



문양제 (컴퓨터과학전공, IT특성화대학, 강원대학교)



## In this chapter ...

보간법 (Interpolation)

### 보간법이란?

통계적 혹은 실험적으로 구해진 데이터들( $x_i$ )로부터,  
주어진 데이터를 만족하는 근사 함수( $f(x)$ )를 구하고,  
이 식을 이용하여 주어진 변수에 대한 함수 값을 구하는 일련의 과정을 의  
미한다.

예를 들어,  $(0, 0)$ ,  $(1, 10)$ ,  $(2, 20)$ 이 주어졌을 때, 이들에 대한 근사 함수  
를  $f(x) = 10x$ 로 구하고, 1.5에 대한 함수 값으로 15를 구하는 것이다.

We will cover ...

- 선형 보간법
- 라그랑제 다항식 보간법
- 네빌레의 반복 보간법
- 뉴턴 다항식에 의한 보간법
- 3차원 스플라인 보간법



### 선형 보간법

- ❏ 라그랑제 다항식 보간법
- ❏ 네빌레의 반복 보간법
- ❏ 뉴턴 다항식에 의한 보간법



### 선형 보간법 개념 (1/2)

- ❏ 선형 보간법은 주어진 두 점을 이은 직선의 방정식을 근사 함수로 사용하는 단순한 방법이다.
- ❏ 함수  $f(x)$ 가 폐구간  $[a, b]$  위에서 정의되고, 이 구간에 있는  $n$ 개의 점  $x_1, x_2, \dots, x_n$ 에 대하여 각각의 함수 값을 안다고 하자.
- ❏ 이때, 임의의 두 점  $(x_i, f(x_i)), (x_{i+1}, f(x_{i+1}))$ 을 지나는 직선의 방정식은 다음과 같다.

$$g(x) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i) + f(x_i)$$

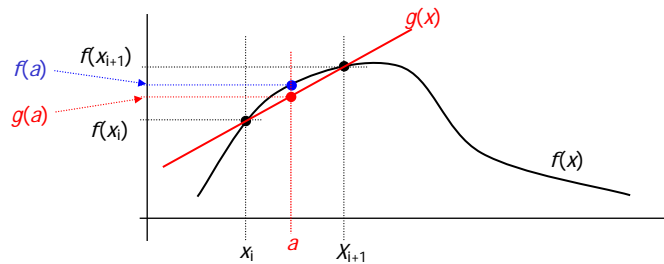
- ❏ 상기 식에서,  $g(x)$ 는  $(x_i, x_{i+1})$  사이의 임의의  $x$  값에 대한 선형 보간 값이 되는 것이다.



## 선형 보간법 개념 (2/2)

Linear Interpolation

### 선형 보간법의 원리



## 선형 보간법 - 알고리즘

Linear Interpolation

```
procedure linear_inter( $x_1, x_2, y_1, y_2$ : real numbers,  $x$ : real number)  
{ ( $x_1, y_1$ ) and ( $x_2, y_2$ ) are the initial points. }  
{  $x$  is the value that we want to get the  $f(x)$ . }
```

$$y := \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1;$$

```
return y;
```



## 선형 보간법 - 프로그램

Linear Interpolation

주어진 문제: 함수  $f(x) = e^x$ 에서  $f(1) = e$ ,  $f(2) = e^2$ 를 안다고 할 때,  $f(1.0)$ ,  $f(1.1)$ ,  $f(1.2)$ , ...,  $f(1.9)$ ,  $f(2.0)$ 의 값을 선형 보간법으로 구하라.

```
210.115.58.78 - Zterm
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

float f(float);

main(int argc, char *argv[])
{
    int i, j, k, n;
    float x1, x2, y1, y2, x, y;

    x1 = 1.0; y1 = f(x1);
    x2 = 2.0; y2 = f(x2);

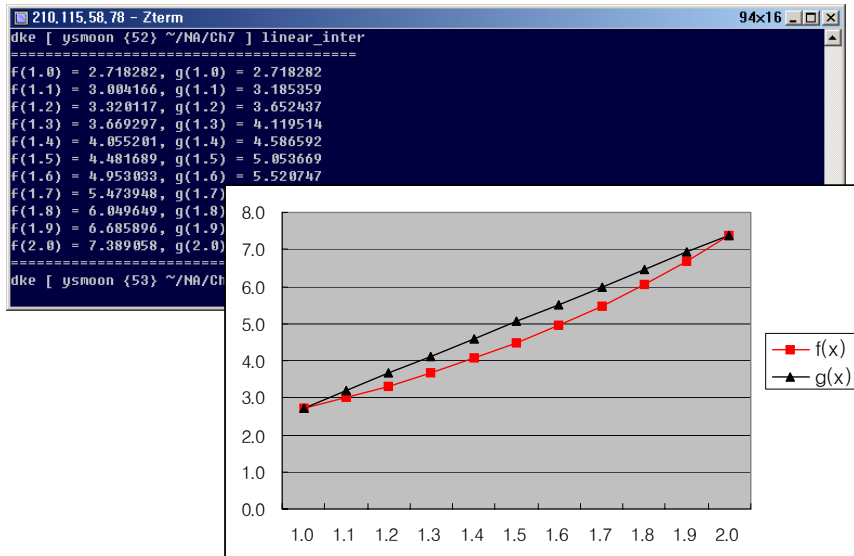
    printf("=====\\n");
    for(x = x1; x <= (x2+0.00001); x += 0.1) {
        y = ((y2 - y1)/(x2 - x1))*(x - x1) + y1;
        printf("f(%.1f) = %.6f, g(%.1f) = %.6f\\n", x, f(x), x, y);
    }
    printf("=====\\n");
}

float f(float x)
{
    return exp((float)x);
}
```



