf-norm Approximation Error is less than threshold, the
           program terminates
9  % epoch_max: When the maximum epoch is achieved, the program terminates
10
11 % Output
12 % p: The final approximation polynomial
13 % p_table: The approximation polynomial in each iteration
14 % indices_table: The indices of reference in each iteration
15 % reference_table: The reference in each iteration
16 % h_table: The levelled error h in each iteration
17 % extremum_table: The extremum of error function in each iteration
18
19 % Handle the basic exception
20 if length(x) ~= length(y)
21     error('The size of x and y is not the same!')
22 end
23
24 if length(x) < 0
25     error('Invalid size x!')
26 end
27
28 if length(y) < 0
29     error('Invalid size y!')
30 end
31
32 if n < 0
33     error('Invalid n!')
34 end
35
36 if length(indices) ~= n+2
37     error('Invalid size of initial indices and reference!')
38 end
39
```

```
40 if threshold < 0
41     error('Invalid threshold value, please choose a non-zero threshold
           value!')
42 end
43
44 %
45 for epoch = 1:epoch_max
46
47     % Establish the indices and reference Matrices
48     reference = x(indices);
49     indices_table(epoch,1:n+2) = indices;
50     reference_table(epoch,1:n+2) = reference;
51
52     % Establish the coefficients Matrices A and beta
53     A = ones(n+2,2);
54     for i = 1:n+2
55         A(i,1) = (-1)^i;
56     end
57     for j = 3:n+2
58         for i = 1:n+2
59             A(:,j) = reference.^(j-2);
60         end
61     end
62     beta = y(indices)';
63     A_beta = [A,beta];
64
65     % Gaussian Elimination to solve the simultaneous equations
66     for centre = 1:n+2
67         A_beta(centre,:) = A_beta(centre,:)./A_beta(centre,centre);
68         for j = 1:size(A_beta,1)
69             if centre~=j
70                 A_beta(j,:) = A_beta(j,:) - A_beta(centre,:) *A_beta(j,centre);
71             end
72         end
73     end
74
75     % The first end of row is the value of h, and the remaining are coefficients of p in an
           ascending order(p_0,p_1,...,p_n)
76     h = A_beta(1,end);
77     h_table(epoch) = h;
78     p = A_beta(2:end,end);
79     p_table(epoch,1:n+1) = p';
80
81     % Find the x_enter with the largest magnitude of error under the p
82     extremum_abs = -inf;
83     key = 0;
84     for i = 1:length(x)
85         if abs(y(i)-x(i).^(0:n)* p) > extremum_abs
86             extremum_abs = abs(y(i)-x(i).^(0:n)* p);
87             extremum_sign = sign(y(i)-x(i).^(0:n)* p);
88             key = i;
```

```
        end
    end
    extremum_table(epoch) = extremum_abs * extremum_sign;

    % Termination Condition
    if (extremum_abs - abs(h)) < threshold
        disp('Terminates! We have reached the threshold!')
        break
    end

    % Change the Reference
    if key < indices(1) && extremum_sign == -1*sign(h)
        indices(1) = key;
    elseif key < indices(1) && extremum_sign == sign(h)
        indices(2:n+2) = indices(1:n+1);
        indices(1) = key;
    elseif key > indices(n+2) && extremum_sign == (-1)^(n+2)*sign(h)
        indices(n+2) = key;
    elseif key > indices(n+2) && extremum_sign == (-1)^(n+1)*sign(h)
        indices(1:n+1) = indices(2:n+2);
        indices(n+2) = key;
    elseif key < indices(n+2) && key > indices(1)
        %To discover the location of x_enter
        %[1,2] means the x corresponding to key is located between references 1 and 2
        location = [1,2];
        while key > indices(location(2))
            location = location + 1;
        end

        if extremum_sign == (-1)^location(1)*sign(h)
            indices(location(1)) = key;
        elseif extremum_sign == (-1)^location(2)*sign(h)
            indices(location(2)) = key;
        end
    end

