现代数学概论【科学计算】

Lecture 3 - Approximation & Interpolation

Xian-Liang Hu

School of Mathematical Sciences, Zhejiang University, CHINA.

http://www.mathweb.zju.edu.cn:8080/xlhu/sc.html

Main Contents of Lecture 3

Chapter 5. Equation Roots

Chapter 7. Approximation & Interpolation

Chapter 5. Equation Roots

Chapter 7. Approximation & Interpolation

Nonlinear Equations

For given nonlinear equation $f(x) = y$, let us define its root problem as

Definition (Roots Problem)

To calculate x, such that $f(x) = y$ holds.

Generally, it is simplified by substracting y at the both sides, it yields

$$f(x) = 0. \tag{1}$$

Nonlinear Equations

For given nonlinear equation $f(x) = y$, let us define its root problem as

Definition (Roots Problem)

To calculate x, such that $f(x) = y$ holds.

Generally, it is simplified by substracting y at the both sides, it yields

$$f(x) = 0. \tag{1}$$

Existence & Uniqueness

Example (One dimension)

- $e^x + 1 = 0$, no solution
- $e^x - x = 0$ has only one solution
- $x^2 - 4\sin x = 0$ has two solutions
- $x^3 + 6x^2 + 11x - 6 = 0$ has three solutions
- $\sin x = 0$ has infinite many solutions

Existence & Uniqueness

Example (One dimension)

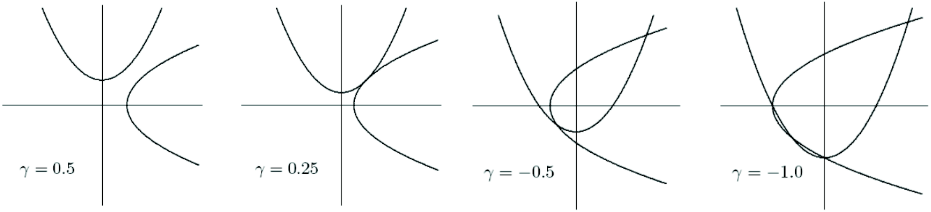
- $e^x + 1 = 0$, no solution
- $e^x - x = 0$ has only one solution
- $x^2 - 4\sin x = 0$ has two solutions
- $x^3 + 6x^2 + 11x - 6 = 0$ has three solutions
- $\sin x = 0$ has infinite many solutions

Example (Two dimension)

An example of a system of nonlinear equation in two dimension is

$$f(x) = \begin{bmatrix} x_1^2 - x_2 + 0.25 \\ -x_1 + x_2^2 + 0.25 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

where $x_1 = x_2 = 0.5$ is one approximated solution(root).

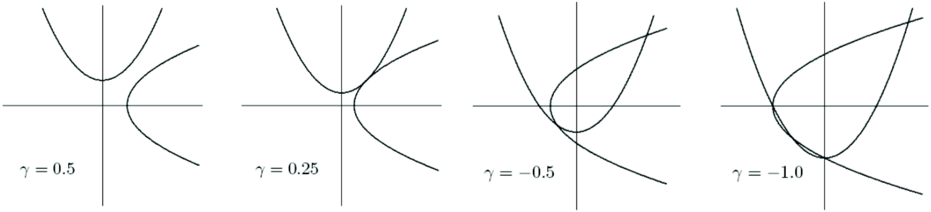


Example (Two dimension)

An example of a system of nonlinear equation in two dimension is

$$f(x) = \begin{bmatrix} x_1^2 - x_2 + \gamma \\ -x_1 + x_2^2 + \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

where $x_1 = x_2 = 0.5$ is one approximated solution(root).



Local criteria for the existence

1. 介值定理若f是闭区间上[a,b]的连续函数，c介于f(a)和f(b)，之间，则必存在一个$x^1 \in [a, b]$，满足$f(x^1) = c$，取c为0即可证明f在[a,b]上必有根
2. 反函数定理$x = f^{-1}(0)$
3. 压缩映射理论（也是迭代法收敛性的基本定理）
4. topological degree of function f与重根理论

Local criteria for the existence

1. 介值定理若f是闭区间上[a,b]的连续函数，c介于f(a)和f(b)，之间，则必存在一个$x^1 \in [a, b]$，满足$f(x^1) = c$，取c为0即可证明f在[a,b]上必有根
2. 反函数定理$x = f^{-1}(0)$
3. 压缩映射理论（也是迭代法收敛性的基本定理）
4. topological degree of function f与重根理论

Local criteria for the existence

1. 介值定理若f是闭区间上[a,b]的连续函数，c介于f(a)和f(b)，之间，则必存在一个$x^1 \in [a, b]$，满足$f(x^1) = c$，取c为0即可证明f在[a,b]上必有根
2. 反函数定理$x = f^{-1}(0)$
3. 压缩映射理论（也是迭代法收敛性的基本定理）
4. topological degree of function f与重根理论

Local criteria for the existence

1. 介值定理若f是闭区间上[a,b]的连续函数，c介于f(a)和f(b)，之间，则必存在一个$x^1 \in [a,b]$，满足$f(x^1) = c$，取c为0即可证明f在[a,b]上必有根
2. 反函数定理$x = f^{-1}(0)$
3. 压缩映射理论（也是迭代法收敛性的基本定理）
4. topological degree of function f与重根理论

Fixed Point theorem

Definition (Fixed Point)

$g:\mathbb{R}^n \to \mathbb{R}^n$是集合$S \in \mathbb{R}^n$上的压缩映射，即存在$\gamma \in [0,1]$,对任意集合S内的两个点x，z满足

$$\|g(x) - g(z)\| \leq \gamma \|x - z\|$$

满足$g(x) = x$ 的x 称为g的不动点。

Theorem (Fixed Point theorem)

若g在闭集S上是压缩映射，且$g(S) \in S$, 则g在S内存在唯一的不动点。此时如果非线性方程有形如$f = x - g$，则我们可称，$f = 0$在闭集S上有唯一解

Fixed Point theorem

Definition (Fixed Point)

g:$\mathbb{R}^n \to \mathbb{R}^n$是集合$S \in \mathbb{R}^n$上的压缩映射，即存在$\gamma \in [0,1]$,对任意集合S内的两个点x，z满足

$$\|g(x) - g(z)\| \leq \gamma \|x - z\|$$

满足$g(x) = x$ 的x 称为g的不动点。

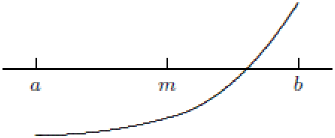
Theorem (Fixed Point theorem)

若g在闭集S上是压缩映射，且$g(S) \in S$, 则g在S内存在唯一的不动点。此时如果非线性方程有形如$f = x - g$，则我们可称，$f = 0$在闭集S上有唯一解

Numerical methods for single nonlinear equation

1. Bisection
2. Fixed-Point Iteration
3. Newton Method
4. Secant Method

1. Bisection




```
while((b-a)>tol) do
    m = a+(b-a)/2
    if sign(f(a)) = sign(f(m))
      a= m
    else
      b=m
    end
  end
```

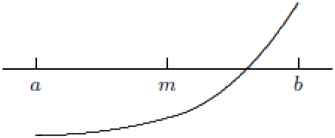
Example (Bisection)

To seek a root on interval $[1, 3]$ for

$$f(x) = x^2 - 4\sin(x) = 0.$$

Firstly, evaluate function f at the midpoint $m = 0.5(a + b) = 2.0$ and it is clear that $f(2.0)$ and $f(1.0)$ have different sign. Then we keep the left half of the interval, that is $b = m = 2.0$. Next, find a root in $[1.0, 2.0]$. We illustrate the iteration in the following table

1. Bisection




```
while((b-a)>tol) do
   m = a+(b-a)/2
   if sign(f(a)) = sign(f(m))
     a= m
   else
     b=m
   end
end
```

Example (Bisection)

To seek a root on interval $[1,3]$ for

$$f(x) = x^2 - 4\sin(x) = 0.$$

Firstly, evaluate function f at the midpoint $m = 0.5(a+b) = 2.0$ and it is clear that $f(2.0)$ and $f(1.0)$ have different sign. Then we keep the left half of the interval, that is $b = m = 2.0$. Next, find a root in $[1.0, 2.0]$. We illustrate the iteration in the following table

Bisection - Results

To seek a root on interval $[1, 3]$ for

$$f(x) = x^2 - 4\sin(x) = 0.$$

a	$f(a)$	b	$f(b)$
1.000000	−2.365884	3.000000	8.435520
1.000000	−2.365884	2.000000	0.362810
1.500000	−1.739980	2.000000	0.362810
1.750000	−0.873444	2.000000	0.362810
1.875000	−0.300718	2.000000	0.362810
1.875000	−0.300718	1.937500	0.019849
1.906250	−0.143255	1.937500	0.019849
1.921875	−0.062406	1.937500	0.019849
1.929688	−0.021454	1.937500	0.019849
1.933594	−0.000846	1.937500	0.019849
1.933594	−0.000846	1.935547	0.009491
1.933594	−0.000846	1.934570	0.004320
1.933594	−0.000846	1.934082	0.001736
1.933594	−0.000846	1.933838	0.000445
1.933716	−0.000201	1.933838	0.000445
1.933716	−0.000201	1.933777	0.000122
1.933746	−0.000039	1.933777	0.000122
1.933746	−0.000039	1.933762	0.000041
1.933746	−0.000039	1.933754	0.000001
1.933750	−0.000019	1.933754	0.000001
1.933752	−0.000009	1.933754	0.000001
1.933753	−0.000004	1.933754	0.000001

2.Fixed-Point Iteration

To find the fixed point of given function $g : \mathbb{R} \to \mathbb{R}$, such that

$$x = g(x).$$

It is equivalent to do

$$x_{k+1} = g(x_k)$$

repeatedly until convergence, then the fixed points for g are solutions of nonlinear equaiton $f(x) := x - g(x) = 0$.