## 선형 보간법 - 실행 결과

Linear Interpolation





We are now ...

Lagrange Interpolation

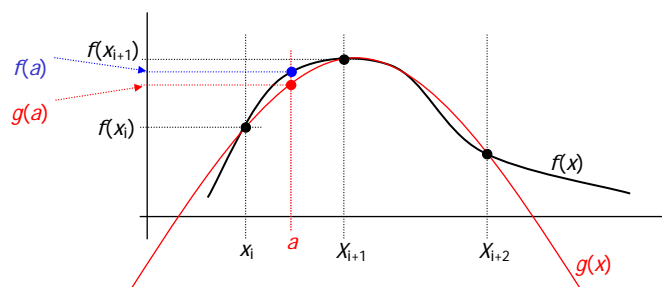
- 선형 보간법
- 라그랑제 다항식 보간법**
- 네빌레의 반복 보간법
- 뉴턴 다항식에 의한 보간법



## 라그랑제 보간법 개념 (1/6)

Lagrange Interpolation

- 점들을 단순히 직선으로 연결하는 것이 아니라, 여러 개의 점들을 지나는 곡선으로 연결하는 방법을 사용한다.
- 즉, 여러 개의 점들이 주어졌을 경우, 이들 점들을 지나는 다항식을 구하고, 이 다항식을 사용하여 주어진 점에 대한 보간 값을 구한다.





## 라그랑제 보간법 개념 (2/6)

Lagrange Interpolation

☞  $(n+1)$ 개의 점을 지나는  $n$ 차 다항식은 오로지 한 개 존재한다.

☞  $(n+1)$ 개의 점들이 다음과 같이 주어진다고 가정하자.

$$\begin{array}{lcl} x_0 & \rightarrow & y_0 \quad (= f(x_0)) \\ x_1 & \rightarrow & y_1 \quad (= f(x_1)) \\ x_2 & \rightarrow & y_2 \quad (= f(x_2)) \\ \vdots & & \vdots \\ x_n & \rightarrow & y_n \quad (= f(x_n)) \end{array}$$

☞ 여기에서,  $x_0, x_1, \dots, x_n$ 은  $(n+1)$ 개 점들의  $x$ 축 값이며, 그 간격은 일정하지 않아도 된다.



## 라그랑제 보간법 개념 (3/6)

Lagrange Interpolation

☞ 구하고자 하는  $n$ 차 다항식은 다음과 같이 표현할 수 있다.

$$g(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

☞ 다항식  $g(x)$ 에  $(n+1)$ 개 점들을 대입하여 다음의  $(n+1)$ 개 연립 방정식을 얻을 수 있다.

$$\begin{array}{lcl} y_0 & = & a_0 + a_1x_0 + a_2x_0^2 + \dots + a_nx_0^n \\ y_1 & = & a_0 + a_1x_1 + a_2x_1^2 + \dots + a_nx_1^n \\ y_2 & = & a_0 + a_1x_2 + a_2x_2^2 + \dots + a_nx_2^n \\ \vdots & & \vdots \\ y_n & = & a_0 + a_1x_n + a_2x_n^2 + \dots + a_nx_n^n \end{array}$$



## 라그랑제 보간법 개념 (4/6)

Lagrange Interpolation

- 상기 연립 방정식을 풀면, 계수  $a_0, a_1, \dots, a_n$ 을 구할 수 있고, 결국 다항식  $g(x)$ 를 구하여 다른  $x$  값에 대한 보간 값을 구하는데 사용할 수 있다.
- 연립 방정식을 풀기 위해서는 다른 프로그램이 필요하고, 정확성도 보장할 수 없다.

→ 라그랑제 보간법: 연립 방정식을 풀지 않고 다항식을 결정함



## 라그랑제 보간법 개념 (5/6)

Lagrange Interpolation

- 라그랑제의 식은  $n$ 차일 때의 식이 다음과 같다.

$$F(x) = (x - x_0)(x - x_1)(x - x_2) \cdots (x - x_n)$$

- 함수  $F(x)$ 는  $x = x_0, x_1, x_2, \dots, x_n$ 일 때 각각 0이 된다.