end
```

6 Minimax Approximation on Discrete Points: Results, Analysis and Visualization

6.1 Case 1: A trivial test

We consider the following example: $x = -1 : 0.01 : 1$, $y = 2x^3 + x^2 + x - 1$, $p(x) = p_0 + p_1 x + p_2 x2 + p_3 x^3 + p_4 x^4$,initial reference $= \{-1, -0.99, -0.98, -0.97, -0.96, -0.95\}$

Table 1: $\delta = 10^{-12}$

Epoch	Reference						h	Error Extremum	Coefficients				
	ξ_0	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5			p_0	p_1	p_2	p_3	p_4
1	-1.00	-0.99	-0.98	-0.97	-0.96	-0.95	6.25e-17	2.10e-8	-1	1	1	2	0
2	-1.00	-0.99	-0.98	-0.97	-0.96	1.00	8.30e-17	-2.98e-11	-1	1	1	2	0
3	-1.00	-0.99	-0.98	-0.97	0.51	1.00	1.10e-16	1.98e-13	-1	1	1	2	0
4	-1.00	-0.99	-0.98	-0.16	0.51	1.00	1.65e-16	1.78e-15	-1	1	1	2	0

6.2 Case 2: Visualization of Results

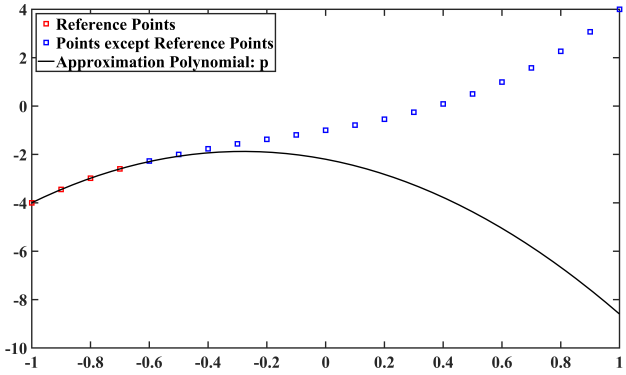
We consider the following example: $x = -1 : 0.1 : 1$, $y = 2x^3 + x^2 + 2x - 1$, $p(x) = p_0 + p_1 x + p_2 x^2$initial reference $= \{-1, -0, .9, -0.8, -0.7, -0.6, -0.5\}$ Notice that this function is a bit different from the case 1.

Table 2: $\delta = 10^{-12}$

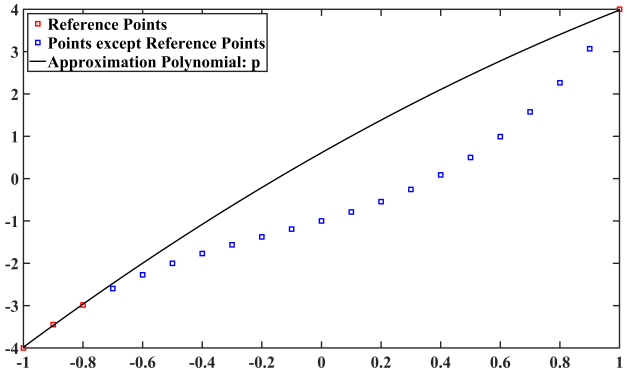
Epoch	Reference				h	Error Extremum	Coefficients		
	ξ_0	ξ_1	ξ_2	ξ_3			p_0	p_1	p_2
1	-1.0	-0.9	-0.8	-0.7	1.5e-03	12.60	-2.20	-2.30	-4.10
2	-1.0	-0.9	-0.8	1.0	1.9e-02	-2.02	0.61	3.98	-0.61
3	-1.0	-0.9	0.4	1.0	2.2e-01	0.96	-1.43	3.78	1.43
4	-1.0	-0.5	0.4	1.0	4.9e-01	-0.52	-0.98	3.51	0.98
5	-1.0	-0.5	0.5	1.0	5.0e-01	-0.50	-1.00	3.50	1.00

6.3 Case3: Test for Sensitivity to Initial Reference

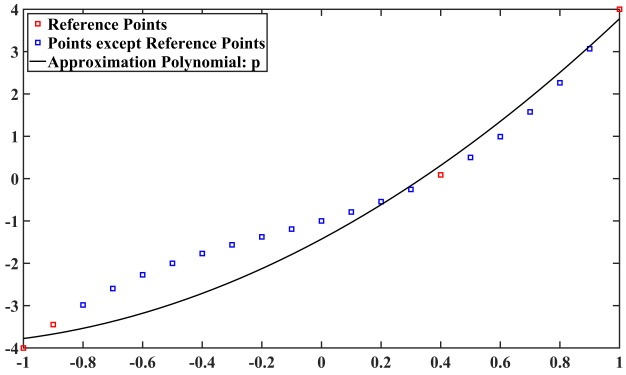
Consider the following example: $x = -1 : 0.1 : 1$, $y = e^x$, $p(x) = p_0 + p_1 x + p_2 x^2$. We use 5 groups of initial references. In this example, we want to verify that the discrete exchange algorithm is not sensitive to the initial reference. We will only provide the final results of each initial reference, rather than the whole procedure.



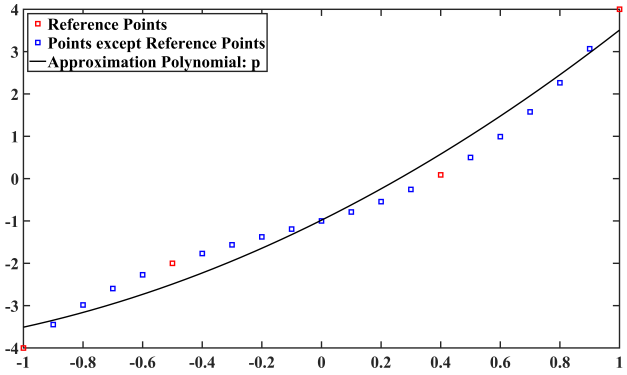
(a) Epoch 1



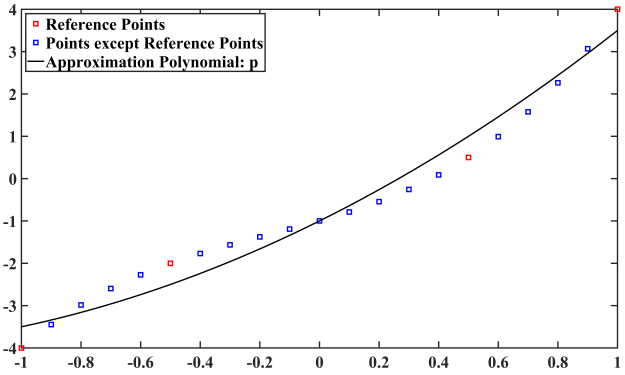
(b) Epoch 2



(c) Epoch 3



(d) Epoch 4



(e) Epoch 5

Table 3: $\delta = 10^{-12}$

Initial Reference				Termination Reference				Iteration Epoch	Coefficient		
ξ_0	ξ_1	ξ_2	ξ_3	ξ_0	ξ_1	ξ_2	ξ_3		p_0	p_1	p_2
-1	-0.9	-0.8	-0.7	-1	-0.4	0.6	1	5	0.989150	1.130472	0.553930
0.7	0.8	0.9	1	-1	-0.4	0.6	1	5	0.989150	1.130472	0.553930
-0.8	-0.4	0.2	0.8	-1	-0.4	0.6	1	4	0.989150	1.130472	0.553930
-1	-0.4	0.8	1	-1	-0.4	0.6	1	2	0.989150	1.130472	0.553930
-1	-0.4	0.6	1	-1	-0.4	0.6	1	0	0.989150	1.130472	0.553930

6.4 Analysis

The discrete ∞-norm approximation achieves high accuracy, when the split of the data point is reasonable. Additionally, we spot that the convergence speed of the Discrete Exchange Algorithm is rather fast. We can also conclude that the final results is not sensitive to the initial reference. Whatever which initial point you choose, the results will eventually converge to the real one.

7 Minimax Approximation on Continuous Function: Procedure and Codes

This section is roughly the same as the case in the discrete points. However, there are some difficulty in searching for the maximum of a continuous function (As we cannot go through infinitely many points in a closed interval), according to the textbook. Some reference book reminds us of using Newton Method and Gradient Descent Method, by approximating the global extremum with local extremum. However, the Newton Method and Gradient Descent Method are not the best choice. Firstly, the Newton Method requires the second derivative of the continuous function. In computer, we cannot really compute the derivative of a function, we are using the definition of derivative, by taking the epsilon small enough to approximate the derivative, which suffers from both rounding error and approximation loss. When taking the second derivative, the error will be enlarged twice. Additionally, the Gradient descent Method requires the stride to be pre-decided. Sometimes the result will be sensitive to the stride. One of the other problem is that sometimes the gradient descent method and newton method falls into a local extremum (maximum or minimum) which is not larger than absolute value of h. In this case, we may discover that the absolute value of h is even less than the local maximum you found, which freezes the procedure. Sometimes the gradient descent and newton method will also fall into some points with zero derivative but the derivative does not change sign at that zero(For instance, the derivative of function $f(x) = x^3$ is zero

at point $x = 0$, but does not change sign), and this zero of derivative is even not an local extremum of the function.

According to the reasons stated above, we recommend to use Binary Search method in this task.

7.1 Formulate the questions as simultaneous linear equations

This step is completely the same as the discrete case:

$$\begin{cases} f(\xi_0) - p_0 - p_1\xi_0 - p_2\xi_0^2 - \cdots - p_n\xi_0^n &= -h \\ f(\xi_1) - p_0 - p_1\xi_1 - p_2\xi_1^2 - \cdots - p_n\xi_1^n &= h \\ \vdots & \vdots \\ f(\xi_{n+1}) - p_0 - p_1\xi_{n+1} - p_2\xi_{n+1}^2 - \cdots - p_n\xi_{n+1}^n &= (-1)^n h \end{cases}$$

$(\Rightarrow)$

$$\begin{cases} -h + p_0 + p_1\xi_0 + p_2\xi_0^2 + \cdots + p_n\xi_0^n &= f(\xi_0) \\ h + p_0 + p_1\xi_1 + p_2\xi_1^2 + \cdots + p_n\xi_1^n &= f(\xi_1) \\ \vdots & \vdots \\ (-1)^n h + p_0 + p_1\xi_{n+1} + p_2\xi_{n+1}^2 + \cdots + p_n\xi_{n+1}^n &= f(\xi_{n+1}) \end{cases}$$

$(\Rightarrow)$

$$\begin{cases} -h + p_0 + \xi_0 p_1 + \xi_0^2 p_2 + \cdots + \xi_0^n p_n &= f(\xi_0) \\ h + p_0 + \xi_1 p_1 + \xi_1^2 p_2 + \cdots + \xi_1^n p_n &= f(\xi_1) \\ \vdots & \vdots \\ (-1)^n h + p_0 + \xi_{n+1} p_1 + \xi_{n+1}^2 p_2 + \cdots + \xi_{n+1}^n p_n &= f(\xi_{n+1}) \end{cases}$$

This simultaneous linear equations can be solved by Gaussian Elimination, the readers could refer to codes for details.

7.2 Solve the simultaneous linear equations by Gaussian Elimination

Firstly, we convert the simultaneous linear equations to be coefficient matrix:

$$A = \begin{pmatrix} -1 & \xi_0 & \xi_0^2 & \dots & \xi_0^n \\ 1 & \xi_1 & \xi_1^2 & \dots & \xi_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (-1)^n & \xi_{n+1} & \xi_{n+1}^2 & \dots & \xi_{n+1}^n \end{pmatrix} \quad \beta = \begin{pmatrix} f(\xi_0) \\ f(\xi_1) \\ \vdots \\ f(\xi_{n+1}) \end{pmatrix}$$

We do not discuss the theory of Gaussian Elimination in this section. In the programming, we will combine matrix A and b to an augmented matrix, and perform Gaussian Elimination to render the diagonal elements to be 1 and the other elements except the last column to be 0(which is exactly Gaussian Elimination). Then we extract the elements of the last columns to be h and p.

7.3 Prerequisite: Find a local extremum x_0 such that $|e(x_0)| > |h|$, where e is a specified continuous function and h is a value

Our task is as follows: Search for a local extremum whose absolute value is larger than $|h|$ The main idea is that the global extremum $\max_{x\in[a,b]} \|f - p\|$ decreases and the levelled error h increases in each iteration. We will also approximate the global extremum with an local extremum such that the absolute of this local extremum is larger than $|h|$. This can be achieved by Continuous Binary Search on the closed interval. We need some theoretical basis of this question: The global extremum (maximum and minimum) of a closed continuous interval $[a, b]$ must be occur in one of the following three cases:

- Left boundary point $x = a$
- Right boundary point $x = b$