- Also called functional iteration, since function g is applied repeatedly to initial starting value x_0
- For given equation $f(x) = 0$, there may be many equivalent fixed-point problems $x = g(x)$ with different choices for g

2.Fixed-Point Iteration

To find the fixed point of given function $g : \mathbb{R} \to \mathbb{R}$, such that

$$x = g(x).$$

It is equivalent to do

$$x_{k+1} = g(x_k)$$

repeatedly until convergence, then the fixed points for g are solutions of nonlinear equaiton $f(x) := x - g(x) = 0$.

- Also called functional iteration, since function g is applied repeatedly to initial starting value x_0
- For given equation $f(x) = 0$, there may be many equivalent fixed-point problems $x = g(x)$ with different choices for g

2.Fixed-Point Iteration

To find the fixed point of given function $g : \mathbb{R} \to \mathbb{R}$, such that

$$x = g(x).$$

It is equivalent to do

$$x_{k+1} = g(x_k)$$

repeatedly until convergence, then the fixed points for g are solutions of nonlinear equaiton $f(x) := x - g(x) = 0$.

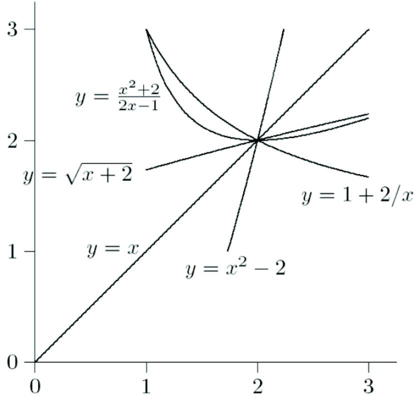
- Also called functional iteration, since function g is applied repeatedly to initial starting value x_0
- For given equation $f(x) = 0$, there may be many equivalent fixed-point problems $x = g(x)$ with different choices for g

Example (Fixed-Point Iteration)

Different Scheme for seeking roots($x_1^* = -1, x_2^* = 2$) for $f(x) = x^2 - x - 2 = 0$.

1. $g(x) = x^2 - 2$;
2. $g(x) = \sqrt{x+2}$;
3. $g(x) = 1 + 2/x$;
4. $g(x) = (x^2+2)/(2x-1)$;

- Not Unique! !
- May result in different roots.
- Different convergence rate!

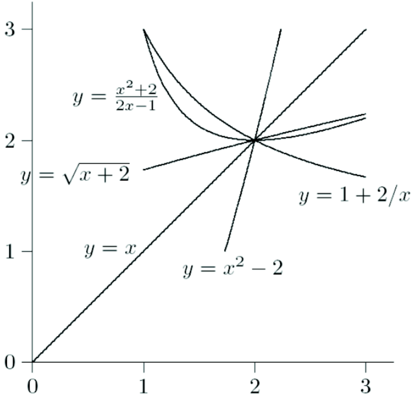


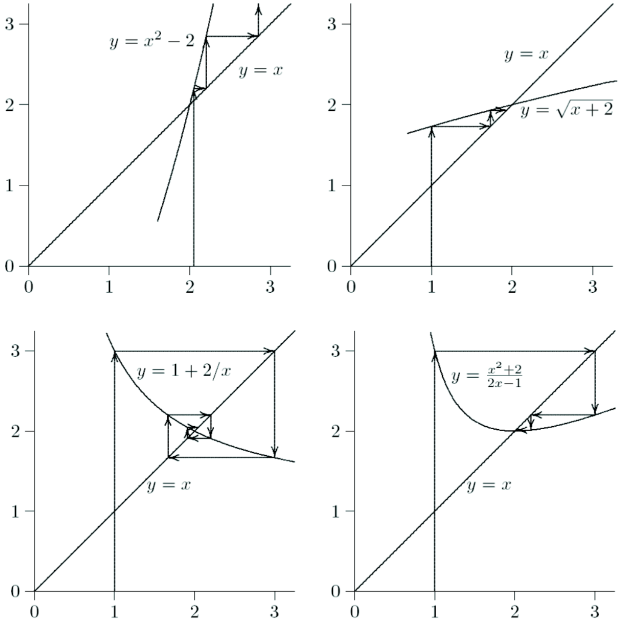
Example (Fixed-Point Iteration)

Different Scheme for seeking roots($x_1^* = -1, x_2^* = 2$) for $f(x) = x^2 - x - 2 = 0$.

1. $g(x) = x^2 - 2$;
2. $g(x) = \sqrt{x+2}$;
3. $g(x) = 1 + 2/x$;
4. $g(x) = (x^2 + 2)/(2x - 1)$;

- Not Unique！！
- May result in different roots.
- Different convergence rate!





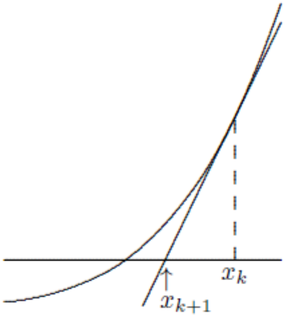
$y = x^2 - 2$
$y = x$
$y = x$
$y = \sqrt{x+2}$
$y = 1 + 2/x$
$y = x$
$y = \frac{x^2+2}{2x-1}$
$y = x$

3. Newton's Method

Fundamental idea:

由于零点是其切线与x的交点，非零点则不是，于是在点x_k附近，使用$f(x_k)$处的切线来近似，使用这个切线的零点作为新的近似值，以此来靠近真正的零点。

$$f(x^* + \Delta x) \approx f(x^*) + f'(x^*)\Delta x$$



Initialize x_0

for k = 0,1,2,...

$x_{k+1} = x_k - f(x_k)/f'(x_k)$

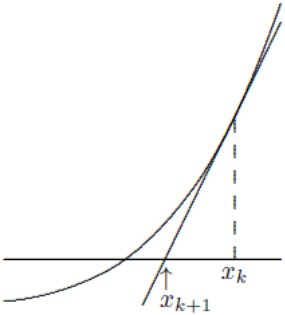
end

3. Newton's Method

Fundamental idea：

由于零点是其切线与x的交点，非零点则不是，于是在点x_k附近，使用$f(x_k)$处的切线来近似，使用这个切线的零点作为新的近似值，以此来靠近真正的零点。

$$f(x^* + \Delta x) \approx f(x^*) + f'(x^*)\Delta x$$



Initialize x_0

for k = 0,1,2,...

$x_{k+1} = x_k - f(x_k)/f'(x_k)$

end

Example (Newton's method)

Find a root with Newton's method for the equation

$$f(x) = x^2 - 4\sin(x) = 0.$$

Solution: the derivative of the function is

$$f'(x) = 2x - 4\cos(x)$$

Then the iterative scheme of Newton's methods is

$$x_{k+1} = x_k - \frac{x_k^2 - 4\sin(x_k)}{2x_k - 4\cos(x_k)}.$$

k	x_k	$f(x_k)$	$f'(x_k)$	h_k
0	3.000000	8.435520	9.959970	−0.846942
1	2.153058	1.294772	6.505771	−0.199019
2	1.954039	0.108438	5.403795	−0.020067
3	1.933972	0.001152	5.288919	−0.000218
4	1.933754	0.000000	5.287670	0.000000

Example (Newton's method)

Find a root with Newton's method for the equation

$$f(x) = x^2 - 4\sin(x) = 0.$$

Solution: the derivative of the function is

$$f'(x) = 2x - 4\cos(x)$$

Then the iterative scheme of Newton's methods is

$$x_{k+1} = x_k - \frac{x_k^2 - 4\sin(x_k)}{2x_k - 4\cos(x_k)}.$$

k	x_k	$f(x_k)$	$f'(x_k)$	h_k
0	3.000000	8.435520	9.959970	−0.846942
1	2.153058	1.294772	6.505771	−0.199019
2	1.954039	0.108438	5.403795	−0.020067
3	1.933972	0.001152	5.288919	−0.000218
4	1.933754	0.000000	5.287670	0.000000

4. Secant Method

牛顿法有一个计算上的缺陷，即在每次迭代时都要计算函数及其导数的值，导数在计算中往往不方便或者计算量很大，因此在步长较小的情况下，我们可以用有限差分来近似代替导数，即用相邻两次迭代的函数值来代替导数。这种方法叫割线法:

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

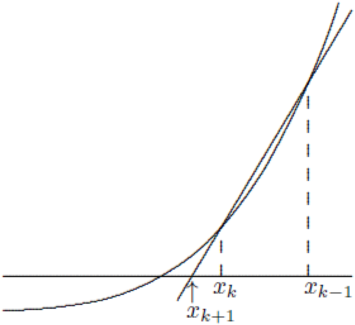
其算法流程可以表示为:

given initial value x_0

for k = 0,1,2,...

$x_{k+1} = x_k - f(x_k)(x_k - x_{k-1}/[f(x_k) - f(x_{k-1})]$

end



4. Secant Method

牛顿法有一个计算上的缺陷，即在每次迭代时都要计算函数及其导数的值，导数在计算中往往不方便或者计算量很大，因此在步长较小的情况下，我们可以用有限差分来近似代替导数，即用相邻两次迭代的函数值来代替导数。这种方法叫割线法:

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

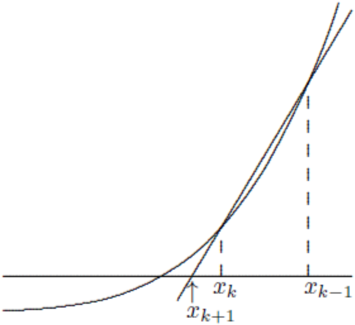
其算法流程可以表示为:

given initial value x_0

for k = 0,1,2,...