- $F(x)$ 를 각각의  $F(x_i)$ 에서 나눈 식을 다음과 같이  $G_i(x)$ 라 놓는다.  
(단, 나눌 때  $(x - x_i)$ 은 분모 및 분자에서 제외한다.)

$$\begin{aligned} G_0(x) &= \frac{F(x)}{F(x_0)} = \frac{(x - x_1)(x - x_2) \cdots (x - x_n)}{(x_0 - x_1)(x_0 - x_2) \cdots (x_0 - x_n)} \\ G_1(x) &= \frac{F(x)}{F(x_1)} = \frac{(x - x_0)(x - x_2) \cdots (x - x_n)}{(x_1 - x_0)(x_1 - x_2) \cdots (x_1 - x_n)} \\ &\vdots \\ G_i(x) &= \frac{F(x)}{F(x_i)} = \frac{(x - x_0)(x - x_1) \cdots (x - x_n)}{(x_i - x_0)(x_i - x_1) \cdots (x_i - x_n)} \end{aligned}$$



## 라그랑제 보간법 개념 (6/6)

Lagrange Interpolation

- 각각의  $G_i(x)$ 에  $y_i$ 를 곱하고, 이를 서로 더하면 그 합은 다음과 같이  $n$ 차 다항식이 된다.

$$\begin{aligned} g(x) &= G_0(x) \cdot y_0 + G_1(x) \cdot y_1 + \cdots + G_n(x) \cdot y_n \\ &= \frac{(x-x_1)(x-x_2)\cdots(x-x_n)}{(x_0-x_1)(x_0-x_2)\cdots(x_0-x_n)} \cdot y_0 + \frac{(x-x_0)(x-x_2)\cdots(x-x_n)}{(x_1-x_0)(x_1-x_2)\cdots(x_1-x_n)} \cdot y_1 \\ &\quad \cdots + \frac{(x-x_0)(x-x_1)\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_n)} \cdot y_i + \cdots \end{aligned}$$

- 상기  $g(x)$ 는 0과  $n$  사이의 모든  $x$ 에 대해서  $g(x_i) = y_i$ 를 만족한다.  
즉,  $g(x)$ 는 모든  $(x_i, f(x_i))$ 를 지나는 다항식이 된다.



## 라그랑제 보간법 - 알고리즘

Lagrange Interpolation

procedure *lagrange*( $x_0 \cdots x_n, y_0 \cdots y_n$ : real numbers,  $x$ : real number)

{  $(x_i, y_i)$ 's are the given points. }

{  $x$  is the value that we want to get the  $f(x)$ . }

$$\begin{aligned} y := & \frac{(x-x_1)(x-x_2)\cdots(x-x_n)}{(x_0-x_1)(x_0-x_2)\cdots(x_0-x_n)} \cdot y_0 + \frac{(x-x_0)(x-x_2)\cdots(x-x_n)}{(x_1-x_0)(x_1-x_2)\cdots(x_1-x_n)} \cdot y_1 \\ & \cdots + \frac{(x-x_0)(x-x_1)\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_n)} \cdot y_i \cdots \end{aligned}$$

return  $y$ ;





## 라그랑제 보간법 - 프로그램 (1/3)

Lagrange Interpolation

주어진 문제: 다음 표와 같이 네 개의  $x$  값과 이의 함수 값이 주어졌을 때,  $x = 1.5, 2.7, 3.4$ 일 때의 근사 함수 값을 라그랑제 보간법으로 구하시오.

$x$	$f(x)$
1	0.0
2	0.30103
3	0.47712
4	0.60206

참고: 상기 표의 함수 값은  $f(x) = \log_{10}(x)$ 에 해당한다.



## 라그랑제 보간법 - 프로그램 (2/3)

Lagrange Interpolation

```
210.115.58.78 - Zterm 80x30
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX_PTS 10

float f(float);


main(int argc, char *argv[])
{
    int i, j, n;
    float xi[MAX_PTS], yi[MAX_PTS], x, y, dividend, divisor;

    if(argc < 3) {
        printf("Usage: %s input-file x\n", argv[0]);
        exit(0);
    }

    if(read_input(argv[1], xi, yi, &n) < 0) {
        printf("cannot read the input file: %s\n", argv[1]);
        exit(-1);
    }

    x = (float)atof(argv[2]);

    printf("(xi, yi): input =====\n");
    for(i=0; i < n; i++)
        printf("%.5f, %.5f\n", xi[i], yi[i]);
    printf("===== \n");
}
```



## 라그랑제 보간법 - 프로그램 (3/3)

Lagrange Interpolation

210.115.58.78 - Zterm

```

y = 0.0;
for(i=0; i < n; i++) {
    dividend = divisor = 1.0;
    for(j=0; j < n; j++)
        if(i != j) {
            dividend *= (x - xi[j]);
            divisor  *= (xi[i] - xi[j]);
        }
    y += (dividend/divisor)*yi[i];
}

printf("(x, g(x), log(x)) = (%.5f, %.5f, %.5f)\n", x, y, log10(x));
}

read_input(char *file, float x[], float y[], int *n)
{
    int i;
    FILE *fp;


    if((fp = fopen(file, "r")) == (FILE *)0) return -1;

    fscanf(fp, "%d", (char *)n);
    for(i=0; i < (int)*n; i++)
        fscanf(fp, "%f%f", (char *)&x[i], (char *)&y[i]);

    return 1;
}