- Zero point (root) of the derivative function and derivative changes sign in this point

The Procedure of the Continuous Binary Search is simple but hard to understand, so we will use an example for clear understanding. Consider the following example:

$$-70.0000 + 160.0714x - 137.0972x^2 + 62.2958x^3 - 16.6806x^4 + 2.6250x^5 - 0.2222x^6 + 0.0077x^7$$

We have draw the graphs of iterations and functions in the next page. In the first step, we can easily check that $|e(a)|$ and $|e(b)|$ are less than $|h|$, so we know that the two boundary points are not a point larger than $|h|$. We search for local extremum instead. The local extremum, whatever maximum or minimum, are achieved when the derivative equals to zero and changes sign at a certain point. One of the advantage of the Binary Search for finding roots is that it will not achieve the zero point that does not change sign, and this is what we are expecting (if the sign of derivative does not change in the zero, this point is not a

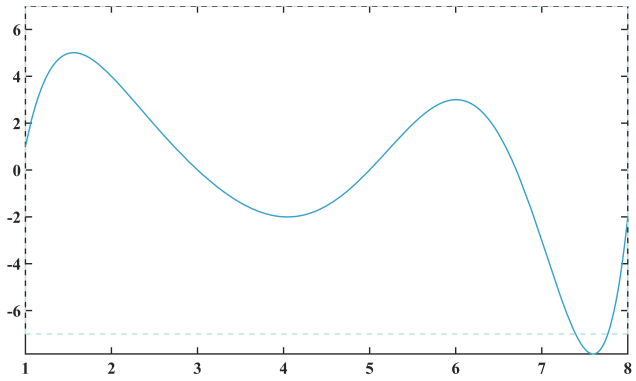
local extremum of the original function, just like $x = 0$ of $f = x^3$). Secondly, we use the binary search method to find the zero of the derivative function e'. Theoretically, if value of derivatives on two boundary points [a,b] has different sign and non-zero, we can conclude that there must be at least one root in this interval(Notice that 'at least' implies there can be more than one roots). By using binary search the algorithm converges extremely fast to one of those roots. However, if the one of the derivative of the boundary point point is zero, or the signs of the derivative on the boundary points are the same, we cannot make sure that there are no root in this interval. Therefore, we continuously halve this interval to see whether there are boundary points with different signs. Let us go bakc to the example:

- Epoch 1: The boundary points $x = 1$ and $x = 8$ have derivatives $e'(1) > 0$, $e'(8) > 0$. The signs are the same. So in this step we cannot implement the binary search directly. Instead, we halve the interval to be $[1, 4.5]$ and $[4.5, 8]$, to see whether there are some sub-intervals have derivative $e'(x)$ with different signs on boundary points.

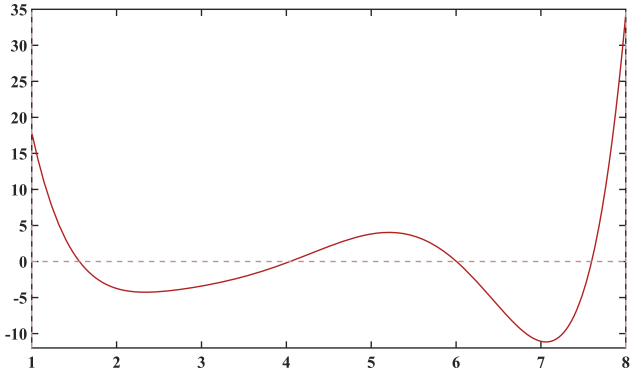
- Epoch 2: The boundary points of the first interval $[1, 4.5]$ are $x = 1$ and $x = 4.5$ with $e'(1) > 0$, $e'(4.5) > 0$. So we cannot conclude that there must be a root in this sub-interval. We just leave it alone and check the other sub-interval $[4.5, 8]$. Unfortunately, $e'(4.5) > 0$ and $e'(8) > 0$, we cannot implement the binary search as well. We halve both intervals and goes to Epoch 3.

- Epoch 3: There are four sub-intervals now: $[1, 2.75]$,$[2.75, 4.5]$,$[4.5, 6.25]$,$[6.25, 8]$. Spotting that $e'(1) > 0$, $e'(2.75) < 0$ we conclude that there must be at least one roots in this interval. Using binary search we can find it quickly, where $root = 1.561686$ (round to 6 decimals). We have labeled it on the graph. The corresponding value $e'(1.561686) = 0$. However, $e(1.561686) = 5.006886 < 7 = |h|$, we may conclude that this local extremum is not what we want. Then we go to the second interval $[2.75, 4.5]$. Also we see that the signs of $e'(x)$ are different on the boundary points, by using binary search technique, a new root is discovered $root = 4.044272$ (round to 6 decimals). We have also labeled it on the graph. But the corresponding value of original function $e(4.044272) = -2.004333$ where $|e(4.044272)| = 2.004333 < 7 = |h|$. This is not what we want as well. Similarly, the reader can spot a root $root = 6.005277$ in the interval $[4.5, 6.25]$ with $e(6.005277) = 3.000145 < 7 = |h|$ as well. But fortunately, in the last interval we spot a satisfying result: $root = 7.597493$ with $e(7.597493) = -7.854337$, $|e(7.597493)| = 7.854337 > 7 = |h|$. The iteration terminates here.

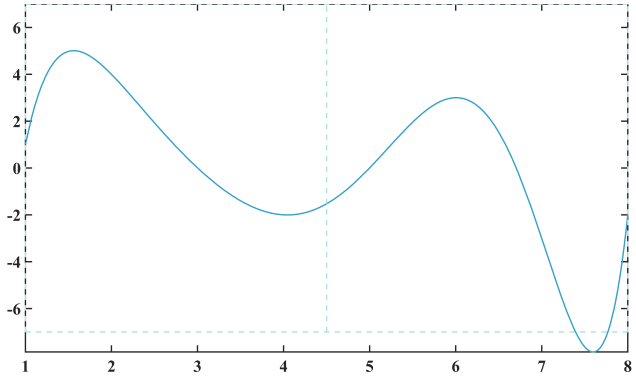
- If you believe that there are more roots, you can keep iterating, by halving all intervals one more times, results in: $[1, 1.875]$, $[1.875, 2.75]$, $[2.75, 3.625]$, $[3.625, 4.5]$,..... But since we have found a local extremum with $|e(x)| > |h|$, it is enough.



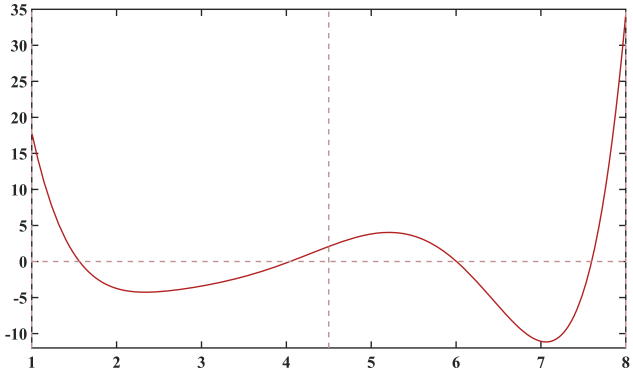
(a) e(x):Epoch1



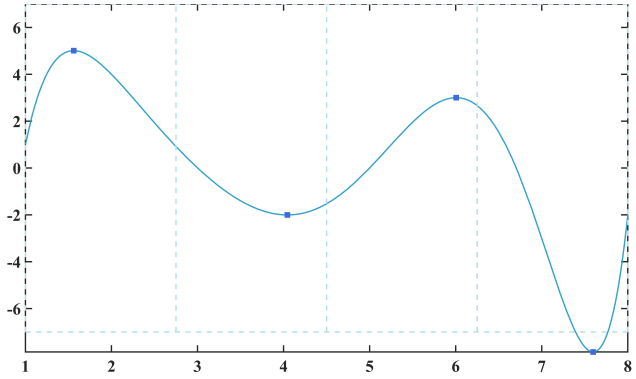
(b) e'(x):Epoch1



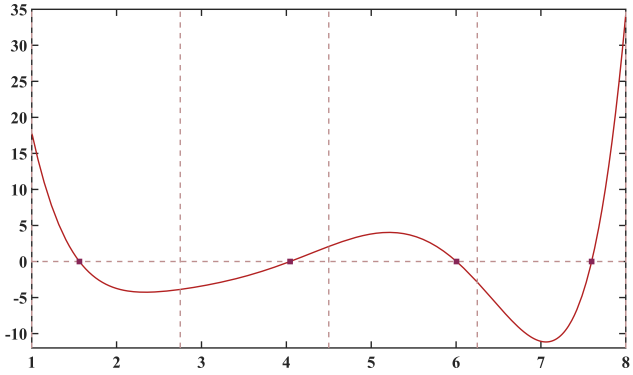
(c) e(x):Epoch2



(d) e'(x):Epoch2



(e) e(x):Epoch3



(f) e'(x):Epoch3

By this means, we can make sure that if the epochs of iteration is set to be large enough, all roots of $e'(x)$ that change sign at the root can be found. We can also make sure that all roots of $e'(x)$ does not change sign at the root will never be found(Because the binary search cannot found the root that does not change the sign). This is because when the interval is small enough, each interval contains no more than one root such that the boundary points have different signs and $e'(x)$ changes sign at this root. Our binary splitting make sure that arbitrary small intervals can be obtained. ($[0, 1]$,$[0, 0.5]$,$[0, 0.25]$,$[0, 0.125]$,...)

7.4 Change the Reference

This is roughly the same as the Discrete Exchange Algorithm. In the last section we have introduce the method of obtaining a local extremum *root* that $|e(root)| > |h|$. The continuous binary search converges dramatically fast and will be extremely useful in this section. By using the continuous binary search, we can find a local extremum *root* that $|e(root)| > |h|$. The *root* is exactly the x_{enter} we want. But since we can use function handle, we have no need to create variable 'indices'. The criteria of changing the reference is as follows:

- If $x_{enter} < \xi_0$ and the sign of $e(x_{enter})$ is the same as the sign of $e(\xi_0)$, the new reference is constructed as: $\{x_{enter}, \xi_1, \xi_2 ...\xi_{n+1}\}$
- If $x_{enter} < \xi_0$ and the sign of $e(x_{enter})$ is the same as the sign of $-e(\xi_0)$, the new reference is constructed as: $\{x_{enter}, \xi_0, \xi_1 ...\xi_n\}$
- If $x_{enter} > \xi_{n+1}$ and the sign of $e(x_{enter})$ is the same as the sign of $e(\xi_{n+1})$, the new reference is constructed as: $\{\xi_0, \xi_1 ...\xi_n, x_{enter}\}$
- If $x_{enter} > \xi_{n+1}$ and the sign of $e(x_{enter})$ is the same as the sign of $-e(\xi_{n+1})$, the new reference is constructed as: $\{\xi_1, \xi_2 ...\xi_{n+1}, x_{enter}\}$
- If $\xi_i < x_{enter} < \xi_{i+1}$ for an $i \in \{0, 1, ..., n\}$ and the sign of $e(x_{enter})$ is the same as the sign of $e(\xi_i)$, the new reference is constructed as: $\{\xi_0, \xi_1, ...\xi_{i-1}, x_{enter}, \xi_{i+1}, ..., \xi_{n+1}\}$
- If $\xi_i < x_{enter} < \xi_{i+1}$ for an $i \in \{0, 1, ..., n\}$ and the sign of $e(x_{enter})$ is the same as the sign of $e(\xi_{i+1})$, the new reference is constructed as: $\{\xi_0, \xi_1, ...\xi_i, x_{enter}, \xi_{i+2}, ..., \xi_{n+1}\}$

7.5 Termination Condition

Since we may fall into some local extremum points, we cannot directly use

$$\|f - p\|_\infty - |h| < \delta$$

Instead, we can use the following criterion: When the $|h_i| - |h_{i-1}| < \delta$, where h_i is the levelled error in Epoch i. The reason for this is that the $\|f - p\|_\infty - |h|$ decreases and $|h|$ increases after each iteration, and we know that $|h| \leqslant \|f - p\|_\infty$. We know that the the space for $|h|$ to increase will be less and less. Therefore, the increase space $|h_i| - |h_{i-1}|$ will be smaller and smaller. This can be a criterion of termination.

7.6 Handle the basic exceptions