$\quad x_{k+1} = x_k - f(x_k)(x_k - x_{k-1})/[f(x_k) - f(x_{k-1})]$

end



Example (Secant method)

Find a root with secant method for the equation

$$f(x) = x^2 - 4\sin(x) = 0.$$

Then the iterative scheme with $x_0 = 1$ and $x_1 = 3$:

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k).$$

We find that it convergence more slowly:

k	x_k	$f(x_k)$	h_k
0	1.000000	−2.365884	
1	3.000000	8.435520	−1.561930
2	1.438070	−1.896774	0.286735
3	1.724805	−0.977706	0.305029
4	2.029833	0.534305	−0.107789
5	1.922044	−0.061523	0.011130
6	1.933174	−0.003064	0.000583
7	1.933757	0.000019	−0.000004
8	1.933754	0.000000	0.000000

Question: How to find **ALL** roots of polynomial functions

For the special case of a polynomial $p(x)$ with degree n, it is complex to find all n of its zeros, even if the coefficients are all real. There are several practical methods

- Firstly, find one single root x_1 with any numerical method, i.e., Newton's method, and then continue to find another root via solving $p(x)/(x - x_1) = 0$. Do it repeatedly!
- Build the *Companion Matrix* of the given polynomial $p(x)$, then the eigenvalues of companion matrix are happen to be the roots of $p(x)$. (This is the method used by the *roots* function in matlab)
- Any other specially designed method, such as the method of Laguerre, of Bairstow, and of Traub, etc.

Question: How to find **ALL** roots of polynomial functions

For the special case of a polynomial $p(x)$ with degree n, it is complex to find all n of its zeros, even if the coefficients are all real. There are several practical methods

- Firstly, find one single root x_1 with any numerical method, i.e., Newton's method, and then continue to find another root via solving $p(x)/(x - x_1) = 0$. Do it repeatedly!
- Build the *Companion Matrix* of the given polynomial $p(x)$, then the eigenvalues of companion matrix are happen to be the roots of $p(x)$. (This is the method used by the *roots* function in matlab)
- Any other specially designed method, such as the method of Laguerre, of Bairstow, and of Traub, etc.

Question: How to find **ALL** roots of polynomial functions

For the special case of a polynomial $p(x)$ with degree n, it is complex to find all n of its zeros, even if the coefficients are all real. There are several practical methods

- Firstly, find one single root x_1 with any numerical method, i.e., Newton's method, and then continue to find another root via solving $p(x)/(x - x_1) = 0$. Do it repeatedly!
- Build the *Companion Matrix* of the given polynomial $p(x)$, then the eigenvalues of companion matrix are happen to be the roots of $p(x)$. (This is the method used by the *roots* function in matlab)
- Any other specially designed method, such as the method of Laguerre, of Bairstow, and of Traub, etc.

Question: How to find **ALL** roots of polynomial functions

For the special case of a polynomial $p(x)$ with degree n, it is complex to find all n of its zeros, even if the coefficients are all real. There are several practical methods

- Firstly, find one single root x_1 with any numerical method, i.e., Newton's method, and then continue to find another root via solving $p(x)/(x - x_1) = 0$. Do it repeatedly!
- Build the *Companion Matrix* of the given polynomial $p(x)$, then the eigenvalues of companion matrix are happen to be the roots of $p(x)$. (This is the method used by the *roots* function in matlab)
- Any other specially designed method, such as the method of Laguerre, of Bairstow, and of Traub, etc.

Roots for Nonlinear Equation Systems

It is more difficult for finding the roots for the equation system:

- A much wider range of behavior is possible, and a theoretical analysis of the existence and number of solution is much more complex
- No simple way as the conventional numerical methods. only Homotopy methods or interval methods are convergent globally
- Curse of Dimension !
- We are mainly concerned in the Fixed-Point Iteration, Newton's method and Secant Method as well.

Roots for Nonlinear Equation Systems

It is more difficult for finding the roots for the equation system:

- A much wider range of behavior is possible, and a theoretical analysis of the existence and number of solution is much more complex
- No simple way as the conventional numerical methods. only Homotopy methods or interval methods are convergent globally
- Curse of Dimension !
- We are mainly concerned in the Fixed-Point Iteration, Newton's method and Secant Method as well.

Roots for Nonlinear Equation Systems

It is more difficult for finding the roots for the equation system:

- A much wider range of behavior is possible, and a theoretical analysis of the existence and number of solution is much more complex
- No simple way as the conventional numerical methods. only Homotopy methods or interval methods are convergent globally
- Curse of Dimension !
- We are mainly concerned in the Fixed-Point Iteration, Newton's method and Secant Method as well.

Roots for Nonlinear Equation Systems

It is more difficult for finding the roots for the equation system:

- A much wider range of behavior is possible, and a theoretical analysis of the existence and number of solution is much more complex
- No simple way as the conventional numerical methods. only Homotopy methods or interval methods are convergent globally
- Curse of Dimension !
- We are mainly concerned in the Fixed-Point Iteration, Newton's method and Secant Method as well.

1. Fixed-Point Iteration

Given $g:\mathbb{R}^n \to \mathbb{R}^n$, then seek $\mathrm{x} \in \mathbb{R}^n$ as the fixed point of g, such that

$$\mathrm{x} = g(\mathrm{x}).$$

As well as the one dimensional case, the convergence (rate) is determined by certain $|g'(x^*)|$, precisely,

$$\rho(\mathrm{G}(\mathrm{x}^*)) < 1,$$

where $\mathrm{G}(\mathrm{x}^*)$ is the Jacobian at solution x^*.

$$\left\{ \mathrm{G}(\mathrm{x})_{ij} = \frac{\partial g_i(\mathrm{x})}{\partial x_j} \right\}$$

- If sufficient close, the fixed-point iteration convergence. Faster if closer.

1. Fixed-Point Iteration

Given $g:\mathbb{R}^n \to \mathbb{R}^n$, then seek $\mathrm{x} \in \mathbb{R}^n$ as the fixed point of g, such that

$$\mathrm{x} = g(\mathrm{x}).$$

As well as the one dimensional case, the convergence (rate) is determined by certain $|g'(x^*)|$, precisely,

$$\rho(\mathrm{G}(\mathrm{x}^*)) < 1,$$

where $\mathrm{G}(\mathrm{x}^*)$ is the Jacobian at solution x^*.

$$\left\{ \mathrm{G}(\mathrm{x})_{ij} = \frac{\partial g_i(\mathrm{x})}{\partial x_j} \right\}$$

- If sufficient close, the fixed-point iteration convergence. Faster if closer.

1. Fixed-Point Iteration

Given $g : \mathbb{R}^n \to \mathbb{R}^n$, then seek $\mathrm{x} \in \mathbb{R}^n$ as the fixed point of g, such that

$$\mathrm{x} = g(\mathrm{x}).$$

As well as the one dimensional case, the convergence (rate) is determined by certain $|g'(x^*)|$, precisely,

$$\rho(\mathrm{G}(\mathrm{x}^*)) < 1,$$

where $\mathrm{G}(\mathrm{x}^*)$ is the Jacobian at solution x^*.

$$\left\{ \mathrm{G}(\mathrm{x})_{ij} = \frac{\partial g_i(\mathrm{x})}{\partial x_j} \right\}$$

- If sufficient close, the fixed-point iteration convergence. Faster if closer.

1. Fixed-Point Iteration

Given $g: \mathbb{R}^n \to \mathbb{R}^n$, then seek $\mathrm{x} \in \mathbb{R}^n$ as the fixed point of g, such that

$$\mathrm{x} = g(\mathrm{x}).$$

As well as the one dimensional case, the convergence (rate) is determined by certain $|g'(x^*)|$, precisely,

$$\rho(\mathrm{G}(\mathrm{x}^*)) < 1,$$

where $\mathrm{G}(\mathrm{x}^*)$ is the Jacobian at solution x^*.

$$\left\{ \mathrm{G}(\mathrm{x})_{ij} = \frac{\partial g_i(\mathrm{x})}{\partial x_j} \right\}$$

- If sufficient close, the fixed-point iteration convergence. Faster if closer.

Example (Fixed-Point Iteration for System)

$$F(\mathrm{x}) := \begin{bmatrix} 3x_1 - \cos(x_1) - \sin(x_2) \\ 4x_2 - \sin(x_1) - \cos(x_2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

It is obvious that the nonlinear equation system can be rewritten with

$$\mathrm{x} = G(\mathrm{x}) = \begin{bmatrix} \big(\cos(x_1) + \sin(x_2)\big)/3 \\ \big(\sin(x_1) + \cos(x_2)\big)/4 \end{bmatrix},$$

the calculation is performed in matlab, in which the initial guess is $\mathrm{x}_0 = [1.5, 1]^T$.

Implementation of Fixed-Point iteration

```
f = @(x1,x2) [(cos(x1) + sin(x2))/3;( sin (x1) + cos(x2))/4];
h = @(x1,x2) [3*x1-cos(x1) - sin(x2);4*x2-sin(x1) - cos(x2)];
x0 = [1.5;1]; err = 1; temp = x0; i = 1;
while err>1e-5
    x(:, i) = f(temp(1),temp(2));
    err = max(abs(h(x(1,i),x(2, i))));
    temp = x(:,i);       i = i+ 1;
end
```

k	x_1^k	x_2^k	$f(x_1^k)$	$f(x_2^k)$
1	0.3041	0.3844	-0.4170	0.3114
2	0.4431	0.3066	0.1239	-0.1557
3	0.4018	0.3455	-0.0538	0.0501
4	0.4197	0.3330	0.0190	-0.0206
5	0.4134	0.3381	-0.0074	0.0075
6	0.4158	0.3363	0.0028	-0.0029
7	0.4149	0.3370	0.000396	-0.000410
8	0.4153	0.3367	-0.000150	0.000155
9	0.4152	0.3368	0.000057	-0.000059

Implementation of Fixed-Point iteration

```
f = @(x1,x2) [(cos(x1) + sin(x2))/3;( sin (x1) + cos(x2))/4];
h = @(x1,x2) [3*x1-cos(x1) - sin(x2);4*x2-sin(x1) - cos(x2)];
x0 = [1.5;1]; err = 1; temp = x0; i = 1;
while err>1e-5
    x (:, i) = f(temp(1),temp(2));
    err = max(abs(h(x(1,i),x(2, i))));
    temp = x(:,i);      i = i+ 1;
end
```

k	x_1^k	x_2^k	$f(x_1^k)$	$f(x_2^k)$
1	0.3041	0.3844	-0.4170	0.3114
2	0.4431	0.3066	0.1239	-0.1557
3	0.4018	0.3455	-0.0538	0.0501
4	0.4197	0.3330	0.0190	-0.0206
5	0.4134	0.3381	-0.0074	0.0075
6	0.4158	0.3363	0.0028	-0.0029
7	0.4149	0.3370	0.000396	-0.000410
8	0.4153	0.3367	-0.000150	0.000155
9	0.4152	0.3368	0.000057	-0.000059

2. Newton's Methods

Considering differentiable function f:$\mathbb{R}^n \to \mathbb{R}^n$ is based on the truncated Taylor series,

$$f(x+s) \approx f(x) + J(x)s,$$

where J(x)s is the Jacobian matrix of f,$\{J(x)\}_{ij} = \partial f_i(x)/\partial x_j$.