```


58,1 Bot



김영대학교  
컴퓨터공학과

Page 19

*Numerical Analysis  
by Yang-Sae Moon*



## 라그랑제 보간법 - 실행 결과 (1/2)

Lagrange Interpolation

**입력 파일**

210.115.58.78 - Zterm

```

dke [ ysmoon {62} ~/NA/Ch7 ] cat la_input1
4
1.0 0.0
2.0 0.30103
3.0 0.47712
4.0 0.60206
dke [ ysmoon {63} ~/NA/Ch7 ]

```


**실행 결과 ( $x = 1.5$ )**

210.115.58.78 - Zterm

```

dke [ ysmoon {67} ~/NA/Ch7 ] lagrange la_input1 1.5
-(xi, yi): input =====
(1.00000, 0.00000)
(2.00000, 0.30103)
(3.00000, 0.47712)
(4.00000, 0.60206)
=====
(x, g(x), log(x)) = (1.50000, 0.17609, 0.17609)
dke [ ysmoon {68} ~/NA/Ch7 ]

```



김영대학교  
컴퓨터공학과

Page 20

*Numerical Analysis  
by Yang-Sae Moon*



## 라그랑제 보간법 - 실행 결과 (2/2)

Lagrange Interpolation

### 실행 결과 ( $x = 2.7$ )

```
210.115.58.78 - Zterm
dke [ ysmoon {68} ~/NA/Ch7 ] lagrange la_input1 2.7
=(xi, yi): input =====
(1.00000, 0.00000)
(2.00000, 0.30103)
(3.00000, 0.47712)
(4.00000, 0.60206)
=====
(x, g(x), log(x)) = (2.70000, 0.43302, 0.43136)
dke [ ysmoon {69} ~/NA/Ch7 ]
```

### 실행 결과 ( $x = 3.4$ )

```
210.115.58.78 - Zterm
dke [ ysmoon {69} ~/NA/Ch7 ] lagrange la_input1 3.4
=(xi, yi): input =====
(1.00000, 0.00000)
(2.00000, 0.30103)
(3.00000, 0.47712)
(4.00000, 0.60206)
=====
(x, g(x), log(x)) = (3.40000, 0.52910, 0.53148)
dke [ ysmoon {70} ~/NA/Ch7 ]
```



## We are now ...

Neville Interpolation

### 선형 보간법

### 라그랑제 다항식 보간법

### 네빌레의 반복 보간법

### 뉴턴 다항식에 의한 보간법



## 네빌레의 반복 보간법 개념 (1/5)

Neville Interpolation

### ❏ 라그랑제 보간법의 문제점:

기존 데이터에 덧붙여 새로운 점이 하나만 추가되어도,  
(앞서 구한 다항식을 사용하지 못하고) 다항식을 다시 계산해야 한다.

→ 네빌레의 반복 보간법: 앞서 구한 계산이나 결과를 다음 단계에서 사용하는 방법으로서, 새로운 점이 지속적으로 추가될 경우 매우 적합하다.

### ❏ 네빌레의 반복 보간법의 다항식 구성 개념

- 한 점에 대한 0차 다항식을 구한다.
- 앞서 구한 0차 다항식을 사용하여, 두 점에 대한 1차 다항식을 구한다.
- 앞서 구한 1차 다항식을 사용하여, 세 점에 대한 2차 다항식을 구한다.
- ...
- 앞서 구한  $(n-1)$ 차 다항식을 사용하여,  $(n+1)$ 개 점에 대한  $n$ 차 다항식을 구한다.



## 네빌레의 반복 보간법 개념 (2/5)

Neville Interpolation

### ❏ 한 점에 대한 0차 다항식을 다음과 같이 구한다. (단, $g_i(x) = g(x_i)$ 이다.)

$$\begin{aligned} g_0(x) &= f(x_0) \\ g_1(x) &= f(x_1) \\ g_2(x) &= f(x_2) \\ &\vdots \end{aligned}$$

### ❏ 두 점에 대한 1차 다항식을 앞서의 0차 다항식을 사용하여 구한다.

- 라그랑제 보간법에 따르면, 두 점  $x_0, x_1$ 을 지나는 1차 다항식은 다음과 같다.

$$g_{0,1}(x) = \frac{x-x_1}{x_0-x_1} f(x_0) + \frac{x-x_0}{x_1-x_0} f(x_1) = \frac{x-x_1}{x_0-x_1} g_0(x) + \frac{x-x_0}{x_1-x_0} g_1(x)$$

- 마찬가지로, 두 점  $x_1, x_2$ 을 지나는 1차 다항식은 다음과 같다.

$$g_{1,2}(x) = \frac{x-x_2}{x_1-x_2} f(x_1) + \frac{x-x_1}{x_2-x_1} f(x_2) = \frac{x-x_2}{x_1-x_2} g_1(x) + \frac{x-x_1}{x_2-x_1} g_2(x)$$