```
if a > b
    error('Invalid interval, please choose a < b!')
end

if n < 0
    error('Invalid n!')
end

if k < 0
    error('Invalid k!')
end

if length(reference) ~= n+2
    error('Invalid size of initial reference!')
end

if threshold < 0
    error('Invalid threshold value, please choose a non-zero threshold
        value!')
end
```

7.7 Codes

```
function [p,p_table, reference_table ,h_table] = Minimax_Approximation_Continuous(f
        ,a,b,n,k,reference,threshold,epoch_max)

% Input
% f: Approximation function
% a: Left boundary point of Approximation Interval [a,b]
% b: Right boundary point of Approximation Interval [a,b]
% n: Order of Approximation polynomial
% k: The mini step value in calculating the derivative e(x)
% reference: The initial reference of iteration
% threshold: When the inf-norm Approximation Error is less than threshold, the
        program terminates
% epoch_max: When the maximum epoch is achieved, the program terminates

% Output
% p: The final approximation polynomial
% p_table: The approximation polynomial in each iteration
% reference_table: The reference in each iteration
% h_table: The levelled error h in each iteration

% Handle the basic exception
if a > b
    error('Invalid interval, please choose a < b!')
```

```
22 end
23
24 if n < 0
25     error('Invalid n!')
26 end
27
28 if k < 0
29     error('Invalid k!')
30 end
31
32 if length(reference) ~= n+2
33     error('Invalid size of initial reference!')
34 end
35
36 if threshold < 0
37     error('Invalid threshold value, please choose a non-zero threshold
           value!')
38 end
39
40 %
41 for epoch = 1:epoch_max
42     % Establish the coefficients Matrices A and beta
43     A = ones(n+2,2);
44     for i = 1:n+2
45         A(i,1) = (-1)^i;
46     end
47     for j = 3:n+2
48         for i = 1:n+2
49             A(:,j) = reference.^(j-2);
50         end
51     end
52     A_beta = [A,f(reshape(reference,length(reference),1))];
53
54     % Gaussian Elimination to solve the simultaneous equations
55     for centre = 1:n+2
56         A_beta(centre,:) = A_beta(centre,:)./A_beta(centre,centre);
57         for j = 1:size(A_beta,1)
58             if centre~=j
59                 A_beta(j,:) = A_beta(j,:) - A_beta(centre,:)*A_beta(j,centre);
60             end
61         end
62     end
63     h = A_beta(1,end);
64     p = A_beta(2:end,end);
65
66     reference_table(epoch,1:n+2) = reference;
67     h_table(epoch) = h;
68     p_table(epoch, 1:n+1) = p;
69
70     % Termination Condition: When the increase of —h— in each epoch is smaller than
           the threshold, the program terminates
```