One can derivative the *Newton's method* for finding root of $f(x) = 0$, that is

Initialize x_0

for k=0,1,2,...

Obtain s_k by solving $J(x_k)s_k = -x_k$

$x_{k+1} = x_k + s_k$

end

- In each step of Newton's method, one linear equation is solved.

2. Newton's Methods

Considering differentiable function $f:\mathbb{R}^n \to \mathbb{R}^n$ is based on the truncated Taylor series,

$$f(x+s) \approx f(x) + J(x)s,$$

where $J(x)s$ is the Jacobian matrix of f, $\{J(x)\}_{ij} = \partial f_i(x)/\partial x_j$.

One can derivative the *Newton's method* for finding root of $f(x) = 0$, that is

Initialize x_0

for k=0,1,2,...

Obtain s_k by solving $J(x_k)s_k = -f(x_k)$ Assume: $f(x_k+s) \to 0$

$x_{k+1} = x_k + s_k$

end

- In each step of Newton's method, one linear equation is solved.

2. Newton's Methods

Considering differentiable function f:$\mathbb{R}^n \to \mathbb{R}^n$ is based on the truncated Taylor series,

$$f(x+s) \approx f(x) + J(x)s,$$

where J(x)s is the Jacobian matrix of f,$\{J(x)\}_{ij} = \partial f_i(x)/\partial x_j$.

One can derivative the *Newton's method* for finding root of $f(x) = 0$, that is

Initialize x_0

for k=0,1,2,...

Obtain s_k by solving $J(x_k)s_k = -f(x_k)$

$x_{k+1} = x_k + s_k$

end

- In each step of Newton's method, one linear equation is solved.

Implementation of Newton's method

```
J = @(x1,x2) [3+sin(x1),-cos(x2);-cos(x1),4+sin(x2)];
h = @(x1,x2)[3*x1-cos(x1) - sin(x2);4*x2-sin(x1) - cos(x2)];
x0 = [1.5;1];   temp = x0; err = 1; i = 1;
while err > 1e-5 && i <= 10
    s = J(temp(1),temp(2))\(-h(temp(1),temp(2)));
    x(:, i) = temp + s;
    err = max(abs(h(x(1,i),x(2,i))));
    temp = x(:,i);   i = i+1;
end
```

k	1	2	3
x_1^k	0.5318	0.4189	0.4152
x_2^k	0.4773	0.3402	0.3368
$f(x_1^k)$	0.2743	0.0095	0.0000084
$f(x_2^k)$	0.5138	-0.0115	0.0000085

Implementation of Newton's method

```
J = @(x1,x2) [3+sin(x1),-cos(x2);-cos(x1),4+sin(x2)];
h = @(x1,x2)[3*x1-cos(x1) - sin(x2);4*x2-sin(x1) - cos(x2)];
x0 = [1.5;1];   temp = x0; err = 1; i = 1;
while err > 1e-5 && i <= 10
    s = J(temp(1),temp(2))\(-h(temp(1),temp(2)));
    x(:, i) = temp + s;
    err = max(abs(h(x(1,i),x(2,i))));
    temp = x(:,i);   i = i+1;
end
```

k	1	2	3
x_1^k	0.5318	0.4189	0.4152
x_2^k	0.4773	0.3402	0.3368
$f(x_1^k)$	0.2743	0.0095	0.0000084
$f(x_2^k)$	0.5138	-0.0115	0.0000085

3. Secant Method

Partial derivatives in the Jacobian matrix is approximated by *divided difference*:

Initialize the x_0

Initialize Jacobian matrix B_0

for k=0,1,2,...

Solve $B_k s_k = -x_k$ $-f(x_k)$

$x_{k+1} = x_k + s_k$

$y_k = f(x_{k+1}) - f(x_k)$

$B_{k+1} = B_k + [(y_k - B_k s_k)S_k^T]/(s_k^T s_k)$

end

```
x0 = [1.5;1]; J0 = J(x0(1),x0(2));
tempx1 = x0; tempJ = J0; err = 1; i = 0;
while err > 1e-5 && i <= 10
    i = i + 1; x(:, i) = tempx1;
    s = tempJ\(-h(tempx1(1),tempx1(2)));
    tempx2 = tempx1 + s;
    yk = h(tempx2(1),tempx2(2)) - h(tempx1(1),
            tempx1(2));
    tempJ = tempJ + ((yk - tempJ*s)*s')/(s'*s); %
            Update J
    err = max(abs(h(x(1,i),x(2,i))));
    tempx1 = tempx2;
%   printf("Step %d: x = %f, f(x) = %f.\n",i,
            tempx1, yk);
```

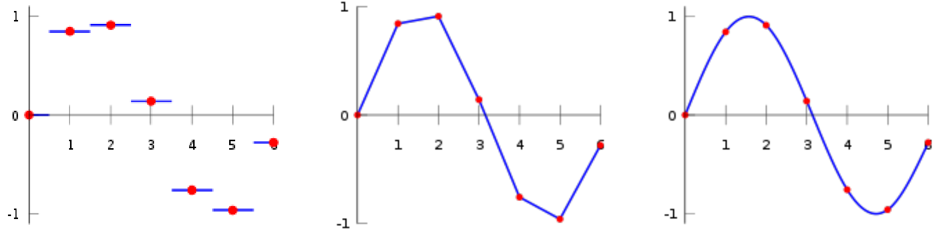
Implementation of Secant Method - conti.

k	1	2	3	4
x_1^k	0.5318	0.4382	0.4160	0.4152
x_2^k	0.4773	0.3563	0.3376	0.3358
$f(x_1^k)$	0.2743	0.0601	0.0022	0.00000315
$f(x_2^k)$	0.5138	0.0637	0.0029	-0.00000108

Chapter 5. Equation Roots

Chapter 7. ~~Approximation &~~ Interpolation

Interpolation Problem

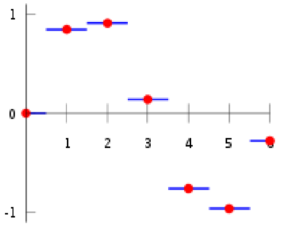


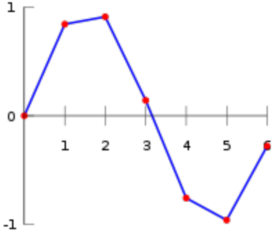
In one dimensional case: for given data $\{t_i, y_i\}_{i=0}^n$ with $t_0 < t_1 < \cdots < t_n$, one need to seek a function $f : \mathbb{R} \to \mathbb{R}$ such that

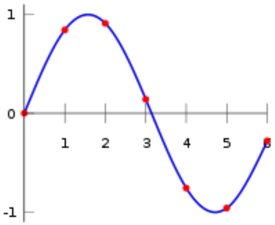
$$f(t_i) = y_i, \qquad \forall i = 0, 1, ..., n,$$

where f is referred as an *interpolating function*, or simply *interpolant* for given data, and $\{t_i\}_{i=0}^n$ is the knots.

Interpolation Problem







In one dimensional case: for given data $\{t_i, y_i\}_{i=0}^n$ with $t_0 < t_1 < \cdots < t_n$, one need to seek a function $f : \mathbb{R} \to \mathbb{R}$ such that

$$f(t_i) = y_i, \qquad \forall i = 0, 1, ..., n,$$

where f is referred as an *interpolating function*, or simply *interpolant* for given data, and $\{t_i\}_{i=0}^n$ is the knots.

About Interpolation

Purpose of Interpolation:

- Plot a smooth curve through discrete data points
- Reading between the lines of a table
- Differentiating or integrating tabular data
- Evaluating a mathematical function quickly and easily
- Replacing(Surrogate) a complicated function by a simple one

Main issues for numerical computation:

1. Accurately approximating infinite-dimensional problems by finite-dimensional problems
2. Developing methods for solving the resulting finite-dimensional problems accurately and efficiently.

About Interpolation

Purpose of Interpolation:

- Plot a smooth curve through discrete data points
- Reading between the lines of a table
- Differentiating or integrating tabular data
- Evaluating a mathematical function quickly and easily
- Replacing(Surrogate) a complicated function by a simple one

Main issues for numerical computation:

1. Accurately approximating infinite-dimensional problems by finite-dimensional problems
2. Developing methods for solving the resulting finite-dimensional problems accurately and efficiently.

About Interpolation

Purpose of Interpolation:

- Plot a smooth curve through discrete data points
- Reading between the lines of a table
- Differentiating or integrating tabular data
- Evaluating a mathematical function quickly and easily
- Replacing(Surrogate) a complicated function by a simple one

Main issues for numerical computation:

1. Accurately approximating infinite-dimensional problems by finite-dimensional problems
2. Developing methods for solving the resulting finite-dimensional problems accurately and efficiently.

About Interpolation

Purpose of Interpolation:

- Plot a smooth curve through discrete data points
- Reading between the lines of a table
- Differentiating or integrating tabular data
- Evaluating a mathematical function quickly and easily
- Replacing(Surrogate) a complicated function by a simple one

Main issues for numerical computation:

1. Accurately approximating infinite-dimensional problems by finite-dimensional problems
2. Developing methods for solving the resulting finite-dimensional problems accurately and efficiently.

About Interpolation

Purpose of Interpolation:

- Plot a smooth curve through discrete data points
- Reading between the lines of a table
- Differentiating or integrating tabular data
- Evaluating a mathematical function quickly and easily
- Replacing(Surrogate) a complicated function by a simple one

Main issues for numerical computation:

1. Accurately approximating infinite-dimensional problems by finite-dimensional problems
2. Developing methods for solving the resulting finite-dimensional problems accurately and efficiently.