## 네빌레의 반복 보간법 개념 (3/5)

Neville Interpolation

- 다음 표기법을 사용하여, 두 점을 지나는 1차 다항식을 간략히 나타낸다.

$$\text{표기법} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - cb$$

$$g_{0,1}(x) = \frac{1}{x_1 - x_0} \begin{vmatrix} x - x_0 & g_0(x) \\ x - x_1 & g_1(x) \end{vmatrix} = \frac{(x - x_0)g_1(x) - (x - x_1)g_0(x)}{x_1 - x_0}$$

$$g_{1,2}(x) = \frac{1}{x_2 - x_1} \begin{vmatrix} x - x_1 & g_1(x) \\ x - x_2 & g_2(x) \end{vmatrix} = \frac{(x - x_1)g_2(x) - (x - x_2)g_1(x)}{x_2 - x_1}$$



## 네빌레의 반복 보간법 개념 (4/5)

Neville Interpolation

- 같은 방식으로, 세 점에 대한 2차 다항식을 앞서의 1차 다항식을 사용하여 구한다.

$$g_{0,1,2}(x) = \frac{1}{x_2 - x_0} \begin{vmatrix} x - x_0 & g_{0,1}(x) \\ x - x_2 & g_{1,2}(x) \end{vmatrix} = \frac{(x - x_0)g_{1,2}(x) - (x - x_2)g_{0,1}(x)}{x_2 - x_0}$$

- 마찬가지로, 네 점에 대한 3차 다항식을 앞서의 2차 다항식을 사용하여 구한다.

$$g_{0,1,2,3}(x) = \frac{1}{x_3 - x_0} \begin{vmatrix} x - x_0 & g_{0,1,2}(x) \\ x - x_3 & g_{1,2,3}(x) \end{vmatrix} = \frac{(x - x_0)g_{1,2,3}(x) - (x - x_3)g_{0,1,2}(x)}{x_3 - x_0}$$



## 네빌레의 반복 보간법 개념 (5/5)

Neville Interpolation

지금까지 구한 다항식들은 다음과 같이 표로 나타낼 수 있다.

$x_i$	$x - x_i$	$f_i(x) = g_i(x)$	근사 함수
$x_0$	$x - x_0$	$g_0(x)$	
$x_1$	$x - x_1$	$g_1(x)$	$g_{0,1}(x)$
$x_2$	$x - x_2$	$g_2(x)$	$g_{1,2}(x), g_{0,1,2}(x)$
$x_3$	$x - x_3$	$g_3(x)$	$g_{2,3}(x), g_{1,2,3}(x), g_{0,1,2,3}(x)$
...	...	...	...

결국, 주어진 개수의 점을 사용하여 다항식을 구하고, 이를 근사 값 계산에 사용한다.

그 이후에, 새로운 점이 추가되면, 이전 다항식에 이 점을 추가한 다항식을 다시 구하고, 이를 근사 값 계산에 사용한다.



Page 27

Numerical Analysis  
by Yang-Sae Moon



## 네빌레의 반복 보간법 - 알고리즘 (1/2)

Neville Interpolation

```

procedure neville( $x_0 \sim x_{n-1}, y_0 \sim y_{n-1}$ : real numbers,  $x$ : real number)
{  $(x_i, y_i)$ 's are the given points. }
{  $x$  is the value that we want to get the  $f(x)$ . }
for  $i := 0$  to  $n-1$  {increment}
begin
   $g_{cur}[0] = y_i$ 
   $k := 1$ ;
  for  $j := i$  to 1 {decrement}
  begin
     $g_{cur}[i-j+1] := \text{calc\_product}(g_{prev}[i-j], g_{cur}[i-j], x_i, x_{(i-k)}, x)$ ;
     $k := k+1$ ;
  end
  for  $j := 0$  to  $i$ 
     $g_{prev}[j] = g_{cur}[j]$ ;
  end
end
return y;
  
```



Page 28

Numerical Analysis  
by Yang-Sae Moon



## 네빌레의 반복 보간법 - 알고리즘 (2/2)

Neville Interpolation

procedure *calc\_product*( $g_{prev}, g_{cur}, x_e, x_s, x$ : real numbers)

$$y := \frac{(x - x_s)g_{cur} - (x - x_e)g_{prev}}{x_e - x_s};$$

return  $y$ ;



## 네빌레의 반복 보간법 - 프로그램 (1/4)

Neville Interpolation

주어진 문제: 다음 표와 같이 다섯 개의  $x$  값과 이의 함수 값이 주어졌을 때,  $x = 2.7$ 에 대한 근사 함수 값을 네빌레의 반복 보간법으로 구하시오.

$x$	$f(x)$
1	1.0
2	1.414214
3	1.732051
4	2.0
5	2.236068

참고: 상기 표의 함수 값은  $f(x) = \sqrt{x}$ 에 해당한다.