```
71      if epoch >= 2
72          if h_table(epoch) - h_table(epoch -1) < threshold
73              break
74          end
75      end
76
77      % Define the error function and derivative of error function
78      e = @(x) f(x) - p'*x.^(0:n)';
79      e_derivative = @(x) (e(x+k)-e(x-k))/2/k;
80
81      %The extremum sometimes occurs on the boundary points
82      % The small value 0.0000001 is use to control the rounding error
83      if abs(e(a)) > abs(h) + 0.0000001 && abs(e(a)) >= abs(e(b)) && sign(e(a))
            == sign(e(reference(1)))
84          reference(1) = a;
85      elseif abs(e(a)) > abs(h) + 0.0000001 && abs(e(a)) >= abs(e(b)) && sign(e(
            a)) == -sign(e(reference(1)))
86          reference(2:end) = reference(1:end-1);
87          reference(1) = a;
88      elseif abs(e(b)) > abs(h) + 0.0000001 && abs(e(b)) > abs(e(a)) && sign(e(b)
            ) == sign(e(reference(end)))
89          reference(end) = b;
90      elseif abs(e(b)) > abs(h) + 0.0000001 && abs(e(b)) > abs(e(a)) && sign(e(b)
            ) == -sign(e(reference(end)))
91          reference(1:end-1) = reference(2:end);
92          reference(end) = b;
93
94          % Search for a local maximum or minimum of error function that larger than h
             or less than -h by continuous binary search
95      else
96          break_console = 0;
97          for sub_epoch = 0:10
98              basis = 0:1/2^sub_epoch:1;
99              s = a + basis*(b-a);
100             for i = 1:length(s)-1
101                 if sign(e_derivative(s(i))) == -sign(e_derivative(s(i+1))) && sign(
            e_derivative(s(i)))~=0 && sign(e_derivative(s(i+1)))~=0
102                     left = s(i);
103                     right = s(i+1);
104                     for searching = 1:1000
105                         if e_derivative((left+right)/2) == 0
106                             %disp('well done')
107                             x0 = (left+right)/2;
108                             break
109                         elseif sign(e_derivative((left+right)/2)) == sign(
            e_derivative(right))
110                             right = (left+right)/2;
111                         elseif sign(e_derivative((left+right)/2)) == sign(
            e_derivative(left))
112                             left = (left+right)/2;
113                         else
```