About Interpolation

Purpose of Interpolation:

- Plot a smooth curve through discrete data points
- Reading between the lines of a table
- Differentiating or integrating tabular data
- Evaluating a mathematical function quickly and easily
- Replacing(Surrogate) a complicated function by a simple one

Main issues for numerical computation:

1. Accurately approximating infinite-dimensional problems by finite-dimensional problems
2. Developing methods for solving the resulting finite-dimensional problems accurately and efficiently.

About Interpolation

Purpose of Interpolation:

- Plot a smooth curve through discrete data points
- Reading between the lines of a table
- Differentiating or integrating tabular data
- Evaluating a mathematical function quickly and easily
- Replacing(Surrogate) a complicated function by a simple one

Main issues for numerical computation:

1. Accurately approximating infinite-dimensional problems by finite-dimensional problems
2. Developing methods for solving the resulting finite-dimensional problems accurately and efficiently.

Open Questions

- What form should the function have? (Polynomials, Piecewise polynomials, Trigonometric function, Exponential functions, Rational functions, etc.)
- How should the function behave between data point?
- Should the function inherit properties of the data, such as smooth, monotone or periodic?
- Are we interested primarily in the values of the parameters that define the interpolating function, or simply evaluation of the function at specific points?
- If the function ad data are plotted, should the results be visually pleasing?

General Methods for Interpolation

For given data (t_i, y_i), $i = 1, 2, ..., n$, choose $\phi_1(t), \phi_2(t), ...\phi_n(t)$ as a set of basis function, and to find the interpolating function in the span of the basis.

Let us write the interpolation funciton f into

$$f(t) = \sum_{j=1}^{n} x_j \phi_j(t),$$

where x_j is the parameter need to be decided.

According to the *interpolating condition* at point (t_i, y_i), that is

$$f(t_i) = \sum_{j=1}^{n} x_j \phi_j(t_i) = y_i, \tag{2}$$

which could be written with matrix form Ax = y, with its entries $a_{ij} = \phi_j(t_i)$.

General Methods for Interpolation

For given data $(t_i, y_i), i = 1, 2, ..., n$, choose $\phi_1(t), \phi_2(t), ...\phi_n(t)$ as a set of basis function, and to find the interpolating function in the span of the basis.

Let us write the interpolation funciton f into

$$f(t) = \sum_{j=1}^{n} x_j \phi_j(t),$$

where x_j is the parameter need to be decided.

According to the *interpolating condition* at point (t_i, y_i), that is

$$f(t_i) = \sum_{j=1}^{n} x_j \phi_j(t_i) = y_i, \tag{2}$$

which could be written with matrix form $Ax = y$, with its entries $a_{ij} = \phi_j(t_i)$.

General Methods for Interpolation

For given data $(t_i, y_i), i = 1, 2, ..., n$, choose $\phi_1(t), \phi_2(t), ...\phi_n(t)$ as a set of basis function, and to find the interpolating function in the span of the basis.

Let us write the interpolation funciton f into

$$f(t) = \sum_{j=1}^{n} x_j \phi_j(t),$$

where x_j is the parameter need to be decided.

According to the *interpolating condition* at point (t_i, y_i), that is

$$f(t_i) = \sum_{j=1}^{n} x_j \phi_j(t_i) = y_i, \tag{2}$$

which could be written with matrix form $\mathrm{Ax} = \mathrm{y}$, with its entries $a_{ij} = \phi_j(t_i)$.

Existence, Uniqueness and Conditioning

1. 存在性(Existence) + 唯一性(Uniqueness)
   若基函数个数n与数据个数m相等，则得到的(2)是一个方阵线性方程组。若矩阵A非奇异，则一定有且仅有唯一解。而若矩阵A奇异，则可以有许多参数的解，代表着数据点不能被精确拟合。
2. 病态性(Conditioning)
   基函数可以有许多选择的方式，相对应的会有许多矩阵A的表达形式。A若是单位阵，下三角矩阵，三对角矩阵等等特殊的矩阵，会大大提升求解参数的效率，降低求解的难度，在之后的具体例子里有所体现。
3. Polynomial is the simplest and most common type of interpolation

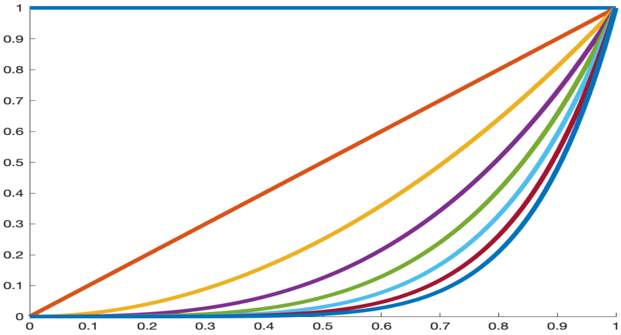
Existence, Uniqueness and Conditioning

1. 存在性(Existence) + 唯一性(Uniqueness)
   若基函数个数n与数据个数m相等，则得到的(2)是一个方阵线性方程组。若矩阵A非奇异，则一定有且仅有唯一解。而若矩阵A奇异，则可以有许多参数的解，代表着数据点不能被精确拟合。
2. 病态性(Conditioning)
   基函数可以有许多选择的方式，相对应的会有许多矩阵A的表达形式。A若是单位阵，下三角矩阵，三对角矩阵等等特殊的矩阵，会大大提升求解参数的效率，降低求解的难度，在之后的具体例子里有所体现。
3. Polynomial is the simplest and most common type of interpolation

Monomial Basis

The vector space of polynomials of degree at most $n-1$, and the basis set is the first *n monomials*, and this is the most natural basis for $\mathbb{P}_{n-1}$.

$$\phi_j(t) = t^{j-1}, \quad j = 1, 2, ..., n.$$



1. Interpolation with Monomial

It is worth to mention that any given polynomial $p_{n-1} \in \mathbb{P}_{n-1}$ has the form of

$$p_{n-1} = x_1 + x_2 t + \cdots, x_n t^{n-1}.$$

Then (2) leads to a $n \times n$ linear system:

$$Ax = \begin{bmatrix} 1 & t_1 & \dots & t_1^{n-1} \\ 1 & t_2 & \dots & t_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & \dots & t_n^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = y$$

1. Interpolation with Monomial

It is worth to mention that any given polynomial $p_{n-1} \in \mathbb{P}_{n-1}$ has the form of

$$p_{n-1} = x_1 + x_2 t + \cdots, x_n t^{n-1}.$$

Then (2) leads to a $n \times n$ linear system:

$$Ax = \begin{bmatrix} 1 & t_1 & \dots & t_1^{n-1} \\ 1 & t_2 & \dots & t_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & \dots & t_n^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = y$$

Example (Interpolating with Monomial Basis)

Example 7.1 Monomial Basis. To illustrate polynomial interpolation using the monomial basis, we will determine the polynomial of degree two interpolating the three data points $(-2,-27)$, $(0,-1)$, $(1,0)$. In general, there is a unique polynomial

$$p_2(t) = x_1 + x_2 t + x_3 t^2$$

of degree two interpolating three points (t_1, y_1), (t_2, y_2), (t_3, y_3). With the monomial basis, the coefficients of the polynomial are given by the system of linear equations

$$\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ 1 & t_3 & t_3^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \boldsymbol{y}.$$

For this particular set of data, this system becomes

$$\begin{bmatrix} 1 & -2 & 4 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -27 \\ -1 \\ 0 \end{bmatrix}.$$

Solving this system by Gaussian elimination yields the solution $\boldsymbol{x} = [\,-1 \quad 5 \quad -4\,]^T$, so that the interpolating polynomial is

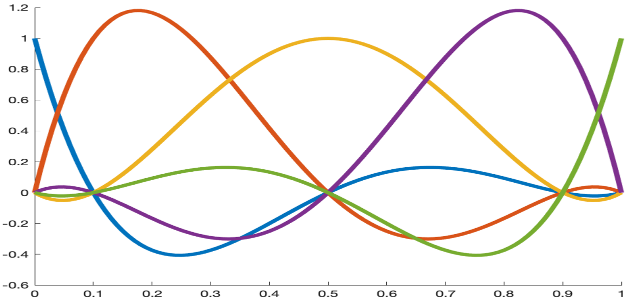
$$p_2(t) = -1 + 5t - 4t^2.$$

Fundamental Polynomials

For a given set of data points $(t_i, y_i), i = 1, 2, ..., n$, the Lagrange basis function for $\mathbb{P}_{n-1}$ are given by

$$l_j(t) = \frac{\prod_{k=1,k\neq j}^{n}(t - t_k)}{\prod_{k=1,k\neq j}^{n}(t_j - t_k)}, j = 1, 2, ..., n,$$

which is also called *fundamental polynomials*



2. Lagrange Interpolation

It is straightforward that

$$l_j(t) = \begin{cases} 1, & i = j, \\ 0, & i \neq j \end{cases} \quad i, j = 1, 2, \ldots, n,$$

Thus, the polynomial interpolating the data points (t_i, y_i) is given by

$$p_{n-1}(t) = y_1 l_1(t) + y_2 l_2(t) + \cdots + y_n l_n(t). \tag{3}$$

- $l_i(t)$ could be pre-computed, however, re-compute if order changed
- the above combination has a good geometrical explanation

2. Lagrange Interpolation

It is straightforward that

$$l_j(t) = \begin{cases} 1, & i = j, \\ 0, & i \neq j \end{cases} \quad i, j = 1, 2, \ldots, n,$$

Thus, the polynomial interpolating the data points (t_i, y_i) is given by

$$p_{n-1}(t) = y_1 l_1(t) + y_2 l_2(t) + \cdots + y_n l_n(t). \tag{3}$$

- $l_i(t)$ could be pre-computed, however, re-compute if order changed
- the above combination has a good geometrical explanation

2. Lagrange Interpolation

It is straightforward that

$$l_j(t) = \begin{cases} 1, & i = j, \\ 0, & i \neq j \end{cases} \quad i, j = 1, 2, \ldots, n,$$