## 네빌레의 반복 보간법 - 프로그램 (2/4)

Neville Interpolation

210.115.58.78 - Zterm
95x31

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX_PTS    10

float calc_product(float, float, float, float, float);

main(int argc, char *argv[])
{
    int i, j, k, n;
    float xi[MAX_PTS], yi[MAX_PTS], x, y;
    float gp[MAX_PTS], gc[MAX_PTS];

    if(argc < 3) {
        printf("Usage: %s input-file x\n", argv[0]);
        exit(0);
    }

    if(read_input(argv[1], xi, yi, &n) < 0) {
        printf("cannot read the input file: %s\n", argv[1]);
        exit(-1);
    }

    x = (float)atof(argv[2]);

    printf("(xi, yi): input =====\n");
    for(i=0; i < n; i++)
        printf("(%.5f, %.5f)\n", xi[i], yi[i]);
    printf("===== \n");

```

1,1
Top

Page 31

Numerical Analysis  
by Yang-Sae Moon

## 네빌레의 반복 보간법 - 프로그램 (3/4)

Neville Interpolation

210.115.58.78 - Zterm
95x21

```

for(i=0; i < n; i++) {
    gc[0] = yi[i];
    for(j=i, k=1; j >= 1; j--, k++) {
        y = calc_product(gp[i-j], gc[i-j], xi[i], xi[i-k], x);
        gc[i-j+1] = y;
    }
    for(j=0; j <= i; j++)
        gp[j] = gc[j];
}

printf("(x, g(x), sqrt(x)) = (%.6f, %.6f, %.6f)\n", x, y, sqrt((float)x));
}

float calc_product(float gp, float gc, float xe, float xs, float x)
{
    return( ((x-xs)*gc - (x-xe)*gp) / (xe-xs) );
}

```

31,0-1
68%

Page 32

Numerical Analysis  
by Yang-Sae Moon





## 네빌레의 반복 보간법 - 프로그램 (4/4)

Neville Interpolation

```
210.115.58.78 - Zterm
read_input(char *File, float x[], float y[], int *n)
{
    int i;
    FILE *Fp;

    if((Fp = fopen(File, "r")) == (FILE *)0) return -1;

    fscanf(Fp, "%d", (char *)n);
    for(i=0; i < (int)*n; i++)
        fscanf(Fp, "%F%F", (char *)&x[i], (char *)&y[i]);

    return 1;
}
```

51,0-1 Bot



## 네빌레의 반복 보간법 - 실행 결과


Neville Interpolation

### 입력 파일

```
210.115.58.78 - Zterm
dke [ ysmoon {52} ~/NA/Ch7 ] cat ne_input1
5
1.0 1.0
2.0 1.414214
3.0 1.732051
4.0 2.0
5.0 2.236068
dke [ ysmoon {53} ~/NA/Ch7 ]
```


### 실행 결과 ( $x = 2.7$ )