```
114                             %disp('dwduhwodq')
115                         end
116                     end
117                     x0 = (left+right)/2;
118                     if abs(e(x0)) > abs(h) + 0.0000001 %this small number is used
        to control the deviation of derivative
119                         break_console = 1;
120                         break
121                     end
122                 end
123             end
124         end
125
126         % Change the reference
127         if x0 < reference(1) && sign(e(x0)) == sign(e(reference(1)))
128             reference(1) = x0
129         elseif x0 < reference(1) && sign(e(x0)) == -sign(e(reference(1)))
130             reference(2:end) = reference(1:end-1);
131             reference(1) = x0
132         elseif x0 > reference(end) && sign(e(x0)) == sign(e(reference(end)))
133             reference(end) = x0
134         elseif x0 > reference(end) && sign(e(x0)) == -sign(e(reference(1)))
135             reference(1:end-1) = reference(2:end);
136             reference(end) = x0
137         elseif x0 > reference(1) && x0 < reference(end)
138             %To discover the location of key
139             %[1,2] means the x corresponding to key is located between references 1 and
        2
140             location = [1,2];
141             while x0 > reference(location(2))
142                 location = location + 1;
143             end
144
145             if sign(e(x0)) == (-1)^location(1)*sign(h)
146                 reference(location(1)) = x0;
147             elseif sign(e(x0)) == (-1)^location(2)*sign(h)
148                 reference(location(2)) = x0;
149             end
150             %The other cases may be one of e(x0) or sign(h) is zero, which
151             %means we have achieved a zero-error approximation
152         else
153             disp('The excellent zero error is achieved!')
154         end
155     end
156
157 end
```

8 Minimax Approximation on Continuous Function: Results, Analysis and Visualization

8.1 Case 1: A Trivial Test

We consider the following example: $[a, b] = [-1, 1]$, $f(x) = 2x^3 + x^2 + x - 1$, $p(x) = p_0 + p_1 x + p_2 x2 + p_3 x^3 + p_4 x^4$,initial reference $= \{-1, -0.9, -0.8, -0.7, -0.6, -0.5\}$

Table 4: Threshold :$\delta = 10^{-5}$

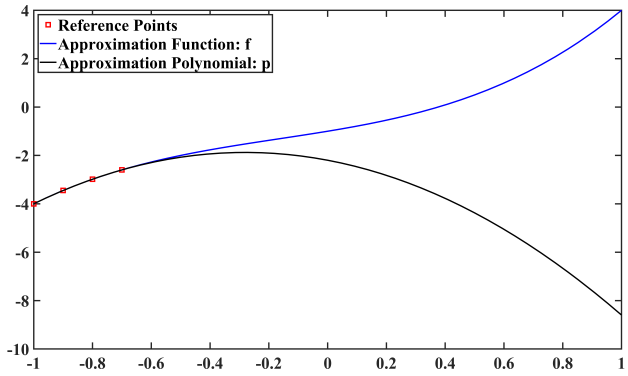
Epoch	Reference						h	Coefficient				
	ξ_0	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5		p_0	p_1	p_2	p_3	p_4
1	-1	-0.9	-0.8	-0.7	-0.6	-0.5	3.122502e-17	-1	1	1	2	0
2	-1	-0.9	-0.8	-0.7	-0.6	-0.5	3.122502e-17	-1	1	1	2	0

8.2 Case 2: Visualization of Results

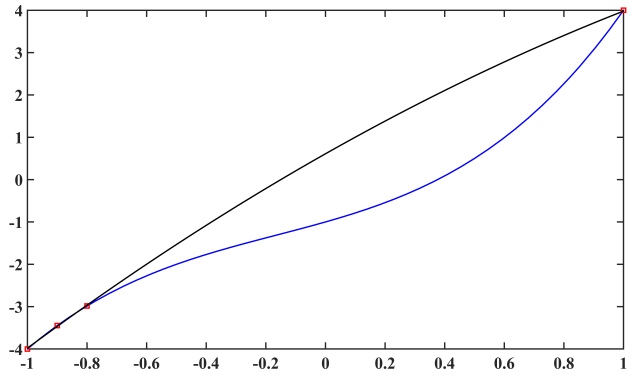
We consider the following example: $[a, b] = [-1, 1]$, $f(x) = 2x^3 + x^2 + 2x - 1$, $p(x) = p_0 + p_1 x + p_2 x^2$,initial reference $= \{-1, -0.9, -0.8, -0.7, -0.6, -0.5\}$ Notice that this function is a bit different from the case 1.

Table 5: Threshold :$\delta = 10^{-5}$

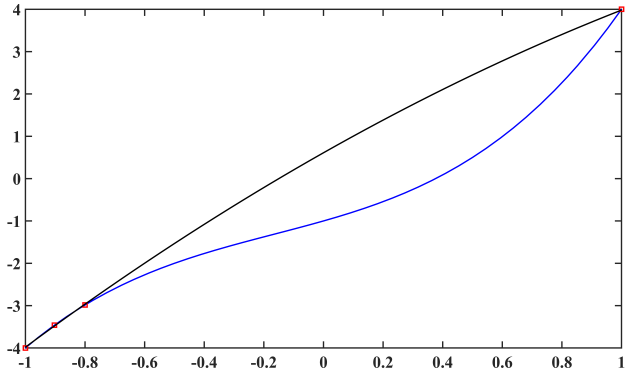
Epoch	Reference				h	Coefficient		
	ξ_0	ξ_1	ξ_2	ξ_3		p_0	p_1	p_2
1	-1	-0.9	-0.8	-0.7	0.001500	-2.198500	-2.300000	-4.100000
2	-1	-0.9	-0.8	1	0.018947	0.610526	3.981053	-0.610526
3	-1	-0.902633	-0.8	1	0.018961	0.610533	3.981039	-0.610534
4	-1	-0.902633	0.365789	1	0.214130	-1.393946	3.785870	1.393946
5	-1	-0.483848	0.365789	1	0.483548	-0.969137	3.516452	0.969137
6	-1	-0.507905	0.365789	1	0.484190	-0.968126	3.515810	0.968126
7	-1	-0.507905	0.497344	1	0.499930	-1.000111	3.500070	1.000111
8	-1	-0.499993	0.497344	1	0.499930	-0.999986	3.500007	0.999986
9	-1	-0.499993	0.499999	1	0.500000	-1.000000	3.500000	1.000000



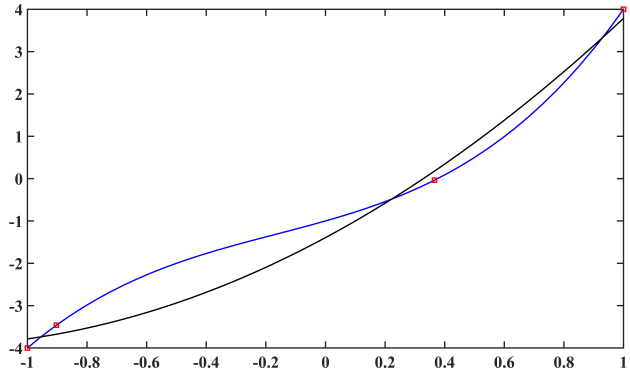
(a) Epoch 1



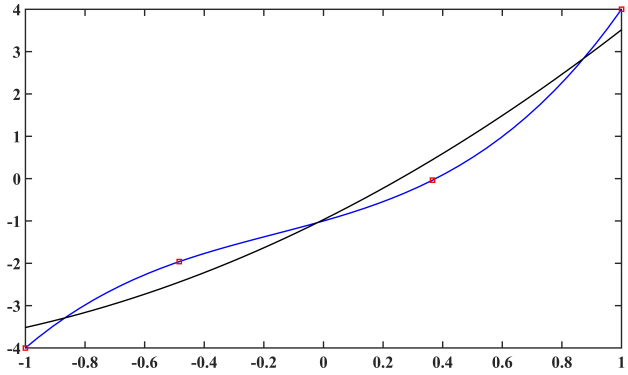
(b) Epoch 2



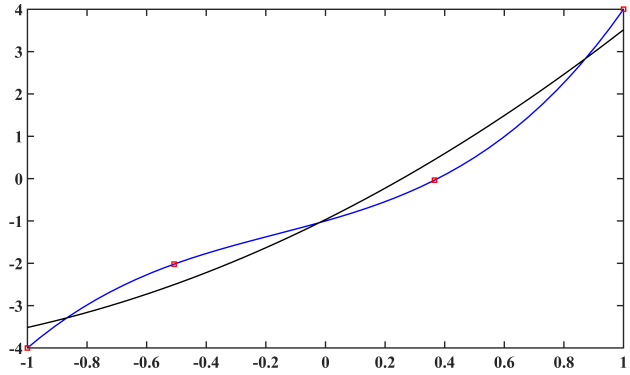
(c) Epoch 3



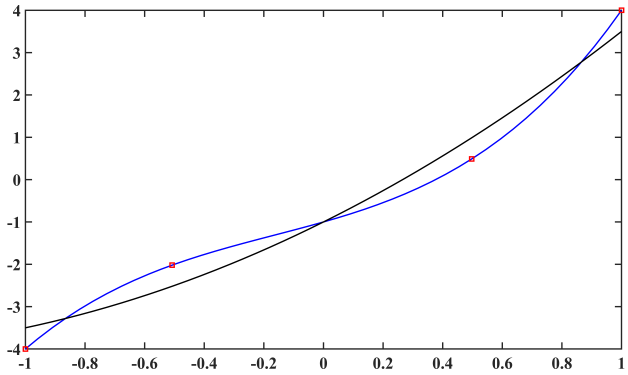
(d) Epoch 4



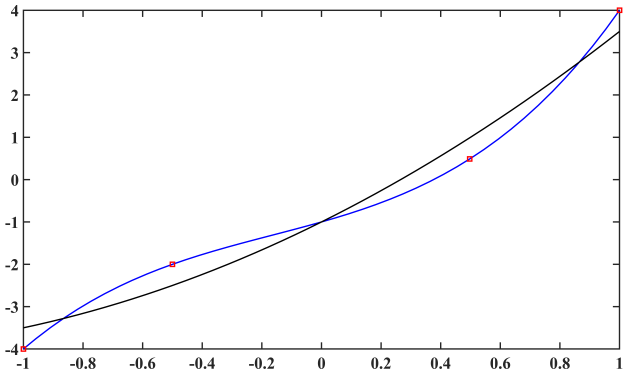
(e) Epoch 5



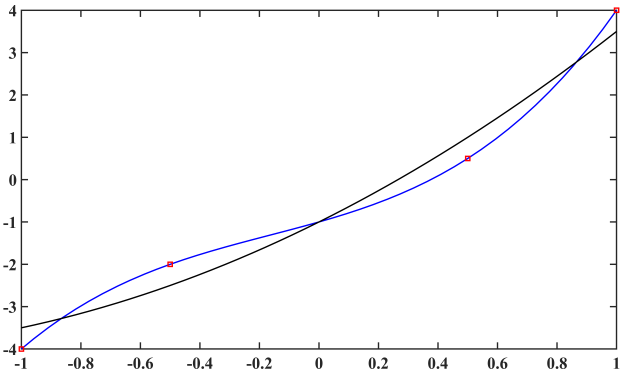
(f) Epoch 6



(a) Epoch 7



(b) Epoch 8



(c) Epoch 9

8.3 Case3: Test for Sensitivity to Initial Reference

Consider the following example: $[a, b] = [-1, 1]$, $f(x) = e^x$, $p(x) = p_0 + p_1 x + p_2 x^2$. We use 5 groups of initial references. In this example, we want to verify that the continuous exchange algorithm is also not sensitive to the initial reference. We will only provide the final results of each initial reference, rather than the whole procedure.

Table 6: Threshold :$\delta = 10^{-11}$

Initial Reference				Termination Reference				Iteration Epoch	Coefficient		
ξ_0	ξ_1	ξ_2	ξ_3	ξ_0	ξ_1	ξ_2	ξ_3		p_0	p_1	p_2
-1	-0.5	0.5	1	-1	-0.43695788	0.56005776	1	5	0.98914108	1.13086433	0.55363656
-1	0	0.5	1	-1	-0.43695762	0.56005776	1	6	0.98903973	1.13018381	0.55404091
-0.8	-0.3	0.4	0.7	-1	-0.43693924	0.56005776	1	8	0.98903973	1.13018381	0.55404091
-1	-0.9	-0.8	-0.7	-1	-0.43695063	0.56005776	1	9	0.98903973	1.13018381	0.55404091
0.7	0.8	0.9	1	-1	-0.43695797	0.56005776	1	12	0.98903973	1.13018381	0.55404091

8.4 Analysis

From Case 1 and Case 2 we can verify that the algorithm works very well and converges swiftly. (You can check the correctness of this program in the screenshots from other links and textbook). In Case 3 we also see that the exchange algorithm on a continuous function is insensitive to the initial chosen reference. They will finally converges to almost the same reference points.

9 2-norm Approximation on Continuous Function: Procedure and Codes

To completes the 2-norm approximation, we just need to follow the main idea of Chapter 11 of textbook: (11.47),(11.48), and Theorem 11.3. But we need to modify the computational procedure.

9.1 Numerical Weighted Inner Product

Since we could not use built-in function of Matlab, we need to approximate the Integration by numerical Riemann Integral. However, this makes our program slow. The modified Riemann integral is defined as follows:

$$I(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{n-1} (x_{i+1} - x_i)(\frac{f(x_{i+1}) + f(x_i)}{2})$$

The numerical inner product makes use of the numerical Riemann Integral. By (11.2) in Chapter 11:

$$(f, g) = \int_a^b w(x) f(x) g(x) \, \mathrm{d}x$$

Define the weighted function to be $w(x) = 1$, which is exactly 2-norm inner product. More generally, we can use arbitrary weighted function.

9.2 Codes: Numerical weighted Inner Product

```
function values = Numerical_Inner_Product(f1,f2,w,a,b, integral_division )

% Input
% f1: The first function of inner product ¡f1,f2¿
% f2: The second function of inner product ¡f1,f2¿
% w: weight function
% a: Left boundary point of integral interval [a,b]
% b: Right boundary point of integral interval [a,b]
% integral_division: In Numerical Integration, the number of division of sub-intervals

% Output
% values: The weighted inner product

% Integral and Inner Product by Riemann Integral
values = 0;
g = @(x) w(x)*f1(x)*f2(x);
for i = 1: integral_division
    values = values + (b-a)/integral_division/2*(g(a+(b-a)*i/integral_division)+g(a
        +(b-a)/integral_division*(i-1)));
end
```

9.3 Handle the basic exceptions