Thus, the polynomial interpolating the data points (t_i, y_i) is given by

$$p_{n-1}(t) = y_1 l_1(t) + y_2 l_2(t) + \cdots + y_n l_n(t). \tag{3}$$

- $l_i(t)$ could be pre-computed, however, re-compute if order changed
- the above combination has a good geometrical explanation

Example (Interpolating with Lagrange Basis)

Example 7.2 Lagrange Interpolation. To illustrate Lagrange interpolation, we use it to determine the interpolating polynomial of degree two for the three data points from Example 7.1. Substituting these data, we obtain

$$\ell(t) = (t - t_1)(t - t_2)(t - t_3) = (t + 2)t(t - 1),$$

and the weights are given by

$$\begin{aligned} w_1 &= \frac{1}{(t_1 - t_2)(t_1 - t_3)} = \frac{1}{(-2)(-3)} = \frac{1}{6}, \\ w_2 &= \frac{1}{(t_2 - t_1)(t_2 - t_3)} = \frac{1}{2(-1)} = -\frac{1}{2}, \\ w_3 &= \frac{1}{(t_3 - t_1)(t_3 - t_2)} = \frac{1}{3 \cdot 1} = \frac{1}{3}, \end{aligned}$$

so that the interpolating quadratic polynomial is given by

$$p_2(t) = (t + 2)t(t - 1)\left(-27\frac{1/6}{t + 2} - 1\frac{-1/2}{t} + 0\frac{1/3}{t - 1}\right).$$

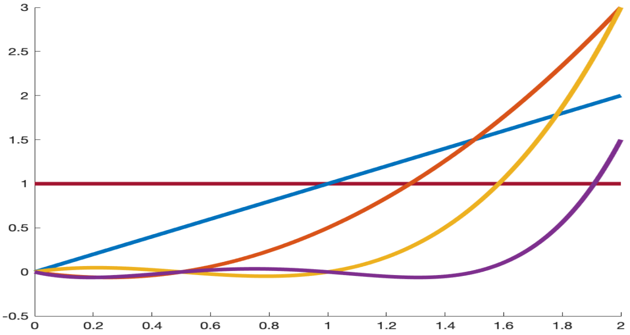
The polynomial can be evaluated efficiently in this form for any t, or it can be simplified to produce the same result we obtained in Example 7.1 using the monomial basis (as expected, since the interpolating polynomial is unique).

Newton basis functions

Given data points $\{(t_i, y_i)\}_{i=1}^n$, the Newton basis functions for $\mathbb{P}_{n-1}$ are given by

$$\pi_j(t) = \prod_{k=1}^{j-1}(t - t_k), \qquad \forall j = 1, 2, ..., n,$$

where we take the value of the product to be 1 when the limits make it vacuous.



3. Newton Interpolation

Let us consider now to construct a interpolating polynomial $Q_n(t)$ of degree $n-1$ via the *Newton iterpolation*. In the Newton basis, it is

$$Q_n(t) = x_1 + x_2(t-t_1) + x_3(t-t_1)(t-t_2) + \cdots + x_n(t-t_1)(t-t_2)\cdots(t-t_{n-1})$$

- In this sense, the coefficient matrix A of the interpolating system (2) is lower triangular. Hence the complexity of solving linear system Ax = y is $O(n^2)$.
- If one more data point (t_{n+1}, y_{n+1}) is taken into account, one can obtain a higher order interpolating

$$Q_{n+1}(t) = Q_n(t) + x_{n+1}\pi_{n+1}(t)\text{fi}$$

where the definition of π is the same as previous. Moreover,

$$x_{n+1} = \frac{y_{n+1} - Q_n(t_{n+1})}{\pi_{n+1}(t_{n+1})}.$$

3. Newton Interpolation

Let us consider now to construct a interpolating polynomial $Q_n(t)$ of degree $n-1$ via the *Newton iterpolation*. In the Newton basis, it is

$$Q_n(t) = x_1 + x_2(t-t_1) + x_3(t-t_1)(t-t_2) + \cdots + x_n(t-t_1)(t-t_2)\cdots(t-t_{n-1})$$

- In this sense, the coefficient matrix A of the interpolating system (2) is lower triangular. Hence the complexity of solving linear system Ax = y is $O(n^2)$.
- If one more data point (t_{n+1}, y_{n+1}) is taken into account, one can obtain a higher order interpolating

$$Q_{n+1}(t) = Q_n(t) + x_{n+1}\pi_{n+1}(t)\text{fi}$$

where the definition of π is the same as previous. Moreover,

$$x_{n+1} = \frac{y_{n+1} - Q_n(t_{n+1})}{\pi_{n+1}(t_{n+1})}.$$

3. Newton Interpolation

Let us consider now to construct a interpolating polynomial $Q_n(t)$ of degree $n-1$ via the *Newton iterpolation*. In the Newton basis, it is

$$Q_n(t) = x_1 + x_2(t-t_1) + x_3(t-t_1)(t-t_2) + \cdots + x_n(t-t_1)(t-t_2)\cdots(t-t_{n-1})$$

- In this sense, the coefficient matrix A of the interpolating system (2) is lower triangular. Hence the complexity of solving linear system Ax = y is $O(n^2)$.
- If one more data point (t_{n+1}, y_{n+1}) is taken into account, one can obtain a higher order interpolating

$$Q_{n+1}(t) = Q_n(t) + \underline{x_{n+1}}\pi_{n+1}(t)$$

where the definition of π is the same as previous. Moreover,

待定系数 $$\underline{x_{n+1}} = \frac{y_{n+1} - Q_n(t_{n+1})}{\pi_{n+1}(t_{n+1})}.$$

Example (Interpolating with Newton Basis)

Example 7.3 Newton Interpolation. To illustrate Newton interpolation, we use it to determine the interpolating polynomial for the three data points from Example 7.1. With the Newton basis, we have the lower triangular linear system

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & t_2 - t_1 & 0 \\ 1 & t_3 - t_1 & (t_3 - t_1)(t_3 - t_2) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}.$$

For the data from Example 7.1, this system becomes

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 3 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -27 \\ -1 \\ 0 \end{bmatrix},$$

whose solution, obtained by forward-substitution, is $\boldsymbol{x} = \begin{bmatrix} -27 & 13 & -4 \end{bmatrix}^T$. Thus, the interpolating polynomial is

$$p(t) = -27 + 13(t+2) - 4(t+2)t,$$

which reduces to the same polynomial we obtained earlier by either of the other two methods.

Example (Interpolating Incrementally with Newton Basis)

Example 7.4 Incremental Newton Interpolation. We illustrate by building the Newton interpolant for the previous example incrementally as new data points are added. We begin with the first data point, $(t_1, y_1) = (-2, -27)$, which is interpolated by the constant polynomial

$$p_1(t) = y_1 = -27.$$

Incorporating the second data point, $(t_2, y_2) = (0, -1)$, we modify the previous polynomial so that it interpolates the new data point as well:

"正交化"

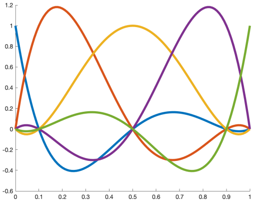
$$\begin{aligned} p_2(t) &= p_1(t) + x_2\,\pi_2(t) = p_1(t) + \frac{y_2 - p_1(t_2)}{\pi_2(t_2)}\pi_2(t) \\ &= p_1(t) + \frac{y_2 - y_1}{t_2 - t_1}(t - t_1) = -27 + 13(t+2). \end{aligned}$$

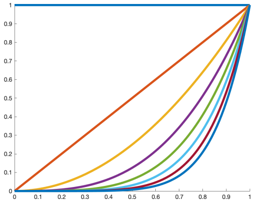
Finally, we incorporate the third data point, $(t_3, y_3) = (1, 0)$, modifying the previous polynomial so that it interpolates the new data point as well:

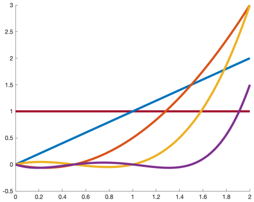
$$\begin{aligned} p_3(t) &= p_2(t) + x_3\,\pi_3(t) = p_2(t) + \frac{y_3 - p_2(t_3)}{\pi_3(t_3)}\pi_3(t) \\ &= p_2(t) + \frac{y_3 - p_2(t_3)}{(t_3 - t_1)(t_3 - t_2)}(t - t_1)(t - t_2) \\ &= -27 + 13(t+2) - 4(t+2)t. \end{aligned}$$

Question & Exercises

Calculated with three different schemes, is the interpolating polynomial the same?





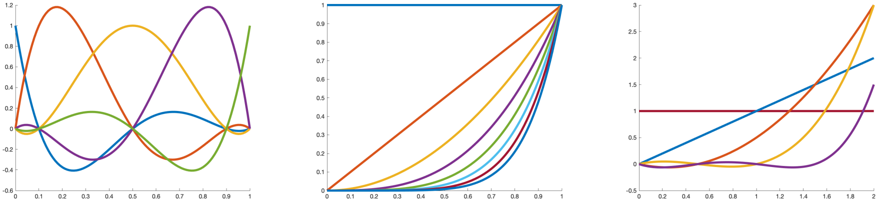


Do It Yourself: For given data point $(-1,1),(0,0),(1,1)$, please calculate the interpolating polynomial of degree 2 with

1. monomial basis
2. Lagrange basis
3. Newton basis

Question & Exercises

Calculated with three different schemes, is the interpolating polynomial the same?



Do It Yourself: For given data point $(-1,1),(0,0),(1,1)$, please calculate the interpolating polynomial of degree 2 with

1. monomial basis
2. Lagrange basis
3. Newton basis

Divided differences

The Newton polynomial interpolant could be computed via quantities known as

Definition (Divided differences)

$$f[x_0,\cdots,x_n] = \frac{f[x_1,\cdots,x_n] - f[x_0,\cdots,x_{n-1}]}{x_n - x_0},$$

which is defined in a recursive manner with lowest one as $f[x_k] = y_k, \forall k$.