```
210.115.58.78 - Zterm
dke [ ysmoon {61} ~/NA/Ch7 ]
dke [ ysmoon {61} ~/NA/Ch7 ] neville ne_input1 2.7
=(xi, yi): input =====
(1.00000, 1.00000)
(2.00000, 1.41421)
(3.00000, 1.73205)
(4.00000, 2.00000)
(5.00000, 2.23607)
=====
(x, g(x), sqrt(x)) = (2.700000, 1.643503, 1.643168)
dke [ ysmoon {62} ~/NA/Ch7 ]
```





We are now ...


Interpolation on Newton Polynomials


선형 보간법


라그랑제 다항식 보간법


네빌레의 반복 보간법



뉴턴 다항식에 의한 보간법



강원대학교  
컴퓨터공학과


Page 35

Numerical Analysis  
by Yang-Sae Moon



뉴턴 보간법 개요


Interpolation on Newton Polynomials



뉴턴 보간법은 (네빌레 보간법과 유사하게) 라그랑제 보간법의


- 1) 하나의 보간을 위해 필요한 계산량이 많고,
- 2) 데이터의 수가 증가할 때, 바로 직전의 결과를 사용하지 못하며,
- 3) 에러 계산이 용이하지 않은

문제점을 해결한다.




뉴턴 보간법에서는 기존 데이터를 기초로 차분표(differential table)를 구성하고, 이 차분표를 사용하여 보간 공식을 구한다.

또한, 새로운 데이터가 추가되어도 그 차수를 늘리기 쉬운 장점이 있다.



뉴턴 보간법은 데이터의 종류 및 방법에 따라 다음 세 가지가 있다.

- 주어진 점의  $x$  값 간격이 등간격이 아닌 경우: 분할 차분법
- 주어진 점의  $x$  값 간격이 등간격인 경우: 전향 차분법 혹은 후향 차분법



강원대학교  
컴퓨터공학과

Page 36

Numerical Analysis  
by Yang-Sae Moon



## 분할 차분법 개념 (1/4)

Interpolation on Newton Polynomials

- 서로 다른  $(n+1)$ 개의 점  $x_0, x_1, x_2, \dots, x_n$ 에 대해서, 함수  $f(x)$ 와 함수 값이 같은  $n$ 차 이하의 다항식  $P_n(x)$ 가 다음과 같이 주어진다고 하자.  
(다음과 같은 형태를 뉴턴형이라 한다.)

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \dots + a_n(x - x_0)(x - x_1)\dots(x - x_{n-1})$$

- 그러면, 각  $x_i$  값에 따라서 다음 관계가 만족하고, 이에 따라 상수항  $a_0, a_1, \dots$ 을 순서대로 구할 수 있다.

$$\begin{aligned} f(x_0) &= P_n(x_0) = a_0 \\ f(x_1) &= P_n(x_1) = a_0 + a_1(x_1 - x_0) \\ f(x_2) &= P_n(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \\ &\vdots \end{aligned}$$



양성대학교  
컴퓨터공학과

Page 37

Numerical Analysis  
by Yang-Sae Moon



## 분할 차분법 개념 (2/4)

Interpolation on Newton Polynomials

- 중복된 계산식(예:  $(x_2 - x_0)$ )을 줄이기 위해, 분할 차분 기호를 사용한다.

$$\begin{aligned} f[x_i] &= f(x_i) \\ f[x_i, x_j] &= \frac{f[x_j] - f[x_i]}{x_j - x_i} = \frac{f(x_j) - f(x_i)}{x_j - x_i} \\ f[x_i, x_j, x_k] &= \frac{f[x_j, x_k] - f[x_i, x_j]}{x_k - x_i} = \frac{\frac{f(x_k) - f(x_j)}{x_k - x_j} - \frac{f(x_k) - f(x_i)}{x_k - x_i}}{x_k - x_i} \\ &\vdots \end{aligned}$$

- 이를 일반화 시켜서 표현하면 다음과 같다.

$$f[x_0, x_1, \dots, x_n] = \frac{f[x_1, x_2, \dots, x_n] - f[x_0, x_1, \dots, x_{n-1}]}{x_n - x_0}$$



양성대학교  
컴퓨터공학과

Page 38

Numerical Analysis  
by Yang-Sae Moon



## 분할 차분법 개념 (3/4)

Interpolation on Newton Polynomials

분할 차분 기호를 사용하여  $P_n(x)$ 를 다시 표현하면 다음과 같다.

$$\begin{aligned}
 a_0 &= f(x_0) \\
 a_1 &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} \\
 P_n(x) &= f[x_0] + f[x_0, x_1](x - x_0) + \cdots \\
 &\quad + f[x_0, x_1, \dots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}) \\
 &\quad + \sum_{i=0}^n \left( f[x_0, x_1, \dots, x_i] \cdot \prod_{j=0}^{i-1} (x - x_j) \right)
 \end{aligned}$$



## 분할 차분법 개념 (4/4)

Interpolation on Newton Polynomials

차분 기호를 이용한 방정식 풀이를 위해 차분표를 작성하면 다음과 같다.

$i$	$x_i$	$f[x_i]$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$	...
0	$x_0$	$f[x_0]$	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$	
1	$x_1$	$f[x_1]$	$f[x_1, x_2]$	$f[x_1, x_2, x_3]$	$f[x_1, x_2, x_3, x_4]$	
2	$x_2$	$f[x_2]$	$f[x_2, x_3]$	$f[x_2, x_3, x_4]$		
3	$x_3$	$f[x_3]$	$f[x_3, x_4]$			
4	$x_4$	$f[x_4]$				
...	...	...	...	...	...	...



## 분할 차분법 - 알고리즘

Interpolation on Newton Polynomials

```

procedure newton-diff( $x_0, \dots, x_{n-1}$ ,  $y_0, \dots, y_{n-1}$ : real numbers,  $x$ : real number)
{  $(x_i, y_i)$ 's are the given points. }
{  $x$  is the value that we want to get the  $f(x)$ . }

  for  $i := 0$  to  $n-1$     $f_{\text{cur}}[i] := f_{\text{prev}}[i] := y_i$ ;

  for  $i := 1$  to  $n-1$ 
  begin
    for  $j := i$  to  $n-1$     $f_{\text{cur}}[j] := (f_{\text{prev}}[j] - f_{\text{prev}}[j-1]) / (x_j - x_{j-1})$ ;
    for  $j := i$  to  $n-1$     $f_{\text{prev}}[j] := f_{\text{cur}}[j]$ ;
  end