```
if n < 0
    error('Invalid n!')
end

if threshold < 0
    error('Invalid threshold value, please choose a non-zero threshold
        value!')
end

if a > b
    error('Invalid interval, please choose a < b!')
end

```

```
if  integral_division  < 10000
    warning('The integral division is not dense enough, please choose an
        integral_division larger than 10000!')
end
```

9.4 Procedure of the Algorithm

$$\phi_0(x) = 1, \quad a \le x \le b$$

$$\alpha_j = (\phi_j, x\phi_j) / \|\phi_j\|^2$$

$$\phi_1(x) = (x - \alpha_0)\phi_0(x), \quad a \le x \le b$$

$$\beta_j = \|\phi_j\|^2 / \|\phi_{j-1}\|^2$$

$$\phi_{j+1}(x) = (x - \alpha_j)\phi_j(x) - \beta_j\phi_{j-1}(x), \quad a \le x \le b$$

$$c_j^* = (\phi_j, f) / \|\phi_j\|^2$$

$$p^*(x) = \sum_{j=0}^{n} c_j^*\phi_j(x)$$

9.5 Codes

```
function [p,polynomial_table,phi_polynomial,alpha,beta] = Approximation_2_norm(f,
        weight,a,b,n,threshold,integral_division)

% Input
% f: Approximation function
% weight : The weight function of inner product, if you want to use 2-norm
        approximation, use weight = @(x) 1
% a: Left boundary point of Approximation Interval [a,b]
% b: Right boundary point of Approximation Interval [a,b]
% n: Highest Order of Approximation Polynomial
% threshold: When the 2-norm Approximation Error is less than threshold, the program
        terminates
% integral_division: In Numerical Integration, the number of division of sub-intervals

% Output
% p: The final approximation polynomial
% polynomial_table: The approximation polynomial in each epoch
% phi_polynomial: The phi_polynomial in each epoch
% alpha: The alpha in each epoch
```

```
% beta: The beta in each epoch

% Handle the basic exception
if n < 0
    error('Invalid n!')
end

if threshold < 0
    error('Invalid threshold value, please choose a non-zero threshold
        value!')
end

if a > b
    error('Invalid interval, please choose a < b!')
end

if  integral_division  < 10000
    warning('The integral division is not dense enough, please choose an
        integral_division larger than 10000!')
end

% If n == 0, the polynomial is order 0. Remark: phi1 in the Matlab records phi_0, as we
        do not have 0-index in Matlab
%If n == 0, only this part of code will be executed
if n >= 0
    %This is used to control the termination condition
    break_console = 0;
    phi{1} = @(x) 1;
    phi_polynomial(1) = 1;
    phi_norm_square(1) = Numerical_Inner_Product(phi{1},phi{1},weight,a,b,
        integral_division);
    x_multiply_phi{1} = @(x) phi{1}(x)*x;
    alpha(1) = Numerical_Inner_Product(x_multiply_phi{1},phi{1},weight,a,b,
        integral_division)/phi_norm_square(1);
    c(1) = Numerical_Inner_Product(phi{1},weight,f,a,b,integral_division )/
        phi_norm_square(1);
    p(1,1:1) = c(1)*phi_polynomial(1,:);
    polynomial_table(1,1:1) = p;
    beta(1) = NaN;

    if abs(Numerical_Inner_Product(@(x) f(x) - c(1) * phi{1}(x),@(x) f(x) - c(1) *
        phi{1}(x),weight,a,b, integral_division )) < threshold
        Numerical_Inner_Product(@(x) f(x) - c(1) * phi{1}(x),@(x) f(x) - c(1) * phi
        {1}(x),weight,a,b, integral_division )
        break_console = 1;
        disp('Terminates! We have reached the threshold!')
    end

end

% If n == 1, the polynomial is order 1
%If n == 1, only the codes before and this part will be executed
```

```
61  if n >= 1 && break_console ~= 1
62      phi_polynomial(2,1:2) = 0;
63      phi_polynomial(2,1:1) = phi_polynomial(2,1:1) - alpha(1) * phi_polynomial(1,1:1);
64      phi_polynomial(2,2:2) = phi_polynomial(2,2:2) + phi_polynomial(1,1:1);
65      phi{2} = @(x) phi_polynomial(2,1:2) * x.^(0:1)';
66      phi_norm_square(2)= Numerical_Inner_Product(phi{2},phi{2},weight,a,b,
            integral_division);
67      x_multiply_phi{2} = @(x) phi{2}(x)*x;
68      alpha(2) = Numerical_Inner_Product(x_multiply_phi{2},phi{2},weight,a,b,
            integral_division)/phi_norm_square(2);
69      beta(2) = phi_norm_square(2)/phi_norm_square(1);
70      c(2) = Numerical_Inner_Product(phi{2},f,weight,a,b, integral_division )/
            phi_norm_square(2);
71      p(1,1:2) = [p,0] + c(2) * phi_polynomial(2,:);
72      polynomial_table(2,1:2) = p;
73
74      if abs(Numerical_Inner_Product(@(x) f(x) - p(1:2)*x.^(0:1)',@(x) f(x) - p(1:2)*x
            .^(0:1)', weight,a,b, integral_division )) < threshold
75          break_console = 1;
76          disp('Terminates! We have reached the threshold!')
77      end
78
79  end
80
81
82  % If n ¿= 2, all the codes will be executed
83  if n >= 2 && break_console ~= 1
84      for j = 3:n + 1
85          phi_polynomial(j,1:j) = 0;
86          phi_polynomial(j,2:j) = phi_polynomial(j-1,1:j-1);
87          phi_polynomial(j,1:j) = phi_polynomial(j,1:j) - phi_polynomial(j-1,1:j) *
              alpha(j-1);
88          phi_polynomial(j,1:j) = phi_polynomial(j,1:j) -beta(j-1) * phi_polynomial(j
              -2,:);
89          phi{j} = @(x) phi_polynomial(j,1:j) * x.^(0:j-1)';
90
91          phi_norm_square(j)= Numerical_Inner_Product(phi{j},phi{j},weight,a,b,
              integral_division);
92          x_multiply_phi{j} = @(x) phi{j}(x)*x;
93          alpha(j) = Numerical_Inner_Product(x_multiply_phi{j},phi{j},weight,a,b,
              integral_division )/phi_norm_square(j);
94          beta(j) = phi_norm_square(j)/phi_norm_square(j-1);
95          c(j) = Numerical_Inner_Product(phi{j},f,weight,a,b, integral_division )/
              phi_norm_square(j);
96          p(1,1:j) = [p,0] + c(j)*phi_polynomial(j,:);
97
98          polynomial_table(j,1:j) = p;
99
100         if abs(Numerical_Inner_Product(@(x) f(x) - p(1:j)*x.^(0:j-1)',@(x) f(x) - p
              (1:j)*x.^(0:j-1)',weight,a,b, integral_division )) < threshold
101             disp('Terminates! We have reached the threshold!')
```