The divided differences has certain properties

- $f[x_0,\cdots,x_n] = \dfrac{f^{(n)}(\xi)}{n!}$, where $\min\{x_i\} < \xi < \max\{x_i\}$;
- $f[x_0,\cdots,x_i,\cdots,x_j,\cdots,x_n] = f[x_0,\cdots,x_j,\cdots,x_i,\cdots,x_n]$
- $\frac{\partial}{\partial x} f[x_0,\cdots,x_n,x] = f[x_0,x_1,\cdots,x_n,x,x]$

Divided differences

The Newton polynomial interpolant could be computed via quantities known as

Definition (Divided differences)

$$f[x_0,\cdots,x_n]=\frac{f[x_1,\cdots,x_n]-f[x_0,\cdots,x_{n-1}]}{x_n-x_0},$$

which is defined in a recursive manner with lowest one as $f[x_k]=y_k,\forall k$.

The divided differences has certain properties

- $f[x_0,\cdots,x_n]=\frac{f^{(n)}(\xi)}{n!}$, wheremin$\{x_i\}<\xi<\max\{x_i\}$;
- $f[x_0,\cdots,x_i,\cdots,x_j,\cdots,x_n]=f[x_0,\cdots,x_j,\cdots,x_i,\cdots,x_n]$
- $\frac{\partial}{\partial x}f[x_0,\cdots,x_n,x]=f[x_0,x_1,\cdots,x_n,x,x]$

Error estimation for polynomial interpolant

It is straightforward to verify that

$$f(x) = P_n(x) + \omega_{n+1}(x) f[x_0, x_1, \cdots, x_n, x] \tag{4}$$

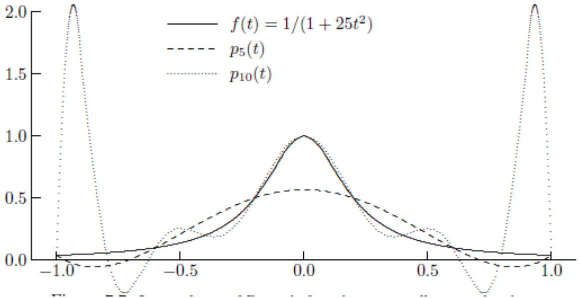
Thus the error estimation for the polynomial interpolant is

$$E_{n+1} = f(x) - P_n(x) = \omega_{n+1}(x) f[x_0, x_1, \cdots, x_n, x].$$

Especially, error for degree 2 interpolant: $E_2 = -\frac{h^2}{8} f''(\xi)$

Piecewise Polynomial Interpolation

If there are too many data points, i.e. 10+ points for one single curve, high order polynomials yield rapid oscillation, which is known as Runge phenomena.

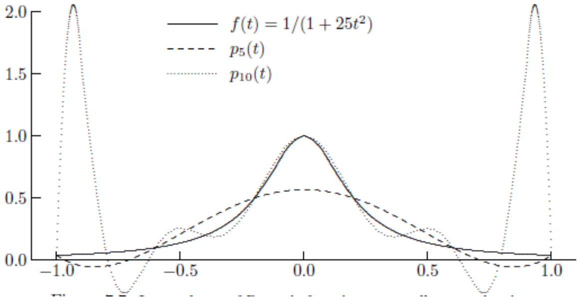


Definition (Piecewise polynomial interpolation)

For given data $(t_i, y_i), i = 1, \cdots, n, \quad t_1 < t_2 < \cdots < t_n$, find different interpolating polynomials for each interval $[t_i, t_{i+1}]$.

Piecewise Polynomial Interpolation

If there are too many data points, i.e. 10+ points for one single curve, high order polynomials yield rapid oscillation, which is known as Runge phenomena.

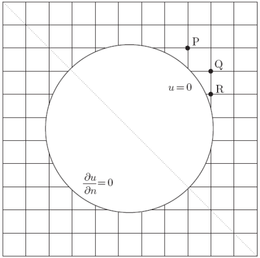


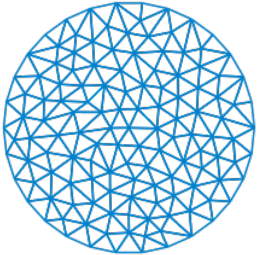
Definition (Piecewise polynomial interpolation)

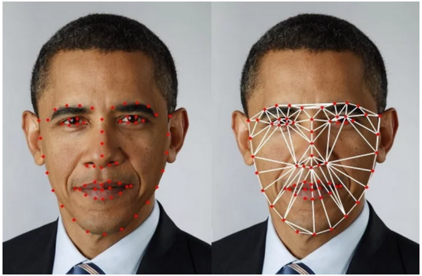
For given data $(t_i, y_i), i = 1, \cdots, n, \quad t_1 < t_2 < \cdots < t_n$, find different interpolating polynomials for each interval $[t_i, t_{i+1}]$.

A special example on piecewise interpolation

Example (Mesh Generation)







Classical piecewise interpolation

1. Cubic spline interpolation(via spline in MATLAB)
2. Cubic Hermit interpolation(via interp1 in MATLAB)

Example

t	0	1	3	4	6	7	9	10
y	8	6	5	2	1.5	1.3	1.1	1

```
t = [0 1 3 4 6 7 9 10];
y = [8 6 5 2 1.5 1.3 1.1 1];
x = 0:0.2:10;
yy = spline(t,y,x);
plot(t,y,'o',x,yy,'linewidth',2)
```

```
t = [0 1 3 4 6 7 9 10];
y = [8 6 5 2 1.5 1.3 1.1 1];
x = 0:0.2:10;
yy=interp1(t,y,x,'pchip');
plot(t,y,'o',x,yy,'linewidth',2)
```

Classical piecewise interpolation

1. Cubic spline interpolation(via spline in MATLAB)
2. Cubic Hermit interpolation(via interp1 in MATLAB)

Example

t	0	1	3	4	6	7	9	10
y	8	6	5	2	1.5	1.3	1.1	1

```
t = [0 1 3 4 6 7 9 10];
y = [8 6 5 2 1.5 1.3 1.1 1];
x = 0:0.2:10;
yy = spline(t,y,x);
plot(t,y,'o',x,yy,'linewidth',2)
```

```
t = [0 1 3 4 6 7 9 10];
y = [8 6 5 2 1.5 1.3 1.1 1];
x = 0:0.2:10;
yy=interp1(t,y,x,'pchip');
plot(t,y,'o',x,yy,'linewidth',2)
```

Classical piecewise interpolation

1. Cubic spline interpolation(via spline in MATLAB)
2. Cubic Hermit interpolation(via interp1 in MATLAB)

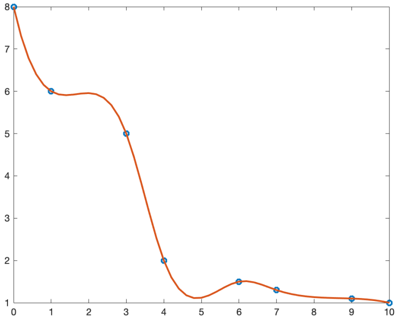
Example

t	0	1	3	4	6	7	9	10
y	8	6	5	2	1.5	1.3	1.1	1

```
t = [0 1 3 4 6 7 9 10];
y = [8 6 5 2 1.5 1.3 1.1 1];
x = 0:0.2:10;
yy = spline(t,y,x);
plot(t,y,'o',x,yy,'linewidth',2)
```

```
t = [0 1 3 4 6 7 9 10];
y = [8 6 5 2 1.5 1.3 1.1 1];
x = 0:0.2:10;
yy=interp1(t,y,x,'pchip');
plot(t,y,'o',x,yy,'linewidth',2)
```

Interpolation Results



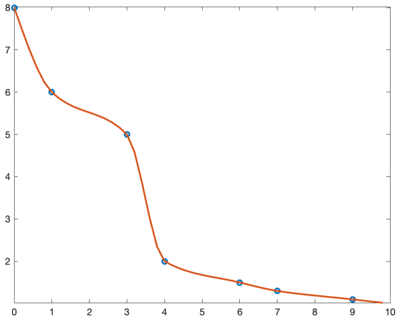


Figure: (Left)Interpolation with cubic splines;(Right)Interpolation with cubic Hermitt

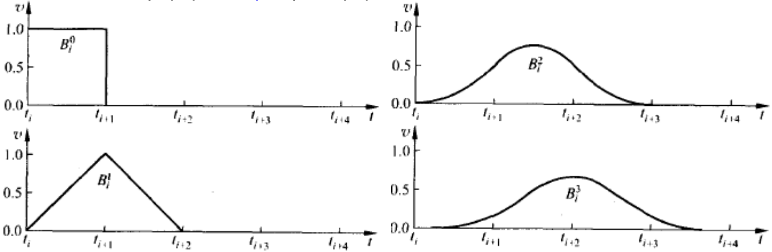
Definition (B-spline)

The 0_{th}-order B-spline is defined as following

$$B_i^0(x) = \begin{cases} 1 & x_i \leq x < x_{i+1} \\ 0 & others, \end{cases}$$

for any $k > 0$, the $k_{th} - order$ B-spline can be defined recursively

$$B_i^k(x) = v_i^k B_i^{k-1}(x) + (1 - v_{i+1}^k) B_{i+1}^{k-1}(x).$$



Definition (An alternated definition of B-spline)

The n_{th}-order B-spline can be formulated with

$$B_n(x) = \sum_{k=0}^{n+1} (-1)^k \binom{n+1}{k} \frac{(x - x_k)_+^n}{n!}, \tag{5}$$

where $x_k = k - \frac{n+1}{2}$ is the knots of spline, and the binom parameter's definition $0! = 1$. $(x - x_k)_+^n$ is the trunked monomial, for any given positive number n, it is defined as

$$(x - x_k)_+^n = \begin{cases} (x - x_k)^n, & if \quad (x - x_k) \geq 0, \\ 0, & if \quad (x - x_k) < 0. \end{cases}$$

- This definition is suitable for numerical analysis.
- The recursive definition is convenient for calculation.