   $y := 0$ ;  $t := 1$ ;
  for  $i := 0$  to  $n-1$ 
  begin
     $y := y + (f_{\text{prev}}[i] \times t)$ ;
     $t := t \times (x - x_i)$ ;
  end

  return  $y$ ;
  
```

교재에서는 2-D Array를 사용하였으나,  
실제로는 1-D Array들로 해결이 가능하다.



## 분할 차분법 - 프로그램 (1/3)

Interpolation on Newton Polynomials

주어진 문제: 다음 데이터를 참고로 하여,  $x = 3.8$ 일 때의 근사 함수 값을 뉴턴의 분할 차분법을 이용하여 구하시오.

$x$	$f(x)$
3.0	1.09861
3.3	1.19392
3.5	1.25276
3.7	1.30833
4.0	1.33500



## 분할 차분법 - 프로그램 (2/3)

Interpolation on Newton Polynomials

210.115.58.78 - Zterm
89x32

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX_PTS    10

main(int argc, char *argv[])
{
    int i, j, n;
    float x, y, t;
    float xi[MAX_PTS], yi[MAX_PTS];
    float fp[MAX_PTS], fc[MAX_PTS];

    if(argc < 3) {
        printf("Usage: %s input-file xWn", argv[0]);
        exit(0);
    }

    if(read_input(argv[1], xi, yi, &n) < 0) {
        printf("cannot read the input file: %sWn", argv[1]);
        exit(-1);
    }

    x = (float)atof(argv[2]);

    printf("(x, y): input =====Wn");
    for(i=0; i < n; i++)
        printf("(%.5f, %.5f)Wn", xi[i], yi[i]);
    printf("=====Wn");


    1,1
    Top
        
```



Yang-Sae Moon

Page 43

Numerical Analysis  
by Yang-Sae Moon



## 분할 차분법 - 프로그램 (3/3)

Interpolation on Newton Polynomials

210.115.58.78 - Zterm
89x20

```

for(i=0; i < n; i++) fc[i] = fp[i] = yi[i];

for(i=1; i < n; i++) {
    for(j=i; j < n; j++)
        fc[j] = (fp[j] - fp[j-1]) / (xi[j] - xi[j-i]);

    for(j=i; j < n; j++) fp[j] = fc[j];
}

y = 0.0;
t = 1.0;
for(i=0; i < n; i++) {
    y += (fp[i] * t);
    t *= (x - xi[i]);
}

printf("(x, P(x)) = (%.6f, %.6f)Wn", x, y);
        
```

32,0-1
67%

210.115.58.78 - Zterm
89x15

```


read_input(char *file, float x[], float y[], int *n)
{
    int i;
    FILE *fp;

    if((fp = fopen(file, "r")) == (FILE *)0) return -1;

    fscanf(fp, "%d", (char *)n);
    for(i=0; i < (*n); i++)
        fscanf(fp, "%f%f", (char *)&xi[i], (char *)&yi[i]);

    return 1;
}
        
```

05,1
Bot



Yang-Sae Moon

Numerical Analysis  
by Yang-Sae Moon



## 분할 차분법 - 실행 결과

Interpolation on Newton Polynomials

**입력 파일**


```

210.115.58.78 - Zterm
dke [ ysmoon {51} ~/NA/Ch7 ] cat nd_input1
5
3.0 1.09861
3.3 1.19392
3.5 1.25276
3.7 1.30833
4.0 1.33500
dke [ ysmoon {52} ~/NA/Ch7 ]
```

**실행 결과 ( $x = 3.8$ )**


```

210.115.58.78 - Zterm
dke [ ysmoon {53} ~/NA/Ch7 ] newton_diff nd_input1 3.8
-(xi, yi): input
(3.00000, 1.09861)
(3.30000, 1.19392)
(3.50000, 1.25276)
(3.70000, 1.30833)
(4.00000, 1.33500)
=====
(x, P(x)) = (3.800000, 1.329137)
dke [ ysmoon {54} ~/NA/Ch7 ]
```


 양승태대학교  
컴퓨터공학과

Page 45

Numerical Analysis  
by Yang-Sae Moon



## 전향 차분법 및 후향 차분법 개념

Interpolation on Newton Polynomials

**전향 차분법과 후향 차분법은 기본적으로 분할 차분법과 동일하나, 주어진 데이터가 등간격인 경우에 사용하는 좀 더 간략화된 방법이다.**

**전향 차분법은 주어진 점  $x_i (i = 0, 1, 2, \dots, n)$ 의 간격이  $h$ 일 때, 함수 값을 구하고자 하는 점  $x$ 를  $(x_0 + sh)$ 로 놓고 다음 관계를 활용하는 방법이다.**

$$x = x_0 + sh \Leftrightarrow x - x_0 = sh \quad (\text{where } s < 0)$$


$$\therefore x - x_i = (x_0 + sh) - (x_0 + ih) = (s - i)h$$

**후향 차분법은 주어진 점  $x_i (i = 0, 1, 2, \dots, n)$ 의 간격이  $h$ 일 때, 함수 값을 구하고자 하는 점  $x$ 를  $(x_n + sh)$ 로 놓고 다음 관계를 활용하는 방법이다.**

$$x = x_n + sh \Leftrightarrow x - x_n = sh \quad (\text{where } s < 0)$$

$$\therefore x - x_i = (x_n + sh) - (x_n - (n - i)h) = (s + n - i)h$$

분할 차분법과 유도 과정이 동일하므로, 알고리즘 및 프로그램은 생략한다.


 양승태대학교  
컴퓨터공학과

Page 46

Numerical Analysis  
by Yang-Sae Moon