```
            break
        end

    end
end
```

10 2-norm Approximation on on Continuous Function: Results, Analysis and Visualization

10.1 Case1: A normal test on 2-norm

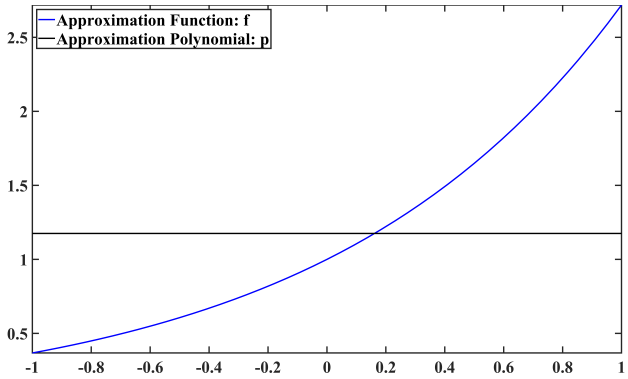
We will use the following example: $[a,b] = [0,1], f(x) = \sqrt{1+x^2}, w(x) = 1, p(x) = p_0 + p_1 x$

Epoch	i	$\phi_j = \sum_{j=0}^{i} p_j \phi_j$		α	β	Coefficient	
		p_0	p_1			p_0	p_1
1	0	1	\	0.5	\	1.147794	\
2	1	-0.5	1	0.5	0.0833	0.934320	0.426947

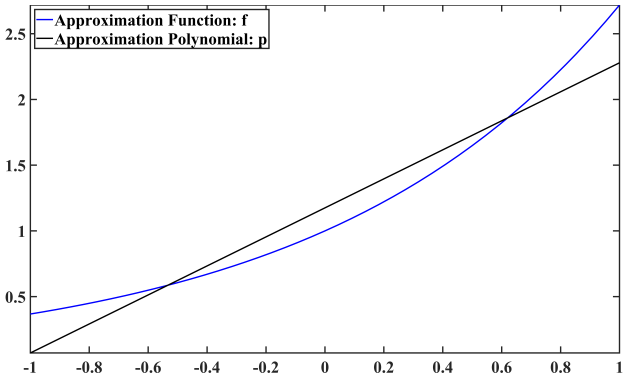
10.2 Case2 : Visualization of Results, 2-norm

We will use the following example: $[a,b] = [-1,1], f(x) = e^x, w(x) = 1, p(x) = p_0 + p_1 x + p_2 x^2 + p_3 x^3$

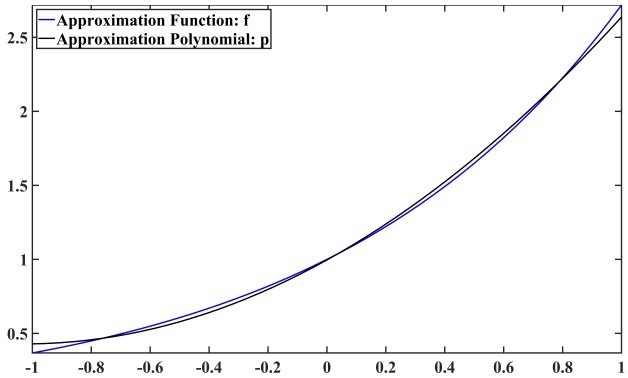
Epoch	i	$\phi_j = \sum_{j=0}^{i} p_j \phi_j$				α	β	Coefficient			
		p_0	p_1	p_2	p_3			p_0	p_1	p_2	p_3
1	0	1	\	\	\	0	\	1.175201	\	\	\
2	1	0	1	\	\	0	0.3333	1.175201	1.103638	\	\
3	2	-0.3333	0	1	\	0	0.2667	0.996294	1.103638	0.536722	\
4	3	0	-0.6000	0	1	0	0.2571	0.996294	0.997955	0.536722	0.176139



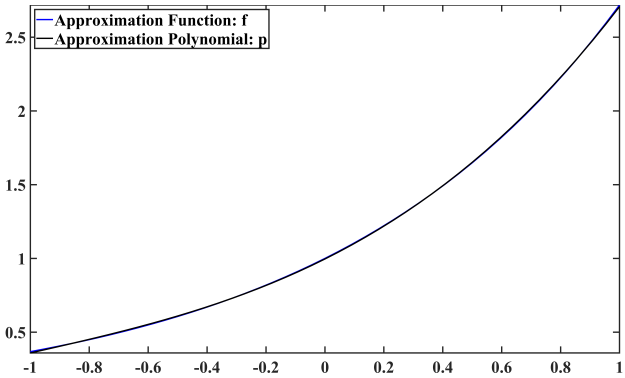
(a) Epoch 1



(b) Epoch 2



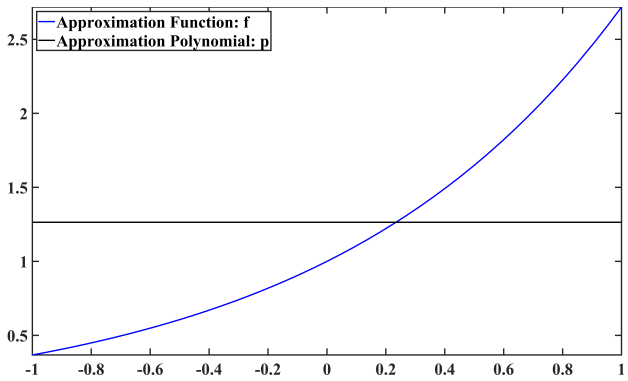
(c) Epoch 3



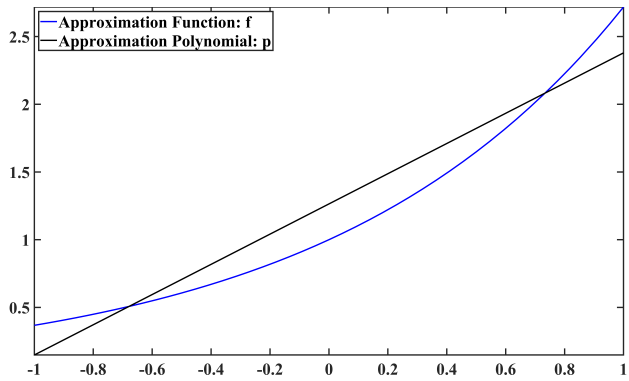
(d) Epoch 4

10.3 Case 3: A more general norm, induced by weighted inner product

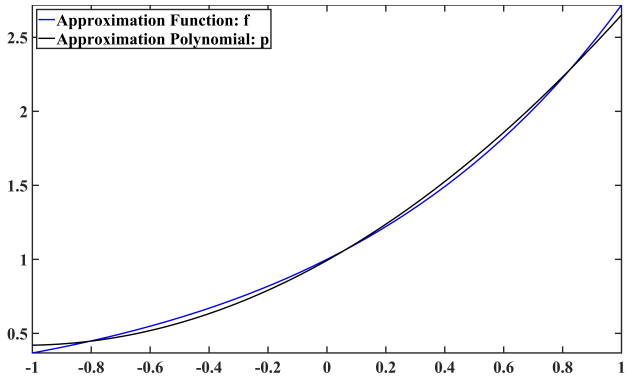
We will use the following example: $[a, b] = [-1, 1], f(x) = e^x, w(x) = |x|, p(x) = p_0 + p_1 x + p_2 x^2 + p_3 x^3$



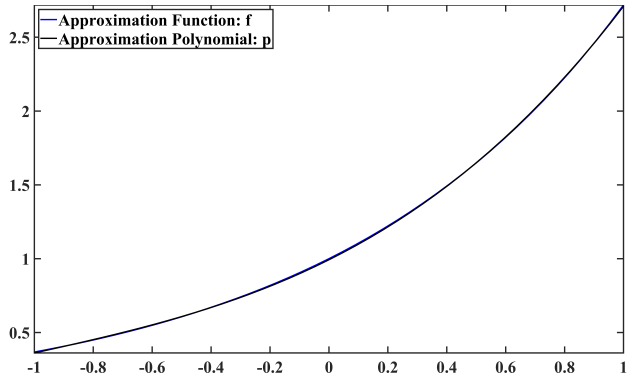
(a) Epoch 1



(b) Epoch 2



(c) Epoch 3



(d) Epoch 4

Epoch	i	$\phi_j = \sum_{j=0}^{i} p_j \phi_j$				α	β	Coefficient			
		p_0	p_1	p_2	p_3			p_0	p_1	p_2	p_3
1	0	1	\	\	\	0	\	1.264241	\	\	\
2	1	0	1	\	\	0	0.5000	1.264241	1.115358	\	\
3	2	-0.5	0	1.0000	\	0	0.1667	0.992773	1.115358	0.542937	\
4	3	0	-0.6667	0	1.0000	0	0.3333	0.992773	0.997419	0.542937	0.176908

10.4 Analysis

As we can see from the examples, 2-norm approximation achieve extremely precise results. But this program suffers from the precision-time consumption trade-off. This program is very slow, because of the numerical inner product. If you decrease the division number of the integration interval, the precision is decreased as well, but the running speed is increased. Conversely, when the division number of the integration interval is increased, the precision will increase, while the running speed is decreased. Compare the case 2 and case3, we have the idea that the weight function will influence the approximation to a large extent. Since the weighted function of case 3 focus much more on the large value of x, but the case 2 focus averagely on the whole interval.