Definition (An alternated definition of B-spline)

The n_{th}-order B-spline can be formulated with

$$B_n(x) = \sum_{k=0}^{n+1} (-1)^k \binom{n+1}{k} \frac{(x-x_k)_+^n}{n!}, \tag{5}$$

where $x_k = k - \frac{n+1}{2}$ is the knots of spline, and the binom parameter's definition $0! = 1$. $(x-x_k)_+^n$ is the trunked monomial, for any given positive number n, it is defined as

$$(x-x_k)_+^n = \begin{cases} (x-x_k)^n, & if \quad (x-x_k) \geq 0, \\ 0, & if \quad (x-x_k) < 0. \end{cases}$$

- This definition is suitable for numerical analysis.
- The recursive definition is convenient for calculation.

Approximation by least square

Definition (Least square approximation)

With some known data $\{t_i, y_i\}_{i=0}^{n}$, find out function $f : \mathbb{R} \to \mathbb{R}$, which satisfies:

$$\min_f J = \min_f \sum_{i=0}^{n} \left(f(t_i) - y_i\right)^2.$$

Choosing some base functions $\phi_1(t), \phi_2(t), \ldots, \phi_n(t)$, the fitting function can be represented as $f(t_i) = \sum_{j=1}^{n} x_j \phi_j(t_i)$. In this sense, the least square approximation can be transferred to

$$\mathrm{A}^T A\mathrm{x} = \mathrm{A}^T \mathrm{y},$$

with entries of the interpolating matrix being $A_{ij} = \phi_j(t_i)$.

Approximation by least square

Definition (Least square approximation)

With some known data $\{t_i, y_i\}_{i=0}^{n}$, find out function $f : \mathbb{R} \to \mathbb{R}$, which satisfies:

$$\min_f J = \min_f \sum_{i=0}^{n} \left(f(t_i) - y_i\right)^2.$$

Choosing some base functions $\phi_1(t), \phi_2(t), \ldots, \phi_n(t)$, the fitting function can be represented as $f(t_i) = \sum_{j=1}^{n} x_j \phi_j(t_i)$. In this sense, the least square approximation can be transferred to

$$\mathrm{A}^T A\mathrm{x} = \mathrm{A}^T \mathrm{y},$$

with entries of the interpolating matrix being $A_{ij} = \phi_j(t_i)$.

Numerical approach for implementation least square

Let us consider three types of commonly used base functions for approximation

1. Polynomial base $\phi_j(t) = t^{j-1}, j = 1, 2, \cdots, k$, such that

$$A^1_{ij} = t_i^{j-1}.$$

2. Non-polynomial base $\phi_j(t) = f_j(t), j = 1, 2, \ldots, k$, such that

$$A^2_{ij} = f_j(t_i).$$

3. B-spline base with knots $\{m_j | t_1 \leq m_1, m_2, \ldots, m_k \leq t_n, k < n-1\}$, and

$$A^3_{ij} = B^k_j(t_i).$$

Example

Using the above base functions to fit the COCID-19 data, including confirmed cases, suspected cases, death and cure cases. The data are shown as following:

Date	1.25	1.26	1.27	1.28	1.29	1.30	1.31
Confirmed	1377	2071	2846	4630	6086	7830	9811
Suspected	1983	2692	5794	6973	9239	12167	15238
Death	41	56	81	106	132	171	213
Cure	39	49	56	73	119	135	214

2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8
11890	14490	17341	20530	24439	31161	31774	34673
17988	19544	21558	23214	23260	26359	27657	27657
259	304	361	426	493	636	722	724
275	434	527	718	1019	1540	2050	2375

Figure: (Confirmed): Polynomial degree $k = 4$, Non-polynomial with $f(x) = lnx$, B-spline with $k = 3$, and Neural network with 56 parameters equipped with 2 hidden layers.

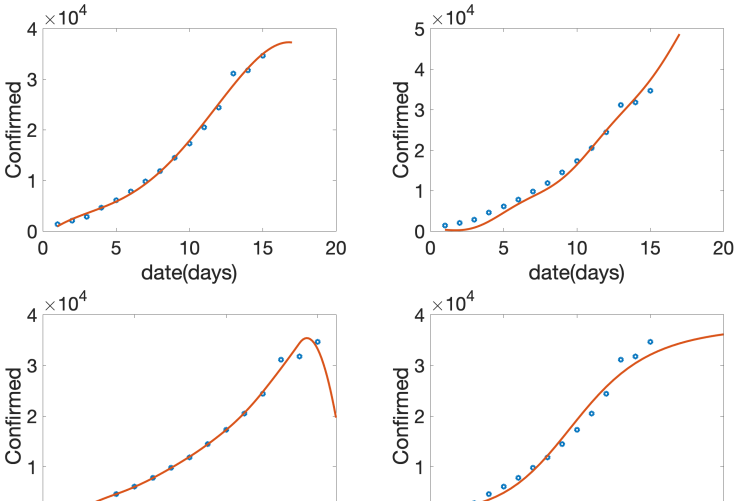


Figure: (Suspected): Polynomial degree $k = 4$, Non-polynomial with $f(x) = lnx$, B-spline with $k = 3$, and Neural network with 56 parameters equipped with 2 hidden layers.

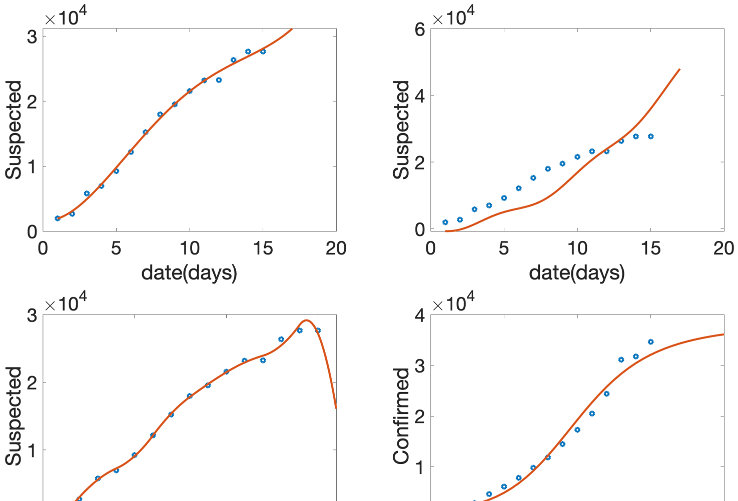


Figure: (Death): Polynomial degree $k = 4$, Non-polynomial with $f(x) = lnx$, B-spline with $k = 3$, and Neural network with 56 parameters equipped with 2 hidden layers.

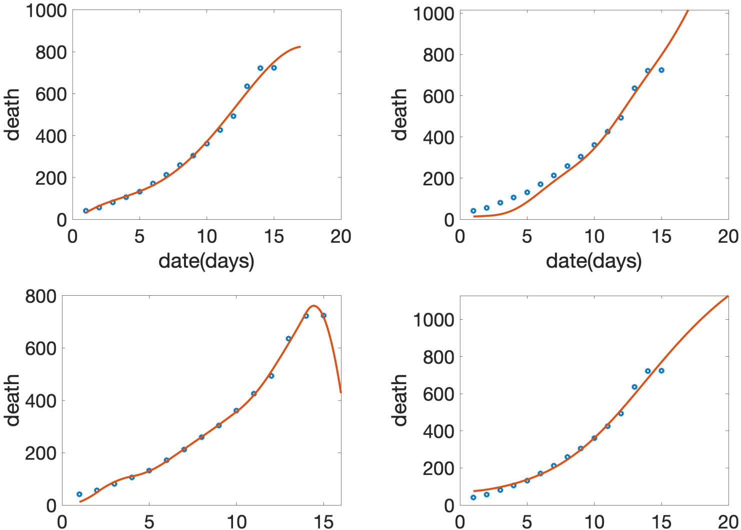
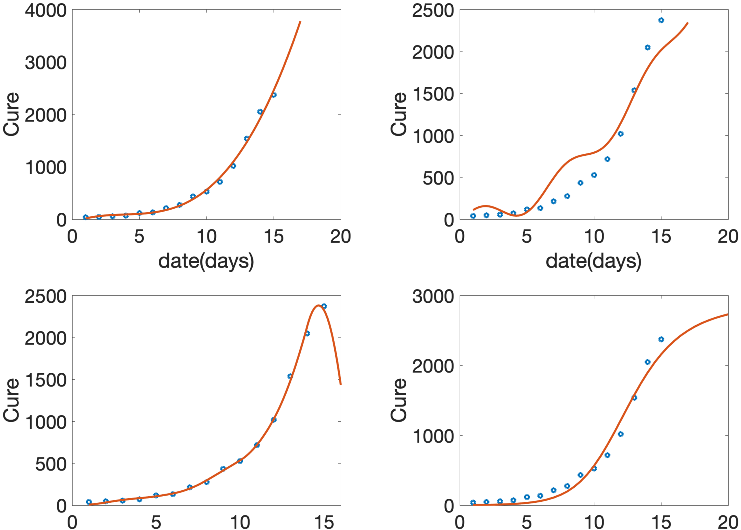


Figure: (Cure): Polynomial degree $k = 4$, Non-polynomial with $f(x) = lnx$, B-spline with $k = 3$, and Neural network with 56 parameters equipped with 2 hidden layers.



Discussion

Difference and Relations between

- Interpolation - 插值
- Approximation - 逼近
- Fit - 拟合
- Regression - 回归

Discussion

Difference and Relations between

- Interpolation - 插值
- Approximation - 逼近
- Fit - 拟合
- Regression - 回归

Conclusion & Homework 3

Chapter 5. Equation Roots

Chapter 7. Approximation & Interpolation

Homework:

1. "Exercises" of Chapter **5**:

   5.1, 5.3(a), 5.9(b), 5.11, 5.13

2. "Exercises" of Chapter **7**:

   7.1, 7.3, 7.9, 7.11, 7.16

Conclusion & Homework 3

Chapter 5. Equation Roots

Chapter 7. Approximation & Interpolation

Homework:

1. "Exercises" of Chapter **5**:

   5.1, 5.3(a), 5.9(b), 5.11, 5.13

2. "Exercises" of Chapter **7**:

   7.1, 7.3, 7.9, 7.11, 7.16